NEAR EAST UNIVERSITY

FACULITY OF ENGINEERING

Department of Electrical and Electronic Engineering

AC MOTOR FORWARD CONTROLLING

Gradu~ti<>ıı Project
EE-400

Student: Hisham Mahmud Tariq (981326)

Supervisor: Mr. Ozgur **Cemal Ozerdem**

Nicosia ·. 2004

ACKNOWLEDGEMENTS

In the name of Allah whose the most gracious and most merciful.

First of all I would like to thank my supervisor Mr. Ozgur Cemal Ozerdem ,without his invaluable advise, inspiration and help this project wôuld never have come to fruition .I thank Mr. Ozgur Cemal Ozerdem for his consistently sfrpport and guiding to me during the course ofthis project.

Second, I would like to express my feeling and gratitude to Near East University for letting me be a part of it. lf it was not for my study in Near East University this project probably would have not materialized.

Third, I thank my father and mother for there for believing in me and sharing in both the good times and the bad. Mom and dad, without your special love and support, I never would have become who I anı today.

Further, I thank Malik Osama Nazar for his outstanding efforts in the making of this project .Also I want to thank Salman Sultan who helped me in ali the way he could and could not.

Finally, I would also like to thank Badr-ud-Duja and Muhammad Awais Janjua fc,r believing in me and commending me when I was right on, and gently letting me know when I have gone off track.

ABSTRACT

The increasing use of motors in all fields of industries has made things easier for many people, but this has also increased the competition and ever growing demand of the berter and new technologies to control them. Motor controlling is one of the main areas of industrial automation development and it is also improving day by day.

The main aim of this project is to develop a program to control an AC motor using a programmable logic controller. in this project we have been able to put our consideration towards the behavior of programmable logic controllers and we have been able to program a Siemens Simatic S7-200 programmable logic controller with CPU 212 to control an AC motor.

The basic structure, functions and methods to program the programmable logic controllers is also discussed in the project.

INTRODUCTION

Motor Controlling is one of the most important aspects of industrial automation. Now a days we can use many different methods other than programmable logic controllers but as the programmable logic controllers are manufactured for motor controlling that' s why they are berter than other systems in many ways. So I took this project to programa Siemens Simatic S7-200 programmable controller to control an AC motor over time.

This project begins by providing an introduction to the programmable controllers and their history in the fırst chapter.

Second chapter explains the internal strength of the programmable controllers to perform a task and theory of the operation that how it controls the inputs, outputs and the actual program of the programmable controllers.

Third chapter explains about process carried to replace relays by programmable controllers, the very basic instructions to write a ladder program used to operate the programmable controllers to control motors.

Fourth chapterexplains them.ain instructions used to write any type of programs for programmable CôntrôllersW:fö control motors and the parts used in the programmable ccrıtroller like the different types ô[Jitıiers; ô.iffete:nftypes<of .côtırıtets,.aıid shift registers and after that the tıiethod of" ğettin.ğ arid ı:nôviiig data from source to destination.

Fifth chapter explains the mathematical instructions carried out inside the programmable controller and the numbers and number systems like binary, decimal, octal, hexadecimal and Boolean algebraic systems used inside it.

Sixth chapter explains about the methods of making connection of the programmable controllers to a system like connected to DC inputs or AC inputs and the outputs of relays and transistor accordingly.

Seventh chapter explains the detailed process of the ways to communicate with a programmable controller like the "RS-232" communications method.

Eighth chapter is about the designing and implementation of a program to operate a programmable controller to control an AC motoragainst specifıed conditions of time.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION AND HISTORY OF PLC

1.1 INTRODUCTION TO PLC

APLC (Prograınmable Logic Controller) is a device that was invented.to replace the necessary sequential relay circuits for machine control. The PLC works by.looking at its inputs and depending upon their state, turning on/off its outputs. The user enters a program, usually via software, that gives the desired results.

PLCs are used in rnany "real world" applications. If there is industry present, chances are good that there is a plc present. If you are involved in rnachining, packaging, material handling, autornated assernbly or countless other industries you are probably already using thern. If you are not, you are wasting rnoney and time. Alrnost any apıplicationthat needs some type-of electrical control hasa need fora plc.

For example, let's assume that when a switch turns on we want to turn a solenoid on for 5 seconds and then turn it off regardless of how long the switch is on for. We can do this with a simple external timer. But what if the process included 10 switches and solenoids? We would need 10 external timers. What if the process also needed to count how many tirnes the switches individually turned on? We need a lot of external counters.

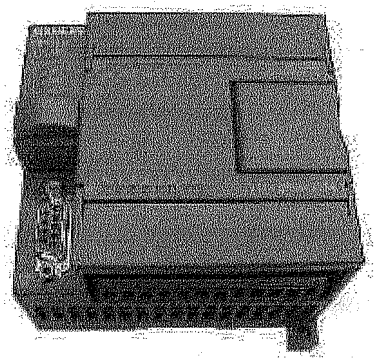


Figure 1.1 A siernens sirnatic s7-200 PLC device

As you can see the bigger the process the more ofa need we have for a PLC. We can simply program the PLC to count its inputs and turn the solenoids on for the specified time.

We will take a look at what is considered to be the "top 20" plc instructions. It can be safely estimated that with a fırın understanding of these instructions one can solve more than 80% of the applications in existence. That's right, more than 80% Of course we'll learn more than just these instructions to help you solve almost ALL your potential plc applications.

1.2 PLC History

In the late 1960's PLCs were fırst 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 compan... at the time nröǫosed comnuter 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. Were there problems? You bet!

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. That's a lot to ask! 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. Still, it was a great decade for the PLC!

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 modemization 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 intemational 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.

Chapter2

THEORY OFOPERTATION OF PLC

2.1 The Guts inside

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

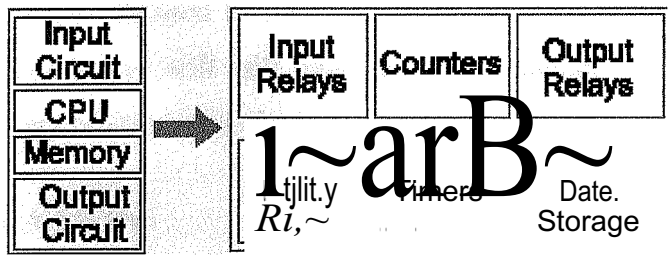


Figure 2.1 internal structure of PLC

2.2 FUNCTION OF EACH PART

- INPUT RELAYS-(contacts) These are connected to the outside world. They physically exist and receive signals from switches, sensors, ete. 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 perferming 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 increrrients. The most common type is an on-delay type. Others include off-delay and both retentive and non-retentive types. Increments vary from lms through Is.
- **OUTPUT** RELAYS-(coils): These are connected to the outside world. They physically exist and send on/off signals to solenoids, lights, ete. They can be transistors, relays, or traces depending upon the model chosen.
- **DAT**A STORAGE-Typically there are registers assigned to simply store <lata. They are usually used as temporary storage for matlı ordata manipulation. They can also typically be used to store <lata when power is removed from the PLC. Upon power-up they will still have the same contents as before power was removed. Very·corıverıieritandnecessary!!

2.3 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 intemal counter and timer values.

CHECKINPUT STATUS

EXECUTE PROGRAM

UPDATE OUTPUT STATUS

Figure 2.2 Scanning steps of PLC programs

2.3.1 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 <lata into its memory to be used during the next step.

2.3.2 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/offfrom the previous step it will be able to decide whether the firstoutput 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.

2.3.3 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 }n step 2 it would now turn on thesfirst 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.

2.4 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 concem, 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 inpüt citcüit 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 <lata 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:

2.4.1 INPUT- it took a certain amount of time for the brain to notice the input signal from the eyes.

2.4.2 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 .

2.4.3 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

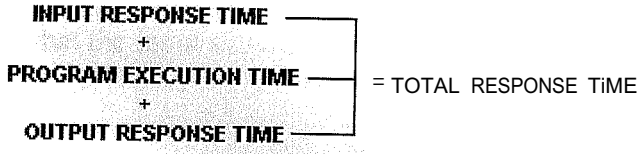


Figure 2.3 Response of PLC to the execution steps and overall

2.5 EFFECTS OF RESPONSE TIME

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

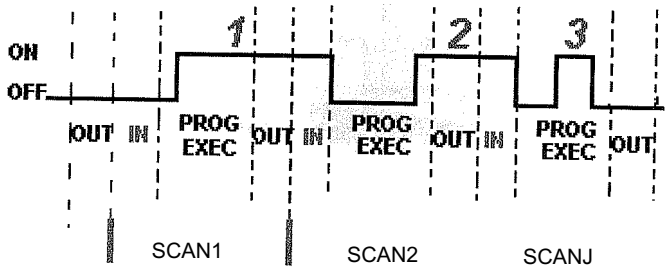


Figure 2.4 Time scan,

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.

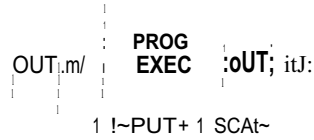


Figure 2.5 Time scan.

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.

2.5.1 Pulse streteh **function.** This function extends the length ofthe input signal until the plc looks at the inputs during the next scan. (i.e. it stretches the duration ofthe pulse.)

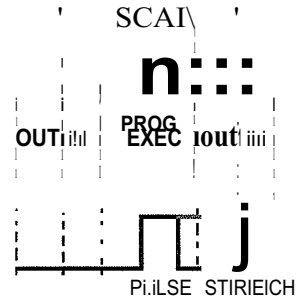


Figure 2.6 Pulse stretch function.

2.5.2 Interrupt function. This function interrupts the scan to process a special routine that you have written. i.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 asa mini program outside of the main program.) After it's done executing the interrupt routine, it goes back to the point it left off at and continues on with the normal scan process.

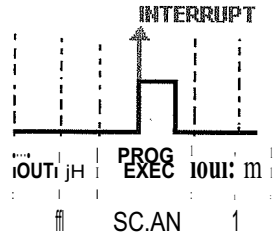


Figure 2.7 Interrupt function

Now let's consider the longest time foran output to actually tum on. Let's assume that when a switch turns on we need to turu 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.

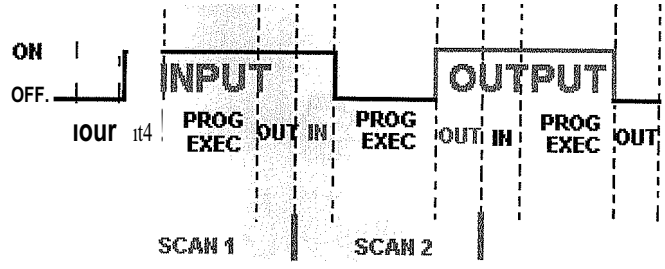


Figure 2.8Time seans

Chapter3

CREATIN6 PROGRAMS

3.1 Relays

Now that we understand how the)PLC ·iprocesses inputs, outptıts, and -the actual program we are almost ready to start writin.g a program. But first lets see höw a relay actually works. After all, the mainpurpôse ofaplc is to replace "real-world" relays.

We can think ofa relay as aıı electromagnetic switch. Apply a voltage to the coil and a magnetic fıeld is generate<l.j)Thismagnetic fıeld sucks the contacts of the relay in, causing them to make a côımection. These contacts can be considered to be a switch. They allow current to flowbetween 2 points thereby closing the circuit.

Let's consider the following exampl- Here we simply tum on a bell (Lunch time!) whenever a switch is closed. We have 3 real-world parts. A switch, a relay and a bell. Whenever the switch closes we apply a current to a bell causing it to sound.

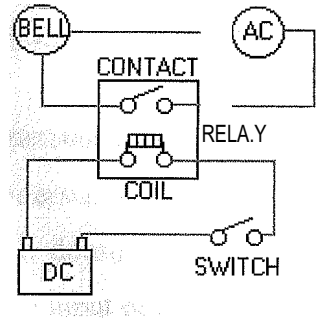
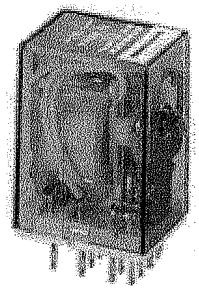


Figure 3.1 Aisifüple DC circuit

Notice in the picture that we have 2 separate circuits. The bottom indicates the DC part. The top indicates the AC part.

Here we are using a de relay to control aıı AC circuit, That's the fun of relays! When the switch is open no current can flow through the coil of the relay. As soon as the switch is closed, however, current runs through the coil causing a magnetic fıeld to

build up. This magnetic field causes the contacts of the relay to close. Now AC current flows through the bell and we hear it. Lunch time!



Figute 3.2 A typical industrial relay

3.2 Replacing Relays

Next, let's use a PLC in place of the relay. (Note that this might not be very cost effective for this application but it does demonstrate the basics we need.) The first thing that's necessary is to create what's called a ladder diagram. After seeing a few of these it will become obvious why it's called a ladder diagram. We have to create one of these because, unfortunately, a plc doesn't understand a schematic diagram. It only recognizes code. Fortunately most PLCs have software which converts ladder diagrams into code. This shields us from actually learning the plc's code.

3.2.1 First step- We have to translate all of the items we're using into symbols the plc understands. The plc doesn't understand terms like switch, relay, bell, ete. It prefers input, output, coil, contact, ete. It dôesıi'tcare what the actual input or output device actually is. It only cares that it's an inputör anoutput.

First we replace the battery with a synıbôl. This symbol is common to all ladder diagrams. We draw what are called bus bars. These simply look like two' vertical ba.fs. üne on each side of the diagram. Think oftheleff ône as being + volta.gea.n.d the .right eme as being ground. Further think of the curreiit (logic) flow as being frôrı:i left to right.

we give the inputs a symbol. In this basic example we have one real world input.

the switch) We give the input that the switch will be connected to, to the symbol shown below. This symbol can also be used as the contact ofa relay.

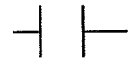


Figure 3.3 A contact symbol

Next we give the outputs a symbol. in this example we use.oneoutput.(i.e. the bell). We give the output that the bell will be physically connected tocthecsymbol shown below. This symbol is used as the coil ofa relay.

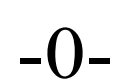


Figure 3.4 A coil symbol

The AC supply is an extemal supply so we don't put it in our ladder. The plc only cares about which output jt.tums on and not what's physically connected to it.

3.2.2 Second step- We must tell the plc where everything is located. In other words we have to give all the devices an address. Where is the switch going to be physically connected to the plc? How about the bell? We start with a blank road map in the PLCs town and give each item an address. Could you find your friends if you didn't know their address? You know they live in the same tô\\111 but which house? The plc town has lot of houses (inputs and outputs) but we have to :figure out who lives where (what device is connected where). We'll get further into the addressing scheme later. The plc manufacturers each do ita different way! For now let's say that our input will be called "0000". The output will be called "500".

3.2.3 Final step- We have to convert the schematic into a logical sequence of events. This is much easier than it sounds. The program we're going to write tells the plc what to do when certain events take place. Inour exanı.plewe have to tell the plc what.to do when the operator tums on the switch. Obviously we want the bell to sound but the plc doesn't know that.

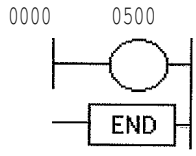


Figure 3.5 Ladder replacement ofrelay to PLC program

The picture above is the final converted diagram. Notice that we eliminated the real world relay from needing a symbol. It's actually "inferred" from the diagram.

3.3 **Basic Instructions**

Now let's examine some of the basic instructions is greater detail to see more about what each one does.

3.3.1 Load

The load (LD) instruction .is >a normally open contact. It is sometimes also called examine if on. (XIO)(asin examin.ethe .inpıt to see if its physically on) The symbol for a load instruction is shown below.

Figure 3.6 A Load (contact) symbol

This is used when an input signal is needed to be present for the symbol to turn on. When the physical input is on we can say that the instruction is True. We examine the input for an on signal. If the input is physically on then the symbol is on. An on condition is also referred to as logic 1 state.

This symbol normally can be used for internal inputs, external inputs and external output contacts. Remember that internal relays don't physically exist. They are simulated (software) relays.

3.3.2 Load Bar

The Load Bar instruction is a normally closed contact. It is sometimes also called Load Not or examine if closed. (XIC) (as in examine the input to see if its physically closed) The symbol for a load bar instruction is shown below.

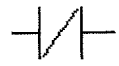


Figure 3.7 A Load Not (normally closed contact) symbol

This is used when an input signal does not need to be present for the symbol to tum on. When the physical input is off we can say that the instruction is True. We examine the input for an off signal. If the input is physically off then the symbol is on. An off condition is also referred to as a logic 0 state.

This symbol normally can be used for intemal inputs, extemal inputs and sometimes, extemal output contacts., ltelll.ember .again that intemal relays don't physically exist. They are simulated(sôftware)rela.ys.Itisthe exact opposite ofthe Load instruction.

Logic State	Load	Load Bar
0	False	True
1	True	False

Taole 3.1

3.3.3 Out

The Out instruction is sometimes also called an Outpıt Energize instruction. The output instruction is like a relay coil. Its sym.bol looks as shown below.

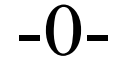


Figure 3.8 An OUT (coil) symbol

When there is a patlı of True instructions preceding this on the ladder rung, it will also be True. When the instruction is True it is physically On. We can think of this

instruction as a normally open output. This instruction can be used for intemal coils and extemal outputs.

3.3.4 Out bar

The Out bar instruction is sometimes also called an Out Not instruction. Some vendors don't have this instruction. The out bar instruction is like a normally closed relay coil. Its symbol looks like that showİı below.

Figure 3.9 An OUT Bar (normally closed coil) symbol

When there is a patlı ofFalse instnıctionspreceding this on the ladder rung, it will be True. When the instruction is True it is physically On. We can think ofthis instruction as a normally closed output. This instruction can be used for internal coils and extemal outputs. It is the exact opposite of the Out instruction.

Logic State	Out	Out Bar
0	False	True
1	True	False

Table 3.2

3.4 A Simple Example

let's compare a simple ladder diagram with its real world extemal physically connectecırelay circuit and see the differences.

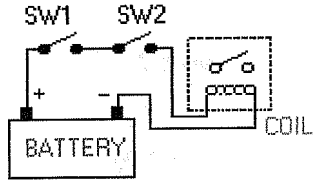


Figure 3.10 A simple coil and battery circuit

In the above circuit, the coil will be energized when there is a closed loop between the + and - terminals of the battery. We can simulate this same circuit with a ladder diagram. A ladder diagram consists of individual rungs just like ona real ladder. Each rung must contain one or more inputs and one orınqr~.outputs. The first instruction ona rung must always be an input instruction and the JastJnstruction on a rung *should* always be an output (or its equivalent).

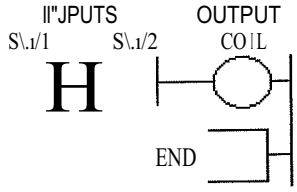


Figure 3.11 A ladder replacement of the circuit in figure 3.10

Notice in this simple one rung ladder diagram we have recreated the extemal circuit above with a ladder diagram. Here we used the Load and Out instructions. Some manufacturers require that every ladder diagram include an END instruction on the last rung. Some PLCs also require an ENDH instruction on the rung after the END rung.

3.5 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 ctıagr~n now looks like this:

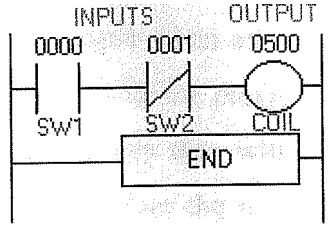


Figure 3.12Alad.dyr diagram

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 tha.t 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.

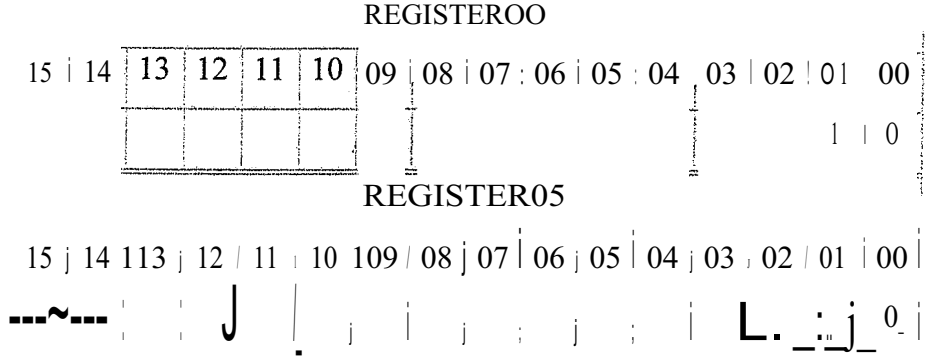


Table 3.3

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 0. The logic 0 örl 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 0. 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 3.4

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 SWI has to be logic 1 and SW2 must be logic 0. 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 ali possible combinations ofthe status ofthe two inputs.

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

Table 3.5

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.

3.6 A Level Application

Now that we've seen how registers work, let's process a program like PLCs do to enhance our understanding ofhow 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.

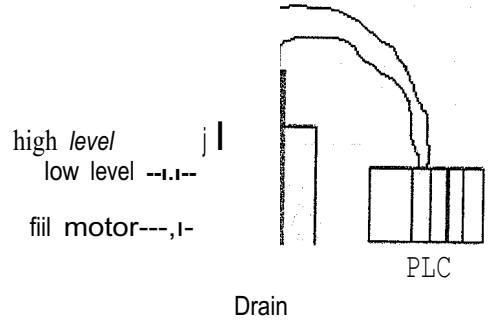


Figure 3.13 Dispensing oil from the tank

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 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 3.6

Below is what the ladderdiağraırıwilLacfua.llyJook like. Notice that we are using an intemal utility relay in this exaırıple. Yôuca:rılisethe corıtacts öfthese relays as many times as required. Here they are used twice fo sirriulate a relay with2 sets of contacts. Remember, these relays do not physically exist in the plc but-raiher they are bits in a register that you can use to simulate a relay.

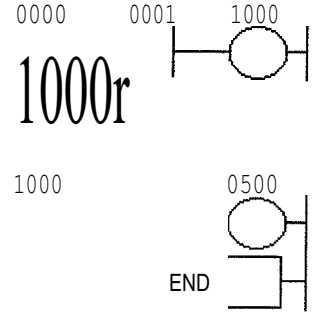


Figure 3.14 Ladder program to control the dispensing oil

We should always remember that the most common reason for using PLCs in our applications is for replacing real-world relays. The intemal 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 includes 1000's while still others include IO's of IOOO's! Typically, plc size (not physical size but rather 1/0 size) is the deciding factor. If we are using a micro-plc with a few 1/0 we don't need many internal relays. If however, we are using a large plc with 100's or 1000's of 1/0 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.

3.7 The Program Scan

Let's watch what happens in this program scan by scan.

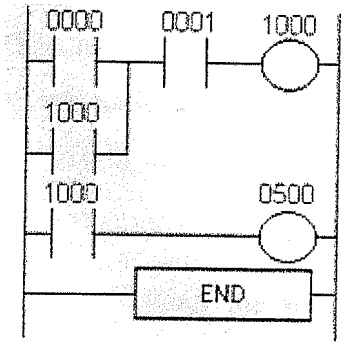
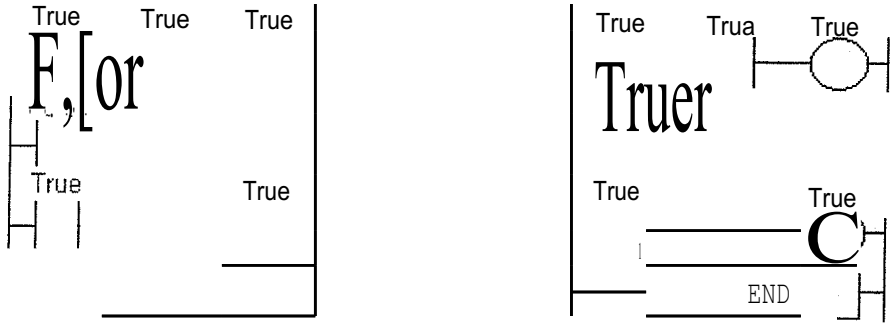


Figure 3.15 Ladder diagram ofthe program

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



Scan 1 Scan 2-100

Figure 3.16 Time seans ofthe program

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

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

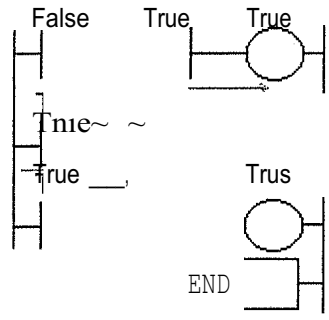
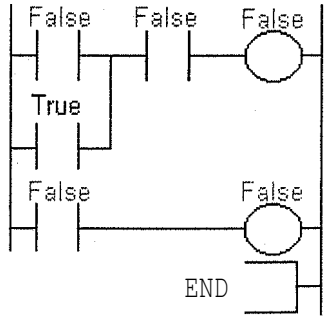


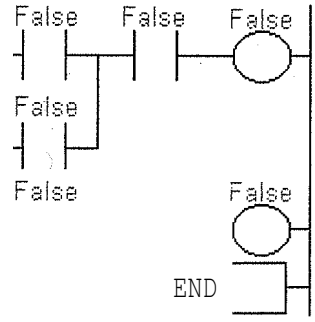
Figure 3.17 Scan 101-1000

Notice that even when the .low level sensor is false there is stil! a patlı of true logic from left to right. This is .whY we used an intemal relay. Relay 1000 is latching the output (500) on. It will sray)tfüs>'Wayuntil there is no true logic patlı from left to right.(i.e. when 0001 becomesfalse)

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



Scan 1001



Scan 1002

Figure 3.18 Time seans ofthe program

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

After 1050 seans the oil level falls below the high level sensor and it will beeome true again.

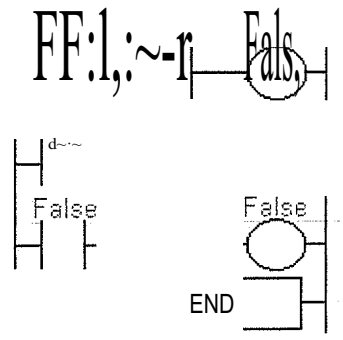


Figure 3.19 Sean 1050

Notiee that even thotıgh>/the high level sensor beeame true there still is Nü eontinuous true logie pathandtherefore eoil 1000 remains false!

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

Chapter 4

MAiN INTSTRUCTIONS SET

4.1 Latch Instructions

Now that we understand ~ow inputs and outputs are processed by the plc, let's look at a variation of our regular outputs. Regiılar output coils are of course an essential part of our programs but we must remember that they are only TRUE when ALL INSTRUCTIONS before them on the rung are also TRUE. What happens if they are not? Then of course, the output will become false. (Turn off)

Think back to the lunch bell example we did a few chapters ago. What would've happened ifwe couldn't find a "push on/push off" switch? Then we would've had to keep pressing the button for as long as we wanted the bell to sound. The latching instructions let us use momentary switches and program the plc so that when we push one the output turns on and when we push another the output turns off.

Maybe now you're saying to yourself "What the heck is he talking about?" So let's do a real world example. Picture the remote control for your TV. it hasa button for ON and another for OFF. (mine does, anyway) When I push the ON button the TV turns on. When I push the OFF button the TV turns off. I don't have to keep pushing the ON button to keep the TV on. This would be the function ofa latching instruction.

The latch instruction is often called a SET or OTL (output latch). The unlatch instruction is often called a RES (reset), OUT (output unlatch) or RST (reset). The diagram below shows how to use them in a program.

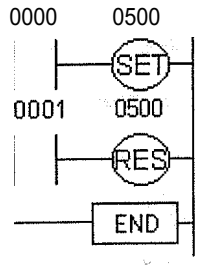


Figure 4.1 A ladder program

Here we are using 2 momentary push button switches. üne is physically connected to input 0000 while the other is physically connected to input 0001. When the operator pushes switch 0000 the instruction "set 0500" will become true and output 0500 physically turns on. Even after the operator stops pushing the switch, the output (0500) will remain on. It is latched on. The only way to turu off output 0500 is turu on input 0001. This will cause the instruction "res 0500" to become true thereby unlatching or resetting output 050Q.

4.2 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 siinply and easily program 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. Confused yet? Can you say "no standardization"? Don't worry; the theory is all the same regardless of what the manufacturers call them. A counter is a counter is a counter...

Tofurther 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 thuınb 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. (2xscan time = 2x2ms= 4ms)

To use them we must know Jthings:

1. Where the pulses thatwe 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 boxthe:rn, for example.
3. *When/how we wilTieset the counter so it can count again. After we count 5 widgets lets resei(Jıecounter, for exemple.*

When the programis 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 Oto 9999, -32,768 to +32,767 or Oto 65535. Why the weird nuınbers? Because 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 Oto 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 ofthem.

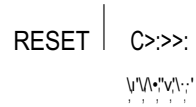


Figure 4.2 Count up counter

In this counter we need 2 inputs. üne 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.

For example, ifwe are counting how many widgets pass in front ofthe 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 wantto call it counter 000 then we would put "COOO" here.

yyyyy is the number of pulses -W~.'*ant to count before doing something. lf we want to count 5 widgets before turning()J.1aphysical output to box them we would put 5 here. If we wanted to count 100 wid.g~t§thenwe would put 100 here, ete. When the counter is finished (i.e. we counted yyyyy '*idgets) it will turn on a separate set of contacts that we also label Cxxx.

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

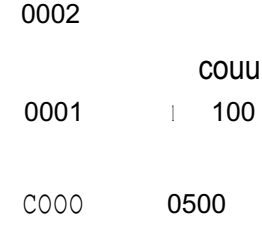


Figure 4.3 A ladder diagram ofthe program using count up counter

Here's the symbol on a ladder 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.

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

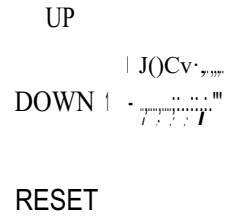


Figure4.4Countup-down counters

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. üne is for counting up and the other is for counting down. In this example we will call the counter UDC000 and we will give ita 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 will 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 below.

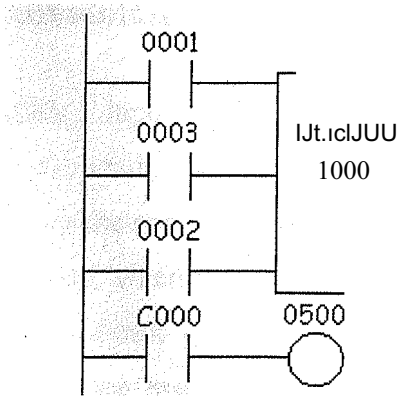


Figure 4.4 Ladder diagra.rı:ı.ofaprogram using count up-down counter

43 Timers

Let's now see how a timer works. What'is a'timer? Its exactly what the word says... it is an instruction that waits a set amount of time before doing something. Sounds simple doesn't it.

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:

4.3.1 On-Delay timer

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

4.3.2 Off-Delay timer

This type of timer isthe>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 turningit 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 oftimer)

4.3.3 Retentive or ~ccumulating timer

This type of timer needS 2 inputs. üne 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/offfor 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 tums 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 ofthe 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 Oto 65535 is 16-bit binary. Each tick ofthe clock is equal to x-seconds.

Typically each manu.fach.:ırer offers several different ticks. Most manufacturers offer 10 and 100 ms increments(ticksôfthe clock). An "ms" is a milli-second or 1/lOOOth of a second. Several manufactu.fersalso offer lms as well as 1 second increments. These different increment timers ~~Jf~~ same as above but sometimes they have different names to show their time base.Some are TMH (high speed timer), TMS (super high speed timer) or TMRAF (accl.lIll.u.la.ting fast timer)

Shown below is a typical tiırı.ebinsttu.ctionsymbol we will encounter (depending on which manufacturer we choose) and>nôWtol.lseit. Remember that while they may look different they are all used basically the sa.ıtı.e"\Va.y. If we can setup one we can setup any ofthem.

Et\lABLE' Txxx
yyyyy

Figure 4.5 A typical timer instruction symbol

This timer is the on-delay type and is named Txxx. When the enable inpunis on the timer starts to tick. When it ticks yyyyy (the preset value) times, it will rurn on .its contacts that we will use later in the program. Remember that the duration ofa tick (increment) varies with the vendor and the time base used. (i.e. a tick might be lms or 1 second or...).Below is the symbol shown ona ladder diagram

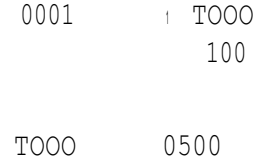


Figure 4.6 A ladder diagram of program using timer

In this diagram we wait for .input 0001 to turn on. When it does, timer TOOO (a lOOmsincrement timer) starts tick.ing. It will tick 100 times. Each tick (increment) is lOOms so the timer will be a lOOOOms (i.e. 10 second) timer. lOOticks X lOOms = 10,000ms. When 10 secoıids füı:ve elapsed, the TOOO contacts close and 500 turns on. When input 0001 turns öff(false)the timer TOOO will reset back to 0 causing its contacts to turn off(become false) the:rebymaking output 500 turn back off. An accumulating timer would look similartôthefi.ğbelow.

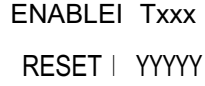


Figure 4.7 An accumulating timer

This timer is named Txxx. Whe:rithe enable input is on the timer starts to tick. When it ticks yyyyy (the preset vah.ıe)tiri:ı.esit will turn on its contacts that we will use later in the program. Remember that the düration ofa tick (increment) varies with the vendor and the time base used. (i.e. a tickilllight be lms or 1 second or...) If however, the enable input turns offbefore the tiınefhas completed, the current value will be retained. When the input turns back on, the tiıner 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. The symbol is shown in the ladder diagram below.

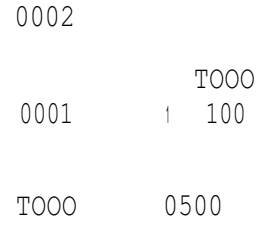


Figure 4.8 An accilinulating timer connected in program

In this diagram we wait for input 0002 to turn on. When it does timer TOOO (a 1Oms increment timer) starts tiu,.m-g It will tick 100 times. Each tick (increment) is lOms so the timer will be a 1000ms (i.e. 1 second) timer. lOOticksX lOms = l,OOOms. When 1 second has elapsed, the T000 contacts close and 500 turns on. If input 0002 turns back off the current elapsed time will be retained. When 0002 turns back on the timer will continue where it left off. When input 0001 turns on (true) the timer TOOO will reset back to 0 causing its contacts to turn off (become false) thereby making output 500 turn back.

4.4 Timer Accuracy

Now that we've seen how timers are created and used, f;Fs learn a little about their precision. When we are creating a timer that lasts a few seconds, or more, we can typically not be very concerned about their precision because it's usually insignificant. However, when we're cr:;;ci.ungiJmers that have duration in the millisecond (lms= 1/1000 second) range we MUST be concemed about their precision.

There are general two when using a timer. The first is called an input error. The other is called The total error is the sum ofboth the input and output errors.

- Input error- An error occuts dependingupon when the timer input turns on during the scan cycle. When the input turns on immediately after the plc looks at the status ofthe inputs during the scan cycle, the input error will be at its largest. (i.e. more than 1 full scan time!). This is because, as you will recall, (see scan time chapter) the inputs are looked at once during a scan. If it wasn't on when

the plc looked and turns on later in the scan we obviously have an error. Further we have to wait until the timer instruction is executed during the program execution part of the scan. If the timer instruction is the last instruction on the rung it could be quite a big error!

- **Output error-** An another error occurs depending upon when in the ladder the timer actually "times out", (expires) and when the plc finishes executing the program to get to the part of the scan when it updates the outputs. (again, see scan time chapter) This is because the timer finishes during the program execution but the plc must first finish executing the remainder of the program before it can turn on the appropriate output.

Below is a diagram illustrating the worst possible input error. You will note from it that the worst possible input error would be 1 complete scan time + 1 program execution time. Remember that a program execution time varies from program to program. (Depends how many instructions are in the program!)

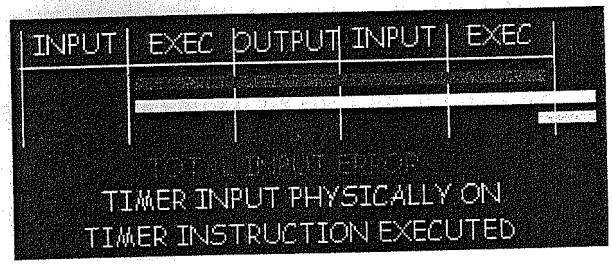


Figure 4.9 illustration of the worst possible input error

Shown below is a diagramilfüstfating the worst possible output error. You can see from it that the worst possibleöutputerror would be 1 complete scan time.

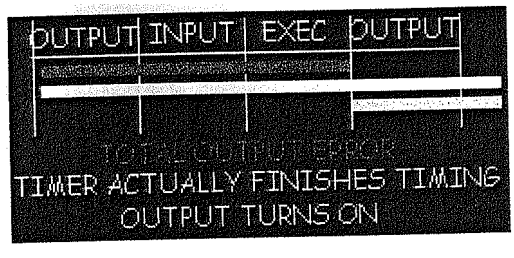


Figure 4.1 0 illustration of the worst possible output error

Based upon the above inforrnation we can now see that the total worst possible timer error would be equal to

1 scan time + 1 program execution time + 1 scan time
= 2 scan times + 1 program execution time.

What does this really mean? It means that even though most manufacturers currently have timers with lms increments they really shouldn't be used for durations lessthana few milliseconds. This assumes that your scan time is 1ms. If your scan time is Sms you had better not use a timer with duration less than about 1 Sms. The point is however, just so that we will know what errors we can expect. If we know what error to expect, we can then think about whether this amount of error is acceptable for our application. In most applications this error is insignificant but in some high speed or very precise applications this error can be very significant.

We should also note that the above errors are only the "software errors". There is also a hardware input error as well as a hardware output error.

The hardware input error is caused by the time it takes for the plc to actually realize that the input is on when it scans its inputs. Typically this duration is about 10ms. This is because many PLCs require that an input should be physically on for a few scans before it determines it's physically on. (1'0 eli111iriate noise or "bouncing" inputs)

The hardware output error is caused by the time it takes from when the plc tells its output to physically turn on until the moment it actually does. Typically a transistor takes about O.Sms whereas a mechanical relay takes about lOms.

The error keeps on growing doesn't it! If it becomes too big for the application, consider using an extemal "hardware" timer.

4.5 One-shots

A one-shot is an interesting and invalual;>le programming tool. At first glance it might be difficult to figure out why such an instruction is needed. After we understand what this instruction does and how to use it, however, the necessity will become clear.

A one-shot is used to make something happen for only 1 scan. Most manufacturers have one-shots that react to an off to on transition and a different type that reacts to an on to off transition. Some names forthe instructions could be difu/difu (differentiate up/down), sotu/sotd (single output up/down), osr (one-shot rising) and others. They all, however, end up with the same result regardless of the name.

~DIFU~

Fiğure 4.11 One-shot Instruction

Above is the symbol for a difu (one-shot) instruction. A difu looks the same but inside the symbol it says "difu". Some of the manufacturers have it in the shape ofa box but, regardless of the symbol, they all function the same way. For those manufacturers that don't include a difTu~t11?0je down instruction, you can get the same effect by putting a NC (normally plgs~§) instruction before it instead of a NO (normally open) instruction. (Le. reverse th.eJôğicbefore the difu instruction)

Let's now setup an application to see how this instruction actually functions in a ladder. This instruction is most often used with some of the advanced instructions where we do some things that must happen only once. However, since we haven't gotten that far yet, let's set up a flip/flop circuit. In simple terms, a flip/flop turns something around each time an action happens. Here we'll use a single pushbutton switch. The first time the operator pushes it we want an output to turn on. it will remain "latched" on until the next time the operator pushes the button. When he does, the output tums off. Here's the ladder diagram that does just that.

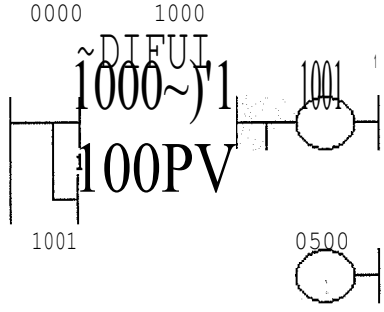


Figure 4. 12 a ladder diagram ofa flip/flop

Now tlıis looks confusing! Actually it's not ifwe take it one step ata time.

- Rung l-When Nü (norınally open) input 0000 becomes true DIFU 1000 becomes true.
- Rung 2- Nü 1000 is true, Nü 1001 remains false, NC 1001 remains true, NC 1000 turns false. Since we have a true patlı, (Nü 1000 & NC 1001) OUT 1001 becomes true.
- Rung 3- Nü 1001. is tı:"lıy tlıerefore OUT 500 turns true.

4.5.1 Next Scan

- Rung 1- NO 0000 remains true. DIFU 1000 now becomes false. This is because the DIFU instruction is only true for one scan. (i.e. the rising edge of the logic before it on the rung)
- Rung 2- NO 1000 is false, NO 1001 remains true, NC 1001 is false, NC 1000 turns true. Since we STILL have a true patlı, (Nü 1001 & NC 1000) OUT 1001 remains true.
- Rung 3- Nü 1001 is true therefore OUT 500 remains true.

After 100 seans, Nü 0000 turns off (becomes false) The logic remains in tlıe same state as "next scan" shown above. (Difu doesn't react tlıerefore tlıe 'logic stays tlıe same on rungs 2 and 3)

On scan 101 Nü 0000 turns back on. (Becomes true)

- Rung l-When Nü (norınally open) becomes true DIFU 1000 becomes true.
- Rung 2- Nü 1000 is true, Nü 1001 remains true, NC 1001 becomes false, NC 1000 also becomes false. Since we no longer have a true patlı, OUT 1001 becomes false.
- Rung 3- Nü 1001 is false tlıerefore OUT 500 becomes false.

Executing the program 1 instruction at a time makes this and any program easy to follow. Actually a larger program tlıat jumps around might be difficult to follow but a pencil drawing of the registers sure does help!

4.6 Master Controls

Let's now look at what are called master controls. Master controls can be thought of as "emergency stop switches". An emergency stop switch typically is a big red button ona machine that will shut it off in cases of emergency. Next time you're at the local gas station look near the door on the outside to see an example of an e-stop. *IMPORTANT- We're not implying that this instruction is a substitute for a "hard wired" e-stop switch. There is no substitute for such a switch! Rather it's just an easy way to get to understand them.

The master control instruction typically is used in pairs with a master control reset. However this varies by manufacturer. Some use MCR in pairs instead ofteaming it with another symbol. It is commonlyal:>breviatedas MC/MCR (master control/master control reset), MCS/MCR (master coııtrô[set/master control reset) or just simply MCR (master control reset). Here is one exaınple.iofhow a master control symbol looks.

Figure 4.13 A master control symbol

Below is an example ofa master control reset.

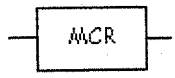


Figure 4.14 A master control reset symbol

To make things interesting, many manufacturers make them act differently. Let's now take a look at how it's used in a ladder diagram. Consider the following example.

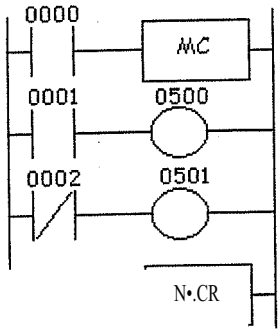


Figure 4.15 A ladder program using MC and MCR

Here's how different PLCs will run this program:

4.6.1 Manufacturer X- In this example, rungs 2 and 3 are only executed when input 0000 is on (true). If input 0000 is not true the plc pretends that the logic between the mc and mcr instructions does not exist. It would therefore bypass this block of instructions and immediately go to the rung after the mcr instruction.

Conversely, if input 0000 is true, the plc would execute rungs 2 and 3 and update the status of outputs 0500 and 0501 accordingly. So, if input 0000 is true, program execution goes to rung·z:•.••lfiriput)000·1····is.··true0500>Vil·l··he.truandhenc.e.it·wi.11.turn.on when the plc updates the outputs. If input 0002is true(i.e. physically off) 0501 will be true and therefore it will turu on when the plc updates the ôutputs.

MCRjust tells the plc "that's the end ofthe mc/mcr block".

In this plc, scan time is not extended when the mc/mcr block is not executed because the plc pretends the logic in the block doesn't exist. In other words, the instructions inside the block aren't seen by the plc and therefore it doesn't execute them.

4.6.2 Manufacturer **Y-** In this example, rungs 2 and 3 are always ex.ecuted regardless of the status of input 0000. If input 0000 is not true the plc executesthe MC instruction. (i.e. MC becomes true) It then forces all the input instructions inside the blockto be off. If input 0000 is true the MC instruction is made to be false.

Then, if input 0000 is true, program execution goes to rung 2. If input 0001 is true 0500 will be true and hence it will turu on when the plc updates the outputs. If input

0002 is true (i.e. physically off) 0501 will be true and therefore it will turn on when the plc updates the outputs. MCR just tells the plc "that's the end of the mc/mcr block". When input 0000 is false, inputs 0001 and 0002 are forced off regardless if they're physically on or off. Therefore, outputs 0500 and 0501 will be false.

The difference between manufacturers X and Y above is that in the Y scheme the scan time will be the same (well close to the same) regardless if the block is on or off. This is because the plc sees each instruction whether the block is on or off.

Most allmartu.facturers will make a previously latched instruction (one that's inside the mc/mcrb'lôck) retain its previous condition.

If it was tı:u.e before, it will remain true.
If it was fü.Isebefore, it will remain false.

Timers sliôuld not be used inside the mc/mcr block because some manufacturers will reset them.tô)z:ero when the block is false whereas other manufacturers will have them retain the curterit time state.

Counters typically retain their current counted value.

Here's the parrto note most of all. When the mc/mcr block is off, (i.e. input 0000 would be false intheladder example shown previously) an OUTB (Out Bar or Out Not) instruction would notbe physically on. It is forced physically off.

-0-

Figure 4.16 Out Bar instruction

In summary, BE CAREFUL! Most manufacturers use the manufacturer Y execution scheme shown above. When in doubt, however, read the manufacturers instruction manual. Betler yet, just ask them.

4.7 Shift Registers

In many applications it is necessary to store the status of an event that has previously happened. As we've seen in. past chapters this is a simple process. But what do we do if we must store many previous events and act upon them later.

Answer: we call upon the shift register instruction.

We use a register or group of registers to form a train of bits (cars) to store the previous on/off status, Each new change in status gets stored in the first bit and the remaining bits get shifted down the train. Huh? Read on.

The shift'register goes by many names. SFT (Shift), BSL (Bit Shift Left), SFR (Shift Forward Register) are some of the common names. These registers shift the bits to the left. BSR(Bit Shift Right) and SFRN (Shift Forward Register Not) are some examples of instructiôns that shift bits to the right. We should note that not all manufacturers have shift registers that shift <lata to the right but most all do have left shifting registers.

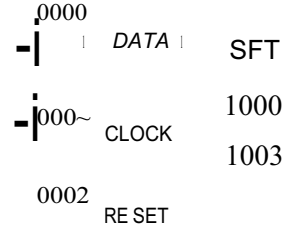


Figure 4.17.t\ladder representation of shift

A typical shift register instruction has a symbôl like that shown above. Notice that the symbol needs 3 inputs and has some data inside the symbol. The reasons for each input are as follows:

- Data- The <lata input gathers the true/false statuses that will be shifted down the train. When the <lata input is true the first bit (car) in theregister (train) will be a 1. This data is only entered into the register (train) on the rising edge ofthe clock input.
- Clock- The clock input tells the shift register to "do its thing". On the rising edge ofthis input, the shift register shifts the <lata one location over inside the

register and enters the status of the data input into the first bit. On each rising edge of this input the process will repeat.

- **Reset-** The reset input does just what it says. it clears ali the bits inside the register we're using to 0.

The 1000 inside the shift register symbol is the location of the first bit of our shift register. If we think of the shift register as a train then this bit is the locomotive. The 1003 inside the symbol above is the last bit of our shift register. it is the caboose. Therefore, we can say that 1001 and 1002 are cars in between the locomotive and the caboose. They are intermediate bits, So, this shift register has 4 bits. (i.e. 1000, 1001, 1002, 1003)

Figure 4.18 A chow-chow train

Let's examine an application to see whylhôW We ca.l'l.luse theshiff.register. Imagine an ice-cream cone machine. We have 4 steps. First we ·verify the/coneis not broken. Next we put ice cream inside the cone.(turn on output 500) Next we add peanuts.rturn on output 501) And finally we add sprinkles.(turn on output 502) If the cone is broken we obviously don't want to add ice cream and the other items. Therefore we have to track the bad cone down our process line so that we can tell the machine not to add each item. We use a sensor to look at the bottom of the cone. (Input 0000) If it' s on then the cone is perfect and if it' s off then the cone is broken. An encoder tracks the cone going down the conveyor. (Input 0001) A push button on the machine will clear the register; (Input 0002).

Here's what the ladder would look like

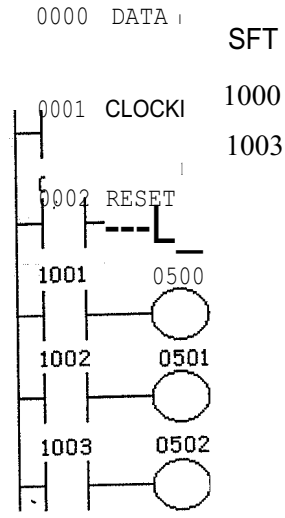


Figure 4.19 A ladder program

Let's now follow the shift register as the operation takes place. Here's what the 1000 series register (the register we're shifting) looks like initially:

lOxx Register

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

Table 4.1

A good cone comes-intfront of the sensor (input 0000). The sensor (<lata input) turns on. 1000 will not turn on until the rising edge of the encoder (input 0001). Finally the encoder now generates a pulse and the status of the <lata input (cone sensor input 0000) is transferred to bit 1000. The register now looks like.

lOxx Register

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

Table 4.2

As the conveying system moves on, another cone comes in front of the sensor. This time it's a broken cone and the sensor remains off. Now the encoder generates another pulse. The old status of bit 1000 is transferred to bit 1001. The old status of 1001 shifts to 1002. The old status of 1002 shifts to 1003. And the new status of the <lata input (cone sensor) is transferred to bit 1000. The register now looks like.

1Oxx Register

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

Table 4.3

Sirıpe the register shows that 1001 is now on, the ladder says that output 0500 will turn orı.ajld ice cream is put in the cone.

As the côrıveying system continues to move on, another cone comes in front of the sensor. Tl:ı.i.stiri:ıe it's a good cone and the sensor turns on. Now the encoder generates another pulsetThe old status of bit 1000 is transferred to bit 1001. The old status of 1001 shiftstô1002. The old status of 1002 shiftsto 1003. And the new status ofthe <lata input (cone seri.sor) is transferred to bit 1000. The register now looks like:

1Oxx Register

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

Table 4.4

Since the register shôws that 1002 is now on the ladder says that output 0501 will tum on and peanuts are put on the cone. Since 1001 now holds the status of a broken cone, 500 remains off in the ladder above and no ice-cream is inserted into this cone. As the conveying system continues to move on, another cone comes in front of the sensor. This time it's also a good cone and the sensor tums on. Now the encoder generates another pulse. The old status of bit 1000 is transferred to bit 1001. The old status of 1001 shifts to 1002. The old status of 1002 shifts to 1003. And the new status ofthe <lata input (cone sensor) is transferred to bit 1000. The register now looks like:

10:xx Register

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

Table 4.5

Since the register shows that 1003 is now on the ladder says that output 0502 will turn on and sprinkles are put on the cone. Since 1002 now holds the status ofa broken cone, 501 remains ôff iri the ladder above and no peanuts are put onto this cone. Since the register sliôwsthatlOOI is now on the ladder says that output 0500 will turn on and ice crearriispufiıı that cone.

As the conveying system continues to move on, another cone comes in front of the sensor. This time it's another broken cone and the sensor turns off. Now the encoder generates another pulse. The old status of bit 1000 is transferred to bit 1001. The old status of 1001 shifts to 1002. The old status of 1002 shifts to 1003. And the new status of the data input (cone sensor) is transferred to bit 1000. The register now looks like:

lOxx Register

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

Table 4.6

Notıd~ tlı.at the status of our fırst cone has disappeared. in reality its sitting in locati6I1T004 but it's useless for us to draw an application with 16 processes here. Suffice iftô say that after the bit is shifted all the way to the left it disappears and is never seeri again. in other words, it has. been shifted out of the register and is erased from myın.pry. Although it's not drawn, the operation above would continue on with each bit shiijiıig on the rising edge of the encoder signal.

The shiffreğister is most commonly used in conveyor systems, labeling or bottling applications, ete. Sometimes it's also conveniently used when the operation must be delayed in a fast moving bottling line. For example, a solenoid can't immediately kick out a bad can of beer when the sensor says its bad. By the time the solenoid would react the can would have already passed by. So typically the solenoid is located further down

the conveyor line and a shift register tracks the can to be kicked out later when it's more convenient.

4.8 Getting and Moving Data

Let's now start working with some <lata. This is what can be considered to be getting into the "advanced" functions ofa plc. This is also the point where we'll see some marked differences between many of the manufacturer's functionality and implementation. Oııthe Iines that follow we'll explore two of the most popular ways to get and manipulate <lata.

Why dôwe want to get or acquire <lata? The answer is simple. Let's say that we are using one Qf the manufacturer' s optional modules. Perhaps it's an A/D module. This modulea.cquires Analog signals from the outside world (a varying voltage or current) and con.vertsthe signal to something the plc can understand (a digital signal i.e. 1 's and O's). Manu:fücturers automatically store this <lata into memory locations for us. However/We have to getthe <lata out of there and move it some place else otherwise the next ana.loğsample willTeplace the one we just took. In other words, move it or lose it! Something else we rn.iğhfwant to do is store a constant (i.e. fancy word for a number), get some binary <lata ôff'the 'input terminals (maybe a thumbwheel switch is connected there, for example). do §ôme foath and store the result in a different location, ete...

As was stated before th~r~are typically 2 common instruction "sets" to accomplish this. Some manufacturers us~cı.:singleinstruction to do the entire operation while others use two separate instructiônsf mP-~\two are used together to accomplish the final result. Let's now look briefly at eachiiıstnfötion.

The single msıruction is com.niori.ly called I\10V (move). Some vendors also include a MOVN (move not). It has the sal'11e function of MOV but it transfers the data in inverted form. The MOV typically looks like that shown below.

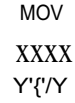


Figure 4.20 MOV instruction symbol

The paired instrucrion typically is called LDA (Load Accumulator) and STA (Stere Accumulator). The accumulator is simply a register inside the CPU where the plc storcs data temporarily while it's working, The LDA instruction typically looks like that shown below. while the STA instruction looks like that shown below to the right.

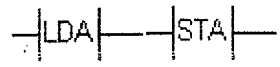


Figure 4.21 Symbols ofLDA and STA

Regardless of whether we use the one symbol or two symbol instruction set (we have no choice as it depends on whose plc we use) they work the same way.

Let's see the single instruction first. The MOV instruction needs to know 2 things from us.

- **Source** (xxxx) - This is where the data we want to move is located. We could write a constant here (2222 for exarnple). This would mean our source data is the number 2222. \Ve could also write a location or address of where the data we want to move is located. If we wrote DMl 00 this would move the data that is located in data memory 100.
- **Destination (yyyy)** - This is the locatiorı where the data will be moved to. We write an address here. For example if we write DM201 here the data would be moved into data memory 201. We could also write 0500 here. This would mean that the data would be moved to the physical outputs. 0500 would have the least significant bit, 0501 would have the next bit... 0515 would have the most significant bit. This would be useful if we had a binary display connected to the outputs and we wanted to display the value inside a counter for the machine operatör at all times (for example).

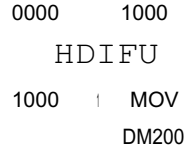


Figure 4.22 A ladder program to move data

The ladder diagram to do this would look similar to that shown above.

Notice that we are also using a "difi.ı" instruction here. The reason is simply because if we didn't the data would be moved during each and every scan. Sometimes this is a good thing (for example if we are acquiring data from an A/D module) but other times it's not (for example an extemal display would be unreadable because the data changes too much).

The ladder shows that each time real world input 0000 becomes true. difi.ı will become true for only one scan. At this time Load 1000 will be true and the plc will move the <lata from data memory 200 and put it into data memory 201. Simple but effective. If. instead of DM200, we had written 2222 in the symbol we would have moved (written) the number (constarıt) 2222 into DM20L

The two symbol instruction works in the same method but looks different. To use them we must also supply two things, one for each instruction:

- LDA- this instruction ıs similar to the source of a I',.1QV instruction, This is where the data we want ta move is located. \\'e could write a ccnstant here (2222 for example). This would mean our source data is the number 2222. We could also write a locaticn or address of where the data we want to n10Ye is located. If we wrote Drv1100 this would move the data that is located in data memory 100.
- STA- this instruction is similar to the destination of a i\.10\T instruction. V./e write an address here. For example if we write DI\·1201 here the dara would be moved into data memory 201. We could also write 0500 here. This would mean that the data would be moved to the physical outnuts. 0500 would have the least significant bit, would have the next bit.. 0515 would have the most significant bit. This would be useful if we had a binary display connected to the outputs and we wanred to display the value inside a counter for the machine operatör at all times (for example).

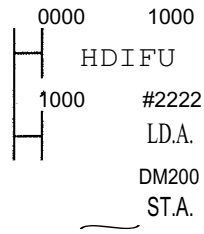


Figure4.23 A ladder program using LDA and STA

The ladder diagram to do this would look similar to that shown above. Here again we notice that we are using a one-shot so that the move only occurs once for each time input 0000 becômes true. In this ladder we are moving the constant 2222 into data memory 200. The "#" is used by some manufactures to symbolize a decimal number. If we just. usecf2222 this plc would think it meant address 2222. PLCs are all the same ... but they ateTalldifferent.

We can think of this instruction as the gateway to advanced instructions. I'm sure you'll find it useful and invaluable as we'll see in future. Many advanced functions are impossible without this instruction.

Chapter 5

NUMBERS AND NUMBER SYSTEMS

5.1 Math Instructions

Let's now look at using some basic matlı functions on our data. Many times in our applications we must execute some type ofmathematicalformula on our data. It's a rare occurrence when our dara is actually exactly what we needed.

As an example, let's say we are manufacturing widgets. We don't want to display the total number we've made töday, but rather we want to display how many more we need to make today to meet outJquota. Let's say our quota for today is 1000 pieces. We'll say X is our current productiörri'I'herefore, we can figure that 1000-X=widgets left to make. To implement this formulawe ôbviously need some matlı capability.

In general, PLCs almost always include these matlı functions:

- **Addition**- The capability to add one piece of data to another. It is commonly called ADD.
- Subtraction- The capability to subtract one piece of data from another. It is commonly called SUB.
- Multiplication- The capahility to multiply one piece of <lata hy another. It is commonly called MUL.
- Division- The capability to divide one piece of data from another. It is commonly called DIV.

As we saw with the MOV instruction there are generally twocommon methods used by the majority of plc makers. The first method includes a single instruction that asks us for a few key pieces of infonn.ation. This method typically requires:

- Source A- This is the address of the first piece of data we will use in our formula. In other words it's the location in memory of where the first "number" is that we use in the formula.

- **Source B-** This is the address of the second piece of <lata we will use in our formula. In other words it's the location in memory of where the second "number" is that we use in the formula. -NOTE:.typically we can only work with 2 pieces of <lata ata time. In other words we can't work directly with a formula like 1+2+3. We would have to break it up into pieces, Dike 1+2=X then X+3= our result.
- **Destination-** This is theaddress where the result oiöutfôrititıla will be put. For example, if 1+2=3, (I hope it still does!), the 3 would automatically be put into this destination memory location.

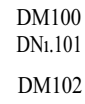


Figure 5.1 ADD symbol

The instructions above typically have a symbol that looks like that shown above. Of course, the word ADD would be replaced by SUB, MUL, DIV, ete. In this symbol, The source A is DMlO(kthe source B is DMI Ol and the destinatioıtisDMl02. .Therefore, the formula is simply whatever value is inDMIOO + whatever value is inDMlOl. The result is automatically stored into DM102.

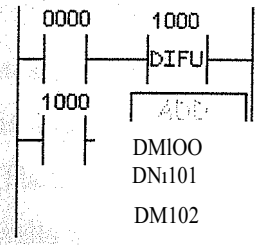


Figure 5.2 A ladder diagram of matlı functions

Shown above is how to use math functions on a ladder diagram. Please note that once again we are using a one-shot instructiorı. As we've seen before, this is because if we didn't use it we would execute the formüla on every scan. Odds are good that we'd only want to execute the function one time when input 0000 becomes true. If we had previously put the number 100 into DM100 and 200 into DM101, the number 300 would be stored in DM102.(i.e. 100+200=300, right??)

Figure 5.3 ADD symbol (dual method)

The dual instruction method would use a symbol similar to that shown above. in this method, we give this symbol only the Source B location. The Source A location is given by the LDA i11ştruction. The Destination would be included in the STA instruction.

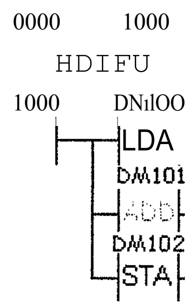


Figure 5.4 A ladder program using DIFU, LDA, ADD and STA

Shown above is a ladder diagram showing what we mean.

The results are the same as the single instruction method shown above.

What would happen if we had a result that was greater than the value that could be stored in a memory location?

Typically the memory locations are 16-bit locations. (More about number types in a later chapter) In plain words this means that if the number is greater than 65535 (2/\16=65536) it is too big to fit. Then we get what's called an overflow. Typically the plc tums on an intemal relay that tells us an overflow has happened. Depending on the plc. we would have different data in the destination location. (DM102 from example) Most PLCs put the remainder here.

Some use 32-bit matlı which solves the problem. (Except for really big numbers) lf we're doing division. for example. and we divide by zero (illegal) thc ovcrflow bit [illegible]pically turns on as well. Suff[illegible] say. check the overflow bit in [illegible] ladder [illegible]

it's true, plan appropriately. Many PLCs also include otlıer matlı ~:pabilities.Some of tlıese functions could include:

- Square roots
- Scaling
- Absolute value
- Sine
- Cosine
- Tangent
- Natura! logarithm
- Base 10 logarithm
- XAY (X to the power of Y)
- Arcsine (tan, cos)
- And more ....clıeck with the manufacturer to be sure.

SomeiPI.t®s>can use floating point matlı as well. Floating point matlı is simply using decimalpöirits:111.ôtlıer words, we could say tlıat 10 divided by 3 is 3.333333 (floating point). Or we.icoüld say tlıat 10 divided by 3 is 3 with a remainder of 1 (long division). Many micro/mifıi J>LCs don't include floating point matlı. Most of larger systems typically do.

5.2 Number Systems

Before we get too/far ahead of ourselves, let's take a look at the various number systems used by PLCs.<Many number systems are used by PLCs. Binary and Binary Coded Decimal are populariwlıile octal and lıexadecimal systems are also common.

Let's Jook at each:

As we do, consider tlıe followirı.ğformula (Matlı again!):

$$Nbase = Ddigit*R/\unit + ... DlR/\1 + DOR/\0$$

Wlıere D=the value of tlıe digit and R= # of digit symbols used in tlıe given number system.

The "*" means multiplication. (5 * 10= 50)

The "A" means "to the power of'.

Where D=the value ofthe digit and R= # of digit symbols used in the given number system.

The "*" means multiplication. (5 * 10 = 50)

The "^" means "to the power of'.

As you'll recall any number raised to the power of 0 is 1. lQAI=lO, lQ/\2 is 10x10=100, 10/\3 is lOxlOxlO=lOOO, lQ/\4 is lOxlOxlOxlO=lOOOO...

This lets us convert frorr.i.aı:ıy number system back into decimal.

5.2.1 Decimal

This is the numbering system we use in everyday life. (well most ofus do anyway!) We can think of this as base 10 counting. it can be called as base 10 because each digit can have 10 different states. (i.e. 0-9) Since this is not easy to implement in an electronic system it is seldom, if ever, used. Ifwe use the formula above we can find out what the number 456 is. From the formula:

Nbase=Ddigit" RAunit + ... D1RA1 + DORAO.

We have (since we're doing base 10, R=lO)

NlO= D410/\2 + D510/\1 + D610AO

= 4*100 + 5*10 + 6*

=400 + 50 + 6

=456.

5.2.2 Binary

This is the numbering system computers and PLCs use. It was far easier to design a system in which only 2 numbers (0 and 1) are manipulated (i.e. used). The binary system uses the same basic principles as the decimal system. In decimal we had 10 digits. (0-9) in binary we only have 2 digits (0 and 1). In decimal we count: 0,1,2,3,4,5,6,7,8,9, and instead of going back to zero, we start a new digit and then start from 0 in the original digit location. In other words, we start by placing a 1 in 'the .second digit location and begin counting again in the original location like this 10,11,12,13, ... When again we hit 9, we increment the second digit and start counting from 0 again in the original digit location. Like 20,21,22,23 .... of course this keeps repeating. And when we run out of

digits in the second digit location we create a third digit and again start from scratch. (i.e. 99, 100, 101, 102...). Binary works the same way. We start with 0 then 1. Since there is no 2 in binary we must create a new digit. Therefore we have 0, 1, 10, and 11 and again we run out of room. Then we create another digit like 100, 101, 110, and 111. Again we ran out of room so weadd another digit... Do you get the idea? The general conversion formula may clear things up:

Nbase=Ddigit * Rounit +.... DlR"1 + DOR"O.

Since we're now doirigbiıiary or base 2, R=2. Let's try to convert the binary number 1101 back into decimal,

NlO= Dl * 2"3 +DO* 2"1 + Dl * 2"0

= 1*8 +

= 13

(Ifyou don't see 2, and 1 came from, refer to the table below).

Now we can see thaf\binatyJlOl is the same as decimal 13. Try translating binary 111. You should get debirnal/7.Try binary 10111. You should get decimal 23.

Here's a simple binaty<cfı.art/föreference. The top row shows powers of 2 while the bottom row shows their eqtiivalent decimal value.

<.!~~.ry Number Conversions

·-·2"fsr2AT4-r2"13 2"12121112"10 f"27'9 12"s 12"112"6 12"s 12"412"3:2"2h"1 :2"o1

32768l16384;8192!4096)2Q4~J1024)512j256j128j 64 j 32 16 j 8 4 2 1 j

Table 5.1

5.2.3 Octal

The binary number system requires a ton of digits to represent a large number. Consider that binary 11111111 is only decimal 257. A decimal number like 1,000,000 ("1 million") would need a lot of binary digits! Plus it's also hard for humans to manipulate such numbers without making mistakes.

So we count like 0.1,2.3.4.5.6. 7.10.11.12 ... 17.20.21.22 ..27.30 ....

Using the formula again. we can convert an octal number to decimal quite easily.

Nbase= Ddigit * RAunit +.... D1R/\1 + DORAO

So octal 654 would be: (remember that here R=8)

NlO= D6 * 8/\2 +DS * 8/\1 + D4 * 8/\0

= 6*64 + 5*84-4*1

= 384 +40+4

=428

(If you don't see where the white 64, 8 and 1 came from, refer to the table below).

Now we can see that octal 321 is the same as decimal 209. Try translating octal 76. You should get decimal 62. Try octal 100. You should get decimal 64.

Here's a simple octal chart for your reference. The top row shows powers of 8 while the bottom row shows their equivalent decimal value.

Octal Number Conversions

8/\7	8^6	8^5	8^4	8^3	8^2	8^1	8/\0
2097152	262144	32768	4096	512	64	8	1

Table 5.2

Lastly. the octal system is a convenient way for us to express or write binary numbers in plc systems, A binary number with a large number of digits can be conveniently written in mi octal form with fewer digits, This is because i octal digit actually represents 3 binary digits,

Believe me that ,vht!Il\·Vc start working with register dara or address locations in the advanced chapters it becomes a great way of expressing <lata. The following chart shows what wc're referring to:

Binary Number with its Octal Equivalent

1	1	1	0	0	1	0	0	1	1	1	0	0	1	0	1
1		6			2			3			4			5	

Table 5.3

From the chart we can see that binary 1110010011100101 is octal 162345. (Decimal 58597) As we can see, when we think ofregisters, it's easier to think in octal than in binary. As you'll soon seethough, hexadecimal is the best way to think.

5.2.4 Hexadecimal

The binary numb:efsystem requires a ton of digits to represent a large number. The octal system iinptôvesiipôr:i.this. The hexadecimal system is the best solution however, because it allows<~~0jô'J::: even less digits. it is therefore the most popular number system used with computersand PLCs. (we should learn each one though) The hexadecimal system is also ret~~':tt •as base 16 or just simply hex. As the name base 16 implies, it has 16 digits. The fügits are

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

So we count like

0, 1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,10,11, 12,13, ...

IA,lB,lC,lD,lE,lF,20,21... 2A, 2B, 2C, 2D, 2E, 2F,

Using the formula again, we can convert a hex number to deciinal quite easily.

Nbase= Ddigit * RAunit +.... D1RAI + DORAO

So hex 6A4 would be:(remember here that R=l6)

NlO= D6 * 16/\2 +DA* 16/\1 + D4 * 16/\0

= 6*256 + A(A=decima110)*16 + 4*1

= 1536 +160 +4

= 1700

(if you don't see where the 256, 16 and 1 came from, refer to the table below)

Now we can see that hex FFF is the same as decimal 4095. Try translating hex 76. You should get decimal 118. Try hex 100. You should get decimal 256.

Here's a simple hex chart for reference. The top row shows powers of 16 while the bottom row shows their equivalent decimal value. Notice that the numbers get large rather quickly.

Hex Number Conversions

16/\8	16/\7	16/\6	J6A5	16/\4	16^3	16^2	16^1	16^0
4294967296	268435456	16777216	1048576	65536	4096	256	16	1

Table 5.4

Finally, the hex system is perhaps the most convenient way for us to express or write binary numbers in plc systems. A binary number with a large number of digits can be conveniently written in hex form with fewer digits than octal. This is because 1 hex digit actnally represents 4 binary digits.

Believe ıne that when we start working with register <lata or address locations in the advanced chapters it becomes the best way of expressing data. The following chart shows what we're referring to:

Binary Number with its Hex Equivalent

0	1	1	1	0	1	0	0	1	0	0	0	1	0	1
	7				4				A				5	

Table 5.5

From the chart we can see that binary 0111010010100101 is hex 74A5. (Decimal 29861) As we can see, when we think of registers, it's far easier to think in hex than in binary or octal.

5.3 Boolean Matlı

Let's now take a look at some simple "Boolean matlı". Boolean matlı lets us do some vary basic functions witlı the bits in our registers. These basic functions typically include AND, OR and XOR functions. Eaclı is described below.

5.3.1 AND Gate

This function enables us to use the trutlı table below. Here, we can see that the AND function is very much related to multiplication. We see this because the only time the Result is true (i.e. 1) is when botlı operators A AND B are true (i.e. 1). The AND instruction is useful wlıen your plc doesn't have a masking function. Oh yeah, a masking function enables a bit in a register to be "left alone" wlıen working on a bit level. Tlıis is simply because any bit that is ANDed with itself will remain the value it currently is. For example, .if you wanted to clear (make them 0) only 12 bits in a 16 bit register you might AND<theregister with 0's everywhere except in the 4 bits you wanted to maintain the status

See the truth table below to figure out what we mean. (1 AND 1 = 1, 0 AND 0= 0)

		Result = A AND B
A		Result
0	0	0
1	0	0
0	1	0
1	1	

Table 5.6

5.3.2 OR Gate

This function based upon the truth table below. Here; we can .see that the OR function is very much related to addition. We see this because the only time the Result is true (i.e. 1) is when operator A OR B is true (i.e. 1). Obviously, when they are both true the result is true. (If A OR B is true...)

Result = A OR B

A	B	Result
0	0	0
1	0	1
0	1	1
1	1	1

Table 5.7

5.3.3 EXCLUSIVE ORGate

This function enables us to use the truth table below. Here, we can see that the EXOR (XOR) function is not related to anything I can think of ! An easy way to remember the results of this function is to think that A and B must be one or the other case, exclusively. Huh? In other words, they must be opposites of each other. When they are both the same (i.e. A=B) the result is false (i.e. 0). This is sometimes useful when you want to compare bits in 2 registers and highlight which bits are different. It's also needed when we calculate some checksums. A checksum is commonly used as error checking in some communications protocols.

Result = A XOR B

A		Result
0		0
1	0	1
0	1	1
1	1	0

5.8

The ladder logic instructions are comrıionly called AND, ANDA, ANDW, OR, ORA, ORW, XOR, EORA XORW.

As we saw with the MOV instruction there are generally two common methods used by the majority of plc makers. The first method includes a single instruction that asks us fora few key pieces of information. This method typically requires:

- **Source** A- This is the address of the first piece of data we will use. In other words its the location in memory of where the A is.
- **Source B-** This is the address of the second piece of data we will use. In other words its the location in memory of where the B is.
- **Destination-** This is the address where the result will be put. For example, if A AND B = 0 the result (0.) would automatically be put into this destination memory location.

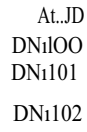


Figure 5.5 AND symbol

The instructions above typically have a symbol that looks like that shown here. Of course, the word AND would befeplaced by OR or XOR. In this symbol, The source A is DMl 00, the source B is DMlOt~~ciJ~e destination is DMl 02. Therefore, we have simply created the equation DM100.<.A..ND DM101 = DM102. The result is automatically stored into DM102. The Bôôlean fu11ctions on a ladder diagram are shown below.

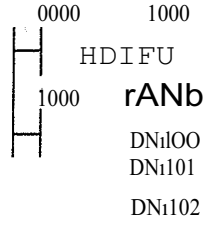


Figure 5.6 A ladder program using AND

Please note that once again we are using a one-shot instruction. As we've seen before, this is because if we didn't use it, we would execute the instruction: on every scan. Odds are good that we'd only want to execute the function 011e time when input 0000 becomes true,

Figure 5.7 AND symbol (dual instruction method)

The dual instruction method would use a symbol similar to that shown above. In this method, we give this symbol only the Source B location. The Source A location is given by the LDA instruction. The Destination would l:,~j11cludedjn the STA instruction. Below is a ladder diagram showing what is meant.

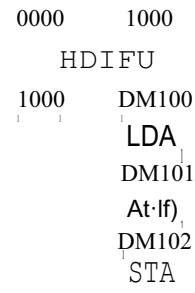


Figure 5.8 A ladder program using DIFU, LDA, AND and STA

The results are the same as the single instruction method shown above. It should be noted that although the symbol and ladder diagram above show the AND instruction, OR or EXOR can be used as well. Simply substitute the word "AND" within the instruction to be either "OR" or "EXOR". The results will be the same as shown in their respective truth tables.

We should always remember that the theory is most iinportant. If we can understand the theory of why things happen as they do, we can use anybody's plc. Ifwe refer to the manufacturers documentation we can find out the details for the particular plc we are using. Try to find the theory in that documentation and you might come up short. The details are insignificant while the theory is very significant.

Chapter 6

WIRING OF PLC

6.1 DC Inputs

Let's now take a look at how the input circuits of a plc work. This will give us a betler understanding of how we should wire them up.

Typically, de input modules are available that will work with 5, 12, 24, and 48 volts. Be sure to purchase the one that fits your needs based upon the input devices you will use.

We'll first look at how the de 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, ete.) we typically don't have to worry about whether we wire it as NPN or PNP. We should note that most PLCs won't Jet us mix NPN and PNP devices on the same module. When we are using a sensor (photo-eye, prox, ete.) we do, however, haveto '\VQrryalJoutits output configuration. Always verify whether it's PNP or NPN. (Check with the manufacturer when unsure)

The difference between the two types is whether the load (in our case, the ole 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 Joad switched to positive voltage. Below is what the outputs look Iike for NPN and PNP sensors.

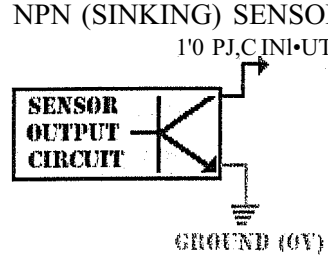


Figure 6.1 NPN sensor

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. NPN sensors are most commonly used in North America.

Many engineers will say that PNP is berter (i.e. safer) because the load is switched to ground, but whatever works for you is best. Just remember toplan for the worst.

On the PNP sensor we connect öne 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.

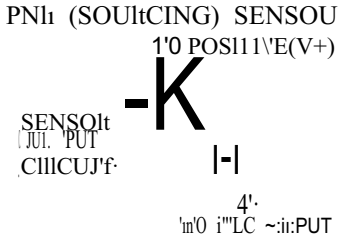


Figure 6.2 PNP sensor

inside the sensor, the transistor is just acting as a switch. The sensors intemal 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).

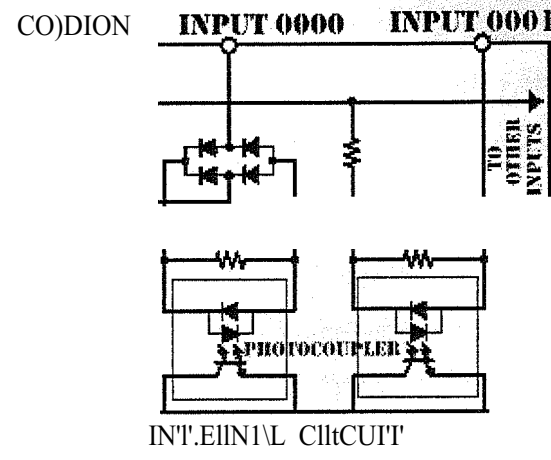


Figure 6.3 internal circuit ofa sensor

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 it's connectedrl~ upon the type of sensor used. When using an NPN sensor this is connected to V+. When using a PNP sensor this terminal is connected to OV (groun.d).

A common switch(i.e. limit switch, pushbutton, toggle, ete.) would be connected to theinputs jpJı sirrı.ilaffasbion. üne side of the switch would be connected directly to v+. The ~:,rm,~ntı~es to tlıe p]c input terminal.This assumesthe common terminal is connecte~~?9;'.(ground). If the common is connected to V+ then simply connect one end oftheswitch to OV (ground) and the other end to the plc input terminal.

Th6phôto couplers are used to isolate the PLCs intemal circuit from the inputs. This elinıinatesthe chance of any electrical noise entering the intemal circuitry. They work b)'côrıverting the electricalirıplitsigrıal to light and then by converting the light back to anelectrical signal to be processed b)'theintemal circuit.

6.2 AC Inputs

Now that we understand how de inputs wôrk,ilet's take .a close look at ac inputs. An ac voltage is nott-polarized. Put simply, this means thatthere is no positive or neganve to "worry about". However, ac voltage can be quite dangerous to work with if we are careless. (Remember when you stuck the knife in the toaster got a shock? Be careful) typically, ac input modules are available that will work with 24, 48, 11 0, and 220 volts. Be sure to purchase the one that fits your needs based upon the input devices (voltage) you will use.

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

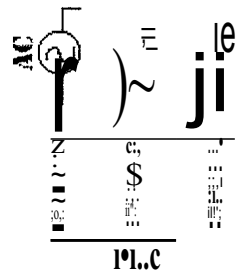


Figure 6.4 shows the connecting ofa PLC

We typically conneet an ac device to our input rnodule as shown above. Cornrnonly the ac "hot" wire is ccnnected to the switch while the "neutral" goes to the plc cornrnon. The ac ground (3rd wire where applicable) should be connected to the frarne ground terminal of the plc.(not shown) As is true with de, ac connections are typically color coded so that the individual.wiring the device knows which wire is which. This coding varies frorn country to country but in the US is cornrnonly white (neutral), black (hot) and green (3rd wire ground when applicable). Outside the US it's cornrnonly 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.

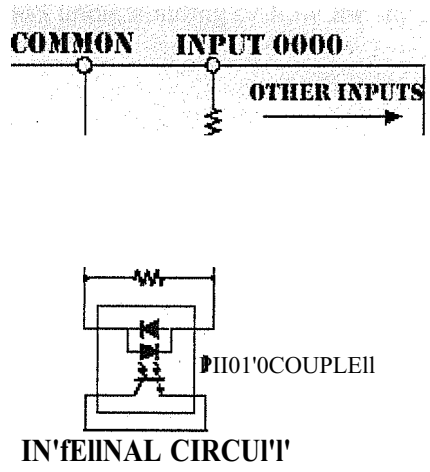


Figure 6.5 PLCs ac input rnodule circuit

The only things accessible to the user are theterminals la.beledCOMMON, INPUT 0000, INPUTxxxx... The cornrnon terminal gets connected to' the neutral wire. A

common switch (i.e. limit switch, pushbutton, toggle, ete.) would be connected to the input terminals directly. üne 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 manufacturer's specifications before wiring, to be sure AND SAFE.

The photo couplers are used to isolate the PLCs intemal circuit from the inputs. This eliminates the chance of any electrical noise entering the intemal circuitıy. 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 intemal circuit.

üne last note, typically an ac input takes Jonger than a de input for the plc to see. ln most cases it doesn't matter to. the programmer because an ac input device is typically a mechanical switch and mechanical devices are slow. lt's quite common for a plc to require that the input be on for 25 or more milliseconds before it's seen. This detay is required because of the filtering which is needed by the plc intemal circuit. Remember that.the plc intemal circuit typically works with 5 or less volts de.

6.3 Relay Outputs

Bynowwe should have a good understanding of how the inputs are used. Next up is the output circuits.

üne of the most common types of outputs available is the relay output. A relay can be used with both AC and DC loads. A load is simply a fancy word for whatever is connected to our outputs. We call it a load because we are "loading the output" with something. lf we connected no load to the output (i.e. just connect it directly to a power supply) we would certainly damage the outputs. This would be similar to replacing the light bulb in the lamp you're using to read this with a piece of wire. lf you did this, the lamp would draw a tremendous amount of current from the outlet and certainly pop your circuit breaker or blow your fuse or your brains.

Some common forms ofa load are a solenoid, lamp, motor, ete. These "lôads" come in all sizes. Electrical sizes, that is. Always check the specifications of yourrload before connecting it to the plc output. You always want to make sure that the maximum current

it will consume is within the specifications of the plc output. If it is not within the specifications (i.e. draws too much current) it will probably damage the output. When in doubt, double check with the manufacturer to see if it can be connected without potential damage.

Some types of loads are very deceiving. These deceiving loads are called "inductive loads". These have a tendency to deliver a "back current" when they turn on. This back current is like a voltage spike coming through the system.

A good example of an inductive load that most of us see about 6 months per year is an air conditioning unit. Perhaps in your home you have an air conditioner. (Unless you live in the arctic you probably do!) Have you ever noticed that when the air conditioner "kicks on" the lights dim for a second or two. Then they return to their normal brightness. This is because when the air conditioner turns on it tries to draw a lot of current through your wiring system. After this initial "kick" it requires less current and the lights go back to normal. This could be dangerous to your PLCs output relays. It can be estimated that this kick is about 30 times the rated current of the load. Typically a diode, varistor, or other "snubber" circuit should be used to help combat any damage to the relay. Enough said. Let's see how we can use these outputs in the "real plc world".

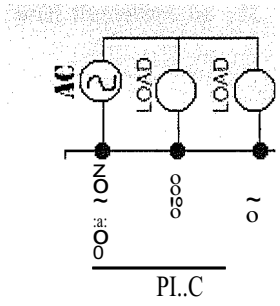


Figure 6.6 PLC connected to AC source

Shown above is a typical method of connecting our outputs to the plc relays. Although our diagram shows the output connected to an AC supply, DC cari be iised as well. A relay is non-polarized and typically it can switch either AC ôr DC. Here the common is connected to one end of our power supply and the other end ôfthe supply is connected to the load. The other half of our load gets connected fothe actual plc output you have designated within your ladder program.

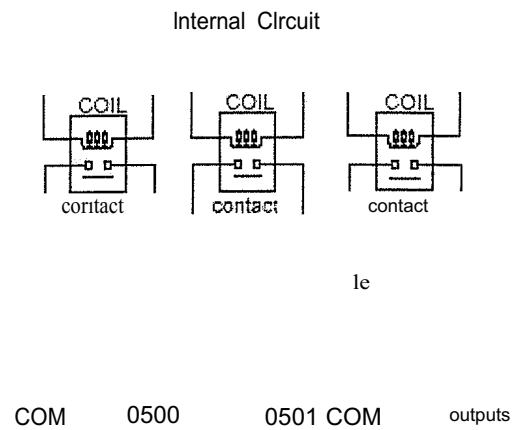


Figure 6.7 Relay as in PLC

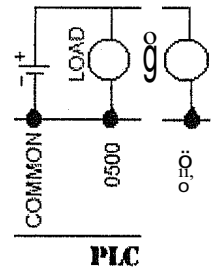
The relay is internal to the plc. Its circuit diagram typically looks like that shown above. When our ladder diagram tells the output to turn on, the plc will internally apply a voltage to .the relay coil. This voltage will allow the proper contact to close. When the contactcloses, an external current is allowed to flow through our external circuit. When the ladde[diagram telis the plc to turn off the output, it will simply remove the voltage from theifüerrıal circuit thereby enabling the output contact to release. Our load will than have atiöp~ti?circuit and will therefore be off.

6.4 Transistor Outputs

The next type of outpli{M'~ shoulcl leam about is our transistor type outputs. it is important to note that a transistôf/carionly switch a de current. For this reason it cannot be used with an AC voltage.

We can think ofa transistor asa solid-stat~sM'itch.)Or more simply put, an electrical switch. A small current applied to the transistôrs\"ba.self (Le. input) lets 'us switch a much larger current through its output. The plc applies a sföallcurrent to the rransistor base and the transistor output "closes". When it's closed,thecI~viceCörinectedtothe plc output will be turned on. The above is a very simple explaria.tioniöfatransistor. There are, of course, more details involved but we don't need to geftöoideep. We should also keep in mind that as we saw before with the input circuits,there a.regenerally 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 ofthe common types available are BJT and MüSFET. A BJT type (Bipolar Junction Transistor) often has less switching capacity (i.e. it can switch less current) than a MüS-FET (Metal üxide Semiconductor- Field Effect Transistor) type. The BJT also has a slightly faster switching time. ünce again, please check the output specifications of the particular plc you are going to use. Never exceed the manufacturer' s maximum switching current.



Figuı:e6.8 PLC connected to NPN type transistor

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. üne end ofthe load is comiected 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.

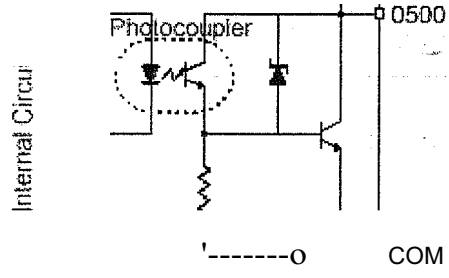


Figure 6.9 Circuit diagram for an NPN type output

Notice that as we saw with the transistor type inputs, there is a photo coupler isolating the "real world" from the intemal circuit. When the ladder diagram calls for it, the intemal circuit tums on the photo coupler by applying a small voltage to the LED side of the photo coupler. This makes the LED emit light and the receiving part of the photo coupler will see it and allow current to flow. This small current will tum on the base of the output transistor connected to output 0500. Therefore, whatever is connected between COM ancl ()500 will turn on. When the ladder tells 0500 to tum off, the LED will stop emitting Iight and hence the output transistor connected between 0500 and COM will tum off.

üne öther 'important thing to note is that a transistor typically canrıot switch as large a load asa relay. Check the manufacturer's specifications to fınd the largest load it can safely switch. If the load current you need to switch exceeds the specification of the output, you can corın~ctteplc output to an extemal relay. Then conrıect the relay to the large load. You may by thinking, "why not just use a relay in the first place"? The answer is because a relayisi0~f0 the correct choice for every output. A transistor gives you the opportunity to usy y:xtemalrelays when and only when necessary.

in summary, a transistor is fast, switches a small current, has a long lifetime and works with de only. Whereas a relay is slow, can switch a large current, has a shorter lifetime and works with ac or de. Select the appropriate one based upon your actual application needs.

Chapter 7

COMMUNICATIONS WITH PLC

7.1 Communications History

By far, the rnost popular rnethod of cornrnunicating with extemal devices is by using the "RS-232" communications rnethod. Cornrnunication with extemal devices is viewed by rnany plc prograrnrners to be difficult if not "all but impossible" to understand. This is far frorn true! It's not "black art", "witchcraft" or "weird science". Read on...

All plc cornrnunication systerns have their roots in the old telegraph we rnay have seen in the old movies. (Remember the guy working at the train station with the arın band and plastic visor?) Early attempts to communicate electronically over long distances began as early as the late 1700's. in 1810 a German man (von Soemmering) was using a device with 26 wires (1 for each letter of the alphabet) attached to the bottom of an aquariurn. When current passed through the wires, electrolytic action produced small bubbles. By choosing the appropriate wires to energize, he was able to send encoded messages "via bubbles", (It's true...really) This then caught the attention of the rnilitary and the race to find a systern was on.

in 1839, 2 Englishmen, Cooke and Wheatstone, had a 13 milejtelegraph in use by a British railroad. Their device had 5 wires powering small electromagnets which deflected low-mass needles. By applying current to different combinations of 2 wires at a time the needles were deflected so that they pointed to Jetters of the alphabet arranged in a matrix. This "2 of 51' code only allowed 20 combinations so the letters "z, v, u, q,j and c" were omitted. This telegraph was a big step for the time, but the code was ıföt binary (on/off) but rather it was binary (the needle moved left, right, or not at'alf):

The biggest problerns with these devices was the fact that they .were parallel (required multiple wires). Cooke and Wheatstone eventually made a<tw9 wire .device but the first practical fully serial binary system generally gets creditedto S.F.B. Morse. in Morse code, characters are symbolized by dots and dashes (binary- 1's and O's).

Morse's first system isn't like we see .today.In .the movies. (It's on display at the Smithsonian in DC if you want to see it) It actually had a needle contacting a rotating drum of paper that made a [illegible] mark. By energizing an electromagnet the needle would "bounce" away from the paper creating a space. Very soon telegraph operators noticed that they didn't have to look at the paper to read the code but they could interpret the code by the sound the needle made when scratching the paper. So this device was replaced by a sounder that produced click sounds instead of paper etchings. Teleprinters came later, and today's serial communications devices are more closely related to them. The rest is history... extinct, but history anyway!

Incideritally, the tenns·MARK and SPACE (we'll see them later) originated from Morse's otiginal device. When the needle contacted the paper we called this a MARK and wherı the needle bounced it was called a SPACE. His device only produced UPPERCASE letters which wasn't a big problem though. Further, the Titanic sinking "standardized" the code of "SOS" which means "Save Our Ship" or if you were ever in the US military you might know it berter as "S*%$ Ona Shingle" which was chipped beef on bread.

7.2 **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 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 systeın(l'sarı.d O's) to transmit our <lata in the ASCII format. (American Standard Code for Inforınation• Interchange- pronounced ASS-KEY) This code translates human readable / .code (letters/numbers) into "computer readable" code (1 's and O's). Our plcs seriafpôrtiis used for transmission/reception of the data. It works by sending/receiving a\lôltage. A positive voltage is called a MARK and a negative voltage is called aSPACEYTypically,

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 co111111öfle:x#ınple)s a computer. The other type is called a DCE device. DCE means Data C6111m1.ır1ications Equipment and a common example is a modem. Your plc may be either#tı:>m~<9rDCE device. Check your documentation.

The plc serial port works by tuming 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 a DTE device)

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

Table 7.1

Each pins purpose in detail:

- Frame ground- This pin should be intemally 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 extemal device.
- Data terminal ready- This pin is a master control for the external device. When this pin is I the extemal 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 asa permanent 0 and the plc basically uses it to determine thatthe extemaldevice is powered up and ready.
- Request to send- This is part of hardware handshakirıg. When the plc wants to send data to the extemal device it sets this pin to a 0. Iıfother words, it sets the pin to a 0 and basically says "I want to send you <lata. Is it ok?" The' extemal device says it's OK to send data by setting its clear to send piiito 0. The plc then sends the da.ta.
- Clear to serıd2 This is.the ôthel' half of hardware handshaking. As noted above, the external device setsthis pirJifô{Owhenit is ready to receive <lata from the plc.
- Ring indicator- only used wheıifü~iplcfiiscôrıiiected to a modem.

What happens when your plc and extemal device are either 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.

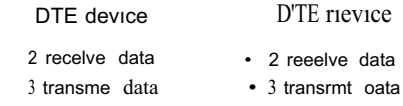


Figure 7.1 DİE devices

Notice that in the picture above, the receive <lata line (pin2) of the fırst device is connected to the receive data line ofthe second device. And the transmit data !ine (pin3) ofthe first device is connected to the transmitdata ofthe second device. .It's Iike jalking 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.

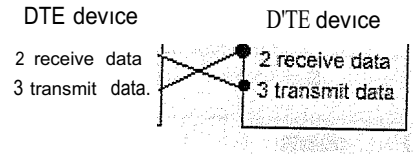


Figure 7.2 A typical communications session

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 'seme data?" The external device responds by turning on it's CTS wnrcn 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]:>le 'rurıs ôtıt of data to send and turns offits RTS terminal. The external device sits and waitsforınôre•data.

7.3 RS-232 Communications (software)

Now that we understand the hardware part of the picture, let's dive right into the software part. We'll take a look at each part of the puzzle by defining a few of the common terms. Ever wondered what phrases like 9600-8-N-l meant? Do you use software-handshaking or hardware-handshaking at forma! parties for a greeting? If you're not sure, read on!

- ASCII is a human-readable to computer-readable translation' code. (Le, each letter/numberis translated to 1's and 0's) It's a 7-bit (a bit is a 1 ora 0) code, so we can translate 128 characters. (2"7 is 128)Character setsthatuse the 8th bit do exist but they are not true ASCII.Below. is an ASCil.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, ete. (the "h" simple means hexadecimal)

Least sig. bits	most significant 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 7.2

- Start **bit-** In RS-232 the first thing we send is called a start bit. This startrbit ("invented" during WWJ by Klein Schmidt) is a synchronizing bitiadde:djust before each character we are sending, .This is considered a SPA.CE on negative voltage ora 0.
- Stop **bit-** The !ast thing we send is called a stop bit. This stop bit.telis us tharthe last character was just sent. Think 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/extemal equipments are byte-oriented (8 bits=lbyte) it seems natura] to handle data asa byte. Although ASCII is a 7-bit code it is rarely transmitted that way.yypically, the 8th bit is used as a parity bit for error checking. This method of errorichecking gets its name from the math idea of parity. (Rernember the ödd-ey~:füipfpp~i"tyof integers? I didn't think so.) in simple terıns, parity means that all characters \\'ili e:itherhave an odd number of 1's or an even rıumber of 1's.Commcü' fotms ôf pa:rityafeNone:, Bven, and Odd. (Mark and Space aren't very common sö Fwônrf'discuss'theın) .. Consider these examples: send "E" (45h or 1000101 (binary))I:frparityôfNône,theparity bit is always 0 so we send 1000101O.In parity of eveilwe,ffiusf.have>an Even number of 1's in our total character so the original characte:ı<currentlyhas3 il's (1000101) therefore our parity bitwewill add must be a L(l0001011)nöw vve have an even number of 1's. In Odd parity 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, right?) 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 theresult tothe: parity bit received. Ifthe calculated and real parity bits don't match, an errôr occurred an we act appropriately. It's strange that this parity method is so popular. The reason because it's only effective halfthe time. That is, parity checking can only find errors that affect 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 affects 1 bit. Block redundancy checks are used in other communication methods to prevent this.
- Baud rate- I'II perpetuate the incorrect meaning since its 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 sentand 9600 means 9600 bits are being transmitted every second. Çoı.n1119n values (speeds) are 1200, 2400, 4800, 9600,19200, and}84()0.
- RS232 data format- (baud Tı:tte-data(t>itpı:ırity-5tpppits)i'I'füs\isithe 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 <lata leaves the serial port for the character "E" (45h 100 0101b) and even parity.

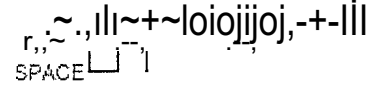


Figure 7.3 Flow control

Another important thing that is sometimes used is called spftware bandshaking (flow control). Like the hardware handshaking we saw in the previous chapter, software handshaking is used to make sure both devices are ready tosend/receivedata. The most popular "character flow control" is called *XON!XOFF.* lt's veıysiınpleto understand. Simply put, the receiver sends the XOFF character when it wants -rhe transmitter to pause sendirıg <lata. >When it's -ready to receive <lata 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 teli the receiver to process the data it has received. The most common is the CR or the CR and LF pair. The CR (carriage return) is like the old typewriters. When you reached the end ofa line while typing, a bell would sound. You would then grab the handle and move the carriage back t6thestart.In•öther words, you retumed the carriage to the beginning. (This is thesame a.s what a Clfdeliıniter will do if you view it ona computer screen.) The plc/external device reCeives this and knows to take the <lata from its buffer. (Where 'the data is stored temporarily before being processed) An LF (line feed) is also sometimes sent with the CR character. lfviewed 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 transmissiofı/recepfiofi' as well. STX is "start of text" and ETX is "end of text". The STX is sent before the data a.riô telis füe external device that <lata is coming. After ali the da.ta has beeii' seni, all ETX 6haracter 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.

7.4 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 pk is a plc... in other words, understand the theory first and then figure out how our" manufacfurer 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 (i.e. spend extra $'s) a module to "talk RS-232" with an extemal device. What is an extemal device, you maybe asking? The answer is difficult because there are so many extemal devices. It may be an operator interface, an extemal computer, a motor controller, a robot, a vision system, a ... get the point??

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

- Where, in <lata memory, will we stere the data to be Sent? Essentially we have to store the data we will send... somewhere. Where else but in our data memory!
- Where, in data memory, will we put the data we .receive from the extemal device?
- How will we tell the plc when it's time to send our data (the data we stored in data memory) out the serial port?
- How will we know when we have received <lata from our extemal device?

If you know the above, then the rest is easy. Ifyou don't knowtheabcıve,.then make something up and now the rest is easy. Huh??? Simple, pick a .rnemory area to .work with and figure out if we can choose the intemal relays .to use .to send and receive dara or ifthe plc has ones that are dedicated to this purpose,

Before we do it, let's get some more technical terms out of the way so we're on the same playing field.

- Buffer- A buffer is a fancy technical word that means a plastic bag. In other words, it's a temporary storage Jocation where the plc or extemal device stores data it has received (or is waiting.t,o se11d) via RS-232. When I go to the supermarket to buy my favorite TV di11ners,Tqı:ı.rry.them home in a plastic bag. The plastic bag is' not a permanent place for ΠlY foed (are TV dinners really food??) but rather a temporary storage place föl"thern until I get home. When I get home, I take them out of the bag and cook thenı. The supermarket was the extemal device where I got the data (TV dinners) from.ahd rny microwave is the plc. The plastic bag was the buffer (temporary storage place) that was holding my data (TV dinner) uiıtil I took them out to use (i.e. cook).
- String- A string is a cool way of saying "a bunch of characters". The word "hello" is a. string. Jt's a bunch of characters (i.e. h-e-1-1-o) that are connected (strung) toğethertcrnı~aıısônıethirig useful. "43770" is also a string. Although it makes no sense to us,if~~y'.~e}~/()ınething valuable to your plc or extemal device. It could be a command thattells your robot to send out its current coordinates. (or it could simplyhe:th~/'\V()td"hellô'f.upside down)
- Concatenate- This word is a mouthful.Sinıpl)'put, it mearıs to combine 2 strings together to make one string. An exaımple is conıbiıiing the 2 strings "laser" and "jet" together to make one string... "LaserJet".

With the mumbo-jumbo out of the way Jet's see it in action. .Again, the memory locations and relays vary by manufacturer but the theory is universal.

1. We assign memory locations DMIOO through DM102 to be where we'll put our data before we send it out the serial port. Note- Many PLCs have dedicated areas of memory for this and only this purpose.
2. We'll assign intemal 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 theserial port to our external device. Note again- Many PLCs have dedicated relays (special utility relays) for this and only this purpose. It's great when the manufacturer makes our life easy!

We'll send the string "alr" 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 yoıi'llseethat "alr" is hexadecimal 61, 6C, 72. (a=61, 1=6C, r=72) We'll write these ASCII characters (in hexadecimal form) into the individual data memory locations. We'll use DMI00-102. How? 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.

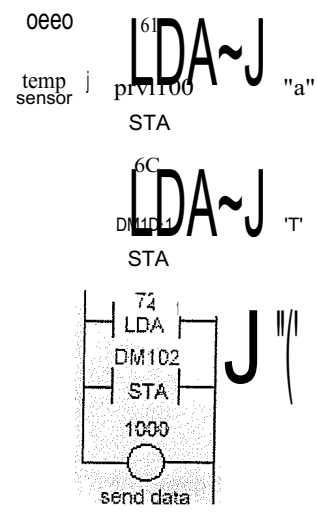


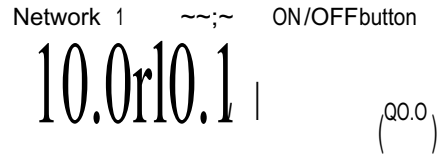
Figure 7.4 Ladder diagram

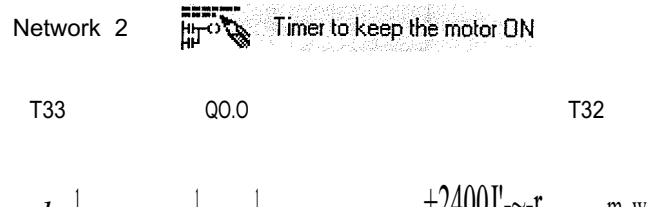
Some PLCs may not have dedicated internal relaysthatsend out ôur data through the RS-232 port. We may have to assign them manually. Further, some PLCs will have a special instruction to tell us where the data is stored and when to send the data. This instruction is commonly called AWT (ASCII Write) or RS. The theory is always the same though. Put the data in a memory location and then turn on a relay to send the data.

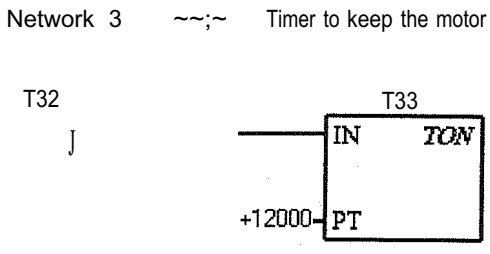
Chapter 8

Programming Siemens Simatic S7-200

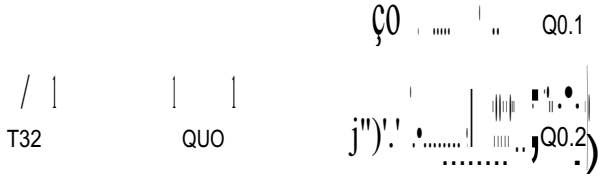
8.1 Ladder Program



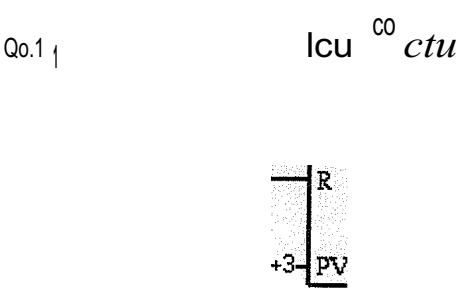




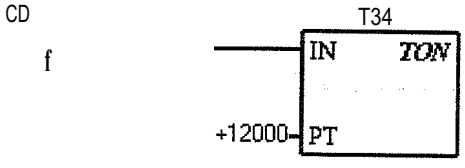
Network 4 ~~;' Timer and Counter dependent functioning of motor and lamp

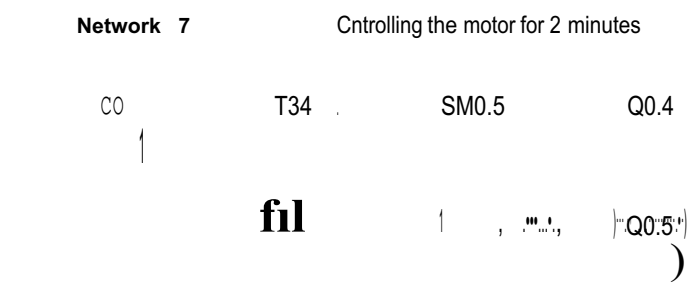


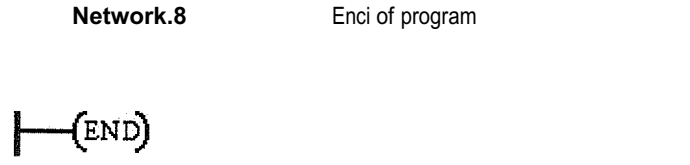
Network 5 ~~;' Counter counting the process



Network 6 Timer







8.2 Statement Line Program

```
11AC motor controlling

NETWORK 1 IIONIOFF button
il
1/NETWORK COMMENTS
il
LD    I0.0
O     Q0.0
AN    IO.1
      Q0.0

NETWORK 2 IITimer to keep the motor ON
```

```
LDN    T33
A      Q0.0
TON    T32,,+24000

NETWORK  3 //Timer  to keep  the motor OFF
LD     T32
TON    T33,  +12000

NETWORK  4 //Timer  and  counter  dependent  functioning of motor and lamp
LDN    T32
A      Q0.0
AN     C0
=      Q0.1
=      Q0.2

NETWORK 5 //Counter counting the process
LD     Q0.1
LD     I0.0
CTU    C0,+3

NETWORK 6 //Timer
LD     C0
TON    T34, +12000

NETWORK 7//Controlling  the motor for 2 minutes
LD     C0
AN     T34
A      SM0.5
       Q0.4
=      Q0.5

NETWORK 8 //End of program
MEND
```

8.3 Functions of AH Networks

The functions of all the networks .used in the program are explained below:

- NETWORK-1 In network-1 we have created an ON/OFF button circuit that controls the whole system.
- NETWORK-2 In network-2 we have created a timercircuitthat will control the motor and keep it in ON state for four minutes.
- NETWORK-3 In network-3 again we have created a timer circuit that will also control the motor but in opposite way like it will keep it in OFF state for two minutes.
- NETWORK-4 In network-4 we have created the circuit that shows how the motor is being controlled by the two timers and. the counter.
- NETWORK-5 In network-5 we have created the counter circuit that will repeat the processes designedin above networks for three times and after that it will shift to the process designed in next network.
- NETWORK-6 In network-6 we have created another timer that will operate when the counter will stop repeating the above process and it will work for two minutes.
- NETWORK- 7 In network- 7 we have created the circuittha.t will operate the motor for the time set on the timer of network 6 and will operate 'the motor in such a way that the motor will change its states in every 0.5 seconds from ON to OFF and back.
- NETWORK-8 This network-8 declares the end to the program.

CONCLUSION

The project of AC motor controlling using a Siemens Simatic S7-200 programmable logic controller with processor CPU-212was concluded to be successful. I was capable to program the programmable logic<controller to control in the way required without facing problems, so I conclude thatJf.is Jl. good, flexible and easy method ofcontrolling motors.

Electrical motors are the basic pillars of industries aiid.ifisıiecessary to control them according to the require~~nts and conditions in orde~ .'.;~i;:;to~ate.the systems. The systems used for motor controlling are improving day by QJıy. J.Jı~ prQgrammable logic controllers are the latesttechnologies of this field and these are the bestaiid long lasting systems.

Programmable logic controllers are the best for motor controlling and sequential processing of the processes carried out in industries. The biggest advantage of programmable logic controllers is their freedom from the need of frequent human involvements and their remote programming and visual display capabilities. They are preferred in industries because of their modularity, scalability, flexibility, .reliability, environmental resistivity and service and spare parts availability.

APENDIX

sp.ıcıaırMmıory Blts

SMO.O	At,.,.ays on	SM1.0	Resuıt ot opsratlon = o
SM0.1	Fırst scan	SM'1	overrıow orlllegaı valus
SM0.2	Retentııe cıata ıost	SM1.2	Negatııı? resuıt
SMO.~	Powerup	SM1	oıvısıon b~' 0
SM0.4	~s01T/30son	SM1.4	Taoıe ruıı
SM0.5	o.s S 01I' ıo.5 son	SM1.!ı	Table empty
SM0.6	oır 1 sean ron 1 scen	SM1.6	BCo fo 1:ıınaıy conversıon ı?rror
SM0.7	swıtctı ın RUN posnon	SM1.7	.ASCII to he:x conversı:ın srror

E\l~mNuınber	lnterru pt Descripti on		Priority inGrl:)Up
8	Port 0: Re::ei,•e charooıer		
9	ı:GifO: Transıııfcornpleı~		
23	Port 0: Recei,•e message corrpleta	Comrrunicaticfıs (M;tıesl)	0
	Porl 1: Re::eiJe message colll)lele		
25	Port 1: Re::eiJe charooter		
26	Porl 1: Transnit compleıı		
19	P''l'\ R ~~~lete -- in1errueł		0
20	PTO 1 complete interrupt		1
0	Rising ed3e, 10.0		2
2	Rising ed3e, I0.1		I
4	Rising edJe, 10.2		4
	Rising ~e. 10.3		5
	Falliıg ecğ;:. 10.0		
I	Fallhg ecl;F-, 10.1		
5	Falliıg ecl;F-, 10.2		
1	Fallilg ecl;F-, 10.3		
12	HSCO CV=Pı/ (currenı ~-alue = prssel \'alue:,		10
21	HSCO dire,,"1ion dıal)Jed		11
28	HSCı) extemal resel		12
13	HSC 1 CV=P•/ (currenl ,-alue = preset •,'alue)		13
14	HSC 1 direction input ctaıged		14
15	HSC1 exıemal reset		15
16	HSC2 CV:P~'		16
17	HSC2 direı::lion dıal)Jed		17
18	HSC2 exiemal resel		18
32	HSC3 C\<:P'ı/ (currenı ,"Slue = prese! ·,'alue)		19
29	HSC4 CV=Pı/ (eorrent value = prese! \'alue)		20
30	HSC4 dire,,"1ion dıal)Jed		21
31	HSC4 exıemal reset		22
33	HSC5 CV=P•/ (currentv::ılue = prese! \'alue)		23
10	Tımed interrupl 0		0
11	Tımed interrupt 1	TimE,j O.oweıst)	1
21	TımerT32 CT=PT iıterrupl		~
22	Tımer T96 CT=PT iıterrupl		I

Do.cripl:icn	CPU 221	CPU 222	CPU 224	CPU 22:3	Bit	Byte	Word	Double Word
		R"ng• Liınit			Accessible as ...			
			4 Kwords	4 Kwords				
			2.5 Kwords	2.5 Kwords				
Prı:ceı:•,inn;,ı,, iııı:o regi~l'-'1'	ı:ı.o ıo ll &7	IJB te 115.7	I0.0 to I15.7	I0.0 to I15.7	Ixy	IBx	IWx	IDx
Prı:ceı:•,inıı,;ıe oı.ılpııl rcgolor	Cı:l.Oıo01.S.7	100.010015.7	Q0.0 to Q15.7	Q0.0 to Q15.7	Qx.y	QBx	QWx	QDx
Ambg iıpul• (reıt:! ı:ntı',ı	-	AIIIIJ ıe Al'A'3J	AIW0 to AIW32	AIW0 to AIW32			AIWx	
fırı.•l:ı,g tupuıeı (wriı.,. , on~~ı	-	ACfi\lJı, *N::Nı/30*	AQW0 to AQW32	AQW0 to AQW32			AQWx	
~ri,1:fome,ırı::rı• ('vıl	VOO.Oıo Vn-20-17;7	\ıl30.0 '" \1132047.7	'ııllO.Oıa 'ı.•tl6111?.7	\•BJBto \ıB5119.7	Vx.y	VBx	VWx	VDx
Local memory (L)[2]	L9J.Oıo Ll363.7	LOO.C•ıo L9S3.7	LE!0.0 ı:, LOOJ.7	LOO.Oto L00&7	Lx.y	LBx	LWx	LDx
Bit memory (M)	M0.0 to M31.7	MJ.Oıo 1.131.7	M0.0 ıor.131.1	r.t:ı.oıo 1,131.1	Mx.y	MBx	MWx	MDx
Special Memory (SM)	SM0.0 to SM179.7	SM0.0 to SM179.7	suoı:ıı, SU179.7	SMJ.Oto SMml.1	SMx.y	SMBx	SMWx	SMDx
Read only	SM0.0 to SM29.7	SM0.0 to SM29.7	SM0.0 to SM29.7	SMJ.O 1.0 SM29.7				
Timers	256 (T0 to T255)	256 (T0 to T255)	256 (T0 to T255)	256 (T0 to T255)	Tx		Tx	
Ret. on-delay 1 ms	T0, T64	T0, T64	T0, T64	T0, T64				
Ret. on-delay 10 ms	T1 to T4, T65 to T68	T1 to T4, T65 to T68	T1 to T4, T65 to T68	T1 to T4, T65 to T68				
Ret. on-delay 100 ms	T5 to T31, T69 to T95	T5 to T31, T69 to T95	T5 to T31, T69 to T95	T5 to T31, T69 to T95				
Onı::::ffı:i:,lor)' 1 ms	T32, T96	T32, T96	T32, T96	T32, T96				
Onı::::ff ı:1:,br 1 0 ''''	T33 to TJB, rn,ıoTW)	T33 to T36, T97 to T100	T33 to T36, T97 to T100	T33 to T36, T97 to T100				
On Ol ı:l:,by 100 ıns	T.37 ıe 163, T101 ıoT255	T37 to T63, T101 to T255	T37 to T63, T101 to T255	T37 to T63, T101 to T255				
Caurtar.;;	1~	C0 to C255	C0 to C255	C0 to C255	Cx		Cx	
Higlı~d ccu,ı,,r	HO), H:::3. HC4. H:::6	HCi.l,HC3, HC4,llC5	HC0 to HC5	HC0 to HC5				HCx
So,querıial o::mııol ''''''''' l'S'l	&:l.Oıo SS1 .7	s0ı:1 ıoSJ1.7	SO.OıoS31.7	S0.0 to S31.7	Sx.y	SBx	SWx	SDx
Accumıɪblı:ır regi5lre;.;	ACOt.oAC3	ACOıoı,ı:;3	AC0 to AC3	AC0 to AC3		ACx	ACx	ACx
Junp~'Lı,be,I<.	010255	Ob2C5	0 to 255	(!l:ı 213,S				
Cıɪll.'Sul:wutiıɪe	o1oe3	Ob03	0106)	01:,63				
lnı.ıruf-"I rc<I:"°"	010127	Ob 127	Oıı, 121	01:ı 127				
PQb<:ıF"	Oıo'l	0 ı , *1*	Oro7	01:ı 7				
Part	Part 0	Porto	Pı:orıO	Port 0, Port 1				

ı Al. *'V* n~ınct)' ci.n L:..:, *ı,:ıɪı'C'd* ·b perrn,:ııı~ııl:llt!tn-.:t)'·
•L930b:ı LB63al!>r'-"""""fl:t{STEP ,.t,ii:;r"'lı'IN 32,.,,..,mi:rn 3.•) t< L-;,or.

Mocte	HSCO			HSC3	HSC4			HSC5
	10.0	10.1	*W.2*	10.1	IQ.3	10.4	10.5	10.4
0	Clk			Clk	Clk			Clk
	ClK		Reset		Clk		Reset	
2								
3	Clk	oırectıon			Clk	oırec:lh:n		
4	Clk	oırectıon	Reset		Clk	oırec:tıcıı	Resel	
5								
ti	Clk				Clk Up	ClK·DC:Ni'n		
7					cx up	ClK Dd,fo	Reset	
8								
9					PhaseA	Phase B		
10					PhaseA	Phase B	Reset	
11								

Mode	HSC1				fis			
	ro.e	10.7	11.0	11.1	1.2	ii	11.4	
0	Clk				Clk			
1	Clk		Reset		Clk		Reset	
2	ClK		Resel	start	Clk		Resel	Start
3	ClK	tnrscnon			Clk	oırectıcn		
4	ClK	oırectıon	Reset		ClK	oırectıcn	Reset	
5	Clk	oırectıon	Reset	start	ClK	oırectım	Resel	st:ırl
6	ClKUp	ClkDCM'n			Clk Up	ClK DCliı'n		
7	Clk Up	Clk Dcı,...n	Reset		GlK Up	ClK oown	Reset	
8	ClK Up	ClK ocı.vn	Reset	Start	ClK Up	Cll(Dcı,vn	Reset	Start
9	PhaseA	=nass B			PhaseA	Phas,,ı B		
10	PhaseA	Pnase B	Resel		PhaseA	=nasa B	Resel	
11	PhaseA	Phase B	Rıiset	sıart	PhaseA	Phase B	Resel	Start

aooıean ıı structions		
LD	N	Lo,ıd
LDI	N	L,,..~ lmnıedi,1'"
LDN	N	L=d tıoı
LDNI	l4	L,,..~ l'lcı· hınledıne
A	t~	Al-O
Al	N	Al-O lırn.,...iinle
i\N	N	AI-DN,ı
i\NI	N	A/,0 Net hnnıcdi,lc
o	t4	OR
Ol	t4	ORlmmed•I"
ON	N	ORNot
ONI	N	Oll N,ı lrnmodalıo
LOlh	H1,N:i:	Lı,,dresıJt ol Byıe C:ompı.-e N1 (x:•:, •::::;::, ::-::., :-, .c'fl) N2
AB,;	H1,NZ	Al-O rerul ol ll~•leı;;;,,lrl"l'" N1 (x:<, """"·:•:,:o,•0•) N2
OE!.x	ll1.N2	Oll ,.,.uıı d 8~1:e CaıiF"'fe t4l (x:<, ,:::,:,),:, "· '-"): N2
LDWx	N1,112	Low.! 11!".Utcf Ward Cıımpıre .Nl(x:•:,•::,,'=, >:,>,'-") t.ız
AWx	N1, N2	AND result of Word Compare N1 (x<, <=, =, >=, >, <>) N2
OWx	N1, N2	OR result of Word Compare N1 (x<, <=, =, >=, >, <>) N2
LDDx	N1, N2	Load result of DWord Compare <Nl.l:it:.::~-:~:_;~-:{7\~•::~"'JiJ',Le;'
ııı,	tı1,t1:ı	AND result of DWord Compare N1 (x<, <=, =, >=, >, <>) N2
00.:	HUI~'.	OR result of DWord Compare t~1-ı)c<, •:~:,;:-~::;.,:ı,ı,:.c,. } N2
l.ORıı:	N1,H2	Lı:0~ w.,o~l ı:l' il'"º' C<ırr@re N1 (.:.:, ,,,,,'=, :>o,:•, ""~) · 142
i\Rx	111,N:1	A/V r=.ıl al ~I Carpım, N1 (x:.;,•::::;:::,;.:-:::,;ı.,ıc",..J-t'ılı
OR•	H1. H2	OR rc,;ull d Real C:ımpı.-ıo N1 fx:•;, ,•:::',,-::,:ı.::,:a•,,ı::r-J t-!2
r~;:,r		Sl•ckt~ıi:ııı
su		t:ı,ıecl:ı:n al Rr;ıng Eızy.,
ED		Do!e:ti:.ı ol Fırılıng Ed"º
	N	ı,...ıg,\/,luo
=I	N	l~g-ı V\~ue hınıeciat~
R	S.sıT N	SetWRaııge
Si	$_BiT, N	ReY.'I bil R"'il"
RI	S_BIl. N	Set. t:iı R.ııııg<> hıımd~e
		Re<e! bl lla,n;r., ln'ffl!!di:ıle

.•J,lalh;lnc:remen:t;.and:Decoonentiııs1111c1ions		
q	IN1,0UT	Add hbgor. Ü'A'wJ cr R....ı IN1•-0LIT=OUT
llll	lt41,0UT	
•R	IN1,0UT	
,l	IN1,0LIT	Sul:!rn<i lrtıeıgor; O'ı\'orcl,er Reı,I OLIT-IN1=CUT
.o	IN1,0LIT	
.R	IN1,C•LIT	
MJL	IN1,0UT	l,kilipl~ hr.ıgı:r l'.lil'1c.,,.3,') ot R<,,ıl
'R	IN1..0LIT	l,llAıiplf.~~g"f ~ Dııbh lnb,g>r
'0,'I	INi,öüi	IN1 • OUT= cor
Dfil	lt41.0UT	CM:I" lnıı,g.r (16/16-"2ı2'.1ıır R"'11
,R	IN1,0UT	C:.ı•i:!<ı.[nı.~ror.lkııl:folrto,;ı<t
D,,I	IN1,0UT	lflIOUT:iiOUT
SORT	lt4,0UT	Square Root
u~	IN,OLIT	tıııürofL<ijurıı́fıı
EXP	IN,OUT	Hıııur.ıl Expcmııkıl
SIN	IN,OUT	Sinıı
0_"s	IN,OLIT	
TAN	IN,OUT	-1r~ııg,,111
INC8	OLIT	
INC'ı\'	OUI	hıcremcd 13)l<>, Werd cr Cfi\'crıl
INCO	oırr	
DECi!	OUI	
DEC'lı'	OUT	D,.ı;rı,n.,,ııl B'/lı>, 'ı'ı'«d, cır O'lı'orcl
Oıscci	OU.l	
PID	Table, Loop	PID Loop
Timer and Counter Instructions		
TON	Txxx, PT	On-Delay Timer
TOF	Txxx, PT	Off-Delay Timer
TONR	Txxx, PT	Retentive On-Delay Timer
CTU	Cxxx, PV	Count Up
CTD	Cxxx, PV	Count Down
CTUD	Cxxx, PV	Count Up/Down
Real Time Clock Instructions		
TODR	T	Read Time of Day clock
TODW	T	Write Time of Day clock
Program Control Instructions		
END		Conditional End of Program
SliJf~		Transition to STOP Mode
'i',IJR		WatchDog Reset (300 ms)
Jl,tP	ti	Jmnpl<id<;lin.~l.<ıoc~
La	H	Dı,fi,.,," L..ı..l ıo J.ımplo
G.ı\LL	IIINt...J	Qıll • &tırruine IN1, ... up ıo 1G opiı.:ml pı,ımıelor.,]
CRET		C«>:lH:n,ıl Rdi.tıı Fraın sr.ll<
FOR	lt•OXINIT. FNAL	For,Neı<I Loop
NDCT		
i:sCR	H	Ll:.'>:l,lr.,ımlir.o,,ııdEnd s""""""ea,ıroıR<ıia,· Sogn,;ııt
SCRT	N	
SCRE		

lobve., Shift;.,Rotafo, and Fıllnstruotions		
M0\'13	ltl.CUT	
t.10\ı'A'	iN. cıır	l,tı:,..,Bytı,, 'lı'cn:1, O'lı'ord, Rc:ıl
t.ıo\,n	ıtt oır	l,tı:,,eByıe hımed,te R.,.,;l
1,10'\ıR	iN, CUT	l,tı:,;eB>ıte lnmedolc Wrile
GIR	ltl,C:IJT	
mw	iti. CIJT	
BhlEl	IN.CUT,N	
BM'i\'	m. C:IJT. N	Bb:k l,b,.., Bı•k:, 'li\:ı«I, IJ;\'oırl
raı.m	lll. CIJT, N	
S'/'1\P	iN	S•."f>Bıie•
Sl.flll	DATA,S_rıIT. N	Shli Re,;.i•la m
sı;s	CIJT,N	
SR'tı'	OIJT,N	Shli Right El>µ:,'lı'oırl. D'lı\:ı«I
S1.D	CIJT, N	
SLB	CIJT,N	
SL'lıı'	OOT,N	Shfl Lı:llı:i>fte,'ı\'o~, Cfii'i'ı:,rd
SLD	OIJT,N	
ı;ı:ıı	OOT.N	
RR'lı'	CUT,N	Roı.ıe Rigl1. Bıt<ı, Yıbrd, Cl/ı'«ı:1
ı;ı:ıo	CIJT,N	
~-B	CUT,N	
R!N	OOT,N	
~-D	CIJT, N	
FILL	111,0..ILN	Fılırıı:ıö.:t)•.,op,ıioeıı.ılhJ"lll"rıı

L()gıcpperatıons		
Al.O		And fot.eontirdi:ım
OLD		Or fa· coot.ımıioıı•
LPS		Logic Pu•h (•t!ı::t.o:::ı,ıııro~
LRD		Logic Re:ıd (,:t,ıclı conııol)
I.FP		Logic Pı:t>(•lad: ooıırı:IJ
LDS		Load Sıack (ol.:.,c:kcı:nln:ıl)
Aı;t~'.)		Aıı~t;N'.)
,V,OIl	1U1.CUT	
N,UW	ltl1,C!J1	Lo9ic,ıl An:! d 9y1,., Waıd. and 11/,'crd
Al•UD	1111,00T	
ORB	IH1.CUT	
OR'ı\'	IN1,ıXJT	Logioı,IC,. el i}ıie. 'l.brd, arıd Clı\'ad
Cıı.D	1111.CIJT	
XOR8	IN1,00T	
XDR'(ı'	lH1, GUT	l:r.,.,:ı' XCl- el eıt,,, •Ncırd, and
XORD	1111,eur	
ltılv'ı\'	CUT	im"'~ B}1c, 'ı\bırl and DWcrd
lt~\ID	cer	(1 S c.oınplemcııı:ı

Table.Find, and C.:ıııversioıı Instructions		
	lABLE.ll!ı,TA	Ad:! d,ta, kı lable
LIFO	1ABLE,OATA	Gel dalı, hı::nı ı.,ble
FIFC	TABLE;.OATA	
FND•	SRC.PATRN. NDX	
FND<>SRC.PATRN	NOX:	fı.ıd dı,ı.,"""" iıı loble imi m:ııche a:rnpı1risco
	F~CutsRd~TRN, NDX	
FND• SRC.PATRN, NDX		
BCDI	OUT	Convert BCD to Integer
IBCD	OUT	Convert Integer to BCD
BTI	N, CUT	Convert Byte to Integer
ITB	N, OUT	Convert Integer to Byte
Iro	N, C!JT	Convert Integer to Double Integer
m1	fil.CUT	Convert Double Integer to Integer
DTR	fil.OUT	Convert DWord to Real
TRU1C	N,CUT	Convert Real to DWord
R:ıJtıO	N, Ol.iT	Convert Real to Double Integer
ATH	N. C!JT, LEN	Cı::rnaı I>.SCIlto H•.•.
HTA	111,0UT,LEN	Cı:m,,rl ı+..x fo ASı:::l1
iTA	N, OIJT, FMT	Cı:rn,,,-ı lrtegertı A..<ı.CIl
DlA	N,OUT.FM	Cı:ro,:rl Doul:ık lrtega lo A.':JCII
RIA	N,OUT,FM	Cı:rn.,,-1 Re:ıl ı:ı ASCII
CECO	fil. C:IJT	D«:ode
ı:;ı~:::cı	N, ı:X.IT	Enecde
srn	N.OIJT	Geıı~le 1.~gmeııpeuem
		nterrupt
a:lcrl		Cm:iiicn:ıif~hirlı-fr6-,-ri-lrt.
tNI		En:1blelı1>0nuı:ı•
DISI		0.-,<ıbl:,lıılaıı.ıpl•
ATCH	Nl. t;\>ENT	Aı:t,ch lrt>0ıruı:'roulini.,ıo """"
OTCH	ı:;vı;ın	Dot"""1m..,ııl

Communicaıion		
XIH	TABLt;,PCRr	Fr«"P"'~.lrarı,ıı'isoi:ııı
RCV	lABLt,ı:>:::Rr	Frcqx:ırt:recdı.'(! ~~
t,ETR	lı\BLE,PCRI'	Nctwoıl< Re<>:l
t-ETW	1ABLE.FCRI'	Netwı:ırk'lı~iı,,
GPA	ılJ.)00,PORl	.:;,,,ıraıAdaı,,;•
SPA	ADCf.!,PORl	Seı Pat Ad::I=

High-Speed Instructions		
mEF	HSC,l,bde	DdiıieH,~lı-Spoed Coon>0rm,de
HSC	N	Activolc,Hi.;ılı-SpeedCt'Ali,er
PLS	X	Puloewlpııı

REFRENCES

[1] Alan J. Crispin, *Programmable .Logic Controllers and their .Engineering Applications,* McGraw-Hill Inc., New York NY, 1996.

[2] lan G. Wamock, *Programmable Controllers Operatıon and Application,* Prentice-Hall Inc., Englewood Cliffs NJ., 1996.

[3] http://www.ad.siemens.de/s7-200/index_76.htm

[4] http://www.plcs.net/contents.shtml

[5] http://www.plcopen.org

[6] http://www.contech.com.au

[7] http://www.cbiss.com/process/plc _systems/plc _siemens_main.htm

[8] http://www.engineeringtalk.com/news/sie/sie126.html

[9] http://www.themanmachines.com/p40-11806-siemens-200-cpu216-plc-wow.html