NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Electrical and Electronic Engineering

AIR CONDITION CONTROL OF AKM (PLC)

Graduation Project
EE – 400

Student: Murat Muhammet GÜVEN(92063)

Supervisor: Özgür C. ÖZERDEM

Lefkoşa – 2001

CONTENTS

INTRODUCTION

Now that understand how inputs and outputs are processed by the PLC , let's look at a variation of our regular outputs. Regular 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.

Think back to the we did a few chapters ago. What would' ve happened if we couldn't find a "push on / push off" switch? Then we would' ve had to keep pressing to 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.

Picture the remote control for your TV. It has a button for on hand another for off. When I push the on button 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 of a 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 latch) or RST (reset). The diagram below shows how to use them in a program.

ABSTRACT

My project is generally is about PLC information's. But my project can be separate into two part.

In the first part **SIEMENS SIMATIC S7 PLC** information's and sample program is given. At the same time in this part has SIMATIC S7 PLC generally information and instructions and history, is give In the second part MITSUBISHI **FC-40 FC-20 PLC's** general is given.

And the last part is about the project and the program that wrote and implemented a bout automation of air conditioner of AKM.

ACKNOWLEDGEMENT

I am deeply indepted to my parents for their love and financial support. They have always encouraged me to pursue my interests and ambitions throughout life.

To my supervisor Mr. Özgür C. ÖZERDEM who was helped me to finish and realize this difficult task , my deep gratitudes and thanks.

Also thanks to Prof. Dr. Haldun GÜRMEN , Prof. Dr. Fakhreddin MAMEDOV , Prof. Dr. Şenol BEKTAŞ Prof. Dr. Khalil ISMAILOV assist Prof. Kadri BÜRÜNCÜK and assist Prof. Kamil DİMİLLER

Mr. Kaan UYAR and to all of my other teachers for their advices.

1.WHAT IS A PLC ?

A programmable logic controller (PLC) 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.

PLC's are used in many real word applications. If there is industry present , chances are good that there is a PLC present. If you are involved in machining , packaging , material handling , automated assembly or countless other industries you are probably already using them. If you are not , you are wasting money and time. Almost any application that needs some type of electrical control has a need for a PLC.

For example , let's assume that when a switch turns on we want 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 times the switches individually turned on? We need a lot of external counters.

As you can see the bigger the process the more of a 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.

This site gives you enough information to be able to write programs far more complicated than the simple one above. We will take a look at what is considered to be the ' top 20' PLC instructions. It can be safely estimated that with a firm understanding of these instructions one can solve more than 80 % of the applications inexistence.

2.PLC HISTORY

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

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

These new controllers also had to be easily programmed by maintenance and plant engineers. The lifetime had to be long and programming changes easily performed. They also had to survive the harsh industrial environment. 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 PLC' s. Conventional microprocessors lacked the power to quickly solve PLC logic in all but the smallest PLC' s. As conventional microprocessor evolved , larger and larger PLC' s were being based upon them. However , even today some are still based upon the 2903. 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 PLC' s and they could be far away from the actual machine they were controlling. They could also now be used to send and receive varying voltages to allow them to enter the analog world. Unfortunately , the lack of standardization coupled with continually changing technology has made PLC communications a nightmare of incompatible protocols and physical networks.

The 80's saw an attempt to standardize communications with General Motor's manufacturing automation protocol . 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.

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

3.GENERAL PHYSICAL BUILD MECHANISM

PLC' s are separated into two according to their building mechanisms.

3.1Compact PLC' s

Compact PLC's are manufactured such that all units forming the PLC are placed in a case. They are low price PLC with lower capacity. They are usually preferred by small or medium size machine manufacturers. In some types compact enlargement module is present.

3.2Modular PLC's

They are formed by combining separate modules together in a board. They can have different memory capacity , I / O numbers , power supply up to the necessary limits.

Some examples: **SIEMENS S5-115U , SIEMENS S7-200 MITSUBISHI PC40 , TEXAS INSTRUMENTS PLC'S , KLOCKNER – MOELLER PS316 OMRON C200H.**

3.3. BASIC INSTRUCTION

Each program written in PLC are started in 2 ways. One at these that we can draw the program with its symbols in the location called Ladder Diagram and load it to the computer as this. The second one is that we can make direct attribution using the key team of PLC. Because of this it will be told example symbol and attribution us. Instructions later whole LOD instruction and the other instructions are being stated.

a) LOD Instructions:

This instructions is used at the beginning of logic diagram lines. It can be used once back by back or more than once to determine the situation at the beginning of the instructions such as AND LOD, OR LOD, SFR, CNT, TIM. As you see below an input relay is wanted to be loaded as a program. Symbol of it is declared as a show in ladder diagram. Program list from the statement.

This program is loaded as 0 LOD 1 and 0 which is seen an address must be given in each line of the end one by one starting from each line of the program. Value is appointed to each line orderly. We have mentioned before which numbers are separated for shift register, output, input, special relay, timer counter. Imaginary internal relay at the machine PLC.

We can divide our load process into 4 groups according to our functions.

b) Input, Output Internal and Special Relays:

In the examples above example relay circuit of relay in ladder diagram and how the process of key and as a result of this the format seen in deplay was given.

- We can choose a value between 0 and 77 except 8 and 9 in the example of input.

  We can choose a value between 200 and 277 cxccpt 8 and 9 in the exampic of output.

  We can choose a value between 400 and 697 except 8.and 9 in the example of internal relay

You can use special relay-which you necd are between 700-717 in the example of special relay for example I use pulse generator of clock for one speed with special relay 714.

c) Timer:

I wanted to use T8 timer from the 80 timer between 0-79 including 8 and 9 here aud you see how the load process had been done.

d) Counter:

You can use any counter between 0 and 46 including 8 and 9. Load process is the saute as aside.

e) Shift Register:

You can use any register from 128 of them between 0 and 127 including 8 and 9. Shift register numbered l was loaded in the next side.

f) AND Instruction:

It is same as AND logic we studied in Logic lessons. Both keys that are connected each other rapidly are on. output is on and is the other situations it becomes OFF in logic. In. a multiplying processes both inputs are 1 than output is 1. And had it ended with 2 limit switches and 1 solenoid valve in~order to understand the logic better by diagranis; it is stated as relay ladder diagnain and logic diagratn. So we cati telt iliat LS 1 relay A and; LS 2 relay is B input and output is .Y. In suck equality it is that Y=A.B according to the compulsion Of Boolean. If both inputs are 1 (ON) Y output will be ON. In other 3 probabilities, output Y will be 0 (OFF) You can seethis in the table of truth.

As known, the series of TTL is Logic entegrate containing 4 and gate with 2 inputs in 7408. As in the circuit 1/4 has been made equal to ladder diagram by using 7408. In both of them The function of output and working are same.

g) OR Instruction:

Or instruction has the same functions as or gate logic we studied in logic lessons. In here, just otily one of flie keys are OFF or 1 is enougli for output to be 1 as 2 keys are connected in the parallel way As a result there is addition process and in this process one of the 2 parallel inputs is enough to be one. 1 gave 2 important information's with or instruction. Onc as thent is Out function that is symbolmed with 200 in the circle. I will speak about out flinction 2 or 3 classes later. But now, I gave output of parallel circuit, output 200 for file first time it means that: 1 mentioned that speciat relay 704 is a clock ptiise generator lint has f=1 HZ You see signal of clock pulse in the diagram. We determined time of I and 0 in. input relay of 36 by chance now so that nothing will be by chance in the following lessoni Let's accept that there is a time diagram for to learn Or let's ass,ame that input 3645 gained by niaking ON/OFF in the f~n. If we think that output 200 is connected to a lamb, the situatiuns that lamb will be on are the times that output 200 is 1.

In this example. in order to understand or instructions better firstly, 2 limit switches were connected to each other rapidly and shown a lader diagram and a solenoid valve control in output of it. And same clrcuit has heen gained Logic equality by using only lor gate of integrate of 7432. It is enou)41 to make on only one of the inputs for the outputs. to be ON in. 3 equaliavence circuit to make output OFF It is necessarv to make both parallel inputs OFF. This position was shown in the truths table below.

h) NOT Instructions:

It has the same duty as NOT gate that you studied in the logic lessons We take the opposite of the sign. If we have a look of the example above, they take the opposite of input relay 1 in PLC.. If you Carry out 1 logic level to input I from the outside, the sign is going to continue from B point as logic 0, because of the instruction of LOD NOT 1.

4.ADVANTAGES

4.1 Accuracy

In relay control systems logical knowledge's carries in electro mechanical contactors, they can lose data because of mechanical errors. But PLC's are microprocessor based system so logical data are carried inside the processor , so that PLC's are more accurate than relay type of controllers.

4.2 Flexibility

When there is need of any change in control , relay type of controllers modification are hard , in PLC this change can be made with PLC programmer equipment.

4.3 Communication

PLC 's are computer based systems. So that they can transfers their data to another PC or they can take external inputs from another PC , with this specification we can control the system were they are we can effect the system with our PC. With relays ties is not possible.

4.4 Logic Control of Industrial Automation

Everyday examples of these systems are machines like dishwashers , clothes washers and dryers , and elevators. In these systems , the outputs tend to be 220 V AC power signals to motors , solenoids , and indicator lights , and the inputs are DC or AC signals from user interface switches , motion limit switches , binary liquid level sensors , etc. Another major function in these types of controllers is timing.

4.5 Data Areas

Data memory contains variable memory , and register , and output image register , internal memory bits , and special memory bits. This memory is accessed by a byte bit convention. For example to access bit 3 of Variable Memory byte 25 you would use the address V25.3.

The following table shows the identifiers and ranges for each of the data area memory types:

Area Identifier	Data area	CPU 212	CPU 214
I	Input	I0.0 to I7.7	I0.0 to I7.7
Q	Output	Q0.0 to Q7.7	Q0.0 to Q7.7
M	Internal memory	M0.0 to M15.7	M0.0 to M31.7
SM	Special memory	SM0.0 to SM 45.7	SM0.0 to SM85.7

V	Variable memory	V0.0 to V1023.7	V0.0 toV4095.7

4.6 Data Object

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

The following table shows the identifiers and ranges for each of the data object memory types:

Object Identifier	**Object**	**CPU 212**	**CPU 214**
T	Timers	T0 to T63	T0 to T127
C	Counters	C0 to C63	C0 to C127
AI	Analog Input	AIW0 to AIW30	AIW0 to AIW30
AQ	Analog Output	AQW0 to AQW30	AQW0 to AQW30
AC	Accumulator	AC0 to AC3	AC0 to AC3
HC	High-Speed Counter Current	HC0	HC0 to HC2

5.LADDER AND STL PROGRAM

SIEMENS SIMATIC S7- 200 PLC SAMPLE PROGRAM

In this program;

Cotton to filament convert during the war. While cottons are to comb by the machine, separate operation during by the war cotton pieces are to gather of under the machine.

This program purposes are cotton pieces convert to back. With this program cotton cost decrease to less.

Program Work

1- For the vakum if we gives the start , motor is start to work.

2- After the 15 s machines are made with raw and once vakum.

3- One machine and other machine between passed time 2 s , and vakum time is for the all machine 8 s.

4- If someone machine not work passed to other machine.

5- If fire alarm or tight alarm gives , fan motor and all other operations are stop. For the machines work are until push the button machines are not work.

Input	***Output***
1- Start	**1- Fan motor start**
2- Stop	**2- Machine 1 vakum**
3- Reset	**3- Machine 2 vakum**
4- Machine work 1	**4- Machine 3 vakum**
5- Machine work 2	**5- Fire alarm**

6- Machine work 3 **6-Tight alarm**

7- Tight

Symbol name	Address	Note
Start	E0.0	Start button
Stop	E0.1	Stop button
Reset	E0.2	Reset button
1. Machine Work	E0.3	If 1. Machine Work send to sign
2. Machine Work	E0.4	If 2. Machine Work send to sign
3. Machine Work	E0.5	If 3. Machine Work send to sign
Fire	E0.6	If the fire alarm is coming
Tight	E0.7	Fan motor has tight
Start – Output	A0.0	Fan motor is start
Vakum 1	A0.1	1. Machine Vakum Valve
Vakum 2	A0.2	2. Machine Vakum Valve
Vakum 3	A0.3	3. Machine Vakum Valve
Fire Alarm	A0.4	If the fire sensor signal coming
Tight Alarm	A0.5	To throw the motor tight
Cleanliness Air valve	A0.6	Tube Cleanliness Valve
Relay of Vakum	T32	After Start Vakum Relay
Vakum Time	T33	Valves are Vakum Time
Stop Time	T34	Between of Valves Stop Time
Counter 0	Z0	1. Machine Valve Work Counter
Counter 1	Z1	2. Machine Valve Work Counter
Counter 2	Z2	3. Machine Valve Work Counter
Counter 3	Z3	Circle Reset
Jump Output	A7.1	Machines are not jump

6. MEMORY DESIGN

Memory is used to store data. This stored information is related with which output sign will be store as, which shows input, and the structure of program necessary amount of memory. It stores special information parts, which is named as memory bit. 1 byte = 8 bit, lO24 byte = lkbyte and the number of memory capacity is stated these units.

a) I. Group Memories:

First group memories are Random Access Memory (RAM) and Read/Write (RIW). In these types memories if the energy is cut, the information is lost. If RAM is supplied program can be stored by battery that battery is in PLC device. When battery energy finishes, program will be erased.

b) II Group Memories:

It is Read Only Memory (ROM). The type memory can be erased and programmable. It is divided four into groups;

6.1) PROM (Programmable Read-Only Memory): it is a special type of ROM.

PROM memory allows to writing of information in chip, these information are provided or there were at the beginning. The information can be written into ROM only one time.

The main disadvantage of PROM is no erasable and no Programmable. In PROM programming is doing as dissolve and pluck logic, for this reason, the erasing of erasable connections is process that there is no to turn back. For this reason, firstly all mistake control process must be finished.

6.2) EPROM (Erasable Programmable Read-Only Memory): this type is the memory ***type*** that is used in PLC devices. Written programmable firsfly, is store in EPROM memory and is sent central processing unit.

6.3) EAROM (Electrically Afterable Read-Only Memory)*:* It is like EPROM memory, but to erase and ultraviolet light supply is not necessary. EAROM chip to clean by erasing, an eraser voltage is exercised to suitable pin. When chip erases one time, it can be programmed again.

6.4) EEPROM (Electrically Erasable Programmable Read-Only Memory): In EEPROM memory type, when energy is cut, information cannot lose as EPROM. Special device is not necessary in writing and erasing processing. EEPROM or EPROM memories that are mounted to PLC make runs as stored program into records.

Data table stores information's, that are necessary to carry to the program, which includes information's such as output and input conditions, timers, and counter results and data records. Includes of table is divided two groups as conditions data and numbers (or codes) 0 and 1 conditions are ON/OFF conditions of information that records the place of bit. Data table is divided 3 sections. Input view table stores the condition of digital input that relations input interface circuits. As ON/OFF condition, in this unit results of input are stored as zero (0) or one (1).

Output view memory is order of bits that control the digital condition of devices which links interface of output. The logic conditions of output units are stored in this memory and it is taken from this logic level memory and transfers to output unit.

6.5. PROGRAMMING DEVICES

The most important one of features of programmable controller is to have programming elements, which are useflil. Programming device provides transformation between operator and circuit of controller (Fig. 7.3.1)

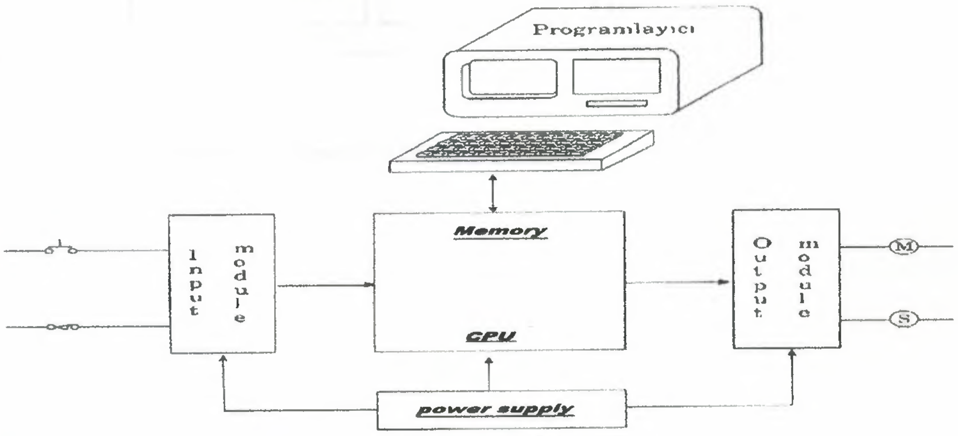


Figure.6.3.1. Transformation of pls circuits

Programming terminal relation between PLC memory and monitor. User sends programming device and PLC control program to device.

Generally, industrial CRT terminals in many devices are used for programmable controllers. These terminals include indicator units, keyboards and CPU and they provide to cornmumcate necessary order.

The advantage of CRT is to check program is easily on monitor.

In small PLCs programming is used cheap, moveable, small and mini programmable devices. The momtor of this type of programming monitor is liquid crystal screen instead of CRT tube, which name LCD. On mini program there are LCD monitor program coding keys and special flinctions keys. FA2 of programming device DEC FAl Junior module is shown at table 6.3.2.

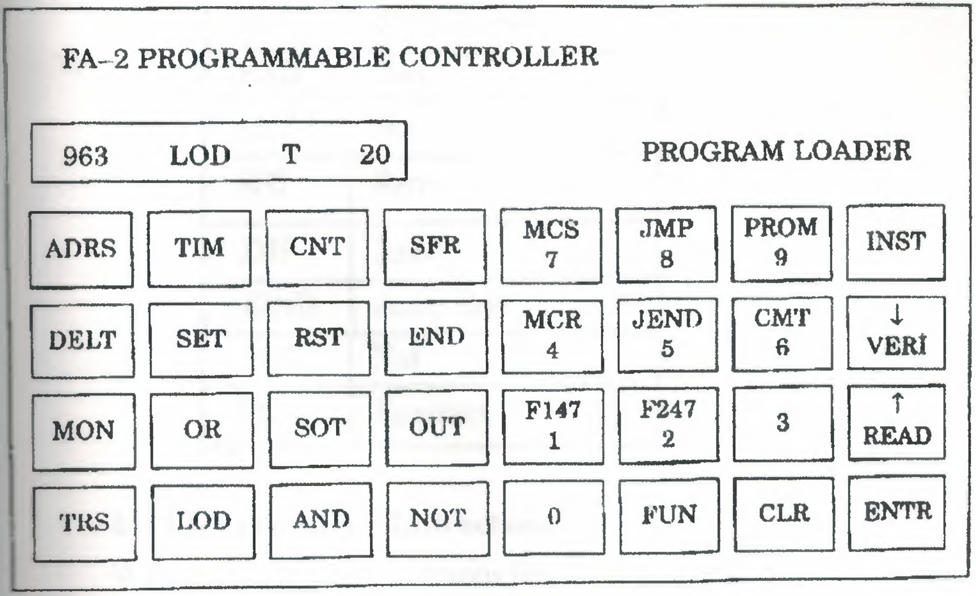


Figure. 6.3.2. Programming Device of DEC FA-1 PLC.

6.6 BASIC INSTRUCTION WORD

Instruction word list

a) Basic Instructions:

Symbol	Name
LOD	Load
AND	AND
OR	OR
OUT	Output
MCS	Master Control Set
MCR	Master Control Reset
SOT	Single Output
TIM	Timer
CNT	Counter
SFR	Shift Register
END	End
SET	Set
RST	Reset
JMP	Jump
JEND	Jump End
NOT	Not
FUN	Function

b) FUN (Function) Instructions:

We can divide the instructions into 2 parts. These are;

One - address instruction

Two - address instruction

There are 2 kinds of address instruction. Generally first address is the instruction word. In LOD, AND, OR, OUT, SET, RST, SOT instructions; there is a instruction word and number and addressing is obstructed with this that single addressed instruction.

Two addressed instructions; SPIt, SPIt NOT, TIM, CNT, FUN 100-146, FUN 200-246, TIM FUN, CNT FUN, FUN 147 and FUN 300. In this instructions first addresses are give instruction word and instruction numbers ~xcept FUN 147, FUN 300). As for second addresses are present peculiarity according to instruction.

There are some deliver numbers that referenced by FAIJ at the below.

c) Input:

0.......7, 10.......7, 20......27, 30.......37, 40.......47, *50*.......57,60........67, 70......77

are numbered like this. In here inputs are considered to OCTAL system which is between 0-77. If you attend 8,9,18,19,28,,29 78,79, numbers are not used. In octal there are 64 unit input number between 0-77 (except 8 and 9).

d) Output:

200......207, 210...... 217,220......227,230...... 237,240...... 247, 250.......257, 260.....267 and 270 277 numbered. Like input there are 64 unit output numbers between 200-277 (except 8 and 9).

e) Internal Relay:

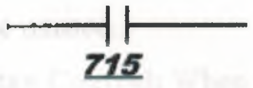
400-407	490-497	580-587
410-417	500-507	590-597
420-427	510-517	600-617
430-437	520-527	610-627
440-447	530-537	620-627
450-457	540-547	630-637
460-467	550-557	640-647
470-447	560-557	650-657
480-487	570-577	660-667
670-677	680-687	690-697

There are 240 units (30x8=240) internal relays between 400 and 697, we can appoint the TISNIER, COUNTER or FUN outputs to the any of 240 sensor and then can use of this sensor for take new data or count value.

f) Special Internal Relay:

There are 16 units become 700-707 and 710-717. As an example of these, we can use the signal generator which produces 1 sec clock sign, that means we can use 1 Hz clock pulse sing ready.

We can use the signal generator which produces 0.1 sec clock sign that means 10 Hz clock pulse sign ready.



g) Timer:

There are totally 80 unit timers between 0 and 79. If you attent you can use 8 ou can use any of TIMER that include 0 and 79. In there its enough to know for totally there are 80 unit TIMER that include 0-79.

h) Counter:

Totally there are 45 unit counter between 0 and 44. If you attent you can use 8 and 9.

i)Reversible Counter:

It is counter which can be counted forward or review. While other counters can only count forward counters number 4546 can count forward or review. Counter 45 has up and down pulse input edge yet counter 46 is connected to only one input of up/down situation and when this edge is 1 up and when it be comes 0 it counts down.

j) Shift Register:

There are 128 shift register between 0 and 27 including 8-9.

k) Single Output:

We can use 96 SOT flinctions between 0 and 95 including 8-9.

1) Data Register:
Between DRO and DR99 and between 800 and 899, we have 100 data register.

6.7. FA1J SERIES ALLOCATION NUMBERS OF SPECIAL RELAYS

As known special relays are 700 and 717 relays except 708 and 704 from these numbers 700 and *705* are unused.

701 and 702 Stan Control: When input number 0, which used to start the program is on or if number *500* has been appointed to automatic start process. It starts to turn the program on. Special relays 701 and 702 are off the process of the program is stopped.

703 All Output OFF: All outputs between 200 and 277 are off when special relay 703 turns into ON.

704 Initialize Pulse: Special flag (1 scan time) 704 becomes on as much as the time equalling I scan time. When program FA1J started being processed.

704 Numerical Value Error: Is there an error in computing instructions results. 706 becomes on for example; if the result of a subtraction process is lower than -10.000, special relay 706 becomes on. They make sure that the program is correct from the point of view numerical process while they register the programs.

707 Curry and Borrow: It there is carry or borrow in the results at computing instructions. 707 is set for example; in a addition process the total of 2 numbers are higher than 9999,707 is on.

713 1 sec. Timer Reset: When 713 is on special relay 714 is always reset mode.

714 1 sec. Clock: It is possible to take signal generator producing clock sign for one second or clock pulse sign for 1 Hz from special relay 714.

715 100-msec. Clock: We can remove our clock pulse that is for 10 speed by using special relay output of 715 with this sign.

716 Timer/Counter Preset Value Changed: Special relay 716 becomes on when timer counter preset value has been changed into unit of FAlJ CPU. It is possible to delete 716 when pressed key of TR S, ENTR and ENTR. If a program is registered in memory.

717 In-operation Output: Relay 717 is always on while FA1J is operating of the program has ended this relay becomes off

7- TYPES OF PLC

The increasing demand from industry for programmable controllers that can be applied to different forms and sizes of control tasks has resulted in most manufacturers producing a range of PLCs with various levels of performance and facilities.

Typical rough definitions of PLC size are given in terms of program memory size and the maximum number of input/ output points the system can support. Table 8.1 gives an example of these categories.

Table 7.1 Categories of PLC

PC size	Max I / O points	Use memory size
Small	40 /40	1 K
Medium	128 /128	4 K
Large	> 128 / >128	>4 K

However , to evaluate properly any programmable controller we must consider many additional features such as its processor , cycle time language facilities , functions , expansion capabilities.

A brief outline of the characteristics of small , medium of large programmable controller is given below , together with typical applications.

7.1 Small PLC s

In general , small and 'mini' PLC s (figure8.2)are designed as robust , compact units which can be mounted on or beside the equipment to be controlled. They are mainly used the replaced hard – wired logic relays , timers , counters. That control individual items of plant or machinery , but can also be used to coordinate several machines working in conjunction with each other.

Small programmable controllers can normally have their total I / O expanded by adding one or two I / O modules , but if any further developments are required this will often mean replacement of the complete unit. This end of the market is very much concerned with non – specialist end – users , therefore ease of programming and a ' familiar' circuit format are desirable. Competition between manufacturers is extremely fierce in this field , as they vie to obtain a maximum share in this partially developed sector of the market.

A single processor is normally used , and programming facilities are kept at a fairly basic level , including conventional sequencing controls and simple standard functions: e.g. timers and counters. Programming of small PLC s is by way of logic instruction list(mnemonics) or relay ladder diagrams.

Program storage is by EPROM or battery – backed RAM. There is now a trend towards EEPROM memory with on – board programming facilities on several controllers.

Table 7.2 Features of a typical small PLC – Mitsubishi F20

Electrical :	240 V a.c. supply;
	24 V d.c. on – board for input requirements;
	12 input , 8 output points;
	LED indicators on all I / O points;
	All I / Opto - isolated
	Choice of output: Relay (240 V 2 A rated)
	Triac (240 V 1 A rated
	Transistor (24 V d.c. 1 A)
	320 – step memory (CMOS battery – backed RAM)
Programming:	Ladder logic or instruction set using hand – held or graphic LCD programmer , with editor , test and monitor facilities;
Facilities :	8 counters , range 1 – 99 (can be cascaded)
	8 timers , range 0.1 – 99 s (can be cascaded)
	64 markers / auxiliary relays ; can be used individually or in blocks of 8 , forming shift registers;
	Special function relays;
	Jump capability.

7. 2 Medium – sized PLC s

In this range modular construction predominates with plug – in modules based around the Euro card 19 inch rack format or another rack mounting system. This construction allows the simple upgrading or expansion of the system by fitting additional I / O cards in to the cards into the rack , since most rack systems have space for several extra function cards. Boards are usually ‘ rugged zed ’ to allow reliable operation over a range of environments.

In general this type of PLC is applied to logic control tasks that cannot be met by small controllers due to insufficient I / O provision , or because the control task is likely to be extended in the future. This might require the replacement of a small PLC , where as a modular system can be expanded to a much greater extent, allowing for growth. A medium - sized PLC may therefore be financially more attractive in the long term.

Communications facilities are likely to provided , enabling the PLC to be including in a ‘ distributed control ’ system.

Combinations of a single and multi – bit processor are likely within the CPU. For programming , standard instructions or ladder and logic diagrams are available. Programming is normally carried out via a small `keypad or a VDU terminal.(If different sizes of PLC are purchased from a single manufacturer , it is likely that programs and programming panels will be compatible between the machines.

7.3 Large PLC’S

Where control of very large numbers of input and output points is necessary or complex control functions are required , a large programmable controller is the obvious choice. Large PLC s are designed for use in large plants or on large machines requiring continuous control. They are also employed as supervisory controllers to monitor and control several other PLC s or intelligent machines. e. g. CNC tools

Modular construction in Euro card format is standard , with a wide range of function cards available including analog input / output modules. There is a move towards 16 – bit processor, and also multi – processor usage in order to efficiently handle a large range of differing control tasks . For example;

- 16 – bit processor as main processor for digital arithmetic and text handling.
- Single – bit processor as co – or parallel processor for fast counting , storage etc.
- Peripheral processor for handling additional tasks which are time – dependent or

  time – critical , such as:

  Closed - loop (PID) control

  Position controls

  Floating – point numerical calculations

Diagnostics and monitoring

Communications for decentralized

Process mimics

Remote input / output racks.

This multi – processor solution optimizes the performance of the overall system as regards versatility and processing speed , allowing the PLC to handle very large programs of 100 K instructions or more. Memory cards can now provide several megabytes of CMOS RAM or EPROM storage.

7.4 Remote input / output

When large numbers of input / output points are located a considerable distance away from the programmable controller , it is uneconomic to run connecting cables to every point. A solution to this problem is to site a remote I / O unit near to the desired I / O points. This acts as a concentrator to monitor all inputs and transmit their status over a single serial communications link to the programmable controller. Once output signals have been produced by the PLC they are fed back along the communications cable to the remote I / O unit , which converts the serial data into the individual output signals to drive the process.

7.5 Programming large PLC s

Virtually any function can be programmed , using the familiar ladder symbols via a graphics terminal or personal computer. Parameters are passed to relevant modules either by incorporating constants in to the ladder , or via on – screen menus for that module.

There may in addition be computer – oriented languages which allow programming of function modules and subroutines.

There is progress towards standardization of programming languages , with programs becoming easier to over – view through improvement of text handling , hand improved documentation facilities. This is assisted by the application of personal computers as work stations.

7.6 Developments

Present trends include the integration of process data from a PLC into management data bases, etc. This allows immediate presentation of information to those involved in scheduling,

production and planning .

The need to pass process information between PC s , PLC sand other devices within an automated plant has resulted in the provision of a communications capability on all but the smallest controller. The development of local area networks (LAN) and in particular the recent MAP specification by General Motors (manufacturing automation protocol) provides the communication link to integrate all levels of control systems.

8 – PROGRAMMING OF PLC SYSTEMS

In the previous chapter we were introduced to logic instruction sets for programming PLC systems. The complete sets of basic logic instruction for two common programmable controllers are given below. Note the inclusion in these lists of additional instructions ORB and ANB to allow programming of more complex , multi branch circuits. The use of all these instructions and others is dealt with in this chapter. Some typical instruction sets for Texas instruments and Mitsubishi PLC s are given in table 8.1

Table8.1 Typical logic instruction sets.

Texas Instruments		Mitsubishi A series	
Mnemonic	**Action**	**Mnemonic**	**Action**
STR	Store	LD	Start rung with an open contact
OUT	Output	OUT	Output
AND	Series components	AND	Series elements
OR	Parallel components	OR	Parallel elements
NOT	Inverse action	..I	As for not
		ORB	Or together parallel branches
		ANB	And together series circuit blocks

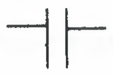
8.1 Logic instruction sets and graphic programming

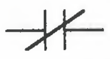
In the last chapter we introduced logic instructions as the basic programming language for programmable controllers. Although logic instructions are relatively easy to learn and use , it can be extremely time – consuming to check and relate a large coded program to the actual circuit function.

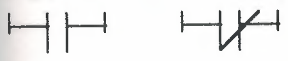
In addition , logic instructions tend to vary between different types of PLC.

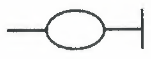
If a factory or plant is equipped with a range of different controllers (a common situation) , confusion can result over differences in the instruction sets.

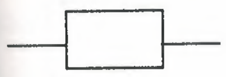
RELAY LOGIC SYMBOLS: (MITSUBISHI PLC)


Input, normally open contacts


Input , normally closed contacts


Inputs in parallel connection


Output device


Special instruction circuit block

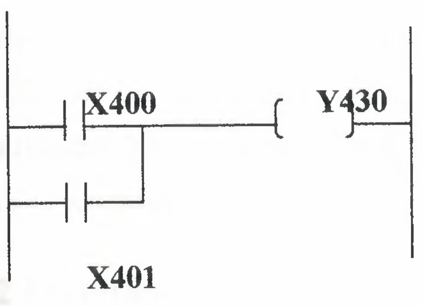
A preferable alternative is to use a graphic programmer , as available for several programmable controllers including the small Mitsubishi and Toshiba models from Japan. Graphic programming allows the user to enter his program as a symbolic ladder circuit layout, using standard logic symbols to represent input contacts , output coils, etc. as shown in the about figure. This approach is more user friendly than programming with mnemonic logic instructions, and can be considered as a higher – level form of language.

The programming panel translates or compiles these graphic symbols in to machine instructions that are stored in the PLC memory , relieving the user of this task.

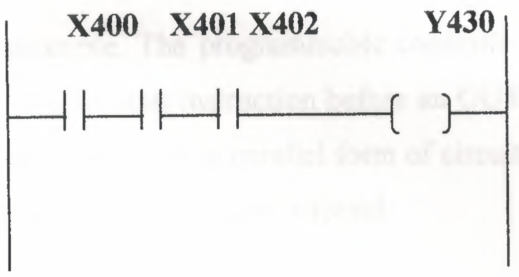
Different types of graphic programmer are normally used for each family of programmable controller , but they all support similar graphic circuit conventions. Smaller , hand – held panels are common for the small to medium – sized PLCs although the same programming panel is often used as a 'field programmer' for these and larger PLCs in the same family. However , the majority of graphic programming for larger systems is carried out on terminal – sized units. Some of these units are also semi portable, and may be operated alongside the PLC system under commissioning or test in – plant. In addition to screen displays , virtually all graphic programming stations can drive printers for hard copy of programs and \ or status information , plus program storage via battery – backed RAM or tape \ floppy disk. The facility to load resident programs into EPROM IC s may be available on more expensive units.

8.1-1 Input /output numbering

It was previously stated that different PLC manufacturers use different numbering systems for input/output points and other functions within the controller.



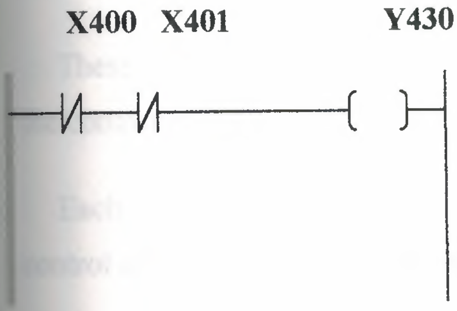
OR gate



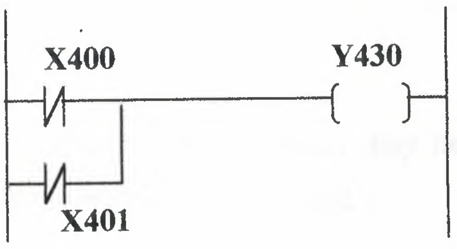
AND gate

8.1-2 Negation – NAND and NOR gates

These logic functions can be produced in ladder form simply by replacing all contacts with their inverse , AND becomes ANI ; OR becomes ORI; etc. this changes the function of the circuit.



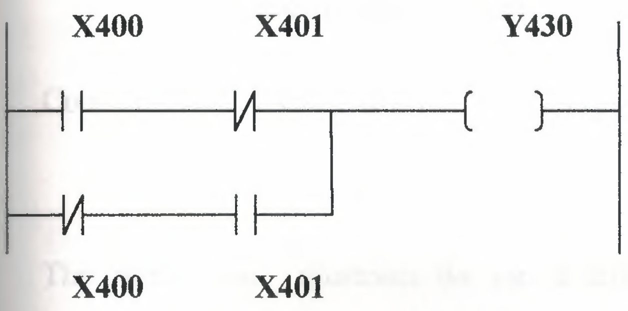
NOR gate



NAND gate

8.1-3 Exclusive – OR gate

This is different form the normal OR gates as it gives an output of 1 when either one input or the other is on , but not both. This is comparable to two parallel circuits , each with one make and one break contact in series as shown in exclusive OR gate figure.



EXCLUSIVE – OR gate

Note the use of an ORB instruction in this example. The programmable controller reads the first two instructions, then finds another rung start instruction before an OUT instruction has been executed. The CPU therefore realizes that a parallel form of circuit exists and reads the subsequent instructions until an ORB instruction is found.

8.2 Facilities

8.2-1 Standard PLC functions

In addition to the series and parallel connection of input and output contacts , the majority of control tasks involve the use of time delays , event counting , storage of process status data , etc. All of these requirements can be met using standard features found on most programmable controllers. These include timers ,counters , markers and shift registers, easily controlled using ladder diagrams or logic instructions.

These internal functions are not physical input or output. They are simulated within the controller.

Each function can be programmed with related contacts which may be used to control different elements in the program . As with physical inputs and outputs , certain number ranges are allocated to each block of functions. The number range will depend both on the size of a PLC , and the manufacturer. For example , for the Mitsubishi F- 40 series , the details are as follows:

Timers T	450 – 457
	550 – 557
Counters C	460 – 467
	560 – 567

The information illustrates the use of different number ranges assigned to each supported function. For example , the timer circuits for this programmable controller are addressed from 450 to 457 and 550 to 557 , a total of 16 timers. It is the specified number that identifies a function and its point to the PLC , not the prefix letter. This prefixes are included only to aid the operator.

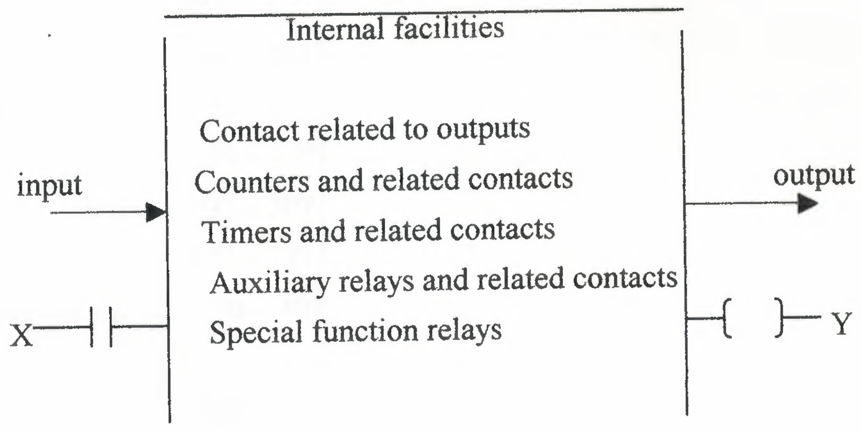

Internal facilities
input
Contact related to outputs
Counters and related contacts
Timers and related contacts
Auxiliary relays and related contacts
Special function relays
output
X
Y

Figure 8.1 Standard PLC function

The functions listed are provided on most programmable controllers , although the exact format will vary between manufacturers. Other functions may also exist , either as standard or by the selection and fitting of function modules to the PLC rack.

PC (F40) I/O ASSIGNMENTS	
Inputs: 24 points	400 – 407
	410 – 413
	500 – 507
	510 – 513
Outputs: 16 points	430 – 437
	530 – 537
Timers: 16 points	450 – 457
	550 – 557
Counters: 16 points	460 – 467
	560 – 567
Auxiliary control	100 – 107
Relays: 128 points	170 – 177
	200 – 207
	270 – 277
Battery – backed: 64 points }	300 – 307
Special function	370 – 377
Auxiliary relays ; 5 points	
70, 71, 72, 75 ,77	

Figure 8.2 Typical number assignments to internal functions

The operation and use of the listed standard functions is covered in the following sections.

8.2-2 Markers / auxiliary relays

Often termed control relays or flags , these provide general memory for the

programmer , plus associated contacts. They also form the basis for shift – register construction. Normally a group of markers with battery back-up is provided allowing process status information to be retained in the event of a power failure. These markers can be used to ensure safe startup \ shut down of process plant by including them as necessary in the logic sequence.

Referring , the Mitsubishi F40 has:

128 auxiliary (marker)relays

64 battery – backed markers

8.2-3 Ghost contacts

In certain cases it will be necessary to derive an output from the combined logic of several ladder rungs , due to the number of contacts involved. The straight forward way of providing this is to common – up the respective circuit rungs and drive an internal relay or marker(M). This acts in the same manner as a 'physical' relay , in that it can have associated contacts – except for the fact that it is simulated by software within the programmable controller , and has no external appearance whatsoever!

In common with other internal functions , auxiliary relays / marker can be programmed with as many associated contacts as desired. These contacts may be used anywhere in a ladder program as elements in a logic circuit or as control contacts driving output relays or other functions.

8.2-4 Retentive battery – backed relays

If power is cut of or interrupted whilst the programmable controller is operating , the output relays and all standard marker relays will be turned off. Thus when power is restored , all contacts associated with output relays and markers will be of possibly resulting in incorrect sequencing. When control tasks have to restart automatically after a power failure , the use of battery – backed markers is required. In the above

PLC ,there are 64 retentive marker points, which can be programmed as for ordinary markers , only storing pre – power failure information that is available once the system is restarted.

In figure 8.3 retentive marker M300 is used to retain data in the event of a power failure. Once input X400 is closed to operate the M300 marker , M300 latches via it is associated contact.

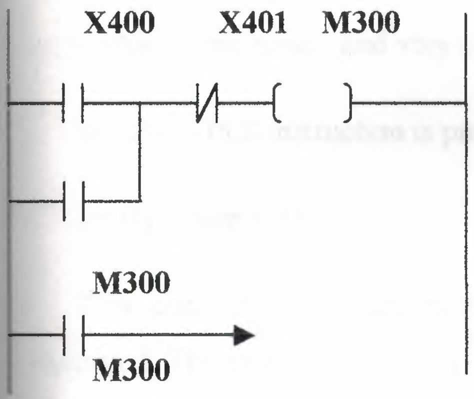


Figure 8.3 Retentive marker used in a latch circuit

So even if X400 is opened due to a power failure , the circuit is holds on restart due to M300 retaining the operated status and placing its associated contacts in the operated positions.

Obviously X401 still controls the circuit , and if this input is likely to be energized (opened) by a power – failure situation , than a further stage of protection may be used.

8.2-5 Optional functions on auxiliary relays

From the above text it is apparent that auxiliary relays constitute an important facility in any programmable controller. This is basically due to their ability to control large numbers of associated contacts and perform as intermediate switching elements in many different types of control circuit.

In addition , many PLC manufacturers have provided additional , programmable functions associated with these auxiliary relays , to further extend their usefulness. A very common example is a 'pulse' function that allows any designated marker to produce a fixed – duration pulse at its contacts when operated , rather than the normal d. c. level change.

This pulse output is irrespective of the duration of relay operation , thus providing a very useful tool for applications such as program triggering , setting / resetting of timers and counters etc.

8.2-6 Pulse operation

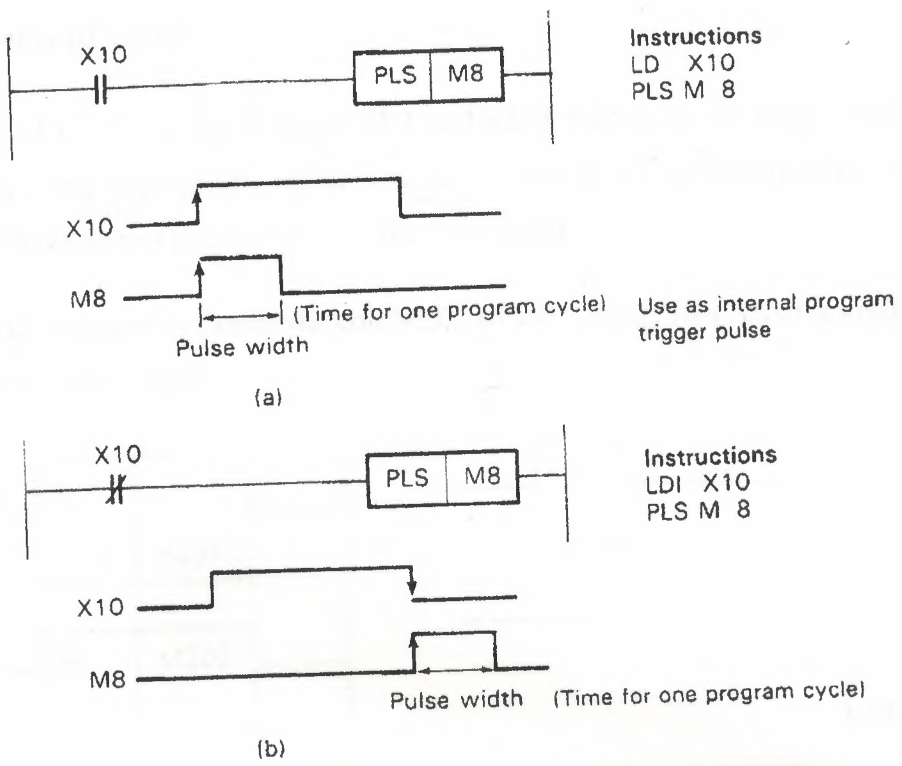
The programming of this feature varies between controllers , but the general procedure is the same , and very straightforward.

A pulse – PLS instruction is programmed onto an auxiliary relay number.

(in the figure 8.4)

This configures the designated relay to output a fixed – duration pulse when operated. The examples show how the relay may be used to output a pulse for either a positive or negative going input.

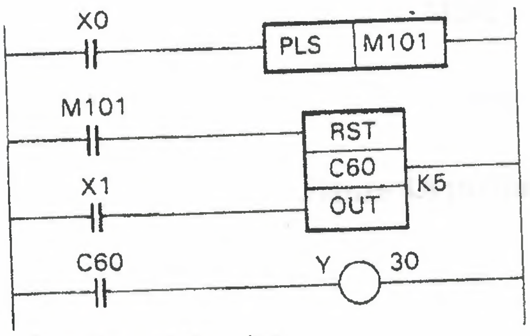
The circuit in figure 9.5 uses a PLS instruction on auxiliary relay 101 to provide a reset signal for a counter circuit C60. When input 0 is operated , a pulse is sent to relay 101 , causing its contacts to pulse and reset counter C60. This is used here because counters and timers often require short duration resetting to allow the restart of the counting or timing


X10
PLS
M8
X10
M8
(Time for one program cycle)
Pulse width
Use as internal program trigger pulse
(a)
X10
PLS
M8
X10
M8
Pulse width
(Time for one program cycle)
(b)

```
Instructions
LD   X10
PLS M 8
```

```
Instructions
LDI  X10
PLS M 8
```

Figure 8.4 Pulse function on auxiliary relays (a)rise detection circuit (b) drop detection circuit



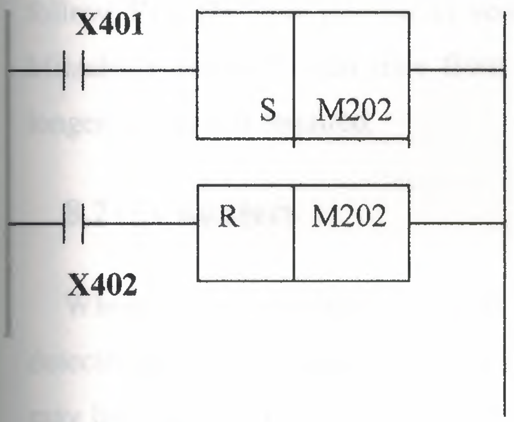
```
Step No.  Instruction
   0      LD   X0
   1      PLS  M101  — Auxiliary relay M101
   2      LD   M101
   3      RST  C60   ┐
   4      LD   X1    ├ Counter C60
   5      OUT  C60   ┘
   7      K    5       Count of five
  10      LD   C60
  11      OUT  Y30
  12      END
```

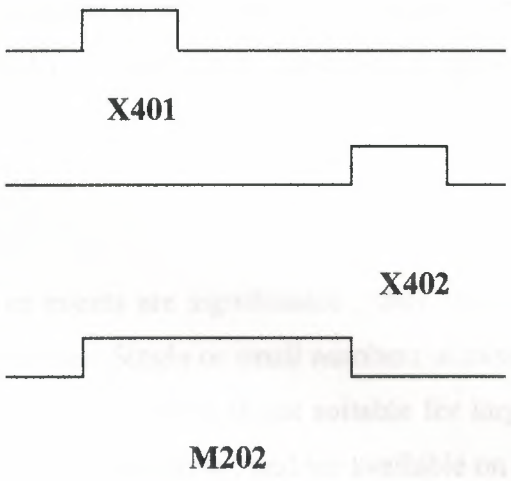
Figure 8.5 Providing a pulse input to a counter circuit

8.2-7 Set and reset

As with pulse – PLS , the ability to SET and RESET an auxiliary relay can often be produced by using appropriate instructions as in figure 8.6 These instructions are used to hold (latch) and reset the operation of the relay coils.

The S – set instruction causes the coil M202 to self – hold. This remains until a reset (R) instruction is activated.





(a)   (b)

Figure 8.6 (a) set/reset   **Figure 8.6 (b) time chart**

8.2-8 Timers

In a large proportion of control applications , there is a requirement for some aspect of timing control. PCs have software timer facilities that are very simple to program and use in a variety of situations.

The common method of programming a timer circuit is to specify the interval to be timed , and the conditions or events that are to start and / or stop the timer function. The initiating event may be produced by other internal or external signals to the controller. In this example the timer T450 is totally controlled by a contact related to output Y430. Thus , T450 begins timing only when Y430 is operated. This is caused by input X400 and not X401. Once activated , the timer will ‘ time – down’ from its preset value – in this case 3.5 seconds – to zero , and then its associated contacts will operate.

As with any other PLC contact, the timer contacts may be used to drive succeeding stages of ladder circuitry. Here the T450 contact is controlling output Y431. The enabling path to a timer may also form the 'reset' path , causing the timer to reset to the preset value whenever the path is opened. This is the case with most small PCs. The enabling path may contain very involved logic, or only a single contact.

Techniques for programming the preset time value vary little between different programmable controllers ,usually requiring the entry of a constant (K) command followed by the time interval in seconds and tenths of a second. The timers on this Mitsubishi controller can time from 0.1 – 999.9 s , and can be cascaded to provide longer intervals if required.

8.2-9 Counters

Whenever the number of process actions or events are significance , they must be detected and stored in some manner by the controller. Single or small numbers of events may be remembered by using latched relay circuits , but this is not suitable for larger event counts. Here programmable counter circuits are desirable , and are available on all PLCs.

Provided as an internal function , counter circuits are programmed in a similar manner to the timer circuits covered above , but with the addition of a control path to signal event counts to the counter block. Most PLC counters work as subtraction or ' down' counters , as the current value is decremented from the programmed set value

8.2-10 Registers

From using a single internal or external relay as a memory device to store a single bit of information , other PLC facilities allow the storage of several bits of data at one time.

The device used to store the data is termed a register ,and commonly holds 8 or 16 bits of information. Registers can be thought of as arrays of individual bit – stores – in fact many programmable controllers form the data registers out of groups of auxiliary marker relays in the figure 8.7

Registers are very important for handling data that originates from sources than simple , single switches. Instead of binary data in one – bit – wide form , information in a parallel data form may be read into and out of appropriately sized registers. Thus , data from devices such as thumbwheel switches , analog – to – digital converters , can be feed into appropriate PLC registers and used in later operations that will generate other bit – or byte – wide (8-bit) data to drive switched outputs or digital – to – analog conversion units.

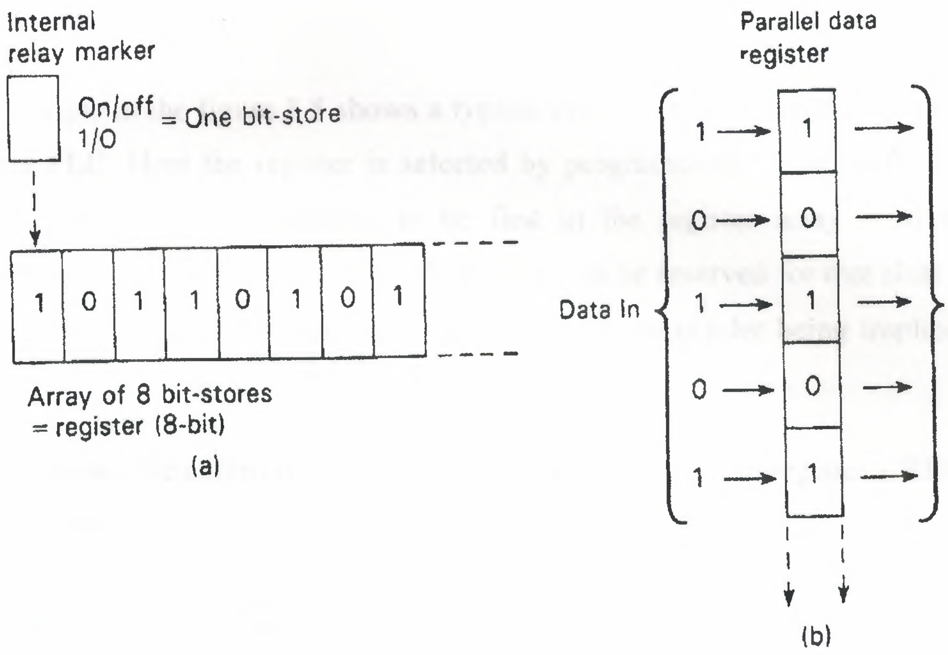


Figure 8.7 Register storage concept (a) array of bit stores; (b) parallel data register

8.2-11 Shift registers

A shift register provides a storage area for a sequence of individual data bits that are offered in series to its input line. The data are moved through the register under control of a shift or clock line as in the figure 8.8. The effect of a valid shift pulse is to move all stored digits one bit further in to the register , entering any new data in to the 'freed' initial bit positions. Since a shift register will only be a certain size . for example 8 or 16 bits , then any data in the last bit of the register will be shifted out and lost.

The usefulness of a shift register (SR) lies in the ability to control other circuits or devices via associated SR contacts that are affected by the shifting data stream through the register. That is , as with marker relays , when a marker is ON any associated contacts are operated.

In programmable controllers , shift registers are commonly formed from groups of the auxiliary relays. This allocation is done automatically by the user programming

a 'shift – register function', which than reserves the chosen block of relays for that register and prohibits their use for any other function (including use as individual relays).

The example in the figure 8.8 shows a typical circuit for shift register operation on a Mitsubishi PLC. Here the register is selected by programming in the shift instruction against the auxiliary relay number to be first in the register array – M160. This instruction causes a block of relays – M160 –167 – to be reserved for that shift register. Note that only the first relay had to be specified , the remainder being implied by the instructions.

This shows the controlling contacts on the input lines to the register – RESET. OUTPUT and SHIFT.

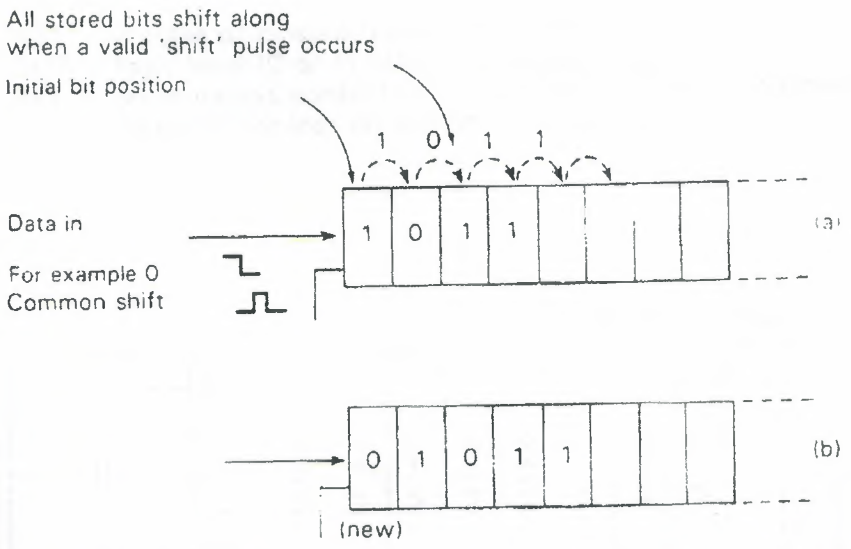
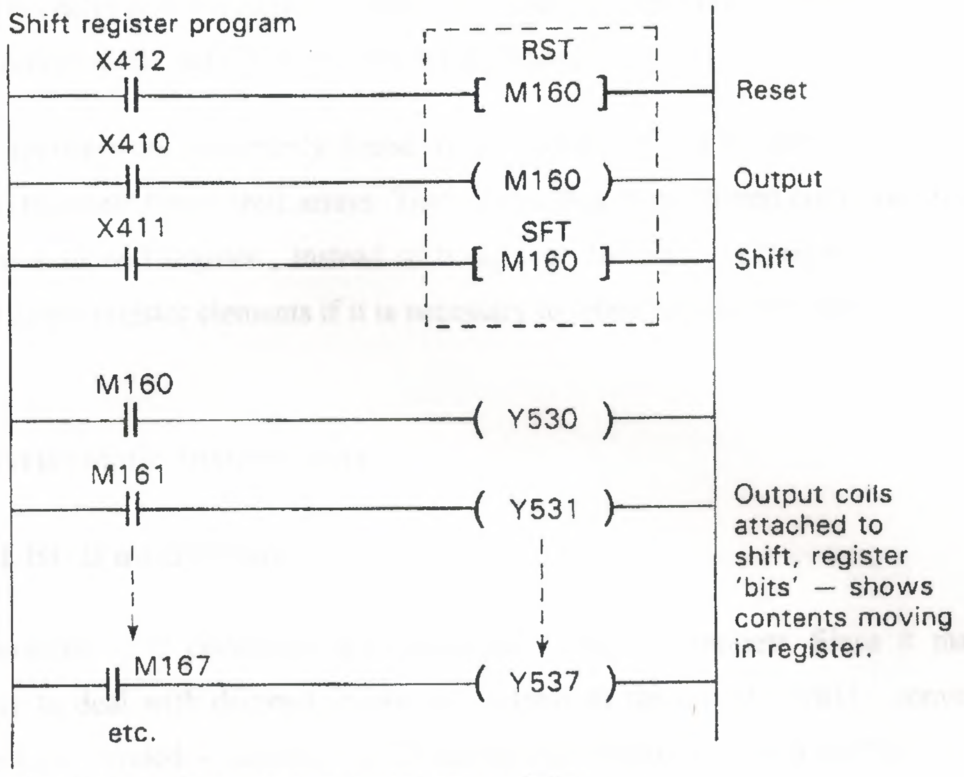


figure 8.8 Shift register operation: (a) before shift; (b) after 1 Shift pulse

The auxiliary relays can be grouped in blocks of 8 to form 8- or 16-bit shift registers. This feature is programmed as shown below using M160–177 internal relays (only M160 is keyed in, the other bits being transparent).



The shift register contacts perform as follows:

RST – a pulse or closure resets SR contents to 0
OUT – logic level (0 or 1) offered to register on this rung.
SFT – pulse moves contents along one bit at a time (eventually contents are lost off the final bit memory).

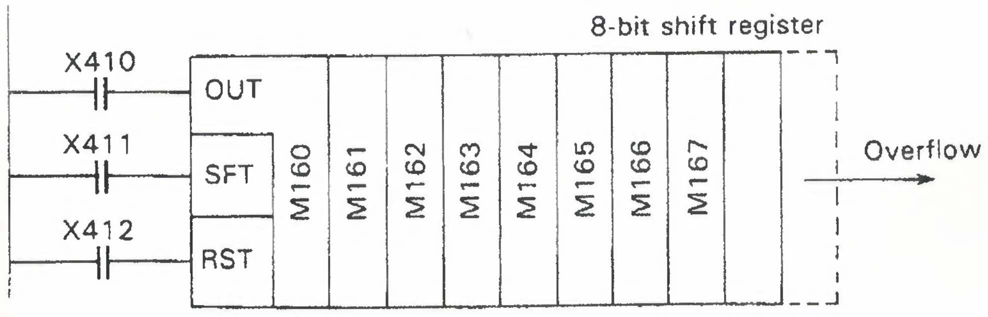


Figure 8.9 Basic shift register circuit shift register Figure 8.10 Equivalent circuit of a

Note the M – contacts below the SR circuit that are used to drive output coils (M160 – 167 driving Y530 –537).

It is easier to understand the function of the register if we look at an equivalent circuit in the figure 9.10. Here we can see the layout of other marker relays following M160. This helps us to visualize the shifting of data from bit to bit , affecting other parts of the circuitry as the data (1 or 0) in each bits change.

Shift registers are commonly found as 8 – bit or 16 – bit , and can usually be cascaded to create larger shift arrays. This allows data to be shifted out of one register and in to a second register , instead of being lost. Battery – backed markers can be selected as the register elements if it is necessary to retain register data through a power failure.

8.3 Arithmetic Instructions

8.3-1 BCD numbering

All internal CPU operations are performed in binary numbers. Since it may be necessary to deal with decimal inputs and outputs in the outside world , conversion using binary – coded - decimal (BCD)numbering is provided on most PLCs . BCD numbering is briefly described in figure 9.11 Readers wanting further information are referred to the many texts dealing with number systems. When data is already in binary format , such as analog values , it is placed directly in registers for use by other instructions.

(1) **BIN (pure binary)**

	X7	X6	X5	X4	X3	X2	X1	X0	
Upper digit	1	1	0	1	0	0	1	1	Lower digit
	128	64	32	16	8	4	2	1	

128 + 64 + 16 + 2 + 1 = 211

In the data made up of 8 bits from X0 to X7, the number 211 is expressed when X0, X1, X4, X6 and X7 are turned on (=1 in the above figure) and the others are turned off (=0 in the above figure).

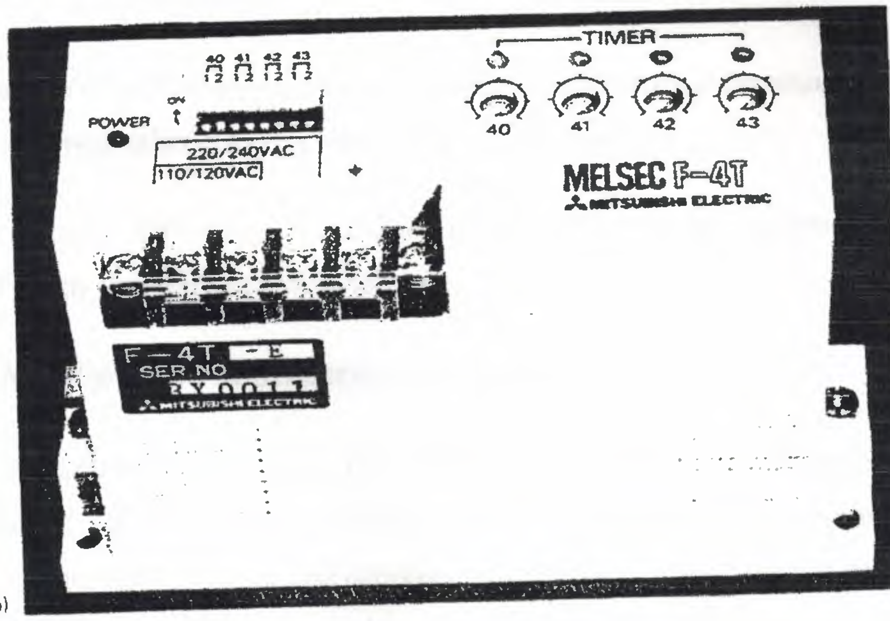
(2) **BCD (binary-coded decimal)**

M113	M112	M111	M110	M107	M106	M105	M104	M103	M102	M101	M100
1	0	0	1	0	1	1	0	0	1	1	1
800	400	200	100	80	40	20	10	8	4	2	1
100s digit				10s digit				1s digit			

(800 + 100) + (40 + 20) + (4 + 2 + 1) = 967

BCD data is such that each digit of the decimal number is expressed in 4-bit binary. No digit will exceed 9.
E.g.: If both M103 and M102 are turned on (= 1) in the BCD data shown in the above figure it will result in an error.
The values of timers or counters may be treated as BCD

(a)



(b)

Figure 8.11 (a) Binary and BCD number systems

(b) Timer unit for data operations

8.3-2 Magnitude comparison

Magnitude comparison instructions are used to compare a digital value read from some input device or timer , etc., with a second value contained in a destination data register. Depending on the instruction - more than , less than , or equal - ,this will result in a further operation when the condition is met. For example, a temperature probe in a furnace returns an analog voltage representing the current internal temperature. This is converted in to a digital value by an analog – to – digital converter module on the PC , where it is read from input points by a data – transfer instruction and stored in data register D10. The process requires that if the temperature is less than 200 C , then the process must halt due to insufficient temperature.

If the temperature is greater than 200 C and less than 250 C , then the process operates at normal rate. If the temperature is between 250 and 280 C , than baking time is to be reduced to 3 minutes 25 seconds , and once temperature exceeds 280 C the process is to be suspended.

This the type of area where magnitude comparison can provide the necessary control, in conjunction with other circuitry to drive the plant equipment.

Other common applications include the checking of counter and timer values for action part – way through a counting sequence.

8.3-3 Addition and subtraction instructions

These instructions are used to alter the value of data held in data registers by a certain amount. This may be used simply to add / subtract an offset to an input value before it is processed by other instructions. For example , when two different sensors are passing values to the controller and one sensor signal has to be compared against the other , but is a fundamentally smaller signal with a narrower output swing. It may be possible to add an offset to the smaller signal to bring it up near to the level of the larger one , thus allowing comparison to take place. The alternative would be to use signal conditioning units to raise the sensor output before the PLC - an expensive option.

Other uses of + and – include the alteration of counter and timer presets by programmed increments when certain conditions occur.

9 - LADDER PROGRAM DEVELOPMENT

9.1 Software Design

When ladder programs are being developed to control simple actions or equipment , the amount of planning and actual design work for these short programs is minimal , mainly because there is no requirement to link with other actions or sections within the program. The ladder networks involved are small enough to be easily understood in terms of circuit representation and operation. In practice , of course , circuits are not limited to AND or OR gates, often involving mixed logic functions together with the many other programmable functions provided by modern programmable controllers.

When larger and more complex control operations have to be performed , it quickly becomes apparent that an informal and unstructured approach to software design will only result in programs that are difficult to understand , modify , troubleshoot and document. The originator of such software may posses an understanding of its

operation , but this knowledge is unlikely to remain after even a short period of time away from that system.

In terms of design methodology , than , ladder programming is no different from conventional computer programming. Thus , considerable attention must be given to :

* Task definition / specification

* Software design techniques

* Documentation

* Program testing

9.1-1 System functions

Most industrial control systems may be considered as a set of functional areas or

blocks , in order to aid the understanding of how the total system operates.

For example , each machine in a plant unit can be treated as a separate sub – process. Each machine process is then broken down in to blocks that may be described in terms of basic sequences and operations in the figure 10.1 illustrates this approach.

A functional block could for example , consist of all actions required to control a certain machine in the process.

Block 1

Block 2

Block 3

The division of programming tasks in to functional blocks is an important part of software design.

In logic programming , there are two different types of network that may be used to implement the function of a given block:

- Interlocks or combinational logic, where the output is purely dependent on the combination of the inputs at any instant in time.
- Sequential networks where the output is dependent not only on the actual inputs but on the sequence of the previous inputs and outputs.

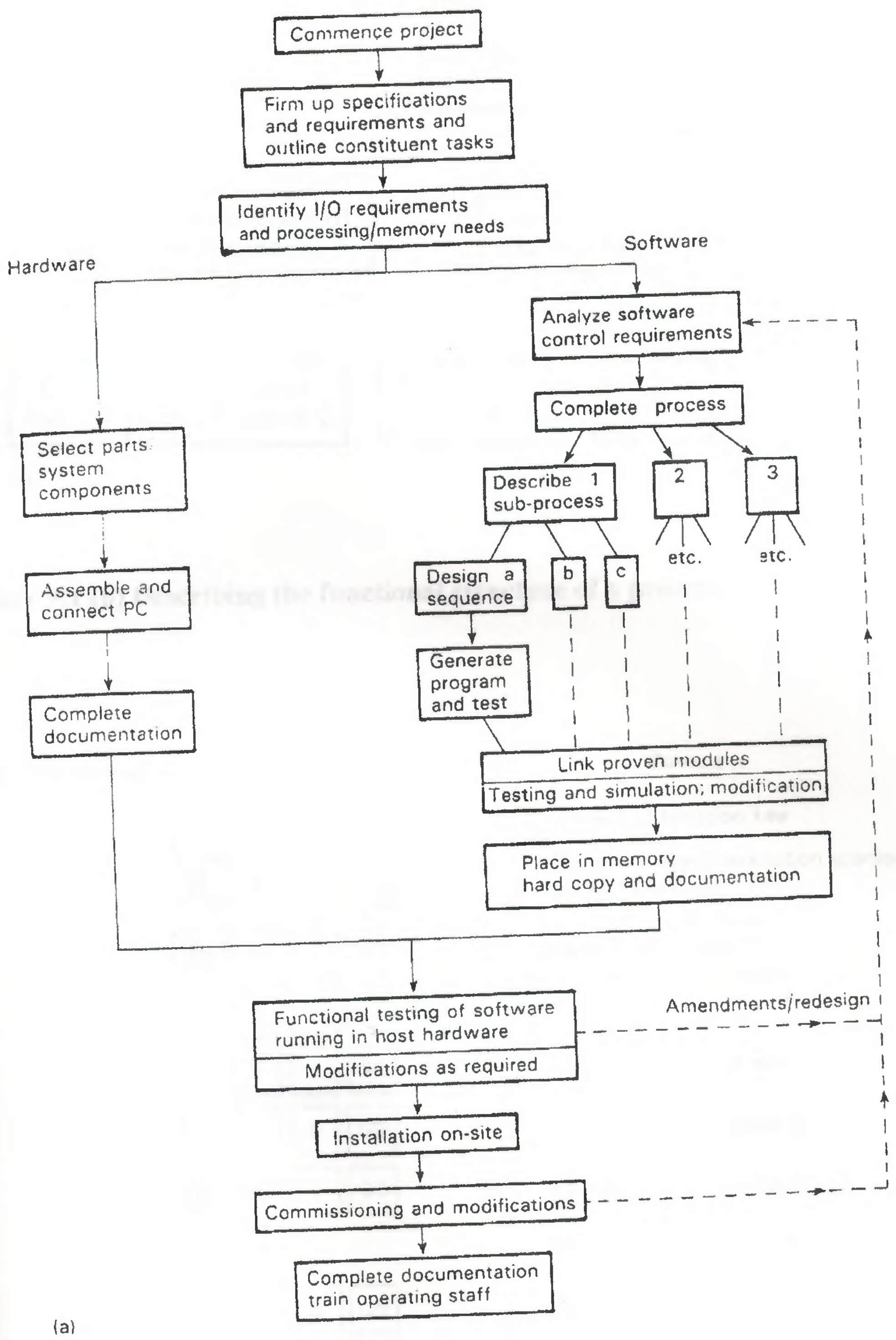
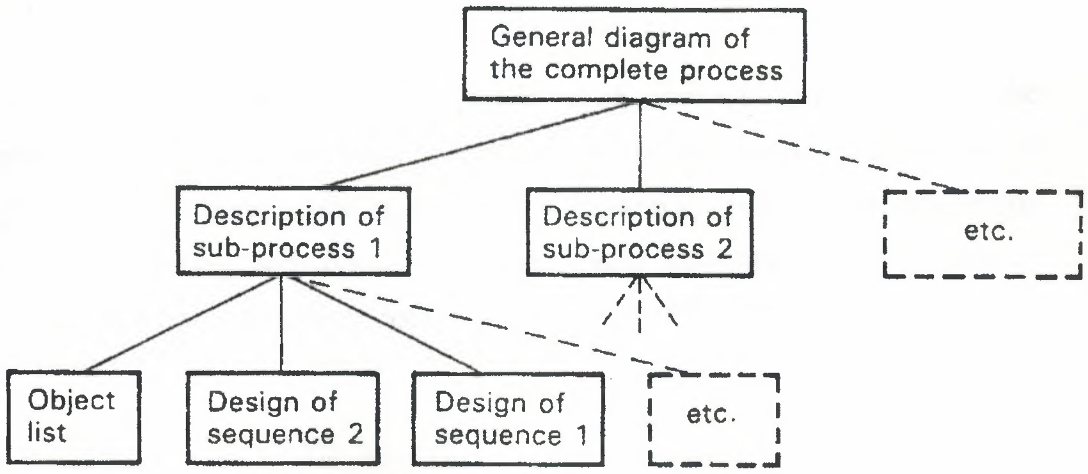

Commence project
Firm up specifications and requirements and outline constituent tasks
Identify I/O requirements and processing/memory needs
Hardware
Software
Analyze software control requirements
Complete process
Describe 1 sub-process
2
3
etc.
etc.
Design a sequence
b
c
Generate program and test
Link proven modules
Testing and simulation; modification
Place in memory hard copy and documentation
Select parts system components
Assemble and connect PC
Complete documentation
Functional testing of software running in host hardware
Modifications as required
Amendments/redesign
Installation on-site
Commissioning and modifications
Complete documentation train operating staff
(a)

Figure 9.1 (a) PC system design procedure


General diagram of the complete process
Description of sub-process 1
Description of sub-process 2
etc.
Object list
Design of sequence 2
Design of sequence 1
etc.

(b)

Figure 9.1 (b) Describing the functional structure of a process

Entry procedure	*Key operation*	*Comment*
1	LDR WR	Mode and function key
2.	⊣⊢ X 2 GO	Symbol element execution (cursor moves to next position)
3.	⊣⊢ X 3 GO	Where the ladder symbol is the same as the one used previously, it need not be rekeyed
4.	⊣⊢ X 4 GO	
5.	-○- Y 1 GO	Output coil, cursor moves to next line
6.	┤├ X 0 GO	Parallel contact across X2
7.	┤├ X 1 GO	Parallel contact across X2
8.	↓ ←	Move cursor to next line
9.	⊣⊢ X 5 GO	Parallel contact across previous contacts
10.	— GO — GO	Horizontal circuit links to point B
11.	| GO ↑ ← | GO	Vertical links up to point A, using cursor keys
12.	CNV GO	Editing and writing code to RAM (compiled into logic instructions)

Figure 9.2 Graphic programming

9.2 Program Structure

At this stage in the investigation of design techniques , it is appropriate to discuss the layout and structure of PLC programs. It is sound practice to base any program layout on the general operating structure of all process – control systems. This means having definite sections dealing with operating modes , basic functions , process chain or sequence , signal outputs and status display , as indicated in Table 9.1.

Table 9.1 Sections of a PLC program

Start
Operating modes and basic functions
starting (basic) position
Enabling / reset conditions
Process operation / sequence logic
Signal outputs
Status / indicator output
Finish

a- Operating modes

Basic position: The controlled equipment is likely to have a basic or normal position , for example when all actuators are off and all limit switches are open. All these elements can be combined logically to signify and initialize a basic position , which may be programmed as a step in a sequential process.

Enabling / reset conditions: Most industrial processes have manual start and stop controls that may be incorporated in to the PLC program structure at this point. These would be included as enabling and reset contacts , having overall control of the PLC in terms of run or stop. There may also be a manual switch to enable the system outputs , which would allow the program to run without driving the physical outputs connected to the PLC a test function.

b- Process operation / sequence

This is the main topic of this chapter , involving the design and programming of combinational and sequential networks as necessary. The resultant outputs do not normally drive actuators directly , but instead are used to operate intermediate marker relays.

c- Signal output

Output signals to process actuators are formed by interlocking the resulting operation sequence outputs (markers) with any enabling conditions that exist in (a) above.

d- Status / indicator outputs

Process status is often displayed using indicator lambs or alarms, etc. Such elements are programmed in this section of the software.

By adopting this systematic approach to program structure , we can create reliable , easily understood software , which will allow rapid fault location and result in short process down times. The program that are developed in this chapter deal mainly with the topic of process operation , but will be structured in this manner where possible.

9.3 Further Sequential Control Techniques

In many practical applications , a control system has to deal with a process sequence that requires the concurrent operation and control of more than one step. Also , steps in a sequence may require a time delay or event count as entry criteria for a succeeding step. To describe the different types of parallel operation , we use the conventions

In figure , actions B OR C are taken , depending on the result of test A2. Either action will allow entry to action D. In the figure shows the format for a process where two actions A AND B are initialized once test A is true; also both tests B AND C must be true before progression to action D.

The equivalent function chart descriptions are illustrated . The number of parallel activities may be extended via the branching and converging rails. The chart in figure shows the tests that allow entry to steps B OR C , and also the individual tests or conditions that will allow resetting of the chosen step (test n and m). Notice the OR signs at each branch rail.

In figure the AND ing of steps is signified by the double connecting rails after test A and before test n. This means that all parallel steps (in this case B and C) are set once state A is active and test A is fulfilled.

9.4 Limitation of Ladder Programming

1 - Ladder programs are ideal for combinational / interlock tasks and simple sequential tasks. However, the lack of comment facilities on most small programmers makes interpretation of any program extremely difficult.

2 - When applied to complex sequential tasks , ladder programs become cumber some , difficult both to design and debug. This is mainly due to having to provide entry , hold and reset elements in every stage to ensure no sequence errors occur.

Several manufacturers are adopting a function block style of programming that removes most of this complexity. This employs basic programming symbols that are closely related to the function chart symbols used for program design purposes , as used earlier in this chapter.

9.4-1 Advanced graphic programming languages

The facility for programming using functional blocks is currently available on a few larger programmable controllers , such as those from Siemens and Telemechanique. This approach uses graphic blocks to represent sections of circuitry related to a particular task or part of a process. Each function block is user programmed to contain a section of ladder circuitry required to carry out that function. The sequential operation of the control system is obtained by progression from one block to next , where a step is entered only if it is entry conditions are fulfilled , in which case it becomes active and the previous step becomes inactive.

Thus , there is no need to reset the previous step – an important advantage over conventional ladder programming.

To examine or program the contents of each block the user would zoom in on the block in question. This windows in on the contents as shown , which are displayed on the programming panel. The necessary details are then entered in normal ladder format.

Provision for displaying simultaneous sequences is a further important feature of this programming methods , displaying the multi – tasking ability of PLC s in an easy – to – follow manner.

9.4-2 Workstations

The traditional tool for programming PLC s is the small , hand – held panel which can provide only limited monitoring and editing facilities. Most manufacturers are now using personal computers as workstations on larger programmable controllers , in order to fully exploit these features , and those of graphic function blocks.

10-CHOOSING INSTALLATION AND COMMISSIONING OF PLC SYSTEM

10.1 Feasibility Study

Under certain circumstances an initial feasibility study may be suggested or warranted , prior to any decision on what solution will be adopted for a particular task. The feasibility study may be carried out either by in – house experts or by external consultants. Often an independent specialist is preferred , having few or no ties to specific vendor equipment.

The scope of such a study can vary enormously , from simply stating the feasibility of the proposal , through to a comprehensive case analysis with complete equipment recommendations. Typically , though , a feasibility study of this nature encompasses several specific areas of investigation:

(a) **economic feasibility** , consisting of the evaluation of possible installation and development costs weighed against the ultimate income or benefits resulting from a developed system ;

(b) **technical feasibility** , where the target process and equipment are studied in terms of function , performance and constraints that may relate to achieving an acceptable system;

(c) **alternatives** , with an investigation and evaluation of alternative approaches to the development of the acceptable system.

Area (a) , economic feasibility and worth , can only be addressed fully once the result of areas (b) and (c) are available ,with estimated castings , and direct / indirect benefits being considered. Area (b) is detailed in the following sections , with background information for area (a) usually being compiled through liaison with company personnel. The achievement of a complete technical proposal requires us to know what the present and future company needs are in terms of plant automation and desired information systems.

Once the control function has been accurately defined , a suitable programmable control system has to be chosen from the wide range available. Following the identification of a suitable PLC , work can begin on aspects of electrical hardware design and software design.

10.2 Design Procedure for PLC Systems

Because the programmable controller is based on standard modules , the majority of hardware and software design and implementation can be carried out independently of , but concurrently with , each other.

Developing the hardware and software in parallel brings advantages both in terms of saving time and of maintaining the most flexible an adaptable position regarding the eventual system function. This allows changes in the actual control functions through software, until the final version is placed in the system memory and installed in the PLC.

An extremely important aspect of every design project is the documentation.

Accurate and up – to – date documentation of all phases of a project need to be fully documented and updated as the job progresses through to completion. This information will form part of the total system documentation, and can often be invaluable during later stages of commissioning and troubleshooting.

10.2-1 Choosing a programmable controller

There is a massive range of PLC systems available today, with new additions or replacement continually being produced with enhanced features of one type or another. Manufacturers quickly adopt advances in technology in order to improve the performance and market status of their products. However, irrespective of make, the majority of PLC s in each size range is very similar in terms of their control facilities. Where significant differences are to be found is in the programming methods and languages, together with differing standards of manufacturer support and backup. This latter point is often overlooked when choosing a suitable make of controller, but the value of good, reliable manufacturers assistance cannot be overstated, both for present and future control needs.

10.2-2 Size and type of PLC system

This may be decided in conjunction with the choice of manufacturer, on the basis that more than one make of machine can satisfy a particular application, but with the vast choice of equipment now available, the customer can usually obtain similar systems from several original equipment manufacturers (OEMs). Where the specification requires certain types of function or input / output, it can result in one system from a single manufacturer standing out as far superior or cost – effective than the competition, but this is rarely the case. Once the stage of deciding actual size of the PLC system is reached, there are several topics to be considered:

- Necessary input / output capacity;
- Types of I / O required;

- Size of memory required;
- Spccd and powcr rcquircd of thc CPU and instruction sct.

All this topics are to a large extent interdependent, with the memory size being directly tied to the amount of I / O as well as program size. As the I / O memory size rises, this takes longer to process and requires a more powerful, faster central processor if scan times are remain acceptable.

10.2-3 I / O requirements

The I / O sections of a PLC system must be able to contain sufficient modules to connect all signal and control lines for the process. These modules must conform to the basic system specifications as regards voltage levels, loading, etc.,

- The number and type of I / O points required per module;
- Isolation required between the controller and the target process;
- The need for high speed I / O, or remote I / O, or any other special facility;
- Future needs of the plant in terms of both expansion potential and installed spare I / O points;
- Power supply requirements of I / O points – is an on – board PSU needed to drive any transducer or actuators?

In certain cases there may be a need for signal conditioning modules to be included in the system , with obvious space demands on the main or remote racks. When the system is to be installed over a wide area, the use of a remote or decentralized form of I / O working can give significant economies in cabling the sensors and actuators to the PLC.

10.2-4 Memory and programming requirements

Depending on the type of programmable controller being considered, the system memory may be implemented on the same card as the CPU, or alternatively on dedicated cards. This ladder method is the more adaptable, allowing memory size to be increased as necessary up to the system maximum, without a reciprocal change in CPU card.

As stated in the previous section, memory size is normally related to the amount of I/O points required in the system. The other factor that affects the amount of memory required is of course the control program that is to be installed. The exact size of any program cannot be defined until of the software has been designed, encoded, installed and tested. However, it is possible to accurately estimate this size based on average program complexity. A control program with complex, lengthy interlocking or sequencing routines obviously requires more memory than one for a simple process. Program size is also related to the number of I/O points, since it must include instructions for reading from or writing to each point. Special functions are required for the control task may also require memory space in the unit PLC memory map to allow data transfer between cards. Finally additional space should be provided to allow for changes in the program, and for future expansion of the system.

There is often a choice of available memory type – RAM or EPROM. The RAM form is the most common, allowing straightforward and rapid program alterations both before and after the system is installed. RAM contents are made semi permanent by the provision of battery – backing on their power supply. RAM must always be used for I / O and data functions, as these involve dynamic data.

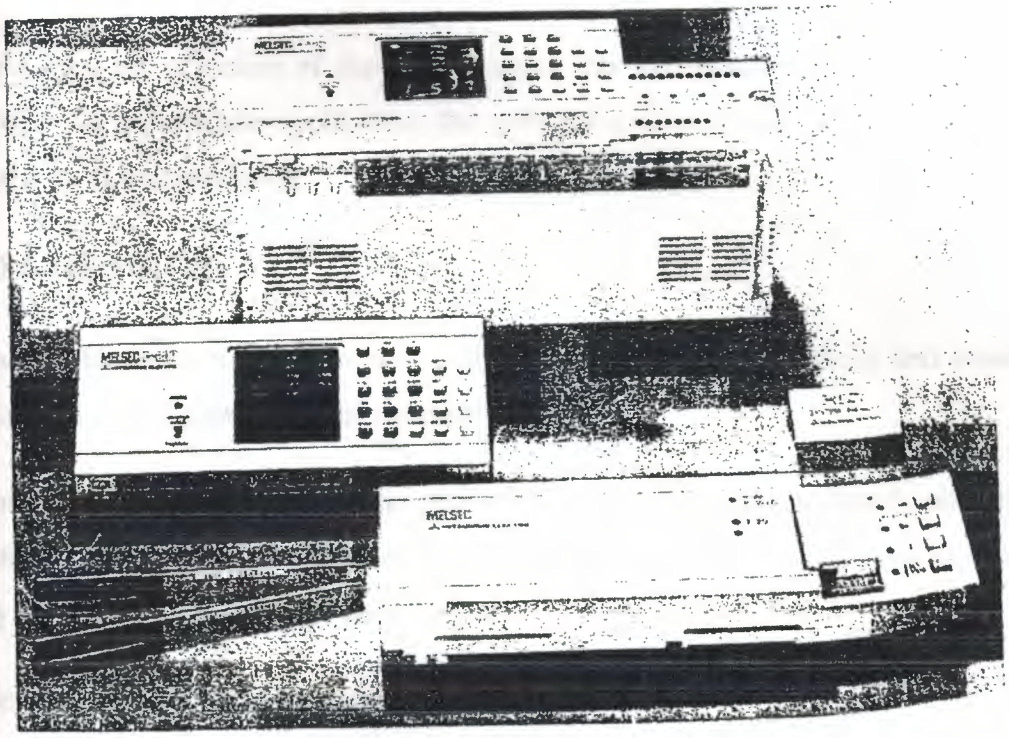
EPROM memory can be employed for program storage only, and requires the use of a special EPROM eraser / programmer to alter the stored code. The use of EPROMS is ideal where identical programmable controllers running the same control several machines.

However, until a program has been a fully developed and tested, RAM storage should be used.

As mentioned in earlier chapters, microcomputers are commonly used as program development stations. The large amount of RAM and disk storage space provided in these machines allows the development and storage of many PLC programs, including related text and documentation. Programs can be transferred between the microcomputer and the target PLC for testing and alteration. EPROM programming can also often be carried out via the microcomputer.

Input/output memory
+
Control program memory
+
Special function tables
+
Space for changes and future expansion

(a)



(b)

Figure 10.1 (a) PLC memory requirements for different tasks.

(b) Custom EPROM programmer for a Mitsubishi F series PLC

10.2-5 Instruction set / CPU

Whatever else is left undefined; any system to be considered must provide an instruction set that is adequate for the task. Regardless of size, all PLCs can handle logic control, sequencing, etc. Where differences start to emerge are in the areas of data handling, special functions and communications. Larger programmable controllers tend to have more powerful instructions than smaller ones in these areas, but careful scrutiny of small / medium machines can often reveal the capability to perform specific functions at surprisingly good levels of performance.

In modular programmable controllers there may be a choice of CPU card, offering different levels of performance in terms of speed and functionality. As the number of I / O and function cards increases, the demands on the CPU also increase, since there are greater numbers of signals to process each cycle. This may require the use of a faster CPU card if scan time is not to suffer.

Following the selection of the precise units that will make up the programmable controller for a particular application, the software and hardware design functions can be carried out independently.

10.3 Installation

The hardware installation consists of building up to necessary racks and cubicles, then installing and connecting the cabling.

The cabinet that contains the programmable controller and associated sub – racks (see figure 10.2) must be adequate for the intended environment, as regards security safety band protection from the elements:

Security in the form of a robust, lockable cabinet;

Safety, by providing automatic cut – off facilities / alarms if the cabinet door is opened;

Protection from humid or corrosive atmospheres by installation of airtight seals on the cubicle. Further electrostatic shielding by ear thing the cubicle body.

For maintenance purposes, there must be easy access to the PLC racks for card inspection, changing etc. Main on / off and status indicators can be built in to the cabinet doors, and glass or Perspex windows fitted to allow visual checking of card status or relay / contactor operation.

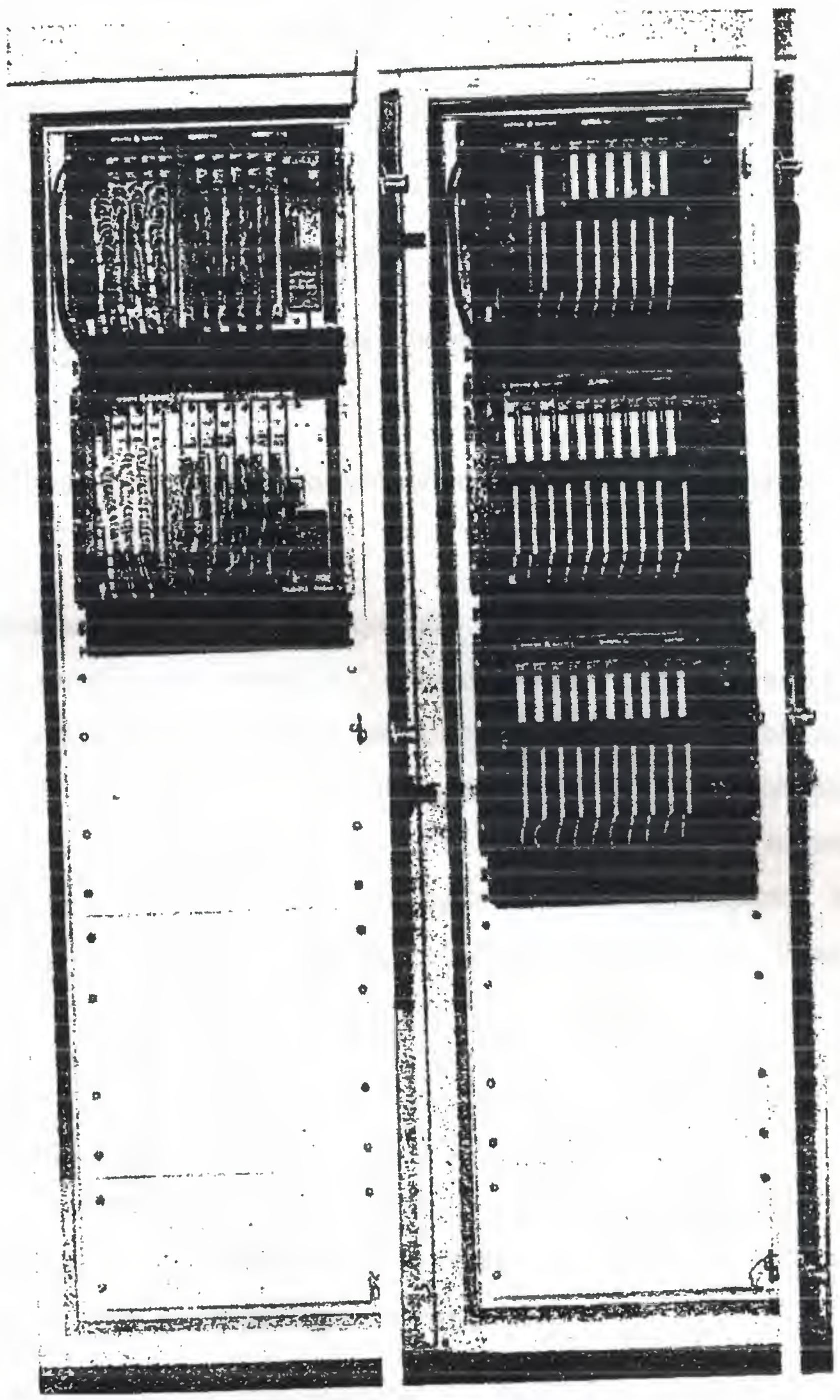


Figure 10.2 Complete PLC installation and cabinet

10.4 Testing and Commissioning

Once the installation work is completed, the next step is to consider the testing and commissioning of the PLC system.

Commissioning comprises two basic stages:

1- Checking the cable connections between the PLC and the plant to be controlled.

2- Installing the completed control software and testing its operation on the target process.

The system interconnections must be thoroughly checked out to ensure all input / output devices are wired to the correct I / O points. In a conventional control system buzzing out the connections with suitable continuity test instruments would do this. With a programmable, however, the programming panel may be used to monitor the status of inputs points directly – this is long before the control software is installed, which will only be done after all hardware testing is satisfactorily completed. Before any hardware testing is started, a thorough test of all mains voltages, ear thing, etc., must be carried out.

With the programmer attached to the PLC, input points are monitored as the related transducer is operated, checking that the correct signal is received by the PLC. The same technique is used to test the various function cards installed in the system. For example, altering can check analog inputs the analog signal and observing a corresponding change in the data stored in the memory table.

In turn , the output devices can be forced by instructions from the programming panel , checking their connection and operation. The commissioning team must ensure that any operation or misoperation of plant actuators will not result in damage to plant or personnel.

Testing of some PLC functions at this stage is not always practical, such as for PID loops and certain communications channel. These require a significant amount of configuring by software before they can be operated, and are preferably tested once the control software has been installed.

Some programmable controllers contain in – built diagnostic routines that can be used to check out the installed cards, giving error codes on a VDU or integral display screen. These diagnostics are run by commands from the programming panel, or from within a control program once the system is fully operational.

10.4-1 Software testing and simulation

The preceding sections have outlined the various stages in hardware design and implementation. Over the same period of time, the software to control the target process is developed, in parallel, for the chosen PLC system. These program modules 0should be tested and proved individually wherever possible, before being linked together to make up the complete applications program. It is highly desirable that any faults or error be removed before the program is installed in the host controller.

The time required to rectify faults can be more than doubled once the software is running in the host PLC.

Virtually all-programmable controllers, irrespective of size, contain elementary software – checking facilities. Typically these can scan through an installed program to check for incorrect labels. Double output coils etc. Listings of all I / O points used, counter / timer settings and other information is also provided. The resulting information is available on the programmer screen or as a printout in the figure 10.3. However, this form of testing is only of limited value, since there is no facility to check the operation of the resident program.

In terms of time and cost economies, an ideal method for testing program modules is to reproduce the control cycle by simulation, since this activity can be carried out in the design workshop without having the actually connect up to the physical process. Simulation of the process is done in a number of ways, depending on the size of process involved.

When the system is relatively small with only a handful of I / O channels, it is often possible to adequately simulate the process by using sets of switches connected up to the PLC as inputs, with outputs represented by connecting arrays of small lambs or relays in the figure 10.4. This allows inputs to be offered to a test – bed controller containing software under test, checking the action of the control program by noting the operation and sequence of the output lambs or relays. By operating the input switches in specific sequences, it is possible to test sequence routines within a program. Where fast response times are involved, the tester should use the programming panel to force larger time intervals into the timers concerned, allowing that part of the circuit to be tested by the manual switch method.

Most I / O modules have LED indicators that show the status of the channels. These can be used instead of additional test actuators where digital outputs are concerned. Analog inputs can be simulated in part by using potential dividers suitably connected to the input channel, and corresponding analog outputs connected either to variable devices such as small motors or to a moving coil meter configured to measure voltage or current. Standard sets of input switches and output actuators are normally available from PLC manufacturers.

When the system is larger with input / output channels and longer, more complex programs, the simple form of simulation described above becomes inadequate. Many larger PLC systems are fitted an integral simulation unit that reads and writes information directly into the I / O memory, removing the need to connect external switches, etc. The simulator is controlled from an associated terminal, which can force changes in input status and record all changes in output status as the program runs, for later scrutiny by the test team.

The program monitoring facility provided with most programming terminals should be used in virtually all these proceedings, since it allows the dynamic checking of all elements in the program including preset and remaining values as the program cycles. In the figure 10.5 illustrates a monitoring display with status information shown on the bottom of the screen.

It is important to realize that the display on the programmer does not up date as rapidly as the control program is executing, due to the delays in transmitting the data across to the terminal.

Contacts and other elements that are operated for only a few scans are unlikely to affect the display, but since a human observer could not detect this fast a change, this is not a significant disadvantage. To display all changes, the PLC should be run in single step mode.

The monitor display shows a select portion of the ladder program, using standard symbols to depict contacts, output and present functions. All elements within the display are dynamically monitored, indicating their status as shown in the figure 10.6

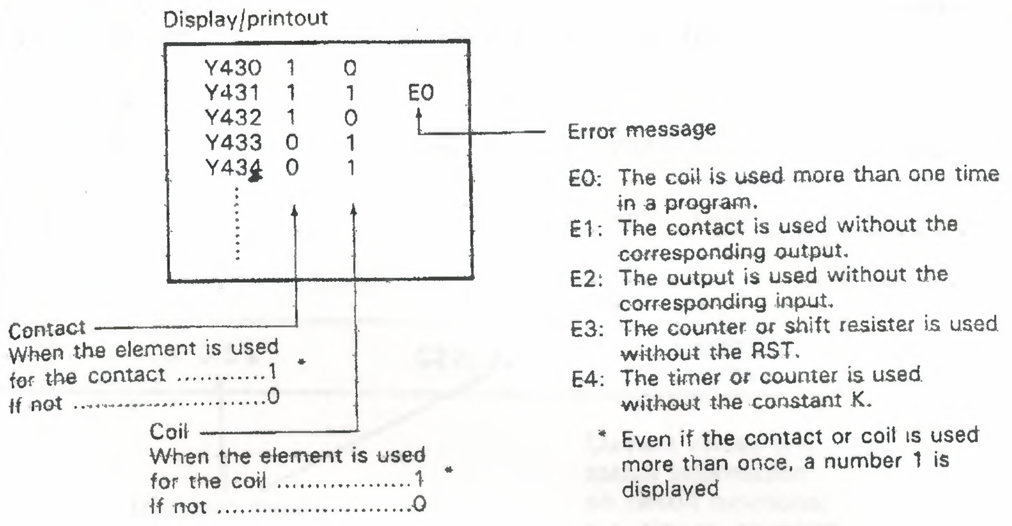

Display/printout
Y430 1 0
Y431 1 1 E0
Y432 1 0
Y433 0 1
Y434 0 1
Error message
E0: The coil is used more than one time in a program.
E1: The contact is used without the corresponding output.
E2: The output is used without the corresponding input.
E3: The counter or shift resister is used without the RST.
E4: The timer or counter is used without the constant K.
* Even if the contact or coil is used more than once, a number 1 is displayed
Contact
When the element is used for the contact ....1 *
If not 0
Coil
When the element is used for the coil ....1 *
If not 0

Figure 10.3 PLC printout of I/O static diagnostics information

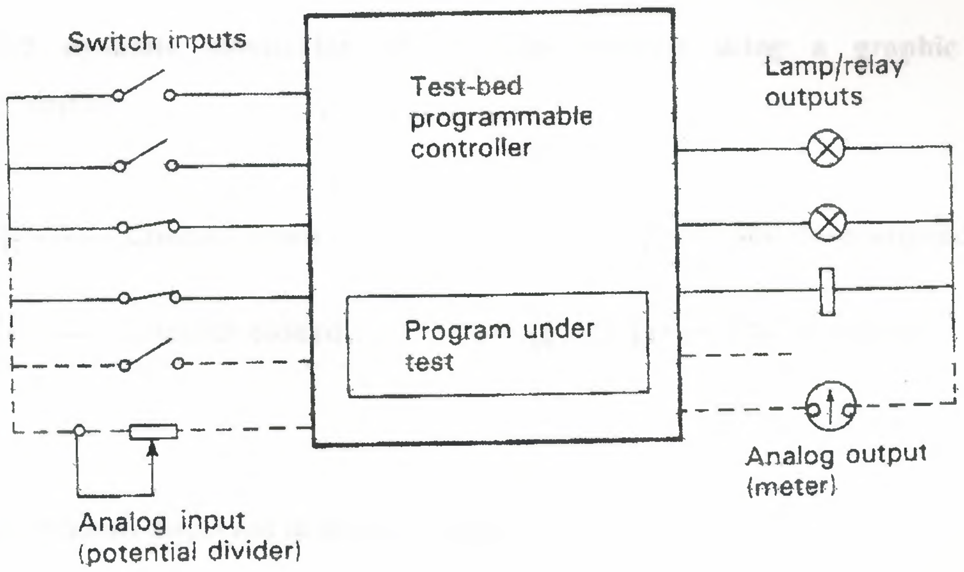
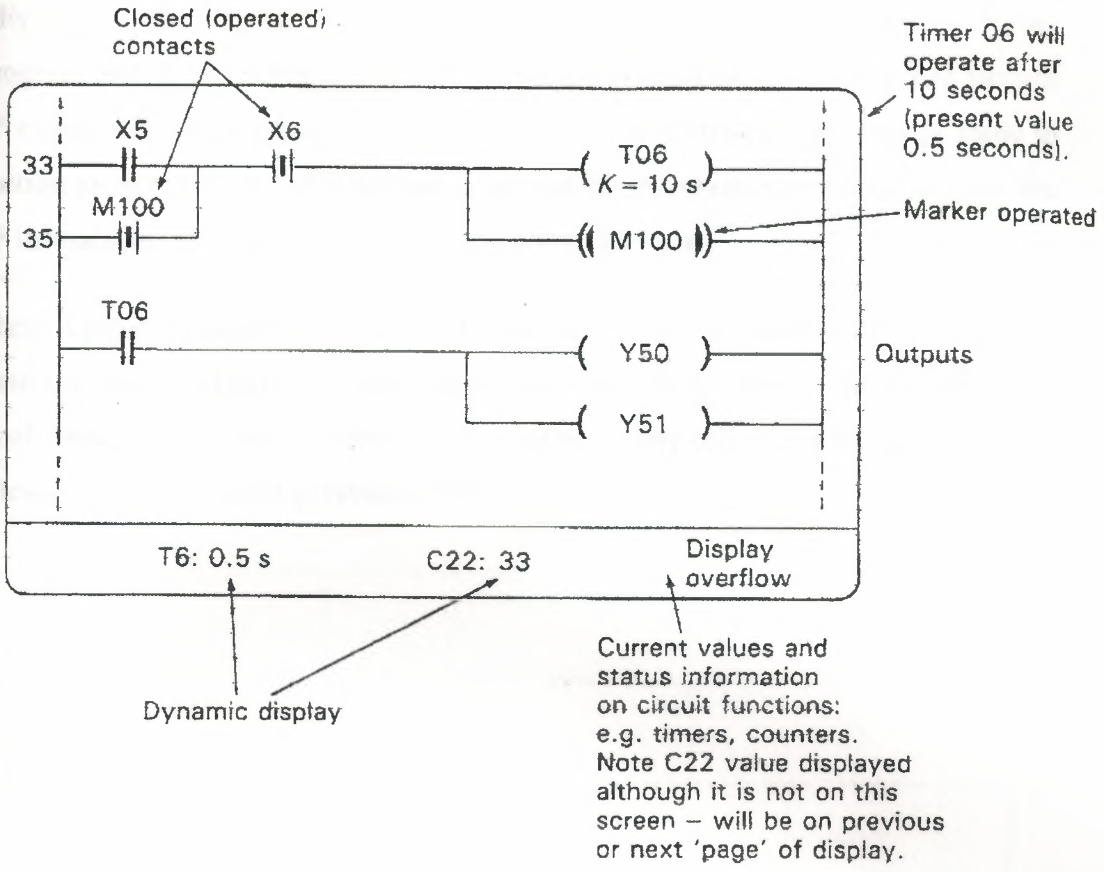


Figure 10.4 Process simulation using switches and lambs



Closed (operated) contacts
X5
X6
33
35
M100
T06
T06 K = 10 s
M100
Y50
Y51
Timer 06 will operate after 10 seconds (present value 0.5 seconds).
Marker operated
Outputs
T6: 0.5 s
C22: 33
Display overflow
Dynamic display
Current values and status information on circuit functions: e.g. timers, counters. Note C22 value displayed although it is not on this screen – will be on previous or next 'page' of display.

Figure 10.5 dynamic monitoring of program contacts using a graphic programming display

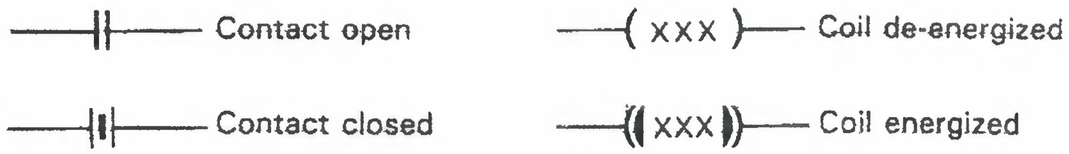


Figure 10.6 Symbols displayed in monitor mode

10.4-2Installing and running the user control program

Once the control software has been proved as far as possible by the above, methods on a test machine, the next step is to try out the program on the tested PLC hardware installation. Ideally each section of code should be downloaded and tested

Individually, allowing faults to be quickly localized if the plant misoperates during the program test. If this subdivided testing is not possible, another method is to include JUMP commands in the complete program to miss out all instructions except those in the section to be tested. As each section is proved, the program is amended to place the JUMP instructions so as to select the next section to be tested.

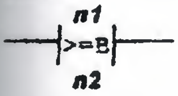
Where a programmable controller supports single – step operation , this can be used the examine individual program steps for correct sequencing. Again , the programming terminal should be utilized to monitor I / O status or any other area of interest during these tests , with continuous printouts if this is possible.

Table of Symbol

INSTRUCTION	LADDER SEMBOL	SIMATIC S7
LOAD	|—| |—	LD
AND	—| |—	A
OR	└| |┘	O
NOT	/	NOT
LOAD NOT	|—|/|—	LDN
AND NOT	—|/|—	AN
OR NOT	└|/|┘	ON
AND BLOCK		ALD
OR BLOCK		OLD
OUT	—O—| —( )—|	=
END	—(END)—	MEND

Compare Byte Greater Than Or Equal Contact

Symbol:



Operands:

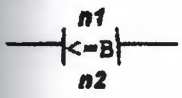
n1. n2 (unsigned byte):	VB. IB. QB. MB. SMB. AC. Constant. *VD. *AC

Description of operation:

The Compare Byte Greater Than or Equal Contact is closed when the byte value stored at address n1 is greater than or equal to the byte value stored at address n2 . Power flows through the contact when closed.

Compare Byte Less Than Or Equal Contact

Symbol:



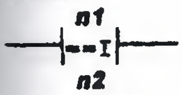
Operands:

n1. n2 (unsigned byte):	VB. IB. QB. MB. SMB. AC. Constant. *VD. *AC

Description of operation:
The Compare Byte Less Than or Equal Contact is closed when the byte value stored at address n1 is less than or equal to the byte value stored at address n2 . Power flows through the contact when closed.

Compare Integer Equal Contact

Symbol:



Operands:

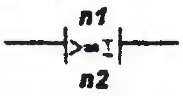
n1. n2 (signed integer word).	VW. T.C.IW. QW. MW. SMW. AC. AIW. Constant. *VD. *AC

Description of operation:

The Compare Integer Equal Contact is closed when the signed integer word value stored at address n1 is equal to the signed integer word value stored at address n2 . Power flows through the contact when closed.

Compare Integer Greater Than Or Equal Contact

Symbol:



Operands:

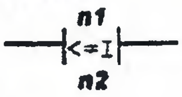
n1. n2 (signed integer word):	VW. T. C. IW. QW. MW. SMW. AC. AIW. Constant. *VD. *AC

Description of operation:

The Compare Integer Greater Than or Equal Contact is closed when the signed integer word value stored at address n1 is greater than or equal to the signed integer word value stored at address n2 . Power flows through the contact when closed.

Compare Integer Less Than Or Equal Contact

Symbol:



Operands:

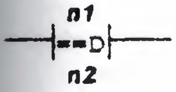
n1. n2 (signed integer word):	VW. T. C. IW. QW. MW. SMW. AC. AIW. Constant. *VD. *AC

Description of operation:

The Compare Integer Less Than or Equal Contact is closed when the signed integer word value stored at address n1 is less than or equal to the signed integer word value stored at address n2 . Power flows through the contact when closed.

Compare Double Integer Equal Contact

Symbol:



Operands:

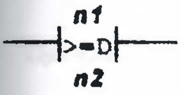
n1. n2 (signed integer double word):	VD. ID. QD. MD. SMD. AC. HC. Constant. *VD. *AC

Description of operation:

The Compare Double Integer Equal Contact is closed when the double word value stored at address n1 is equal to the double word value stored at address n2 . Power flows through the contact when closed.

Compare Double Integer Greater Than Or Equal Contact

Symbol:



Operands:

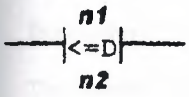
n1. n2 (signed integer double word):	VD, ID, QD. MD. SMD.AC HC. Constant, *VD, *AC

Description of operation:

Compare Double Integer Greater Than Or Equal Contact is closed when the double word value stored at address n1 is greater than or equal to the double word value stored at address n2 . Power flows through the contact when closed.

Compare Double Integer Less Than Or Equal Contact

Symbol:



Operands:

n1. n2 (signed integer double word):	VD, ID, QD. MD. SMD.AC. HC. Constant. *VD. *AC

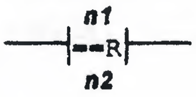
Description of operation:

The Compare Double Integer Less Than Or Equal Contact is closed when the double word value stored at address n1 is less than or equal to the double word value stored at address n2 . Power flows through the contact when closed.

Compare Real Equal Contact

Note: CPU 214 only.

Symbol:



Operands:

n1. n2 (real):	VD. ID, QD. MD. SMD. AC. HC. Constant, *VD. *AC

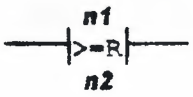
Description of operation:

The Compare Real Equal Contact is closed when the real value stored at address n1 is equal to the real value stored at address n2 . Power flows through the contact when closed.

Compare Real Greater Than Or Equal Contact

Note: CPU 214 only.

Symbol:



Operands:

n1. n2 (Dword):	VD. ID. QD. MD. SMD. AC. HC. Constant. *VD. *AC

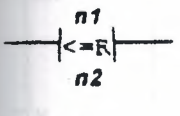
Description of operation:

Compare Real Greater Than Or Equal Contact is closed when the real value stored at address n1 is greater than or equal to the real value stored at address n2 . Power flows through the contact when closed.

Compare Real Less Than Or Equal Contact

Note: CPU 214 only.

Symbol:



Operands:

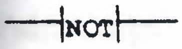
n1, n2 (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

Description of operation:

The Compare Real Less Than Or Equal Contact is closed when the real value stored at address n1 is less than or equal to the real value stored at address n2 . Power flows through the contact when closed.

Invert Power Flow Contact

Symbol:



Operands:

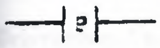
(none)

Description of operation:

The NOT (Invert Power Flow) contact changes the state of power flow. If power flow reaches the Not contact, then it stops. When power flow does not reach the Not contact, it sources power flow.

Positive Transition Contact

Symbol:



Operands:

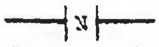
(none)

Description of operation:

The Positive Transition Contact allows power to flow for one scan, for each off-to-on transition

Negative Transition Contact

Symbol:



Operands:

(none)

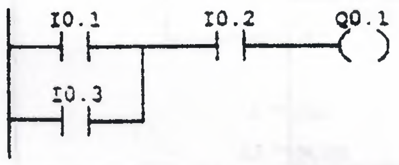
Description of operation:

The Negative Transition Contact allows power to flow for one scan, for each on-to-off transition .

Ladder Contact Examples

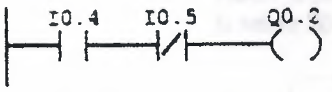
Network 1

When I0.1 or I0.3 is on and I0.2 is on then output Q0.1 is turned on.



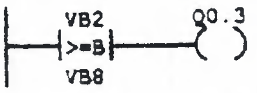
Network 2

When I0.4 is on and I0.5 is not on, then output Q0.2 is turned on.



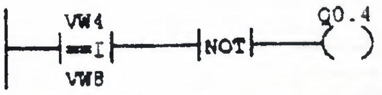
Network 3

When VB2 is greater than or equal to VB8, then output Q0.3 is turned on



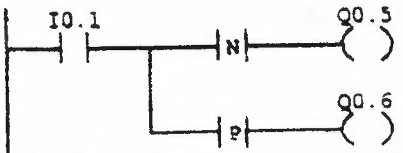
Network 4

When VB4 equals VB8, then output Q0.4 is turned off (*Note: The NOT instruction can be used to create a Not Equal comparison.*)



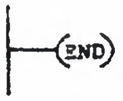
Network 5

When I0.1 transitions from on to off, then output Q0.5 is turned on for one scan cycle. When I0.1 transitions from off to on, then Q0.6 is turned on for one scan.



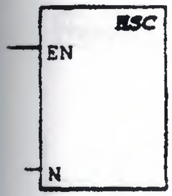
Network 6

End of the main user program.



High Speed Counter

Symbol:



Operands:

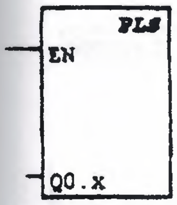
N (word):	CPU 212: 0 CPU 214: 0-2

Description of operation:

When the High-speed Counter (HSC) box is enabled, the state of the HSC special memory bits are examined. The HSC operation defined by the special memory bits is then invoked. The parameter N specifies the High-speed Counter number.

Pulse Output

Symbol:



Operands:

Q0.x (word):	CPU 214: 0-1

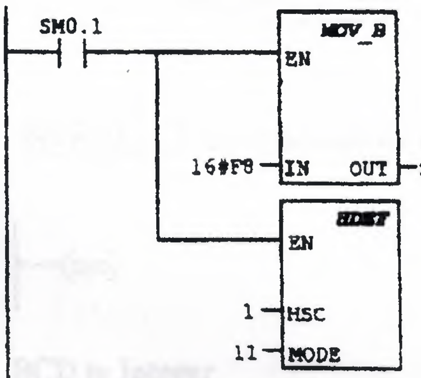
Description of operation:

The Pulse Output (PLS) box examines the special memory bits for that pulse output (Q0.x). The pulse operation defined by the special memory bits is then invoked.

Ladder High-speed Operation Instruction Examples

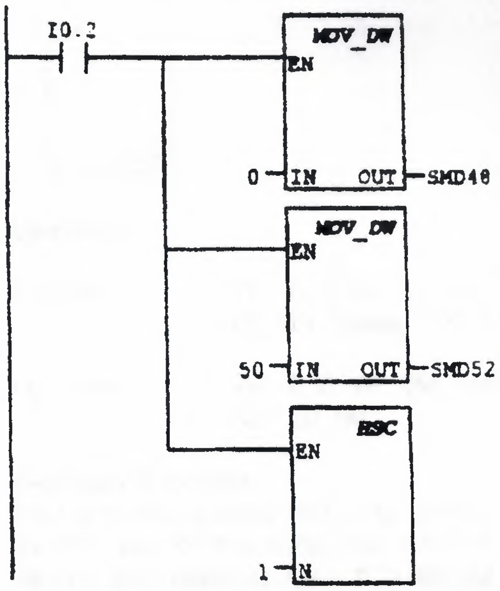
Network 1

On the first scan, the counter is enabled. Initial direction is set to count up. Start and reset inputs are set to active high. 4x mode is set.



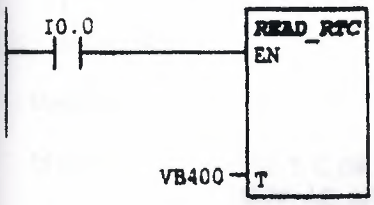
Network 2

When I0.2 is on, the current value of HSC1 is cleared and its preset value is set to 50 .

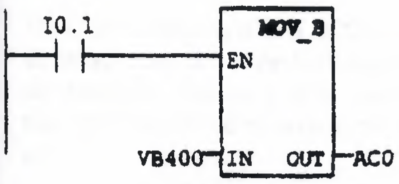


Real-time Clock Instruction Examples

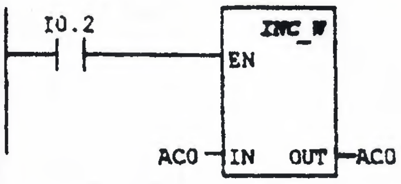
Network 1 When I0.0 is on, the clock is read and the value is stored in the buffer, starting at VB400.



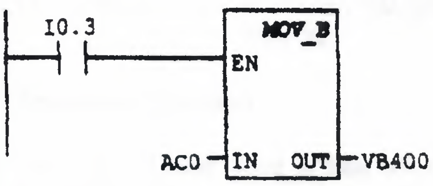
Network 2 When I0.1 is on, the year value (95) from the first byte of VB400 is moved to AC0 .



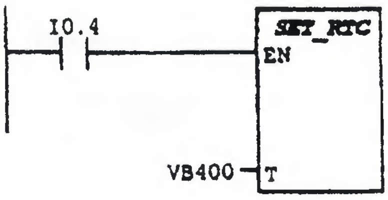
Network 3 When I0.2 is on, the year value in AC0 is incremented by 1.



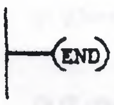
Network 4 When I0.3 is on, the new year value (96) is stored in VB400 .



Network 5 When I0.4 is on, the new year value is written to the clock.

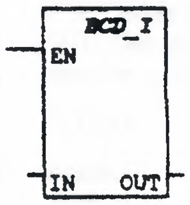


Network 6 End of the main user program.



BCD to Integer

Symbol:



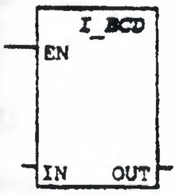
Operands:

IN (word):	VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:
The Convert BCD to Integer (BCD_I) box converts the BCD value (IN) to an integer value (OUT). If the input value contains an invalid BCD digit, the BCD/BIN memory bit (SM1.6) is set.

Integer to BCD

Symbol:



Operands:

IN (word):	VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

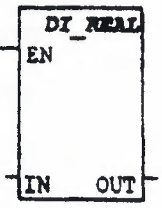
Description of operation:

The Convert Integer to BCD (I_BCD) box converts the integer value (IN) to the BCD value (OUT). If the conversion produces a BCD number greater than 9999, the BCD/BIN memory bit (SM1.6) is set.

Integer Double Word to Real

Note: CPU 214 only.

Symbol:



Operands:

IN (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, MD, SMD, AC, *VD, *AC

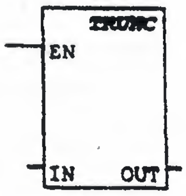
Description of operation:

The Integer Double Word to Real (DI_REAL) instruction converts a 32-bit, signed integer (IN) into a 32-bit real number (OUT).

Truncate

Note: CPU 214 only.

Symbol:



Operands:

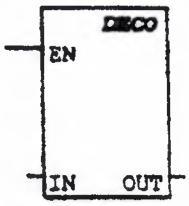
IN (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The Truncate (TRUNC) instruction converts a 32-bit real number (IN) into a 32-bit signed integer (OUT). Only the whole number portion of the real number is converted (round-to-zero).

Decode

Symbol:



Operands:

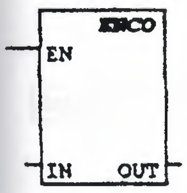
IN (byte):	VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AC, AQW, *VD, *AC

Description of operation:

The Decode (DECO) box sets the bit in the output word (OUT) that corresponds to the bit number represented by the least-significant nibble (LSN) of the input byte (IN). All other bits of the output word are set to 0.

Encode

Symbol:



Operands:

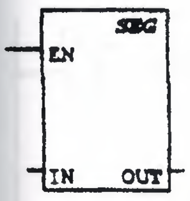
IN (word):	VW. T. C. IW. QW. MW. SMW. AC. AIW. Constant. *VD. *AC
OUT (byte)	VB. IB. QB. MB. SMB. AC. *VD. *AC

Description of operation:

The Encode (ENCO) box writes the bit number (bit #) of the least-significant bit set of the input word (IN) into the least-significant nibble (LSN) of the output byte (OUT).

Segment

Symbol:



Operands:

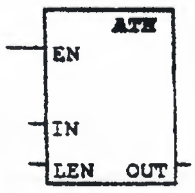
IN (byte):	VB. IB. QB. MB. SMB. AC. Constant. *VD. *AC
OUT (byte):	VB. IB. QB. MB. SMB. AC. *VD. *AC

Description of operation:

The Segment (SEG) box generates a bit pattern (OUT) that illuminates the segments of a seven-segment display. The illuminated segments represent the character in the least-significant digit of the input byte (IN).

ASCII to Hex

Symbol:



Operands:

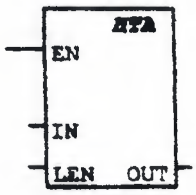
LEN (byte):	VB. IB. QB. MB. SMB. AC. Constant. *VD. *AC
IN (byte):	VB. IB. QB. MB. SMB. *VD. *AC
OUT (byte):	VB. IB. QB. MB. SMB. *VD. *AC

Description of operation:
The ASCII to HEX (ATH) box converts the ASCII string of length LEN, starting with the character IN, to hexadecimal digits starting at the location OUT. The maximum length of the ASCII string is 255 characters.

Legal ASCII characters are the hexadecimal values 30-39, and 41-46. If an illegal ASCII character is encountered, the conversion is terminated, and the NOT_ASCII memory bit (SM1.7) is set.

Hex to ASCII

Symbol:



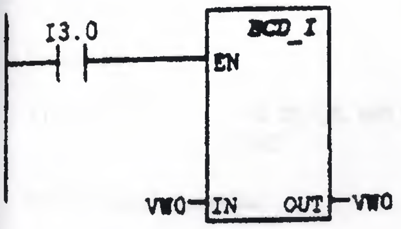
Operands:

LEN (byte):	VB. IB. QB. MB. SMB. AC. Constant. *VD. *AC
IN (byte):	VB. IB. QB. MB. SMB. *VD. *AC
OUT (byte):	VB. IB. QB. MB. SMB. *VD. *AC

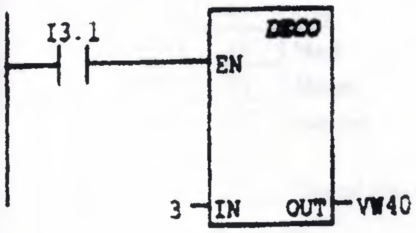
Description of operation:
The HEX to ASCII (HTA) box converts the hexadecimal digits, starting with the input byte IN, to an ASCII string starting at the location OUT. The number of hexadecimal digits to be converted is specified by length LEN. The maximum number of the hexadecimal digits that can be converted is 255.

Ladder Conversion Instruction Examples

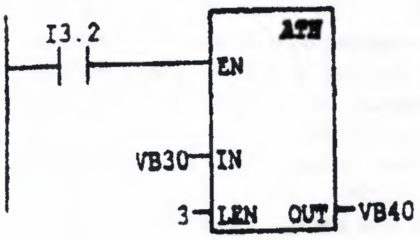
Network 1 When I3.0 is on, the Binary Coded Decimal value in VW0 is converted to an integer value.



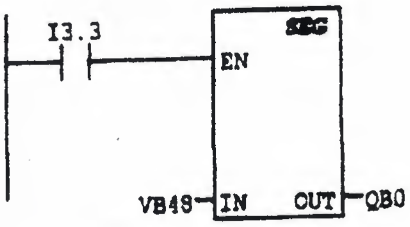
Network 2 When I3.1 is on, 3 is decoded and the corresponding bit of VW40 is set.



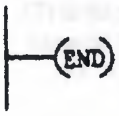
Network 3 When I3.2 is on, the 3-character ASCII string starting with the character at VB30 is converted to hexadecimal digits starting at VB40.



Network 4 When I3.3 is on, a bit pattern is generated at QB0 that illuminates the segments of the character represented by VB48.

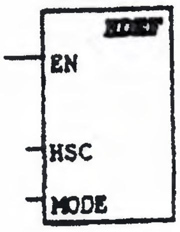


Network 5 End of the main user program.



HSC Definition

Symbol:



Operands:

HSC (byte):	CPU 212: 0 CPU 214: 0-2
MODE (byte):	CPU 212: 0 CPU 214: 0 (HSC0), 0-11 (HSC1-2)

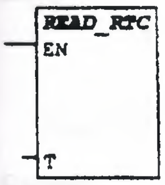
Description of operation:

When the High-speed Counter Definition (HDEF) box is enabled, the referenced counter (HSC) is assigned a high-speed counter type or MODE. Only one HDEF box may be used per counter.

Read Real Time Clock

Note: Real Time Clock instructions are supported by the CPU 214 only.

Symbol:



Operands:

T (byte):	VB, IB, QB, MB, SMB, *VD, *AC

Description of operation:

The Read Real Time Clock (READ_RTC) box reads the current time and date from the clock and loads it in an 8-byte buffer (T).

Example Memory Data Starting at VB400:
READ_RTC (Clock is read)

VB400	95	Year
VB401	03	Month
VB402	24	Day
VB403	08	Hour
VB404	00	Minute
VB405	00	Second
VB406	00	
VB407	06	Day of Week

24-Mar-95
8:00:00
Friday

Note:
The time of day clock initializes the following date and time after extended power outages or memory has been lost:

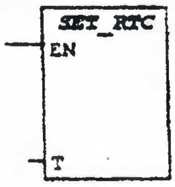
Date:	01-Jan-90
Time:	00:00:00
Day of Week	Sunday

Note:
Do not use the READ_RTC / SET_RTC instructions in both the main program and in an interrupt routine. If you do this and the clock instruction is executing when the the interrupt that also executes the clock instruction occurs, then the clock instruction in the interrupt routine is not executed. SM4.5 is then set, indicating that two simultaneous accesses to the clock were attempted.

Set Real Time Clock

Note: Real Time Clock instructions are supported by the CPU 214 only.

Symbol:



Operands:

T (byte):	VB, IB, QB, MB, SMB, *VD, *AC

Description of operation:

The Set Real Time Clock (SET_RTC) box writes the current time and date loaded in an 8-byte buffer (T) to the clock.

Example Memory Data Starting at VB400:
SET_RTC (New value is written to clock)

VB400	96	Year
VB401	03	Month
VB402	24	Day
VB403	08	Hour
VB404	00	Minute
VB405	00	Second
VB406	00	
VB407	06	Day of Week

24-Mar-96
8:00:00
Friday

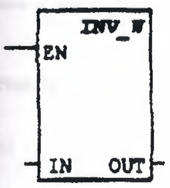
Note:
The time of day clock initializes the following date and time after extended power outages or memory has been lost:

Date:	01-Jan-90
Time:	00:00:00
Day of Week	Sunday

Note:
Do not use the READ_RTC / SET_RTC instructions in both the main program and in an interrupt routine. If you do this and the clock instruction is executing when the the interrupt that also executes the clock instruction occurs, then the clock instruction in the interrupt routine is not executed. SM4.5 is then set, indicating that two simultaneous accesses to the clock were attempted.

Invert Word

Symbol:



Operands:

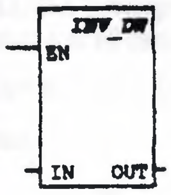
IN (word):	VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:

The Invert Word (INV_W) box takes the ones complement of the input word value (IN) and loads the result in a word value (OUT).

Invert Double Word

Symbol:



Operands:

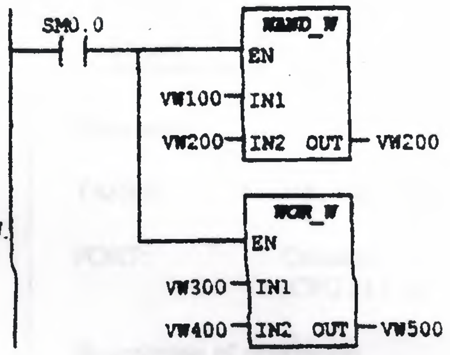
IN (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

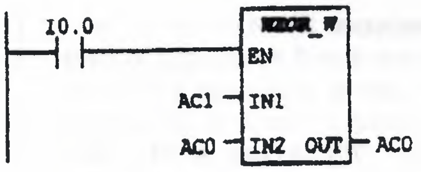
The Invert Double Word (INV_DW) box takes the ones complement of the input double word value (IN) and loads the result in a double word value (OUT).

Ladder Logical Operations Examples

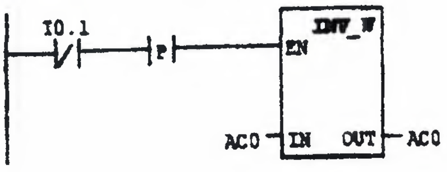
Network 1 Every scan, AND VW100 and VW200 together and store the result in VW200. Also, OR VW300 and VW400 together and store the result in VW500.



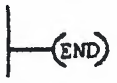
Network 2 When I0.0 is on, "XOR" AC1 and AC0 together and store the result in AC0.



Network 3 When I0.1 transitions from off to on, invert AC0 (ones complement) and store it in AC0.



Network 4 End of main user program.



Enable Interrupts

Symbol:

```
———(ENI)
```

Operands:

(none)

Description:

The Enable Interrupts (ENI) coil globally enables processing of all attached interrupt events.

Disable Interrupts

Symbol:

```
———(DISI)
```

Operands:

(none)

Description:

The Disable Interrupts (DISI) coil globally disables processing of all interrupt events.

Return from Interrupts

Symbol:

```
———(RETI)
```

Conditional Return from Interrupts

```
|———(RETI)
```

Unconditional Return from Interrupts

Operands:

(none)

Description:

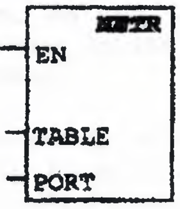
The Conditional Return from Interrupts (RETI) coil returns from an interrupt based upon the condition of the preceding logic.

The Unconditional Return from Interrupts (RETI) coil must be used to terminate each interrupt routine.

Network Read

Note: CPU 214 only.

Symbol:



Operands:

TABLE:	VB, MB, *VD, *AC
PORT:	Constant (CPU 214: 0)

Description of operation:

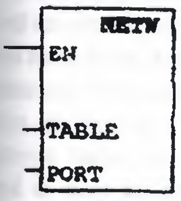
The Network Read (NETR) instruction initiates a communication operation to gather data from a remote device through the specified port (PORT), as defined in the description table (TABLE).

You can use the NETR instruction to read up to 16 bytes of information from a remote station, and use the NETW instruction to write up to 16 bytes of information to a remote station. A maximum of eight NETR and NETW instructions may be activated at any one time. For example, you can have four NETR and four NETW instructions, or two NETR and six NETW instructions.

Network Write

Note: CPU 214 only.

Symbol:



Operands:

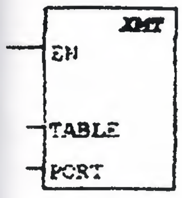
TABLE:	VB, MB, *VD, *AC
PORT:	Constant (CPU 214: 0)

Description of operation:
The Network Write (NETW) instruction initiates a communication operation to write data to a remote device through the specified port (PORT), as defined in the description table (TABLE).

You can use the NETR instruction to read up to 16 bytes of information from a remote station, and use the NETW instruction to write up to 16 bytes of information to a remote station. A maximum of eight NETR and NETW instructions may be activated at any one time. For example, you can have four NETR and four NETW instructions, or two NETR and six NETW instructions.

Transmit

Symbol:



Operands:

TABLE (byte):	VB, IB, QB, MB, SMB, *VD, *AC
PORT (byte):	0

Description of operation:
The Transmit (XMT) box invokes the transmission of the data buffer (TABLE). The first entry in the data buffer specifies the number of bytes to be transmitted. PORT specifies the communication port to be used for transmission. It must always be 0.

Data Sharing with Interrupt Events

Because interrupt events are asynchronous to the main user-program, they can occur at any point during execution of the main user-program. When the main program and an interrupt routine share data, you must understand the nature of the problems that can arise and how to avoid such problems.

Data-sharing problems can occur in situation where a sequence of operations are performed in the main program on data stored in a memory location shared by the main program and an interrupt routine. If an intermediate result is stored in the shared memory location, then an interrupt event occurring before the sequence is complete will cause the interrupt routine to be executed with invalid data, or it will corrupt an intermediate value in the main program.

The situations described above apply whether you write your programs in STL or LAD. If you write your programs in LAD, you should also be aware that many LAD instructions produce a sequence of STL instructions. If the LAD instruction is located in the main program and is operating on data stored in a shared memory location, an interrupt event can occur between the execution of the STL instructions, altering intermediate values and making it appear that the LAD instruction executed incorrectly. For techniques to avoid problems with data sharing, see Programming Techniques for Data Sharing .

Programming Techniques for Data Sharing

The following programming techniques should be followed to avoid problems with data sharing between your main program and interrupt routines. These techniques either restrict the way access is made to shared memory locations, or they make instruction sequences using shared memory locations uninterruptible. The appropriate technique depends upon the size of the data being shared (simple elements such as a byte, word, or double-word variable or complex elements such as multiple variables) and the programming language (STL or LAD).

If the shared data is a single byte, word, or double-word variable and your program is written in STL, then make sure that intermediate or temporary values are not stored in shared memory locations. A shared location should be accessed in the main program only as the initial source value or the final destination value in a sequence of operations

If the shared data is a single byte, word, or double-word variable and your program is written in LAD, then access shared memory locations using a Move instruction. If the main program performs one or more operations on a data value provided by an interrupt routine, the Move instruction must be used to move the data value from the shared memory location to a non-shared memory location or to an accumulator. If the main program performs one or more operations on data in order to provide a value to an interrupt routine, then the last operation must be a Move instruction that moves the data value from an accumulator or non-shared memory location to the shared memory location. Other instructions in the sequence must not directly access the shared memory location.

If the shared data is composed of related bytes, words, or double-words whose values must agree; for example, the pressure and temperature of a gas in a tank, then the interrupt disable/enable instructions, DISI and ENI, must be used to control interrupt routine execution. At the point in your main program (STL or LAD) where operations on shared memory locations are to begin, interrupts must be disabled. Once all actions affecting shared locations are complete, interrupts must be re-enabled. During the time that interrupts are disabled, interrupt routines cannot execute and access shared memory locations.

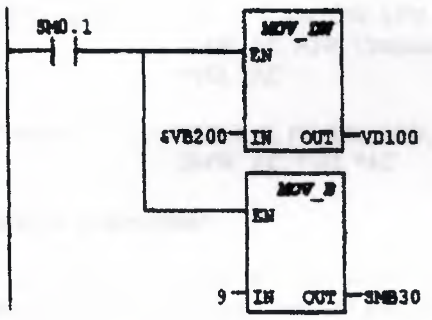
Interrupt Event Priority Table

Interrupt Description (By group priority)	Event #	**In-Group Priority**	**Supported in CPU 21**
Comm. (Highest Priority)			
Receive interrupt	8	0	Y
Transmit complete interrupt	9	0*	Y
Discrete (Middle Priority)			
Rising edge, I0.0**	0	0	Y
Rising edge, I0.1	2	1	
Rising edge, I0.2	4	2	
Rising edge, I0.3	6	3	
Falling edge, I0.0**	1	4	Y
Falling edge, I0.1	3	5	
Falling edge, I0.2	5	6	
Falling edge, I0.3	7	7	
HSC0 CV=PV** *(current value = preset value)*	12	0	Y
HSC1 CV=PV *(current value = preset value)*	13	8	
HSC1 direction input changed	14	9	
HSC1 external reset	15	10	
HSC2 CV=PV *(current value = preset value)*	16	11	
HSC2 direction input changed	17	12	
HSC2 external reset	18	13	
PLS0 pulse count complete interrupt	19	14	
PLS1 pulse count complete interrupt	20	15	
Timed (Lowest Priority)			
Timed interrupt 0	10	0	Y
Timed interrupt 1	11	1	

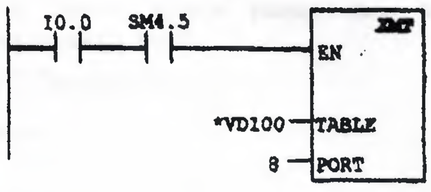
* Since communication is inherently half-duplex, both transmit and receive are the same priority.
**If event 12 (HSC0 CV=PV) is attached to an interrupt, then neither event 0 nor event 1 can be attached to interrupts. Likewise, if either event 0 or 1 is attached to an interrupt, then event 12 cannot be attached to an interrupt.

Ladder Interrupt / Communication Instruction Examples

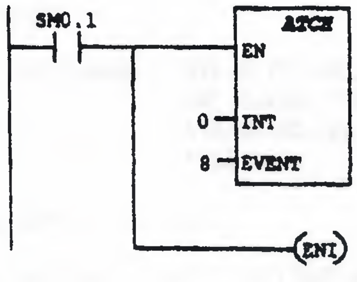
Network 1 On the first scan, create a pointer to the data to be transmitted. Select freeport mode, 9600 baud, no parity, 8 bits per character. SMB30 is the freeport control byte.



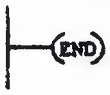
Network 2 When I0.0 and SM4.5 are both on, the message in the buffer (pointed to by VD100) is transmitted. SM4.5 is on when the transmitter is idle.



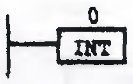
Network 3 Assign receive interrupt event 8 to interrupt routine 0, and enable the routine.



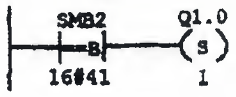
Network 4 End of main ladder program.



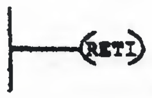
Network 5 Begin interrupt routine 0 .



Network 6 Compare received character in special memory byte SMB2 with capital letter "A" If character is "A", Q0.1 is set.
6



Network 7 Return from interrupt to main program.



Horizontal Lines

In ladder logic, horizontal lines represent wires connecting elements in series.

All lines in a network must be connected to valid elements.
All networks must terminate in a coil or a box.

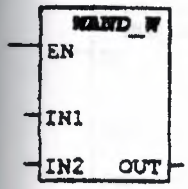
Vertical Lines

In ladder logic, vertical lines represent wires connecting to parallel branches.

All lines in a network must be connected to valid elements.
All networks must terminate in a coil or a box.

AND Word

Symbol:



Operands:

IN1, IN2 (word):	VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

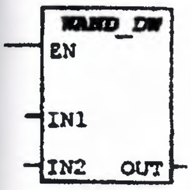
Description of operation:

The AND Word (WAND_W) box ANDs the corresponding bits of the input words IN1 and IN2, and loads the result (OUT) in a word.

Note:
When IN1 ≠ OUT and IN2 ≠ OUT:
- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

AND Double Word

Symbol:



Operands:

IN1, IN2 (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, MD, SMD, AC, *VD, *AC

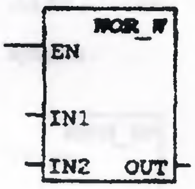
Description of operation:

The AND Double Word (WAND_DW) box ANDs the corresponding bits of the input double words IN1 and IN2, and loads the result (OUT) in a double word.

Note:
When IN1 ≠ OUT and IN2 ≠ OUT:
- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

OR Word

Symbol:



Operands:

IN1, IN2 (word):	VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

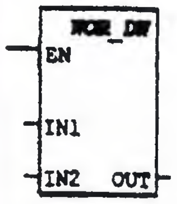
Description of operation:

The OR Word (WOR_W) box ORs the corresponding bits of the input words IN1 and IN2, and loads the result (OUT) in a word.

Note:
When IN1 ≠ OUT and IN2 ≠ OUT:
- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

OR Double Word

Symbol:



Operands:

IN1, IN2 (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, MD, SMD, AC, *VD, *AC

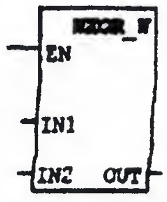
Description of operation:

The OR Double Word (WOR_DW) box ORs the corresponding bits of the input double words IN1 and IN2, and loads the result (OUT) in a double word.

Note:
When IN1 ≠ OUT and IN2 ≠ OUT:
- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

XOR Word

Symbol:



Operands:

IN1, IN2 (word):	VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

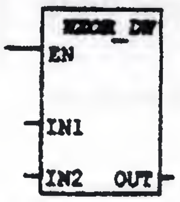
Description of operation:

The Exclusive OR Word (WXOR_W) box XORs the corresponding bits of the input words IN1 and IN2, and loads the result (OUT) in a word.

Note:
When IN1 ≠ OUT and IN2 ≠ OUT:
- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

XOR Double Word

Symbol:



Operands:

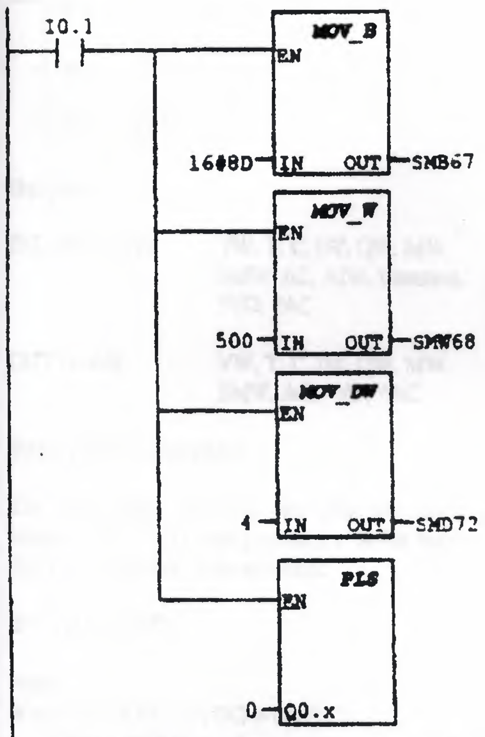
IN1, IN2 (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

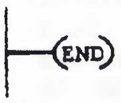
The Exclusive OR Double Word (WXOR_DW) box XORs the corresponding bits of the input double words IN1 and IN2, and loads the result (OUT) in a double word.

Note:
When *IN1 ≠ OUT* and *IN2 ≠ OUT*:
- If IN2 and OUT are direct-addressed operands, and if *OUT* contains one of the bytes of IN2, then the instruction is invalid.
- *If IN2* is an *indirect* address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Network 3 — When I0.1 is on, the Pulse Train Output control byte is set up, and the PTO operation is invoked: cycle time 500ms, pulse count 4, PLS 0 --> Q0.0 .

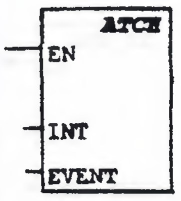


Network 4 — End of the main user program.



Attach Interrupts

Symbol:



Operands:

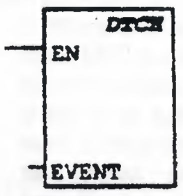
INT (byte):	CPU 212: 0-31 CPU 214: 0-127
EVENT (byte):	CPU 212: 0, 1, 8-10, 12 CPU 214: 0-20

Description of operation:

The Attach Interrupts (ATCH) box associates an interrupt event (EVENT) with an interrupt routine number (INT), and enables the interrupt event.

Detach Interrupts

Symbol:



Operands:

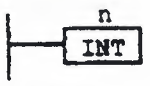
EVENT (byte):	CPU 212: 0, 1, 8-10, 12 CPU 214: 0-20

Description of operation:

The Detach Interrupts (DTCH) box disassociates an interrupt event (EVENT) from all interrupt routines, and disables the interrupt event.

Interrupt Routine

Symbol:



Operands:

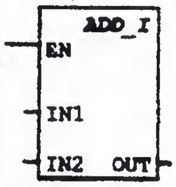
n (word):	CPU 212: 0-31 CPU 214: 0-127

Description of operation:

The Interrupt Routine (INT) label marks the beginning of the interrupt routine (n). The maximum number of interrupts supported by the CPU 212 is 32, and by the CPU 214, 128.

Add Integer

Symbol:



Operands:

IN1, IN2 (word):	VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:

The Add Integer (ADD_I) box adds two 16-bit integers (IN1, IN2), and produces a 16-bit result (OUT), as is shown in the equation:

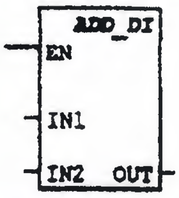
IN1 + IN2 = OUT

Note:
When IN1 ≠ OUT and IN2 ≠ OUT:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Add Double Integer

Symbol:



Operands:

IN1, IN2 (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The Add Double Integer (ADD_DI) box adds two 32-bit integers (IN1, IN2), and produces a 32-bit result (OUT), as is shown in the equation:

IN1 + IN2 = OUT

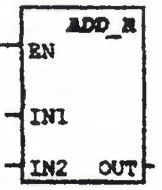
Note:
When IN1 ≠ OUT and IN2 ≠ OUT:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Add Real

Note: CPU 214 only.

Symbol:



Operands:

IN1, IN2 (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, SMD, AC, *VD, *AC

Description of operation:

The Add Real (ADD_R) box adds two 32-bit real numbers (IN1, IN2), and produces a 32-bit real number result (OUT), as is shown in the equation:

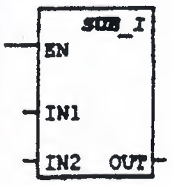
IN1 + IN2 = OUT

Note:
When IN1 ≠ OUT and IN2 ≠ OUT:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Subtract Integer

Symbol:



Operands:

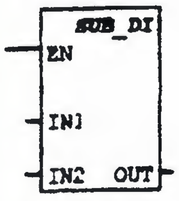
IN1, IN2 (word):	VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:
The Subtract Integer (SUB_I) box subtracts two 16-bit integers (IN1, IN2), and produces a 16-bit result (OUT), as is shown in the equation:
IN1 - IN2 = OUT
Note:
When IN1 ≠ OUT and IN2 ≠ OUT:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Subtract Double Integer

Symbol:



Operands:

IN1, IN2 (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The Subtract Double Integer (SUB_DI) box subtracts two 32-bit integers (IN1, IN2), and produces a 32-bit result (OUT), as is shown in the equation:

IN1 - IN2 = OUT

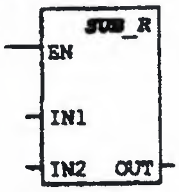
Note:
When IN1 ≠ OUT and IN2 ≠ OUT:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Subtract Real

Note: *CPU 214 only.*

Symbol:



Operands:

IN1, IN2 (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, SMD, AC, *VD, *AC

Description of operation:
The Sutract Real (SUB_R) box subtracts two 32-bit real numbers (IN1, IN2), and produces a 32-bit real number result (OUT), as is shown in the equation:

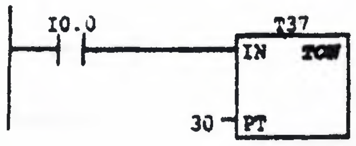
IN1 - IN2 = OUT

Note:
When IN1 ≠ OUT and IN2 ≠ OUT:

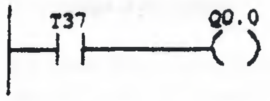
- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Ladder Timer / Counter Examples

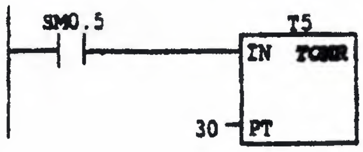
Network 1 When I0.0 is on then the timer will start. After 3 seconds (30 X 100ms) T37 bit will come on.



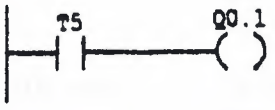
Network 2 When Timer 37 reaches its preset, turn on Q0.0 .



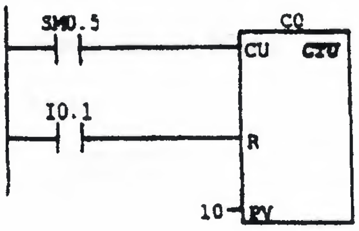
Network 3 When SM0.5 (1 sec. clock pulse, .5 sec. on and .5 sec. off) is ON, then the timer will time. The T5 bit will come on after 6 seconds.



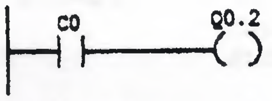
Network 4 When Timer 5 reaches its preset, turn on Q0.1 .



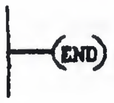
Network 5 By using SM0.5 (1 sec. clock pulse) the counter will count pulses and turn on the C0 bit when a count of 10 is reached. I0.0 resets the counter.



Network 6 When C0 reaches its preset, turn on Q0.2

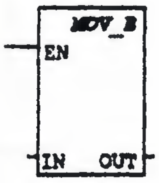


Network 7 End of the main user program.



Move Byte

Symbol:



Operands:

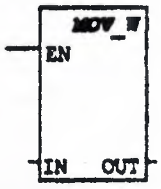
IN (byte):	VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC
OUT (byte):	VB, IB, QB, MB, SMB, AC, *VD, *AC

Description of operation:

The Move Byte (MOV_B) box moves the input byte (IN) to the output byte (OUT). The input byte is not altered by the move.

Move Word

Symbol:



Operands:

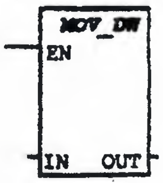
IN (word):	VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AC, AQW, *VD, *AC

Description of operation:

The Move Word (MOV_W) box moves the input word (IN) to the output word (OUT). The input word is not altered by the move.

Move Double Word

Symbol:



Operands:

IN (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC, &VB, &IB, &QB, &MB, &T, &C
OUT (Dword):	VD, ID, QD, MD, SMD, AC, *VD, *AC

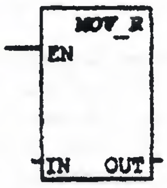
Description of operation:

The Move Double Word (MOV_DW) box moves the input double word (IN) to the output double word (OUT). The input double word is not altered by the move.

Move Real

Note: CPU 214 only.

Symbol:



Operands:

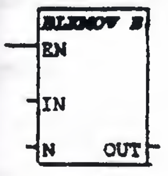
IN (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The Move Real (MOV_R) box moves a 32-bit real input double word (IN) to the output double word (OUT). The input double word is not altered by the move.

Block Move Byte

Symbol:



Operands:

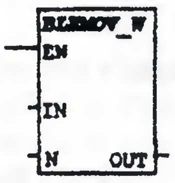
IN (byte):	VB, IB, QB, MB, SMB *VD, *AC
OUT (byte):	VB, IB, QB, MB, SMB, *VD, *AC
N (byte):	VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

Description of operation:

The Block Move Byte (BLKMOV_B) box moves the number of bytes specified (N), from the input array starting at IN, to the output array starting at OUT. N has a range of 1 to 255.

Block Move Word

Symbol:



Operands:

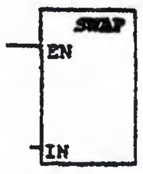
IN (word):	VW, T, C, IW, QW, MW, SMW, AIW, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AQW, *VD, *AC
N (byte):	VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

Description of operation:

The Block Move Word (BLKMOV_B) box moves the number of words specified (N), from the input array starting at IN, to the output array starting at OUT. N has a range of 1 to 255.

Swap

Symbol:



Operands:

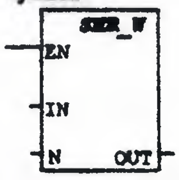
IN (word):	VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:

The Swap Byte box exchanges the most-significant byte with the least-significant byte of the word (IN).

Shift Right Word

Symbol:



Operands:

IN (word):	VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
N (byte):	VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:
The Shift Right Word (SHR_W) box shifts the word value (IN) right by the shift count (N), and loads the result in the output word (OUT).

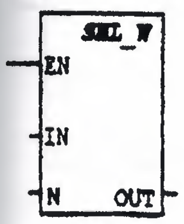
SM1.0 (zero)	= 1 if OUT = 0
SM1.1 (overflow)	= 1 if last bit shifted out = 0

Note:
When IN ≠ OUT:

- If N and OUT are direct-addressed operands, and if OUT contains N, then the instruction is invalid.
- If N is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.
- If N and OUT are indirect address pointers and the pointers are equal, then the instruction is invalid.

Shift Left Word

Symbol:



Operands:

IN (word):	VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
N (byte):	VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:
The Shift Left Word (SHL_W) box shifts the word value (IN) left by the shift count (N), and loads the result in the output word (OUT).

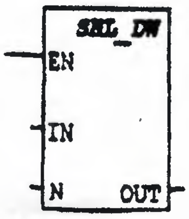
SM1.0 (zero)	= 1 if OUT = 0
SM1.1 (overflow)	= 1 if last bit shifted out = 0

Note:
When IN ≠ OUT:

- If N and OUT are direct-addressed operands, and if OUT contains N, then the instruction is invalid.
- If N is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.
- If N and OUT are indirect address pointers and the pointers are equal, then the instruction is invalid.

Shift Left Double Word

Symbol:



Operands:

IN (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC
N (byte):	VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:
The Shift Left Double Word (SHL_DW) box shifts the double word value (IN) left by the shift count (N), and loads the result in the output double word (OUT).

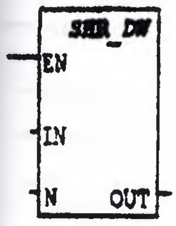
SM1.0 (zero)	= 1 if OUT = 0
SM1.1 (overflow)	= 1 if last bit shifted out = 0

Note:
When IN ≠ OUT:

- If N and OUT are direct-addressed operands, and if OUT contains N, then the instruction is invalid.
- If N is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.
- If N and OUT are indirect address pointers and the pointers are equal, then the instruction is invalid.

Shift Right Double Word

Symbol:



Operands:

IN (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC
N (byte):	VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The Shift Right Double Word (SHR_DW) box shifts the double word value (IN) right by the shift count (N), and loads the result in the output double word (OUT).

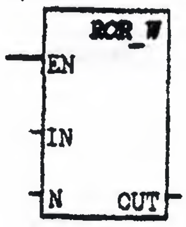
SM1.0 (zero)	= 1 if OUT = 0
SM1.1 (overflow)	= 1 if last bit shifted out = 0

Note:
When IN ≠ OUT:

- If N and OUT are direct-addressed operands, and if OUT contains N, then the instruction is invalid.
- If N is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.
- If N and OUT are indirect address pointers and the pointers are equal, then the instruction is invalid.

Rotate Right Word

Symbol:



Operands:

IN (word):	VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
N (byte):	VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

Description of operation:
The Rotate Right Word (ROR_W) box rotates the word value (IN) right by the shift count (N), and loads the result in the output word (OUT).

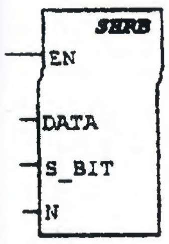
SM1.0 (zero)	= 1 if OUT = 0
SM1.1 (overflow)	= 1 if last bit rotated = 0

Note:
When IN ≠ OUT:

- If N and OUT are direct-addressed operands, and if OUT contains N, then the instruction is invalid.
- If N is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.
- If N and OUT are indirect address pointers and the pointers are equal, then the instruction is invalid.

Shift Register Bit

Symbol:



Operands:

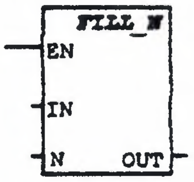
DATA, S_BIT (bit):	I, Q, M, SM, T, C, V
N (byte):	VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

Description of operation:

The Shift Register Bit (SHRB) instruction shifts the value of DATA into the shift register. S_BIT specifies the least-significant bit of the shift register. N specifies the length of the shift register and the direction of the shift (shift plus = N, shift minus = -N).

Fill Memory

Symbol:



Operands:

IN (word):	VW, T, C, IW, QW, MW, SMW, AIW, Constant, *VD, *AC
OUT (word):	VW, T, C, IW, QW, MW, SMW, AQW, *VD, *AC
N (byte):	VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

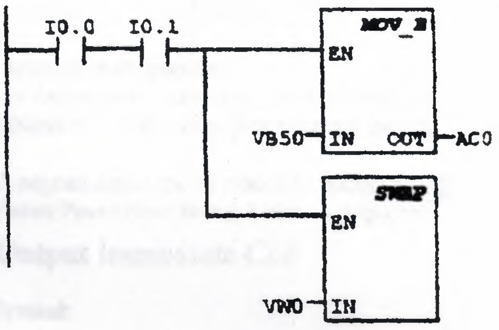
Description of operation:

The Fill Memory Box (FILL_N) fills the memory starting at the output word (OUT) with the word input pattern (IN) for the number of words specified by N. N has a range of 1 to 255.

Move / Shift / Rotate / Fill Examples

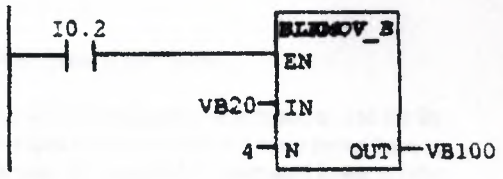
Network 1

When I0.0 and I0.1 are on then *move VB50 to AC0, and swap* the most significant byte (MSB) of VW0 with the LSB of VW0.



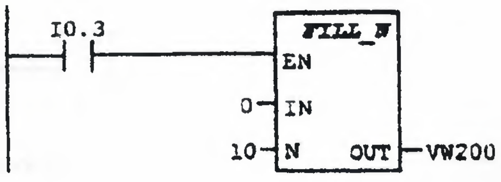
Network 2

When I0.2 is on then move VB20-VB23 toVB100-VB103.



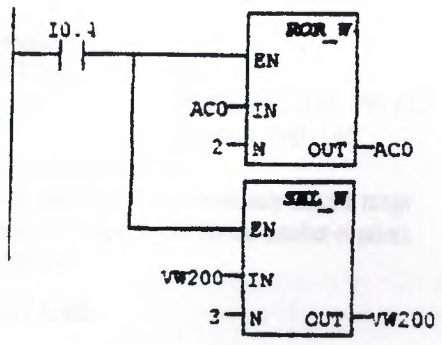
Network 3

When I0.3 is on then fill VW200-VW218 with 0's.



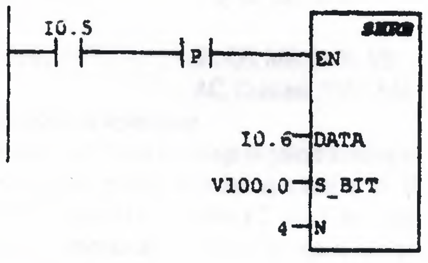
Network 4

When I0.4 is on, then the word value in AC0 is rotated right twice and stored in AC0, and the word value in VW200 is shifted left 3 times and stored in VW200.



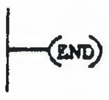
Network 5

Upon every 0 to 1 transition of I0.5, the value of I0.6 is shifted into the shift register starting at V100.0 and of length 4.



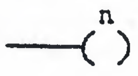
Network 6

Main end of the user program.



Output

Symbol:



Operands:

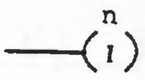
n (bit): I, Q, M, SM, T, C, V

Description of operation:
An Output coil is turned on and the Bit stored at address *n* is set to 1 when power flows to the coil.

A negated output can be created by placing a NOT (Invert Power Flow) contact before an output coil.

Output Immediate Coil

Symbol:



Operands:

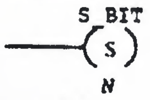
n (bit): Q

Description of operation:

An Output Immediate Coil is turned on and the Bit at output address *n* is set to 1 when power flows to the coil. An update of the addressed image register output Bit and also the corresponding physical output Bit occurs immediately after the coil is scanned without waiting for scan cycle completion.

Set

Symbol:



Operands:

S_BIT (bit): I, Q, M, SM, T, C, V

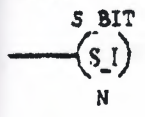
N (byte): IB, QB, MB, SMB, VB, AC, Constant, *VD, *AC

Description of operation:

The Set Coil sets the range of points starting at S_BIT for the number of points specified by N

Set Immediate Coil

Symbol:



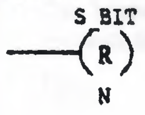
Operands:

S_BIT (bit):	Q
N (byte):	IB, QB, MB, SMB, VB, AC, Constant, *VD, *AC

Description of operation:
The Set Immediate Coil immediately sets the range of points starting at S_BIT for the number of points specified by N .

Reset Coil

Symbol:



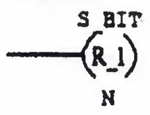
Operands:

S_BIT (bit):	I, Q, M, SM, T, C, V
N (byte):	IB, QB, MB, SMB, VB, AC, Constant, *VD, *AC

Description of operation:
The Reset Coil resets the range of points starting at S_BIT for the number of points specified by N. If S_BIT is specified to be either a T or a C bit, then both the timer/counter bit and the timer/counter current value are reset.

Reset Immediate Coil

Symbol:



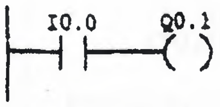
Operands:

S_BIT (bit):	Q
N (byte):	IB, QB, MB, SMB, VB, AC, Constant, *VD, *AC

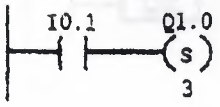
Description of operation:
The Reset Immediate Coil immediately resets the range of points starting at S_BIT for the number of points specified by N .

Ladder Output Coil Examples

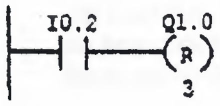
Network 1 — When I0.0 is on, then output Q0.1 is turned on.



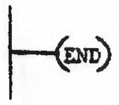
Network 2 — When I0.1 is on, then outputs Q1.0, Q1.1 and Q1.2 are set (turned on). These outputs will remain on, even if I0.1 is turned off, until they are reset.



Network 3 — When I0.2 is turned on, then outputs Q1.0, Q1.1 and Q1.2 are reset (turned off

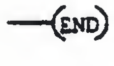


Network 4 — End of the main user program.

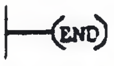


End

Symbols:



Conditional End



Unconditional End

Operands:

(none)

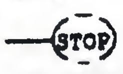
Description of operation:

The Conditional End coil terminates the main user program based on the condition of the preceding logic.

The Unconditional End coil must be used to terminate the user program.

Stop

Symbol:



Operands:

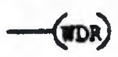
(none)

Description of operation:

The Stop coil terminates execution of the user program by causing a transition to the stop mode.

Watchdog Reset

Symbol:



Operands:

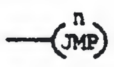
(none)

Description of operation:

The Watchdog Reset (WDR) coil allows the watchdog timer to be retriggered. This extends the time the scan takes without getting a watchdog error.

Jump

Symbol:



Operands:

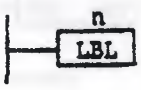
n:	CPU 212: 0-63 CPU 214: 0-255

Description of operation:

The Jump to Label (JMP) coil performs a branch to the specified label (*n*) within the program.

Label

Symbol:



Operands:

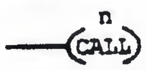
n:	CPU 212: 0-63 CPU 214: 0-255

Description of operation:

The Label (LBL) instruction marks the location of the jump destination (*n*). The CPU 212 allows 64 labels, and the CPU 214 allows 256.

Call

Symbol:



Operands:

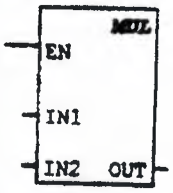
n:	CPU 212: 0-15 CPU 214: 0-63

Description of operation:

The Subroutine Call (CALL) coil transfers control to the subroutine (n).

Multiply Integer

Symbol:



Operands:

IN1, IN2 (word):	VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:
The Multiply Integer (MUL) box multiplies two 16-bit integers (IN1, IN2), and produces a 32-bit result (OUT), as is shown in the equation:

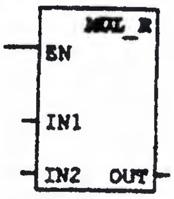
IN1 * IN2 = OUT

Note:
Some overlapping input and output operands are invalid.

Multiply Real

Note: CPU 214 only.

Symbol:



Operands:

IN1, IN2 (Dword):	VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, SMD, AC, *VD, *AC

Description of operation:
The Multiply Real (MUL_R) box multiplies two 32-bit real numbers (IN1, IN2), and produces a 32-bit real number result (OUT), as is shown in the equation:

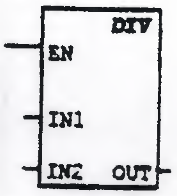
IN1 * IN2 = OUT

Note:
When IN1 ≠ OUT and IN2 ≠ OUT:

- If IN2 and OUT are direct-addressed operands, and if OUT contains one of the bytes of IN2, then the instruction is invalid.
- If IN2 is an indirect address and OUT is a direct address containing one of the bytes of the indirect address pointer, then the instruction is invalid.

Divide Integer

Symbol:



Operands:

IN1, IN2 (word):	VW, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC
OUT (Dword):	VD, ID, QD, MD, SMD, AC, *VD, *AC

Description of operation:

The Divide Integer (DIV) box divides two 16-bit integers (IN1, IN2), and produces a 32-bit result (OUT) composed of of a 16-bit quotient and a 16-bit remainder, as is shown in the equation:

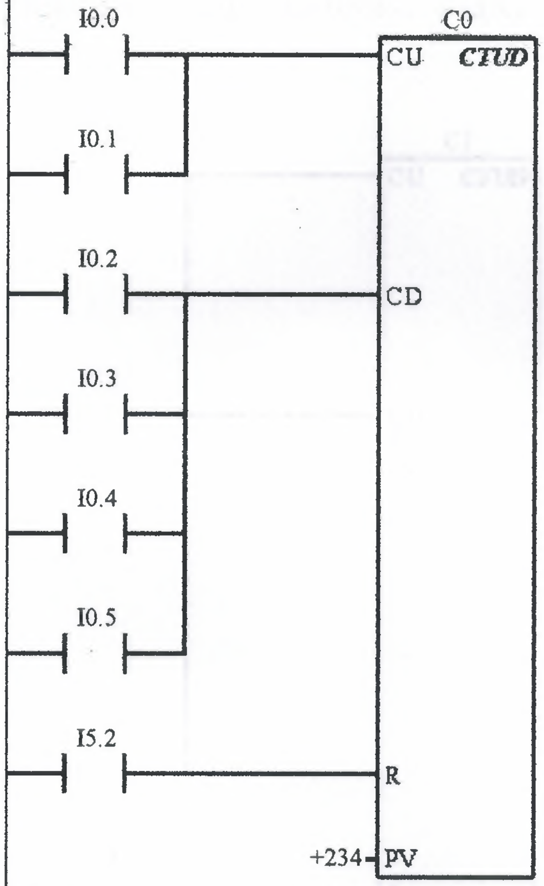
IN1 / IN2 = OUT

Notes:

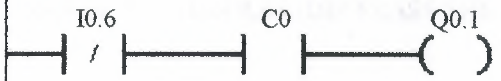
- Some overlapping input and output operands are invalid.
- The 32-bit result (OUT) cannot be the same as the second input (IN2).

PROGRAM TITLE COMMENTS

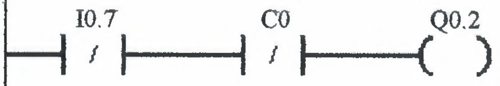
Network 1 COUNT ENTERING OR LEAVING PEOPLE IN BIG HALL (1 TO 234)



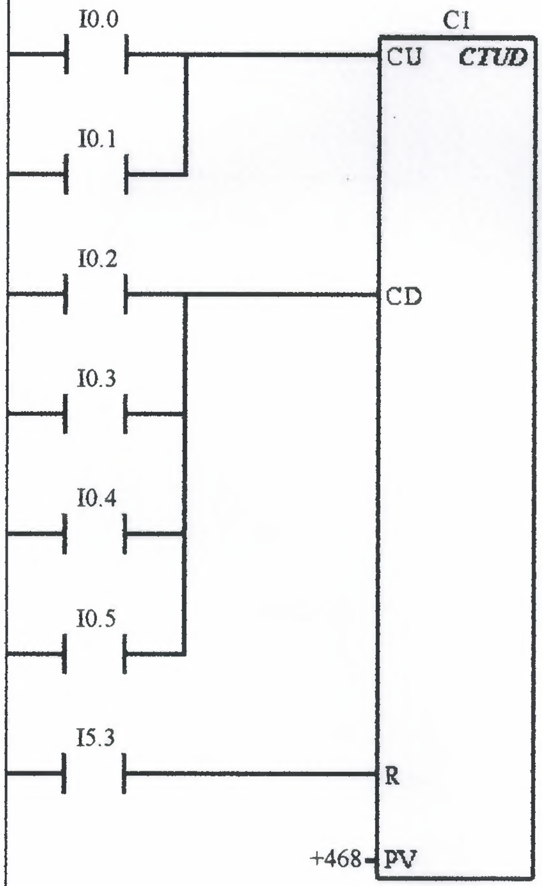
Network 2 COOLER.(DECREASE THE 1 DEGREE CELCIUS)



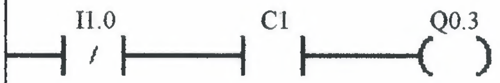
Network 3 HEATER (INCERASE THE 1 DEGREE CELCIUS)



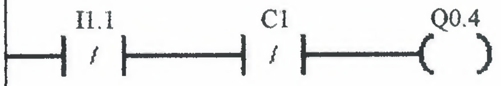
Network 4 COUNT ENTERING OR LEAVING PEOPLE IN BIG HALL (234 TO 468)



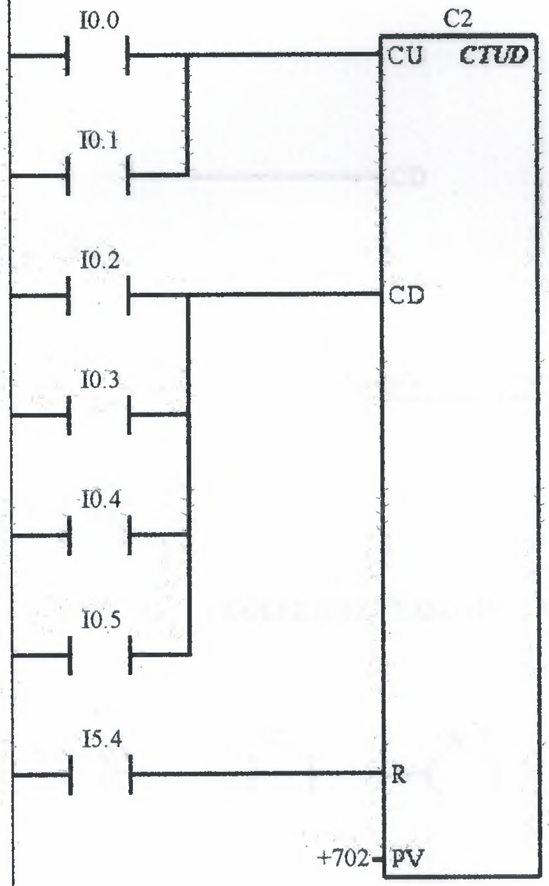
Network 5 COOLER.(DECREASE THE 1 DEGREE CELCIUS)



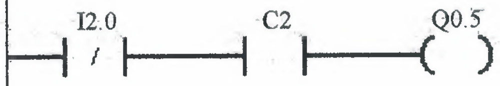
Network 6 HEATER (INCERASE THE 1 DEGREE CELCIUS)



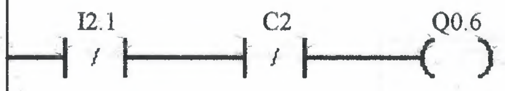
Network 7 COUNT ENTERING OR LEAVING PEOPLE IN BIG HALL (468 TO 702)



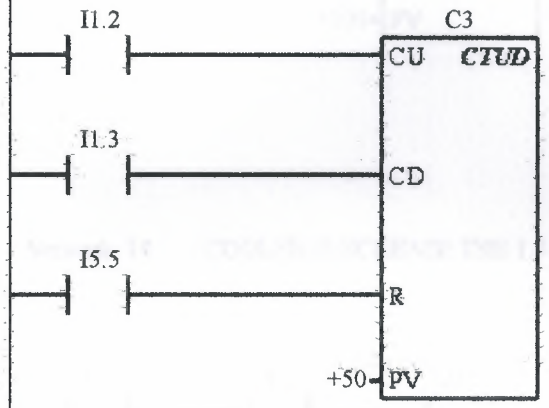
Network 8 COOLER (DECREASE THE 1 DEGREE CELCIUS)



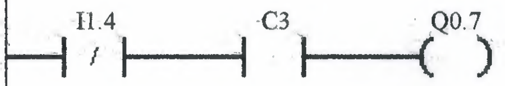
Network 9 HEATER (INCERASE THE 1 DEGREE CELCIUS)



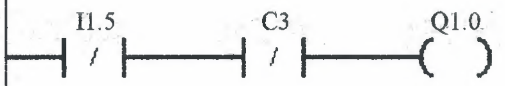
Network 10 COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 1 (0 TO 50)



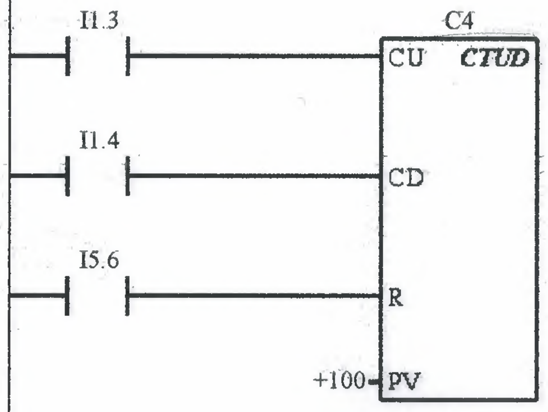
Network 11 COOLER (DECREASE THE 1.5 DEGREE CELCIUS)



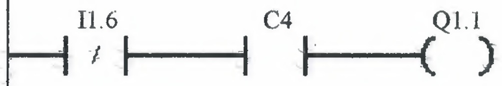
Network 12 HEATER (INCERASE THE 1.5 DEGREE CELCIUS)



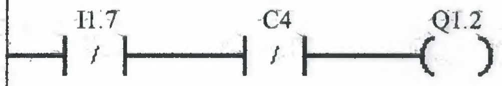
Network 13 COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 1 (50 TO 100)



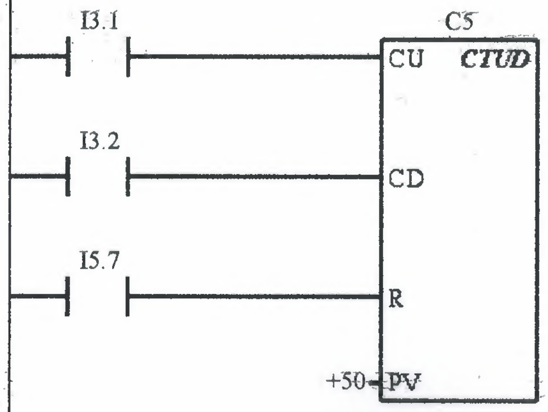
Network 14 COOLER.(DECREASE THE 1.5 DEGREE CELCIUS)



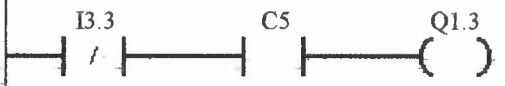
Network 15 HEATER (INCERASE THE 1.5 DEGREE CELCIUS)



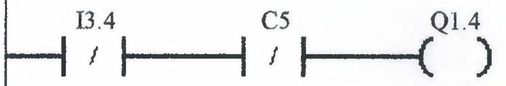
Network 16 COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 2 (0 TO 50)



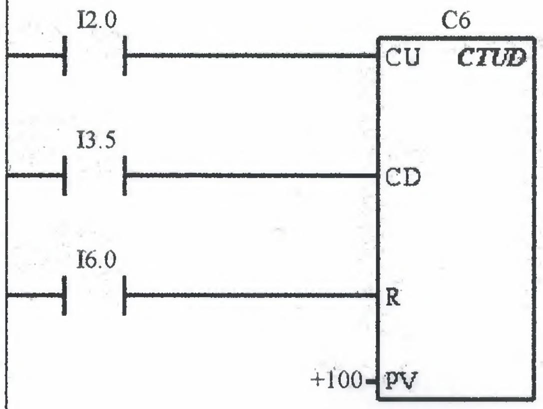
Network 17 COOLER.(DECREASE THE 1.5 DEGREE CELCIUS)



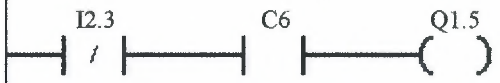
Network 18 HEATER (INCERASE THE 1.5 DEGREE CELCIUS)



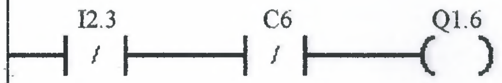
Network 19 COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 2 (50 TO 100)



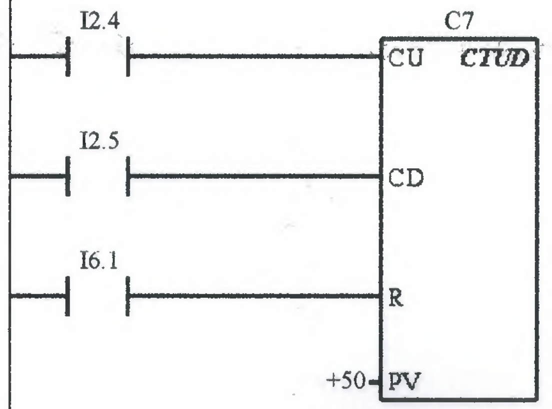
Network 20 COOLER.(DECREASE THE 1.5 DEGREE CELCIUS)



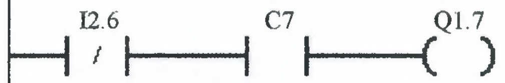
Network 21 HEATER (INCERASE THE 1.5 DEGREE CELCIUS)



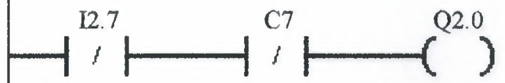
Network 22 COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 3 (0 TO 50)



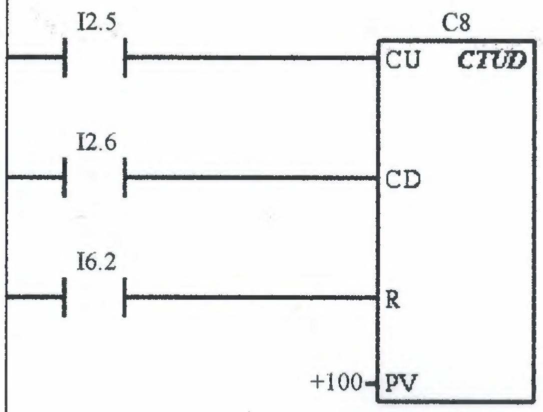
Network 23 COOLER.(DECREASE THE 1 DEGREE CELCIUS)



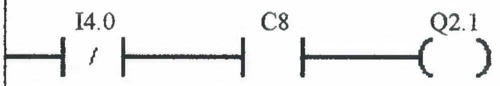
Network 24 HEATER (INCERASE THE 1 DEGREE CELCIUS)



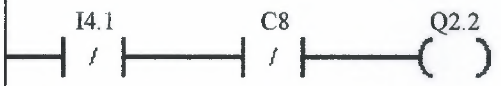
Network 25 COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 3 (50 TO 100)



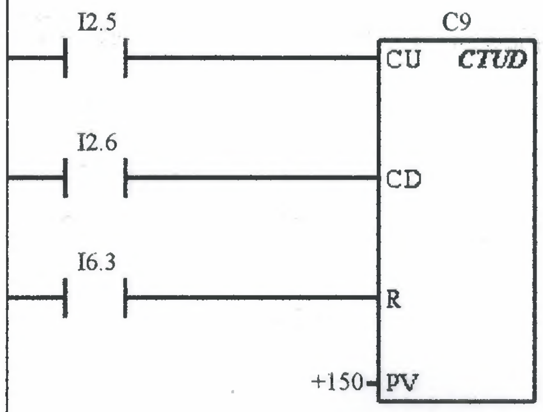
Network 26 COOLER (DECREASE THE 1 DEGREE CELCIUS)



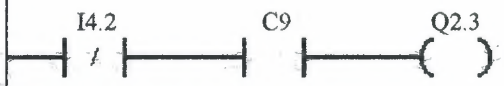
Network 27 HEATER (INCERASE THE 1 DEGREE CELCIUS)



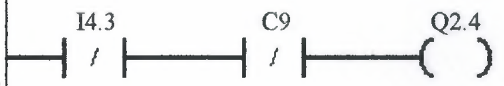
Network 28 COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 3 (100 TO 150)



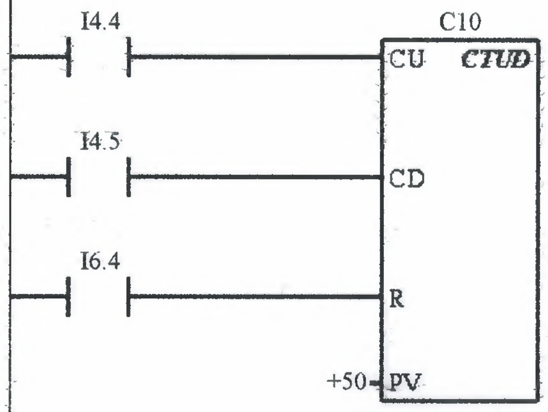
Network 29 COOLER (DECREASE THE 1 DEGREE CELCIUS)



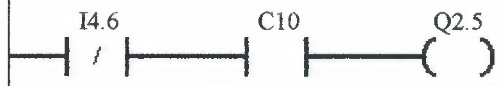
Network 30 HEATER (INCERASE THE 1 DEGREE CELCIUS)



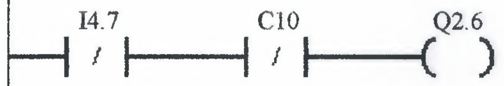
Network 31 COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 4 (0 TO 50)



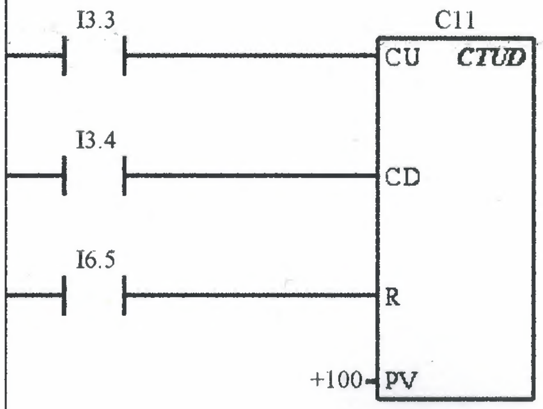
Network 32 COOLER (DECREASE THE 1 DEGREE CELCIUS)



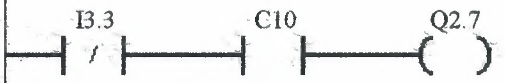
Network 33 HEATER (INCERASE THE 1 DEGREE CELCIUS)



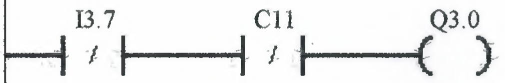
Network 34 COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 4 (50 TO 100)



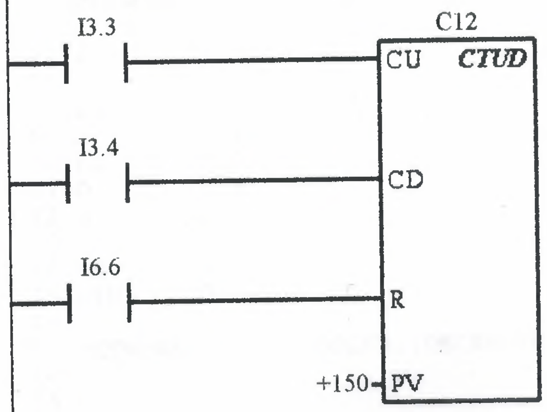
Network 35 COOLER (DECREASE THE 1 DEGREE CELCIUS)



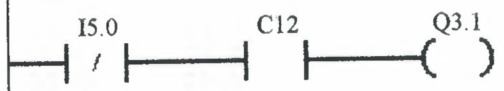
Network 36 HEATER (INCERASE THE 1 DEGREE CELCIUS)



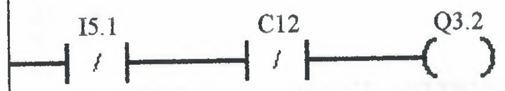
Network 37 COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 4 (100 TO 150)



Network 38 COOLER.(DECREASE THE 1 DEGREE CELCIUS)



Network 39 HEATER (INCERASE THE 1 DEGREE CELCIUS)



Network 40

```
//
//PROGRAM TITLE COMMENTS
//

NETWORK 1    //COUNT ENTERING OR LEAVING  PEOPLE IN BIG HALL (1 TO 234)
//
//
LD      I0.0
O       I0.1
LD      I0.2
O       I0.3
O       I0.4
O       I0.5
LD      I5.2
CTUD    C0, +234

NETWORK 2    //COOLER.(DECREASE THE 1 DEGREE CELCIUS)
LDN     I0.6
A       C0
=       Q0.1

NETWORK 3    //HEATER (INCERASE THE 1 DEGREE CELCIUS)
LDN     I0.7
AN      C0
=       Q0.2

NETWORK 4    //COUNT ENTERING OR LEAVING  PEOPLE IN BIG HALL (234 TO 468)
LD      I0.0
O       I0.1
LD      I0.2
O       I0.3
O       I0.4
O       I0.5
LD      I5.3
CTUD    C1, +468

NETWORK 5    //COOLER.(DECREASE THE 1 DEGREE CELCIUS)
LDN     I1.0
A       C1
=       Q0.3

NETWORK 6    //HEATER (INCERASE THE 1 DEGREE CELCIUS)
LDN     I1.1
AN      C1
=       Q0.4

NETWORK 7    //COUNT ENTERING OR LEAVING  PEOPLE IN BIG HALL (468 TO 702)
LD      I0.0
O       I0.1
LD      I0.2
O       I0.3
O       I0.4
O       I0.5
LD      I5.4
CTUD    C2, +702

NETWORK 8    //COOLER.(DECREASE THE 1 DEGREE CELCIUS)
LDN     I2.0
A       C2
=       Q0.5

NETWORK 9    //HEATER (INCERASE THE 1 DEGREE CELCIUS)
LDN     I2.1
AN      C2
=       Q0.6
```

```

NETWORK 10   //COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 1 (0 TO 50)
LD      I1.2
LD      I1.3
LD      I5.5
CTUD    C3, +50

NETWORK 11   //COOLER.(DECREASE THE 1.5 DEGREE CELCIUS)
LDN     I1.4
A       C3
=       Q0.7

NETWORK 12   //HEATER (INCERASE THE 1.5 DEGREE CELCIUS)
LDN     I1.5
AN      C3
=       Q1.0

NETWORK 13   //COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 1 (50 TO 100)
LD      I1.3
LD      I1.4
LD      I5.6
CTUD    C4, +100

NETWORK 14   //COOLER.(DECREASE THE 1.5 DEGREE CELCIUS)
LDN     I1.6
A       C4
=       Q1.1

NETWORK 15   //HEATER (INCERASE THE 1.5 DEGREE CELCIUS)
LDN     I1.7
AN      C4
=       Q1.2

NETWORK 16   //COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 2 (0 TO 50)
      LD      I3.1
      LD      I3.2
      LD      I5.7
      CTUD    C5, +50

      NETWORK 17   //COOLER.(DECREASE THE 1.5 DEGREE CELCIUS)
      LDN     I3.3
      A       C5
      =       Q1.3

      NETWORK 18   //HEATER (INCERASE THE 1.5 DEGREE CELCIUS)
      LDN     I3.4
      AN      C5
      =       Q1.4

      NETWORK 19   //COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 2 (50 TO 100)
      LD      I2.0
      LD      I3.5
      LD      I6.0
      CTUD    C6, +100

      NETWORK 20   //COOLER.(DECREASE THE 1.5 DEGREE CELCIUS)
      LDN     I2.3
      A       C6
      =       Q1.5

      NETWORK 21   //HEATER (INCERASE THE 1.5 DEGREE CELCIUS)
      LDN     I2.3
      AN      C6
      =       Q1.6
```

```

NETWORK 22   //COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 3 (0 TO 50)
LD      I2.4
LD      I2.5
LD      I6.1
CTUD    C7, +50

NETWORK 23   //COOLER.(DECREASE THE 1 DEGREE CELCIUS)
LDN     I2.6
A       C7
=       Q1.7

NETWORK 24   //HEATER (INCERASE THE 1 DEGREE CELCIUS)
LDN     I2.7
AN      C7
=       Q2.0

NETWORK 25   //COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 3 (50 TO 100)
LD      I2.5
LD      I2.6
LD      I6.2
CTUD    C8, +100

NETWORK 26   //COOLER.(DECREASE THE 1 DEGREE CELCIUS)
LDN     I4.0
A       C8
=       Q2.1

NETWORK 27   //HEATER (INCERASE THE 1 DEGREE CELCIUS)
LDN     I4.1
AN      C8
=       Q2.2

NETWORK 28   //COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 3 (100 TO 150)
LD      I2.5
LD      I2.6
LD      I6.3
CTUD    C9, +150

NETWORK 29   //COOLER.(DECREASE THE 1 DEGREE CELCIUS)
LDN     I4.2
A       C9
=       Q2.3

NETWORK 30   //HEATER (INCERASE THE 1 DEGREE CELCIUS)
LDN     I4.3
AN      C9
=       Q2.4

NETWORK 31   //COUNT ENTERING OR LEAVING PEOPLE IN SMALL HALL 4 (0 TO 50)
LD      I4.4
LD      I4.5
LD      I6.4
CTUD    C10, +50

NETWORK 32   //COOLER.(DECREASE THE 1 DEGREE CELCIUS)
LDN     I4.6
A       C10
=       Q2.5

NETWORK 33   //HEATER (INCERASE THE 1 DEGREE CELCIUS)
LDN     I4.7
AN      C10
=       Q2.6
```

```

    NETWORK 34   //COUNT ENTERING OR LEAVING PEOPLE IN SMALL
HALL 4 (50 TO 100)
    LD     I3.3
    LD     I3.4
    LD     I6.5
    CTUD   C11, +100

    NETWORK 35   //COOLER.(DECREASE THE 1 DEGREE CELCIUS)
    LDN    I3.3
    A      C10
    =      Q2.7

    NETWORK 36   //HEATER (INCERASE THE 1 DEGREE CELCIUS)
    LDN    I3.7
    AN     C11
    =      Q3.0

    NETWORK 37   //COUNT ENTERING OR LEAVING PEOPLE IN SMALL
HALL 4 (100 TO 150)
    LD     I3.3
    LD     I3.4
    LD     I6.6
    CTUD   C12, +150

    NETWORK 38   //COOLER.(DECREASE THE 1 DEGREE CELCIUS)
    LDN    I5.0
    A      C12
    =      Q3.1

    NETWORK 39   //HEATER (INCERASE THE 1 DEGREE CELCIUS)
    LDN    I5.1
    AN     C12
    =      Q3.2

    NETWORK 40
    MEND
```

CONCLUSION

When developing this project we see that PLC is making the operation in industrial , Places and that's the reasons, Why it is gaining interest , Notice of most of the companies.

With the information observed from our lecturer and our researchers for this topic

PLC , is a convenient tool with a wide rage of useful ways to be used. Such examples can be mentioned several machines can be used at the same time , easy adjustments from the PLC program can be made within a few minutes by the keyboard , installed PLC programs can be controlled or checked before within the office and laboratory , even the PLC program as for firm can be meet at the home. It's very protective and safe for the workers which they protected from danger , communications programs of PLC's within each other or within operates can happen with the PLC ; the developed lantues have constructed the productivity , security establishment security fast productivity , quality , and we can see that PLC is a very cheap program that can be fundamentally used.

The result of our program is that every ewquipment in a building can be operated with a PLC and in this way automation is achieved.

REFERENCES

Reference : Programmable Controllers – Operation and Application
Ian G. Warnock (1988) Prentice Hall International Ltd.

Reference: SIMATIC S7 – 200 and Industrial Automation
Doç. Dr. Salman Kurtulan (July 1998) İTÜ Electric & Electronic Department

Reference: PLC
Richard Baldry (November 1999)

Reference: Programmable Logic Controllers
Hugh Jack (June 1999)

Reference:
Mustafa Yağımlı & Feyzi Akar (1999). Programmable Logic Controllers.

Reference:
Erdoğan Teközgen İstanbul, (1998). PLC ve Uygulamaları

Reference:
HAKER Soğuk Döküm San. Tic. Lti. Şti. Tansel Sarıçam İZMİR

Reference:
World Wide Web: www.siemens.com

Reference:
EGESİM Siemens Ana Bayii. 1204 Sok. No:41/1-l Bulanalp

APPENDIX