

NEAR EAST UNIVERSITY



Faculty of Engineering

**Department of Electrical and Electronic
Engineering**

**FULLY CONTROL A PARKING AREA WITH
(PLC)**

**Graduation Project
EE 400**

Student: Ömer Doğan (960355)

Supervisor: Mr. Özgür C. Özerdem

Lefkoşa- 2001

CONTENTS

ACKNOWLEDGEMENT	i
ABSTRACT.....	ii
INTRODUCTION.....	1
CHAPTER I	3
1.1. THE TYPES OF PLC.....	3
a) PLC Size And Practise.....	4
b) I/O Unit.....	4
c) Different I/O Units.....	6
d) Analog Input/Output Unit.....	11
e) CPU.....	11
f) Processor Memory Module.....	12
1.2. MEMORY DESIGN.....	12
a) I. Group Memories.....	13
b) II. Group Memories.....	13
1)PROM	
2)EPROM	
3)EAROM	
4)EEPROM	
1.3. PROGRAMMING DEVICES.....	14
CHAPTER II	16
2.1. PLC PROGRAMMING SOFTWARE.....	16
2.2. CREATE OF LADDER DIAGRAM.....	16
a) Start Command.....	16
b) AND and OR Exercising.....	18
c) Output Stored Exercises.....	18
2.3. SPECIFICATION OF EXAMINED PLC.....	19
a) Mitsubishi F1 20 MR.....	19
b) Siemens Simatic S5-90U.....	20
c) AEG Teachware Modicon A020.....	20
d) FESTO (FPC 202C).....	21
2.4. CREATING COMMAND LINE FOR LOGIC PROCESS.....	22
a) Loading Of Close And Open Contact.....	22
b) AND Exercise.....	24
c) AND NOT Exercise.....	24
d) OR Exercise.....	25
e) OR NOT Exercise.....	25
2.5. GET COMMUNICATE OF COMMAND BLOCK TOGETHER.....	26
a) Serial Contact.....	26
b) Parallel Contact.....	26
2.6. SET AND RESET INSTRUCTION.....	27
2.7. SINGLE OUTPUT INSTRUCTIONS.....	29
2.8. JUMP INSTRUCTION.....	29
2.9. TIMERS.....	30
2.10. COUNTERS.....	33

2.11. SHIFT REGISTER.....	40
2.12. COMPUTING FUNCTION.....	42
SECTION III	44
3.1. BASIC INSTRUCTION WORD.....	44
a) Basic Instructions.....	44
b) FUN (Function) Instructions.....	44
c) Input.....	45
d) Output.....	45
e) Internal Relay.....	45
f) Special Internal Relay.....	46
g) Timer.....	46
h) Counter.....	46
i) Reversible Counter.....	46
j) Shift Register.....	47
k) Single Output.....	47
l) Data Register.....	47
3.2. FA1J SERIES ALLOCATION NUMBERS OF SPECIAL RELAYS.....	47
3.3. BASIC INSTRUCTION.....	48
a) LOD Instructions.....	48
b) Input, Output, Internal and Special Relays.....	48
c) Timer.....	49
d) Counter.....	49
e) Shift Register.....	49
f) AND Instruction.....	49
g) OR Instruction.....	50
h) NOT Instructions.....	50
SECTION IV.....	51
4.1. CHOOSING INSTALLATION AND COMMISSIONING OF PLC SYSTEM.....	51
4.2. FEASIBILITY STUDY.....	51
4.3. DESIGN PROCEDURE FOR SYSTEM.....	52
a) Choosing Programmable Controller.....	52
b) Size and Type of PLC System.....	53
c) I/O Requirements.....	53
d) Memory and Programming Requirements.....	54
e) Instruction	57
4.4. INSTALLATION.....	57
4.5. TESTING AND COMMISSIONING.....	59
4.6. a) Soft Ware Testing and Simulation.....	60
4.6. b) Installing and Runningn The User Control Program.....	64
MY PROJECT.....	65
CONCLUSION.....	69
TABLE OF SYMBOL.....	70
APPENDIX	71
REFERENCES.....	100

ACKNOWLEDGEMENTS

I must first acknowledge some debts from my more distant past. To my parents for their support to an upwardly mobile son. To my friend Murat Muhammet GÜVEN, Mehmet KİNSİZ, for his help.

I want to thank separately to my teachers that during my education they support me, the Dean of Engineering Faculty Prof. Dr. Fakhreddin MAMEDOV, Head of Electrical & Electronic Department Prof. Dr. Fakhreddin MAMEDOV and also Mr. Özgür ÖZERDEM, Mr. Kaan UYAR, Assis. Prof. Dr. Kadri BÜRÜNCÜK.

And finally, I want to thank everybody who helped and supported me to come to today and graduated from a university.

ABSTRACT

PLC (Programmable Logic Controllers) is a thing that programmable with computer support to take more efficiency from time and workers. It is divided into two parts. Hardware and software.

The hardware are the parts of machine those are CPU, I/O device and Programming device. CPU is basic microprocessor system and it carries out as control sensor, counter, timer function. CPU carries out stored user program in memory will input informations from various sensor circuits and can sending suitable output to commands and control circuits. I/O Module receives 120 VAC signal in device or processing device and transforms 5 VDC signal form.

There are many specialisation such as timer, counter, master control set, which works data and controls program, master control reset, JMP. There are command which are mathematics process that are comparator processes. These are the main function and feature of software part of PLC.

INTRODUCTION

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

When production requirements changed so did the control system. This becomes very expensive when the change is frequent. Since relays are mechanical devices they also have a limited lifetime which required strict adherence 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 PLCs. Conventional microprocessors lacked the power to quickly solve PLC logic in all but the smallest PLCs. As conventional microprocessors evolved, larger and larger PLCs were being based upon them. However, even today some are still based upon the 2903. Modicon has yet to build a faster PLC than their 984A/B/X, which was based upon the 2901.

Communications abilities began to appear in approximately 1973. The first such system was Modicon's Modbus. The PLC could now talk to other PLCs and they could be far away from the actual machine they were controlling. They could also now be used to send and receive varying voltages to allow them to enter the analogue world. Unfortunately, the lack of standardisation coupled with continually changing technology has made PLC communications a nightmare of incompatible protocols and physical networks.

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

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

CHAPTER I

1.1. THE TYPES OF PLC

In general, PLC divides to three sections;

- *Central Processing unit(CPU)
- *The input/output section
- *The programming device

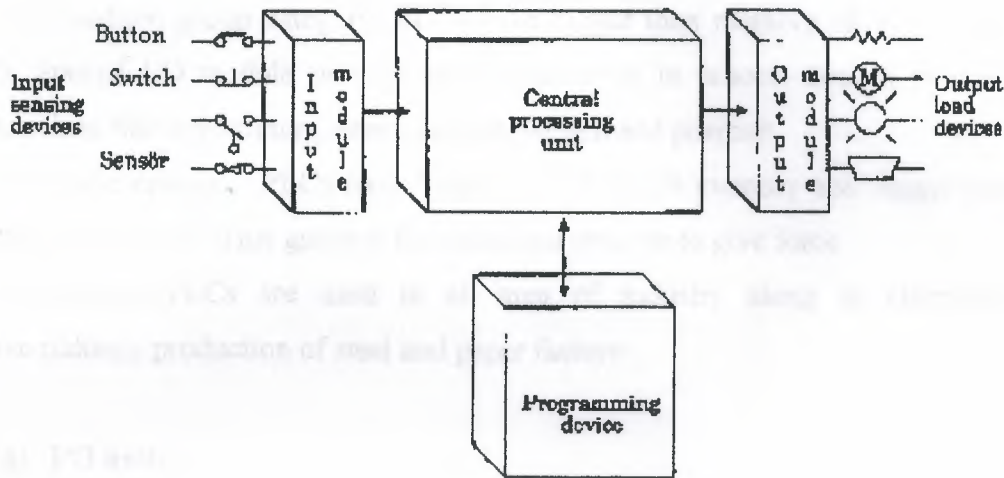


Figure.1.1.1. PLC sections

(CPU), PLC system and there are various logic circuit gates. CPU is basic microprocessor system and it carries out as control relay, counter, timer functions. CPU carries out user programs stored in memory and read input data from various sensor circuits and can send suitable outputs to commands and to control circuits.

Direct current power supply must be used for the low level voltage that these are using in processor and I/O models. This power supply is a part of CPU. PLC system is independent in its structure and also it can be dependent to its system.

I/O system forms can be connected to controller by other devices. The aim of interface is to send various signals and to take situations to external devices. The output devices for example, motor starters, solenoid valves, indicator lights connected to terminals on the output module.

The desired program loads to processor's memory by programming device or terminal. This program can enter to relay during using ladder logic. Program can be obtained till the main control or machines by sequential processes.

a) PLC size and practice:

There are 3 different categories of PLC; as small, medium and large.

*In small group category, PLC has bigger than input/output of 128 I/O and bigger than memory of 2 KB.

*In medium group category, PLCs have bigger than memory of 32 KB and 2048 I/O. Special I/O module provide easily adaptation in process control practice, analog functions like temperature, press, current, weight and position.

*In large category, PLCs have bigger than 750 KB memory and bigger than input/output of 8192 I/O. This group is for unlimited practice to give force.

Nowadays, PLCs are used in all area of industry along in chemistry, automotive industry production of steel and paper factory.

b) I/O unit:

I/O unit forms is the input/output rack of PLC. I/O unit receives 120 Vac signal in device or processing devices and transforms 5 Vdc signal form. In output units controller signals (5Vdc) are used to devices or processor control as 120 Vac. These output signals provide low current control that used in power electronic elements or optic isolators. Input/output unit in PLC can be put in the same structure or different structure with CPU. This standard input/output unit is in the following shape.

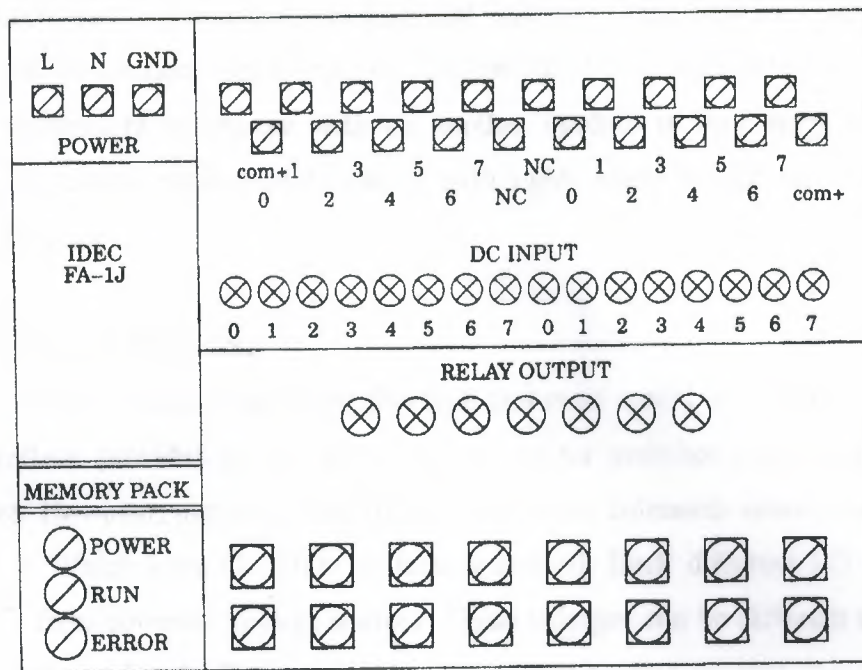


Figure. 1.1.2. In the same structure CPU with PLC I/O unit

Between processor and I/O rack communication different connection cables are permitted. This condition is as the following figure 1.1.3.

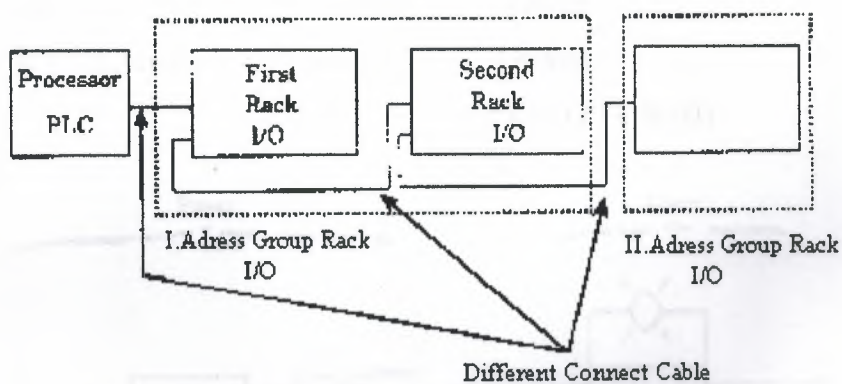


Figure. 1.1.3 Between Processing I/O Racks communication

I/O units each input/output has a special address. These addresses are known by the processor. To connect output/input an element with I/O or separating is very easy and quick. Furthermore to change with an another module is very easy. ON/OFF condition of I/O circuit each module shows with light. Many output modules have rubbish fuse indicator.

c) Different I/O units:

Many output I/O units are from this type and most useful is interface module. This type interface provides to link of inputs as selector switches push buttons and limits switches. However, output control lights small motor solenoids sensor and motor starters limit it. Which have ON/OFF contacting control. Each different I/O module takes its power from common voltage sources. These voltages can be different size and type. These are showed in the following table.

<u>Input Interface</u>	<u>Output Interface</u>
24Vac/dc	12-48Vac
48Vac/dc	120Vac
120Vac/dc	230Vac
230Vac/dc	120Vdc
5Vdc (TTL level)	230Vdc
	5Vdc (TTL level)

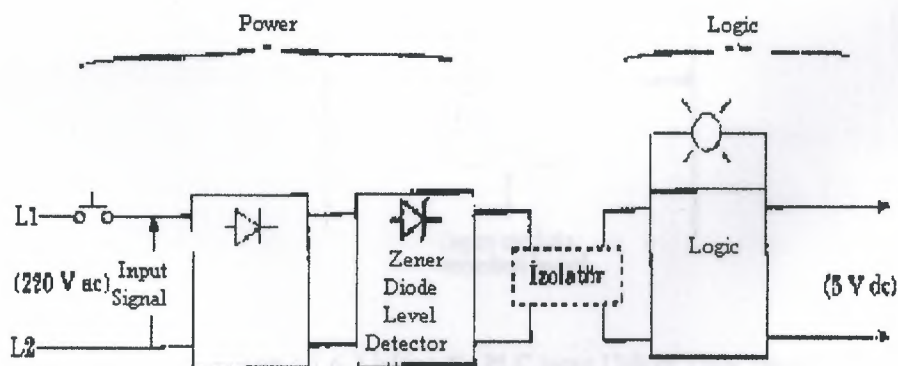


Figure.1.1.4 AC input interface block diagram

Shows that entries block diagram for an alternative current to input module.
Input circuit compose of to main section as power and logic section.

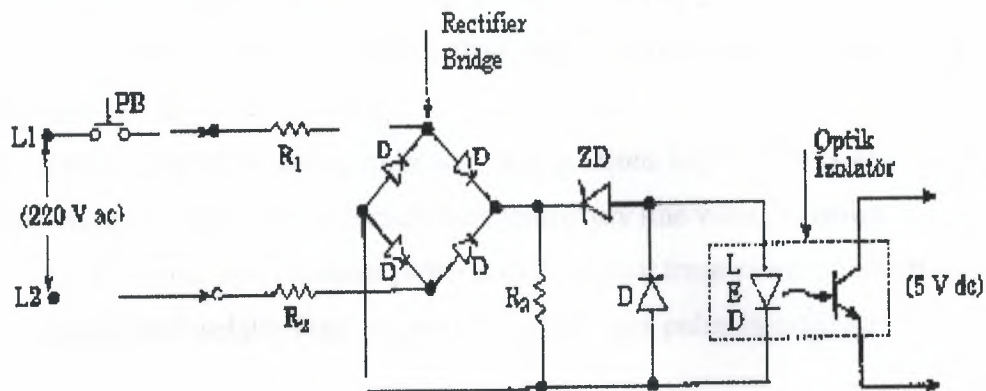


Figure.1.1.5. Simplified Circuit For a AC Module

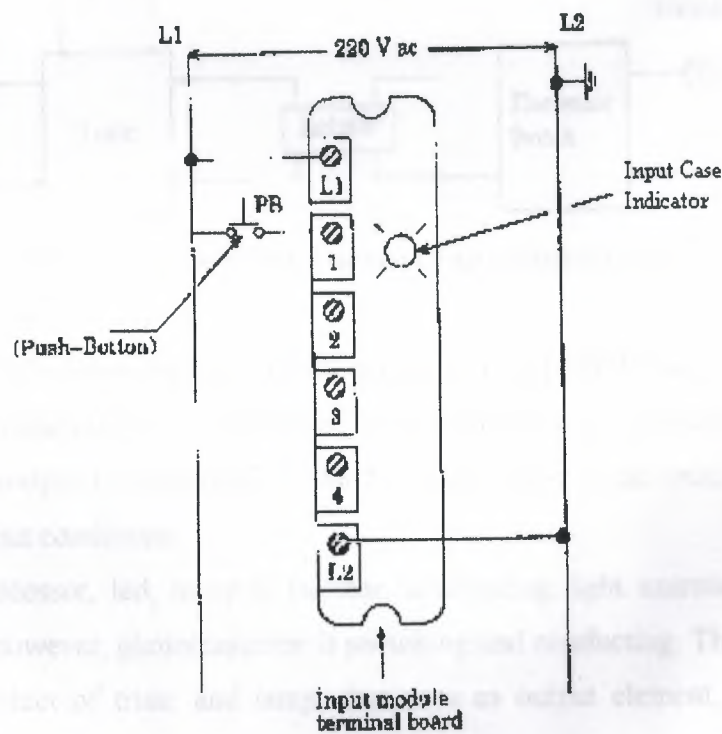


Figure.1.1.6. Linking To PLC Input Unit of 220V Input

Figure 1.1.4 and 1.1.5 shows figural diagram of Ac input module for input,
also figure 1.1.6 shows connect terminal.

When push button shuts down, bridge type treatment exercise 220V AC voltage from R_1 and R_2 resistance's.

Zener diode (ZD) voltage limit regulates according to low level voltage.

When light come to processor from led with phototransistor that means low level voltage (5V' dc) is transmitted.

Optic isolator separates high AC voltage from logic circuits also protects to processor from damages, which comes from temporary line voltage change.

Furthermore, optic isolator protects to processor from effect of electrical noise.

Kuplaj and isolation can be created with using a pulse transformation.

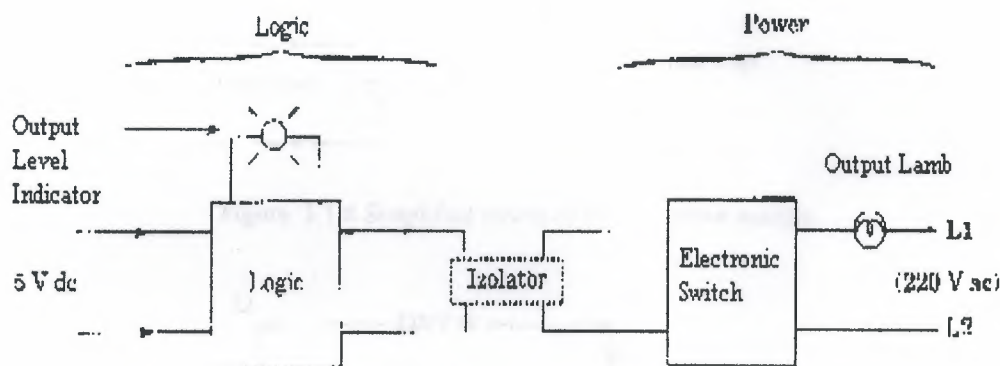


Figure. 1.1.7 typical a block diagram of output interface module.

Figure. 1.1.7 shows typical a block diagram of output interface module. Also output module, as input module, composes of two departments such as power and logic.

Device in output is controlled by the 5V comes from logic unit. In this unit, processor sets output conditions.

When processor, led, in optic isolator, distributing light exercises an output voltage (5V' dc), however, phototransistor is switching and conducting. This means that to detect and conduct of triac, and lamp, that uses as output element, turn on ON condition.

When led in logic unit turn off, logic become 0 condition and phototransistor cannot conduct. If a DC device in output will be controlled, it is carried with circuit.

PLC device will not be damaged from optic isolation that will be from power department.

If many high fast ON-OFF is necessary, in right current transistor and also alternative current triac circuits are used. Current cannot pull on PLC from output modules. Maximum current capacity of each device exists in their catalogs of that model.

In high currents instead of triac or other effect elements, standard relay must use as table 6. There are output/input unit as analog/digital translator (ADC) and digital/analog translator (DAC) that it is necessary for feedback control exercises in PLC devices.

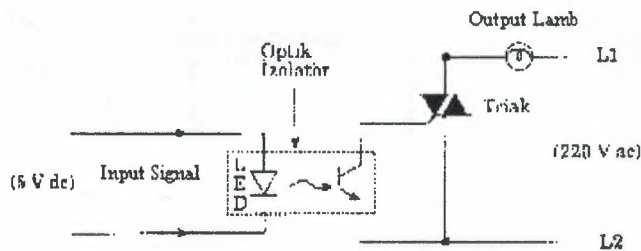


Figure. 1.1.8 Simplified circuit of an AC output module.

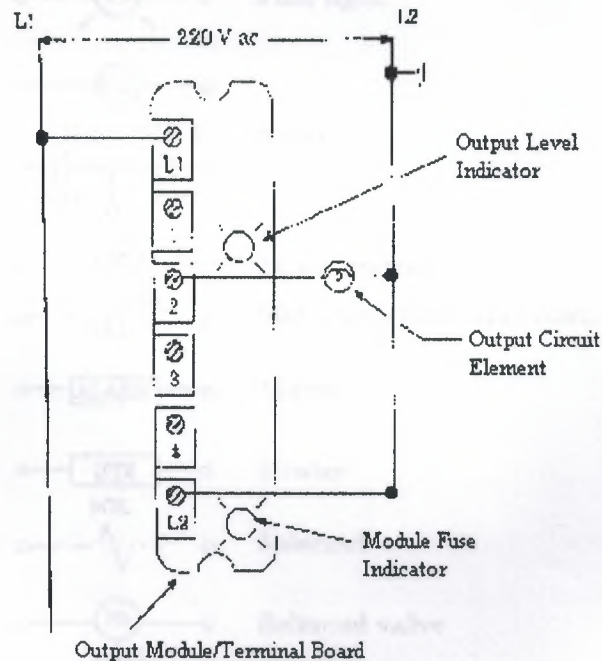


Figure. 1.1.9 Internal wire connection typical an output module

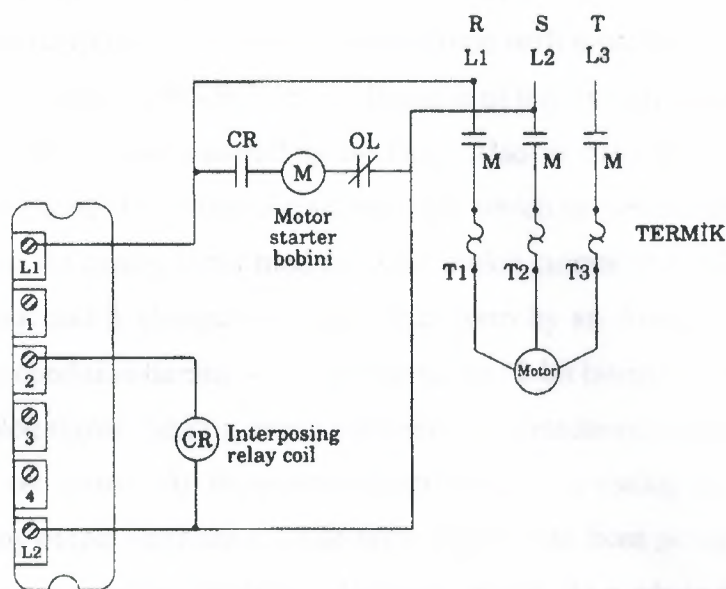


Figure. 1.1.10 Sensor connection points

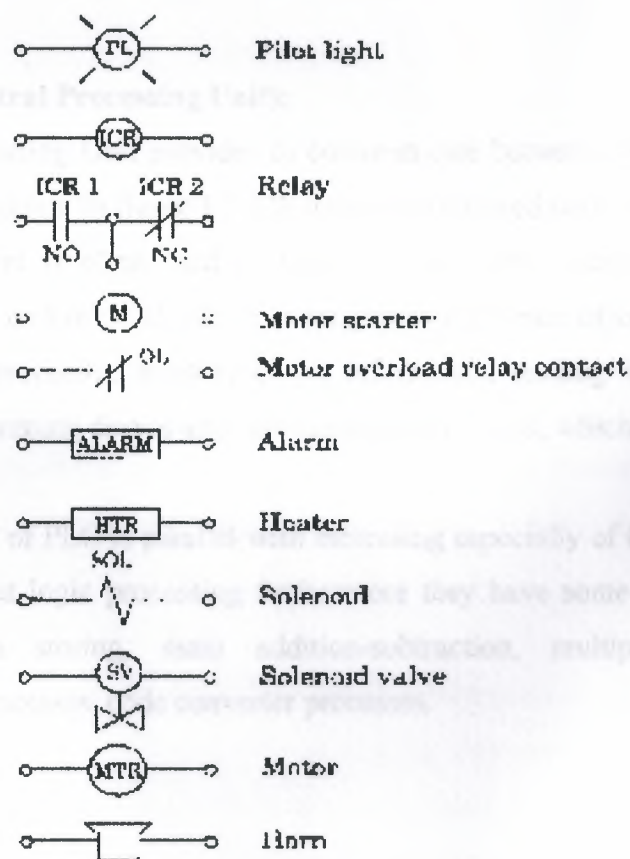


Figure. 1.1.11 Symbols of output control circuit

d) Analog input/output unit (I/O modules):

First produced PLCs only had been limited with separate I/O interfaces which had been allow to link to ON/OFF device. Because of this limitation many of processing exercises could be as part controlling by PLC. Also in days PLCs included analog interface and separate (I/O) input/output interface, which carries out practically many of control process. An analog input module takes analog current and voltage that is taken off analog input and it changed to digital data form by an Analog Digital Converter (ADC). In this condition turning levels are shown as 12-bit binary or 3 digit BCD that is rates with analog signal. Analog sensor elements are transducers as heat, light, velocity, pressure, and wet sensors. All these sensors can be linked to analog input

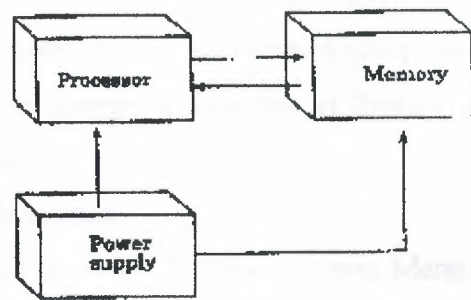
Analog output interface module takes digital data from processor, charges rate with voltage and current and controls a device as analog. As a whole digital data passes from Digital/Analog output device are small motors, valves and analog measure devices.

e) CPU (Central Processing Unit):

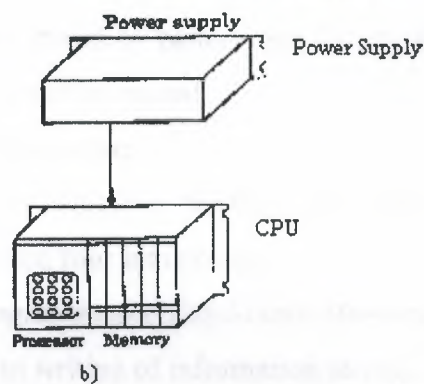
Central Processing Unit provides to communicate between power supply and processor memory modules. In figure 1.2.12b it can find covered both of two units.

CPU statement is often used as mean of processor statement. Processor-memory creates a big unit of CPU, which is programmable brain of controller. In this unit, there are microprocessor, memory chips, information reading and request data from memory, programming device and communication circuits, which is necessary for processor.

Development of PLC is parallel with increasing especially of CPU. In our day PLC systems carry out logic processing furthermore they have some especially such timer, counter, data storing, main addition-subtraction, multiplication-division processes, compare processes, code converter processes.



a) Simplifies of CPU Structure



b)

Figure. 1.1.12 CPU; the elements of central processing unit (a) the structure of simplified CPU
(b) power supply unit different from CPU.

f) Processor-Memory Module:

CPU is the brain of programmable of controller and a big part of CPU family forms from processor memory unit. This module cover microprocessor, memory chips programming device and necessarily communication circuits for processor interface.

Furthermore processor carries out other functions. For example, it carries out timer, counter, compare, keeper and addition, subtraction, multiplication and division functions, which are four main functions of mathematics.

1.2. 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, 1024byte = 1kbyte and the number of memory capacity is stated these units.

The memory types are divided into two groups;

The first group: the energy of power supply is cut that supplied memory, it means that memory had been erased. Also second group: hide information cannot lose if the energy is cut. But to change of includes of those types of memories, there is a necessary a special system.

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;

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.

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

3) *EAROM (Electrically Alterable 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.

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.

1.3. PROGRAMMING DEVICES

The most important one of features of programmable controller is to have programming elements, which are useful. Programming device provides transformation between operator and circuit of controller. (Fig. 1.3.1)

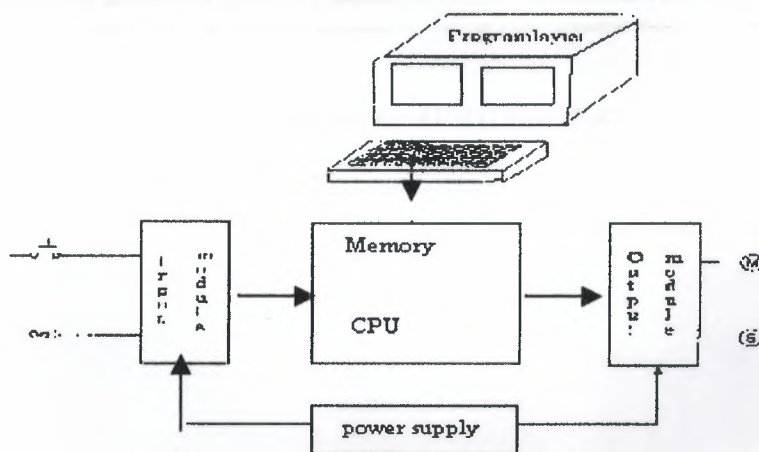


Figure. 1.3.1. Transformation of PLC Circuits

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 communicate 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 monitor 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 functions keys. FA2 of programming device IDEC FA1 Junior module is shown at table 1.3.2.

FA-2 PROGRAMMABLE CONTROLLER							
963 LOD T 20				PROGRAM LOADER			
ADRS	TIM	CNT	SFR	MCS 7	JMP 8	PROM 9	INST
DELT	SET	RST	END	MCR 4	JEND 5	CMT 6	↓ VERI
MON	OR	SOT	OUT	F147 1	F247 2	3	↑ READ
TRS	LOD	AND	NOT	0	FUN	CLR	ENT'R

Figure. 1.3.2. Programming Device of IDEC FA-1 PLC.

CHAPTER II

2.1. PLC PROGRAMMING SOFTWARE

In this section, PLC programming fundamental is prepared, student's capacity, which met PLC programming, is considered first time.

AND

OR

NOT

NAND

NOR

SET

RESET

Furthermore there are many specialisations such as TIMER, COUNTER, and MASTER CONTROL SET (MCS), which works data and controls PROGRAM, MASTER CONTROL RESET (MCR), JMP. There command which are mathematics process that are comparator processes ($=$, $<$, $>$).

In all PLC systems, to create logic process is programmed as the same are carried out some function. However, the main logic is the same that TIMER, COUNTER and SHIFT REGISTER functions are to get command and programmed but there can be some differences.

2.2. CREATE OF LEADER DIAGRAM

a) Start Commands:

These commands are first element of program. There are two type contact conditions as at table 2.2.1. First normally is open also second close.

Normally, starting with open contact this program command is to get command as LD IN, LD, LOD A, on PLC device. And also close contact is stated as LDI, LD NOT, LOD NOT, AN.



LADDER SYMBOL	COMMAND LINE					
	IDEC	FESTO	AEG	Mitsubishi	Siemens	OMRON
 F Normally open contact	LOD F	LD FLAG F LD IN F	UF	LD F	A F	LD F
 F Normally close contact	LOD NOT F	LD NOT FLAG F LD NOT IN F	UN F	LDI F	AN F	LD NOT F

Table 2.2.1. Load Exercising

Note: in table F value is constant and input/output interval relay, special relay, timer, counter can be SFR number.

According to this table at MITSUBISHI and HITACHI model normally open contact is shown with LD, also close contact is shown with LDI.

Also at AEG PLC, U (UND) command is used for open contact and (UN) UND-NICHT command is used for closed contact.

Also at SIEMENS PLC, A (AND) command is used to open contact and AN (AND-NOT) is used for closed contact.

At OMRON PLC, open contact is shown LD, also close contact is shown with LD NOT.

Also at FESTO PLC, open contact LD FLAG is used for flag load other conditions LD IN command is used to contact load. In normally, also close contact is programmed for flag exercising as LD NOT FLAG... For other contacts are programmed as LD NOT IN...

b) AND and OR Exercising:

LADDER SYMBOL	COMMAND LINE						
	IDEC	OMRON	AEG	MITSUBISHI	Siemens SIMATIC	OMRON	HITACHI
	LD X1 AND X2	LD IN X1 AND IN X2	U X1 U X2	LD X1 AND X2	AX1 AX2	LD X1 AND X2	LD X1 AND X2
	LD X1 AND NOT X2	LD IN X1 AND NOT IN X2	U X1 UN X2	LD X1 ANI X2	AX1 AN X2	LD X1 AND NOT X2	LD X1 ANIX2
	LD X1 OR X2	LD IN X1 OR X2	U X1 O X2	LD X1 OR X2	AX1 O X2	LD X1 OR X2	LD X1 OR X2
	LD X1 OR NOT X2	LD IN X1 OR NOT X2	U X1 ON X2	LD X1 ORI X2	AX1 ON X2	LD X1 OR NOT X2	LD X1 ORIX2

Table 2.2.2: Symbol and command line AND and OR exercises.

c) Output Stored Exercises:

At a PLC system relay, it is used as output function, can be divided into two groups. First group output which charge can be linked to it according to program as (solenoid valves, neon lamp, conductor, led, etc.) are real output. Also second group outputs are internal and image relays. Physical connection cannot link to these relays but outputs of these sensors are transferred to real output and output can be taken.

If commands will be observed, there are similarities between PLC devices that output program commands are different. At both output and input functions, X1, X2, are used as addresses.

LADDER SYMBOL	COMMAND LINE						
	IDEC	FESTO	AEG	MITSUBISHI	Siemens SIMATIC	OMRON	HITACHI
	LD X1 OUT X2	LD IN X1 = OUT X2	U X1 = X2	LD X1 OUT X2	A X1 = X2	LD X1 OUT X2	LD X1 OUT X2
	LD X1 SET X2	LD IN X1 SE FLAG X2	U X1 SL X2	LD X1 S X2	A X1 S X2	LD X1 SET X2	LD X1 SET X2

Figure.2.2.3.

2.3. SPECIFICATION OF EXAMINED PLC

a) Mitsubishi F1 20 MR

ELEMENT	Symbol	F1 20MR
(Inputs)	X	12 Unit 400 - 413
(Outputs)	Y	8 Unit 430 - 437
(Timer) 0.1 s	T	24 Unit 50 - 57, 450 - 457
(Timer) 0.01 s	T	8 Unit 650 - 657
(Counters)	C	30 Unit 60 - 67, 460 - 467
(Big speed counter)	C	2 Unit 660 - 661
(Internal Relay)	M	64 Unit 10 - 177
(Special Internal Relay)	M	16 Unit 70 - 77, 470 - 477, 570 - 575
Battery of Feeding Sensor	M	64 Unit 300 - 377
(Jump)	M	64 Unit 700 - 777

Table 2.3.1: table of element and element numbers

F1 10ER	
X	4 Unit 414 - 417
Y	6 Unit 440 - 445

Table 2.3.2. Increasing unit

F1 20 MR PLC as 12 inputs 8 outputs, which we use. If more input and output are necessary, input/output-increasing units are plugged to PLC. These units have various numbers output and input. At table 2.3.1, there are 4 inputs 6 outputs for F1 10 ER model.

b) Siemens Simatic S5-90U

Element Name	ELEMENT ADDRESS
(Input)	I0.0 – I127.7
(Output)	Q0.0-Q127.7
(Flag)	(retentive) F0.0 – F63.7
(Flag)	(nonretentive) F64.0 – F127.7
Accumulator	ACCUM1 ACCUM2
Timer	T0 – T31
(Counter)	(retentive) C0 – C7
	(nonretentive) C8 – C31
KB	(Constant) 1 byte 0 – 255
KC	(Constant count) 0 – 999
KI	(Tam sayılar) - 32768 +32767
KF	(Heksadesimal) 0 – FFFF
KY	(2 byte) 0 – 255 (her bit)
KT	(Timer) 0.0 – 999.3
FB	(Function block) 0 – 63
DB	(Data block) 2 – 63 [9,10]

Table 2.3.3: Specifications of S5-90U model Siemens Simatic.

c) AEG Teachware modicon A020

Operand Type	Operand	Unit
(inputs)	E1 – E24	24
(outputs)	A1 – A16	16
Analog Input	EWA 1 – EWA 4	4 analog
Analog Output	AWA 1	1 analog
Memory	M1 – M128	128 Unit
Timer	T1 – T16	16 timer
Counter	Z1 – Z16	16 Counter

Table 2.3.4. Specifications of AEG Teachware A020

d) FESTO (FPC 202C)

TOTAL UNIT	PARAMETERS	SYMBOL	EXPLANATION
16	Internal inputs	I 0.X and I 1.X	input 0.0-0.7 1.0-1.7
2	Internal half-words	IW0 and IW1	2 Unit
16	Internal outputs	O 0.X and O 1.X	Output 0.0-0.7 1.0-1.7
2	Internal output half-words	OW0 and OW1	2 Unit
256	Flags	F0.Y to F15.Y	Flag: (0.0-0.15) (1.0-1.15) (2.0-2.15).....(15.0-15.15)
16	Flag words	FW0 to FW15	16 Unit Present
1	Initialization Flag	FI	1 — —
24	Special function units	FU0 to FU23	24 — —
16	Field bus flag words	FU32 to FU47	16 — —
32	Timers	T0 to T31	32 — —
32	Timer words	TW0 to TW31	32 — —
32	Counters	C0 to C31	32 — —
32	Counters words	CW0 to CW31	32 — —
32	Counters preset	CW0 to CW31	32 — —
64	Registers	R0 to R63	64 — —
8	programs	P0 to P7	8 — —
8	prog/function modules	B0 to B7	8 — —
1	Errors	E	1 — —
1	Error word	EW	1 — —
48	External inputs	I 2.X to I 7.X	input (2.0-2.7) (3.0-3.7)(7.0....7.7) = Top. 48
6	External input words	IW2 to IW7	6
48	External output	O 2.X to O 7.X	Output (2.0...2.7).. (3.0....3.7) (7.0....7.7)
6	External output words	OW2 to OW7	6

Table 2.4.8 Specification of FESTO (FPC 202C) Module PLC

In this table, $x=(0,1,2,3,4,5,6,7)$ and $y=(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)$ are.

2.4. CREATING COMMAND LINE FOR LOGIC PROCESS

Each process in PLC programming is stated by a command and these commands provides connections of relay and contacts together, designations of outputs, counter, programming of timers and making of arithmetic comparison processes.

In our days, to experience PLC device of all firms are very hard. We will experience five brands. These brands are enough for us.

BRAND**MODEL**

1) IDEC	FA1-JUNIOR (FA1J)
2) FESTO	202-C
3) MITSUBISHI	F120 R
4) SIEMENS-SIMATIC	S5-90U
5) AEG TEACHWARE	MODICON A020

a) Loading of Open and Close Contact:

Normally open contact

LOD (LOAD)-IDEC

LD IN (LOAD)-FESTO

LD (LOAD)- MITSUBISHI

A (AND)- SIEMENS-SIMATIC

U (UND)-AEG



Normally close contact

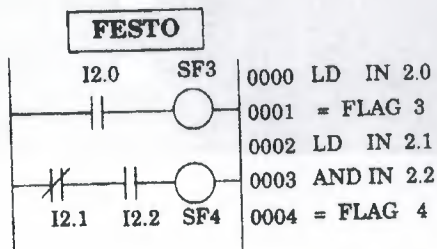
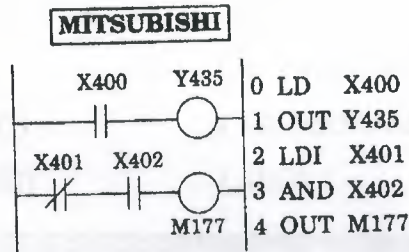
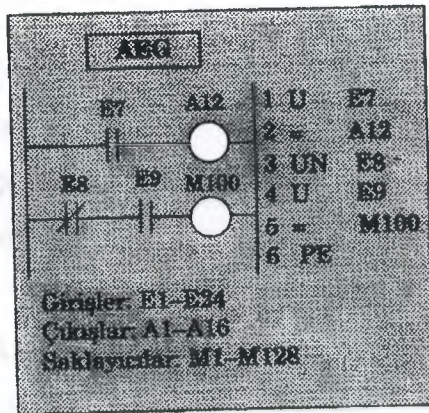
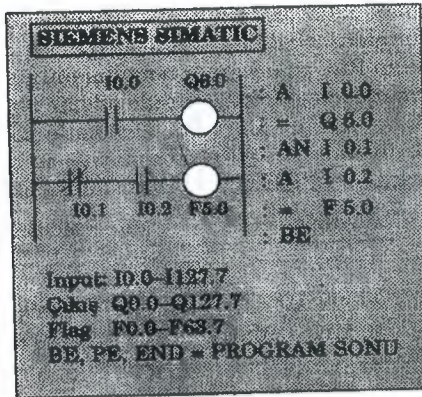
LOD NOT (LOAD NOT)-IDEC

LD NOT IN (LOAD NOT)-FESTO

LDI (LOAD INVERSE)- MITSUBISHI

AN (AND NOT)- SIEMENS-SIMATIC

UN (UND NICHT)-AEG



In here, commands for giving different brand and module normally. Explain to designation of contact and contact numbers are written after command.

In AEG and Siemens PLC, a load command is not used in Siemens Module, open contact command normally is load written A (AND), load process is relaying with AN (AND NOT) command.

In AEG module U (UND) and UN (UND NOT) commands are used for load process. As we know that these commands are used to serial AND and AND NOT exercises.

b) AND exercise:

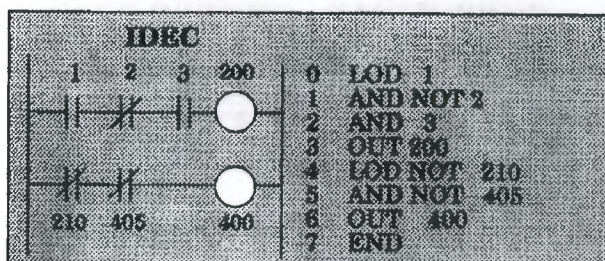
Serial contact linking commands

AND	-(IDEC)
AND IN	-(FESTO)
AND	-(MITSUBISHI)
A(AND)	-(SIEMENS-SIMATIC)
U (UND)	-(AEG)

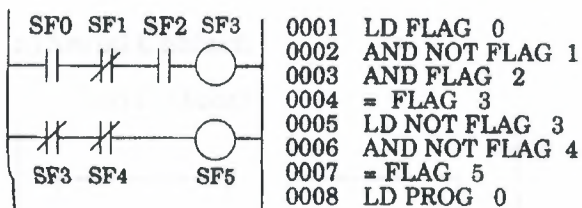
c) AND NOT exercise:

Serial contact linking commands

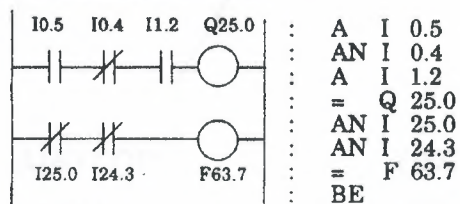
AND NOT	-(IDEC)
AND NOT IN	-(FESTO)
AND	-(MITSUBISHI)
A(AND)	-(SIEMENS-SIMATIC)
U (UND)	-(AEG)



FESTO



SIEMENS SIMATIC



d) OR exercise:

Parallel contact linking commands

OR	-(IDEC)
OR	-(FESTO)
OR	-(MITSUBISHI)
O(OR)	-(SIEMENS-SIMATIC)
O(ODER)	-(AEG)

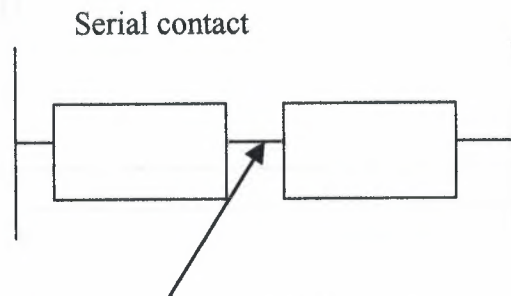
e) OR NOT exercise:

Parallel contact linking commands

OR NOT	-(IDEC)
OR NOT	-(FESTO)
ORI(OR INVERSE)	-(MITSUBISHI)
ON(OR NOT)	-(SIEMENS-SIMATIC)
ON(ODER NICHT)	-(AEG)

2.5. GET COMMUNICATE OF COMMAND BLOCK TOGETHER

a) Serial Contact:



AND LOD -(IDEC)

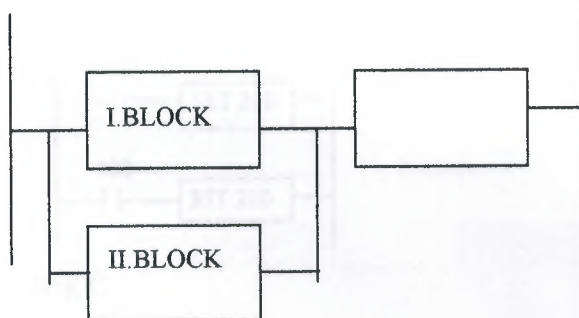
AND LD -(FESTO)

ANB (AND BLOCK) -(MITSUBISHI)

A(.....)	-(SIEMENS)
--------------	------------

U(.....)	-(AEG)
--------------	--------

b) Parallel Contact:



OR LOD -(IDEC)

OR LD -(FESTO)

ORB (OR BLOCK) -(MITSUBISHI)

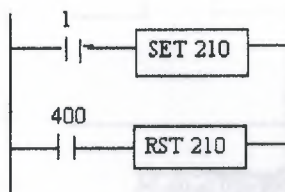
A(.....	-(SIEMENS)
O	
.....	
)	

O(-(AEG)
)	

2.6 SET AND RESET INSTRUCTION

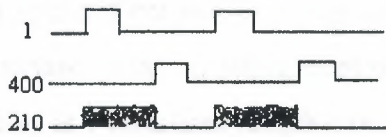
If any of the OFF position relay (eg. Input, output register and internal relay) pass the ON position that is from logic 0 to logic 1. Pass instruction called SET command. RESET command is opposite of SET command that is ON position to OFF position, from logic 1 to logic 0.

Another peculiarity of SET and RESET instructions for working instructions input must be control with relay. It does not require any continuous signal or stroke. That means SET relay always logic 1 position with input relay. If input relay done OFF position does not effect setted relay while that RESET command come.

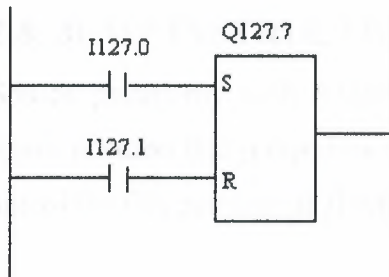


IDEC

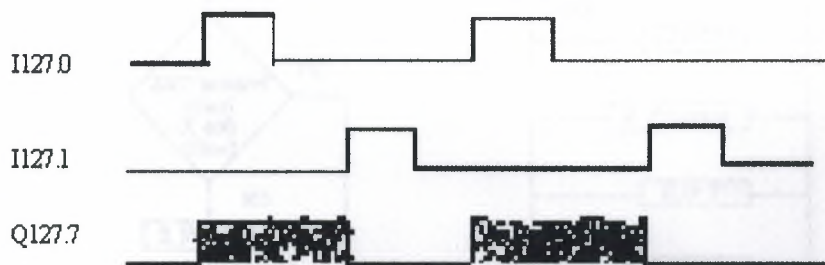
0	LOD	1
1	SET	210
2	LOD	400
3	RST	210
4	END	



SIEMENS



A I 127.0
 S Q 127.7 (Jump)
 A I 127.1
 R Q 127.7
 BE



2.7. SINGLE OUTPUT INSTRUCTIONS

Our aim is make ON position, on scan time length. With these aim we use two different relays. First one is which makes control, other one is where we take output. The important point is; while controlling relay passing OFF position to ON, where output relay is 1 scan time length mould pass ON position to OFF. It is unimportant that controlling relay is protecting ON position. When the OFF position relay pass to ON position, we take 1 scan time length from output relay.

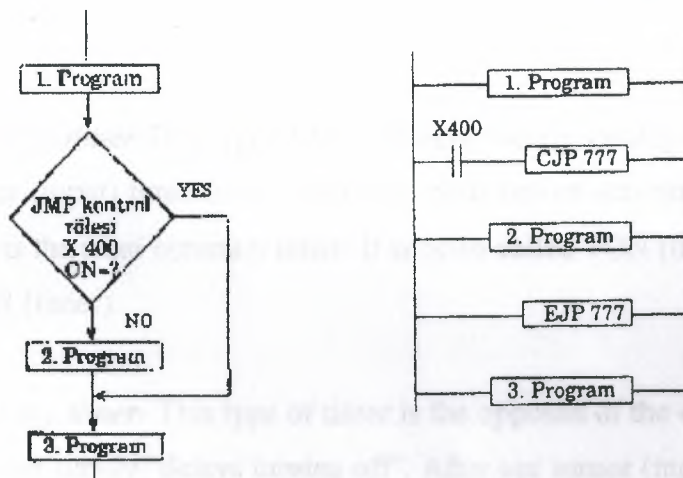
2.8. JUMP INSTRUCTION

Source peculiarity with JUMP instruction; determined program line or lines makes possible position that jumped by some condition, or conditions. Provided jumped relay is time of the ON position of JUMP command.

MITSUBISHI

CJP (Conditional Jump)

EJP (End of Jump)



Note: JUMP instructions are between 700 – 777

Above program is between the 1. and 2. Programs because of using JUMP instruction, 400-numbered input relay when passed logic 1 position, JUMP instruction come to active condition and 2. program jumped 3. program, and 3. program started to work. Because after the EJP, JUMP ending operation instruction.

With 401 numbered input came logic 1 (ON) jumping operation starts and from CJP 700 until EJP 700 line program line jumps.

Jumping operation goes when X401 OFF. When X401 OFF done program return to work normally and scan operation works line by line.

While X401 OFF position, JMP function does not work. The important point is; before CJP instruction, EJP used must go to last EJP operation. Others will be invalid.

2.9 TIMERS

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

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

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

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

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

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

What will enable the timer. Typically this is one of the inputs.(a sensor connected to input 0000 for example)

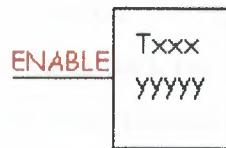
How long we want to delay before we react. Let's wait 5 seconds before we turn on a solenoid, for example.

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

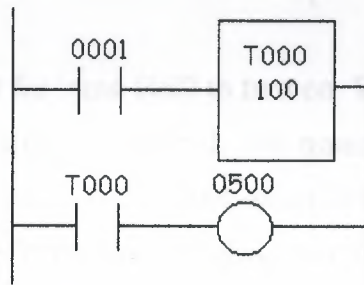
Why the weird numbers? Again its because most PLCs have 16-bit timers. We'll get into what this means in a later chapter but for now suffice it to say that 0-9999 is 16-bit BCD (binary coded decimal) and that 0 to 65535 is 16-bit binary. Each tick of the clock is equal to x-seconds.

Typically each manufacturer offers several different ticks. Most manufacturers offer 10 and 100 ms increments (ticks of the clock). An "ms" is a mili-second or *1/1000th* of a second. Several manufacturers also offer 1ms as well as 1 second increments. These different increment timers work the same as above but sometimes they have different names to show their time-base. Some are TMH (high speed timer), TMS (super high speed timer) or TMRAF (accumulating fast timer).

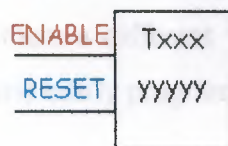
Shown below is a typical timer instruction symbol we will encounter (depending on which manufacturer we choose) and how to use it. Remember that while they may look different they are all used basically the same way. If we can setup one we can setup any of them.



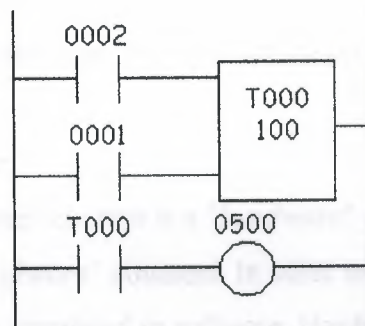
This timer is the on-delay type and is named Txxx. When the enable input is on the timer starts to tick. When it ticks yyyyy (the preset value) times, it will turn on its contacts that we will use later in the program. Remember that the duration of a tick (increment) varies with the vendor and the time-base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



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



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



In this diagram we wait for input 0002 to turn on. When it does timer T000 (a 10ms increment timer) starts ticking. It will tick 100 times. Each tick (increment) is 10ms so the timer will be a 1000ms (i.e. 1 second) timer. $100\text{ticks} \times 10\text{ms} = 1,000\text{ms}$. When 1 second has elapsed, the T000 contacts close and 500 turns on. If input 0002 turns back off the current elapsed time will be retained. When 0002 turns back on the timer will continue where it left off. When input 0001 turns on (true) the timer T000 will reset back to 0 causing its contacts to turn off (become false) thereby making output 500 turn back off.

2.10. COUNTERS

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

What kinds of counters are there? Well, there are up-counters (they only count up 1,2,3...). These are called CTU,(count up) CNT,C, or CTR. There are down counters (they only count down 9,8,7,...). These are typically called CTD (count down) when they are a separate instruction. There are also up-down counters (they count up and/or down 1,2,3,4,3,2,3,4,5,...) These are typically called UDC(up-down counter) when they are separate instructions.

Many manufacturers have only one or two types of counters but they can be used to count up, down or both. *Confused yet?* Can you say "no standardisation"? Don't worry, the theory is all the same regardless of what the manufacturers call them. A counter is a counter is a counter...

To further confuse the issue, most manufacturers also include a limited number of high-speed counters.

High-speed Counter :

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

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

To use them we must know 3 things:

Where the pulses that we want to count are coming from. Typically this is from one of the inputs.(a sensor connected to input 0000 for example)

How many pulses we want to count before we react. Let's count 5 widgets before we box them, for example.

When/how we will reset the counter so it can count again. After we count 5 widgets lets reset the counter, for example.

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

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

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



In this counter we need 2 inputs.

One goes before the reset line. When this input turns on the current (accumulated) count value will return to zero.

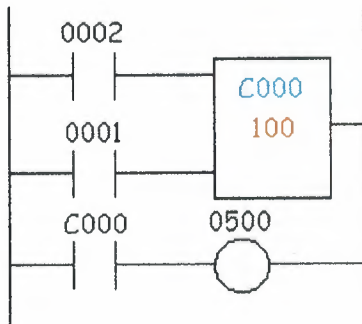
The second input is the address where the pulses we are counting are coming from.

For example, if we are counting how many widgets pass in front of the sensor that is physically connected to input 0001 then we would put normally open contacts with the address 0001 in front of the pulse line.

Cxxx is the name of the counter. If we want to call it counter 000 then we would put "C000" here.

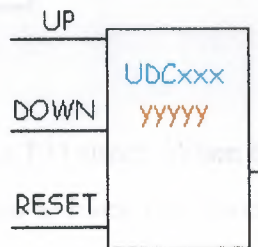
yyyyy is the number of pulses we want to count before doing something. If we want to count 5 widgets before turning on a physical output to box them we would put 5 here. If we wanted to count 100 widgets then we would put 100 here, etc. When the counter is finished (i.e we counted yyyyy widgets) it will turn on a separate set of contacts that we also label Cxxx.

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

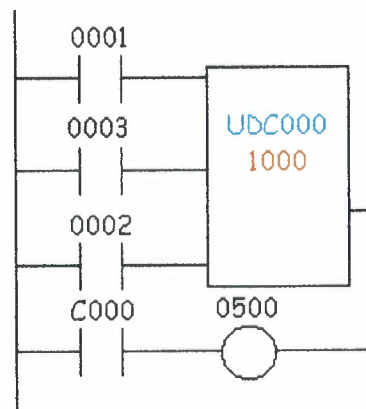


Here's the symbol on a ladder showing how we set up a counter (we'll name it counter 000) to count 100 widgets from input 0001 before turning on output 500. Sensor 0002 resets the counter.

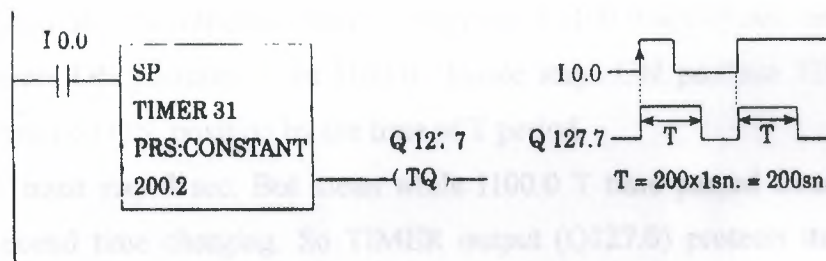
Below is one symbol we may encounter for an up-down counter. We'll use the same abbreviation as we did for the example above.(i.e. UDCxxx and yyyyy)



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



Siemens Simatic : Pulse Timer (SP)



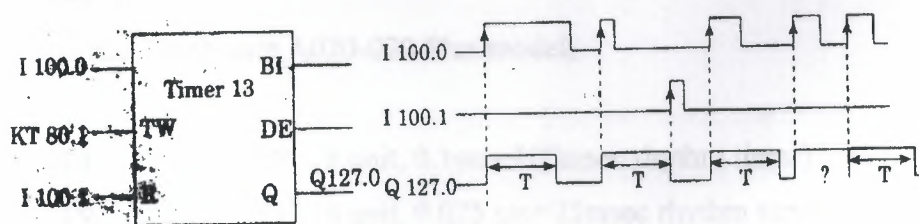
10.0 input sensor works T31 timer. When this sensor takes ON position, settled till 200 sec, Q127.7 out put done 1. Even time over, if input signal I0.0 logic 1, output will reset.

```

: A      I      0.0
: L      KT     200.2
: SP     T      31
: =      Q      127.7
: BE

```

Extended Pulse Timer



```

: A      I      0.0
: L      KT     100.2
: SD     T      12
: A      I      0.1
: R      T      12
: A      T      12
: =      Q      100.0
: BE

```

This kind of timer controls I100.0 input sensor 13 numbered TIMER. When I100.0 sensor was made 1, the sensor which was obliged Q127.0 numbered TIMER pass ON position. The important event is the pass of I100.0 to ON position not the time of this sensors ON position. Even I100.0 1msec stays ON position TIMER protects Q127.0 sensor on ON position by the time of T period.

T must stay 8 sec. But mean while I100.0 T time passed from logic 0 to 1 without second time charging. So TIMER output (Q127.0) protects its ON position again. But it returns beginning again to count from 0, of the T time.

```

: A      I      100.0
: L      KT      80.1
: SE     T      13
: A      I      100.1
: R      T      13
: A      T      13
: =      Q      127.0

```

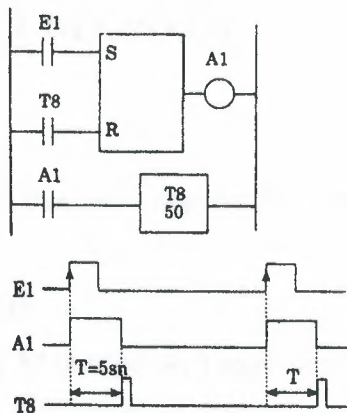
AEG

In the Teachware A020-020 Plus model;

T1.....T8 (8 unit, 0.1sec=100msec rhythm timer)

T9.....T16 (8 unit, 0.025 sec=25msec rhythm timer)

In order to 16 unit (T1.....T16) TIMER there are so programs be smallest and biggest time value is 25 msec which is 110 minutes.



1	U	E1
2	SL	A1
3	U	T8
4	RL	A1
5	U	A1
6	=	T8
7		50
8	PE	

In this example A1 is setted with E1 output. Reset position is the time of, when T8 pass ON position.

When E1 pass ON position A1 output makes set. By the setting of A1,T8 timer (present value $50 \times 0.1\text{sec} = 5\text{sec}$) count in its inside 5sec and at the end of this time logic done 1. As to program; when T8 is on,A1 output makes reset and T8 output goes OFF position because T8 output is armed reset sensor. The event to care on TIMER present value; chosen TIMER's rhythm times by its number, because of its changes, present value must count right.

The program on above; 413 numbered input sensor and M73 numbered private internal sensor are used to reset 467 numbered counter. Counting input is controlled by 412 numbered input sensor. Present value of counter is showed with K20-20. The input of counter pulse's every present pulse value is lowered 1 degree.

2.11. SHIFT REGISTER

IDEC

This model in PLC shift register unit has studied extensively.

MITSUBISHI

Internal relay M is used shift register at the some time. So 16 sensor must be 1 group at the same time First helping sensor number, shift register address and following 16 sensor can not use another arm.

Shift Register Addresses

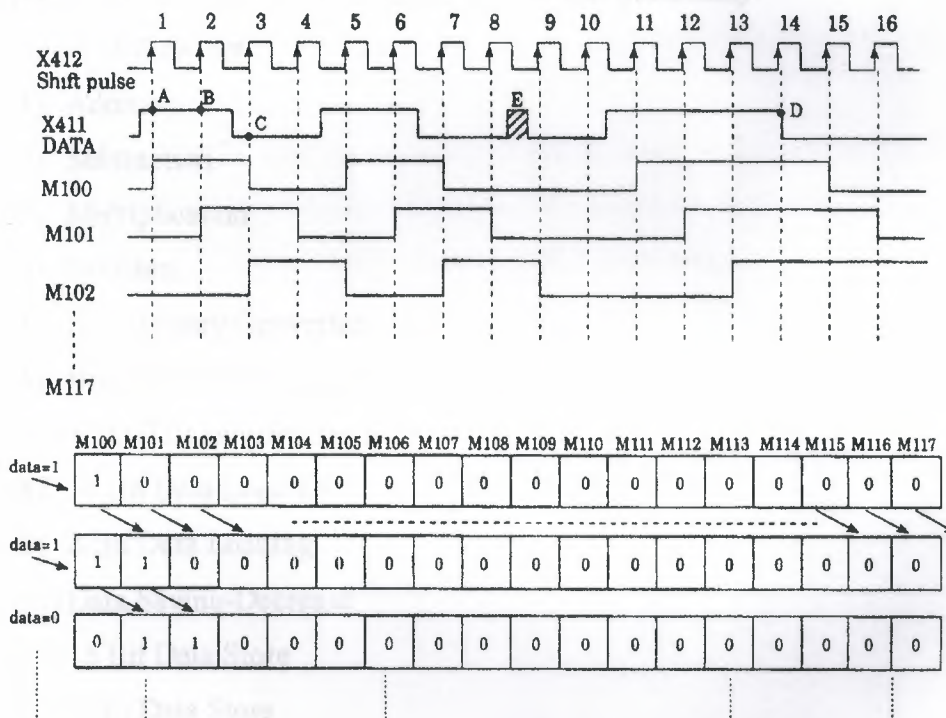
M100 – M117	=	M100.....M107, M110.....M117	= 16 unit
M120 – M137	=	M120.....M127, M130.....M137	= 16 unit
M140 – M157	=	M140.....M147, M150.....M157	= 16 unit
M160 – M177	=	M160.....M167, M170.....M177	= 16 unit
M200 – M217	=	M200.....M207, M210.....M217	= 16 unit
M220 – M237	=	M220.....M227, M230.....M237	= 16 unit
M240 – M257	=	M240.....M247, M250.....M257	= 16 unit
M260 – M277	=	M260.....M267, M270.....M277	= 16 unit
M300 – M317	=	M300.....M307, M310.....M317	= 16 unit
M320 – M337	=	M320.....M327, M330.....M337	= 16 unit
M340 – M357	=	M340.....M347, M350.....M357	= 16 unit
M360 – M377	=	M360.....M367, M370.....M377	= 16 unit

X413	DATA	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
X413	PULSE	100	101	102	103	104	105	106	107	110	111	112	113	114	115	116	117
X413	RESET																

1-Data input: Data signal which must be given to Register, is designed ON-OFF position to X411 sensor. Data, entered to register, firstly apply to M100 register. But every shift operation can make by shift pulse.

2-Shift pulse: It is shift input which is transferred to M100 by X411 entered data but while X412 is passing from 0 to 1. It can be used 72 numbered which produces 100msec time pulse or 73 numbered which produces on msec time pulse generator instead of X412.

3-Reset input: X413 input sensor is used for reset of the above. So all the register sensor with X413's passing OFF position to ON position makes reset and pass of position (M100.....M117).



(1100110000111100) data is applied with X411 data input on the above example. In here the important thing is decisive position of data in the shift pulse time. For example, 1 data's is in A point 1 data's is in B point 0 data's is in C point examples.

Decisive position in D point is 1, because while shift pulse going from 0 to 1; data value stayed decisively periods 1 pulse time in ON position, so D point of data's the time of going from 1 to 0, shift pulse which is still formed, can't catch and it can't be seen and examples the time of going from 1 to 0 of 14 pulse.

If you attend E area of data diagram; it can't be exemplified by data which is between 8 and 9 pulse and it doesn't accept like this data. According to this, for to load of data's to registers is the time of passing the time of piece of referans shift pulse (OFF→ON)

2.12. COMPUTING FUNCTION

one of the most important peculiarity of PLC system is computing and data embroidery function. As a main structure, PLC has this peculiarity.

Some of these are:

- 1) Addition
- 2) Subtraction
- 3) Multiplication
- 4) Division
- 5) BCD Binary Converter
- 6) BINARY BCD Converter
- 7) 4 DIGIT Comparation
- 8) 16 Bit Data Loading
- 9) 8 Bit Data Loading
- 10) Data Saving-Decrease
- 11) 16 Bit Data Store
- 12) 8 Bit Data Store
- 13) Data Display
- 14) BCD Shifting Left
- 15) Data Shifting

SIEMENS (Simatic) Comparison Function

In comparison operations:

!=F (equal)

≠F (not equal)

>F (big)

>=F (big equal)

<F (small)

<=F (small equal)

Instructions are used to make desired comparison, and if YES decision is reached, Q output will give ON position. >F control was done at the above. According to this, IB0 value which is in ACCU2 will be compared with IB1 in ACCU1, if $ACCU2 > ACCU1$, Q100 will remove ON position. If this condition is not provided, Q100 will stay OFF position.

Arithmetically +F instruction will provide addition of 2 complete numbers. This instruction adds ACCUM1 and ACCUM2, or for -F instruction subtracts the 2 numbers.

From ACCU2's contents will subtract ACC1's contents.

LD	Load
SPR	Shift Register
END	End
SET	Set
RES	Reset
MOV	Move
SWP	Swap
STC	Set Carry Flag
CLC	Clear Carry Flag
ST	Store
LD	Load

CHAPTER III

DETAIL ANALYSIS OF PROGRAMMING

3.1 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; SFR, SFR 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 (Except FUN 147, FUN 300). As for second addresses are present peculiarity according to instruction.

There are some deliver numbers that referenced by FA1J at the below.

c) Input:

0.....7, 10.....17, 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).

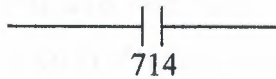
e) Internal Relay:

400 – 407	490 – 497	580 – 587
410 – 417	500 – 507	590 – 597
420 – 427	510 – 517	600 – 607
430 – 437	520 – 527	610 – 617
440 – 447	530 – 537	620 – 627
450 – 457	540 – 547	630 – 637
460 – 467	550 – 557	640 – 647
470 – 477	560 – 567	650 – 657
480 – 487	570 – 577	660 – 667

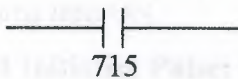
There are 240 units ($30 \times 8 = 240$) internal relays between 400 and 697, we can appoint the TIMER, COUNTER or FUN outputs to the any of 240 sensor and then can use of this sensor for take new data or count value.

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 and 9. You can use any of TIMER that include 0 and 79. In there its enough to know for you that 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 45-46 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 functions between 0 and 95 including 8-9.

1) Data Register:

Between DRO and DR99 and between 800 and 899, we have 100 data register.

3.2 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 Start 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 1 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 Carry and Borrow: If there is carry or borrow in the results at computing instructions. 707 is set for example; in an 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 FA1J 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.

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 except 8 and 9 in the example 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 need 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.

e) Timer:

I wanted to use T8 timer from the 80 timers between 0-79 including 8 and 9 here and 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 same 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 1 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 in the other situations it becomes OFF in logic. In a multiplying processes both inputs are 1 then output is 1. And had it ended with 2 limit switches and 1 solenoid valve in order to understand the logic better. In diagrams, it is stated as relay ladder diagram and logic diagram. So we can tell that LS1 relay A and LS2 relay is B input and output is Y. In such equality it is that $Y = A \cdot B$ 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 see this in the table of truth.

As known, the series of TTL is Logic integrate containing 4 and gate with 2 inputs in 7408. As in the circuit $\frac{1}{4}$ 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 only one of the keys are OFF or 1 is enough 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. I gave 2 important information's with or instruction. One as them is out function that is symbolised with 200 in the circle. I will speak about out function 2 or 3 classes later. But now, I gave output of parallel circuit, output 200 for the first time it means that: I mentioned that special relay 704 is a clock pulse generator that has $f=1$ Hz. You see signal of clock pulse in the diagram. We determined time of 1 and 0 in input relay of 36 by chance now so that nothing will be by chance in the following lessons. Let's accept that there is a time diagram for to learn or let's assume that input 36 is gained by making ON/OFF in the form. If we think that output 200 is connected to a lamb, the situations 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 ladder diagram and a solenoid valve control in output of it. And same circuit has been gained Logic equality by using only 1or gate of integrate of 7432. It is enough to make on only one of the inputs for the outputs to be ON in 3 equaliavence circuit to make output OFF. It is necessary 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 1 from the outside, the sign is going to continue from B point as logic 0, because of the instruction of LOD NOT 1.

NEAR EAST UNIVERSITY



Faculty of Engineering

**Department of Electrical and Electronic
Engineering**

**FULLY CONTROL A PARKING AREA WITH
(PLC)**

**Graduation Project
EE 400**

Student: Ömer Doğan (960355)

Supervisor: Mr. Özgür C. Özerdem

Lefkoşa- 2001

CONTENTS

ACKNOWLEDGEMENT	i
ABSTRACT.....	ii
INTRODUCTION.....	1
CHAPTER I	3
1.1. THE TYPES OF PLC.....	3
a) PLC Size And Practise.....	4
b) I/O Unit.....	4
c) Different I/O Units.....	6
d) Analog Input/Output Unit.....	11
e) CPU.....	11
f) Processor Memory Module.....	12
1.2. MEMORY DESIGN.....	12
a) I. Group Memories.....	13
b) II. Group Memories.....	13
1)PROM	
2)EPROM	
3)EAROM	
4)EEPROM	
1.3. PROGRAMMING DEVICES.....	14
CHAPTER II	16
2.1. PLC PROGRAMMING SOFTWARE.....	16
2.2. CREATE OF LADDER DIAGRAM.....	16
a) Start Command.....	16
b) AND and OR Exercising.....	18
c) Output Stored Exercises.....	18
2.3. SPECIFICATION OF EXAMINED PLC.....	19
a) Mitsubishi F1 20 MR.....	19
b) Siemens Simatic S5-90U.....	20
c) AEG Teachware Modicon A020.....	20
d) FESTO (FPC 202C).....	21
2.4. CREATING COMMAND LINE FOR LOGIC PROCESS.....	22
a) Loading Of Close And Open Contact.....	22
b) AND Exercise.....	24
c) AND NOT Exercise.....	24
d) OR Exercise.....	25
e) OR NOT Exercise.....	25
2.5. GET COMMUNICATE OF COMMAND BLOCK TOGETHER.....	26
a) Serial Contact.....	26
b) Parallel Contact.....	26
2.6. SET AND RESET INSTRUCTION.....	27
2.7. SINGLE OUTPUT INSTRUCTIONS.....	29
2.8. JUMP INSTRUCTION.....	29
2.9. TIMERS.....	30
2.10. COUNTERS.....	33

2.11. SHIFT REGISTER.....	40
2.12. COMPUTING FUNCTION.....	42
SECTION III	44
3.1. BASIC INSTRUCTION WORD.....	44
a) Basic Instructions.....	44
b) FUN (Function) Instructions.....	44
c) Input.....	45
d) Output.....	45
e) Internal Relay.....	45
f) Special Internal Relay.....	46
g) Timer.....	46
h) Counter.....	46
i) Reversible Counter.....	46
j) Shift Register.....	47
k) Single Output.....	47
l) Data Register.....	47
3.2. FA1J SERIES ALLOCATION NUMBERS OF SPECIAL RELAYS.....	47
3.3. BASIC INSTRUCTION.....	48
a) LOD Instructions.....	48
b) Input, Output, Internal and Special Relays.....	48
c) Timer.....	49
d) Counter.....	49
e) Shift Register.....	49
f) AND Instruction.....	49
g) OR Instruction.....	50
h) NOT Instructions.....	50
SECTION IV.....	51
4.1. CHOOSING INSTALLATION AND COMMISSIONING OF PLC SYSTEM.....	51
4.2. FEASIBILITY STUDY.....	51
4.3. DESIGN PROCEDURE FOR SYSTEM.....	52
a) Choosing Programmable Controller.....	52
b) Size and Type of PLC System.....	53
c) I/O Requirements.....	53
d) Memory and Programming Requirements.....	54
e) Instruction	57
4.4. INSTALLATION.....	57
4.5. TESTING AND COMMISSIONING.....	59
4.6. a) Soft Ware Testing and Simulation.....	60
4.6. b) Installing and Runningn The User Control Program.....	64
MY PROJECT.....	65
CONCLUSION.....	69
TABLE OF SYMBOL.....	70
APPENDIX	71
REFERENCES.....	100

ACKNOWLEDGEMENTS

I must first acknowledge some debts from my more distant past. To my parents for their support to an upwardly mobile son. To my friend Murat Muhammet GÜVEN, Mehmet KİNSİZ, for his helps.

I want to thank separately to my teachers that during my education they support me, the Dean of Engineering Faculty Prof. Dr. Fakhreddin MAMEDOV, Head of Electrical & Electronic Department Prof. Dr. Fakhreddin MAMEDOV and also Mr. Özgür ÖZERDEM, Mr. Kaan UYAR, Assis. Prof. Dr. Kadri BÜRÜNCÜK.

And finally, I want to thank everybody who helped and supported me to come to today and graduated from a university.

ABSTRACT

PLC (Programmable Logic Controllers) is a thing that programmable with computer support to take more efficiency from time and workers. It is divided into two parts. Hardware and software.

The hardware are the parts of machine those are CPU, I/O device and Programming device. CPU is basic microprocessor system and it carries out as control sensor, counter, timer function. CPU carries out stored user program in memory will input informations from various sensor circuits and can sending suitable output to commands and control circuits. I/O Module receives 120 VAC signal in device or processing device and transforms 5 VDC signal form.

There are many specialisation such as timer, counter, master control set, which works data and controls program, master control reset, JMP. There are command which are mathematics process that are comparator processes. These are the main function and feature of software part of PLC.

INTRODUCTION

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

When production requirements changed so did the control system. This becomes very expensive when the change is frequent. Since relays are mechanical devices they also have a limited lifetime which required strict adherence 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 PLCs. Conventional microprocessors lacked the power to quickly solve PLC logic in all but the smallest PLCs. As conventional microprocessors evolved, larger and larger PLCs were being based upon them. However, even today some are still based upon the 2903. Modicon has yet to build a faster PLC than there 984A/B/X, which was based upon the 2901.

Communications abilities began to appear in approximately 1973. The first such system was Modicon's Modbus. The PLC could now talk to other PLCs and they could be far away from the actual machine they were controlling. They could also now be used to send and receive varying voltages to allow them to enter the analogue world. Unfortunately, the lack of standardisation coupled with continually changing technology has made PLC communications a nightmare of incompatible protocols and physical networks.

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

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

CHAPTER I

1.1. THE TYPES OF PLC

In general, PLC divides to three sections;

- *Central Processing unit(CPU)
- *The input/output section
- *The programming device

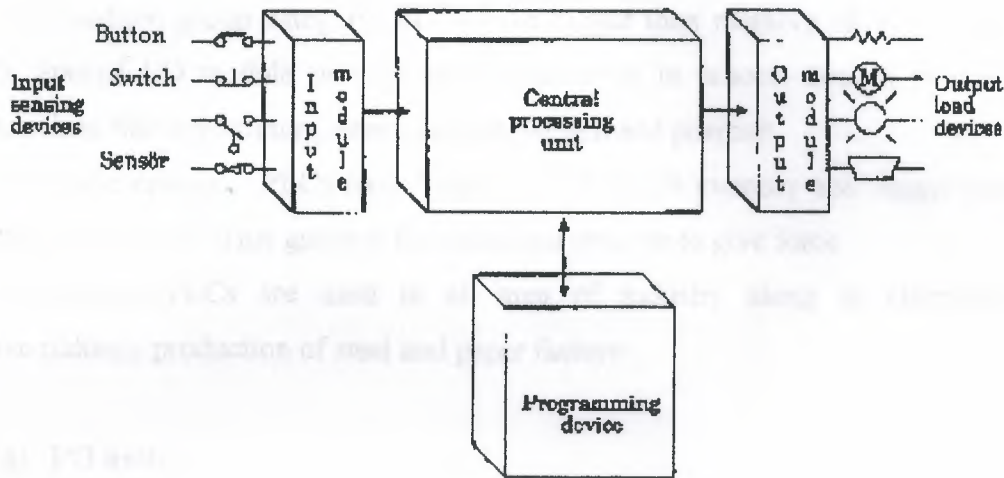


Figure.1.1.1. PLC sections

(CPU), PLC system and there are various logic circuit gates. CPU is basic microprocessor system and it carries out as control relay, counter, timer functions. CPU carries out user programs stored in memory and read input data from various sensor circuits and can send suitable outputs to commands and to control circuits.

Direct current power supply must be used for the low level voltage that these are using in processor and I/O models. This power supply is a part of CPU. PLC system is independent in its structure and also it can be dependent to its system.

I/O system forms can be connected to controller by other devices. The aim of interface is to send various signals and to take situations to external devices. The output devices for example, motor starters, solenoid valves, indicator lights connected to terminals on the output module.

The desired program loads to processor's memory by programming device or terminal. This program can enter to relay during using ladder logic. Program can be obtained till the main control or machines by sequential processes.

a) PLC size and practice:

There are 3 different categories of PLC; as small, medium and large.

*In small group category, PLC has bigger than input/output of 128 I/O and bigger than memory of 2 KB.

*In medium group category, PLCs have bigger than memory of 32 KB and 2048 I/O. Special I/O module provide easily adaptation in process control practice, analog functions like temperature, press, current, weight and position.

*In large category, PLCs have bigger than 750 KB memory and bigger than input/output of 8192 I/O. This group is for unlimited practice to give force.

Nowadays, PLCs are used in all area of industry along in chemistry, automotive industry production of steel and paper factory.

b) I/O unit:

I/O unit forms is the input/output rack of PLC. I/O unit receives 120 Vac signal in device or processing devices and transforms 5 Vdc signal form. In output units controller signals (5Vdc) are used to devices or processor control as 120 Vac. These output signals provide low current control that used in power electronic elements or optic isolators. Input/output unit in PLC can be put in the same structure or different structure with CPU. This standard input/output unit is in the following shape.

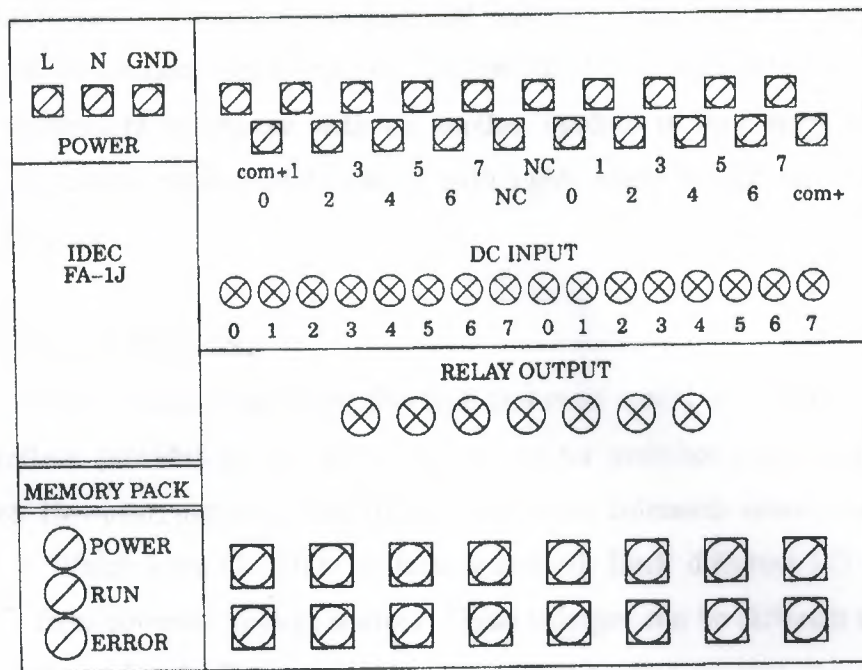


Figure. 1.1.2. In the same structure CPU with PLC I/O unit

Between processor and I/O rack communication different connection cables are permitted. This condition is as the following figure 1.1.3.

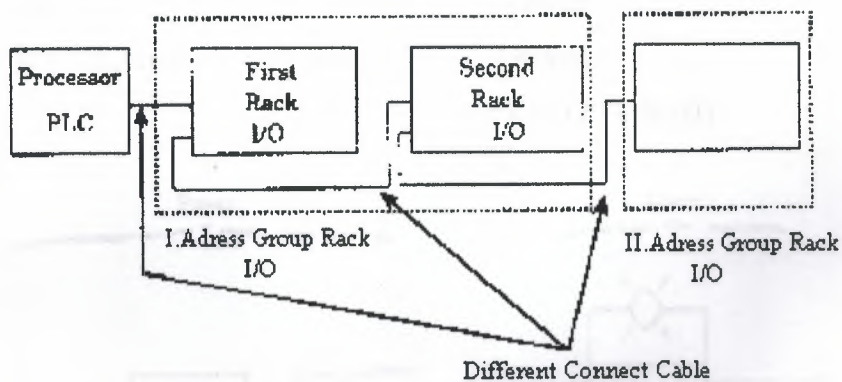


Figure. 1.1.3 Between Processing I/O Racks communication

I/O units each input/output has a special address. These addresses are known by the processor. To connect output/input an element with I/O or separating is very easy and quick. Furthermore to change with an another module is very easy. ON/OFF condition of I/O circuit each module shows with light. Many output modules have rubbish fuse indicator.

c) Different I/O units:

Many output I/O units are from this type and most useful is interface module. This type interface provides to link of inputs as selector switches push buttons and limits switches. However, output control lights small motor solenoids sensor and motor starters limit it. Which have ON/OFF contacting control. Each different I/O module takes its power from common voltage sources. These voltages can be different size and type. These are showed in the following table.

<u>Input Interface</u>	<u>Output Interface</u>
24Vac/dc	12-48Vac
48Vac/dc	120Vac
120Vac/dc	230Vac
230Vac/dc	120Vdc
5Vdc (TTL level)	230Vdc
	5Vdc (TTL level)

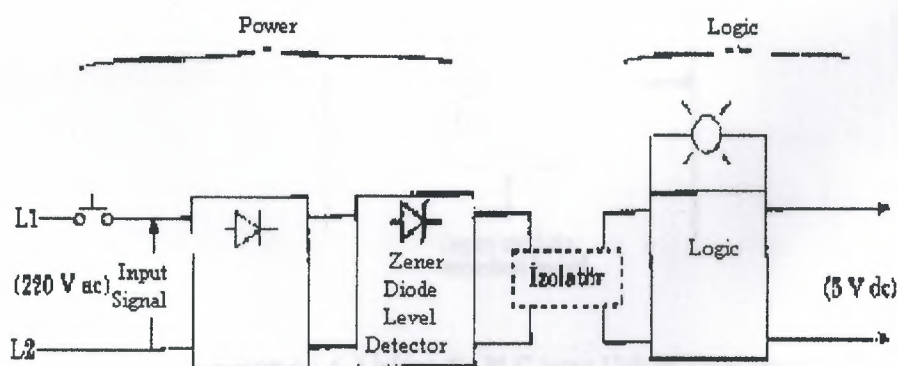


Figure.1.1.4 AC input interface block diagram

Shows that entries block diagram for an alternative current to input module.
Input circuit compose of to main section as power and logic section.

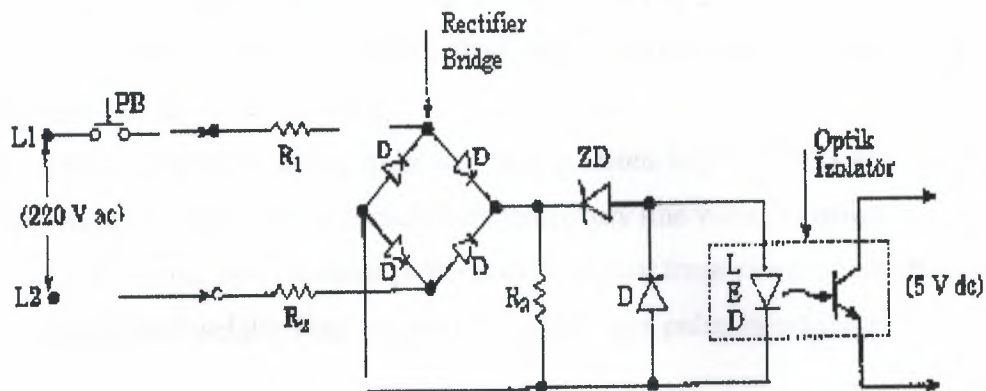


Figure.1.1.5. Simplified Circuit For a AC Module

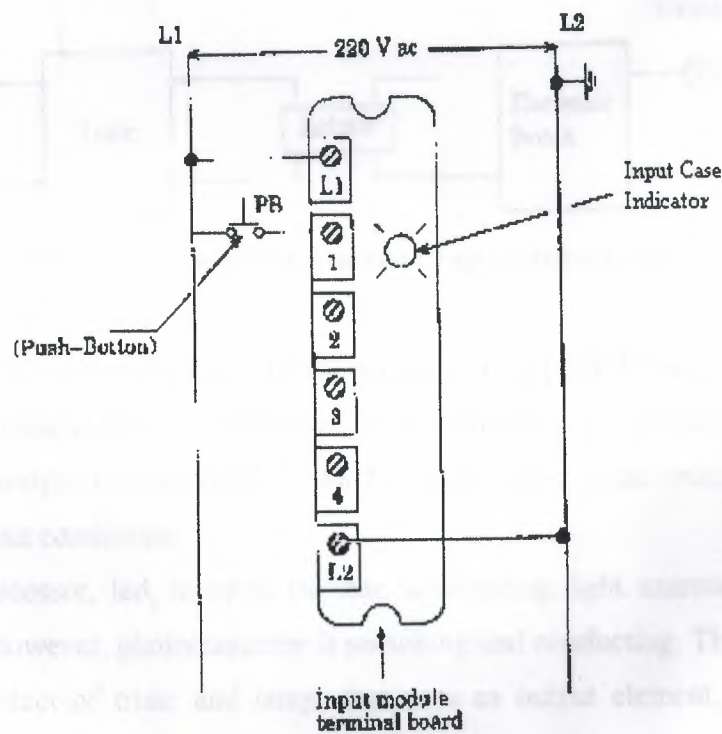


Figure.1.1.6. Linking To PLC Input Unit of 220V Input

Figure 1.1.4 and 1.1.5 shows figural diagram of Ac input module for input,
also figure 1.1.6 shows connect terminal.

When push button shuts down, bridge type treatment exercise 220V AC voltage from R_1 and R_2 resistance's.

Zener diode (ZD) voltage limit regulates according to low level voltage.

When light come to processor from led with phototransistor that means low level voltage (5V' dc) is transmitted.

Optic isolator separates high AC voltage from logic circuits also protects to processor from damages, which comes from temporary line voltage change.

Furthermore, optic isolator protects to processor from effect of electrical noise.

Kuplaj and isolation can be created with using a pulse transformation.

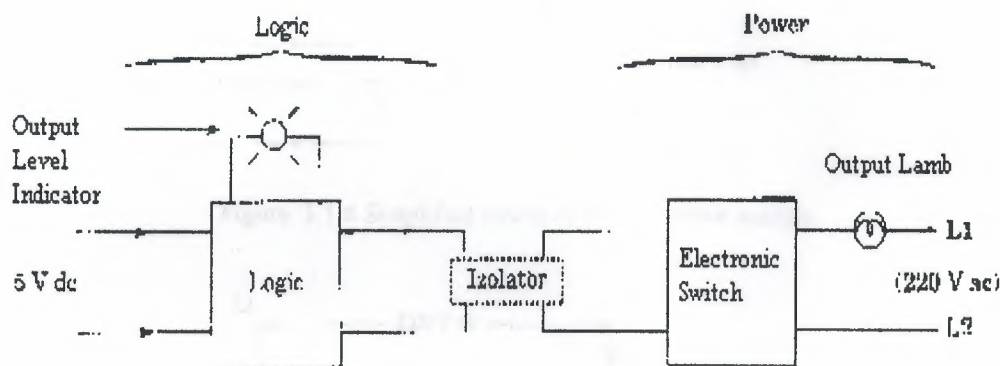


Figure. 1.1.7 typical a block diagram of output interface module.

Figure. 1.1.7 shows typical a block diagram of output interface module. Also output module, as input module, composes of two departments such as power and logic.

Device in output is controlled by the 5V comes from logic unit. In this unit, processor sets output conditions.

When processor, led, in optic isolator, distributing light exercises an output voltage (5V' dc), however, phototransistor is switching and conducting. This means that to detect and conduct of triac, and lamp, that uses as output element, turn on ON condition.

When led in logic unit turn off, logic become 0 condition and phototransistor cannot conduct. If a DC device in output will be controlled, it is carried with circuit.

PLC device will not be damaged from optic isolation that will be from power department.

If many high fast ON-OFF is necessary, in right current transistor and also alternative current triac circuits are used. Current cannot pull on PLC from output modules. Maximum current capacity of each device exists in their catalogs of that model.

In high currents instead of triac or other effect elements, standard relay must use as table 6. There are output/input unit as analog/digital translator (ADC) and digital/analog translator (DAC) that it is necessary for feedback control exercises in PLC devices.

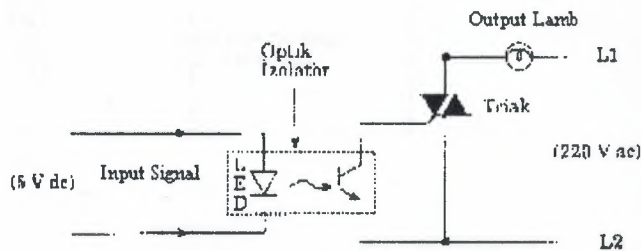


Figure. 1.1.8 Simplified circuit of an AC output module.

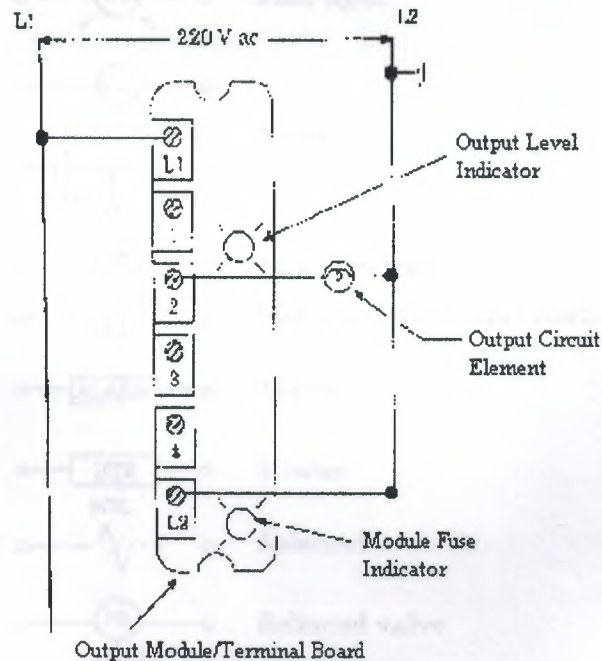


Figure. 1.1.9 Internal wire connection typical an output module

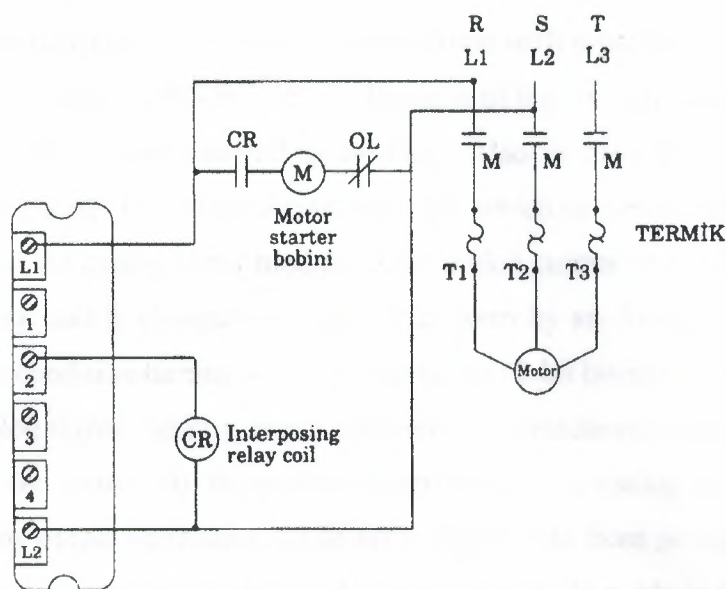


Figure. 1.1.10 Sensor connection points

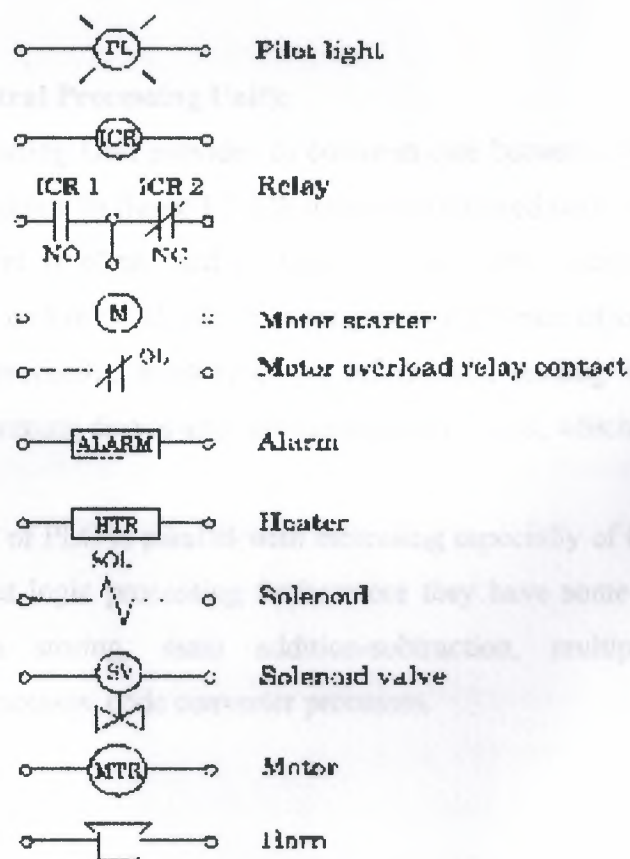


Figure. 1.1.11 Symbols of output control circuit

d) Analog input/output unit (I/O modules):

First produced PLCs only had been limited with separate I/O interfaces which had been allow to link to ON/OFF device. Because of this limitation many of processing exercises could be as part controlling by PLC. Also in days PLCs included analog interface and separate (I/O) input/output interface, which carries out practically many of control process. An analog input module takes analog current and voltage that is taken off analog input and it changed to digital data form by an Analog Digital Converter (ADC). In this condition turning levels are shown as 12-bit binary or 3 digit BCD that is rates with analog signal. Analog sensor elements are transducers as heat, light, velocity, pressure, and wet sensors. All these sensors can be linked to analog input

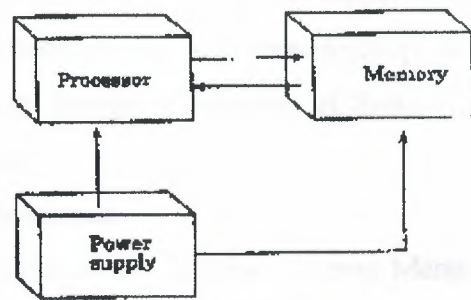
Analog output interface module takes digital data from processor, charges rate with voltage and current and controls a device as analog. As a whole digital data passes from Digital/Analog output device are small motors, valves and analog measure devices.

e) CPU (Central Processing Unit):

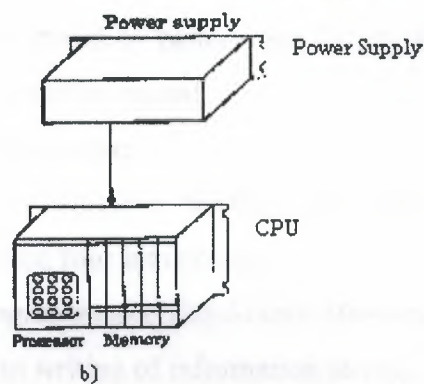
Central Processing Unit provides to communicate between power supply and processor memory modules. In figure 1.2.12b it can find covered both of two units.

CPU statement is often used as mean of processor statement. Processor-memory creates a big unit of CPU, which is programmable brain of controller. In this unit, there are microprocessor, memory chips, information reading and request data from memory, programming device and communication circuits, which is necessary for processor.

Development of PLC is parallel with increasing especially of CPU. In our day PLC systems carry out logic processing furthermore they have some especially such timer, counter, data storing, main addition-subtraction, multiplication-division processes, compare processes, code converter processes.



a) Simplifies of CPU Structure



b)

Figure. 1.1.12 CPU; the elements of central processing unit (a) the structure of simplified CPU
(b) power supply unit different from CPU.

f) Processor-Memory Module:

CPU is the brain of programmable of controller and a big part of CPU family forms from processor memory unit. This module cover microprocessor, memory chips programming device and necessarily communication circuits for processor interface.

Furthermore processor carries out other functions. For example, it carries out timer, counter, compare, keeper and addition, subtraction, multiplication and division functions, which are four main functions of mathematics.

1.2. 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, 1024byte = 1kbyte and the number of memory capacity is stated these units.

The memory types are divided into two groups;

The first group: the energy of power supply is cut that supplied memory, it means that memory had been erased. Also second group: hide information cannot lose if the energy is cut. But to change of includes of those types of memories, there is a necessary a special system.

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;

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.

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

3) EAROM (Electrically Alterable 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.

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.

1.3. PROGRAMMING DEVICES

The most important one of features of programmable controller is to have programming elements, which are useful. Programming device provides transformation between operator and circuit of controller. (Fig. 1.3.1)

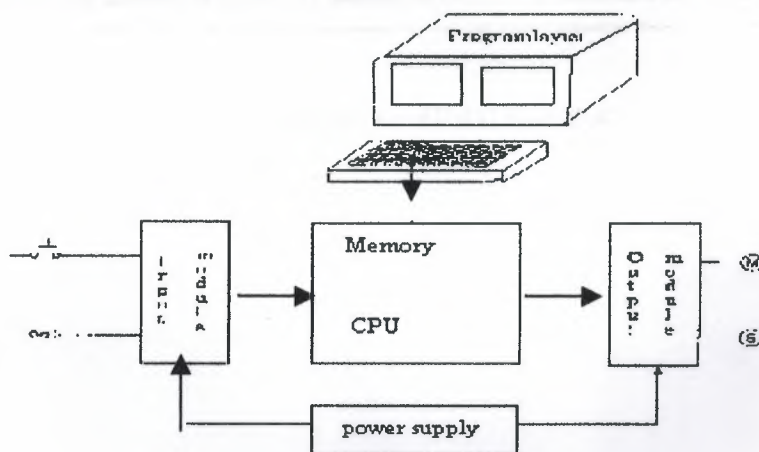


Figure. 1.3.1. Transformation of PLC Circuits

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 communicate 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 monitor 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 functions keys. FA2 of programming device IDEC FA1 Junior module is shown at table 1.3.2.

FA-2 PROGRAMMABLE CONTROLLER							
963 LOD T 20				PROGRAM LOADER			
ADRS	TIM	CNT	SFR	MCS 7	JMP 8	PROM 9	INST
DELT	SET	RST	END	MCR 4	JEND 5	CMT 6	↓ VERI
MON	OR	SOT	OUT	F147 1	F247 2	3	↑ READ
TRS	LOD	AND	NOT	0	FUN	CLR	ENT'R

Figure. 1.3.2. Programming Device of IDEC FA-1 PLC.

CHAPTER II

2.1. PLC PROGRAMMING SOFTWARE

In this section, PLC programming fundamental is prepared, student's capacity, which met PLC programming, is considered first time.

AND

OR

NOT

NAND

NOR

SET

RESET

Furthermore there are many specialisations such as TIMER, COUNTER, and MASTER CONTROL SET (MCS), which works data and controls PROGRAM, MASTER CONTROL RESET (MCR), JMP. There command which are mathematics process that are comparator processes ($=$, $<$, $>$).

In all PLC systems, to create logic process is programmed as the same are carried out some function. However, the main logic is the same that TIMER, COUNTER and SHIFT REGISTER functions are to get command and programmed but there can be some differences.

2.2. CREATE OF LEADER DIAGRAM

a) Start Commands:

These commands are first element of program. There are two type contact conditions as at table 2.2.1. First normally is open also second close.

Normally, starting with open contact this program command is to get command as LD IN, LD, LOD A, on PLC device. And also close contact is stated as LDI, LD NOT, LOD NOT, AN.



LADDER SYMBOL	COMMAND LINE					
	IDEC	FESTO	AEG	Mitsubishi	Siemens	OMRON
 F Normally open contact	LOD F	LD FLAG F LD IN F	UF	LD F	A F	LD F
 F Normally close contact	LOD NOT F	LD NOT FLAG F LD NOT IN F	UN F	LDI F	AN F	LD NOT F

Table 2.2.1. Load Exercising

Note: in table F value is constant and input/output interval relay, special relay, timer, counter can be SFR number.

According to this table at MITSUBISHI and HITACHI model normally open contact is shown with LD, also close contact is shown with LDI.

Also at AEG PLC, U (UND) command is used for open contact and (UN) UND-NICHT command is used for closed contact.

Also at SIEMENS PLC, A (AND) command is used to open contact and AN (AND-NOT) is used for closed contact.

At OMRON PLC, open contact is shown LD, also close contact is shown with LD NOT.

Also at FESTO PLC, open contact LD FLAG is used for flag load other conditions LD IN command is used to contact load. In normally, also close contact is programmed for flag exercising as LD NOT FLAG... For other contacts are programmed as LD NOT IN...

b) AND and OR Exercising:

LADDER SYMBOL	COMMAND LINE						
	IDEC	OMRON	AEG	MITSUBISHI	Siemens SIMATIC	OMRON	HITACHI
	LD X1 AND X2	LD IN X1 AND IN X2	U X1 U X2	LD X1 AND X2	AX1 AX2	LD X1 AND X2	LD X1 AND X2
	LD X1 AND NOT X2	LD IN X1 AND NOT IN X2	U X1 UN X2	LD X1 ANI X2	AX1 AN X2	LD X1 AND NOT X2	LD X1 ANIX2
	LD X1 OR X2	LD IN X1 OR X2	U X1 O X2	LD X1 OR X2	AX1 O X2	LD X1 OR X2	LD X1 OR X2
	LD X1 OR NOT X2	LD IN X1 OR NOT X2	U X1 ON X2	LD X1 ORI X2	AX1 ON X2	LD X1 OR NOT X2	LD X1 ORIX2

Table 2.2.2: Symbol and command line AND and OR exercises.

c) Output Stored Exercises:

At a PLC system relay, it is used as output function, can be divided into two groups. First group output which charge can be linked to it according to program as (solenoid valves, neon lamp, conductor, led, etc.) are real output. Also second group outputs are internal and image relays. Physical connection cannot link to these relays but outputs of these sensors are transferred to real output and output can be taken.

If commands will be observed, there are similarities between PLC devices that output program commands are different. At both output and input functions, X1, X2, are used as addresses.

LADDER SYMBOL	COMMAND LINE						
	IDEC	FESTO	AEG	MITSUBISHI	Siemens SIMATIC	OMRON	HITACHI
	LD X1 OUT X2	LD IN X1 = OUT X2	U X1 = X2	LD X1 OUT X2	A X1 = X2	LD X1 OUT X2	LD X1 OUT X2
	LD X1 SET X2	LD IN X1 SE FLAG X2	U X1 SL X2	LD X1 S X2	A X1 S X2	LD X1 SET X2	LD X1 SET X2

Figure.2.2.3.

2.3. SPECIFICATION OF EXAMINED PLC

a) Mitsubishi F1 20 MR

ELEMENT	Symbol	F1 20MR
(Inputs)	X	12 Unit 400 - 413
(Outputs)	Y	8 Unit 430 - 437
(Timer) 0.1 s	T	24 Unit 50 - 57, 450 - 457
(Timer) 0.01 s	T	8 Unit 650 - 657
(Counters)	C	30 Unit 60 - 67, 460 - 467
(Big speed counter)	C	2 Unit 660 - 661
(Internal Relay)	M	64 Unit 10 - 177
(Special Internal Relay)	M	16 Unit 70 - 77, 470 - 477, 570 - 575
Battery of Feeding Sensor	M	64 Unit 300 - 377
(Jump)	M	64 Unit 700 - 777

Table 2.3.1: table of element and element numbers

F1 10ER	
X	4 Unit 414 - 417
Y	6 Unit 440 - 445

Table 2.3.2. Increasing unit

F1 20 MR PLC as 12 inputs 8 outputs, which we use. If more input and output are necessary, input/output-increasing units are plugged to PLC. These units have various numbers output and input. At table 2.3.1, there are 4 inputs 6 outputs for F1 10 ER model.

b) Siemens Simatic S5-90U

Element Name	ELEMENT ADDRESS
(Input)	I0.0 – I127.7
(Output)	Q0.0-Q127.7
(Flag)	(retentive) F0.0 – F63.7
(Flag)	(nonretentive) F64.0 – F127.7
Accumulator	ACCUM1 ACCUM2
Timer	T0 – T31
(Counter)	(retentive) C0 – C7 (nonretentive) C8 – C31
KB	(Constant) 1 byte 0 – 255
KC	(Constant count) 0 – 999
KF	(Tam sayılar) - 32768 +32767
KF	(Heksadesimal) 0 – FFFF
KY	(2 byte) 0 – 255 (her bit)
KT	(Timer) 0.0 – 999.3
FB	(Function block) 0 – 63
DB	(Data block) 2 – 63 [9,10]

Table 2.3.3: Specifications of S5-90U model Siemens Simatic.

c) AEG Teachware modicon A020

Operand Type	Operand	Unit
(inputs)	E1 – E24	24
(outputs)	A1 – A16	16
Analog Input	EWA 1 – EWA 4	4 analog
Analog Output	AWA 1	1 analog
Memory	M1 – M128	128 Unit
Timer	T1 – T16	16 timer
Counter	Z1 – Z16	16 Counter

Table 2.3.4. Specifications of AEG Teachware A020

d) FESTO (FPC 202C)

TOTAL UNIT	PARAMETERS	SYMBOL	EXPLANATION
16	Internal inputs	I 0.X and I 1.X	input 0.0-0.7 1.0-1.7
2	Internal half-words	IW0 and IW1	2 Unit
16	Internal outputs	O 0.X and O 1.X	Output 0.0-0.7 1.0-1.7
2	Internal output half-words	OW0 and OW1	2 Unit
256	Flags	F0.Y to F15.Y	Flag: (0.0-0.15) (1.0-1.15) (2.0-2.15).....(15.0-15.15)
16	Flag words	FW0 to FW15	16 Unit Present
1	Initialization Flag	FI	1 — —
24	Special function units	FU0 to FU23	24 — —
16	Field bus flag words	FU32 to FU47	16 — —
32	Timers	T0 to T31	32 — —
32	Timer words	TW0 to TW31	32 — —
32	Counters	C0 to C31	32 — —
32	Counters words	CW0 to CW31	32 — —
32	Counters preset	CW0 to CW31	32 — —
64	Registers	R0 to R63	64 — —
8	programs	P0 to P7	8 — —
8	prog/function modules	B0 to B7	8 — —
1	Errors	E	1 — —
1	Error word	EW	1 — —
48	External inputs	I 2.X to I 7.X	input (2.0-2.7) (3.0-3.7)(7.0....7.7) = Top. 48
6	External input words	IW2 to IW7	6
48	External output	O 2.X to O 7.X	Output (2.0...2.7).. (3.0....3.7) (7.0....7.7)
6	External output words	OW2 to OW7	6

Table 2.4.8 Specification of FESTO (FPC 202C) Module PLC

In this table, $x=(0,1,2,3,4,5,6,7)$ and $y=(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)$ are.

2.4. CREATING COMMAND LINE FOR LOGIC PROCESS

Each process in PLC programming is stated by a command and these commands provides connections of relay and contacts together, designations of outputs, counter, programming of timers and making of arithmetic comparison processes.

In our days, to experience PLC device of all firms are very hard. We will experience five brands. These brands are enough for us.

BRAND**MODEL**

1) IDEC	FA1-JUNIOR (FA1J)
2) FESTO	202-C
3) MITSUBISHI	F120 R
4) SIEMENS-SIMATIC	S5-90U
5) AEG TEACHWARE	MODICON A020

a) Loading of Open and Close Contact:

Normally open contact

LOD (LOAD)-IDEC

LD IN (LOAD)-FESTO

LD (LOAD)- MITSUBISHI

A (AND)- SIEMENS-SIMATIC

U (UND)-AEG



Normally close contact

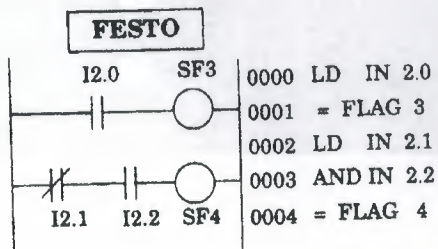
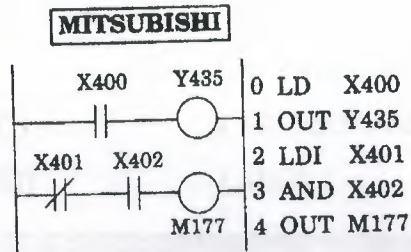
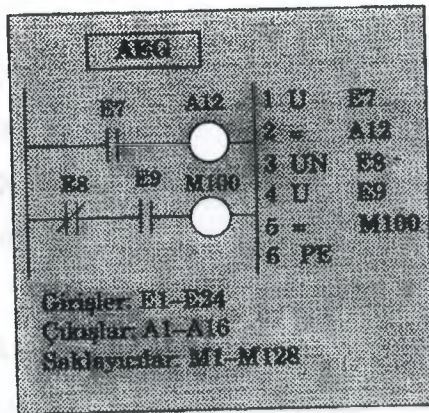
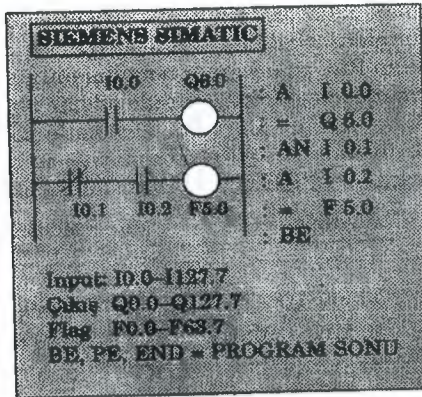
LOD NOT (LOAD NOT)-IDEC

LD NOT IN (LOAD NOT)-FESTO

LDI (LOAD INVERSE)- MITSUBISHI

AN (AND NOT)- SIEMENS-SIMATIC

UN (UND NICHT)-AEG



In here, commands for giving different brand and module normally. Explain to designation of contact and contact numbers are written after command.

In AEG and Siemens PLC, a load command is not used in Siemens Module, open contact command normally is load written A (AND), load process is relaying with AN (AND NOT) command.

In AEG module U (UND) and UN (UND NOT) commands are used for load process. As we know that these commands are used to serial AND and AND NOT exercises.

b) AND exercise:

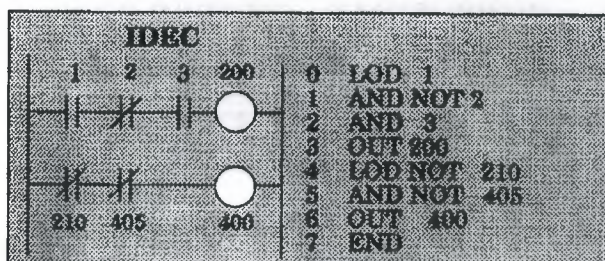
Serial contact linking commands

AND	-(IDEC)
AND IN	-(FESTO)
AND	-(MITSUBISHI)
A(AND)	-(SIEMENS-SIMATIC)
U (UND)	-(AEG)

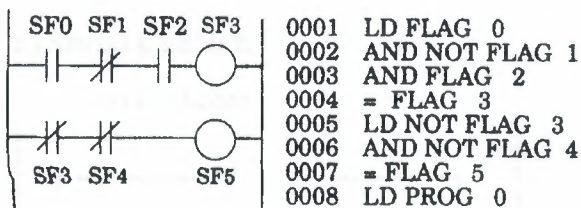
c) AND NOT exercise:

Serial contact linking commands

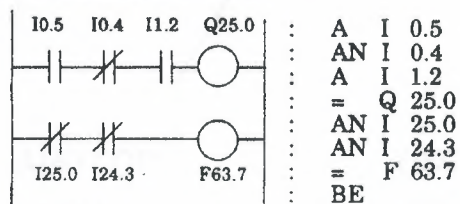
AND NOT	-(IDEC)
AND NOT IN	-(FESTO)
AND	-(MITSUBISHI)
A(AND)	-(SIEMENS-SIMATIC)
U (UND)	-(AEG)



FESTO



SIEMENS SIMATIC



d) OR exercise:

Parallel contact linking commands

OR	-(IDEC)
OR	-(FESTO)
OR	-(MITSUBISHI)
O(OR)	-(SIEMENS-SIMATIC)
O(ODER)	-(AEG)

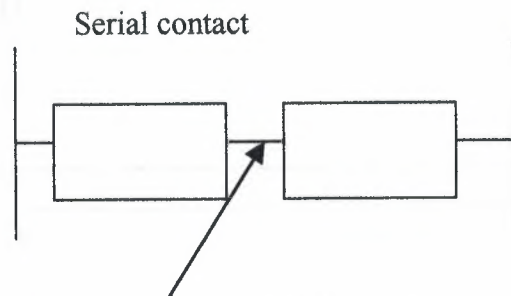
e) OR NOT exercise:

Parallel contact linking commands

OR NOT	-(IDEC)
OR NOT	-(FESTO)
ORI(OR INVERSE)	-(MITSUBISHI)
ON(OR NOT)	-(SIEMENS-SIMATIC)
ON(ODER NICHT)	-(AEG)

2.5. GET COMMUNICATE OF COMMAND BLOCK TOGETHER

a) Serial Contact:



AND LOD -(IDEC)

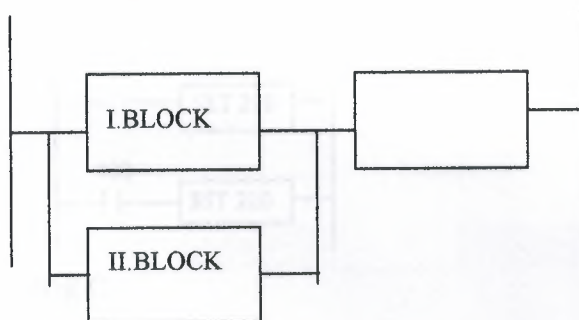
AND LD -(FESTO)

ANB (AND BLOCK) -(MITSUBISHI)

A(.....)	-(SIEMENS)
--------------	------------

U(.....)	-(AEG)
--------------	--------

b) Parallel Contact:



OR LOD -(IDEC)

OR LD -(FESTO)

ORB (OR BLOCK) -(MITSUBISHI)

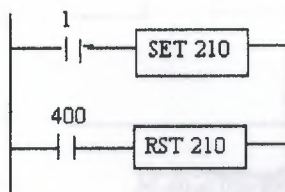
A(.....	-(SIEMENS)
O	
.....	
)	

O(-(AEG)
)	

2.6 SET AND RESET INSTRUCTION

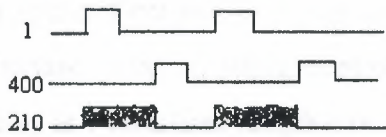
If any of the OFF position relay (eg. Input, output register and internal relay) pass the ON position that is from logic 0 to logic 1. Pass instruction called SET command. RESET command is opposite of SET command that is ON position to OFF position, from logic 1 to logic 0.

Another peculiarity of SET and RESET instructions for working instructions input must be control with relay. It does not require any continuous signal or stroke. That means SET relay always logic 1 position with input relay. If input relay done OFF position does not effect setted relay while that RESET command come.

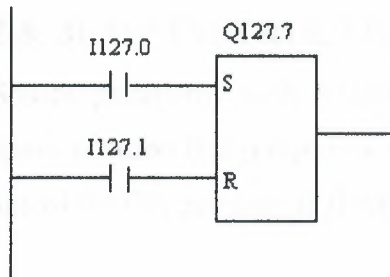


IDEC

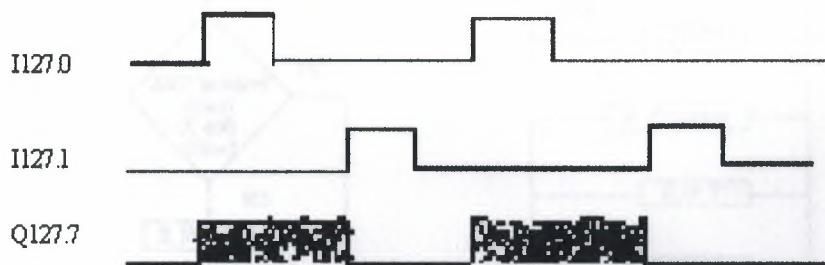
0	LOD	1
1	SET	210
2	LOD	400
3	RST	210
4	END	



SIEMENS



A I 127.0
 S Q 127.7 (Jump)
 A I 127.1
 R Q 127.7
 BE



2.7. SINGLE OUTPUT INSTRUCTIONS

Our aim is make ON position, on scan time length. With these aim we use two different relays. First one is which makes control, other one is where we take output. The important point is; while controlling relay passing OFF position to ON, where output relay is 1 scan time length mould pass ON position to OFF. It is unimportant that controlling relay is protecting ON position. When the OFF position relay pass to ON position, we take 1 scan time length from output relay.

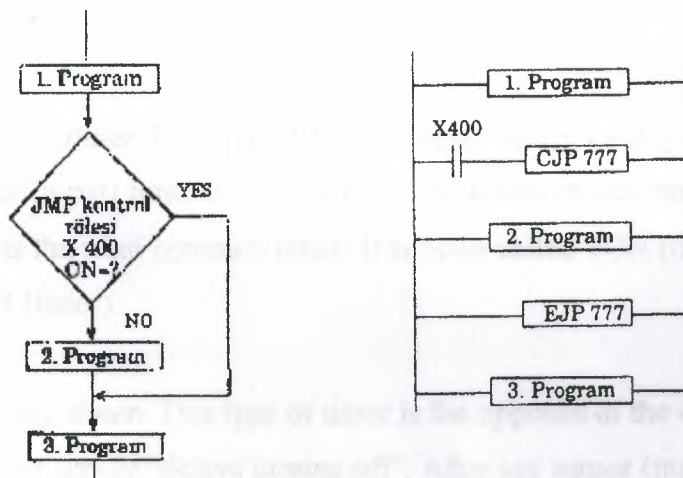
2.8. JUMP INSTRUCTION

Source peculiarity with JUMP instruction; determined program line or lines makes possible position that jumped by some condition, or conditions. Provided jumped relay is time of the ON position of JUMP command.

MITSUBISHI

CJP (Conditional Jump)

EJP (End of Jump)



Note: JUMP instructions are between 700 – 777

Above program is between the 1. and 2. Programs because of using JUMP instruction, 400-numbered input relay when passed logic 1 position, JUMP instruction come to active condition and 2. program jumped 3. program, and 3. program started to work. Because after the EJP, JUMP ending operation instruction.

With 401 numbered input came logic 1 (ON) jumping operation starts and from CJP 700 until EJP 700 line program line jumps.

Jumping operation goes when X401 OFF. When X401 OFF done program return to work normally and scan operation works line by line.

While X401 OFF position, JMP function does not work. The important point is; before CJP instruction, EJP used must go to last EJP operation. Others will be invalid.

2.9 TIMERS

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

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

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

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

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

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

What will enable the timer. Typically this is one of the inputs.(a sensor connected to input 0000 for example)

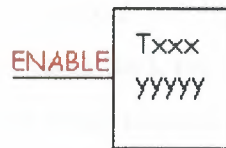
How long we want to delay before we react. Let's wait 5 seconds before we turn on a solenoid, for example.

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

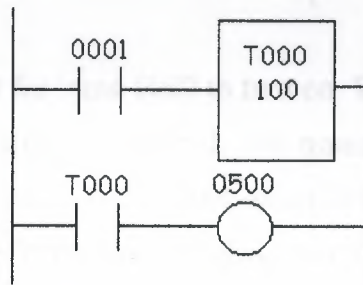
Why the weird numbers? Again its because most PLCs have 16-bit timers. We'll get into what this means in a later chapter but for now suffice it to say that 0-9999 is 16-bit BCD (binary coded decimal) and that 0 to 65535 is 16-bit binary. Each tick of the clock is equal to x-seconds.

Typically each manufacturer offers several different ticks. Most manufacturers offer 10 and 100 ms increments (ticks of the clock). An "ms" is a mili-second or *1/1000th* of a second. Several manufacturers also offer 1ms as well as 1 second increments. These different increment timers work the same as above but sometimes they have different names to show their time-base. Some are TMH (high speed timer), TMS (super high speed timer) or TMRAF (accumulating fast timer).

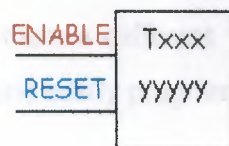
Shown below is a typical timer instruction symbol we will encounter (depending on which manufacturer we choose) and how to use it. Remember that while they may look different they are all used basically the same way. If we can setup one we can setup any of them.



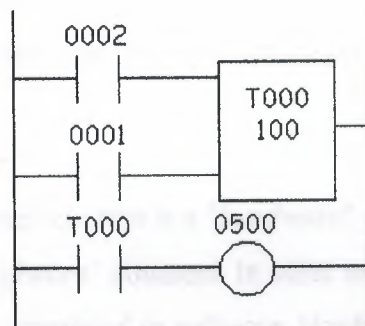
This timer is the on-delay type and is named Txxx. When the enable input is on the timer starts to tick. When it ticks yyyyy (the preset value) times, it will turn on its contacts that we will use later in the program. Remember that the duration of a tick (increment) varies with the vendor and the time-base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



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



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



In this diagram we wait for input 0002 to turn on. When it does timer T000 (a 10ms increment timer) starts ticking. It will tick 100 times. Each tick (increment) is 10ms so the timer will be a 1000ms (i.e. 1 second) timer. $100\text{ticks} \times 10\text{ms} = 1,000\text{ms}$. When 1 second has elapsed, the T000 contacts close and 500 turns on. If input 0002 turns back off the current elapsed time will be retained. When 0002 turns back on the timer will continue where it left off. When input 0001 turns on (true) the timer T000 will reset back to 0 causing its contacts to turn off (become false) thereby making output 500 turn back off.

2.10. COUNTERS

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

What kinds of counters are there? Well, there are up-counters (they only count up 1,2,3...). These are called CTU,(count up) CNT,C, or CTR. There are down counters (they only count down 9,8,7,...). These are typically called CTD (count down) when they are a separate instruction. There are also up-down counters (they count up and/or down 1,2,3,4,3,2,3,4,5,...) These are typically called UDC(up-down counter) when they are separate instructions.

Many manufacturers have only one or two types of counters but they can be used to count up, down or both. *Confused yet?* Can you say "no standardisation"? Don't worry, the theory is all the same regardless of what the manufacturers call them. A counter is a counter is a counter...

To further confuse the issue, most manufacturers also include a limited number of high-speed counters.

High-speed Counter :

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

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

To use them we must know 3 things:

Where the pulses that we want to count are coming from. Typically this is from one of the inputs.(a sensor connected to input 0000 for example)

How many pulses we want to count before we react. Let's count 5 widgets before we box them, for example.

When/how we will reset the counter so it can count again. After we count 5 widgets lets reset the counter, for example.

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

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

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



In this counter we need 2 inputs.

One goes before the reset line. When this input turns on the current (accumulated) count value will return to zero.

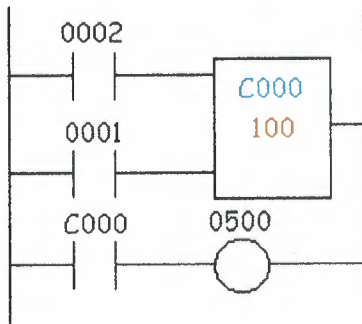
The second input is the address where the pulses we are counting are coming from.

For example, if we are counting how many widgets pass in front of the sensor that is physically connected to input 0001 then we would put normally open contacts with the address 0001 in front of the pulse line.

Cxxx is the name of the counter. If we want to call it counter 000 then we would put "C000" here.

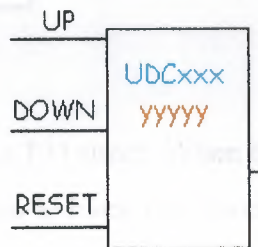
yyyyy is the number of pulses we want to count before doing something. If we want to count 5 widgets before turning on a physical output to box them we would put 5 here. If we wanted to count 100 widgets then we would put 100 here, etc. When the counter is finished (i.e we counted yyyyy widgets) it will turn on a separate set of contacts that we also label Cxxx.

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

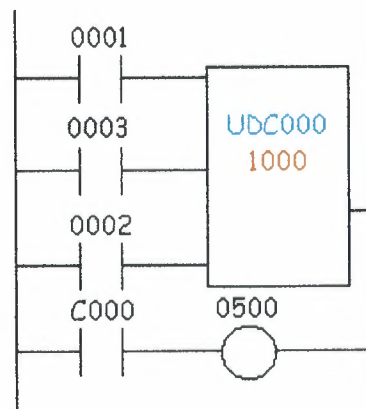


Here's the symbol on a ladder showing how we set up a counter (we'll name it counter 000) to count 100 widgets from input 0001 before turning on output 500. Sensor 0002 resets the counter.

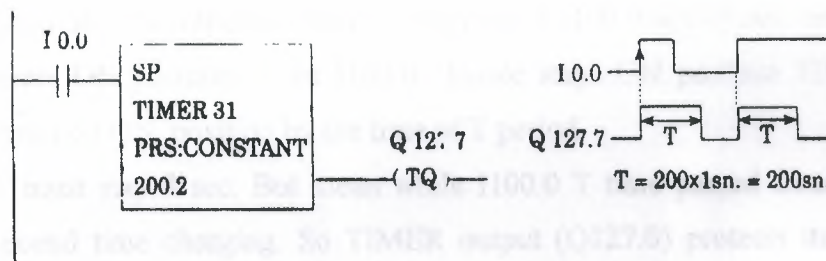
Below is one symbol we may encounter for an up-down counter. We'll use the same abbreviation as we did for the example above.(i.e. UDCxxx and yyyyy)



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



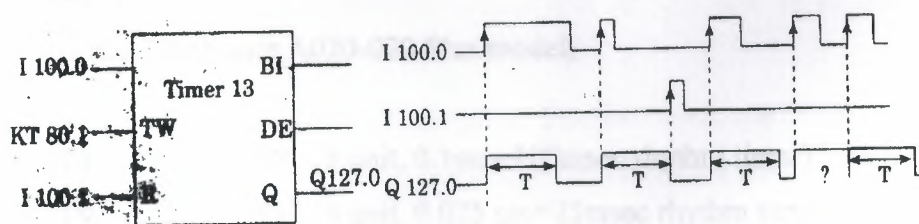
Siemens Simatic : Pulse Timer (SP)



10.0 input sensor works T31 timer. When this sensor takes ON position, settled till 200 sec, Q127.7 out put done 1. Even time over, if input signal I0.0 logic 1, output will reset.

```
: A      I      0.0
: L      KT     200.2
: SP     T      31
: =      Q      127.7
: BE
```

Extended Pulse Timer




```

: A      I      0.0
: L      KT     100.2
: SD     T      12
: A      I      0.1
: R      T      12
: A      T      12
: =      Q      100.0
: BE

```

This kind of timer controls I100.0 input sensor 13 numbered TIMER. When I100.0 sensor was made 1, the sensor which was obliged Q127.0 numbered TIMER pass ON position. The important event is the pass of I100.0 to ON position not the time of this sensors ON position. Even I100.0 1msec stays ON position TIMER protects Q127.0 sensor on ON position by the time of T period.

T must stay 8 sec. But mean while I100.0 T time passed from logic 0 to 1 without second time charging. So TIMER output (Q127.0) protects its ON position again. But it returns beginning again to count from 0, of the T time.

```

: A      I      100.0
: L      KT      80.1
: SE     T      13
: A      I      100.1
: R      T      13
: A      T      13
: =      Q      127.0

```

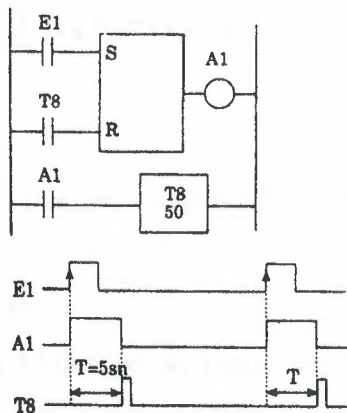
AEG

In the Teachware A020-020 Plus model;

T1.....T8 (8 unit, 0.1sec=100msec rhythm timer)

T9.....T16 (8 unit, 0.025 sec=25msec rhythm timer)

In order to 16 unit (T1.....T16) TIMER there are so programs be smallest and biggest time value is 25 msec which is 110 minutes.



1	U	E1
2	SL	A1
3	U	T8
4	RL	A1
5	U	A1
6	=	T8
7		50
8	PE	

In this example A1 is setted with E1 output. Reset position is the time of, when T8 pass ON position.

When E1 pass ON position A1 output makes set. By the setting of A1, T8 timer (present value $50 \times 0.1\text{sec} = 5\text{sec}$) count in its inside 5sec and at the end of this time logic done 1. As to program; when T8 is on, A1 output makes reset and T8 output goes OFF position because T8 output is armed reset sensor. The event to care on TIMER present value; chosen TIMER's rhythm times by its number, because of its changes, present value must count right.

The program on above; 413 numbered input sensor and M73 numbered private internal sensor are used to reset 467 numbered counter. Counting input is controlled by 412 numbered input sensor. Present value of counter is showed with K20-20. The input of counter pulse's every present pulse value is lowered 1 degree.

2.11. SHIFT REGISTER

IDEC

This model in PLC shift register unit has studied extensively.

MITSUBISHI

Internal relay M is used shift register at the some time. So 16 sensor must be 1 group at the same time First helping sensor number, shift register address and following 16 sensor can not use another arm.

Shift Register Addresses

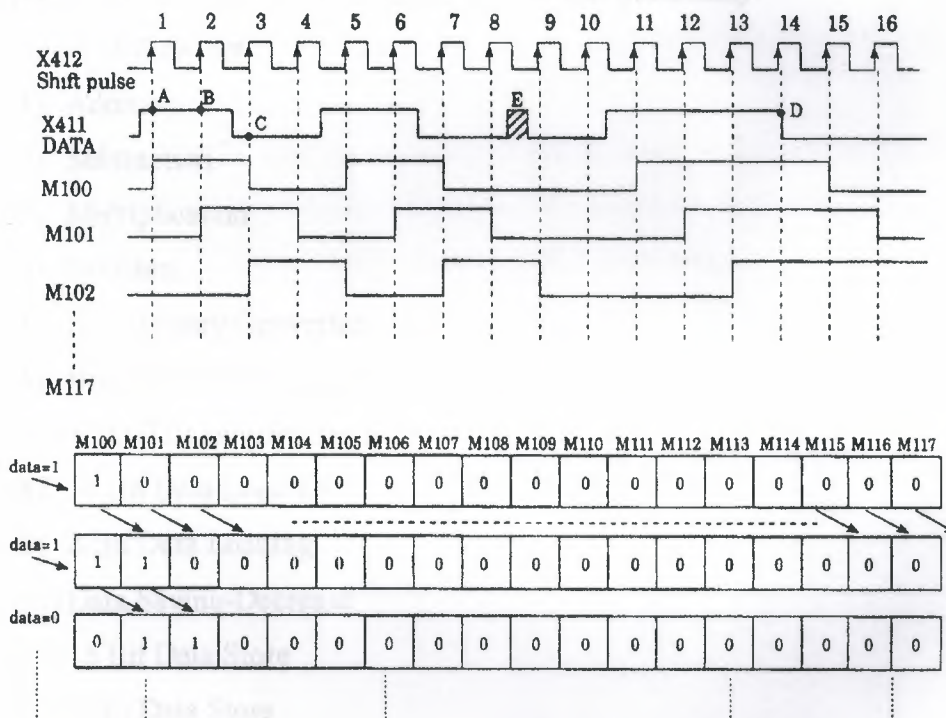
M100 – M117	=	M100.....M107, M110.....M117	= 16 unit
M120 – M137	=	M120.....M127, M130.....M137	= 16 unit
M140 – M157	=	M140.....M147, M150.....M157	= 16 unit
M160 – M177	=	M160.....M167, M170.....M177	= 16 unit
M200 – M217	=	M200.....M207, M210.....M217	= 16 unit
M220 – M237	=	M220.....M227, M230.....M237	= 16 unit
M240 – M257	=	M240.....M247, M250.....M257	= 16 unit
M260 – M277	=	M260.....M267, M270.....M277	= 16 unit
M300 – M317	=	M300.....M307, M310.....M317	= 16 unit
M320 – M337	=	M320.....M327, M330.....M337	= 16 unit
M340 – M357	=	M340.....M347, M350.....M357	= 16 unit
M360 – M377	=	M360.....M367, M370.....M377	= 16 unit

X413	DATA	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
X413	PULSE	100	101	102	103	104	105	106	107	110	111	112	113	114	115	116	117
X413	RESET																

1-Data input: Data signal which must be given to Register, is designed ON-OFF position to X411 sensor. Data, entered to register, firstly apply to M100 register. But every shift operation can make by shift pulse.

2-Shift pulse: It is shift input which is transferred to M100 by X411 entered data but while X412 is passing from 0 to 1. It can be used 72 numbered which produces 100msec time pulse or 73 numbered which produces on msec time pulse generator instead of X412.

3-Reset input: X413 input sensor is used for reset of the above. So all the register sensor with X413's passing OFF position to ON position makes reset and pass of position (M100.....M117).



(1100110000111100) data is applied with X411 data input on the above example. In here the important thing is decisive position of data in the shift pulse time. For example, 1 data's is in A point 1 data's is in B point 0 data's is in C point examples.

Decisive position in D point is 1, because while shift pulse going from 0 to 1; data value stayed decisively periods 1 pulse time in ON position, so D point of data's the time of going from 1 to 0, shift pulse which is still formed, can't catch and it can't be seen and examples the time of going from 1 to 0 of 14 pulse.

If you attend E area of data diagram; it can't be exemplified by data which is between 8 and 9 pulse and it doesn't accept like this data. According to this, for to load of data's to registers is the time of passing the time of piece of referans shift pulse (OFF→ON)

2.12. COMPUTING FUNCTION

one of the most important peculiarity of PLC system is computing and data embroidery function. As a main structure, PLC has this peculiarity.

Some of these are:

- 1) Addition
- 2) Subtraction
- 3) Multiplication
- 4) Division
- 5) BCD Binary Converter
- 6) BINARY BCD Converter
- 7) 4 DIGIT Comparation
- 8) 16 Bit Data Loading
- 9) 8 Bit Data Loading
- 10) Data Saving-Decrease
- 11) 16 Bit Data Store
- 12) 8 Bit Data Store
- 13) Data Display
- 14) BCD Shifting Left
- 15) Data Shifting

In comparison operations:

 \leq F (small equal)

Arithmetically +F instruction will provide addition of 2 complete number this instruction add ACCUM1 and ACCUM2, of for -F instruction distinct the 2 number.

From ACCU2's contents will distinct ACC1's contents.

CHAPTER III

DETAIL ANALYSIS OF PROGRAMMING

3.1 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; SFR, SFR 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 (Except FUN 147, FUN 300). As for second addresses are present peculiarity according to instruction.

There are some deliver numbers that referenced by FA1J at the below.

c) Input:

0.....7, 10.....17, 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).

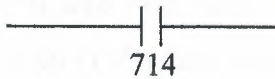
e) Internal Relay:

400 – 407	490 – 497	580 – 587
410 – 417	500 – 507	590 – 597
420 – 427	510 – 517	600 – 607
430 – 437	520 – 527	610 – 617
440 – 447	530 – 537	620 – 627
450 – 457	540 – 547	630 – 637
460 – 467	550 – 557	640 – 647
470 – 477	560 – 567	650 – 657
480 – 487	570 – 577	660 – 667

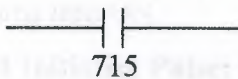
There are 240 units ($30 \times 8 = 240$) internal relays between 400 and 697, we can appoint the TIMER, COUNTER or FUN outputs to the any of 240 sensor and then can use of this sensor for take new data or count value.

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 and 9. You can use any of TIMER that include 0 and 79. In there its enough to know for you that 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 45-46 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 functions between 0 and 95 including 8-9.

1) Data Register:

Between DRO and DR99 and between 800 and 899, we have 100 data register.

3.2 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 Start 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 1 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 Carry 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 FA1J 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.

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 except 8 and 9 in the example 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 need 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.

e) Timer:

I wanted to use T8 timer from the 80 timers between 0-79 including 8 and 9 here and 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 same 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 1 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 in the other situations it becomes OFF in logic. In a multiplying processes both inputs are 1 then output is 1. And had it ended with 2 limit switches and 1 solenoid valve in order to understand the logic better. In diagrams, it is stated as relay ladder diagram and logic diagram. So we can tell that LS1 relay A and LS2 relay is B input and output is Y. In such equality it is that $Y = A \cdot B$ 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 see this in the table of truth.

As known, the series of TTL is Logic integrate containing 4 and gate with 2 inputs in 7408. As in the circuit $\frac{1}{4}$ 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 only one of the keys are OFF or 1 is enough 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. I gave 2 important information's with or instruction. One as them is out function that is symbolised with 200 in the circle. I will speak about out function 2 or 3 classes later. But now, I gave output of parallel circuit, output 200 for the first time it means that: I mentioned that special relay 704 is a clock pulse generator that has $f=1$ Hz. You see signal of clock pulse in the diagram. We determined time of 1 and 0 in input relay of 36 by chance now so that nothing will be by chance in the following lessons. Let's accept that there is a time diagram for to learn or let's assume that input 36 is gained by making ON/OFF in the form. If we think that output 200 is connected to a lamb, the situations 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 ladder diagram and a solenoid valve control in output of it. And same circuit has been gained Logic equality by using only 1or gate of integrate of 7432. It is enough to make on only one of the inputs for the outputs to be ON in 3 equaliavence circuit to make output OFF. It is necessary 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 1 from the outside, the sign is going to continue from B point as logic 0, because of the instruction of LOD NOT 1.

4.1-CHOOSING INSTALLATION AND COMMISSIONING OF C SYSTEM

4.2 Feasibility Study

Under certain circumstances an initial feasibility study may be suggested or granted , 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 costings , 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 required 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.

4.3 Design Procedure for PLC System

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 and 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.

4.3-a) 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. Advances in technology are quickly adopted by manufacturers 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 are 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.

4.3-b) 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;
- speed and power required of the CPU and instruction set.

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.

4.3-c) 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.

4.3-d) 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 semipermanent 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 several machines are controlled by identical programmable controllers running the same.

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 amounts 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.

Figure 4.1 (a) PLC memory requirements for different tasks.

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

Figure 4.1 (b) PLC memory requirements for different tasks.

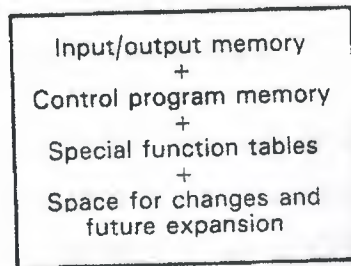
Figure 4.1 (c) PLC memory requirements for different tasks.

Figure 4.1 (d) PLC memory requirements for different tasks.

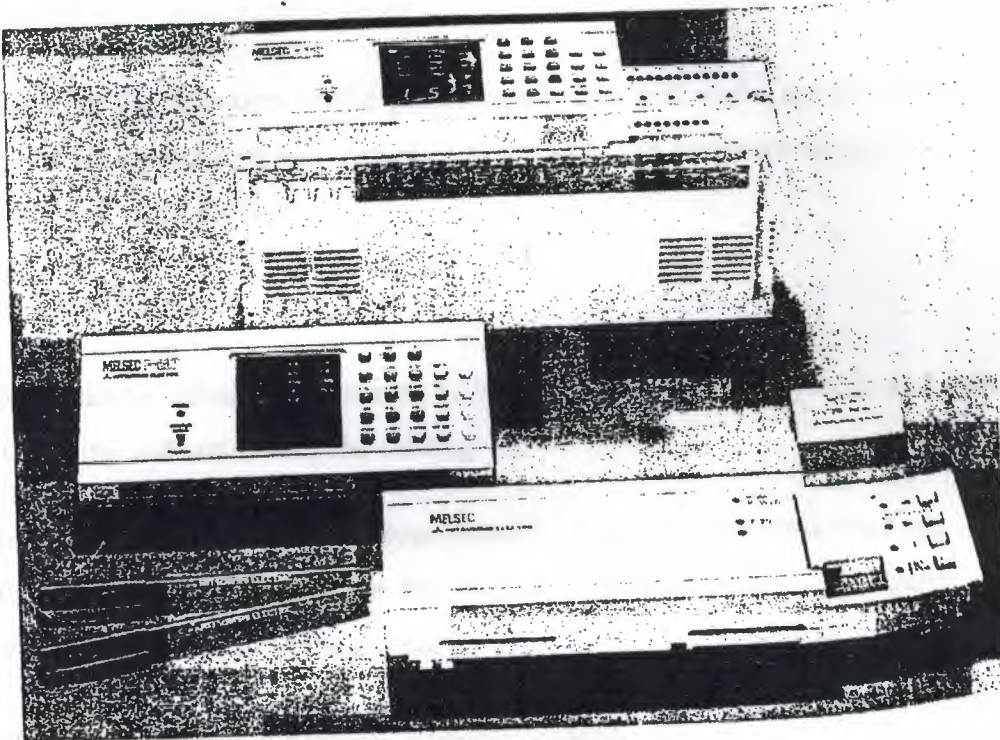
Figure 4.1 (e) PLC memory requirements for different tasks.

Figure 4.1 (f) PLC memory requirements for different tasks.

Figure 4.1 (g) PLC memory requirements for different tasks.



(a)



(b)

Figure 4.1 (a) PLC memory requirements for different tasks.

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

4.3-e) 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.

4.4 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 4.2) must be adequate for the intended environment , as regards security and 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 earthing 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.

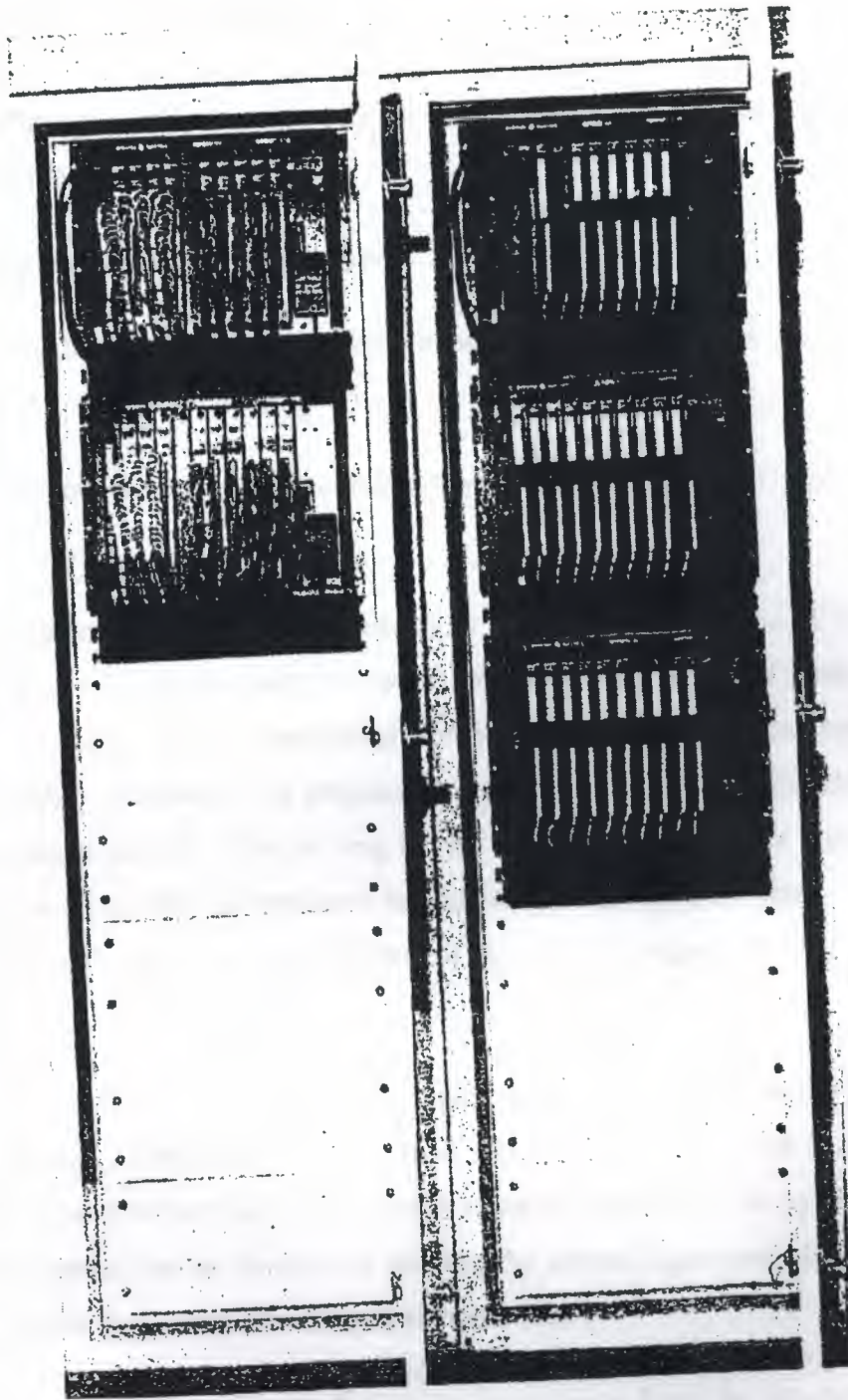


Figure 4.2 Complete PLC installation and cabinet

4.5 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 this would be done by buzzing out the connections with suitable continuity test instruments. 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, earthing, 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, analog inputs can be checked by altering 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

figuring 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 diagnostic are run by commands from the programming panel , or from within a control program once the system is fully operational.

4.5-a) 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 should 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 11.3 . 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

PLC as inputs , with outputs represented by connecting arrays of small lamps or LEDs in the figure 4.4 . This allows inputs to be offered to a test – bed controller maintaining software under test , checking the action of the control program by noting the operation and sequence of the output lamps 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 load 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 closer 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. The figure 11.5 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 update as rapidly as the control program is executing , due to the delays in transmitting the data across to the terminal.

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 11.6



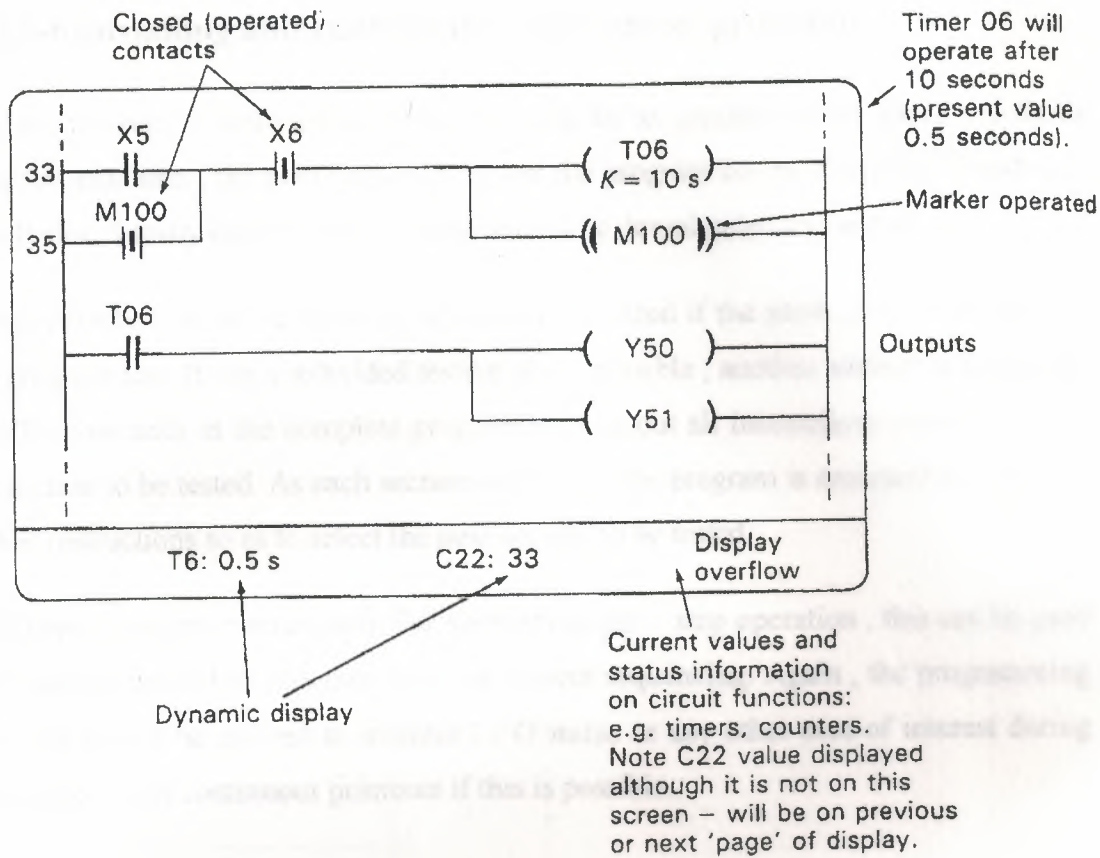


Figure 4.5 Dynamic monitoring of program contacts using a graphic programming display



Figure 4.6 Symbols displayed in monitor mode

4.5-b)Installing 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

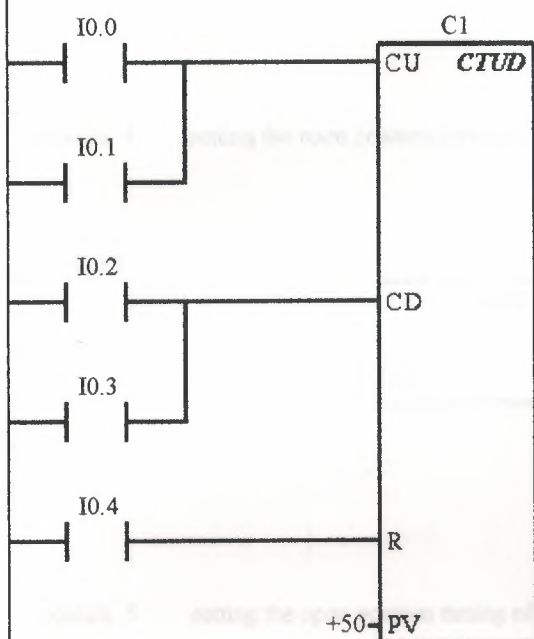
individually , allowing faults to be quickly localized if the plant misoperates during the program test. If this subdivided testing is not possible , another method is to include JUMP commands in the complete program to miss out all instructions except those in the section to be tested. As each section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

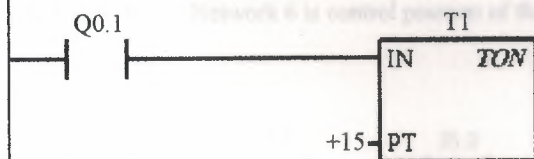
PROGRAM TITLE COMMENTS

Press F1 for help and example program

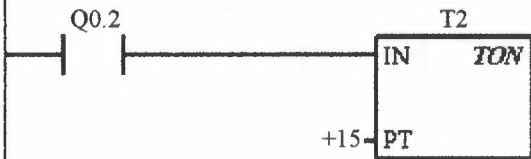
Network 1 Count the entering or leaving cars in otoprak



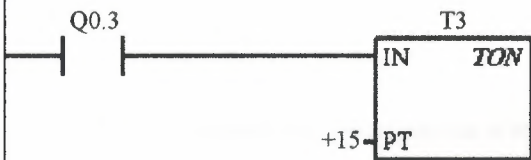
Network 2 setting the open position timing of gate 1 (entering gate Q0.1)



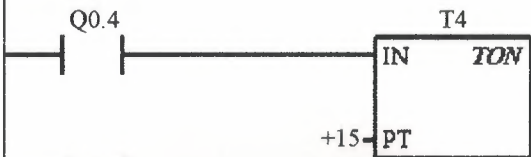
Network 3 setting the open position timing of gate 2(entering gate Q0.2)



Network 4 setting the open position timing of gate 3(exit gate Q0.3)



Network 5 setting the open position timing of gate 4(exit gate Q0.4)



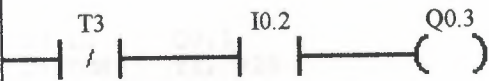
Network 6 Network 6 is control position of the first enter gate (Open or Close)



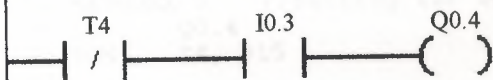
Network 7 Network 7 is control position of the second enter gate (Open or Closee)



Network 8 Network 8 is control position of the first exit gate (Open or Closee)



Network 9 Network 9 is control position of the second exit gate (Open or Closee)



Network 10




```

1  //
2  //PROGRAM TITLE COMMENTS
3  //
4  //Press F1 for help and example program
5  //
6
7  NETWORK 1    //Count the entering or leaving cars in otoprak
8  LD      I0.0
9  O       I0.1
10 LD      I0.2
11 O       I0.3
12 LD      I0.4
13 CTUD    C1, +50
14
15 NETWORK 2    //setting the open position timing of gate 1(entering gate
    Q0.1)
16 //
17 //
18 //
19 LD      Q0.1
20 TON     T1, +15
21
22 NETWORK 3    //setting the open position timing of gate 2(entering gate
    Q0.2)
23 LD      Q0.2
24 TON     T2, +15
25
26 NETWORK 4    //setting the open position timing of gate 3(exit gate Q0.3)
27 LD      Q0.3
28 TON     T3, +15
29
30 NETWORK 5    //setting the open position timing of gate 4(exit gate Q0.4)
31 LD      Q0.4
32 TON     T4, +15
33
34 NETWORK 6    //Network 6 is control position of the first enter gate (Open
    or Closee)
35 LDN     T1
36 AN      C1
37 A       I0.0
38 =       Q0.1
39
40 NETWORK 7    //Network 7 is control position of the second enter gate
    (Open or Closee)
41 LDN     T2
42 AN      C2
43 A       I0.1
44 =       Q0.2
45
46 NETWORK 8    //Network 8 is control position of the first exit gate (Open
    or Closee)
47 LDN     T3
48 A       I0.2
49 =       Q0.3
50
51 NETWORK 9    //Network 9 is control position of the second exit gate (Open
    or Closee)
52 LDN     T4
53 A       I0.3
54 =       Q0.4
55
56 NETWORK 10
57 MEND

```

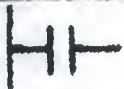






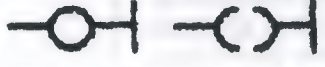

CONCLUSION

When developing this project we see that PLC the individual's life easier, which it has gained our interest and notice.

With the information observed from our lecturer and our researchers for this topic PLC, is a convenient tool with a wide range 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 met within a few minutes by the keyboard, installed PLC programs can be controlled or checked before within the office and laboratory, even the PLC programs for firm can be met at home. It is very protective and safe for the workers which they are protected from danger, communication programs of PLCs within each other or within operates can happen with the PLC; the developed industries 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.

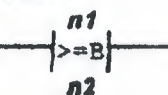
The PLC program is to fully control a parking area. Automation is achieved in this parking area and this shows that PLC is a very important device to control complicated processes.

Table of Symbol

INSTRUCTION	LADDER SYMBOL	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		=
END		MEND

Compare Byte Greater Than Or Equal Contact

Symbol:



Operands:

n1, n2 (unsigned byte):

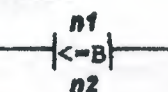
VB, IB, QB,
MB, SMB, AC,
Constant, *VD,
*AC

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):

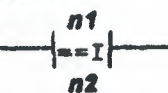
VB, IB, QB,
MB, SMB, AC,
Constant, *VD,
*AC

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):

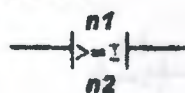
VW, T, C, IW, QW,
MW, SMW, AC,
AIW, Constant, *VD, *AC

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):

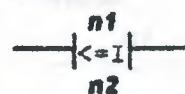
VW, T, C, IW, QW, MW,
SMW, AC, AIW, Constant,
*VD, *AC

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):

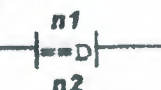
VW, T, C, IW, QW, MW,
SMW, AC, AIW, Constant,
*VD, *AC

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):

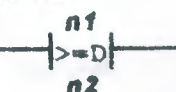
VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

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:

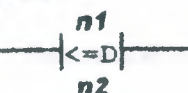
n1, n2 (signed integer double word): VD, ID, QD, MD, SMD, AC, HC, Constant, *VD, *AC

Description of operation:

The 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

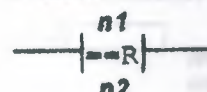
Description of operation:

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

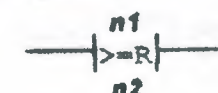
Description of operation:

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

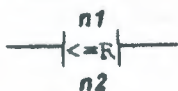
Description of operation:

The 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:

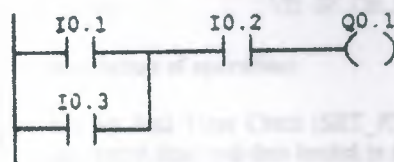
(none)

Description of operation:

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

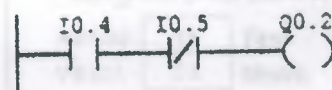
Ladder Contact Examples

Network 1



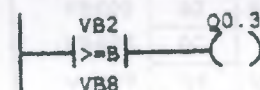
When I0.1 or I0.3 is on and I0.2 is on then output Q0.1 is turned on.

Network 2



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

Network 3



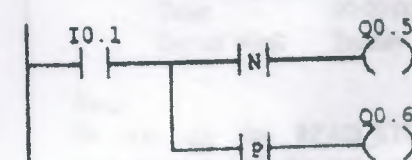
When VB2 is greater than or equal to VB8, then output Q0.3 is turned on

Network 4



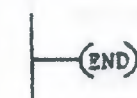
When VB4 equals VB8, then output Q0.4 is turned off
(Note: The NOT instruction can be used to create a Not Equal comparison.)

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

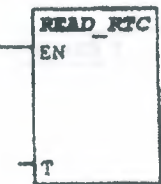


End of the main user program.

Read Real Time Clock

Note: Real Time Clock instructions are supported by the CPL 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

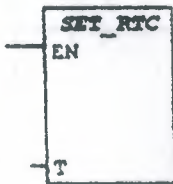
Note:

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 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 CPL 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:

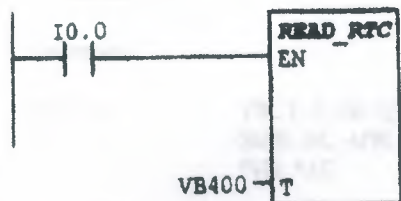
Date: 01-Jan-90
Time: 00:00:00
Day of Week: Sunday

Note:

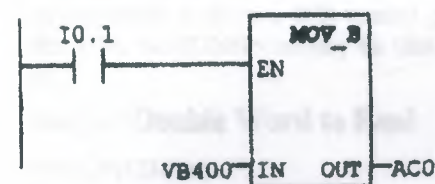
Do not use the READ_RTC / SET_RTC instructions in both the main program and in an interrupt routine. If you do this and the clock instruction is executing when the interrupt that also executes the clock instruction occurs, then the clock instruction in the interrupt routine is not executed. SM4.5 is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

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.



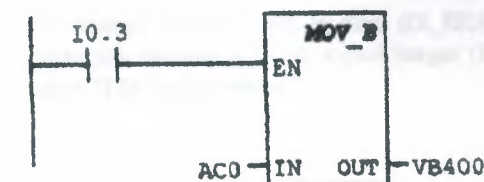
Network 2 When I0.1 is on, the year value (95) from the first byte of VB400 is moved to AC0.



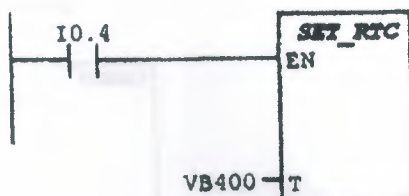
Network 3 When I0.2 is on, the year value in AC0 is incremented by 1.



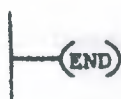
Network 4 When I0.3 is on, the new year value (96) is stored in VB400.



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

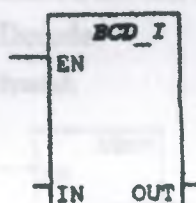


Network 6 End of the main user program.



BCD to Integer

Symbol:



Operands:

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

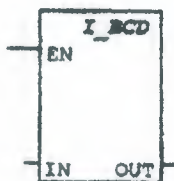
OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

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

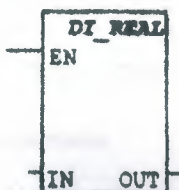
Description of operation:

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

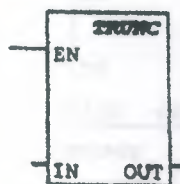
Description of operation:

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

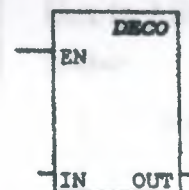
OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

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

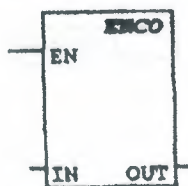
OUT (word): VW, T, C, IW, QW, MW, SMW, AC, AQW, *VD, *AC

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): VB, T, C, IW, QW, MW, SMW, AC, AIW, Constant, *VD, *AC

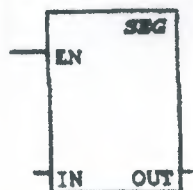
OUT (byte): VB, IB, QB, MB, SMB, AC, *VD, *AC

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

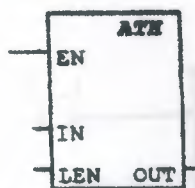
OUT (byte): VB, IB, QB, MB, SMB, AC, *VD, *AC

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

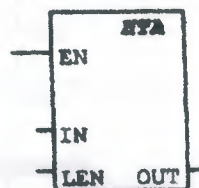
Description of operation:

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:

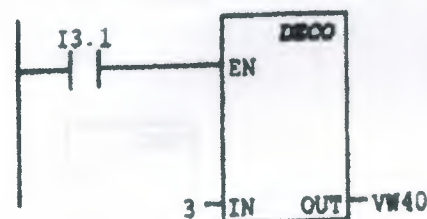
The HEX to ASCII (HTA) box converts the hexadecimal digits, starting with the input byte IN, to an ASCII string starting at the location OUT. The number of hexadecimal digits to be converted is specified by length LEN. The maximum number of the hexadecimal digits that can be converted is 255.

Ladder Conversion Instruction Examples

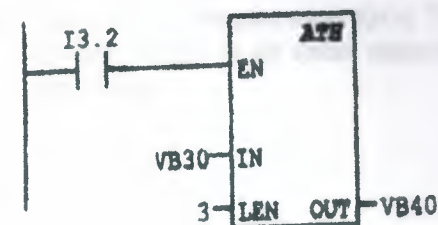
Network 1 When I3.0 is on, the Binary Coded Decimal value in VW0 is converted to an integer value.



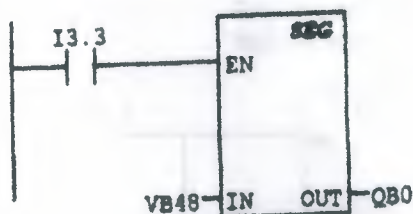
Network 2 When I3.1 is on, 3 is decoded and the corresponding bit of VW40 is set.



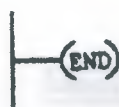
Network 3 When I3.2 is on, the 3-character ASCII string starting with the character at VB30 is converted to hexadecimal digits starting at VB40.



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:

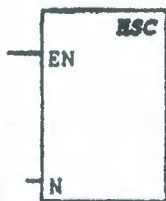
HSC (byte):	CPU 212: 0 CPU 214: 0-2
MODE (byte):	CPU 212: 0 CPU 214: 0 (HSC0), 0-11 (HSC1-2)

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.

High Speed Counter

Symbol:



Operands:

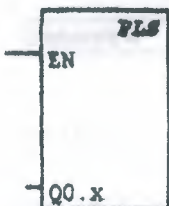
N (word): CPU 212: 0
CPU 214: 0-2

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

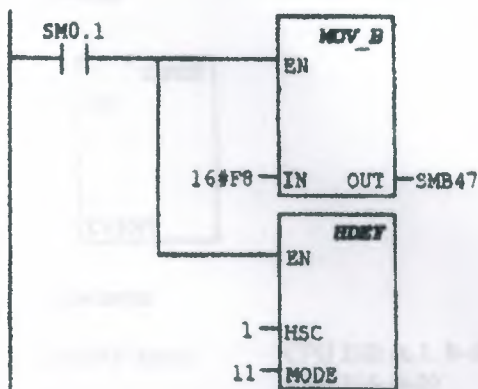
Description of operation:

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

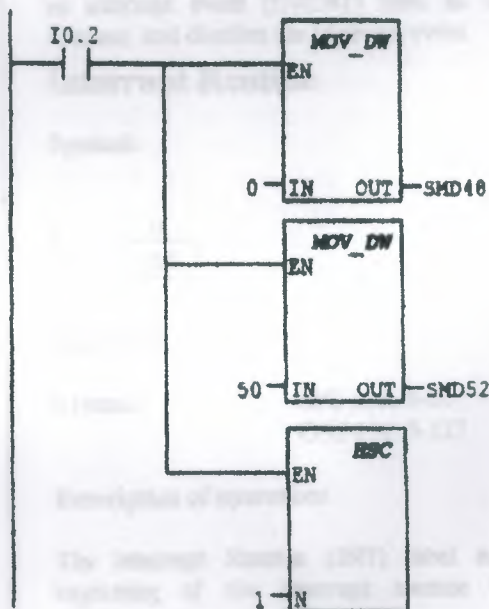
Network 1

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.



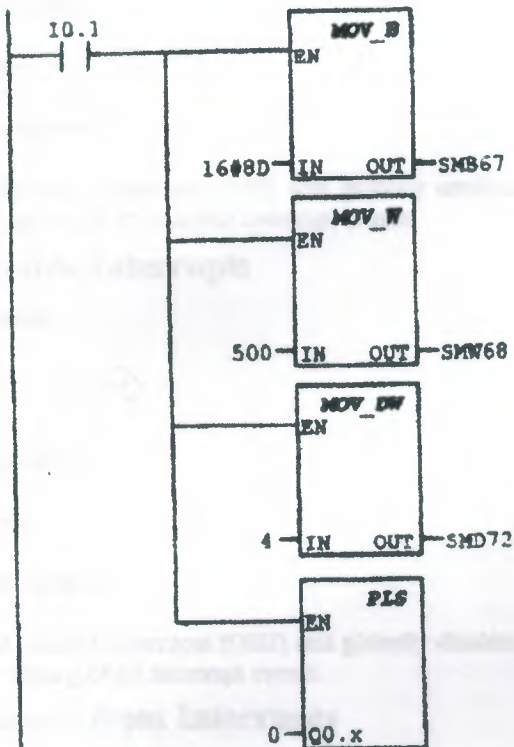
Network 2

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



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



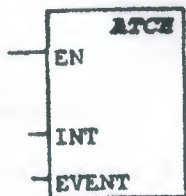
Network 4

End of the main user program.

(END)

Attach Interrupts

Symbol:



Operands:

INT (byte): CPU 212: 0-31
CPU 214: 0-127

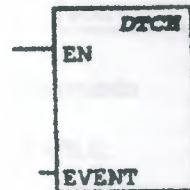
EVENT (byte): CPU 212: 0, 1, 8-10, 12
CPU 214: 0-20

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.

Enable Interrupts

Symbol:



Operands:

Flags:

Description:

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

Disable Interrupts

Symbol:



Operands:

Flags:

Description:

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

Return from Interrupts

Symbol:



Conditional Return from

Interrupts



Unconditional Return from

Interrupts

Operands:

Flags:

Description:

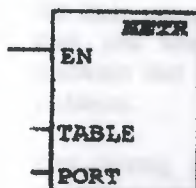
Conditional Return from Interrupts (RETI) returns from an interrupt based upon the condition of the preceding logic.

Unconditional Return from Interrupts (RETI) must be used to terminate each interrupt sequence.

Network Read

Note: CPU 214 only.

Symbol:



Operands:

TABLE: VB, MB, *VD, *AC

PORT: Constant
(CPU 214: 0)

Description of operation:

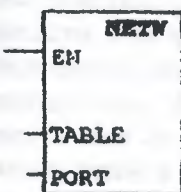
The Network Read (NETR) instruction initiates a communication operation to gather data from a remote device through the specified port (PORT), as defined in the description table (TABLE).

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)

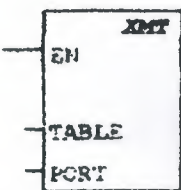
Description of operation:

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

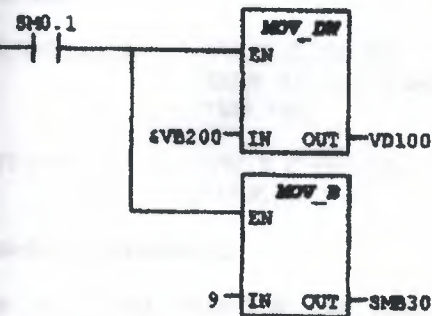
Interrupt Description (By group priority)	Event #	In-Group Priority	Supported in CPU 21
Comm. (Highest Priority)			
Receive interrupt	8	0	Y
Transmit complete interrupt	9	0*	Y
Discrete (Middle Priority)			
Rising edge, IO.0**	0	0	Y
Rising edge, IO.1	2	1	
Rising edge, IO.2	4	2	
Rising edge, IO.3	6	3	
Falling edge, IO.0**	1	4	Y
Falling edge, IO.1	3	5	
Falling edge, IO.2	5	6	
Falling edge, IO.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.

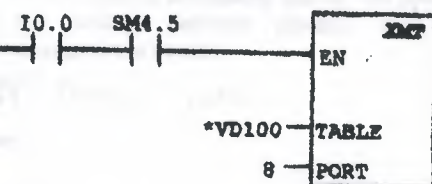
**If event 12 (HSC0 CV=Pv) is attached to an interrupt, then neither event 0 nor event 1 can be attached to interrupts. Likewise, if either event 0 or 1 is attached to an interrupt, then event 12 cannot be attached to an interrupt.

Ladder Interrupt / Communication Instruction Examples

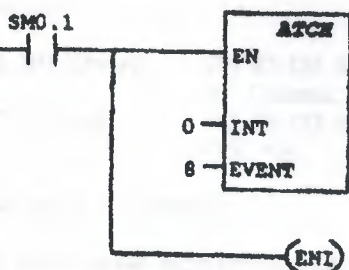
Network 1 On the first scan, create a pointer to the data to be transmitted. Select freepoint mode, 9600 baud, no parity, 8 bits per character. SMB30 is the freepoint control byte.



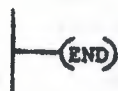
Network 2 When I0.0 and SM4.5 are both on, the message in the buffer (pointed to by VD100) is transmitted. SM4.5 is on when the transmitter is idle.



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



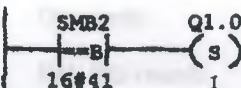
Network 4 End of main ladder program.



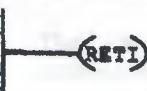
Network 5 Begin interrupt routine 0.



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



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

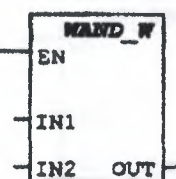
In ladder logic, vertical lines represent wires connecting to parallel branches.

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.

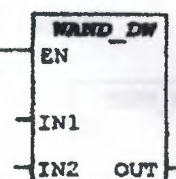
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- 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.

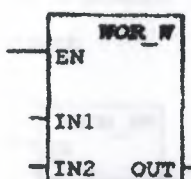
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- 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.

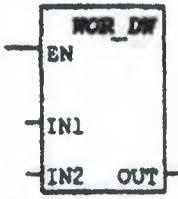
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- 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.

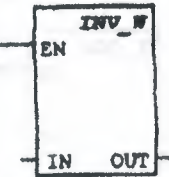
Note:

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.

Invert Word

Symbol:



Operands:

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

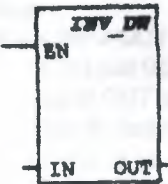
OUT (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

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

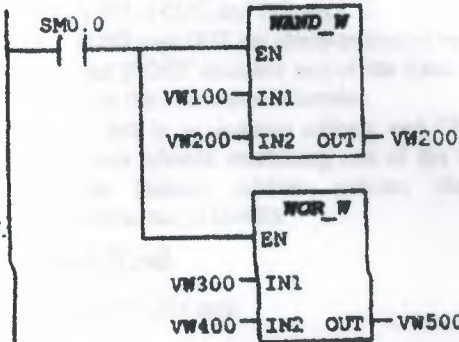
Description of operation:

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

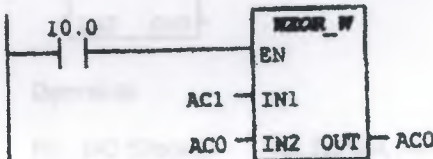
Network 1

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.



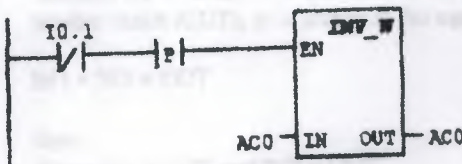
Network 2

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.



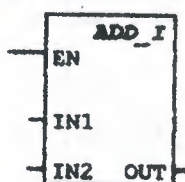
Network 4

End of main user program.



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:

$$IN1 + IN2 = OUT$$

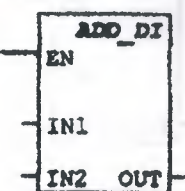
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- 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:

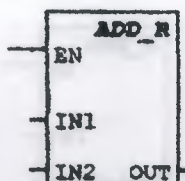
When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- 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: CPC 314 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:

$$IN1 + IN2 = OUT$$

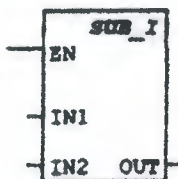
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- 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:

$$IN1 - IN2 = OUT$$

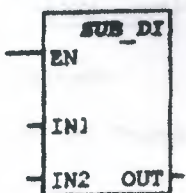
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- 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:

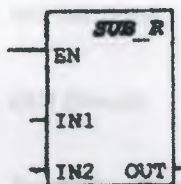
When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- 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 Subtract 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:

$$IN1 - IN2 = OUT$$

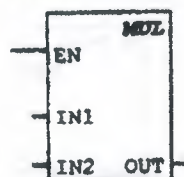
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- 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.

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:

$$IN1 * IN2 = OUT$$

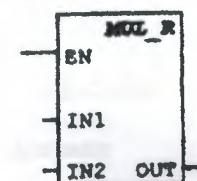
Note:

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:

$$IN1 * IN2 = OUT$$

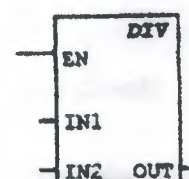
Note:

When $IN1 \neq OUT$ and $IN2 \neq OUT$:

- 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 a 16-bit quotient and a 16-bit remainder, as is shown in the equation:

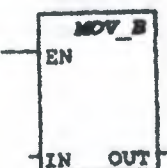
$$IN1 / IN2 = OUT$$

Notes:

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

Move Byte

Symbol:



Operands:

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

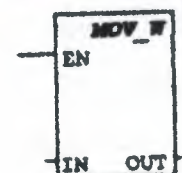
OUT (byte): VB, IB, QB, MB, SMB, AC, *VD, *AC

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

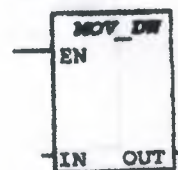
OUT (word): VW, T, C, IW, QW, MW, SMW, AC, AQW, *VD, *AC

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

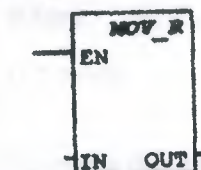
Description of operation:

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

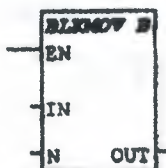
OUT (Dword): VD, ID, QD, MD, SMD, AC, *VD, *AC

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:

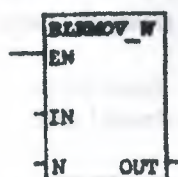
IN (byte): VB, IB, QB, MB, SMB, *VD, *AC
 OUT (byte): VB, IB, QB, MB, SMB, *VD, *AC
 N (byte): VB, IB, QB, MB, SMB, AC, Constant, *VD, *AC

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, 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 Block Move Word (BLKMOV_W) 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:

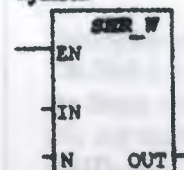
IN (word): VW, T, C, IW, QW, MW, SMW, AC, *VD, *AC

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

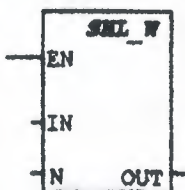
Note:

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

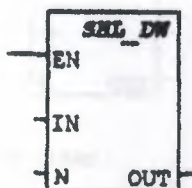
Note:

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

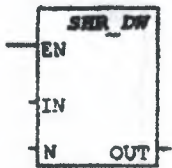
Note:

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

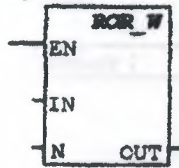
Note:

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

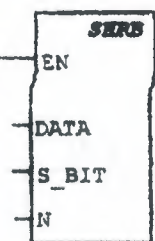
Note:

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): L, Q, M, SM, T, C, V

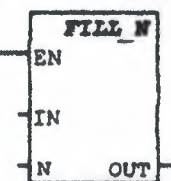
N (byte): VB, IB, QB, MB, SMB, AC.
Constant: *VD, *AC

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:

N (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

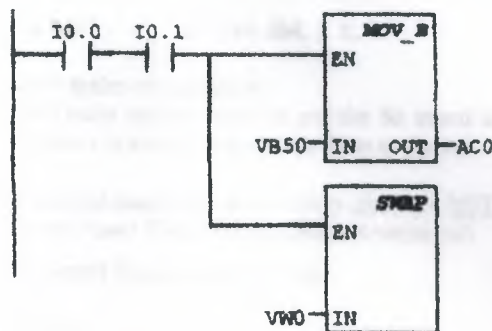
Description of operation:

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

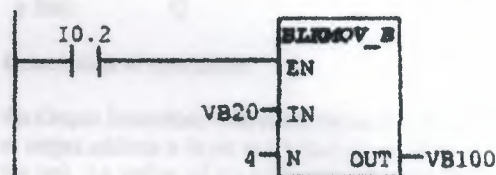
Network 1

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.



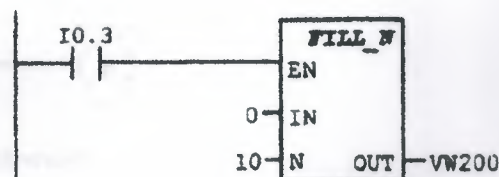
Network 2

When I0.2 is on then move VB20-VB23 to VB100-VB103.

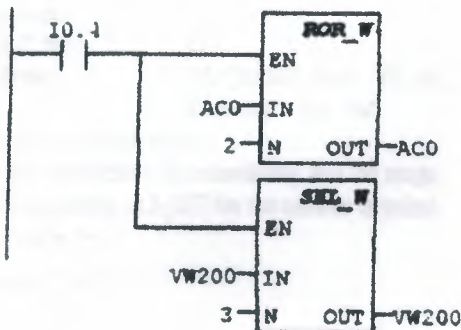


Network 3

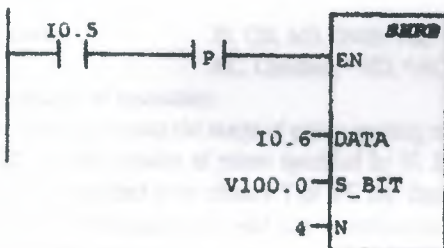
When I0.3 is on then fill VW200-VW216 with 0's.



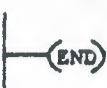
Network 4 When I0.4 is on, then the word value in AC0 is rotated right twice and stored in AC0, and the word value in VW200 is shifted left 3 times and stored in VW200.



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:

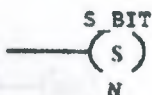
n (bit): Q

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

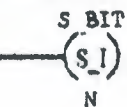
N (byte): IB, QB, MB, SMB, VB, AC, Constant, *VD, *AC

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:

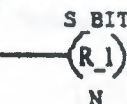
S_BIT (bit): I, Q, M, SM, T, C, V
N (byte): IB, QB, MB, SMB, VB, AC, Constant, *VD, *AC

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

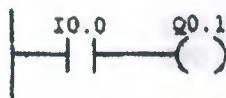
Description of operation:

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

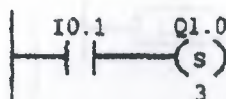
Network 1

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



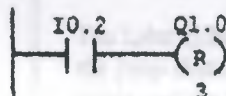
Network 2

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.



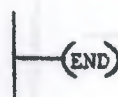
Network 3

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:

—(END) Conditional End

└─(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

Symbols:

—(STOP)

Operands:

(none)

Description of operation:

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

Watchdog Reset

Symbols:

—(WDR)

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:

—ⁿ(JMP)

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:

└─ⁿ
 LBL

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:

—ⁿ(CALL)

Operands:

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

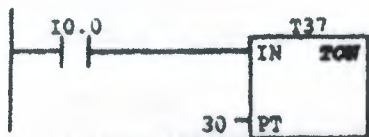
Description of operation:

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

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.



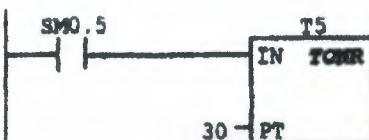
Network 2

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.



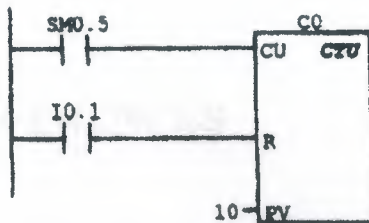
Network 4

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.



REFERENCES

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 2 İş Merkezi Yenişehir
İZMİR.