Lab session 1

Objective

Introduction to PLC

Apparatus

Step 7 Micro / WIN software

Theory

What is a PLC ?

Programmable logic controllers (PLCs) are microprocessor-based control systems designed for automation processes used in industrial environments. These controllers can automate a specific process, machine function, or even an entire production line. It has memory to that is programmable for the internal storage of user-orientated instructions such as arithmetic, counting, logic, sequencing, and timing. PLC's can be programmed to sense, activate, and control industrial equipment. A PLC, is more or less a small computer with a built-in operating system (OS). The PLC has input lines to which sensors are connected to notify of events (such as temperature above/below a certain level, liquid level reached, etc.), and output lines, to which actuators are connected to effect or signal reactions to the incoming events (such as start an engine, open/close a valve, and so on).

Why PLC not Controller?

PLC is flexible and can be easily programmed and maintained by plant engineers and technicians. It is able to survive in an industrial environment and capable of reducing machine downtime.

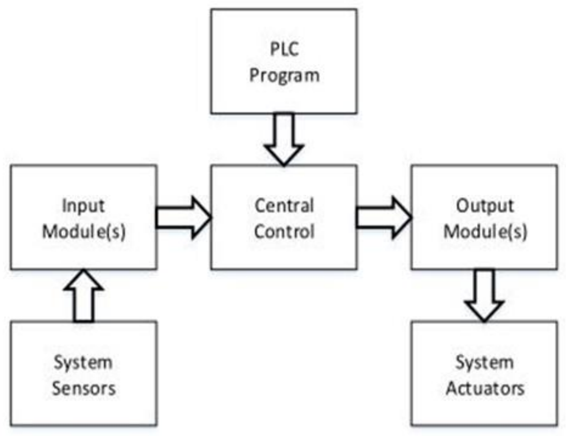
PLC is better than Microcontroller because of intended for Industrial Applications, I/O designed to interface with Control Relays and emphasis on maximum reliability. It is also similar to Microcontroller because it is microprocessor based and has memory and special Programming Language (Ladder Logic) and Input / Output Ports.

The data handling, storage, processing power and communication capabilities of some modern PLC's are approximately equivalent to desktop computers. PLC-like programming combined with remote I/O hardware, allow a general-purpose desktop computer to overlap some PLCs in certain applications. Desktop computer controllers have not been generally accepted in heavy industry because the desktop computers run on less stable operating systems than do PLCs, and because the desktop computer hardware is typically not designed to the same levels of tolerance to temperature, humidity, vibration, and longevity as the processors used in PLCs. A computer can execute a complex programming task and also multitasking. A standard PLC is designed to execute a single program in an orderly fashion. As PLCs are rapidly changing, modern PLCs have multitasking capabilities.

Basic Structure of a PLC

Programmable Logic Controllers are used for continuously monitoring the input values from sensors and produces the outputs for the operation of actuators based on the program. Every PLC system comprises these three modules:

- CPU module
- Power supply module
- One or more I/O module



CPU Module:

A CPU module consists of central processor and its memory. The processor is responsible for performing all the necessary computations and processing of data by accepting the inputs and producing the appropriate outputs.

Power Supply Module:

This module supplies the required power to the whole system by converting the available AC power to DC power required for the CPU and I/O modules. The 5V DC output drives the computer circuitry.

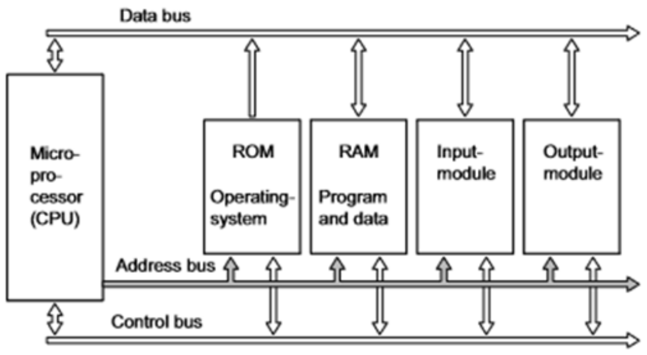
I/O Modules:

The input and out modules of the programmable logic controller are used to connect the sensors and actuators to the system to sense the various parameters such as temperature, pressure and flow, etc. These I/O modules are of two types: digital or analog.

Application program memory:

Programs specifically developed for particular applications require a program memory, from which these can be read cyclically by the central control unit. Three different types of memory are used in practice:

- RAM
- EPROM
- EEPROM



RAM

The RAM (random access memory) is a fast and highly cost effective memory. It can b called as main memory of computers. Rams are read/write memories and can be easily programmed and modified. The disadvantage of a RAM is that it is volatile, i.e. The program stored in the RAM is lost in the event of power failure.

EPROM

The EPROM (erasable programmable read-only memory) is also a fast and low cost memory, which, in comparison with RAM, has the added advantage of being non-volatile, i.e. Remanent. The memory contents therefore remain intact even in the event of power failure.

EEPROM

The EEPROM (electrically erasable programmable ROM), EEROM (electrically erasable ROM) and EAROM (electrically alterable ROM) or also flash-EPROM have been available for some time. The EEPROM in particular, is used widely as an application memory in plcs. The EEPROM is an electrically erasable memory, which can be subsequently written to.

PLC Programming Languages

PLC is user programmable. Five programming languages are used to program PLC:

1. FBD (Function block diagram)
2. LD (Ladder diagram)
3. ST (Structured text, similar to the Pascal programming language)
4. IL (Instruction list, similar to assembly language)
5. SFC (Sequential Function chart)

In this lab session we study about ladder diagram by using step 7 micro/win software and PLC trainer for input/output ports.

Ladder Diagram:

The ladder diagram language like the function block diagram represents a graphic programming language. Most PLC programming software offers programming in Ladder Logic, or "C". Ladder Logic is the traditional programming language. It mimics circuit diagrams with "**rungs**" of logic read left to right. Each rung represents a specific action controlled by the PLC, starting with an input or series of inputs (**contacts**) that result in an output (**coil**). Because of its visual nature, Ladder Logic can be easier to implement than many other programming languages. "C" programming is a more recent innovation. Some PLC manufacturers supply control programming software.

Each input or output instruction is assigned an address indicating the location in the PLC memory where the state of that instruction is stored.

Function mode of a PLC

Programs for conventional data processing are processed once only from top to bottom and then terminated. In contrast with this, the program of a PLC is continually processed cyclically.

Applications

- PLC has low power consumption.
- It is very fast and easy maintenance due to modular assembly and is capable of handling of very complicated logic operations
- A PLC is much more reliable, designed for a mean time between failures measured in years.
- A PLC can be placed in an industrial environment with its substantial amount of electrical noise, vibration, extreme temperatures, and humidity.
- PLCs are easily maintained by plant technicians.

Lab Session # 02

BASIC LOGIC GATES

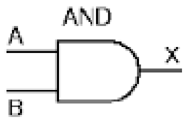
Experiment:

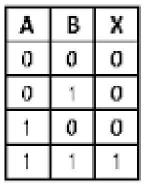
Perform Basic Logic Operation using Ladder Logic.

AND Gate:

A Boolean operator which gives the value one if and only if all the operands are one, and otherwise has a value of zero.

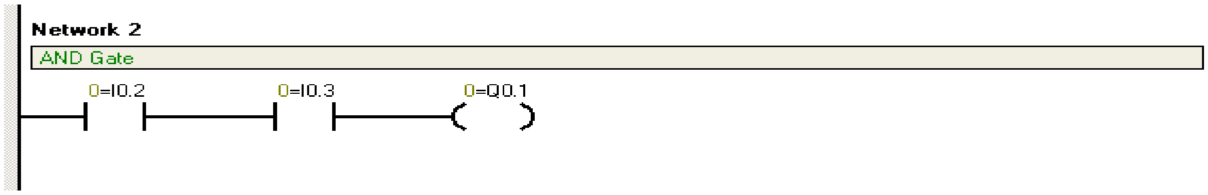
Symbol & Truth Table:





A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Ladder Diagram:

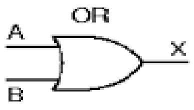


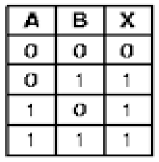
Here, I0.2 and I0.3 are Inputs and Q0.1 is Output.

OR Gate:

A Boolean operator which gives the value one if at least one operand has the value one, and otherwise gives a value of zero.

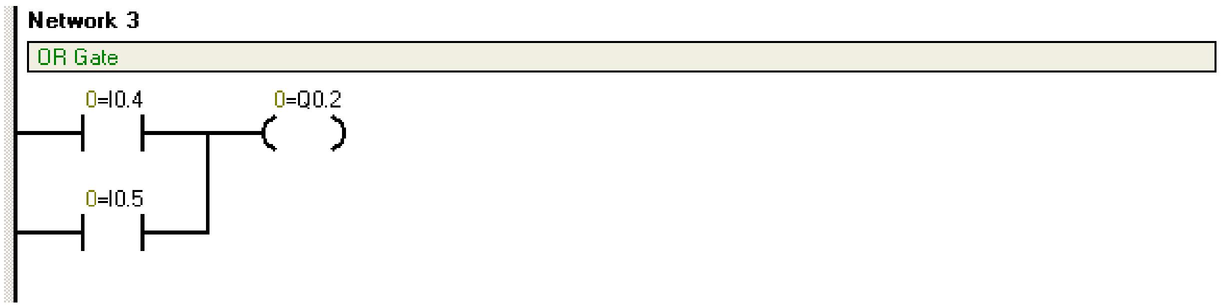
Symbol & Truth Table:





A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Ladder Diagram:

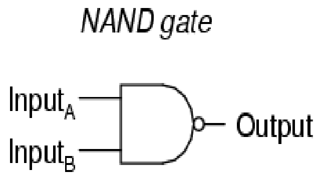


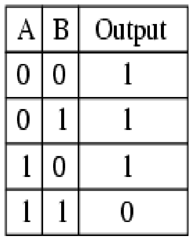
Here, I0.4 and I0.5 are Inputs and Q0.2 is Output.

NAND Gate:

A Boolean operator which gives the value zero if and only if all the operands have a value of one, and otherwise has a value of one.

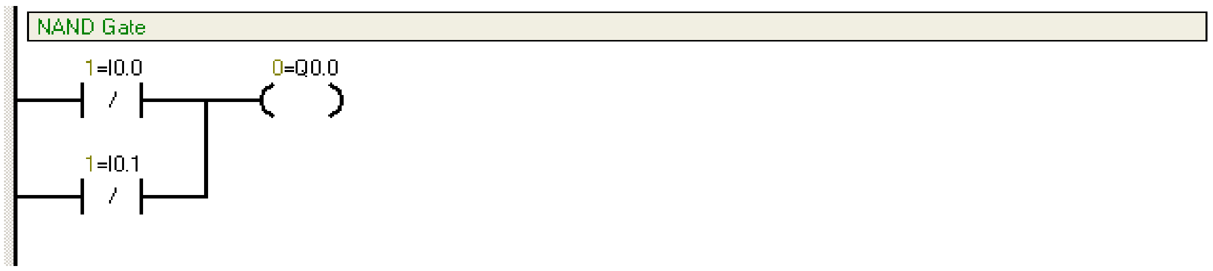
Symbol & Truth table:





A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Ladder Diagram:

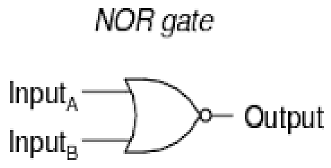


Here, I0.0, I0.1 are Inputs and Q0.0 is Output.

NOR Gate:

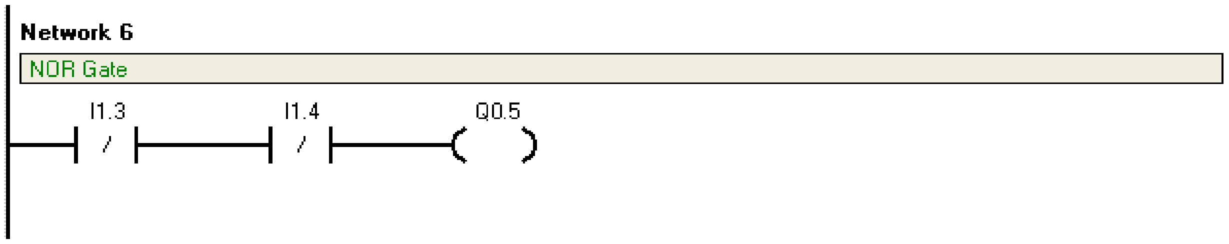
A Boolean operator which gives the value one if and only if all operands have a value of zero and otherwise has a value of zero.

Symbol & Truth Table:



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

Ladder Diagram:

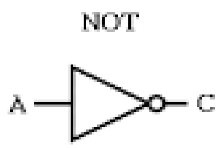


Here, I1.3 and I1.4 are Inputs and Q0.5 is Output.

NOT Gate:

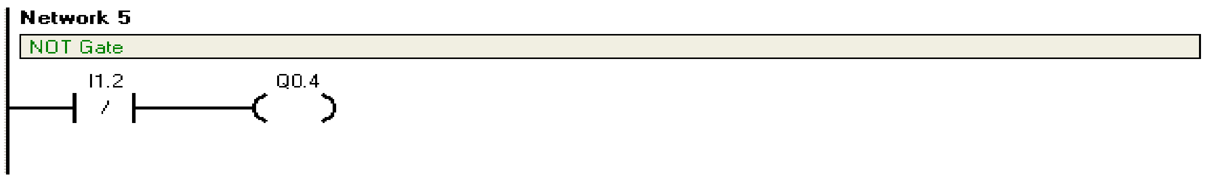
A Boolean operator with only one variable that has the value one when the variable is zero and vice versa.

Symbol & Truth Table:



Input	Output
A	C
0	1
1	0

Ladder Diagram:

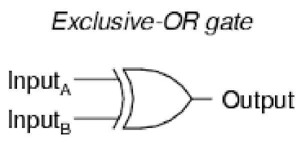


Here, I1.2 is Input and Q0.4 is Output.

XOR Gate:

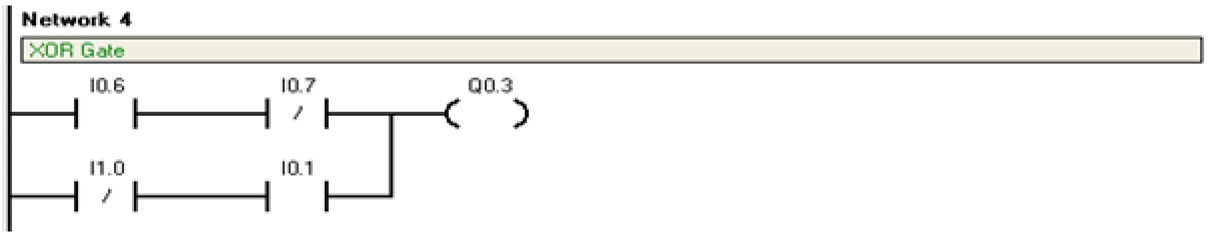
A Boolean operator that gives a true (1/HIGH) output when the number of true inputs is odd. An XOR gate implements an exclusive or; that is, a true output results if one, and only one, of the inputs to the gate is true.

Symbol & Truth Table:



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Ladder Diagram:

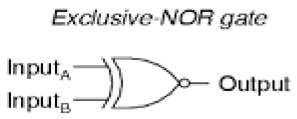


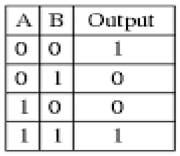
Here, I0.6, I0.7, I1.0 and I0.1 are Inputs and Q0.3 is Output.

XNOR Gate:

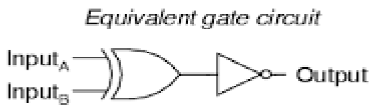
A Boolean operator with two or more inputs and one output that performs logical equality. The output of an XNOR gate is true when all of its inputs are true or when all of its inputs are false.

Symbol & Truth Table:

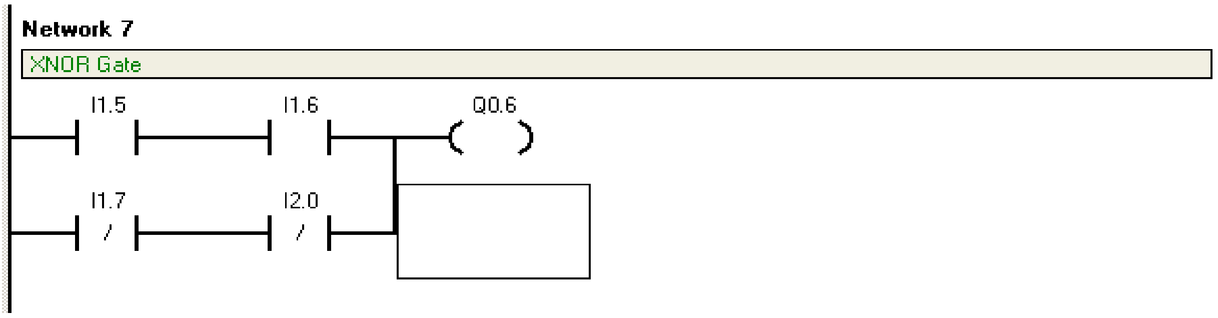




A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1



Ladder Diagram:



Here, I1.5, I1.6, I1.7 and I2.0 are Inputs and Q0.6 is Output.

Lab Session # 03

Timers

Experiment:

Basics of Timers.

Introduction:

A timer is simply a control block that takes an input and changes an output based on time. There are two basic timer types – On-Delay Timer and the Off-Delay Timer.

- **On-Delay Timer** - this timer takes an input, waits a specific amount of time, then turns ON an output (or allows logic to flow after the delay).
- **Off-Delay Timer** – this timer takes turns ON an output (or allows logic to flow) and keeps that output ON until the set amount of time has passed, then turns it OFF (hence off-delay).

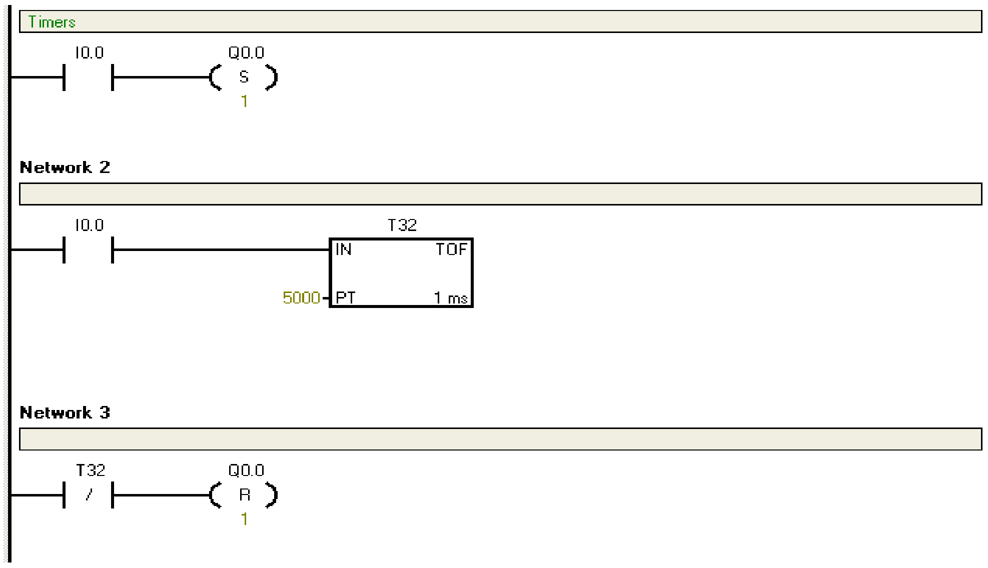
Problem Statement:

A. Fluorescent light for garage door opener is attached. Every time the garage door is open the light is "ON". After the garage door is closed the light remains
"ON" for 5 seconds while driver walks to the house from the car.

Solution:

Input/Output declaration:
Garage Switch → I0.0
Garage light → Q0.0

Ladder Diagram:



As I0.0 is ON the coil Q0.0 is set and the timer starts. After 5 seconds when I0.0 is OFF the set coil is reset and the output Q0.0 is turned OFF.

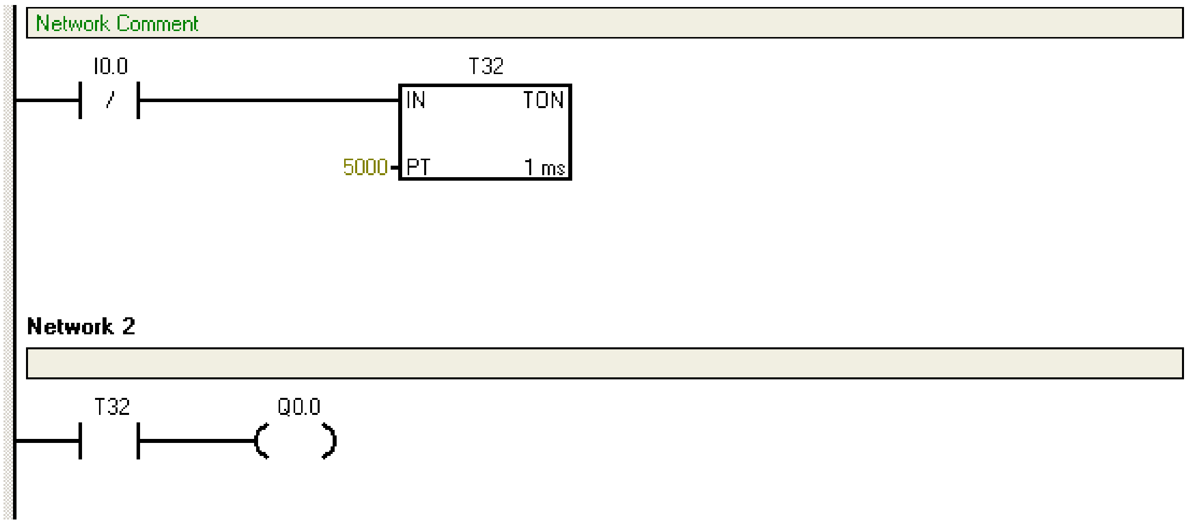
B. There is a transfer line of canned food. The inductive proximity switch is used to detect the movement of cans. If cans become jam, the movement stops. Since the cans will not be passing the detection point. The timer will time out. The output of the timer will either trigger an alarm or stop the machine until system is corrected.

Solution:

Input/Output ports declaration:

Inductive Proximity Switch	→	I0.0
Alarm output	→	Q0.0

Ladder Diagram:



As I0.0 is giving pulsating input when the cans are passing on the line. When the cans are jammed, after 5 seconds the alarm on Q0.0 is turned ON showing the cans are jammed.

Lab Session #04

Counters

Experiment:

Basics of Counters.

Introduction:

A counter simply counts the number of events that occur on an input. There are two basic types of counters – Up counter and a Down counter.

- **Up Counter** – as its name implies, whenever a triggering event occurs, an up counter increments the counter.
- **Down Counter** - whenever a triggering event occurs, a down counter decrements the counter.

Problem Statement:

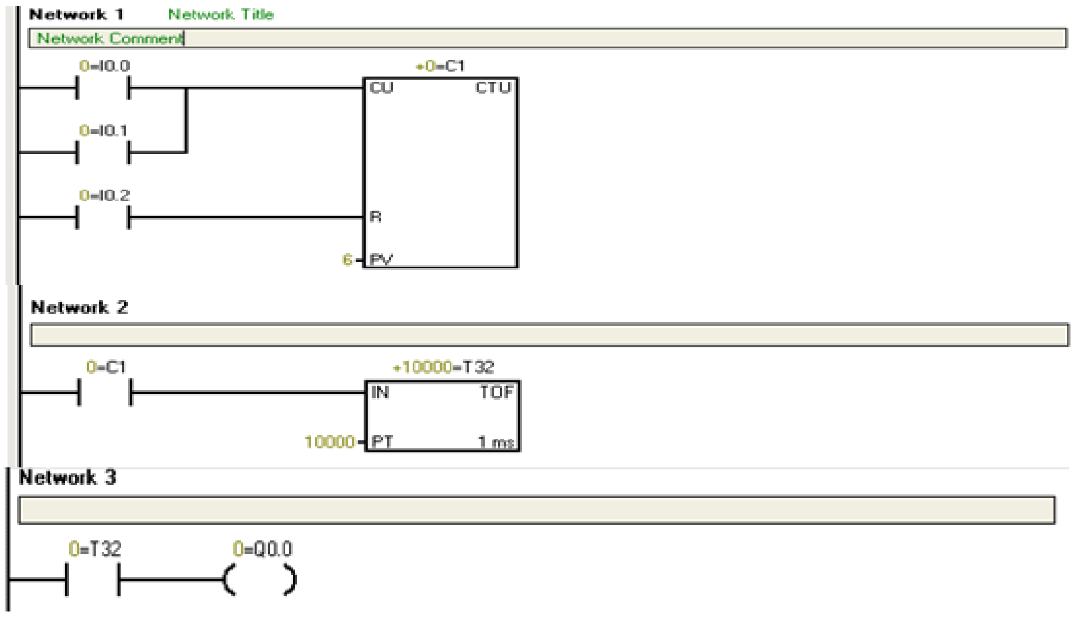
A. Write a program to turn ON a light after 6 events and to turn it OFF after 10 seconds. Events may be treated as two normally open switches.

Solution:

Input/Output port declaration:

Events	→	I0.0& I0.1
Reset	→	I0.2
Light	→	Q0.0

Ladder Diagram:



As any of I0.0 or I0.1 is turned ON the counter value is incremented by one and when the counter reaches the preset value 6 a timer is turned ON and hence, light is turned ON after 10 seconds. The counter is reset by giving a pulse to I0.2.

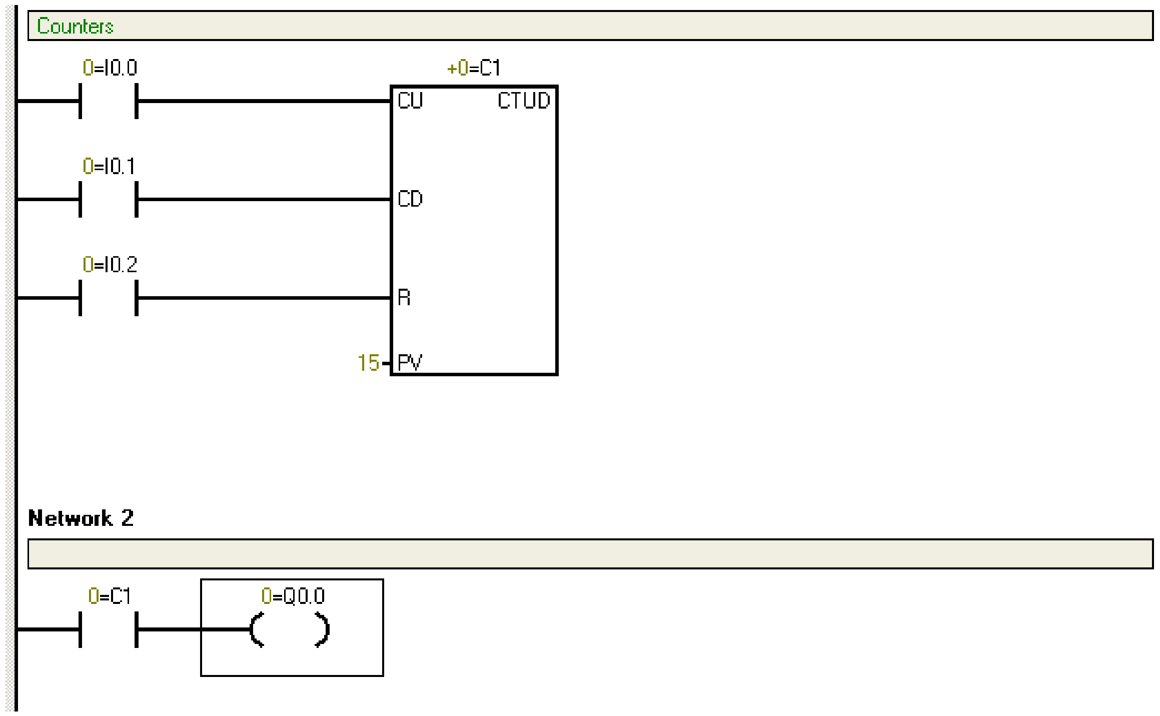
B. Design a ladder logic diagram that will use the counter elements as one UP/DOWN counter to keep track of the cars in a certain parking plaza. If the number of cars exceed a certain value (say 15), CAR LOT FULL light should light up.

Solution:

Input/Ouput ports declaration:

Counter Up	→ I0.0
Counter down	→ I0.1
Counter reset	→ I0.2
Light	→ Q0.0

Ladder Diagram:



As I0.0 is used to increment the counter and I0.1 is used to decrement the counter as the car leaves. Input I0.2 is used to reset the counter. Hence, when the counter value reached to the preset value the light is turned ON showing that the parking is full.

Lab Session #06

Title

Introduction to Following Instructions:

1) Move Instruction
2) Data Inversion
3) Compare
4) Interrupt

Theory

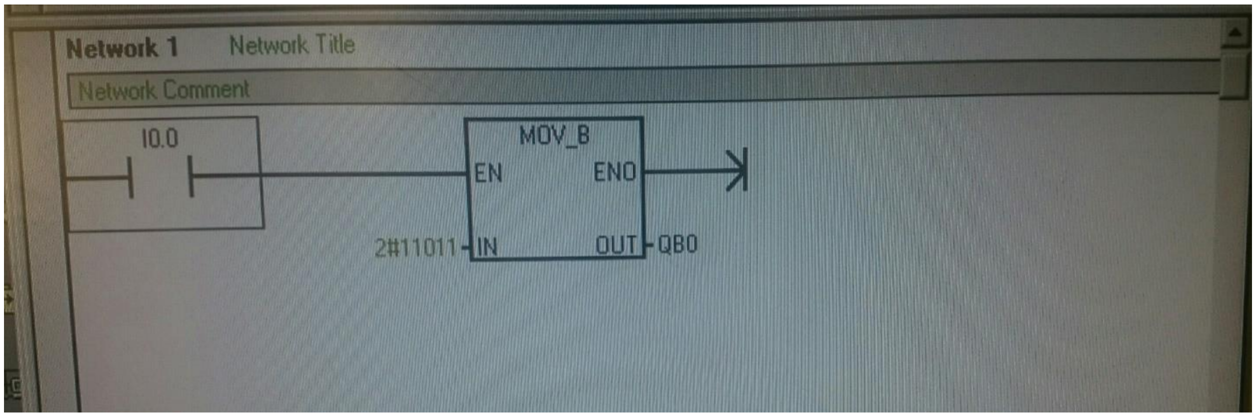
Move instruction is used to Move data (in all basis i.e.) to any memory location. this block has two inputs one of them is EN (enable pin) & other One is IN (input) we can input here either memory location in which our data is placed to move other location or constant number including with base i.e. 2#101011, 16#5AD6. It has a single output where we give a memory location to which we want to move our data.it can be used to move data in Bit, Bytes & Word.

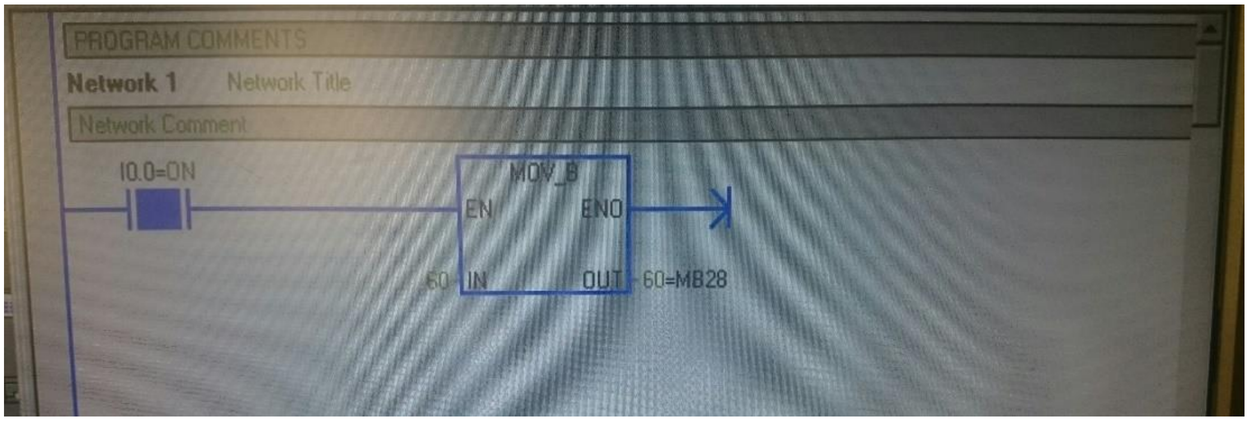
DATA Inversion instruction is used to invert data. it also has 2 inputs one of them is EN and other one is IN (Input) here we input our data that we want to invert this input can be a constant number or a memory location.it has only one output where it store inverted data here we can only input a memory location.

Compare instruction is used to compare input with any desired value. This instruction does not have any block it is just like input with center having comparing condition. If condition fulfils then it will give output otherwise it remains off.it has only one input written on the bottom of this instruction where we input our number to which we want to compare our input.

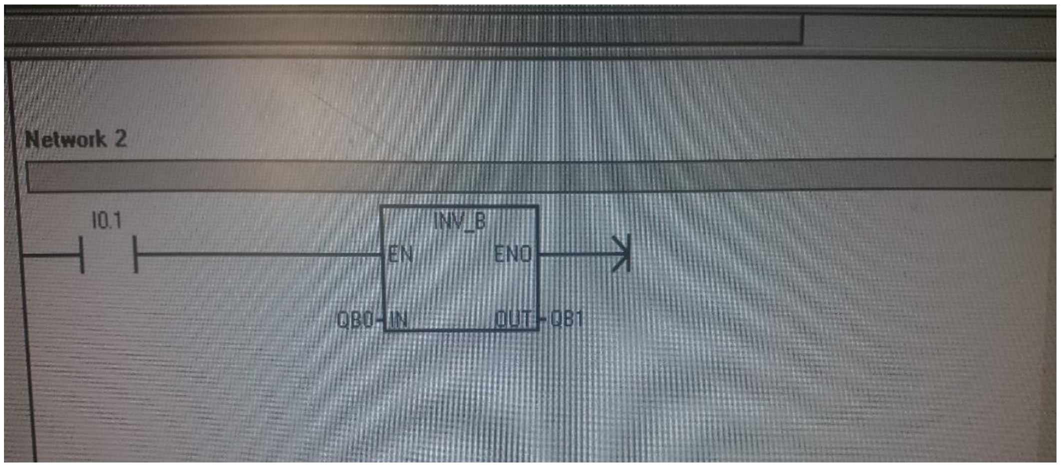
Interrupt Instruction is used to interrupt our main code. We use it for any kind of disturbance. A complete Interrupt consists on Attach & Detach. Attach block has three inputs 1st one is EN, 2nd one is INT (initial value) & 3rd one is EVNT (event) it is the event on which interrupt is required.

Move Instruction

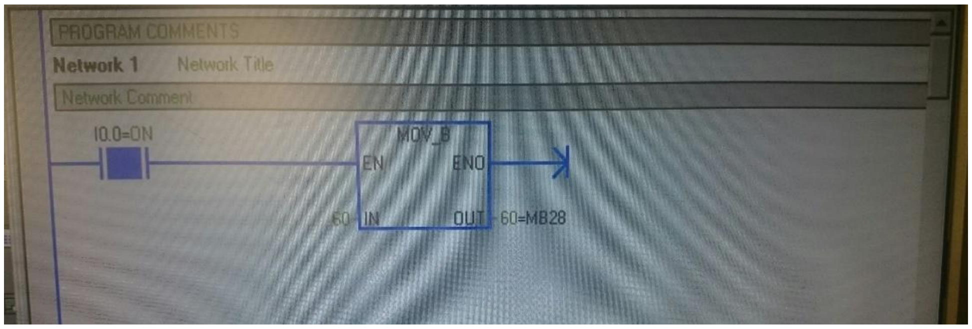


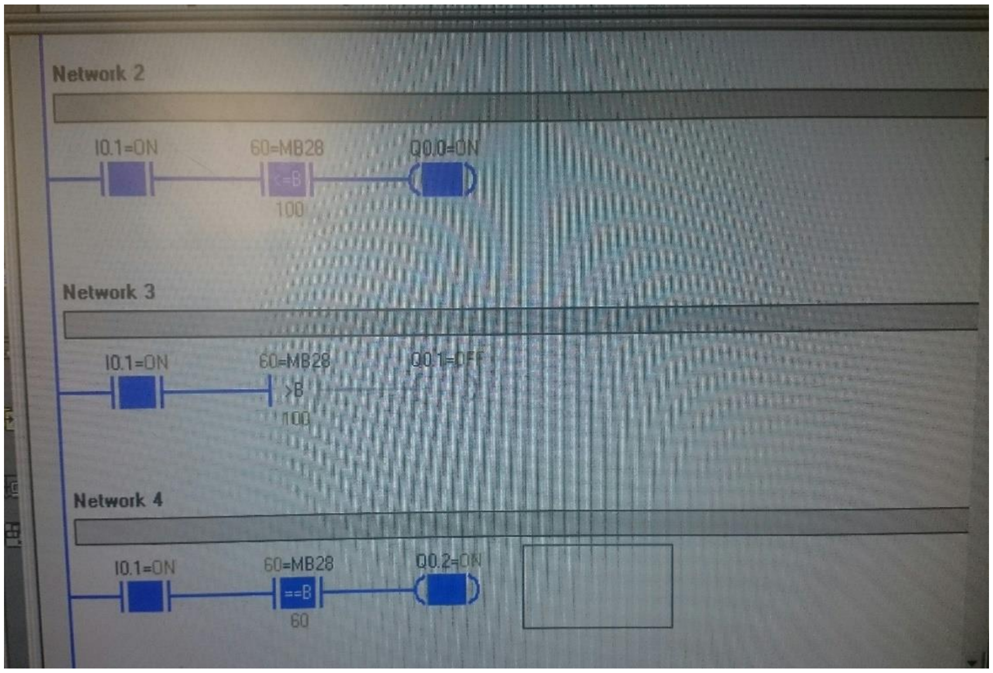


Inversion Instruction

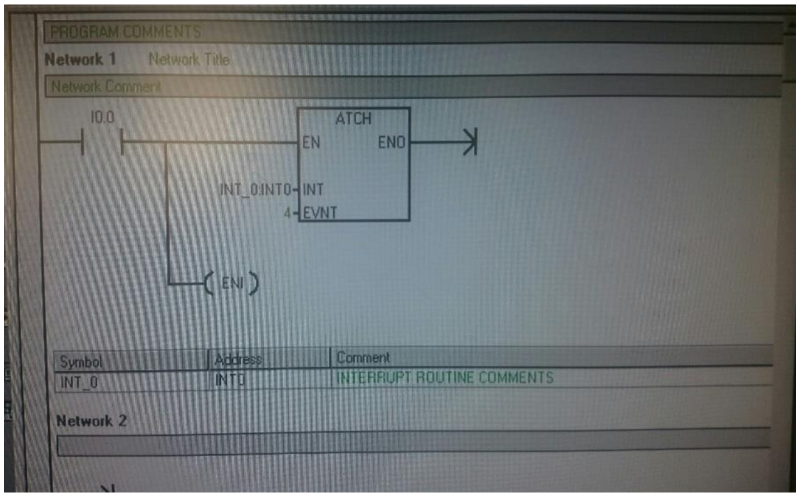


Compare Instruction

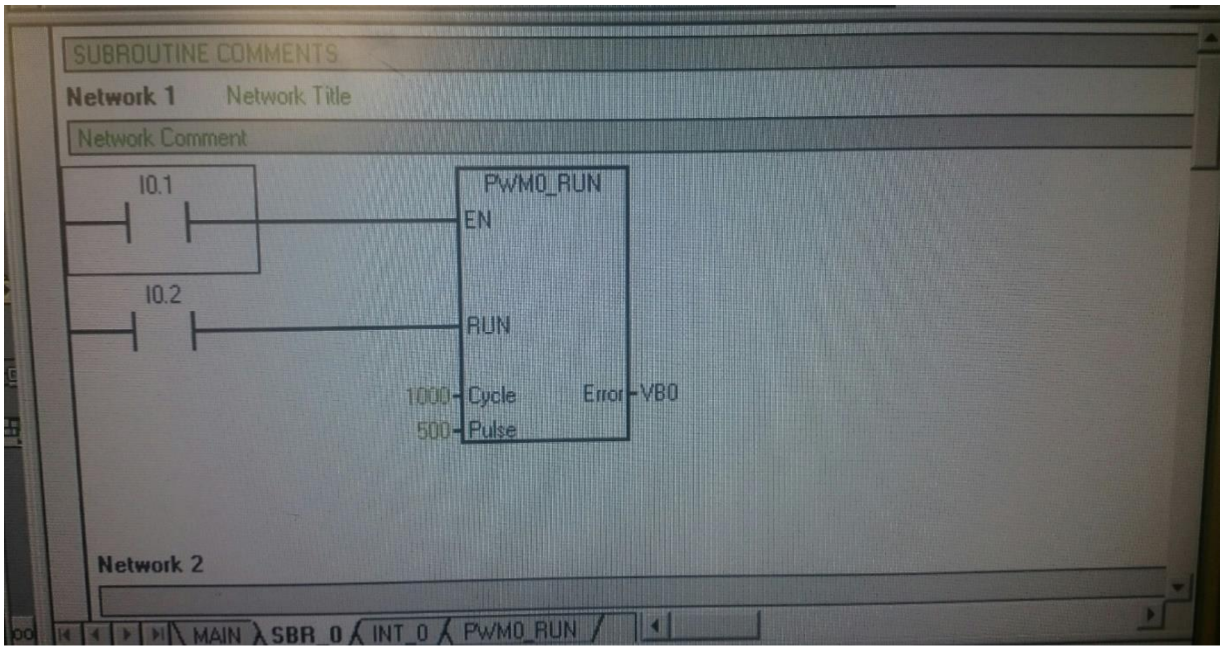


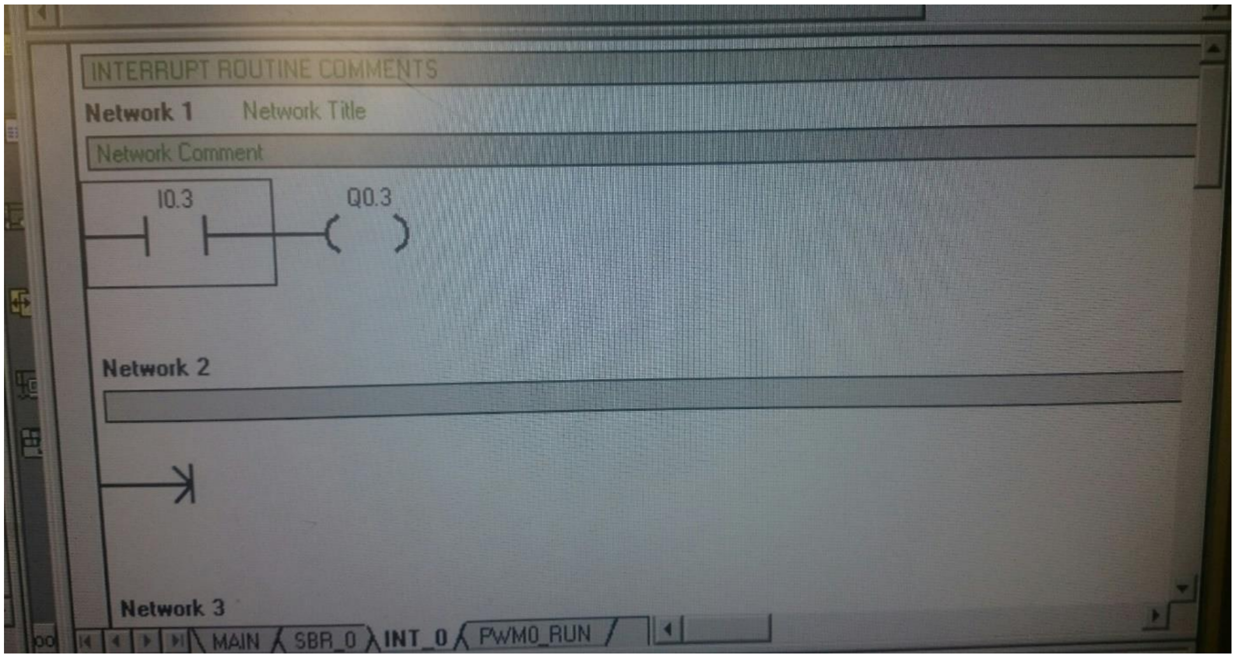


Interrupt Instruction



Symbol	Address	Comment
INT_0	INT0	INTERRUPT ROUTINE COMMENTS


SUBROUTINE COMMENTS
Network 1 Network Title
Network Comment
I0.1
I0.2
PWM0_RUN
EN
RUN
1000 Cycle
500 Pulse
Error VB0
Network 2
MAIN SBR_0 INT_0 PWM0_RUN


INTERRUPT ROUTINE COMMENTS
Network 1 Network Title
Network Comment
I0.3
Q0.3
Network 2
Network 3
MAIN SBR_0 INT_0 PWM0_RUN

Lab Session # 07

Title

Introduction to For Next Instruction (For Loop).

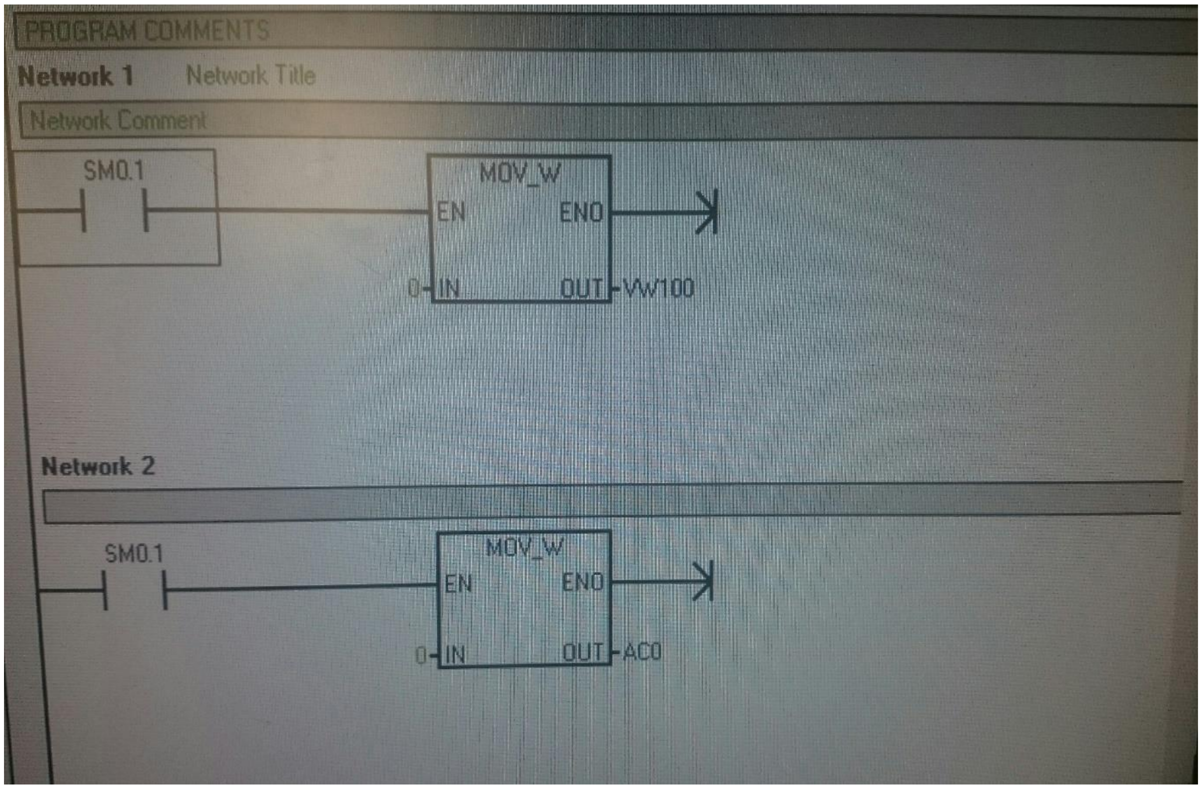
Theory

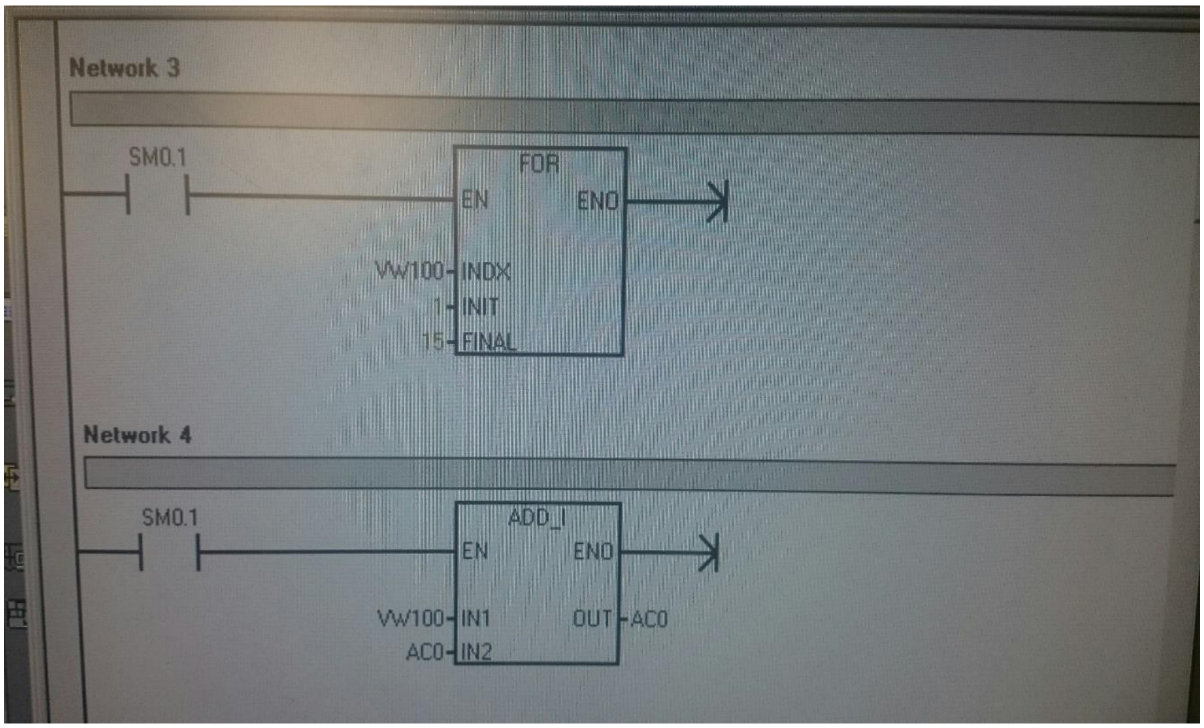
For Next Instruction is used for executing FOR Loop. For instruction have three inputs EN (enable) pin which is used to start for instruction. INDX (Index) pin here we give a memory location in which data stores that tells how many times this loop runs. INT (initial) here we input a constant number from which it starts loop or starting point for loop. FINAL here we input a constant number to which it stops or end the loop or ending point for loop.

In this lab we use special memories SM0.0 (permanently ON) SM0.1 (ON only for 1 cycle).

Example with the application of FOR NEXT Instruction.

Generate a generalized program in which a For Loop runs 10 times.

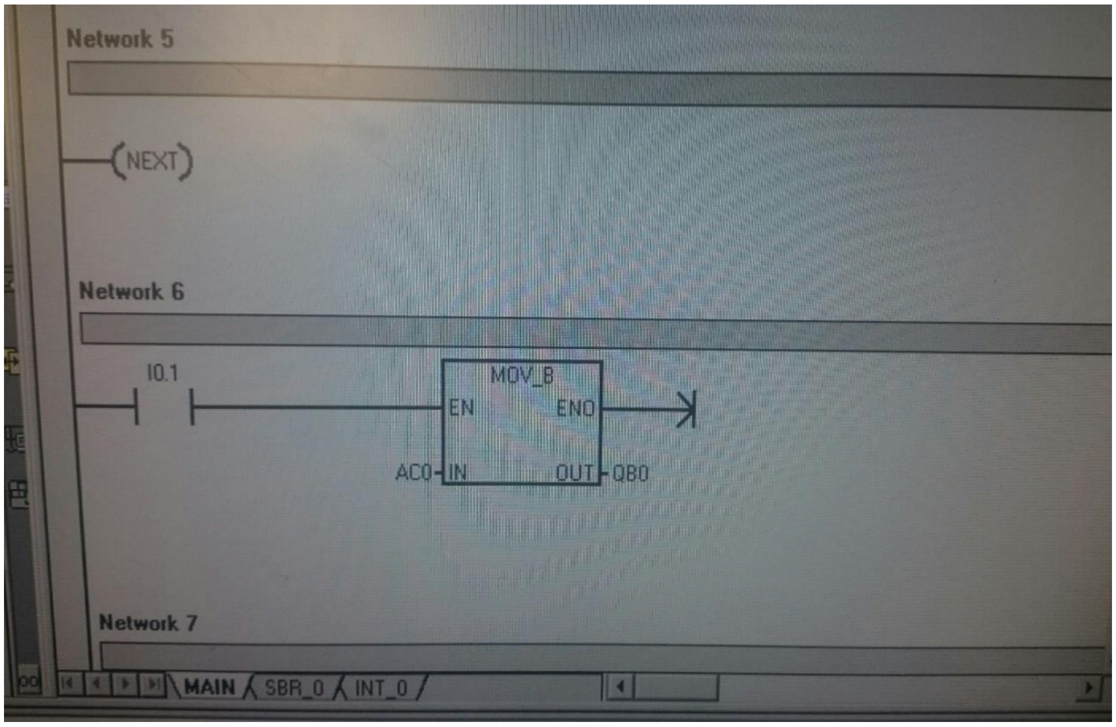



Network 3

SM0.1 — FOR: EN, ENO; VW100 — INDX; 1 — INIT; 15 — FINAL

Network 4

SM0.1 — ADD_I: EN, ENO; VW100 — IN1; AC0 — IN2; OUT — AC0


Network 5

(NEXT)

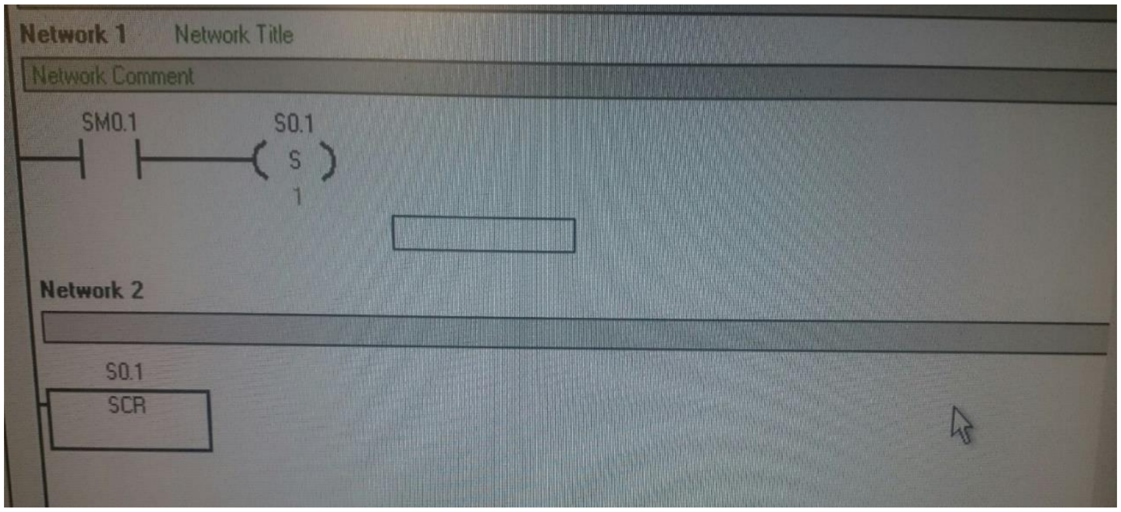
Network 6

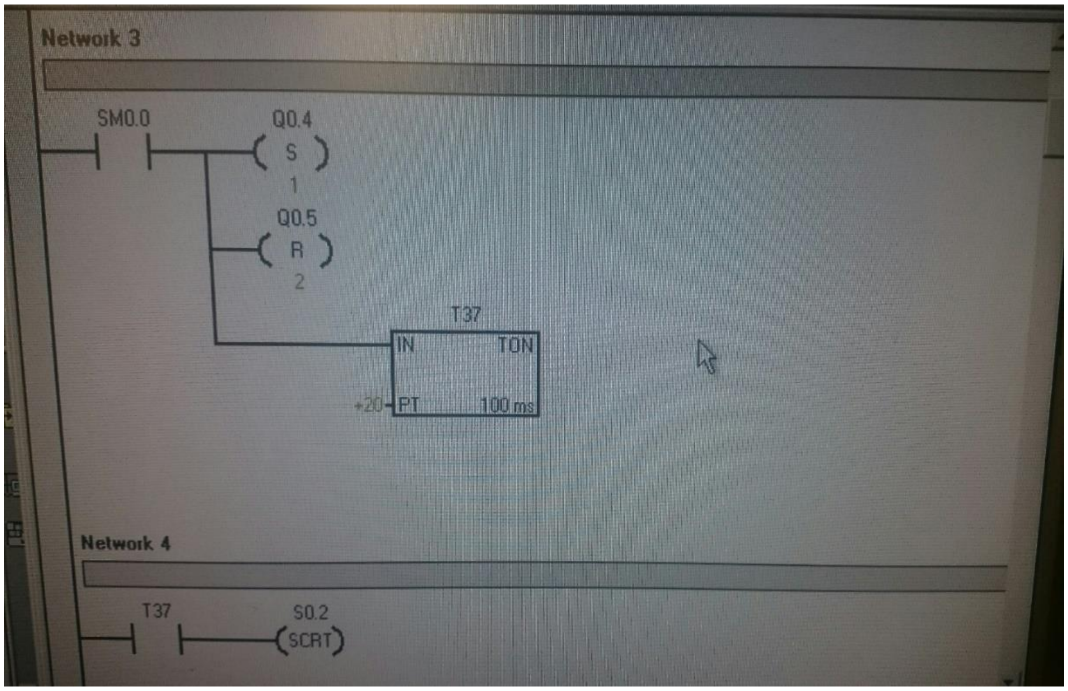
I0.1 — MOV_B: EN, ENO; AC0 — IN; OUT — QB0

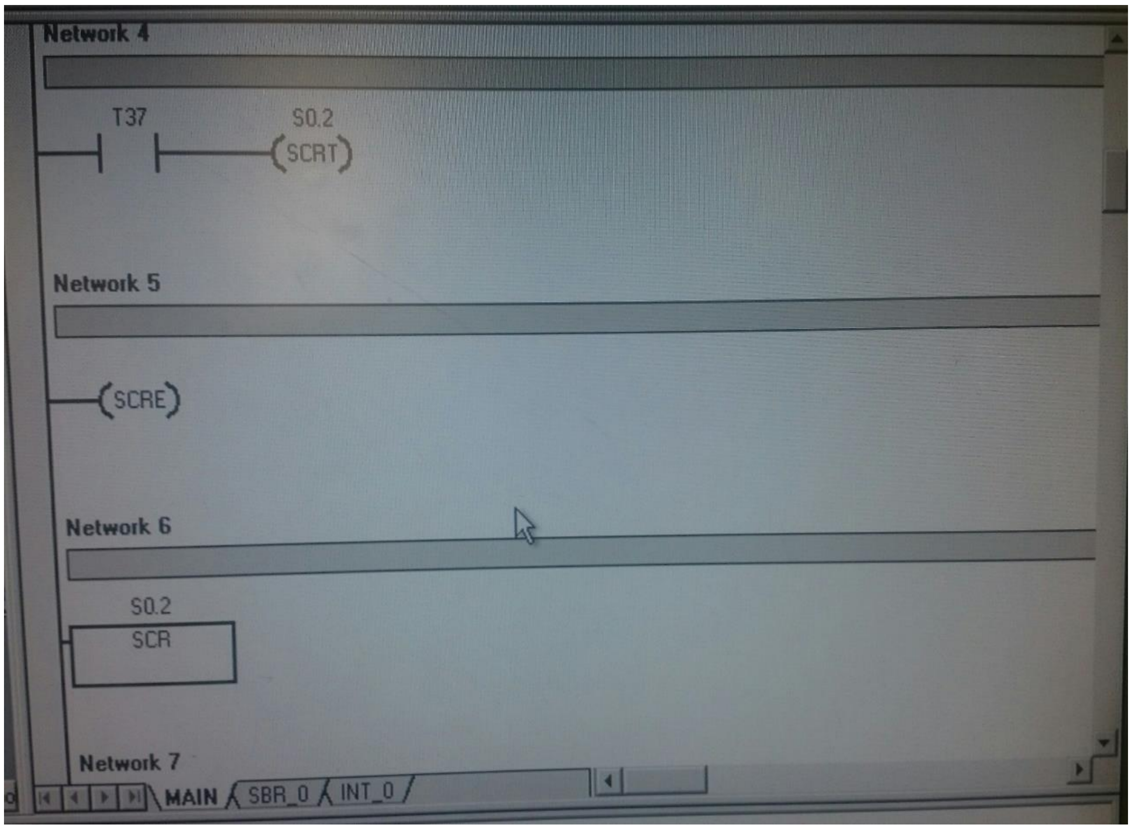
Network 7

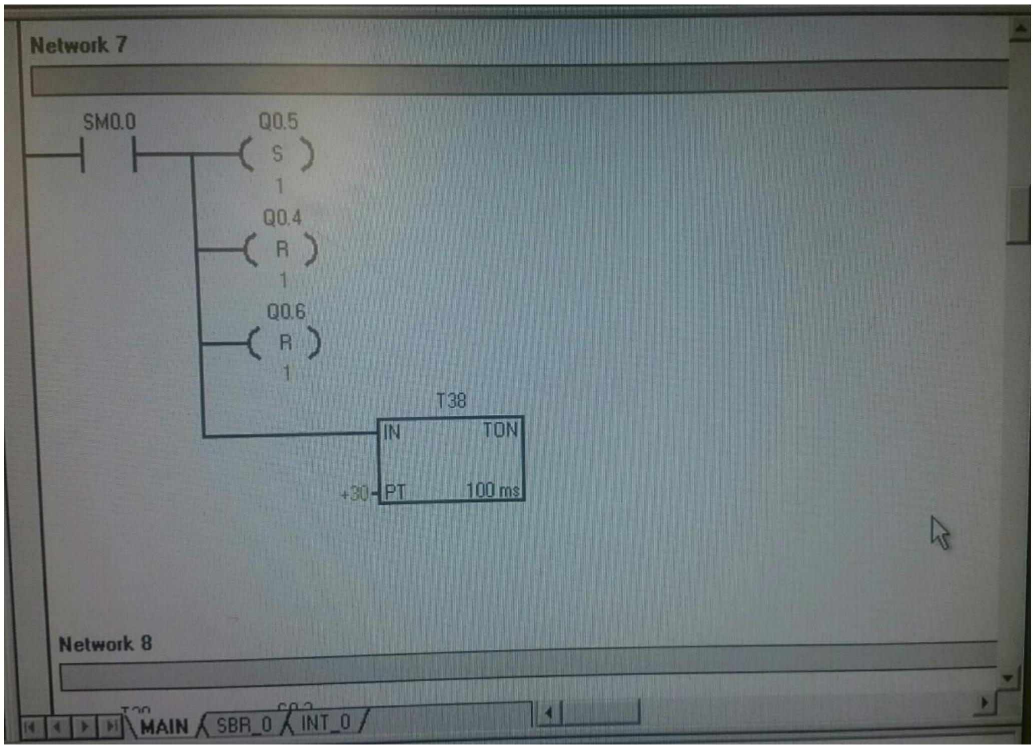
Traffic Control lights

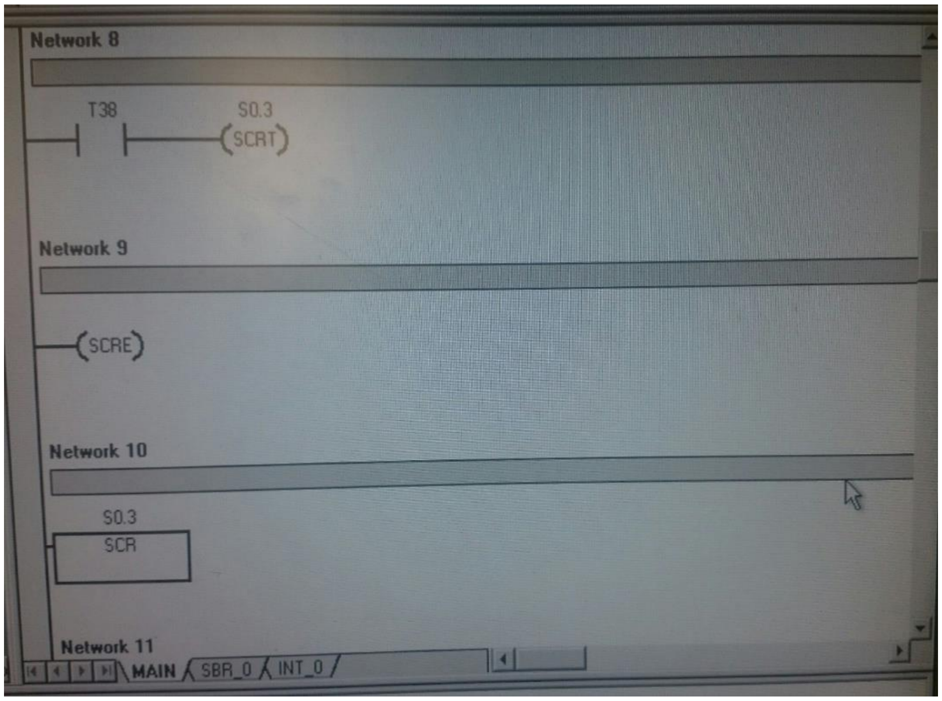
Ladder Logic for Traffic Lights Using SCR

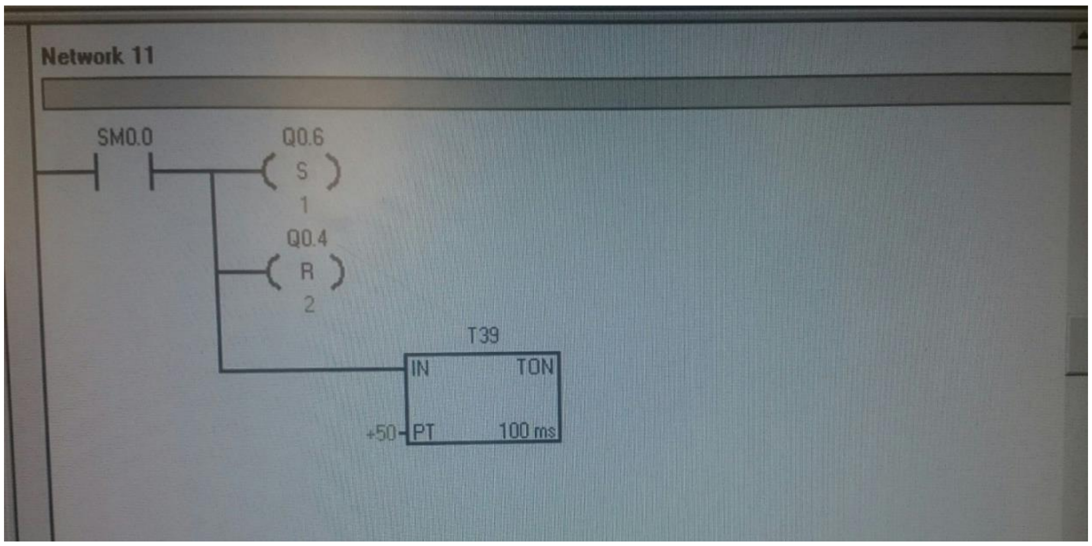


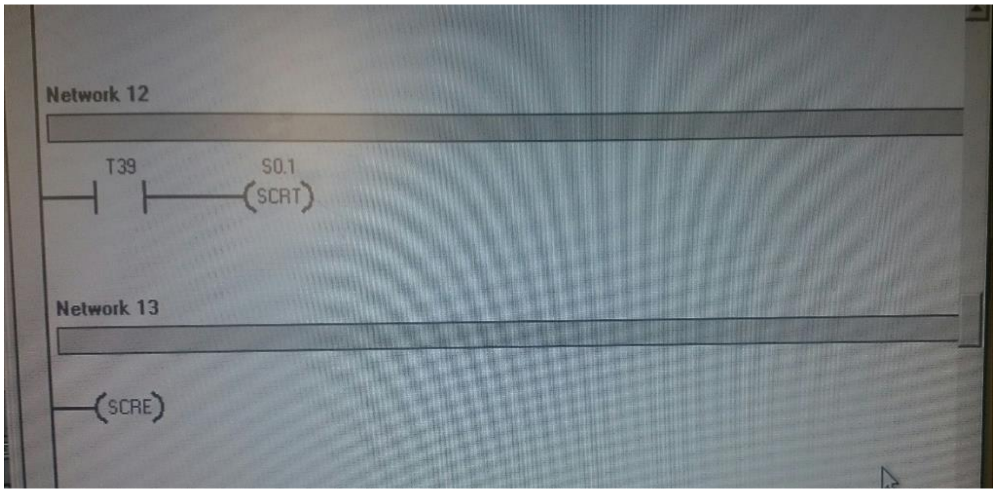



Network 4
T37
S0.2
SCRT
Network 5
SCRE
Network 6
S0.2
SCR
Network 7
MAIN
SBR_0
INT_0


Network 7
SM0.0
Q0.5
S
1
Q0.4
R
1
Q0.6
R
1
T38
IN
TON
+30
PT
100 ms
Network 8
MAIN
SBR_0
INT_0


Network 8
T38
S0.3
SCRT
Network 9
SCRE
Network 10
S0.3
SCR
Network 11
MAIN
SBR_0
INT_0


Network 11
SM0.0
Q0.6
S
1
Q0.4
R
2
T39
IN
TON
+50
PT
100 ms


Network 12
T39
S0.1
SCRT
Network 13
SCRE

Lab Session 08

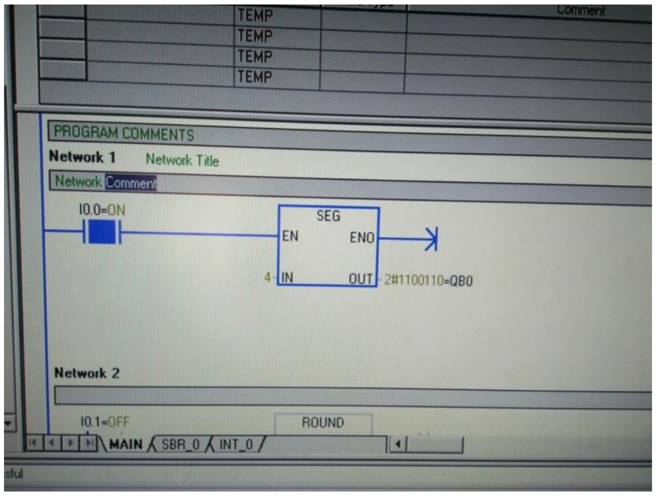
Objective

To study about Instructions

Theory

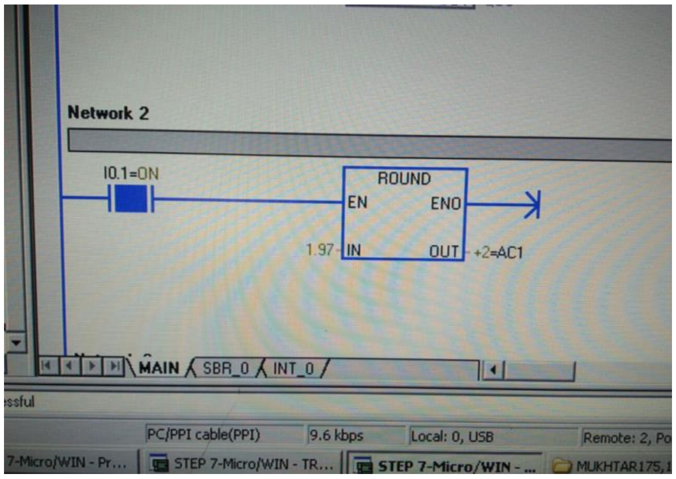
1. Segment

The Segment instruction (SEG) allows you to generate a bit pattern that illuminates the segments of a seven-segment display.



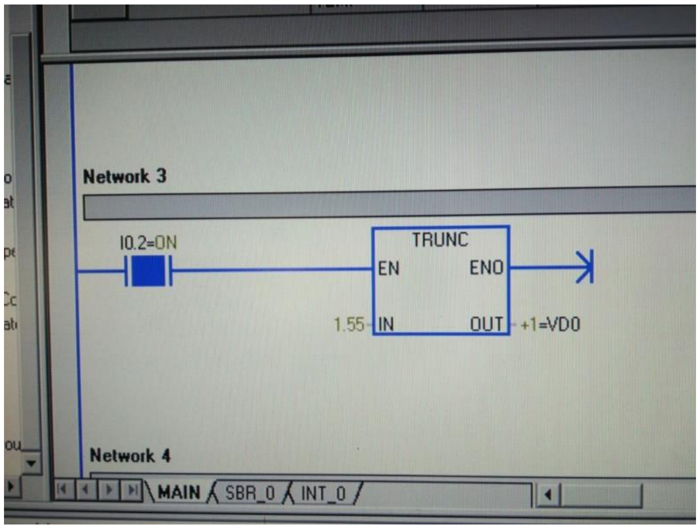
2. Round

The Round instruction (ROUND) converts a real value IN to a double integer value and places the rounded result into the variable specified by OUT.



3. Truncate

The Truncate instruction (TRUNC) converts a real number IN into a double integer and places the whole-number portion of the result into the variable specified by OUT.

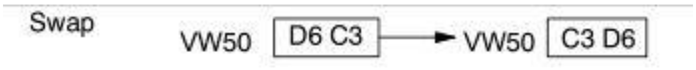


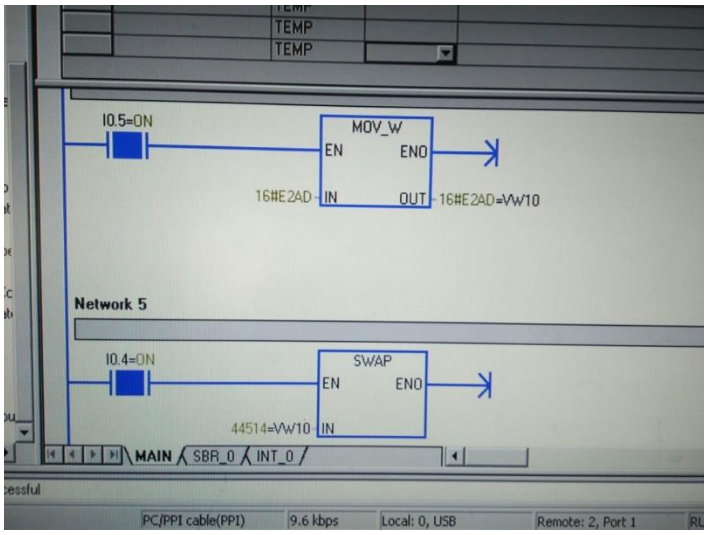
4. Swap Bytes Instruction

The Swap Bytes instruction exchanges the most significant byte with the least significant byte of the word IN.

Error conditions that set ENO = 0

H 0006 (indirect address)





5. <u>Shift</u>

Shift Right and Shift Left Instructions

The Shift instructions shift the input value IN right or left by the shift count N and load the result in the output OUT. The Shift instructions fill with zeros as each bit is shifted out . If the shift count (N) is greater than or equal to the maximum allowed (8 for byte operations, 16 for word operations, and 32 for double word operations), the value is shifted the maximum number of times for the operation. If the shift count is greater than 0, the overflow memory bit (SM1.1) takes on the value of the last bit shifted out. The zero memory bit (SM1.0) is set if the result of the shift operation is zero. Byte operations are unsigned. For word and double word operations, the sign bit is shifted when you use signed data types.

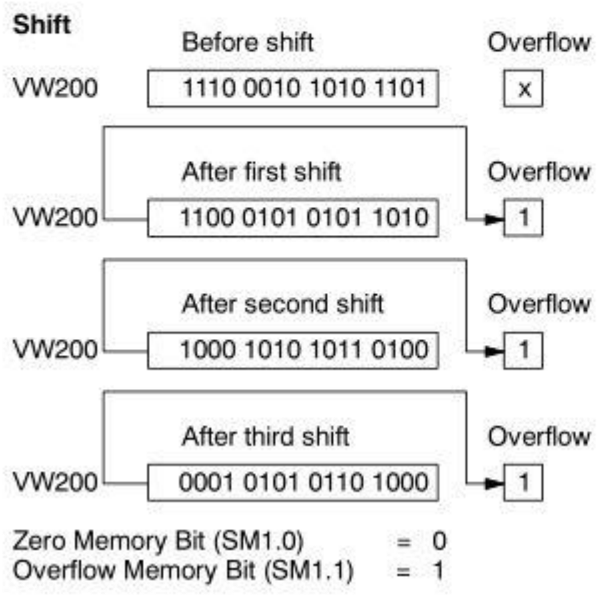
Error conditions that set ENO = 0

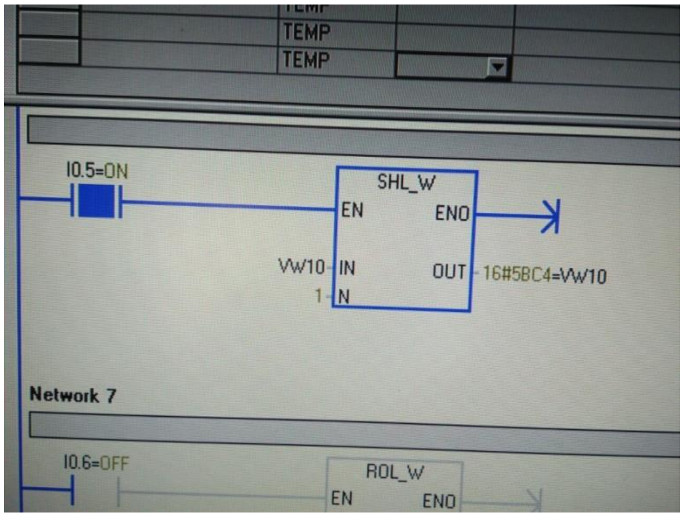
H 0006 (indirect address)

SM bits affected:

H SM1.0 (zero)

H SM1.1 (overflow)





6. <u>Rotate</u>

Rotate Right and Rotate Left Instructions

The Rotate instructions rotate the input value (IN) right or left by the shift count (N) and load the result in the memory location (OUT). The rotate is circular. If the shift count is greater than or equal to the maximum for the operation (8 for a byte operation, 16 for a word operation, or 32 for a double-word operation), the S7-200 performs a modulo operation on the shift count to obtain a valid shift count before the rotation is executed. This result is a shift count of 0 to 7 for byte operations, 0 to 15 for word operations, and 0 to 31 for double-word operations. If the shift count is 0, a rotate operation is not performed. If the rotate operation is performed, the value of the last bit rotated is copied to the overflow bit (SM1.1).

If the shift count is not an integer multiple of 8 (for byte operations), 16 (for word operations), or 32 (for double-word operations), the last bit rotated out is copied to the overflow memory bit (SM1.1). The zero memory bit (SM1.0) is set when the value to be rotated is zero. Byte operations are unsigned. For word and double word operations, the sign bit is shifted when you use signed data types.

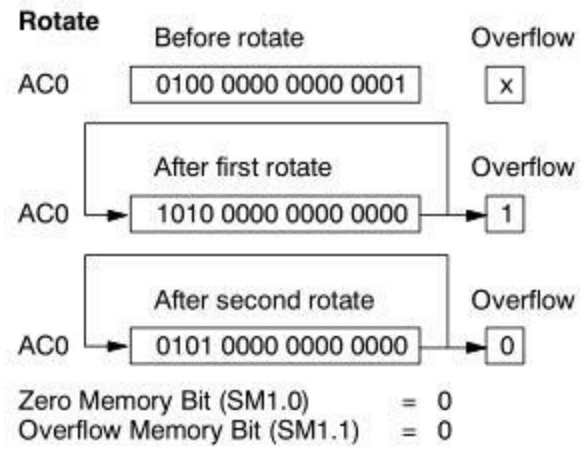
Error conditions that set ENO = 0

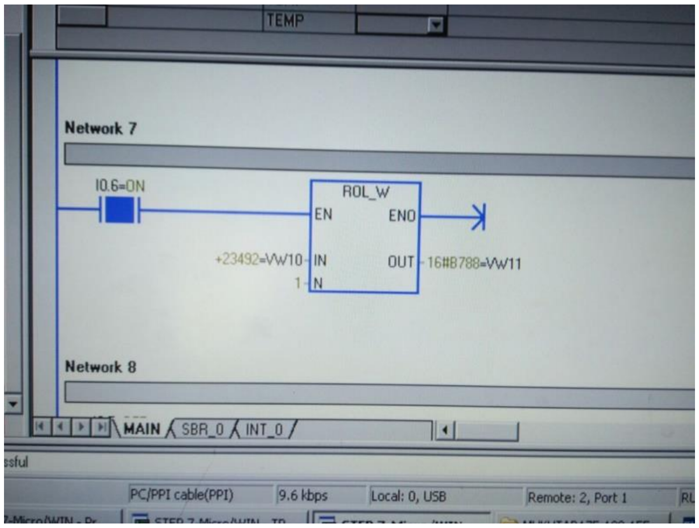
H 0006 (indirect address)

SM bits affected:

H SM1.0 (zero)

H SM1.1 (overflow)



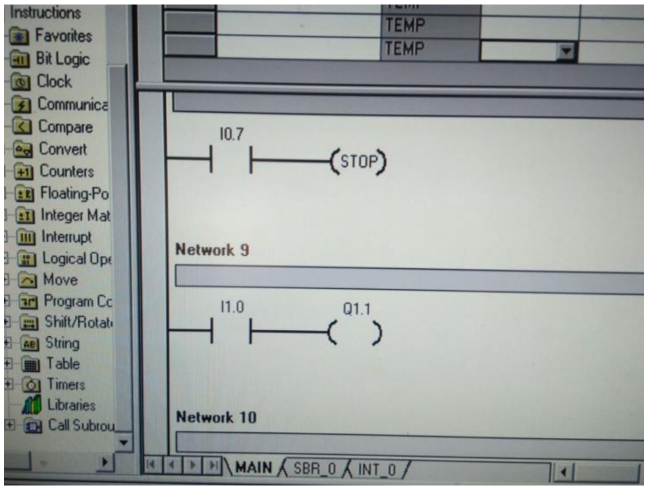


7. Conditional End

The Conditional End instruction (END) terminates the current scan based upon the condition of the preceding logic. You can use the Conditional End instruction in the main program, but you cannot use it in either subroutines or interrupt routines.

Stop

The Stop instruction (STOP) terminates the execution of your program by causing a transition of the S7-200 CPU from RUN to STOP mode. If the Stop instruction is executed in an interrupt routine, the interrupt routine is terminated immediately, and all pending interrupts are ignored. Remaining actions in the current scan cycle are completed, including execution of the main user program, and the transition from RUN to STOP mode is made at the end of the current scan.



8. Add To Table

The Add To Table instruction adds word values (DATA) to a table (TBL). The first value of the table is the maximum table length (TL). The second value is the entry count (EC), which specifies the number of entries in the table. New data are added to the table after the last entry. Each time new data are added to the table, the entry count is incremented. A table can have up to 100 data entries.

Error conditions that set ENO = 0

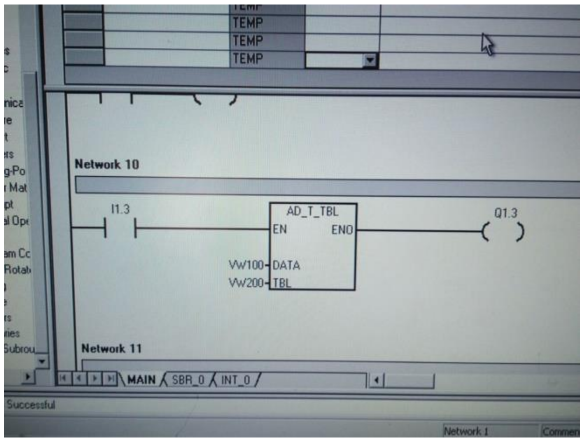
H SM1.4 (table overflow)

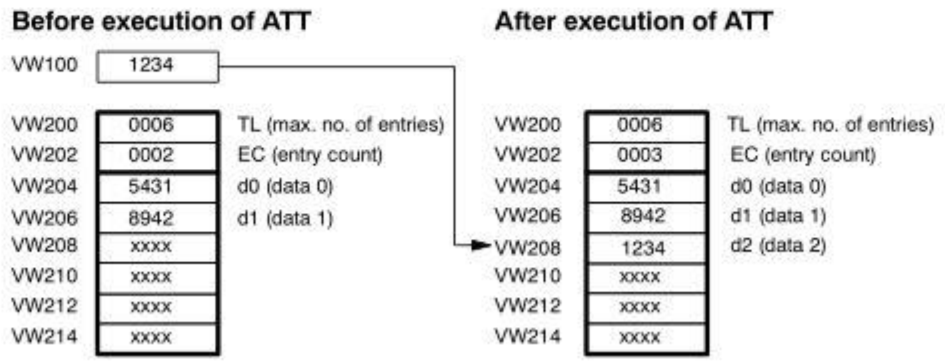
H 0006 (indirect address)

H 0091 (operand out of range)

SM bits affected:

H SM1.4 is set to 1 if you try to overfill the table



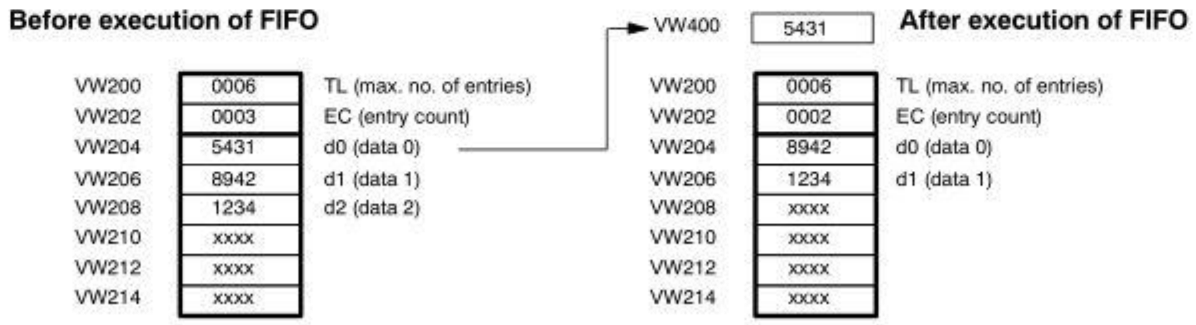


<u>**First-In-First-Out and Last-In-First-Out**</u>

A table can have up to 100 data entries.

First-In-First-Out

The First-In-First-Out instruction (FIFO) moves the oldest (or first) entry in a table to the output memory address by removing the first entry in the table (TBL) and moving the value to the location specified by DATA. All other entries of the table are shifted up one location. The entry count in the table is decremented for each instruction execution.



Last-In-First-Out

The Last-In-First-Out instruction (LIFO) moves the newest (or last) entry in the table to the output memory address by removing the last entry in the table (TBL) and moving the value to the location specified by DATA. The entry count in the table is decremented for each instruction execution.

Error conditions that set ENO = 0

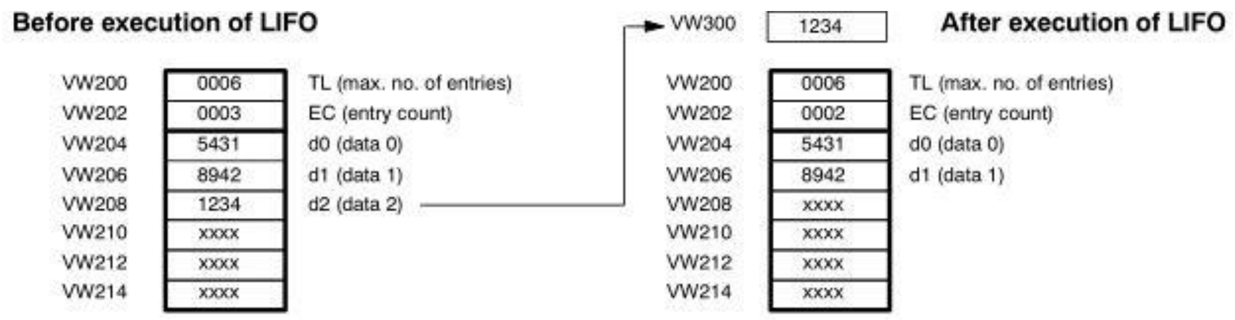
H SM1.5 (empty table)

H 0006 (indirect address)

H 0091 (operand out of range)

SM bits affected:

H SM1.5 is set to 1 if you try to remove an entry from an empty table



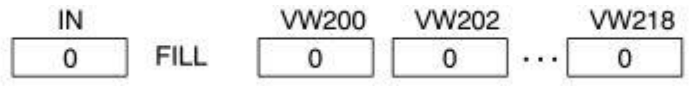
9. Memory Fill

The Memory Fill instruction (FILL) writes N consecutive words, beginning at address OUT, with the word value contained in address IN. N has a range of 1 to 255.

Error conditions that set ENO = 0

H 0006 (indirect address)

H 0091 (operand out of range)



10. Table Find

The Table Find instruction (FND) searches a table for data that matches certain criteria. The Table Find instruction searches the table TBL, starting with the table entry INDX, for the data value or pattern PTN that matches the search criteria defined by CMD. The command parameter CMD is given a numeric value of 1 to 4 that corresponds to =, <>, <, and >, respectively.

If a match is found, the INDX points to the matching entry in the table. To find the next matching entry, the INDX must be incremented before invoking the Table Find instruction again. If a match is not found, the INDX has a value equal to the entry count. A table can have up to 100 data entries. The data entries (area to be searched) are numbered from 0 to a maximum value of 99.

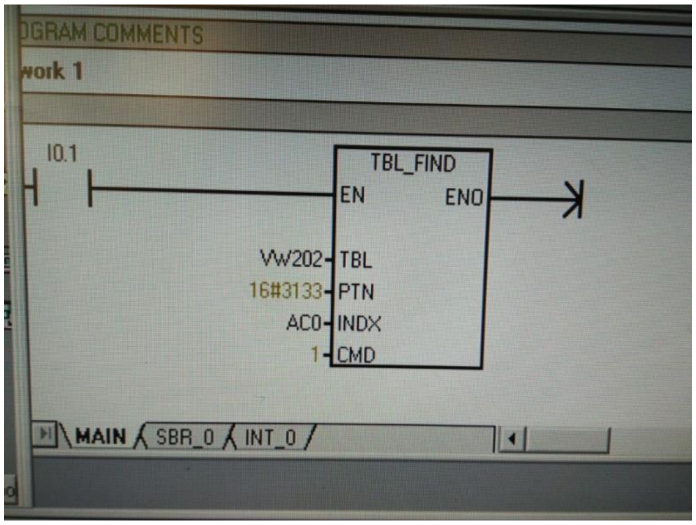
Error conditions that set ENO = 0

H 0006 (indirect address)

H 0091 (operand out of range)

Table format for TBL_FIND

VW202	0006	EC (entry count)
VW204	xxxx	d0 (data 0)
VW206	xxxx	d1 (data 1)
VW208	xxxx	d2 (data 2)
VW210	xxxx	d3 (data 3)
VW212	xxxx	d4 (data 4)
VW214	xxxx	d5 (data 5)



Lab Session # 11

CX- Programming 8.1 (Omron PLC)

Objective:

Programming of different cards.

Introduction to Omron:

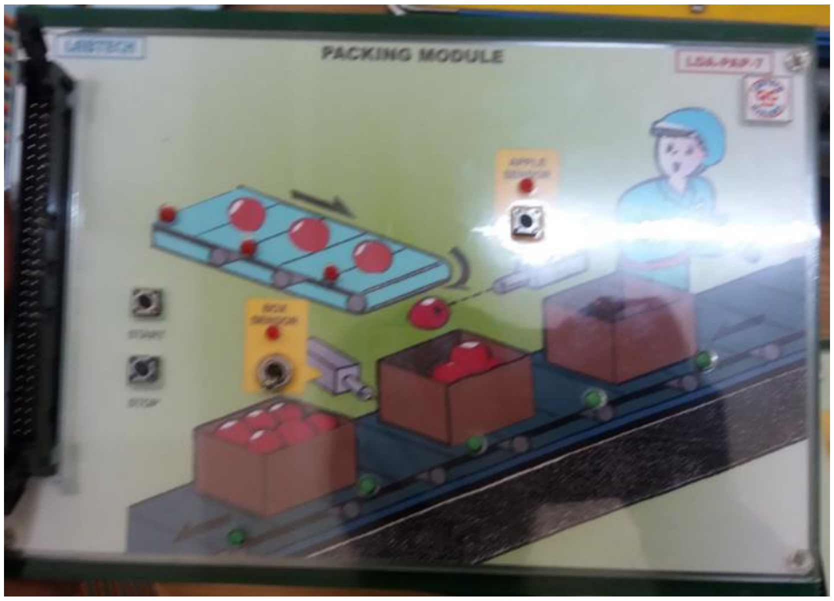
Omron was established by Kazuma Tateishi in 1933 (as the Tateishi Electric Manufacturing Company) and incorporated in 1948. The company originated in an area of Kyoto called "Omuro’’, from which the name "OMRON" was derived. Prior to 1990, the corporation was known as Omron Tateisi Electronics. During the 1980s and early 1990s, the company motto was: "To the machine the work of machines, to man the thrill of further creation".

Omron's primary business is the manufacture and sale of automation components, equipment and systems, but it is generally known for medical equipment such as digital thermometers, blood pressure monitors and nebulizers. Omron developed the world's first electronic ticket gate, which was named an IEEE Milestone in 2007, and was one of the first manufacturers of automated teller machines(ATM) with magnetic stripe card readers.

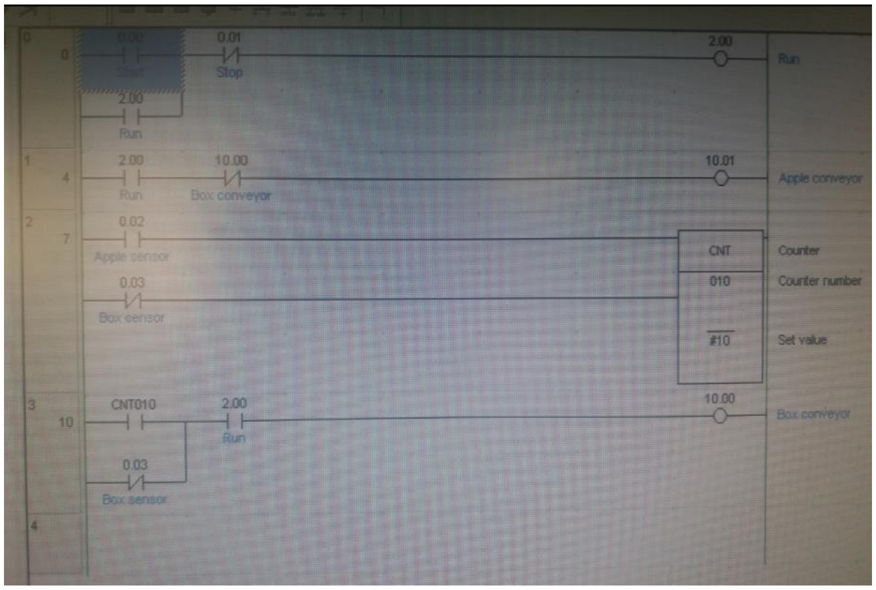
Packing Module:

Two sensors and two conveyer belts works at a time. Box travels on one conveyer belt and workpiece on the second conveyer. First sensor attach with the counter which count the work piece (10 pieces) and other sensor detect the boxes. When box is fill with ten workpieces then it will move forward and this process continue.

Diagram:



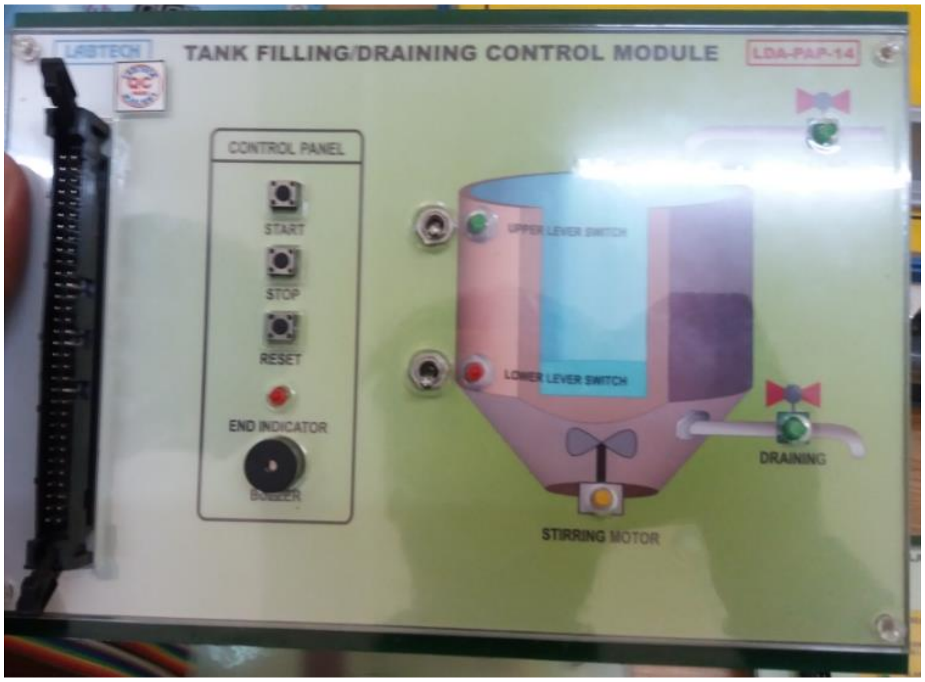
Coding:



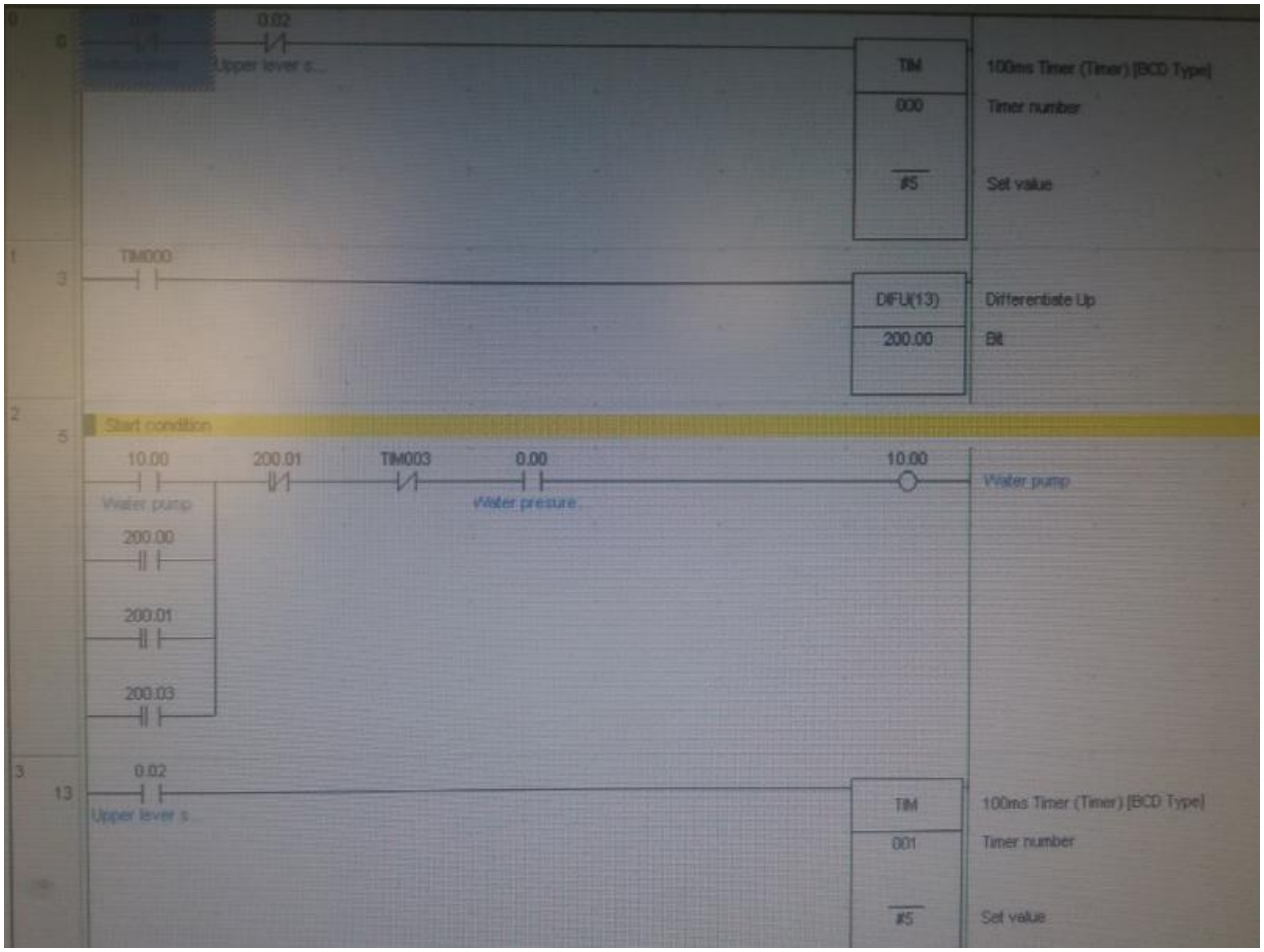
Tank Filling/Draining control module:

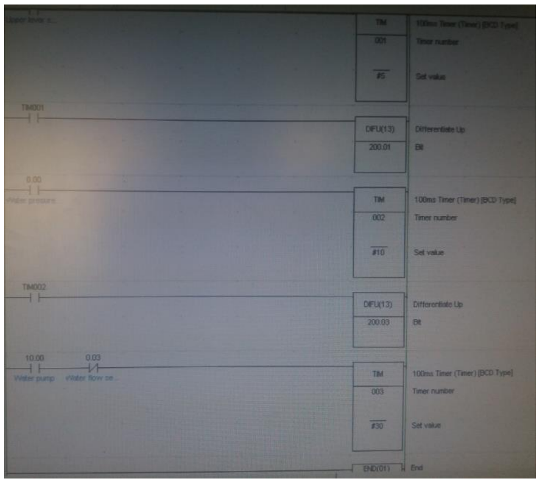
Two sensors are used to control the water level. One sensor tells about upper level of water and other tells about lower level of water. When both sensors gives 0 value then filling process occur when both sensors give 1 then draining is completing and stirring motor is also working.

Diagram:



Coding:

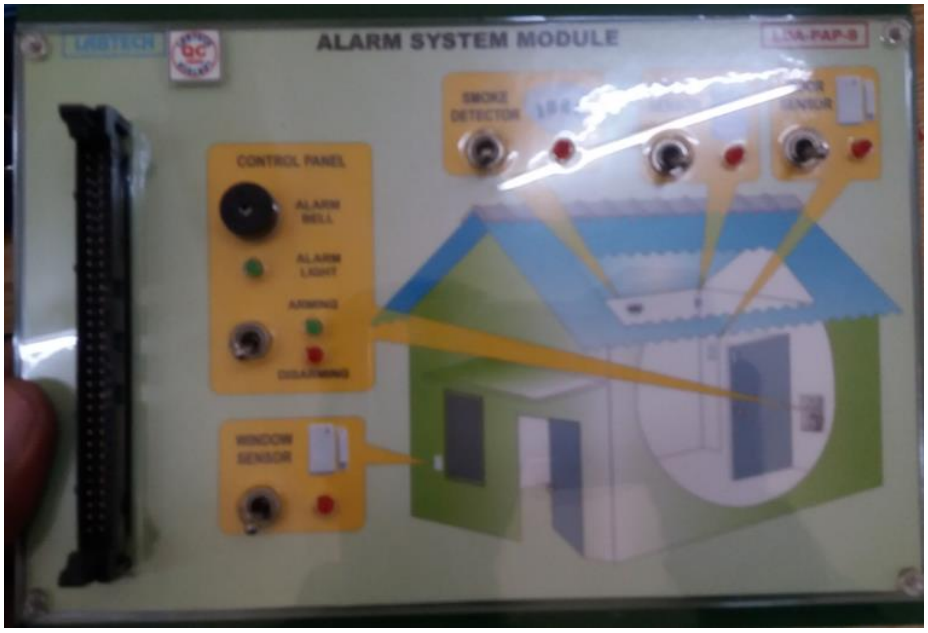




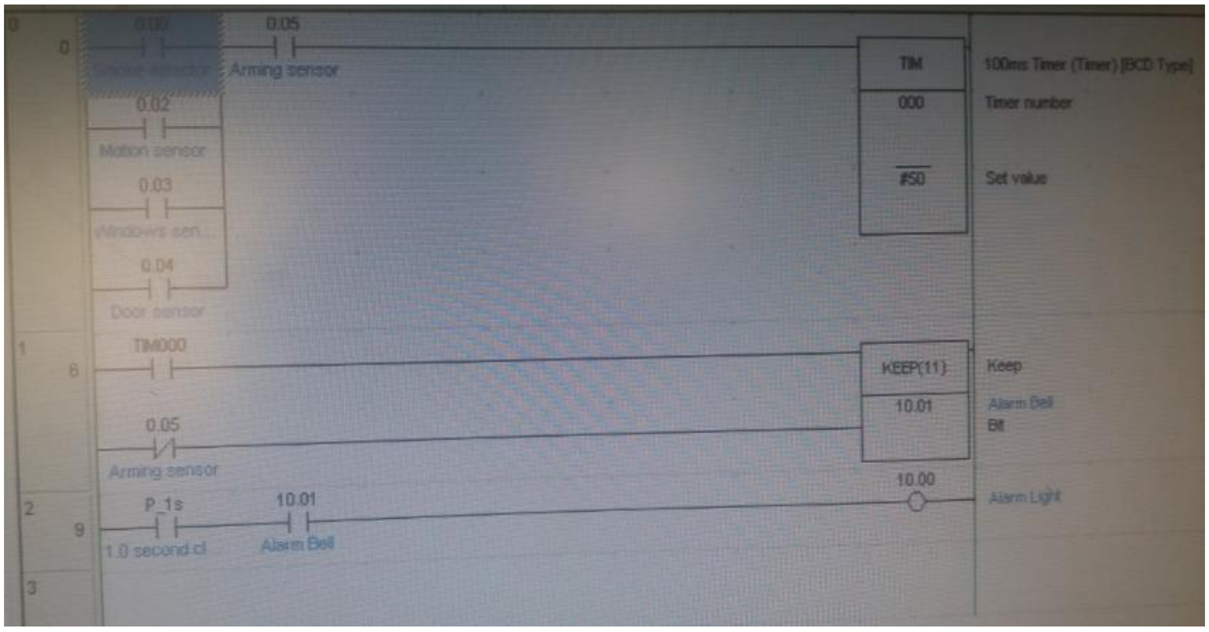
Alarm System module:

A security alarm is a system designed to detect intrusion – unauthorized entry – into a building or other area. Security alarms are used in residential, commercial, industrial, and military properties for protection against burglary (theft) or property damage, as well as personal protection against intruders. Security alarms in residential areas show a correlation with decreased theft.[1] Car alarms likewise help protect vehicles and their contents. Prisons also use security systems for control of inmates. Some alarm systems serve a single purpose of burglary protection; combination systems provide both fire and intrusion protection.

Diagram:



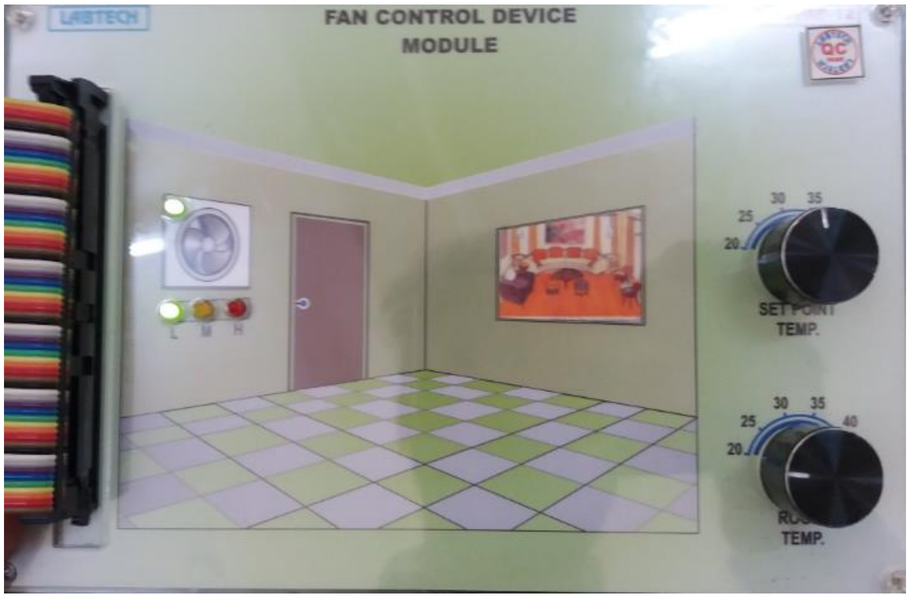
Coding:



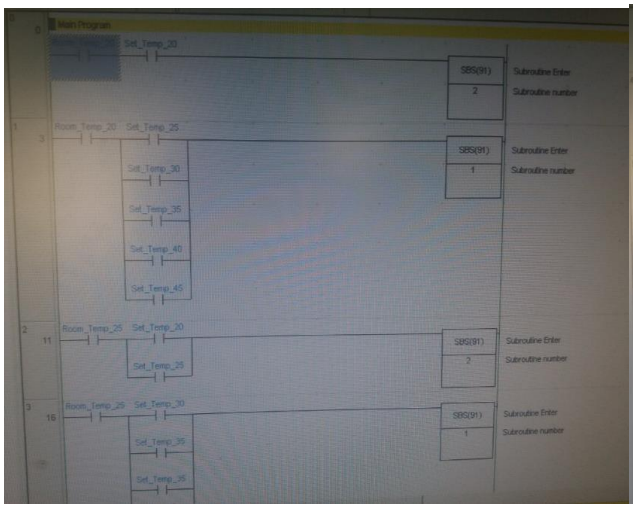
Fan control device Module:

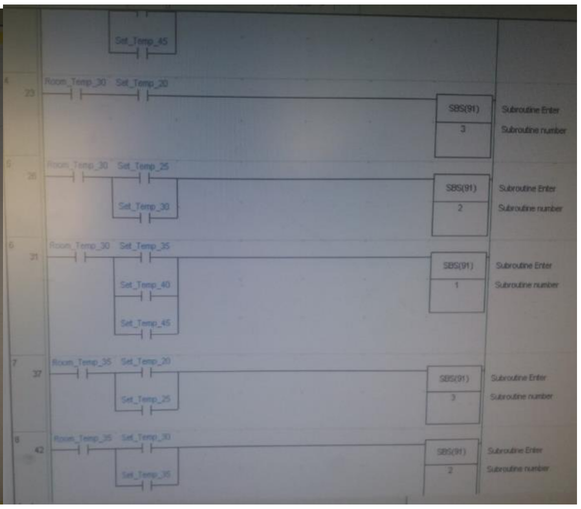
Speed Fan is a system monitor for that can read temperatures, voltages and fan speeds of room. It can change fan speeds depending on the temperature of the room. The program can display system variables as charts and as an indicator in the system tray. Fully configurable user events can be defined to execute specific actions based on system status.

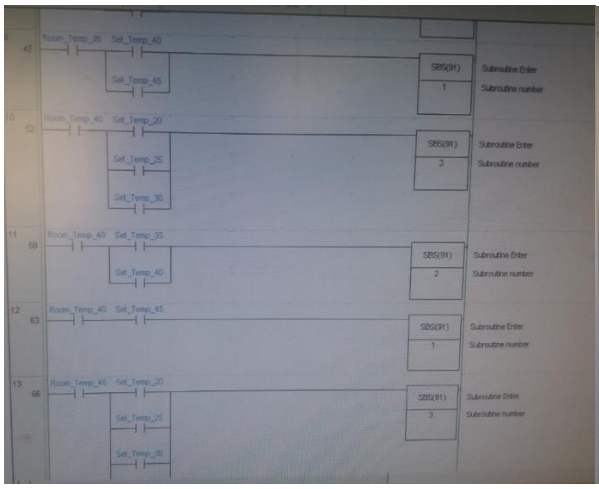
Diagram:

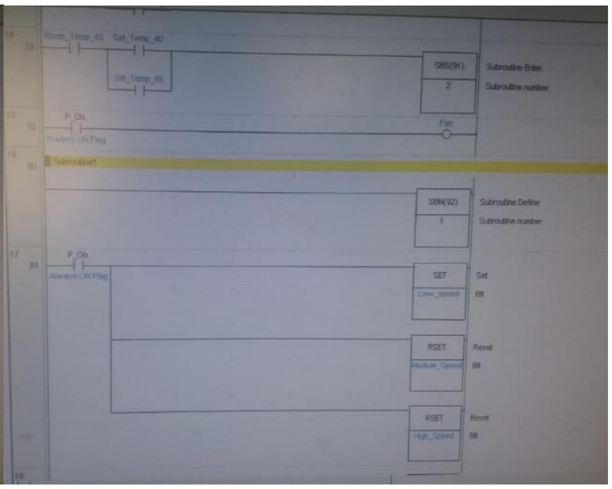


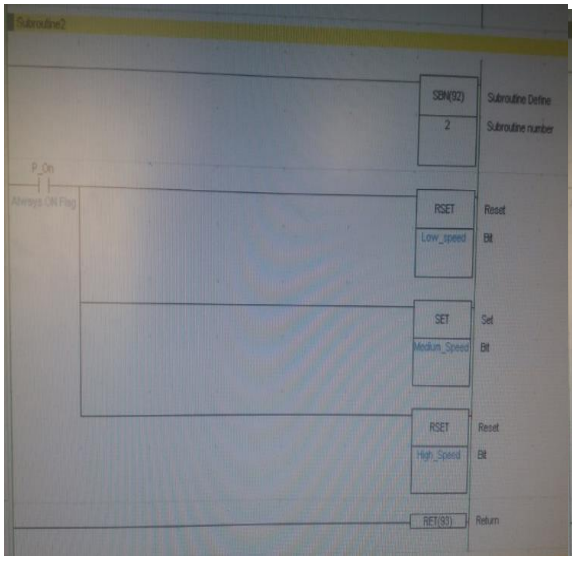
Coding:

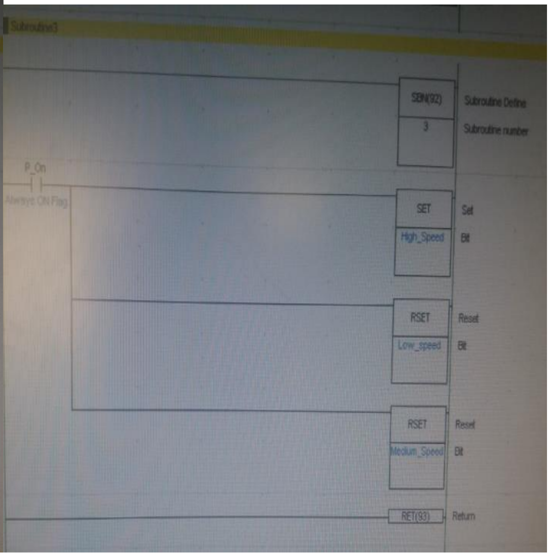








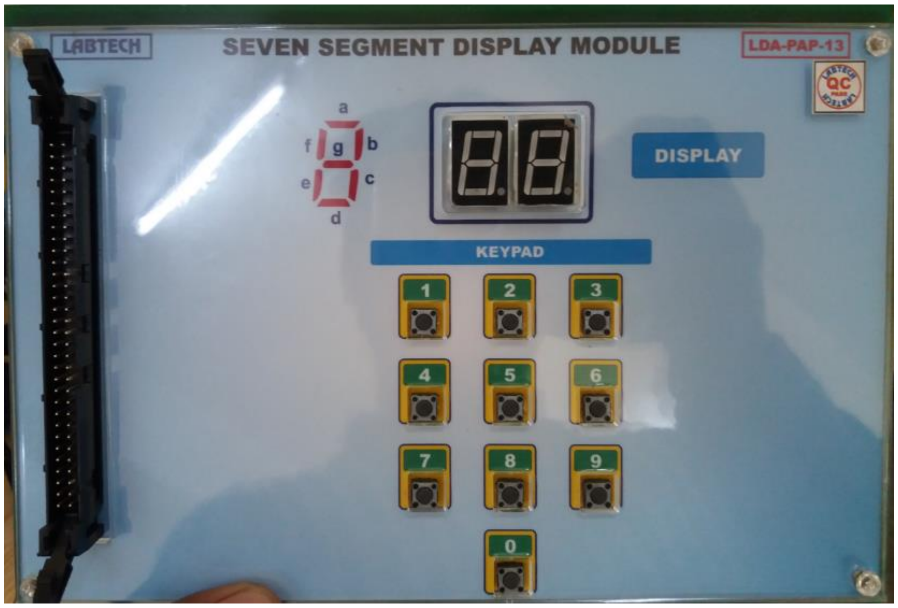




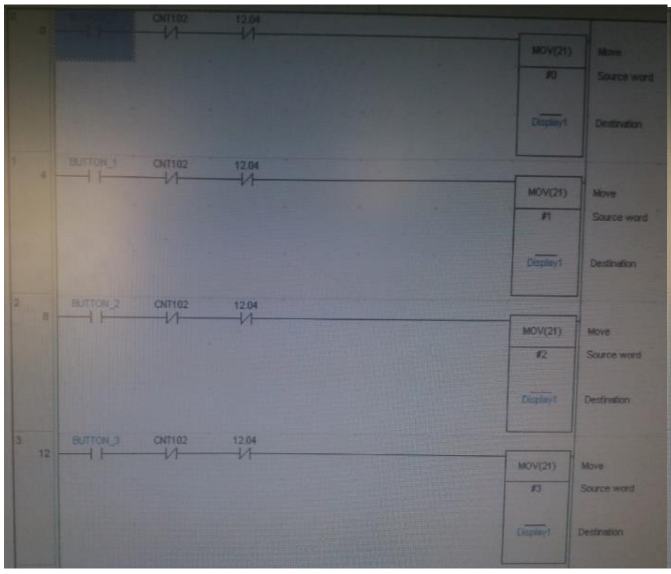
Seven segment display module:

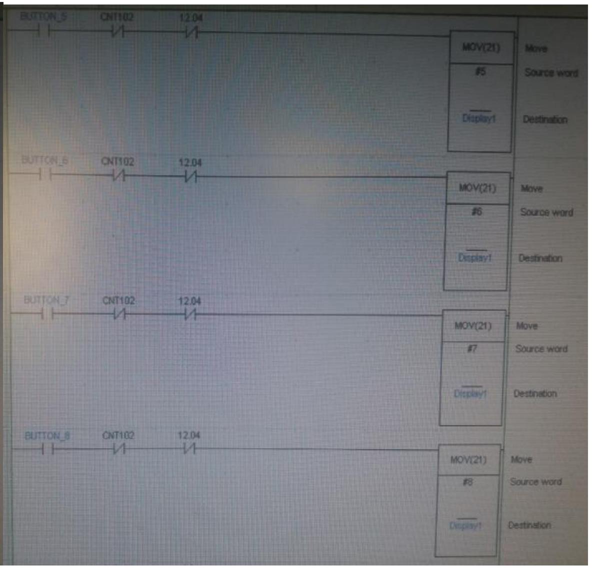
A seven-segment display (SSD), or seven-segment indicator, is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays. Seven-segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information

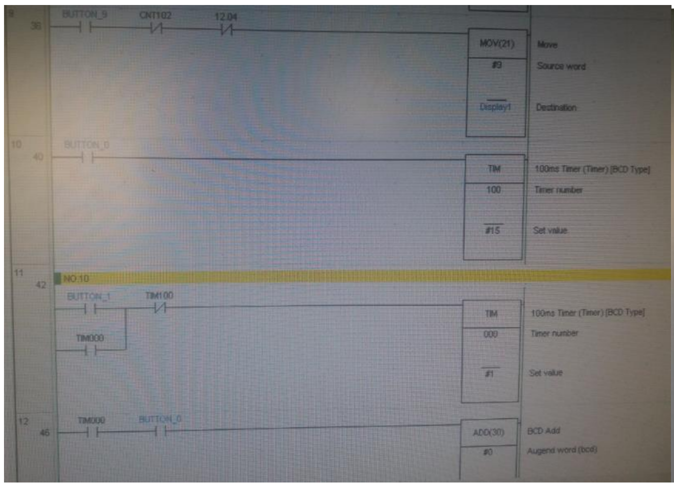
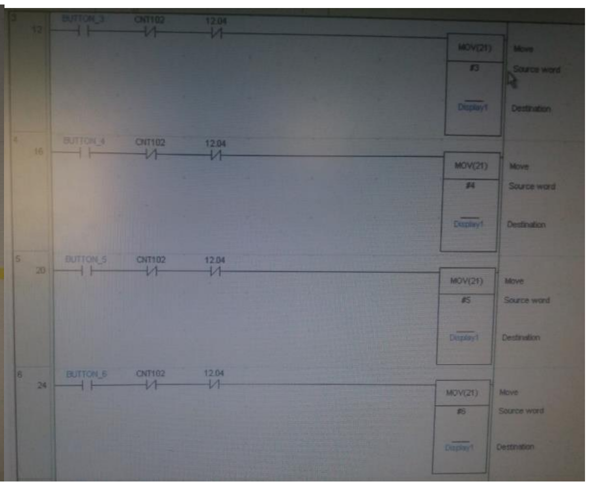
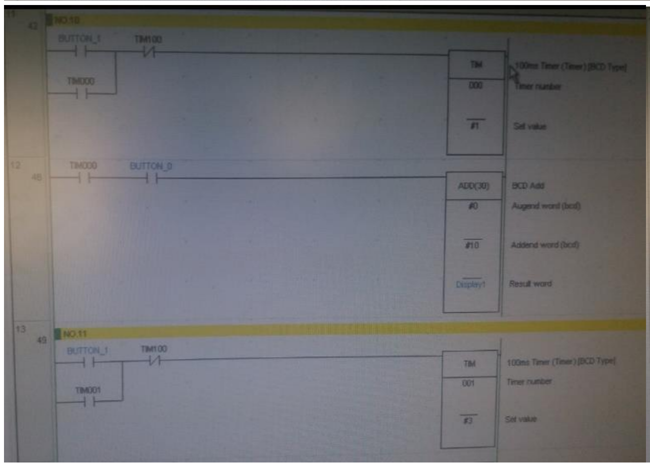
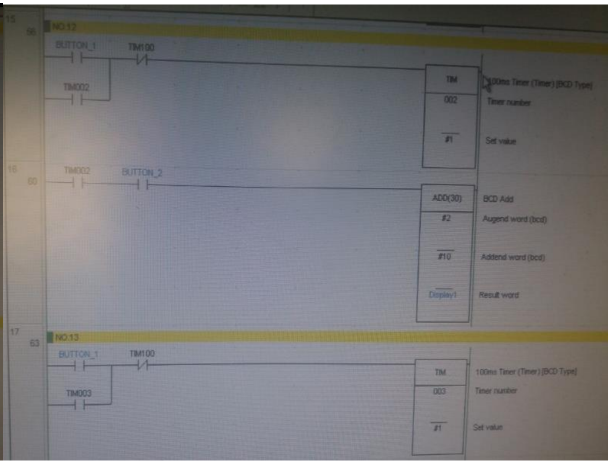
Diagram:



Coding:






BUTTON_9
CNT102
12.04
MOV(21)
Move
#9
Source word
Display1
Destination
BUTTON_0
TIM
100ms Timer (Timer) [BCD Type]
100
Timer number
#15
Set value
NO:10
BUTTON_1
TIM100
TIM000
TIM
000
Timer number
#1
Set value
TIM000
BUTTON_0
ADD(30)
BCD Add
#0
Augend word (bcd)

BUTTON_3
CNT102
12.04
MOV(21)
Move
#3
Source word
Display1
Destination
BUTTON_4
MOV(21)
#4
BUTTON_5
MOV(21)
#5
BUTTON_6
MOV(21)
#6

NO:10
BUTTON_1
TIM100
TIM000
TIM
000
100ms Timer (Timer) [BCD Type]
Timer number
#1
Set value
TIM000
BUTTON_0
ADD(30)
BCD Add
#0
Augend word (bcd)
#10
Addend word (bcd)
Display1
Result word
NO:11
TIM001
TIM
001
#3

NO:12
BUTTON_1
TIM100
TIM002
TIM
002
100ms Timer (Timer) [BCD Type]
Timer number
#1
Set value
TIM002
BUTTON_2
ADD(30)
BCD Add
#2
Augend word (bcd)
#10
Addend word (bcd)
Display1
Result word
NO:13
TIM003
TIM
003
#1