Skills for Life!

Jubail Technical Institute
معهد الجبيل التقني

Department of Electrical and Electronics Skills

STUDENT HANDBOOK

PLC Programming and Interfacing & Microcontroller

INCT 2431

2020

Prepared by

Industrial Instrumentation & Control Skills Team

PLC Programming & Interfacing

INCT 2431

Table of Contents

SECTION-1: PLC

SECTION-2: 8051 MicroController

Preface

PLC's (Programmable Logic Controllers) have nowadays become a very essential part of any industrial plant. The PLC acts like a brain of any plant system. With the help of PLC's, the plant runs smoothly and with all required protection, safety and interlocking. Unlike relay logic any modification can be made easily with help of programming language.

Different manufacturers of PLC's are available in the market, with some difference in architecture, PLC's are similar. Most plc's use three types of programming languages, but most popular and widely used language is **Ladder Logic**.

The goal of this handbook is to provide basic programming skills and troubleshooting technique. After familiarization with some important ladder logic instruction students would be able to use PLC effectively in any process.

SECTION-1

PLC

This Page is Left Blank

Chapter-1: Programmable Logic Controller (PLCs) Overview

Objectives: Upon completion of this chapter, the student should be able to:

1- Define Programmable Logic Controller (PLC).
2- Know the history of PLC.
3- Understand advantages of PLC.
4- Name programming languages of PLC.

What does PLC stand for?

Programmable Logic Controller

What is a PLC?

A PROGRAMMABLE LOGIC CONTROLLER (PLC) is an industrial computer control system that continuously monitors the state of input devices and makes decisions based upon a custom program to control the state of output devices.

1.01 History of PLC

The first Programmable Logic Controllers were designed and developed by Modicon as a relay re-placer for GM and Landis.

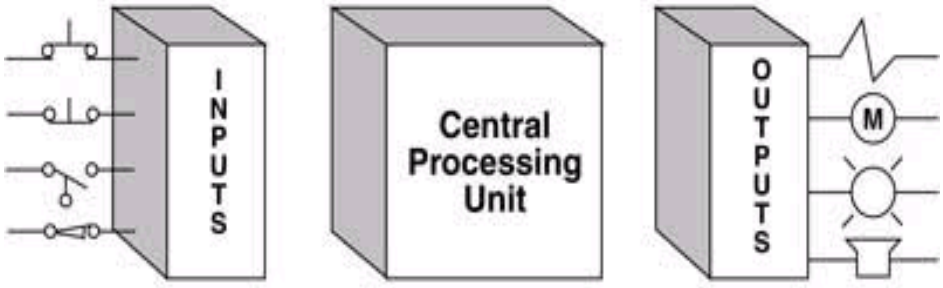
- The first PLC, model 084, was invented by Dick Morley in 1969.
- The first commercial successful PLC, the 184, was introduced in 1973 and was designed by Michael Greenberg.

1.02 What are the advantages of PLC?

1- The PLC eliminated the need for rewiring and adding additional hardware for each new configuration of logic.
2- The PLC drastically increased the functionality of the controls while reducing the cabinet space that housed the logic.

1.03 What is Inside a PLC?

1- Input interface.
2- Central Processing Unit (CPU).
3- Output Interface.



1- The input interface receives the information from external input devices.
2- The Central Processing Unit, the CPU, contains internal hardware and software that tells the PLC how to execute the Control Instructions contained in the user's Programs. It is the brain of the PLC. It has memory and the electronic circuits that process all the data in the PLC.
3- The output interface sends signals to external output devices.

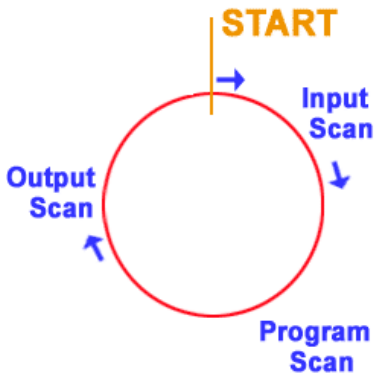
1.04 The Development of PLCs

The original PLC was design to do ON-OFF controls. The modern computer control system of today allowed the PLC to be used in different type of controls in the instrumentation and control field. Now days, PLC exist in many different industries such as manufacturing, petrochemical, food, and many more. It is no longer limited to ON-OFF control. It can do continuous PID control. It is very efficient and precise control machine.

1.05 How Does a PLC Operate?

There are three basic steps in the operation of all PLCs; Input Scan, Program Scan, and Output Scan. These steps continually take place in a repeating loop.

1-Input Scan: Detects the state of all input devices that are connected to the PLC
2-Program Scan: Scans the program from top to bottom. Executes the user created program logic.
3-Output Scan: Energizes or de-energize all output devices that are connected to the PLC. The output of PLC depends on the status of the input and the user's program.



These steps are continually processed in a loop

1.06 What Programing Languages are used to Program a PLC?

1-Ladder Logic Diagram (LAD).
2-Function Block Diagram (FBD).
3-Statement List (STL)

Test your knowledge:

Answer the questions and show your work to the instructor.

1- What are different components in PLC?

2- What are the advantages of PLCs over hard-wired relay?

3- What are the programmable languages used In PLC?

Chapter-2: PLC Instructions and Logic

Objectives: Upon completion of this chapter, the student should be able to:

1- Understand basic PLC instructions.
2- Read a PLC program as logic circuit.
3- Convert logic circuit to a PLC program.
4- Run, and test a logic PLC control program.

2.01 Basic PLC Instructions

In this section we will learn the meaning of the PLC instructions that are used in this book. Below is a table showing instructions and meaning.

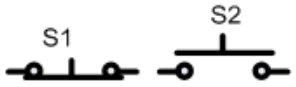
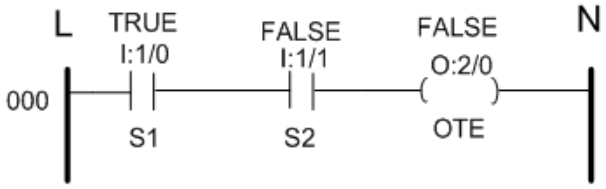
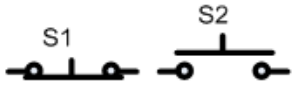
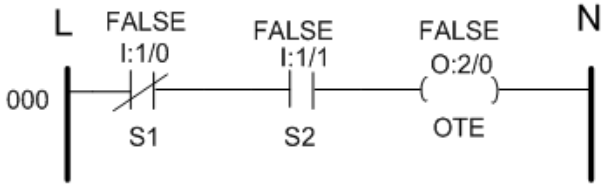
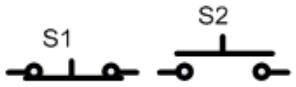
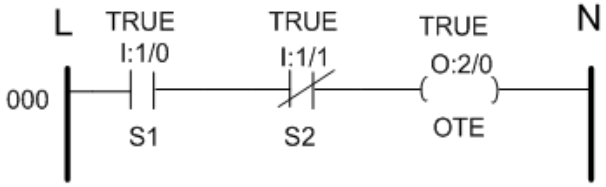
Instruction symbol	Name of instruction
	Examine if closed
-We look at the actual switch and see if it is **closed**, then the instruction is **TRUE** -We look at the actual switch and see if it is **open**, then the instruction is **FALSE**	

Instruction symbol	Name of instruction
	Examine if open
-We look at the actual switch and see if it is **open**, then the instruction is **TRUE** -We look at the actual switch and see if it is **closed**, then the instruction is **FALSE**	

Instruction symbol	Name of instruction
	Output energize
-If there is **TRUE** path from the beginning of the rung (left side) to the output energize instruction, them the output instruction is **TRUE** otherwise the instruction would be **FALSE**.	

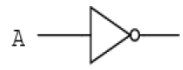
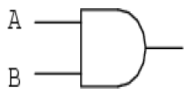
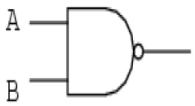
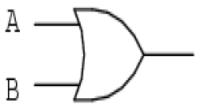
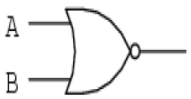
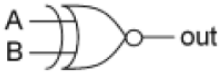
Instruction symbol	Name of instruction
	Latch
	Unlatch
-The latch and unlatch usually both are present together when they are used in a PLC program. If a Latch is ON (TRUE), the only way to make it OFF (FALSE) is by making Unlatch ON.	

The following are 3 examples explain the status **(TRUE/FALSE)** of the instruction in relation to the actual status of the switch

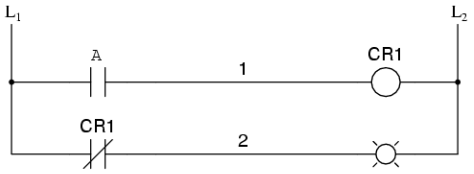
sr	Actual status of the switch Open/Closed	The status of the PLC instruction TRUE/FALSE
1		
2		
3		

2.02 PLC Logic Instructions

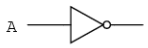
In PLC we can include the logic gates (AND, NAND, OR, NOR,…..) in the program. In this section we will see how to convert logic circuit to PLC program.

If we wish to invert the *output* of any switch-generated logic function, we must use a relay with a normally-closed contact. For instance, if we want to energize a load based on the inverse, or NOT, of a normally-open contact, we could do this:



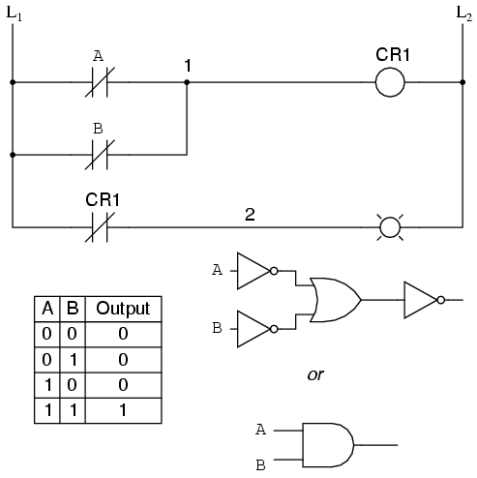
A	CR1	Output
0	0	1
1	1	0



We will call the relay, "control relay 1," or CR1. When the coil of CR1 (on the first rung) is energized, the contact on the second rung *opens*, thus de-energizing the lamp. The normally-closed contact actuated by relay coil CR1 provides a logical inverter function to drive the lamp opposite that of the switch-A actuation status.

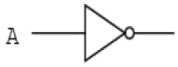
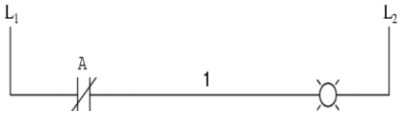
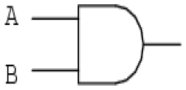
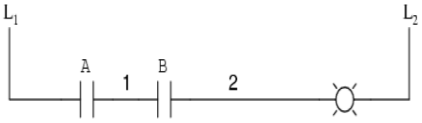
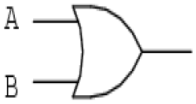
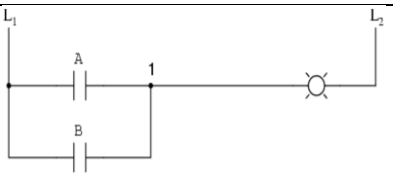
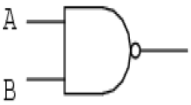
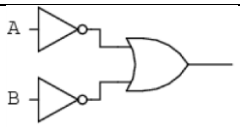
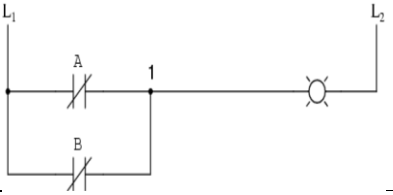
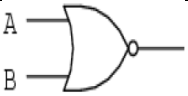
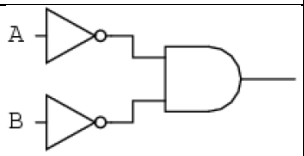
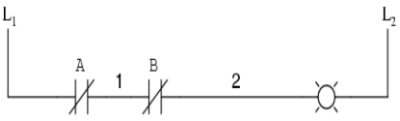
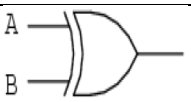
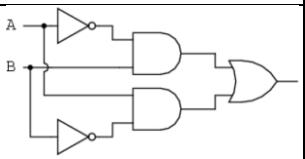
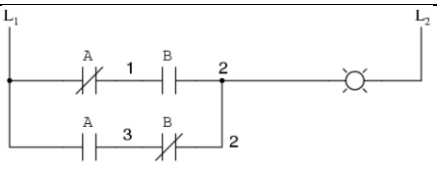
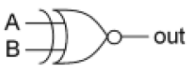
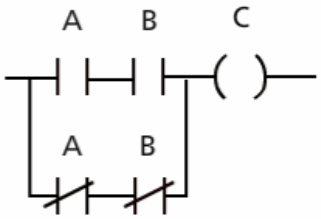
Applying this inversion strategy to one of our inverted-input functions created earlier, such as the OR-to-NAND, we can invert the output with a relay to create a non-inverted function:

From the switches to the coil of CR1, the logical function is that of a NAND gate. CR1 normally-closed contact provides one final inversion to turn the NAND function into an AND function.



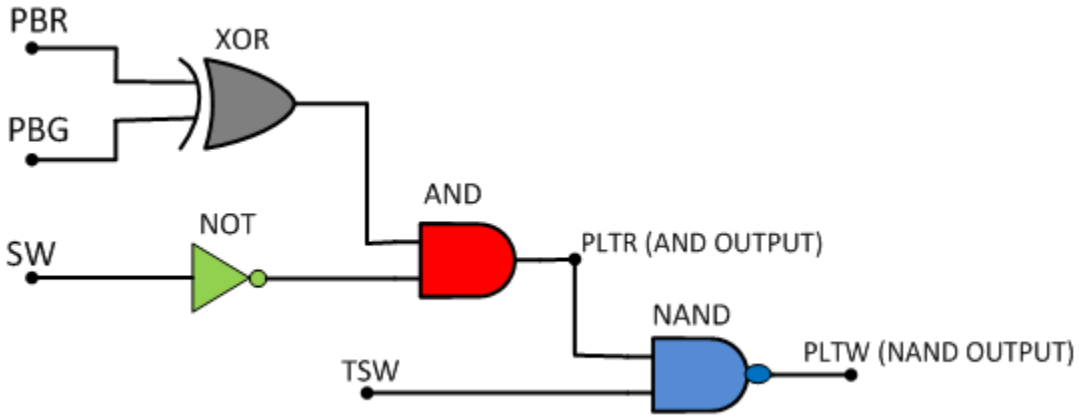
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Below is a table showing the conversion of Logic Gates to Ladder Diagram.

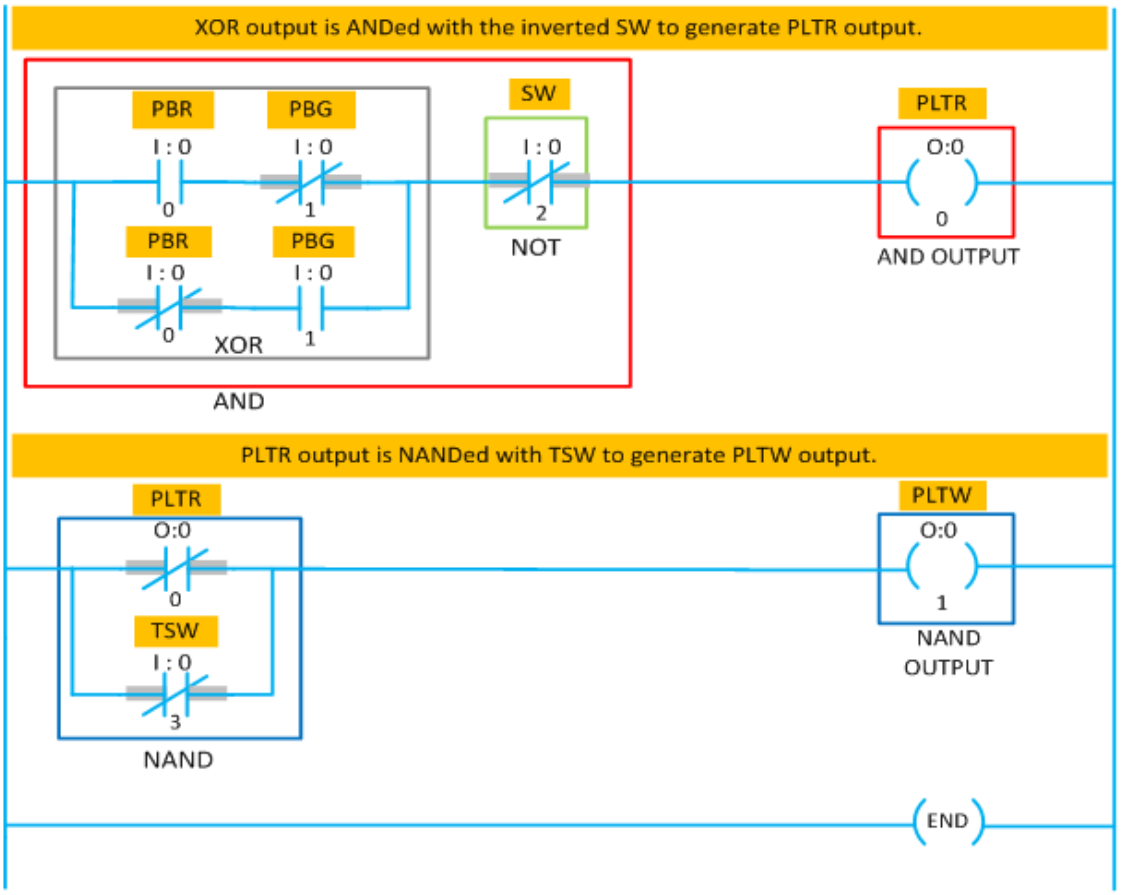
	Symbol	Logic	Truth Table	Ladder Diagram
Inverter (NOT)		Out = $\overline{A}$	A | Output 0 | 1 1 | 0	
AND		Out = A . B	A B | Output 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1	
OR		Out = A + B	A B | Output 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1	
NAND		 Out = $\overline{A \cdot B} = \overline{A} + \overline{B}$	A B | Output 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0	
NOR		 Out = $\overline{A + B} = \overline{A} \cdot \overline{B}$	A B | Output 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0	
XOR		 Out = $A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$	A B | Output 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0	
XNOR		If we place an inverter at the end of the XOR we can obtain an XNOR Out = $\overline{A \oplus B} = A \cdot B + \overline{A} \cdot \overline{B}$	A B | Output 0 0 | 1 1 0 | 0 0 1 | 0 1 1 | 1	

Switch opened → Input = 0
Switch Closed → Input = 1

Example 2-01: Convert the logic circuit to a PLC program.



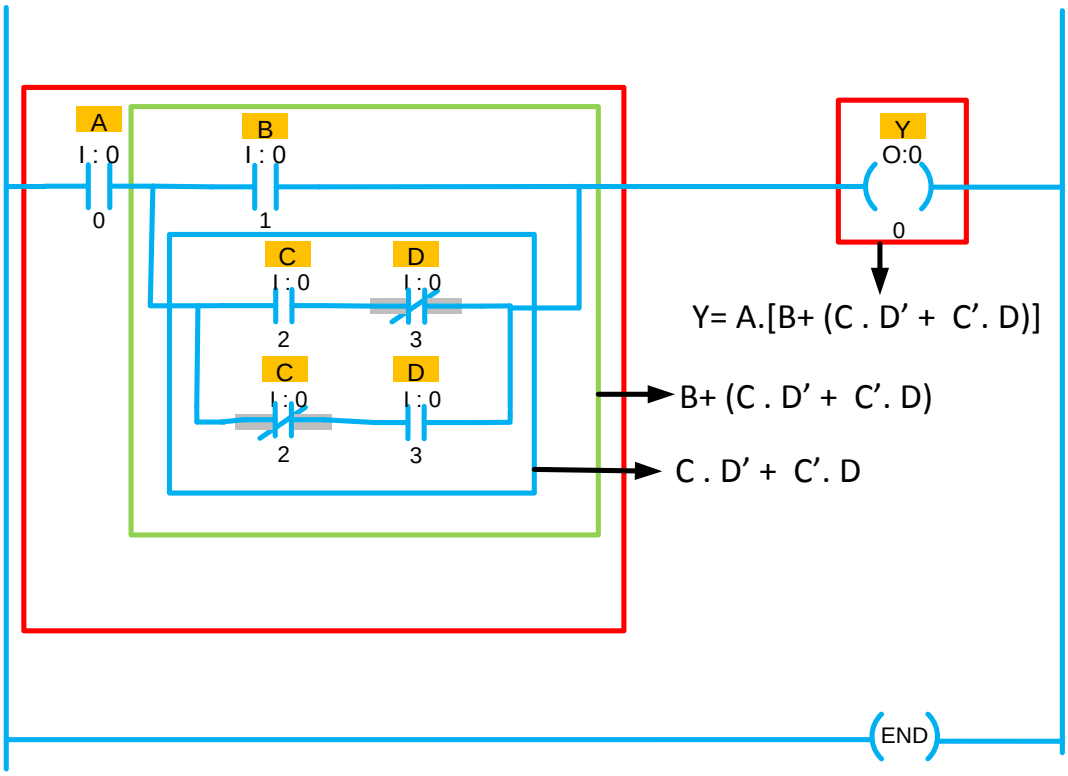
Solution: Using the conversion table on previous page, the PLC program equivalent to the logic circuit is shown below.

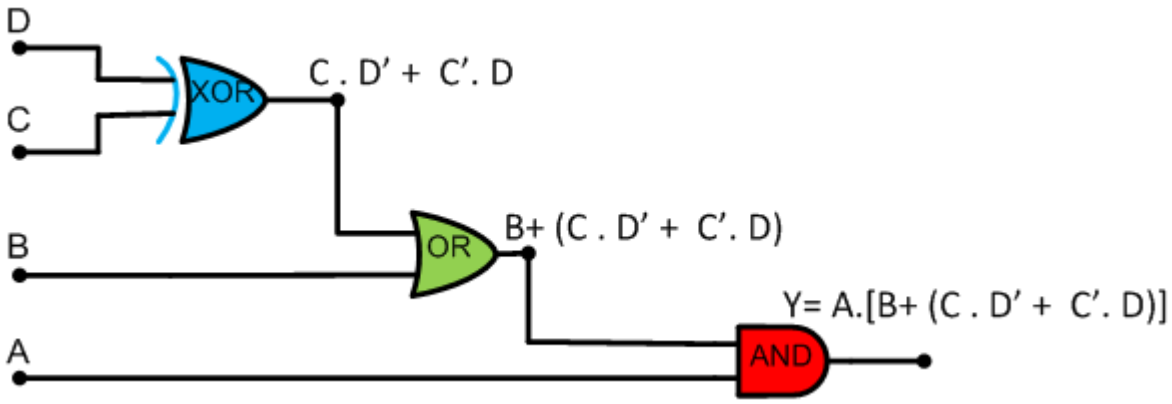


Example 2-02: Convert the Boolean expression to PLC program and to a logic circuit.

$$Y = A \cdot [B + (C \cdot D' + C' \cdot D)]$$

Solution: Using the conversion table, the PLC program equivalent and the logic circuit are shown below.





Test your knowledge:

Show your work to the instructor after completing each exercise.

Exercise 2-1: Write a documented program, which will express the following Boolean equation as a logic ladder rung:

$$Y = [(\overline{A}+\overline{B})C]+DE$$

Use the PLC I/O simulator screen and the following addresses to simulate the program:

A_ I:1/0
B_ I:1/1
C_ I:1/2
D_ I:1/3
E_ I:1/4
Y_O:2/0

Exercise 2-2: Write a documented program, which will express the following Boolean equation as a logic ladder rung:

$$Y = (\overline{A}B\overline{C})+(D\overline{E}F)$$

Use the PLC I/O simulator screen and the following addresses to simulate the program:

A_ I:1/0
B_ I:1/1
C_ I:1/2
D_ I:1/3
E_ I:1/4
F_ I:1/5
Y_O:2/0

Exercise 2-3: Write a documented program, which will express the following Boolean equation as a logic ladder rung:

$$Y = \overline{A}B+A\overline{B}$$

Use the PLC I/O simulator screen and the following addresses to simulate the program:

A_ I:1/0
B_ I:1/1
Y_O:2/0

This Page is Left Blank

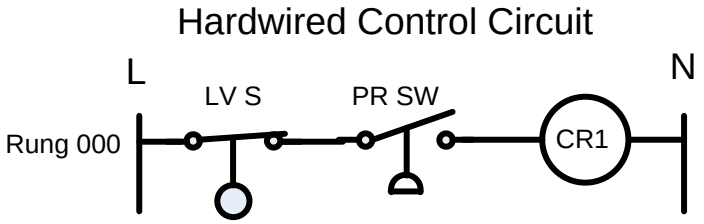
Chapter 3: PLC Simulator

Objectives: Upon completion of this chapter, the student should be able to:

1- Convert control relay to PLC control program.
2- Use LogixPro PLC Simulator.
3- Write a PLC Program.
4- Run and Test a PLC Program.

3.01 How to Convert Control Relay to PLC Control Program

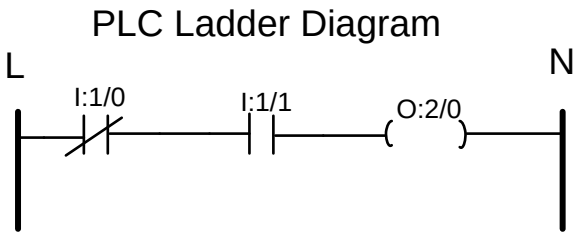
We should be able to convert any hardwired control circuit to PLC control program. The PLC programing language which we will use in this book is Ladder Logic Diagram. We will look at an example of how we convert hard wired control circuit to a PLC program. The following is a hardwired circuit.



To convert this hardwired control circuit to PLC control program, the **general rule** says that you must replace all switches with input contact instructions and the control relay coil with outputs instruction. These instructions are shown in the following table.

sr	Instruction symbol	Name of instruction	Name of PLC instruction
1	⊣⊢	Open contact instruction	XIC Examine if closed
2	⊣/⊢	Closed contact instruction	XIO Examine if open
3	⊣( )⊢	Output instruction	OTE Output energized

If we follow the rule we will end up with the following PLC ladder logic diagram.



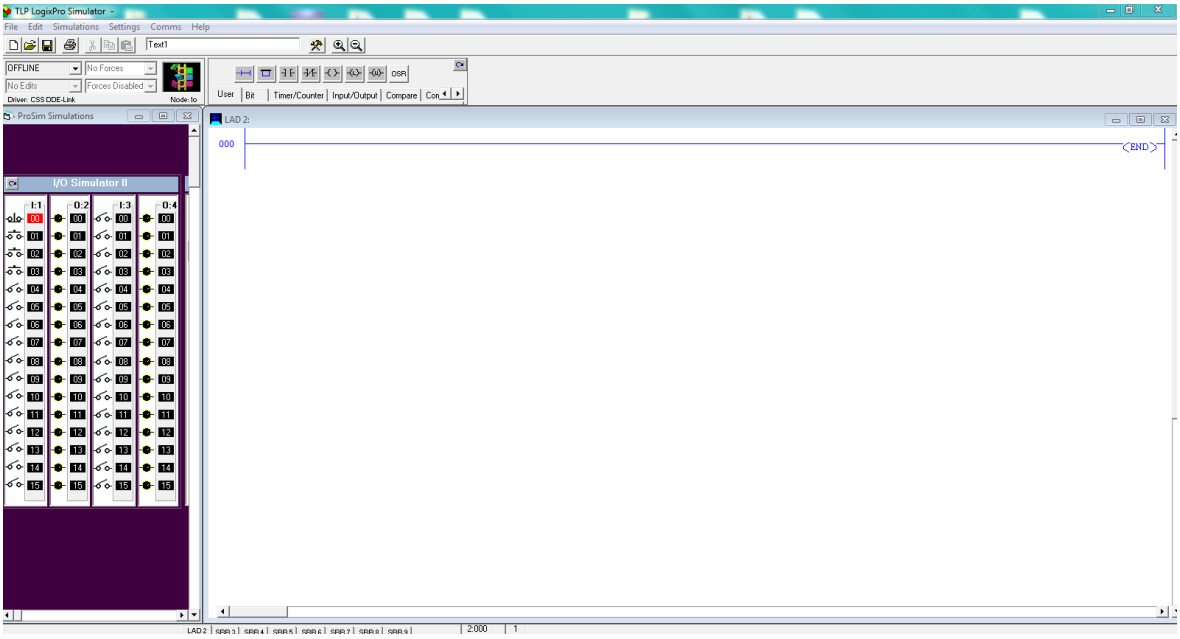
You have just written your first PLC program. The above diagram is an actual PLC control program. In the next section you will learn how to enter and run a PLC program.

3.02 Introduction to LogixPro PLC Simulator

LogixPro simulator is a software that replace the actual PLC and allows you to practice and develop your RSLogix programming skills.

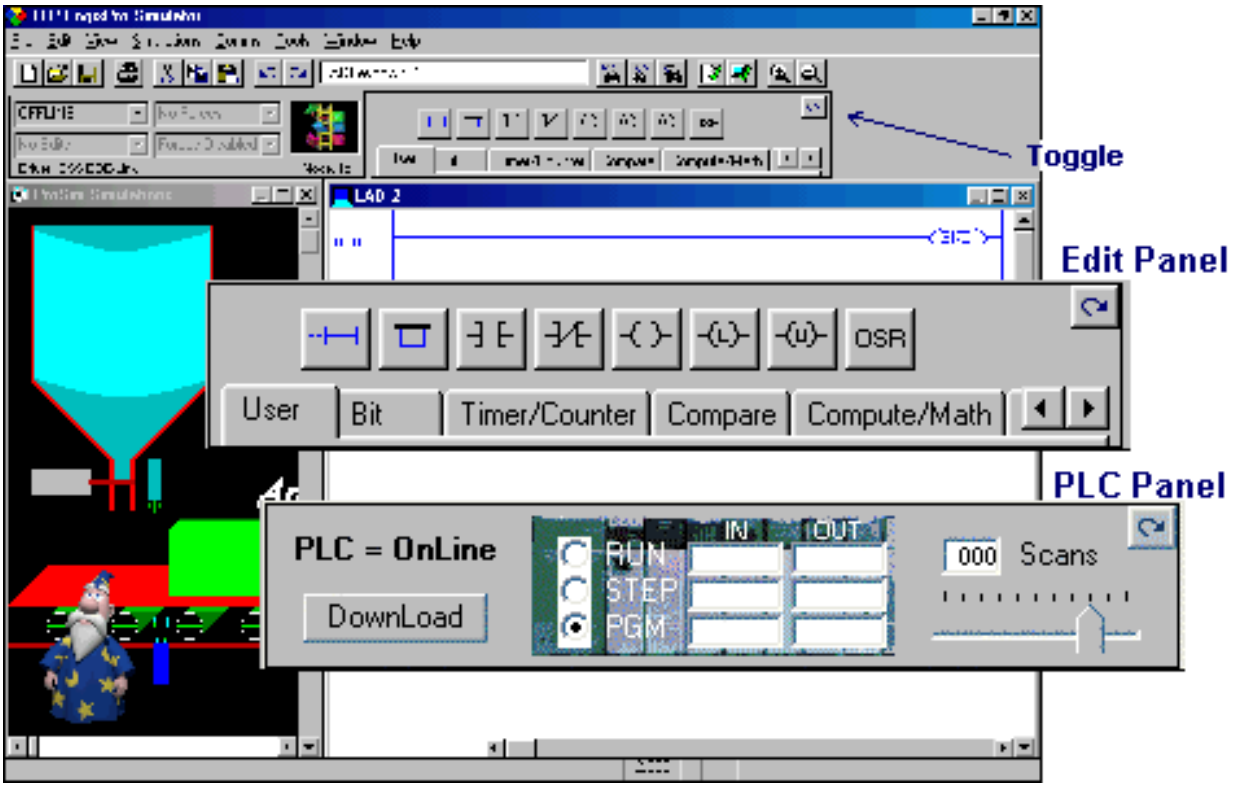
3.02.01 The LogixPro Screen

When you start the LogixPro simulator you will see the following screen.



The most commonly used elements of LogixPro are displayed below. The **Edit Panel** provides easy access to all the RSLogix instructions and they may be simply dragged and dropped into your program.

Once your program is ready for testing, clicking on the "Toggle Button" of the Edit Panel will bring the **PLC Panel** into view. From the PLC Panel you can download your program to the "PLC" and then place it into the "RUN" mode. This will initiate the scanning of your program and the I/O of your chosen simulation.

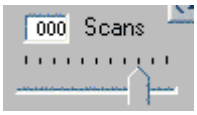


3.02.02 Editing Your Program

Click on an Instruction or Rung with the left mouse button and keep it held down and you will be able to drag it wherever you want to place it. Let go of it on any of the tiny locating boxes that you will see, and the Instruction or Rung will grip to it's new home.

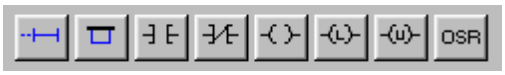
3.02.03 Scanning Time / Speed of Program

If you take a look at the PLC Panel you'll notice an adjustable Speed Control (Shown below). This is provided with LogixPro so that you may adjust the speed of the simulations to suit your particular computer.



3.02.04 How to Write and Run a PLC Program

This exercise is designed to familiarize you with the operation of LogixPro and to step you through the process of creating, editing and testing simple PLC programs utilizing the Relay Logic Instructions supported by RSLogix.



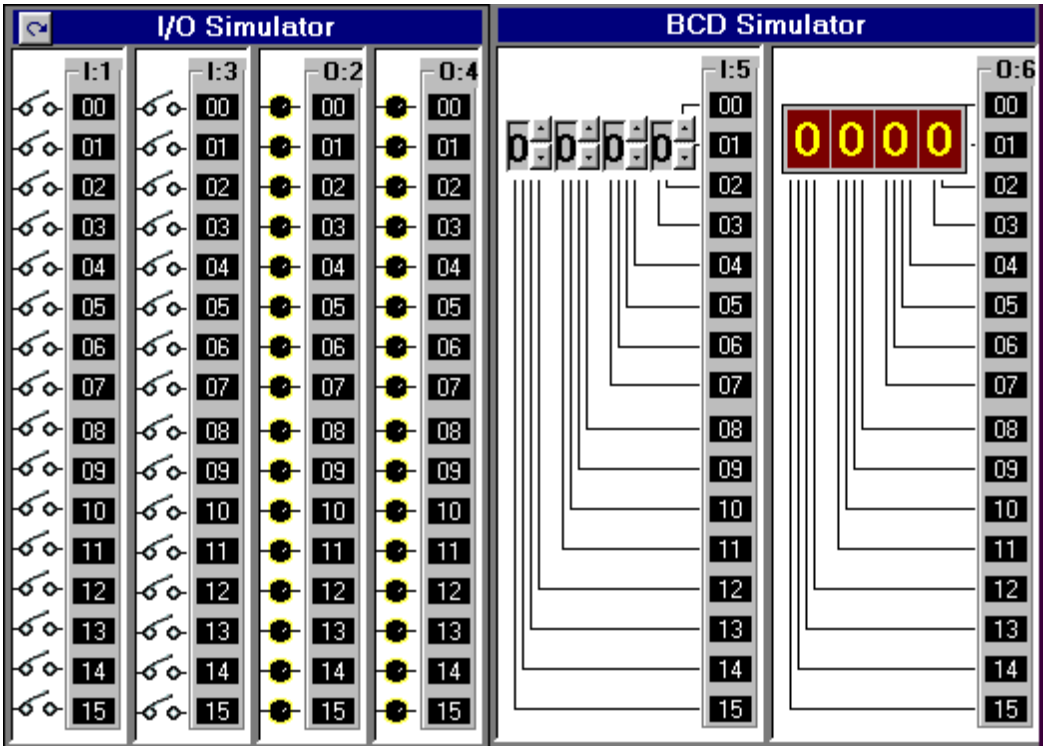
From the Simulations Menu at the top of the screen, Select the I/O Simulation and ensure that the User Instruction Bar shown above is visible.

The program editing window should contain a single rung. This is the End of Program rung and is always the last rung in any program. If this is the only rung visible then your program is currently empty.

If your program is not empty, then click on the File menu entry at the top of the screen and select "New" from the drop-down list. A dialog box will appear asking for you to select a Processor Type. Just click on "OK" to accept the default TLP LogixPro selection.

3.02.05 The I/O Simulator

From the simulation dropdown menu select I/O simulator. The simulator screen shown below, should now be in view. For this exercise we will be using the I/O simulator section, which consists of 32 switches and lights. Two groups of 16 toggle switches are shown connected to 2 Input cards of our simulated PLC. Likewise two groups of 16 Lights are connected to two output cards of our PLC. The two input cards are addressed as "I:1" and "I:3" while the output cards are addressed "O:2" and "O:4".

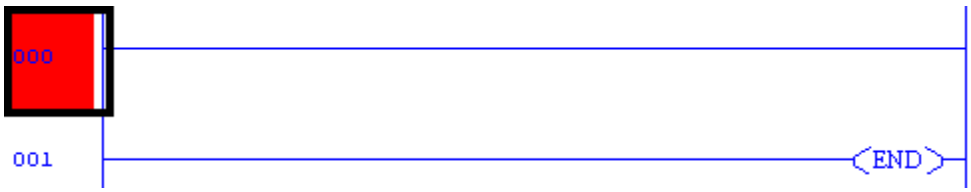


Use your mouse to click on the various switches and note the change in the status color of the terminal that the switch is connected to. Move your mouse slowly over a switch, and the mouse cursor should change to a hand symbol, indicating that the state of switch can be altered by clicking at this location. When you pass the mouse over a switch, a "tool-tip" text box also appears and informs you to "Right Click to Toggle Switch Type". Click your right mouse button on a switch, and note how the switch type may be readily changed.

3.02.06 RSLogix Program Creation

You will now enter the following single run program which consists of a single Input instruction (XIC - Examine If Closed) and a single Output instruction (OTE - Output Energize). There's more than one way to accomplish this task, but for now we will outline what is considered to be the most commonly used approach.

 First click on the "New Rung" button in the User Instruction Bar. It's the first button on the very left end of the Bar. If you hold the mousepointer over any of these buttons for a second or two, you should see a short "ToolTip" which describes the function or name of the instruction that the button represents.

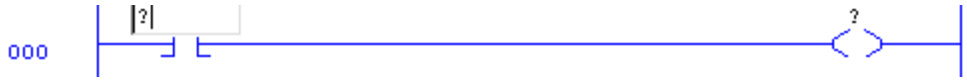


You should now see a new Rung added to your program as shown above, and the Rung number at the left side of the new rung should be highlighted. Note that the new Rung was inserted above the existing (END) End Of Program Rung. Alternatively you could have dragged (left mouse button held down) the Rung button into the program window and dropped it onto one of the locating boxes that would have appeared.

 Now click on the XIC instruction with your left mouse button (Left Click) and it will be added to the right of your highlighted selection. Note that the new XIC instruction is now selected (highlighted). Once again, you could have alternatively dragged and dropped the instruction into the program window.

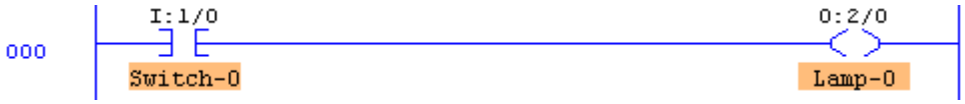
If you accidentally add an instruction which you wish to remove, just Left Click on the instruction to select it, and then press the "Del" key on your keyboard. Alternatively, you may right click on the instruction and then select "Cut" from the drop-down menu that appears.

Left Click on the OTE output instruction and it will be added to the right of your current selection.



Double Click (2 quick left mousebutton clicks) on the XIC instruction and a textbox should appear which will allow you to enter the address (I:1/0) of the switch we wish to monitor. Use the Backspace key to get rid of the "?" currently in the textbox. Once you type in the address, click anywhere else on the instruction (other than the textbox) and the box should close.

Right Click on the XIC instruction and select "Edit Symbol" from the drop-down menu that appears. Another textbox will appear where you can type in a name (Switch-0) to associate with this address. As before, a click anywhere else will close the box.



Enter the address and symbol for the OTE instruction and your first RSLogix program will now be complete. Before continuing however, Double check that the addresses of your instructions are correct.

3.02.07 Testing your Program

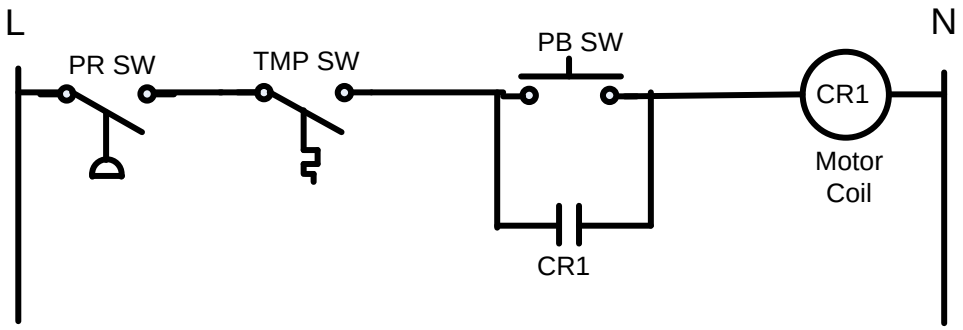
It's now time to "Download" your program to the PLC. First click on the "Toggle" button at the top right corner of the Edit Panel which will bring the PLC Panel into view.

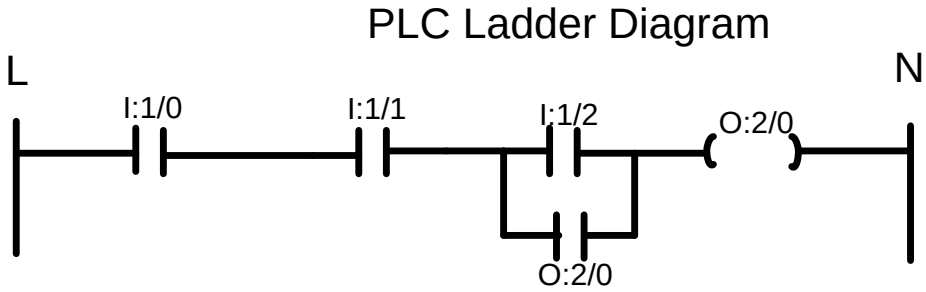
Click on the "DownLoad" button to initiate the downloading of your program to the PLC. Once complete, click inside the "RUN" option selection circle to start the PLC scanning.

Enlarge the Simulation window so that you can see both the Switches and Lamps, by dragging the bar that separates the Simulation and Program windows to the right with your mouse. Now click on Switch I:1/00 in the simulator and if all is well, Lamp O:2/00 should illuminate.

Toggle the Switch On and Off a number of times and note the change in value indicated in the PLC Panel's status boxes which are being updated constantly as the PLC Scans. Try placing the PLC back into the "PGM" mode and then toggle the simulator's Switch a few times and note the result. Place the PLC back into the "Run" mode and the Scan should resume.

The figure below is an example of converting hardwired control to a PLC control program. Enter the program shown below and test it.





Show your work to the instructor.

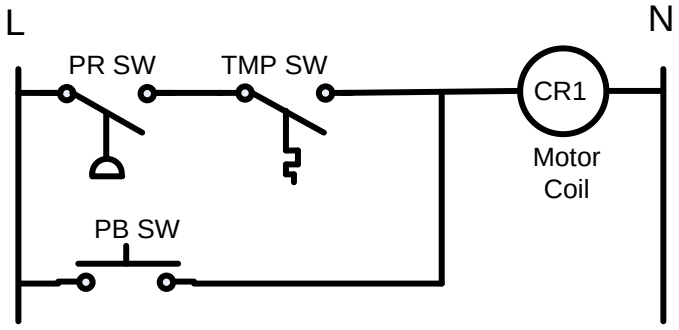
Test your knowledge:

Show your work to the instructor after completing each exercise.

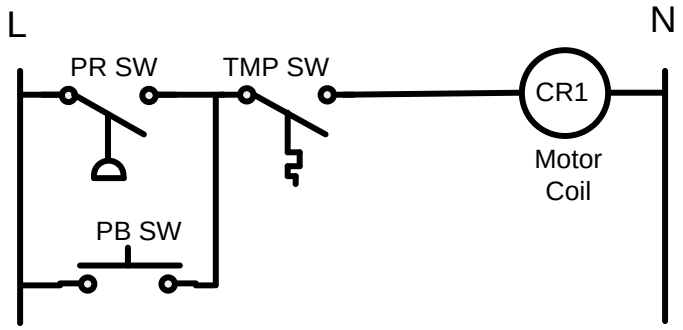
Exercise 3-1(a):

Write a documented program for the relay schematic shown below. Use the I/O simulator screen and the following addresses to simulate the program:

Pressure switch_I:1/0
Temperature switch_I:1/1
Manual pushbutton_I:1/2
Motor starter coil_O:2/0



Exercise 3-1(b): Modify the original program to operate according to the relay schematic shown below.



Exercise 3-1(c): Modify the original program to operate so that the manual pushbutton, pressure switch, and temperature switch all must be closed to energize the motor starter.

Exercise 3-1(d): Modify the original program to operate so that closing the manual pushbutton, pressure switch, or the temperature switch will energize the motor starter.

Chapter-4: Allen Bradley (AB) SLC 500 PLC

Objectives: Upon completion of this chapter, the student should be able to:

1- Use Rockwell RSLogix software.
2- Connect power supply, inputs and outputs to a PLC.
3- Write a PLC control program with Rockwell RSLogix software.
4- Download, Run and test a PLC control program.

4.01 Introduction to AB SLC 500 PLC

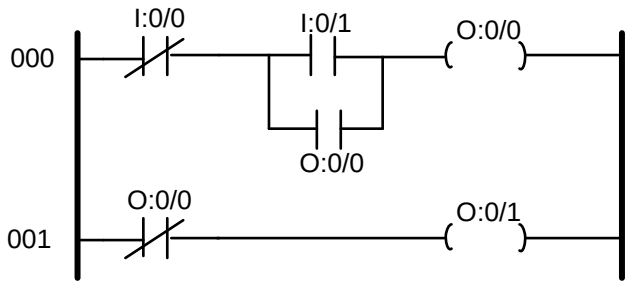
In this chapter we will learn how to write and run program using the actual PLC. Fortunately using AB SLC 500 PLC is very much similar to using LogixPro simulator. To write program for the AB SLC 500 PLC we must use the Rockwell RSLogix software.

4.02 Using Rockwell RSLogix:

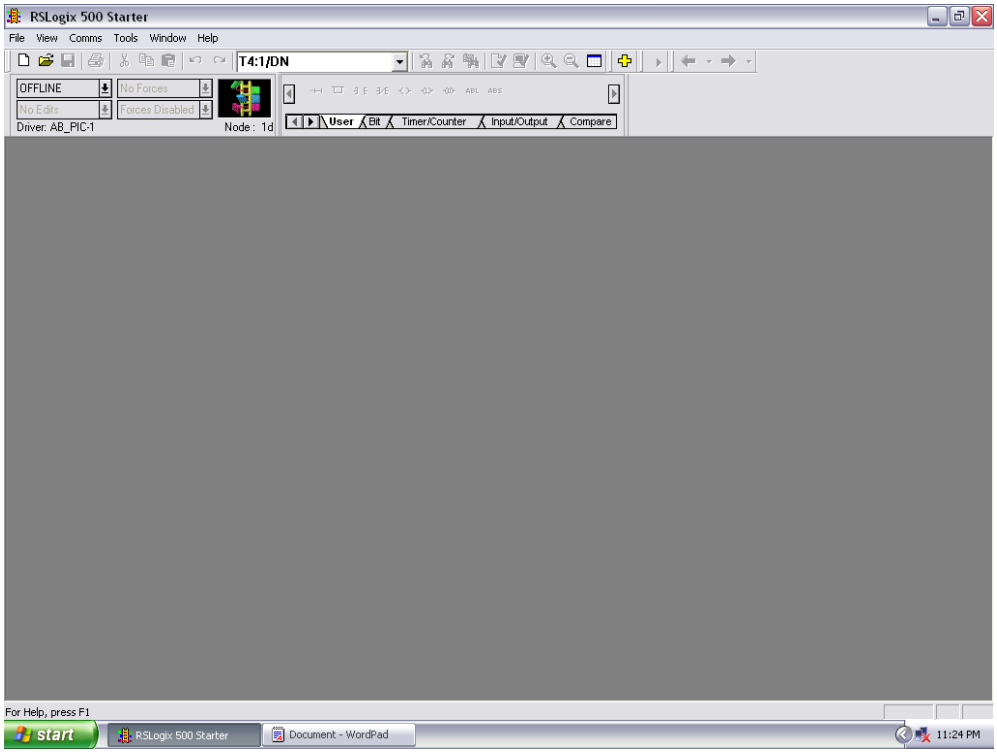
The following steps shows how to use the Rockwell RSLogix software to write and run a PLC program.

Using the steps described below, enter the program shown below and test it.

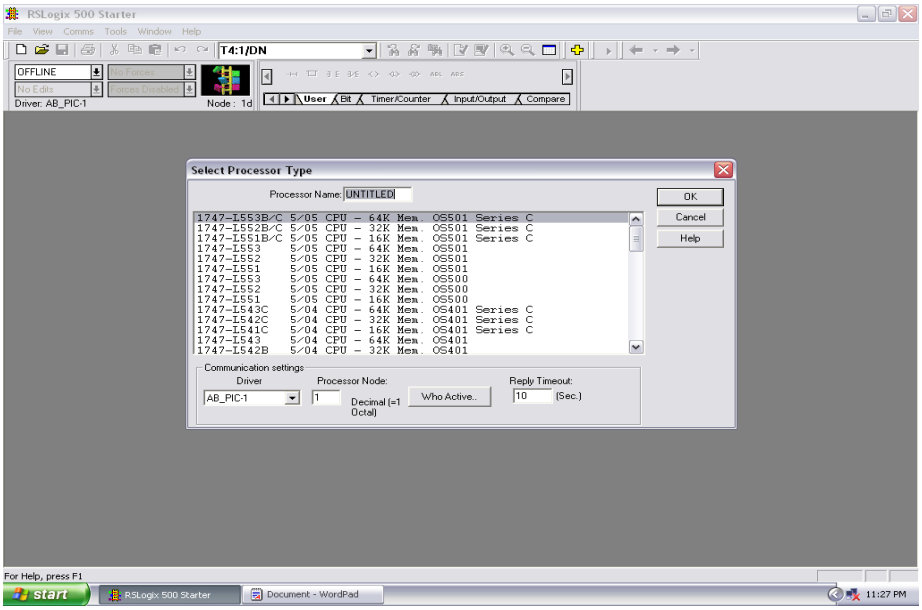
PLC Ladder Diagram



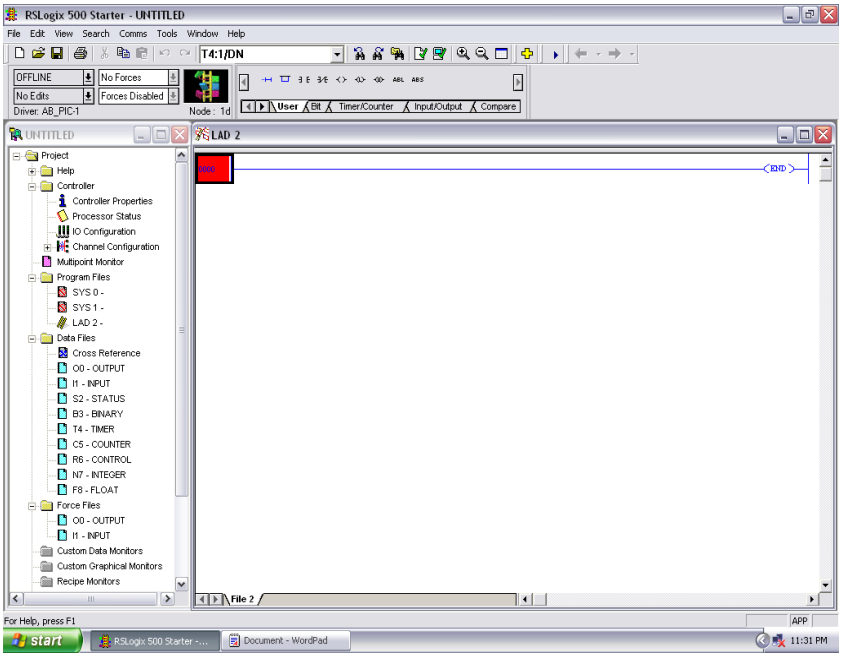
1- Start the **RSLogix 500**, you will get the following screen



2- Go to the file menu and select **NEW**, you should see the following screen with a dialog box. You need to select the processor type.

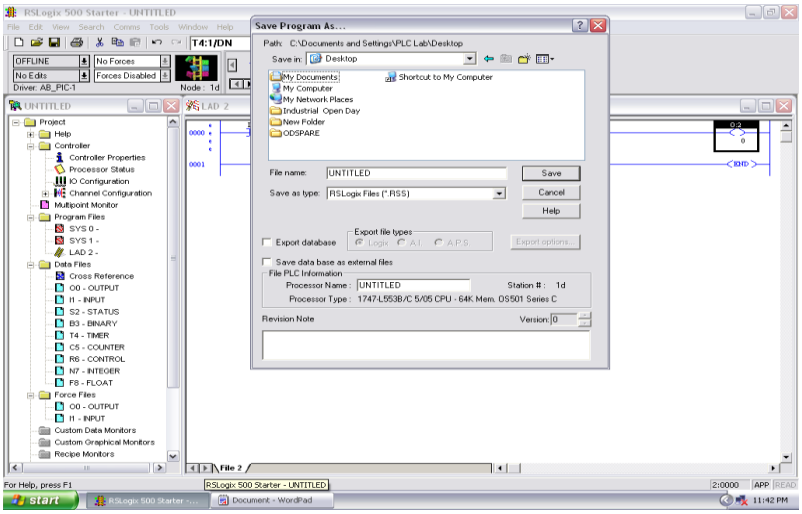


3- Select: 1747-L20A 12-115VAC In, 8-RLY out, and press **OK**. You should see the following screen.



4- Now you can write your program in the same way you did with the **LogixPro** simulator.

5- Save your program. From the file menu click save or save as you should see the following screen with a dialog box to specify the **folder** where to save the **file** and the file **name**. Do so and press **save.**

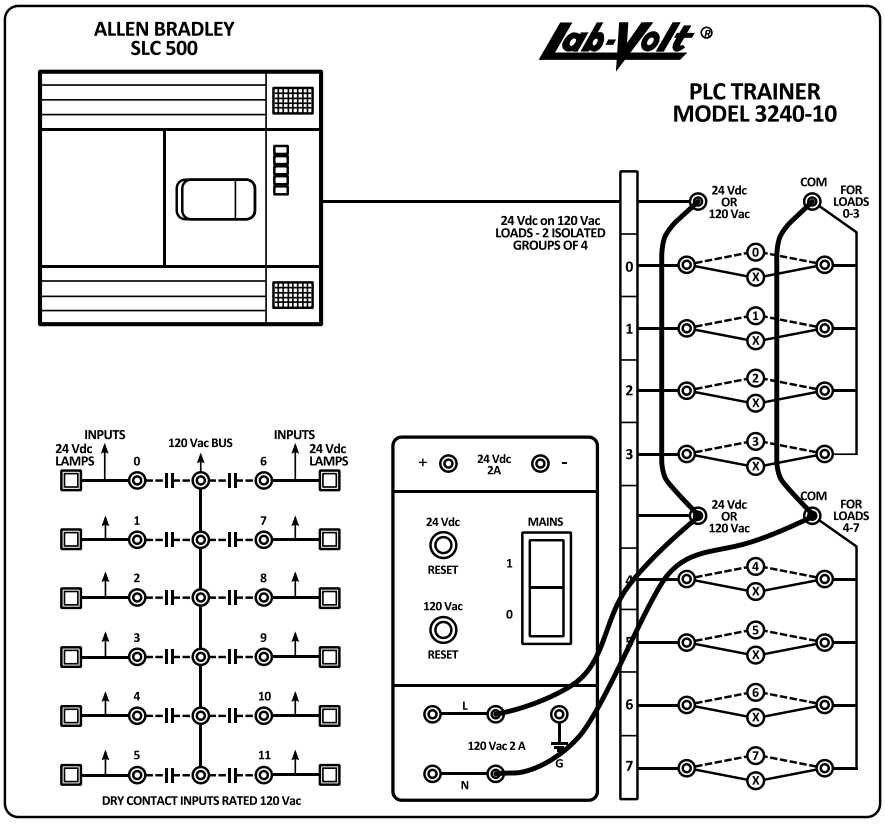


6- Now you should verify that your program does not have any errors.

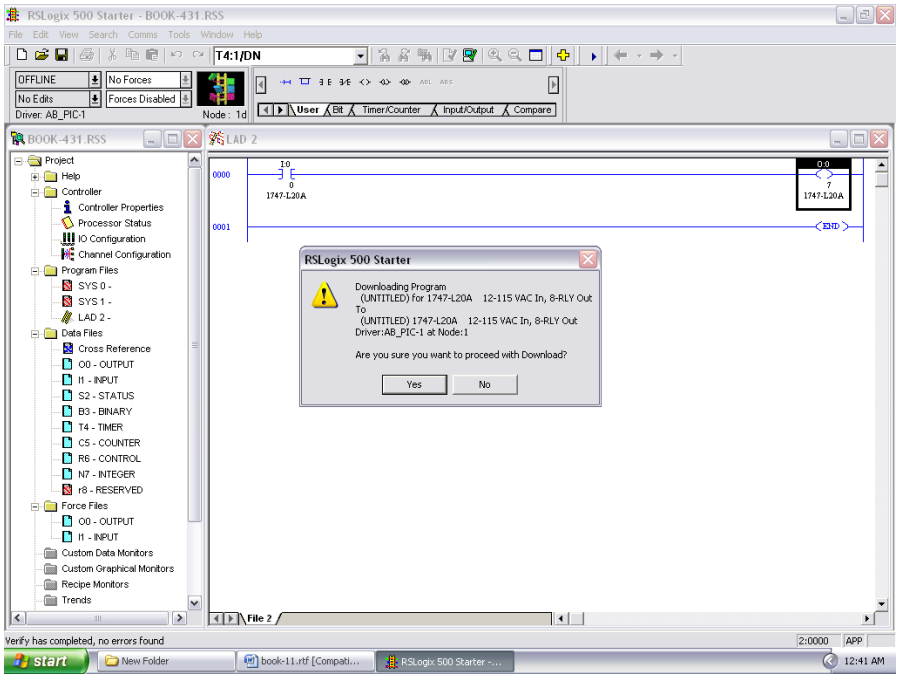
7- Click the **verify project** icon. If there is any error it will show a massage at the bottom of the screen.

Do not turn power ON until the instructor tell you!!

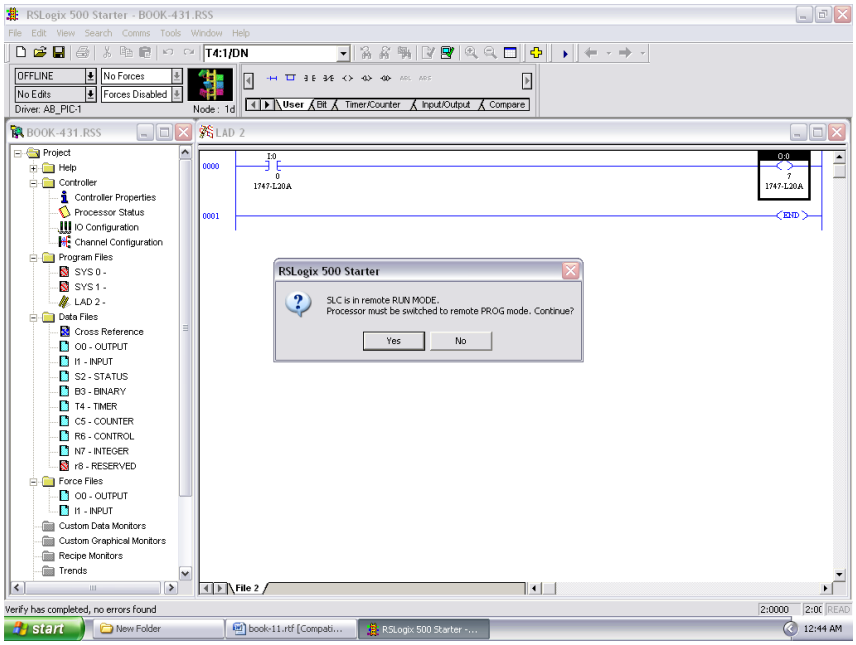
Before downloading the program to the PLC, we must connect the power supply to the output of the Labvolt PLC trainer as shown in the figure below and turn the power switch ON.



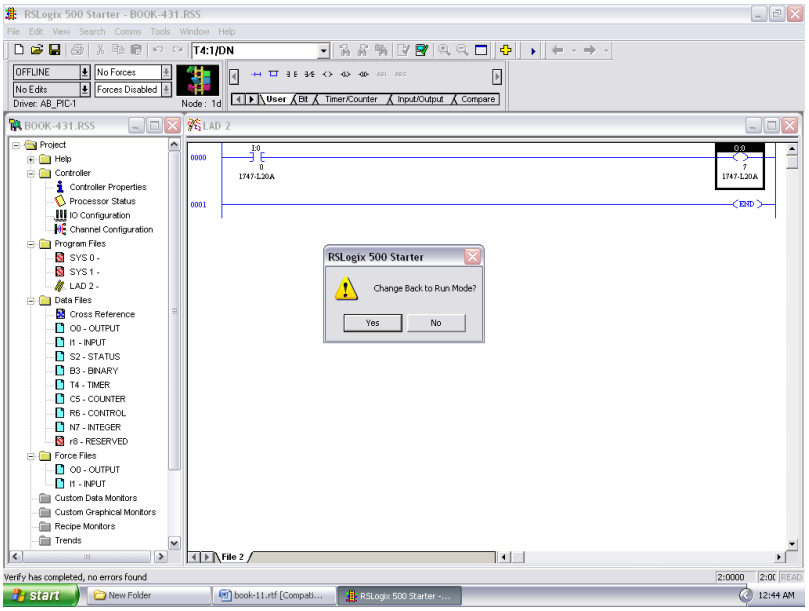
8- Download the program from the **OFFLINE** pull-down menu. Select **Download** and you should see (are you sure you want to proceed to download) the following dialog box, click **YES**



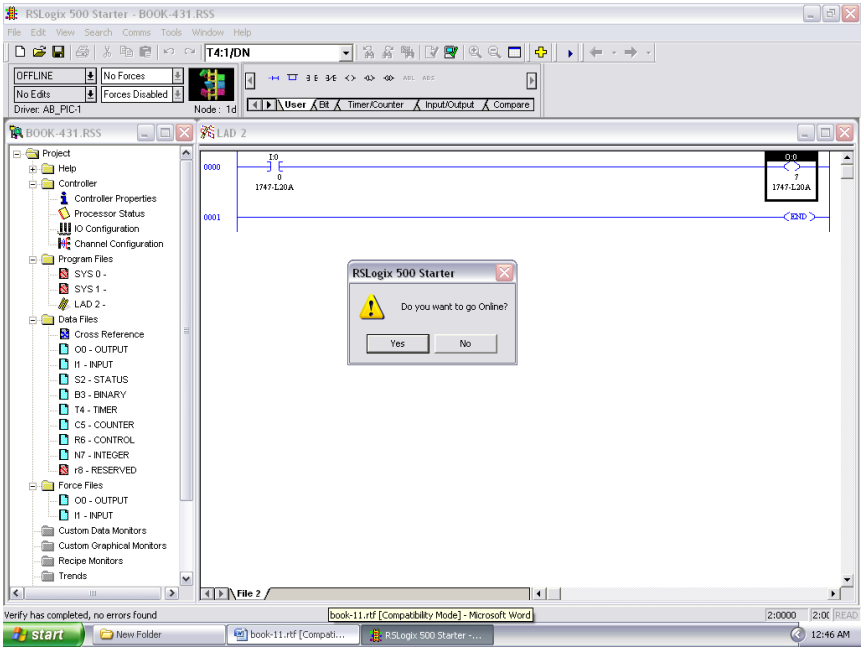
9- You will see (processor must be switched to remote PROG mode. Continue?) the following dialog box, click **YES**



10- You will see (Change to RUN mode) the following dialog box, click **YES**



11- You will see (Go ONLINE) the following dialog box, click **YES**



12- Test the operation of your program.

Show your work to the instructor after completing the above exercise.

Test your knowledge:

Show your work to the instructor after completing each exercise.

For each of the following exercises write the ladder diagram, enter the program, wire the circuit, run, and test the program.

Exercise 4-1: Given two single pole switches, write a documented program that will turn a lamp on when both switch A and switch B are closed. Use the I/O simulator screen and the following addresses to simulate the program:

Switch A_I:0/0
Switch B_I:0/1
Lamp_O:0/0

Exercise 4-2:: Given two normally open pushbuttons, write a documented program that will turn a lamp on when either pushbutton A or pushbutton B is closed. Use the I/O simulator screen and the following addresses to simulate the program:

Pushbutton A_I:0/0
Pushbutton B_I:0/1
Lamp_O:0/0

Exercise 4-3: Given four single pole switches (A-B-C-D), write a documented program that will turn on a lamp if switches A and B or C and D are closed. Use the I/O simulator screen and the following addresses to simulate the program:

Switch A_I:0/0
Switch B_I:0/1
Switch C_I:0/2
Switch D_I:0/3
Lamp_O:0/0

This Page is Left Blank

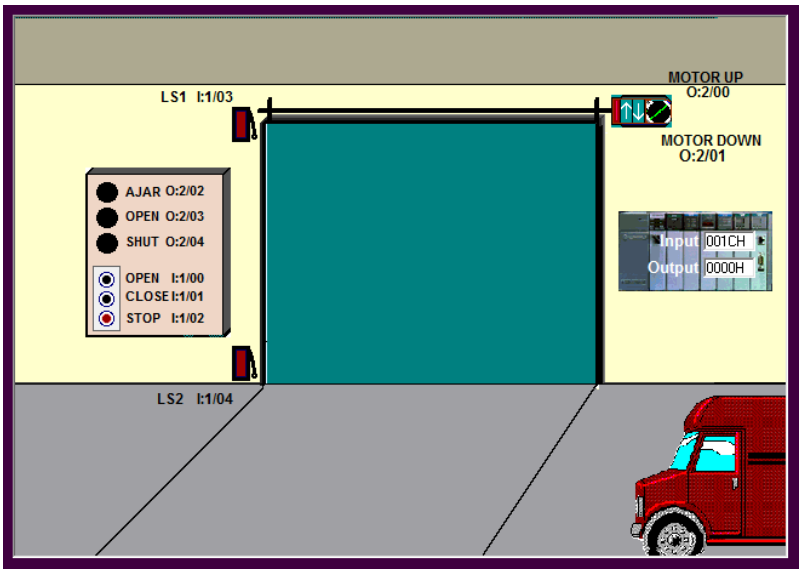
Chapter-5: PLC Industrial Applications

Objectives: Upon completion of this chapter, the student should be able to:

1- Convert logic control statement to PLC control program.
2- Collect inputs and outputs needed for industrial control project.
3- Complete garage door control industrial PLC control project.
4- Complete dual compressor control industrial PLC control project.

5.01 Door Simulation

From the Simulations Menu at the top of the screen, select the Door Simulation. Take the time to familiarize yourself with the components used in the door system, and take particular note of the current state of the limit switches.



In this exercise we want you to apply your knowledge of Relay Logic Instructions to design a program which will control the Door. The Door System includes a Reversible Motor, a pair of Limit Switches and a Control Panel, all connected to your PLC. The program you create will monitor and control this equipment while adhering to the following criteria:

- In this exercise the Open and Close pushbuttons will be used to control the movement of the door. Movement will not be maintained when either switch is released, and therefore the Stop switch is neither required nor used in this exercise. However, all other available Inputs and Outputs are employed in this exercise.

- Pressing the Open Switch will cause the door to move upwards (open) if not already fully open. The opening operation will continue as long as the switch is held down. If the switch is released, or if limit switch LS1 opens, the door movement will halt immediately.

- Pressing the Close Switch will cause the door to move down (close) if not already fully closed. The closing operation will continue as long as the switch is held down. If the switch is released, or if limit switch LS2 closes, the door movement will halt immediately.

- If the Door is already fully opened, Pressing the Open Switch will Not energize the motor.

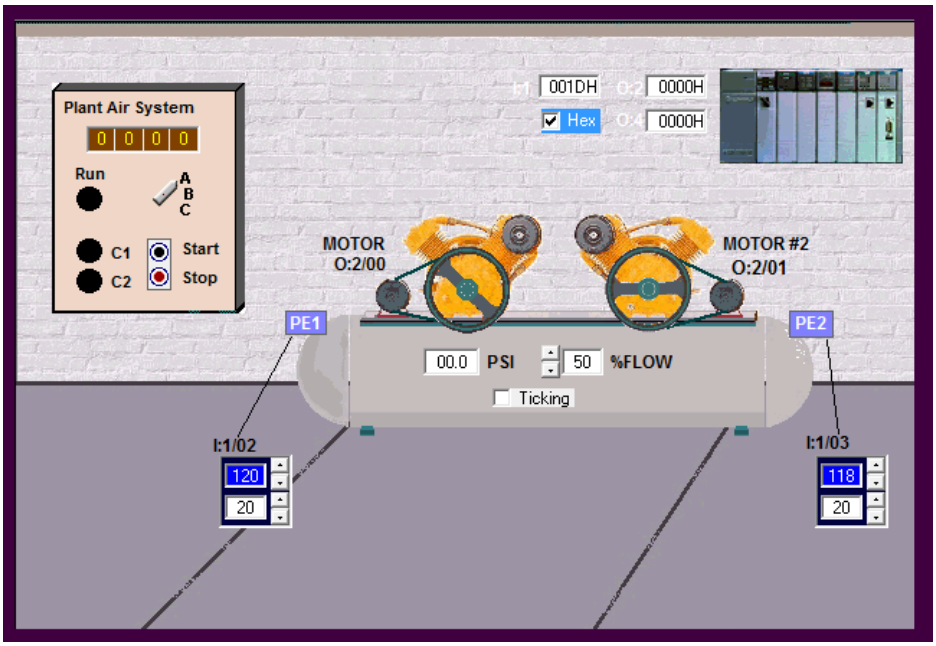
- If the Door is already fully closed, Pressing the Close Switch will Not energize the motor.

- Under no circumstance will both motor windings be energized at the same time.

Do the PLC control program for the door application and test it, then show it to your instructor.

5.02 Dual Compressor

From the Simulations Menu at the top of the screen, select the Dual Compressor Simulation. Take the time to familiarize yourself with the components used in the dual compressor system.



In this exercise we want you to apply your knowledge of Relay Logic Instructions to design a program which will control the air pressure in the tank. The tank system has two compressors operated by two different motors. The program you create will monitor and control this equipment while adhering to the following criteria:

5.02.01 Single Compressor Operation (Left Side Motor)

In this first exercise, the differential pressure switch PE1 (I:1/02) is to be utilized alone to control the operation of motor (O:2/00) and maintain the compressor's storage tank pressure. The pressure range will be dictated by the settings of PE1. Using your mouse, adjust the upper limit (Hi) to (50PSI), and the adjust the lower limit (Lo) setting to (30PSI) of PE1. Adjust the %flow to 56%. Set the selector switch to “A” position.

5.02.02 Single Compressor Operation (Right Side Motor)

In this first exercise, the differential pressure switch PE2 (I:1/03) is to be utilized alone to control the operation of motor (O:2/01) and maintain the compressor's storage tank pressure. The pressure range will be dictated by the settings of PE2. Using your mouse, adjust the upper limit (Hi) to (50PSI), and the adjust the lower limit (Lo) setting to (30PSI) of PE2. Adjust the %flow to 56%. Set the selector switch to “B” position.

Operation:

Allow the user to start and stop the air system using the appropriate panel mounted switches, and ensure that the "Run" lamp is illuminated whenever the system is enabled. Lamp "C1" should be illuminated only when compressor #1 is actually running. Lamp "C2" should be illuminated only when compressor #2 is actually running.

When the start button is pressed, the Compressor(s) should start and begin to build up pressure within the storage tank. Once the pressure reaches upper limit (Hi), the compressor(s) should stop, and remain idle until the pressure in the storage tank drops below (Hi-Lo)

Show your work to the instructor after completing each phase of the exercise.

Test your knowledge:

Show your work to the instructor after completing each exercise.

Exercise 5-1:

Add to the door PLC control program, the following options and test each of the options:

a) The **OPEN** Lamp will be illuminated if the door is in the Fully Open position.
b) The **SHUT** Lamp will be illuminated if the door is in the Fully Closed position.
c) The **AJAR** Lamp will be illuminated if the door is in between the Fully Open and Fully Closed position.
d) Let the door go up and down continuously.

Exercise 5-2:

Add to the compressor PLC control program, the following options and test your program:

Two Compressors Operation (For Large Demand)

In this exercise, both of the differential pressure switches PE1(I:1/02) & PE2 (I:1/03) are to be utilized to control the operation of motors (O:2/00) & (O:2/01) and maintain the compressor's storage tank pressure. The pressure range will be dictated by the settings of PE1 & PE2. Using your mouse, adjust the upper limit (Hi) to (50PSI), and the adjust the lower limit (Lo) setting to (30PSI) of PE1 & PE2. Adjust the %flow to 80%. Set the selector switch to "C" position.

This Page is Left Blank

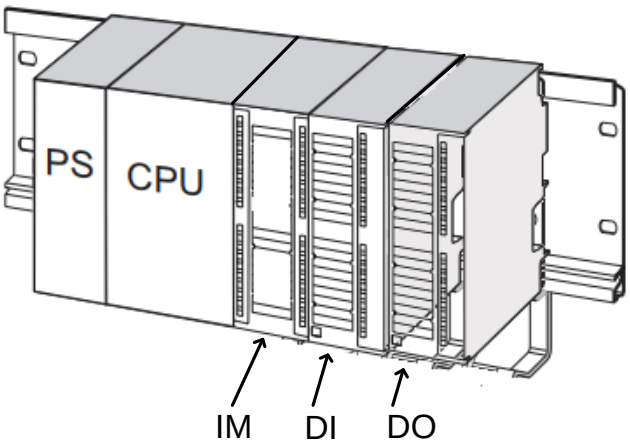
Chapter-6: PLC Hardware Components

Objectives: Upon completion of this chapter, the student should be able to:

1- Understand PLC modular hardware components.
2- Know how PLC and PC are interfaced.
3- Know how inputs and outputs control relays are interfaced to PLC.
4- Determine to connect AC or DC Voltage of PLC Output

6.01 PLC Modular Components

Typically most of PLC systems have the basic functional modular components of Power Supply, Central Processor Unit, Interface Module, Digital Input, and Digital Output. These modules are mounted on a panel next to each other as shown in the figure below.



PS: The power supply unit is needed to provide power to the PLC. Some PLC modules need AC or DC supply at different levels, the power supply convert the voltage to the type and level of voltage needed by each module. As an example the CPU may need 5 volts DC, the DI and DO may need 24 volt AC and other parts of the PLC may need different voltages, the power supply provide all necessary voltages to the PLC system.

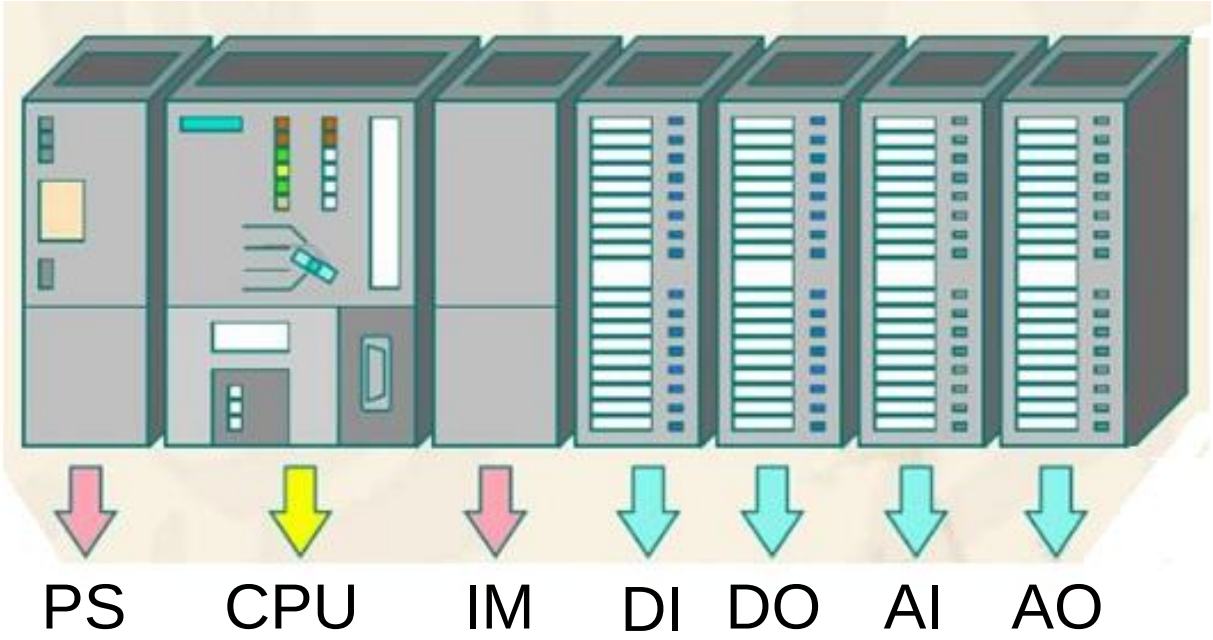
CPU: Central Processing Unit (CPU) is the unit containing the microprocessor. This unit interprets the input signals and carries out the control actions according to the program stored in its memory, communicating the decisions as action signals to the outputs.

IM: Interface Module work like an interpreter between the CPU and the input and output modules. It takes the data from the CPU and transfer it to the input and output modules. Also takes data from the input and output modules and transfer it to the CPU.

DI: Receive signals from input devices such as switches or any other input device.

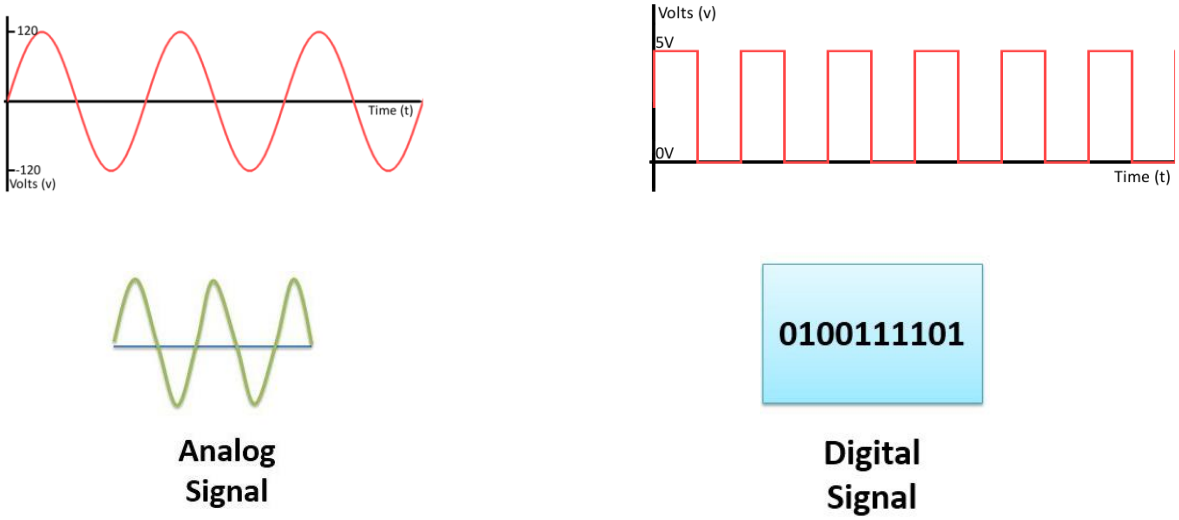
DO: Send signals to output devices such lamp, motor, control relay, or any other output device.

The figure below shows a pictorial PLC system. It is similar to the figure shown previously with the addition of AI and AO modules which represent Analog Input and Analog Output.



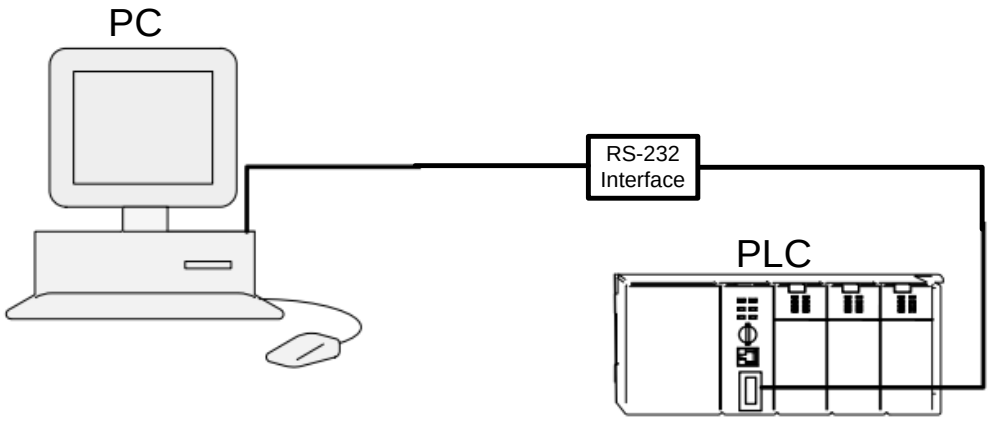
6.02 Digital and Analog Signal:

There are two types of signals that carry information, analog and digital signals. The difference between analog and digital signals is that analog is a continuous electrical signal, whereas digital is a non-continuous (discrete) electrical signal. The figures below show the two types of Signals.



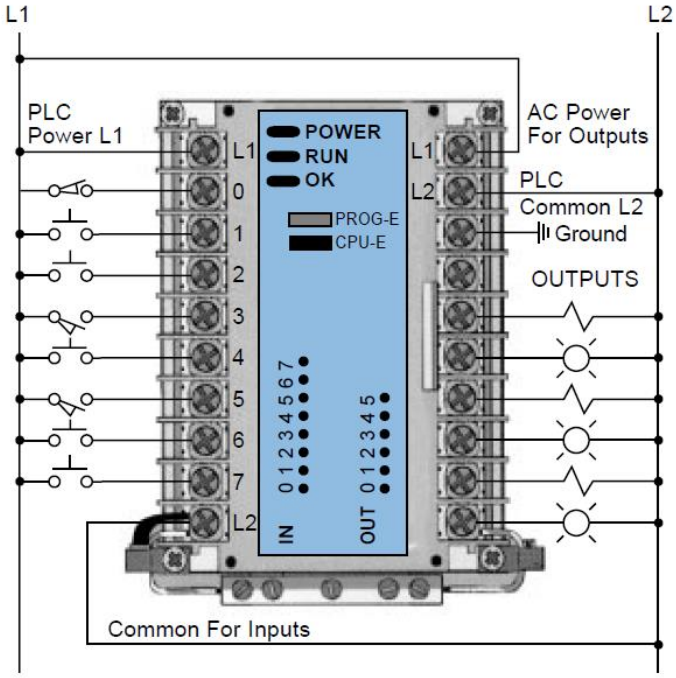
6.03 PLC to PC Interface:

Usually the PLC is connected to PC via interface component as shown in the figure below. The PC is very helpful to program the PLC and to monitor input and output on the screen.



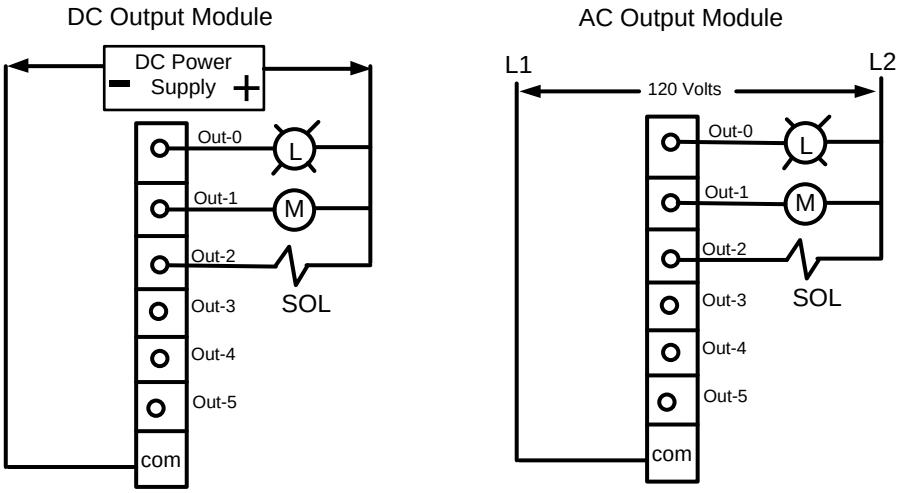
6.04 Input and Output Interface to PLC:

The figure below shows how typically input devices and output devices connected to PLC system.



6.05 Output Voltage Interface to PLC:

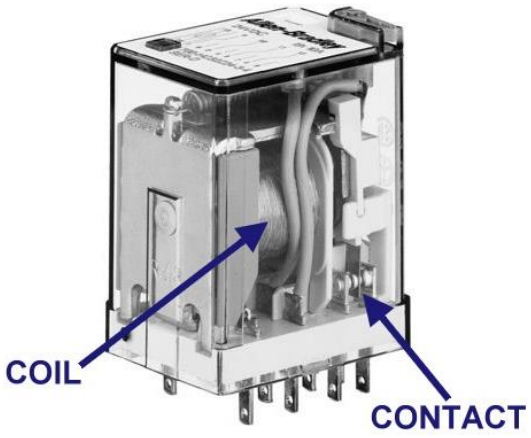
The output of the PLC may be required to be DC or AC voltage as need by the components controlled by the PLC. The figure below shows how the output is connected to the PLC with DC or AC voltage supply.



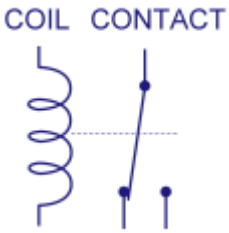
6.06 Control Relay:

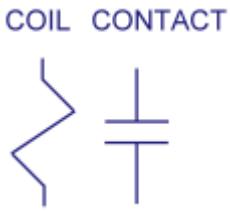
The control logic of a control circuit in a PLC is represented with a program. It is a software with much less hardware components. Now let's take a look how we represent the hardwired contact and control relay in a PLC program.

Typical control relay has coil and contacts (auxiliary contacts). The figure below shows a picture of a control relay.



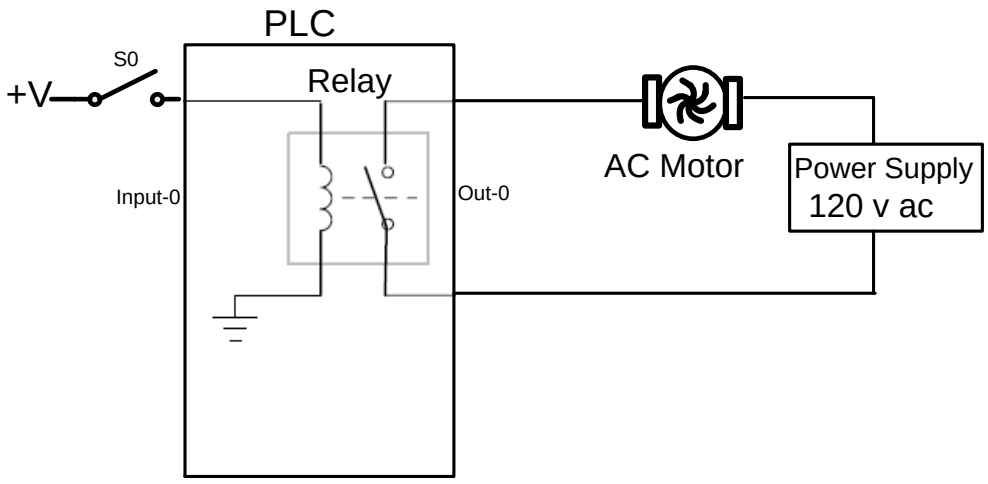
The symbol for control relay can be drawn in different ways. The figure below shows 2 ways to draw a symbol of control relay. Each way is showing the symbol of the coil and contact of a control relay.



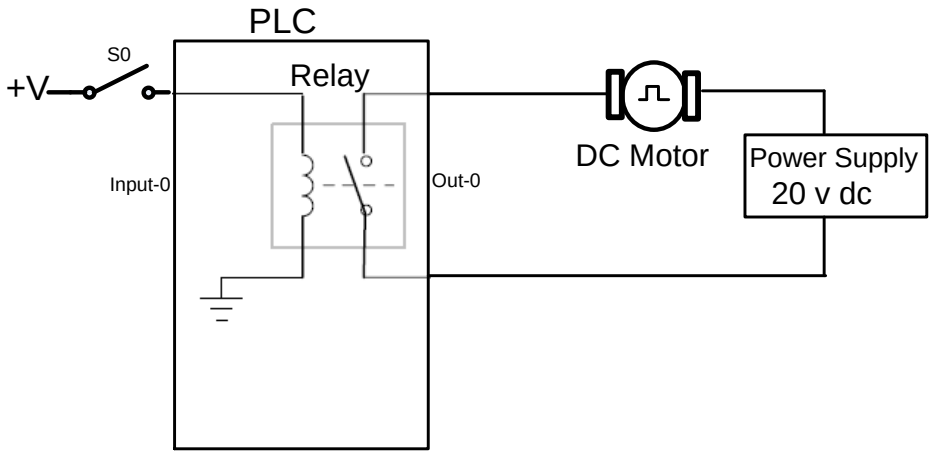


6.07 Control Relay and PLC Output Switch:

PLC outputs are switched ON or OFF using a relay type circuit similar to the relay circuit shown below. Basically the circuits use an opto-coupler to switch external circuitry. This electrically isolates the external electrical circuitry from the internal circuitry. Other circuit components are used to guard against excess or reversed voltage polarity.

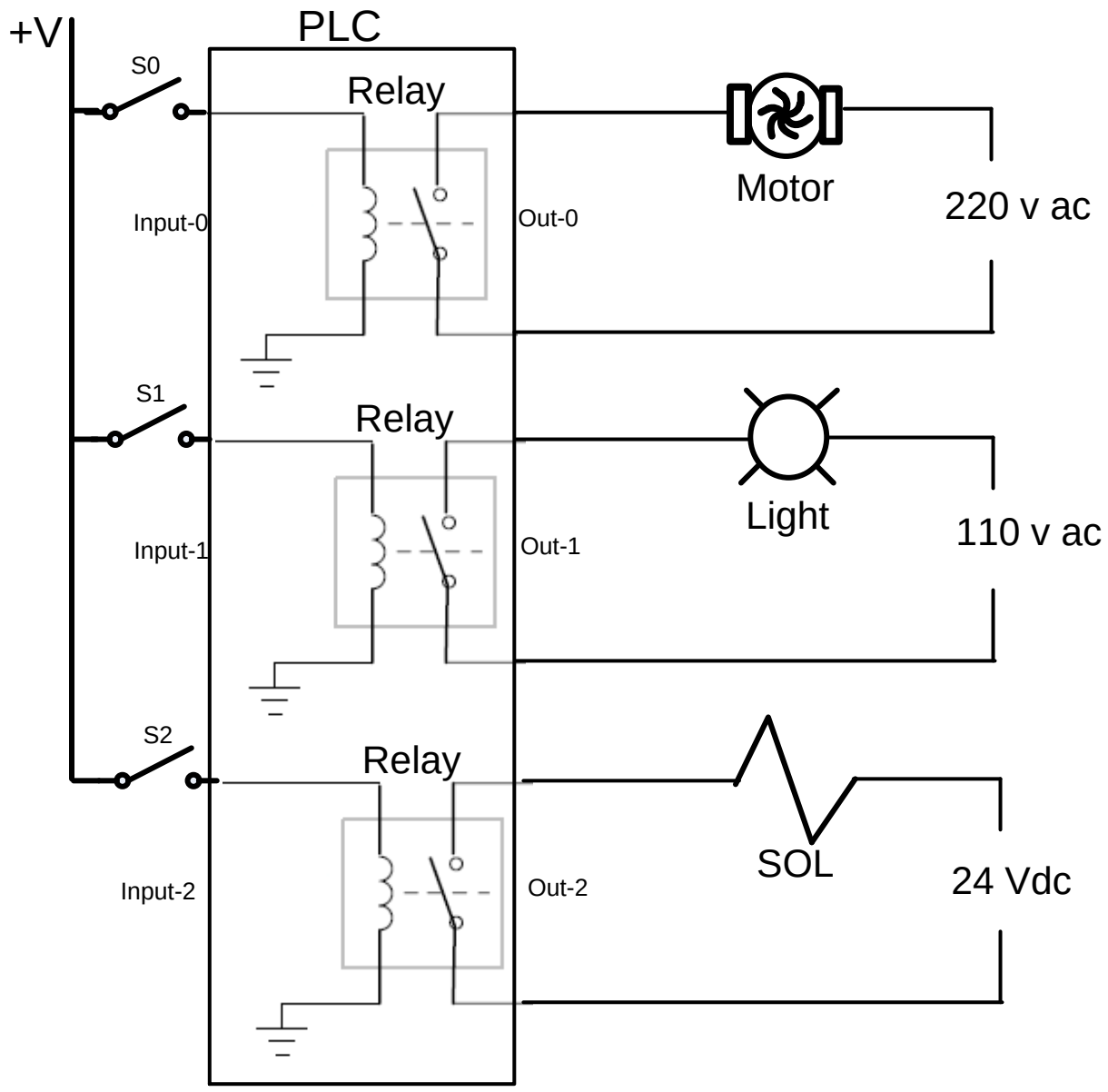


As we see in the figure above the output of PLC is only a switch and the power supply is connected externally. Because the output is only a switch, this will allow us to connect AC voltage as shown in the figure above or DC voltage as shown in the figure below.



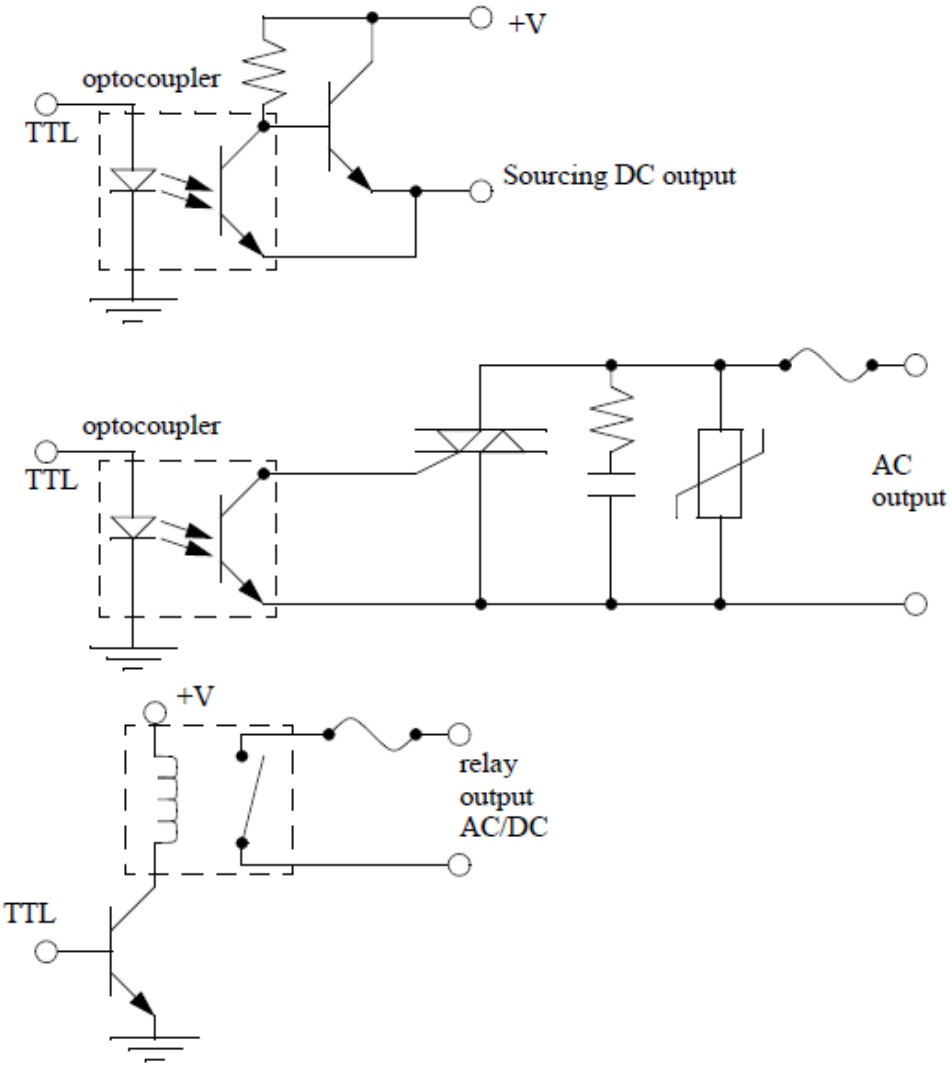
6.08 AC or DC Voltage of PLC Output:

AC or DC voltage can be connected to outputs of the same PLC as shown in the figure below.



6.09 Solid State Semiconductor PLC Output Switch

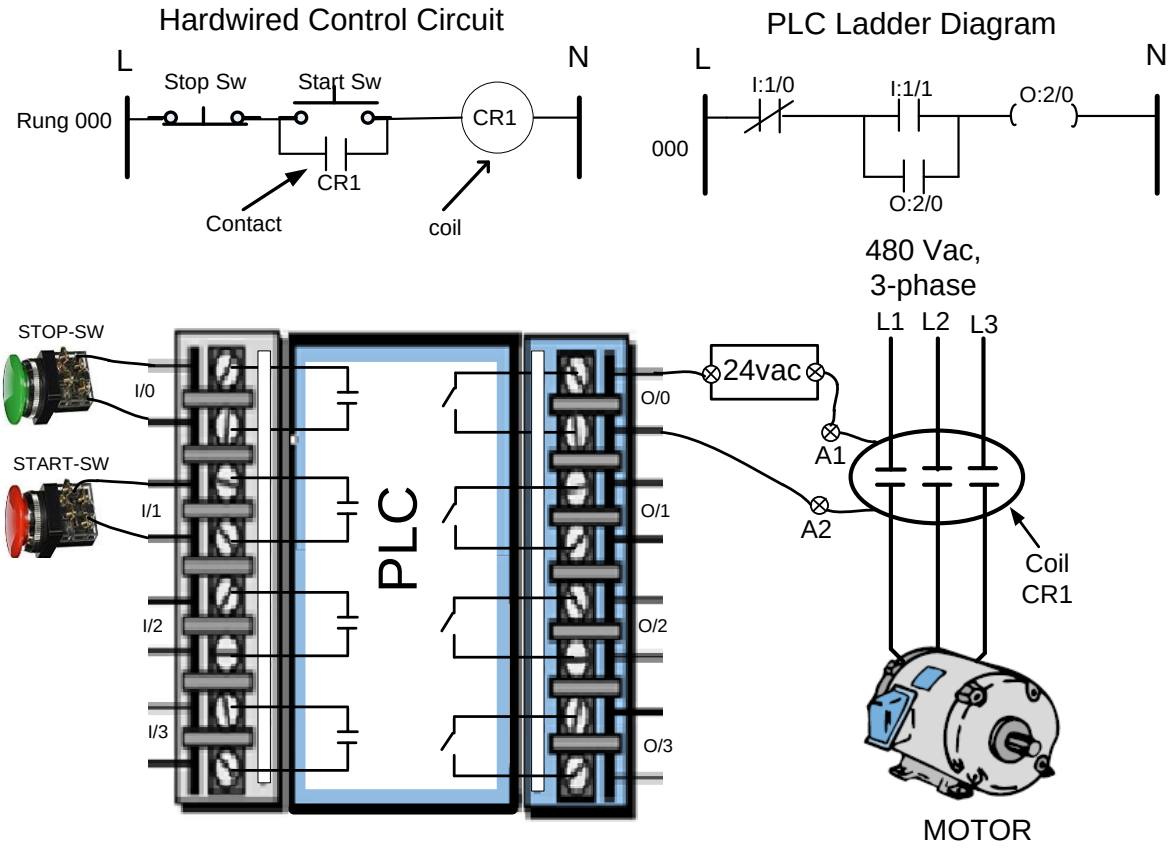
Other relay type circuits are used but the switching mechanism of the outputs is implemented by using solid state semiconductor devices as shown in the figures below.



6.10 PLC Controlled Motor

In this section we will connect a motor control circuit to the PLC, write program and test it.

The diagram below is a typical motor control circuit. Note on the left side the control relay coil CR1and the contact CR1 both have the same name CR1. If the contact belongs to the same control relay, then it must have the same name. On the right side we see in a PLC program, it has the same address O:2/0.



Do not turn power ON until the instructor tells you!!

Connect the components of the PLC control circuit shown in the figure above, enter the PLC program and test it. Show your work to the instructor after completing the exercise.

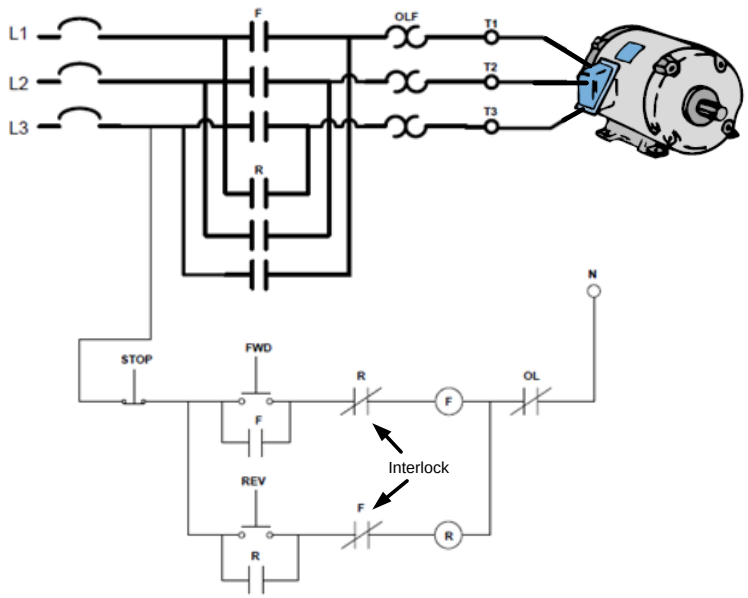
Test your knowledge:

Show your work to the instructor after completing each exercise.

Exercise 6-1:

Write a documented program that will implement the forward/reverse motor starter, with electrical interlocks shown in the diagram below. Use the PLC I/O simulator screen and the following addresses to simulate the program:

Stop PB_ I:0/0
Fwd PB _ I:0/1
Rev PB_ 1:0/2
F_O:0/0
R_O:0/1



a) Using PLC Simulator:

1) Enter the program in the PLC I/O simulator .
2) Run and test the program.
3) Show your work to the instructor.

b) Using AB SLC500 PLC:

Do not turn power ON until the instructor tells you!!

1) Enter the program using the Rockwell RSLogix software.
2) Connect the necessary components to the PLC.
3) Run and test the program.

Chapter-7: Programming Timers

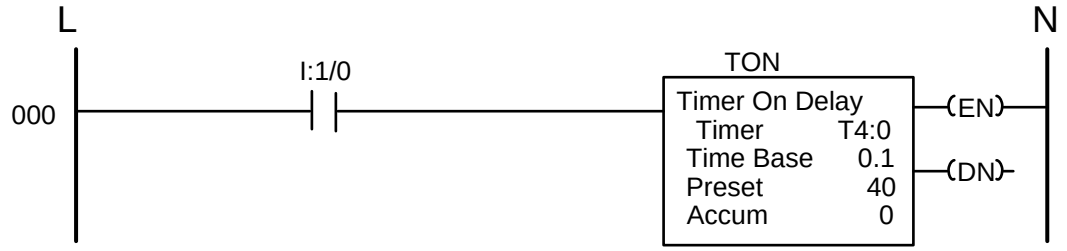
Objectives: Upon completion of this chapter, the student should be able to:

1- Program timer ON Delay (TON) and cascaded timers.
2- Program timer OFF Delay (TOF).
3- Program Retentive timer ON (RTO).
4- Tracing timer operation and timing diagram.

7.01 Introduction to PLC Timer Instruction

No longer needed to buy actual timer, PLC provide for us timer instruction to replace actual timer. Timer can be programmed as PLC software instruction. PLC timer instructions are output instructions that can be used to activate or deactivate a PLC output during definite periods of a number of fixed time intervals.

Timer instruction in our PLC is similar to the timer shown in the following diagram. The address of the timer can be written in a format T4:0, T4:1, T4:2 … The address of the timer shown is T4:0.



7.01.01 Time base

Time base is the duration of each interval counted by the timer instruction

7.01.02 Accumulated Value

Accumulated value is the value accumulated since the timer instruction was last reset to zero. When true, the timer instruction updates the accumulated value continually.

7.01.03 Preset Value

Preset value specifies the final value that the accumulated value must reach. When the accumulated value becomes equal to the preset value, the timer instruction stops timing.

Preset Time value = preset value x time base

7.02 Type of Timers

There are three types of timer available in your PLC trainer:

1) TON (Timer ON Delay)
2) TOF (Timer OFF Delay)
3) RTO (Retentive ON Delay Timer)

7.02.01 TON instruction

The TON instruction starts to count time value when the preceding logic goes from 0 to 1. As long as the rung remains 1, the TON increases its accumulated until it reaches the preset value. If the rung becomes 0, the accumulated value is reset to 0 regardless of whether the preset value has been reached.

7.02.02 TOF instruction

The TOF start to count time value when the preceding logic goes from 1 to 0. As long as the rung remains 0, the TOF instruction increases its accumulated value until it reaches the preset value. If the rung becomes 1, the accumulated value is reset to zero regardless of whether the preset value has been reached.

7.02.03 RTO instruction

RTO starts count time value when the preceding value logic goes from 0 to 1. As long as the rung remains true, the RTO instruction increases its accumulated value until it reaches the preset value.

However, the RTO instruction retains its accumulated value if the rung becomes false. When the rung becomes true again, the RTO resumes timing staring from the accumulated value. The accumulated of the RTO instruction also retained if the PLC is switched from RUN mode to another mode or turned OFF.

Another difference between RTO & TOF is that, with the RTO instruction, the accumulated value is not reset to zero when the rung becomes 0. In fact the accumulated can only be reset by RESET instruction.

7.03 Timer Status Bits

Enable (EN) bit: With a TON or TOF instruction, the EN bit is set when rung conditions are true; it is reset when rung conditions become false.

Timer Timing (TT) bit: With a TON instruction, the TT bit is set when rung conditions are true and the accumulated value is less than the preset value. It is reset when the rung conditions go false or when the done bit is set. With a TOF instruction, the TT bit is set when rung conditions are false and the accumulated value is less than the preset value. It is reset when the rung conditions go true or when the done bit is reset.

Done (DN) bit: With a TON instruction, the DN bit is set when the accumulated value is equal to the preset value. It is reset when rung conditions become false. With a TOF instruction, the DN bit is reset when the accumulated value is equal to the preset value. It is set when rung conditions become true.

The EN, TT, and DN status bits can be used to control Examine If Close (XIC) and Examine If Open (XIO) instructions in a ladder program. These instructions must be addressed by specifying the appropriate status bit number or mnemonic.

Addressing Examples:

T4:0/DN Done bit

T4:0/TT Timer Timing bit

T4:0/EN Timer Enable bit

7.04 The Reset (RES) instruction

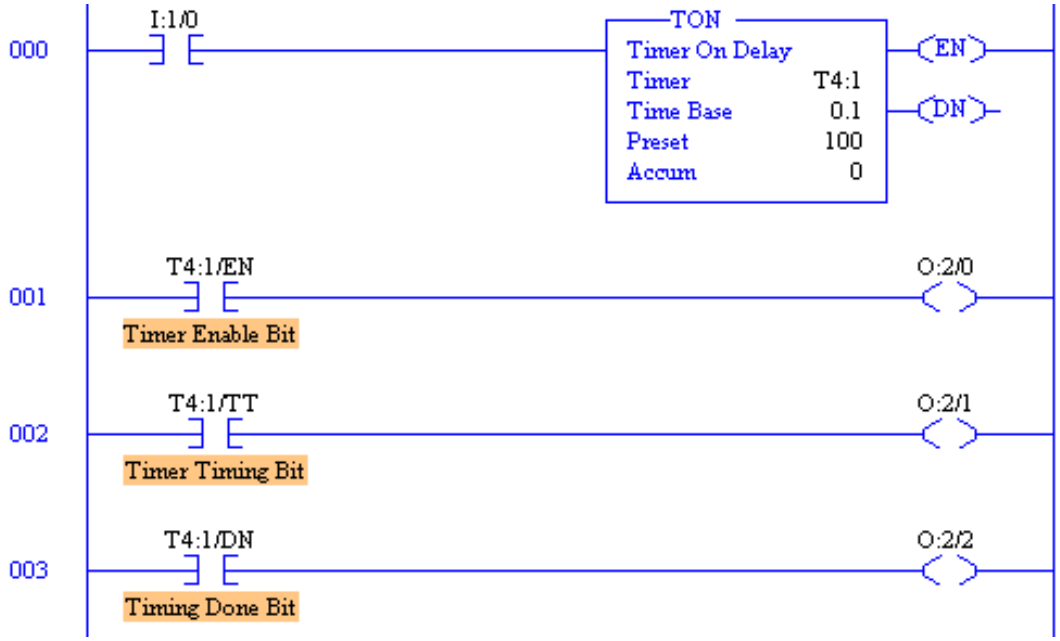
The reset (RES) instruction is used to reset the accumulated value of a timer. When enabled, the reset instruction resets the accumulated value, as well as the Enable, Timer Timing, and Done bits. The reset instruction must have the same address as its associated timer.

7.05 The TON Timer .... (Timer ON Delay)

In this section will enter a TON timer and test it.

- From the LogixPro Simulations Menu, select the I/O Simulation.
- Clear out any existing program by selecting the "New" entry in the File menu, and then select the "Clear Data Table" entry in the Simulations menu.
- Now enter the following program being careful to enter the addresses exactly as shown.
- Confirm that you have entered the number 100 as the timer's preset value. This value represents a 10 second timing interval (10x0.1) as the time base is fixed at 0.1 seconds:

Now write the following ladder logic.



- Once you have your program entered, and have ensured that it is correct, download it to the PLC.

- Ensure that Switch I:1/0 is Open, and then place the PLC into the Run mode.

- Right click on the Timer instruction, and select "GoTo DataTable" from the drop-down menu.

Note the initial value of timer T4:1's accumulator and preset in the spaces below. Also indicate the state of each of the timer's control bits in the spaces provided:

Initial State (Switch I:1/0=Open):

T4:1.ACC = _____ T4:1.PRE = _____ T4:1/EN = ____ T4:1/TT = ____

T4:1/DN= ____

- Close switch I:1/0, and carefully observe the incrementing of the timer's accumulator, and the state of each of it's control bits.

- Once the Timer stops incrementing, note the final value of timer T4:1's accumulator, preset, and the state of it's control bits below:

Final State (Switch I:1/0=Closed):

T4:1.ACC = _____ T4:1.PRE = _____ T4:1/EN = ____ T4:1/TT = ____

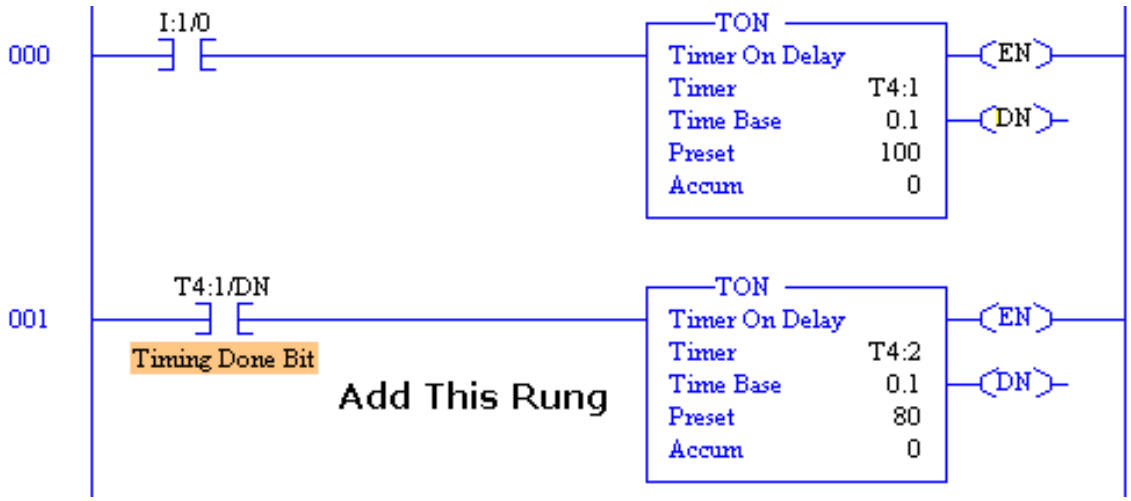
T4:1/DN= ____

- Toggle the state of switch I:1/0 a number of times, and observe the operation of the Timer in both the DataTable display and in the Ladder Rung program display.

- Confirm that when the rung is taken false, the accumulator and all 3 control bits are reset to zero. This type of timer is a non-retentive instruction, in that the truth of the rung can cause the accumulator and control bits to be reset (=0).

7.06 Cascaded TON Timer

Insert a new rung containing a second timer just below the first rung as shown below. This second timer T4:2 will be enabled when the first timer's Done bit T4:1/DN goes true or high (1).



- Once you have completed this addition to your program, download your program to the PLC and select RUN.

- Toggle the state of switch I:1/0 to ON and observe the operation of the timers in your program.

- Bring the DataTable display into view, and pay particular attention to the way in which the timers are cascaded (one timer starts the next).

- Try changing the value of one of the timer presets by double clicking on the preset value in the DataTable display, and then entering a new value.

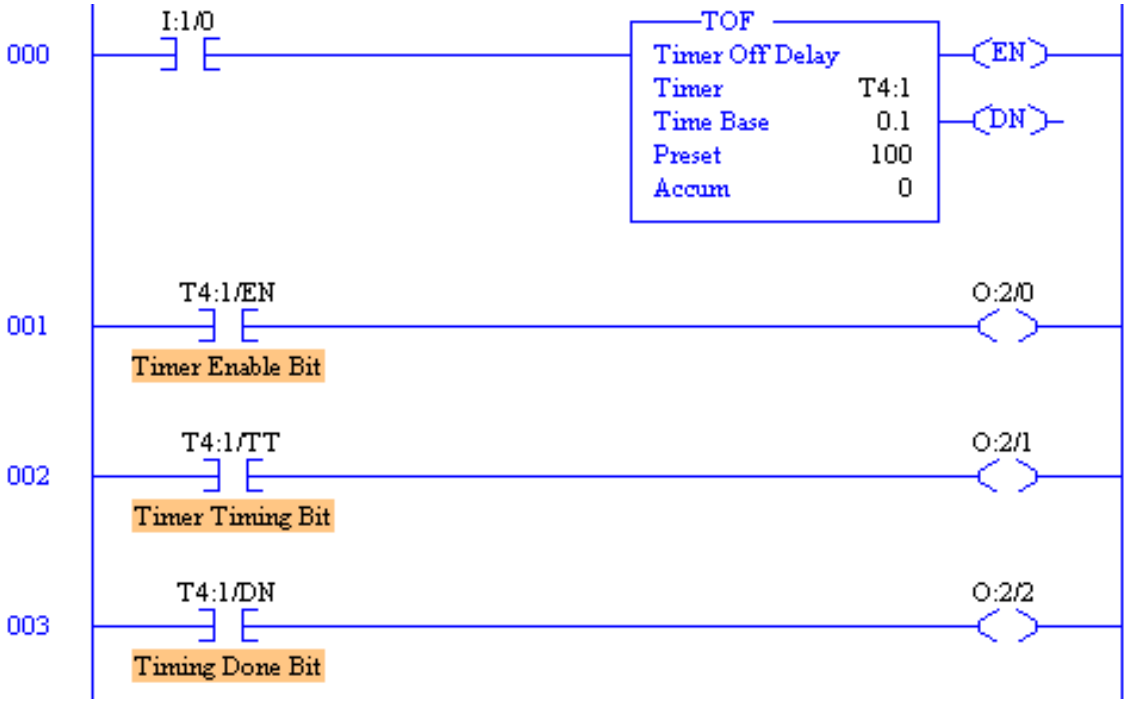
- Run the timers through their timing sequence a number of times. Don't move on until you are satisfied that the timers are working as you would expect

In this exercise we have utilized just two timers, but there is nothing stopping us from sequencing as many timers as we wish.

7.07 The TOF Timer .... (Timer OFF Delay)

In Allen Bradley PLC programming, the TON timer is by far the most commonly used type of timer. Most people consider TON timers to be simple to use and understand. In comparison, many people find the operation of the Allen Bradley TOF (Timer OFF delay) timer to be less intuitive, but I'm going to let you decide for yourself.

- Make sure that switch I:1/0 is Closed, and then enter or modify your existing program to match the one shown below.



- Once you have your program entered, and have ensured that it is correct, download it to the PLC.

- Ensure that Switch I:1/0 is Closed, and then place the PLC into the Run mode.

- Right click on the Timer instruction, and select "GoTo DataTable" from the dropdown menu.

- Note the initial value of timer T4:1's accumulator and preset in the spaces below. Also indicate the state of each of the timer's control bits in the spaces provided:

Initial State (Switch I:1/0=Closed):

T4:1.ACC = _____ T4:1.PRE = _____ T4:1/EN = ____ T4:1/TT = ____

T4:1/DN= ____

- Open switch I:1/0, and carefully observe the incrementing of the timer's accumulator, and the state of each of it's control bits.

- Once the Timer stops incrementing, note the final value of timer T4:1's accumulator, preset, and the state of it's control bits below:

Final State (Switch I:1/0=Open):

T4:1.ACC = _____ T4:1.PRE = _____ T4:1/EN = ____ T4:1/TT = ____

T4:1/DN= ____

- Toggle the state of switch I:1/0 a number of times, and observe the operation of the Timer in both the DataTable display and in the Ladder Rung program display.

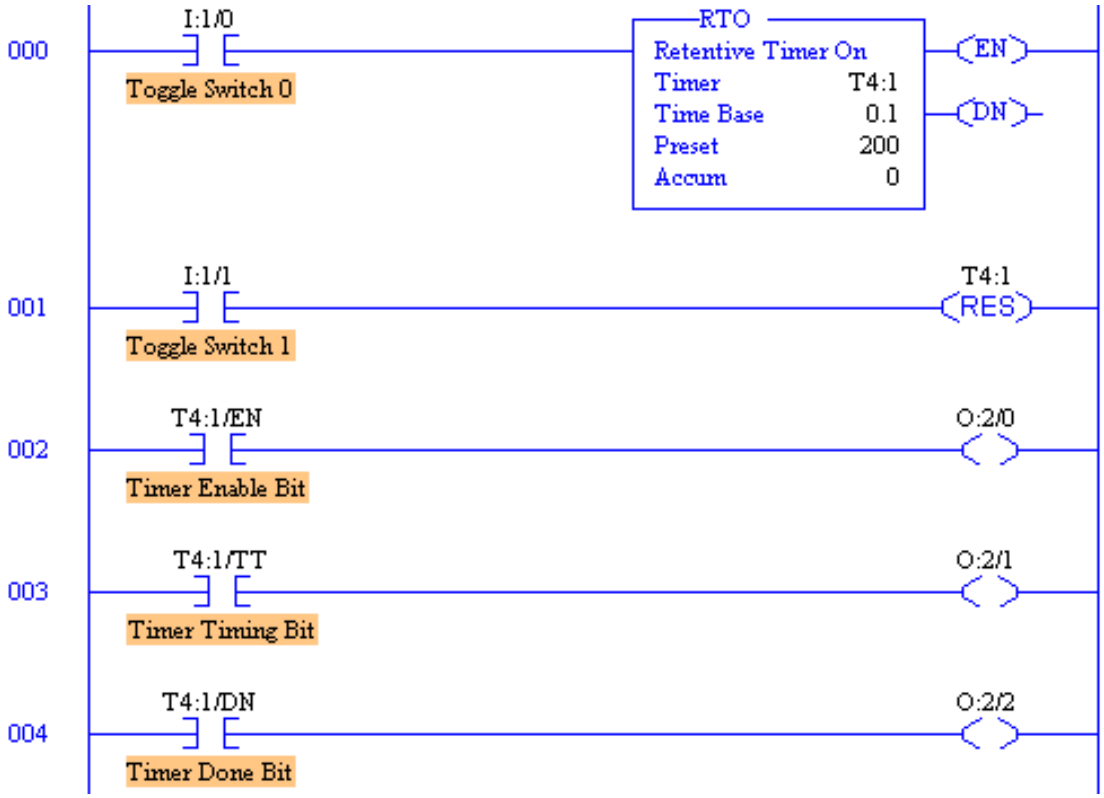
- Confirm that when the rung is taken true, the accumulator and all 3 control bits are reset to zero. The TOF timer like the TON timer is also a non-retentive instruction and can be reset by changing the truth of the rung.

7.08 The RTO Timer .... (Retentive Timer ON)

- Make sure that switch I:1/0 is Open, and then replace the TOF timer in your program with a RTO retentive timer.

- Now insert a new rung below the timer, and add the XIC,I:1/1 and RES,T4:1 instructions.

- Your program should now match the one shown below:



- Once you have your program entered, and have ensured that it is correct, download it to the PLC.

- Ensure that both Switches are Open, and then place the PLC into the Run mode.

- Right click on the Timer instruction, and select "GoTo DataTable" from the dropdown menu.

- Note the initial value of timer T4:1's accumulator, preset and control bits. Are we starting off with the same values we had in the TON exercise? You should be answering Yes.!

- Close switch I:1/0 for 2 or 3 seconds and then Open it again.

- Note that the timer stopped timing when the rung went false, but the accumulator was not reset to zero.

- Close the switch again and leave it closed which will allow the timer to time-out (ACC=PRE).

- Once timed out, note the state of the control bits

- Open the switch, and once again note the state of the control bits.

- Now close Switch I:1/1 and leave it closed. This will cause the Reset instruction to go true.

- Close switch I:1/0 momentarily to see if the timer will start timing again. It should not!

- Open Switch I:1/1 which will cause the Reset instruction return to false.

- Now toggle switch I:1/0 several times and note that the timer should again start timing as expected.

- Repeat the foregoing steps, until you are satisfied that you clearly understand the operation of both the RTO timer, and the Reset instruction.

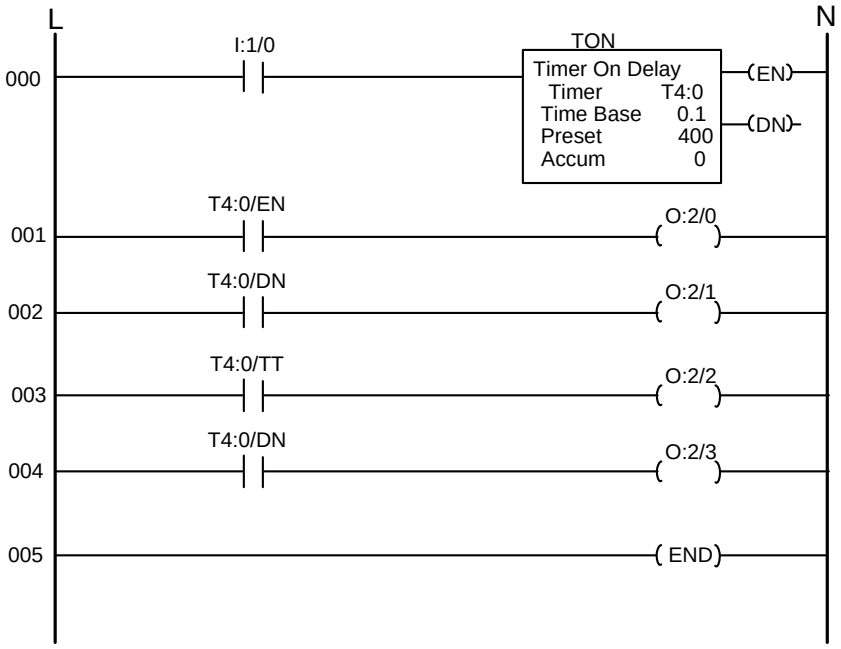
7.09 Timing Diagram

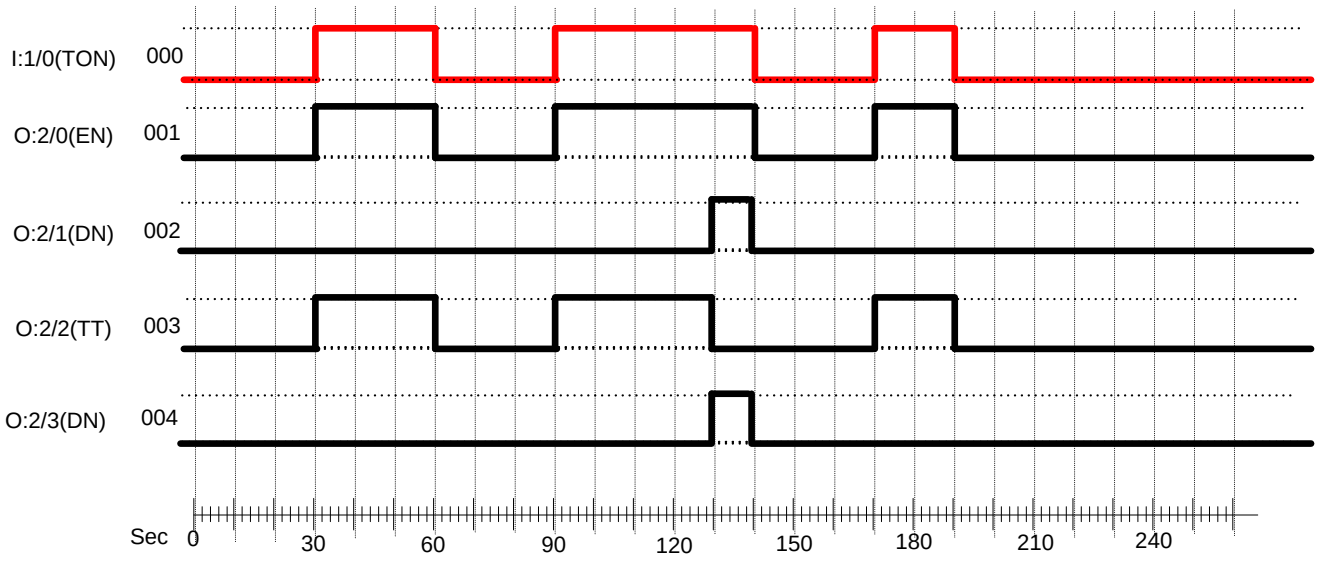
To understand the timer instruction better, you need to learn the drawing of timing diagram. From the timing diagram you can have clear picture of the status (TRUE/FALSE) of each instruction or rung in the PLC program. In the following examples you will learn how to draw the timing diagram.

Example 7.01:

Timing Diagram for TON instruction.

In this example if you are given the Ladder Diagram and the status of rung 000 as a timing diagram, you should be able to find the timing diagram of the rungs 001 to 004.



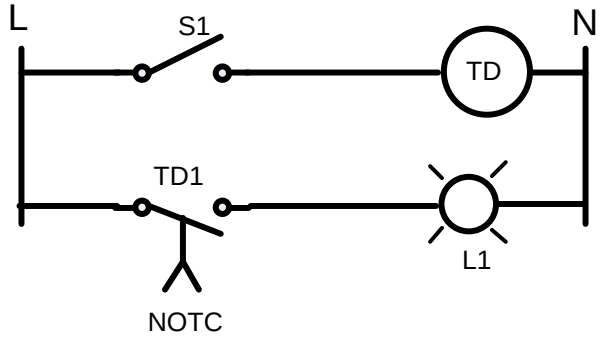


Test your knowledge:

Exercise 7.01:

Write a documented program that will simulate the on-delay relay timer schematic shown. Use the PLC I/O simulator screen and the following addresses to simulate the program. Show your work to the instructor after completing the exercise.

S1_I:0/0
L1_O:0/0
TON_T4:0



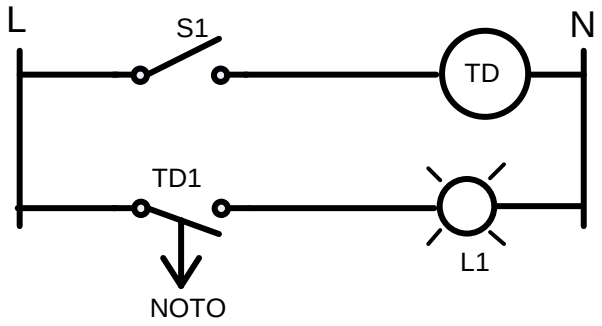
Sequence of operation:

1) S1 open, TD de-energized, TD1 open, L1 off
2) S1 closes, TD energizes, timing period starts, TD1 is still open, L1 is still off.
3) After 10 s, TD1 closes, L1 is switched on
4) S1 is opened, TD de-energizes, TD1 opens instantly, L1 is switched off.

Exercise 7.02:

Write a documented program that will simulate the off-delay relay timer schematic shown. Use the I/O simulator screen and the following addresses to simulate the program. Show your work to the instructor after completing the exercise.

S1_**I:0/0**
L1_O:0/0
TOF_T4:0



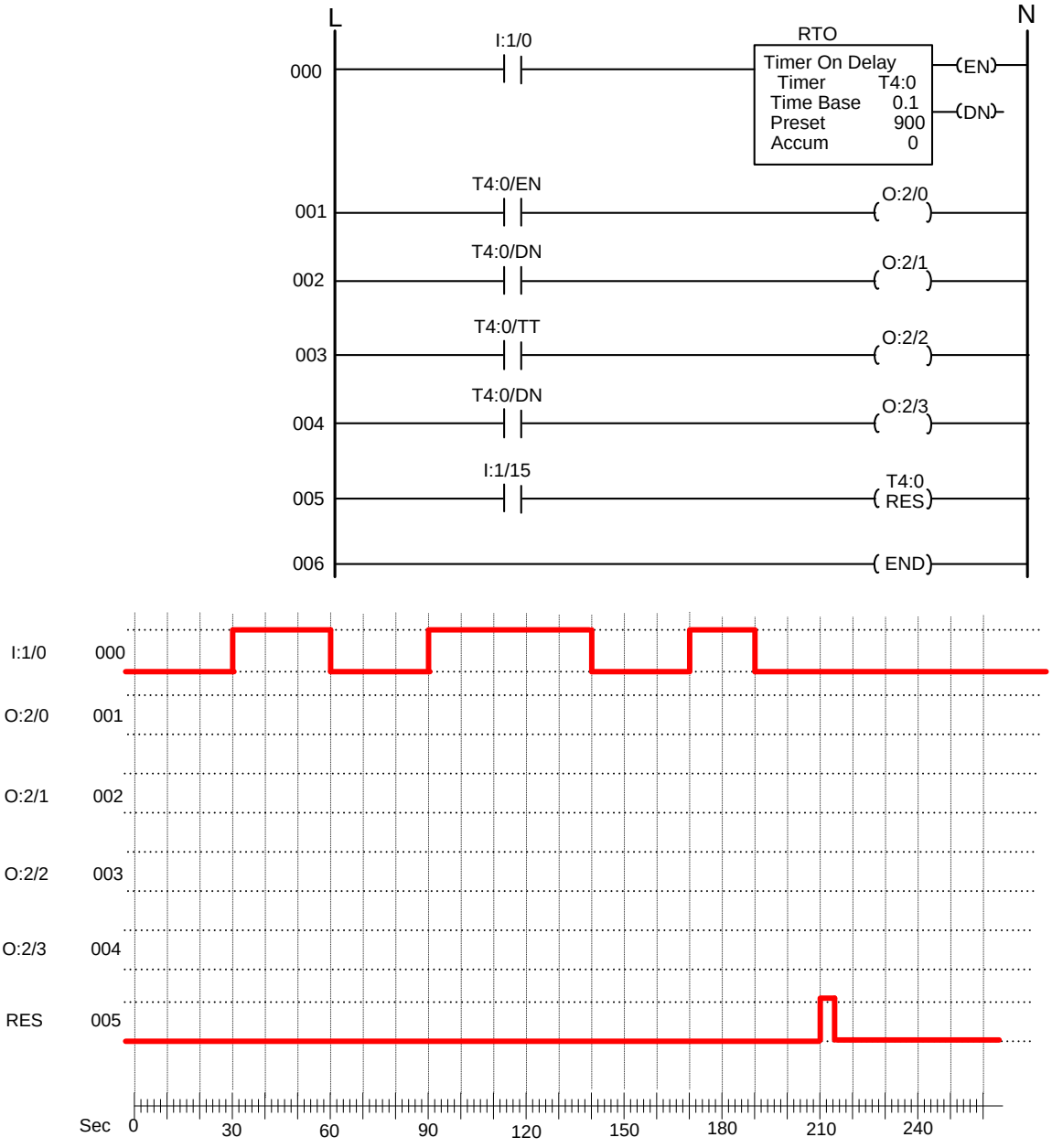
Sequence of operation:

1) S1 open, TD de-energized, TD1 open, L1 off,
2) S1 closes, TD energizes, TD1 closes instantly, L1 is switched on.
3) S1 is opened, TD de-energizes, timing period starts, TD1 is still closed, L1 is still on.
4) After 10 s, TD1 opens, L1 is switched off.

Exercise 7.03:

Timing Diagram for RTO instruction.

In this example if you are given the Ladder Diagram and the status of rung 000 and 005 the reset as a timing diagram, you should be able to find the timing diagram of the rungs 001 to 004.



Show your work to the instructor after completing the exercise.

This Page is Left Blank

Chapter-8: Programming Counters

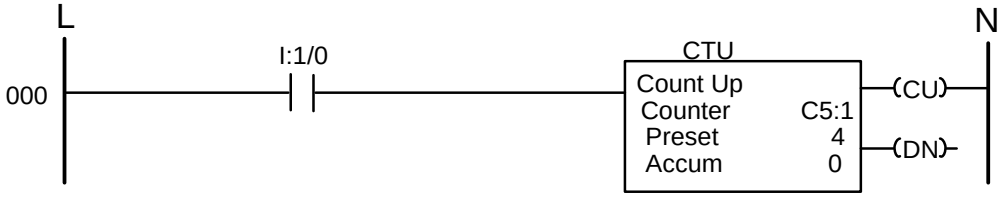
Objectives: Upon completion of this chapter, the student should be able to:

1- Program up counter (CTU)
2- Understand down counter (CTD)
3- Program up/down counter (CTU/CTD)
4- Tracing counter operation and counter timing diagram.

8.01 Introduction to PLC Counter Instruction

PLC counter instructions are output instructions that can be used to activate or deactivate a PLC output after a specific number of events have occurred. A counter instruction counts the number of times that the rung in which it is contained makes transitions. Rung transitions can be caused by events such as parts moving past a sensor or actuating a limit switch.

Timer instruction in our PLC is similar to the timer shown in the following diagram. The address of the timer can be written in a format C5:0, C5:1, C5:2 … The address of the timer shown is C5:1.



8.01.01 Accumulated Value

The accumulated is the number of transitions made by the rung of the counter instruction since this instruction was last reset.

8.01.02 Preset Value

The preset value specifies the final value that the accumulated must reach. There are two types of counter available in this PLC:

8.02 Type of Counters

1) CTU (Count Up)
2) CTD (Count Down)

- The CounT Up (CTU) instruction increases the counter accumulated value by one count on each false-to-true transition of the CTU instruction rung.
- The CounT Down (CTD) instruction decreases the counter accumulated value by one count on each false-to-true transition of the CTD instruction rung.

The counter accumulated value (count) is retained until cleared by a RESET (RES) instruction having the same address as the counter.

The preset value corresponds to the value that the counter must reach before the DONE (DN) bit is set. The preset value of a counter can be set to any value between +32767 and -32768.

8.03 Counter Data File Structure

Counters are represented in memory on the form of 3-word elements. Word 0 is the control word, word 1 stores the preset value, and word 2 stores the accumulated value, as Table 8-1 shows.

word	B_{15}	B_{14}	B_{13}	B_{12}	B_{11}	B_{10}	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	CU	CD	DN	OV	UN				Internal use							
1	Preset Vale (PRE)															
2	Accumulated Value (ACC)															

Table 8-1 counter data file structure

Word 0 includes status bits which provide information on the counting process. These bits are the Count Up enable (CU) bit, the Count Down enable (CD) bit, the DONE (DN) bit, the Over flow (OV) bit, and the Under flow (UN) bit.

Count Up enable (CU) bit: Associated with the CTU instruction. This bit is set when rung conditions are true. It is reset when rung conditions go false or a counter reset instruction is enabled.

Count Down enable (CD) bit: Associated with the CTD instruction. This bit is set when rung conditions are true. It is reset when rung conditions go false or a counter reset instruction is enabled.

DoNe (DN) bit: This bit is set when the accumulated value is equal to or greater than the preset value. It is reset when the accumulated value becomes less than the preset value.

OVerflow (OV) bit: This bit is set when the accumulated value wraps around to -32768 (from +32 767) and continues to be increased from there. It is reset when a counter reset instruction is enabled.

UNderflow (UN) bit: This bit is set when the accumulated value wraps around to +32 767 (from -32 768) and continues to be decreased from there. It is reset when a counter reset instruction is enabled.

8.04 Addressing Counter Status Bits

The counter status bits can be used to control relay-type instructions (Examine If Closed [XIC] and Examine If Open [XIO]) in a ladder program. To do so, the relay type instruction must be addressed by specifying either the status bit number or mnemonic.

Addressing Examples:

C5:0/13 or C5:0/DN : DoNe bit

C5:0/14 or C5:0/CD : Count Down enable bit

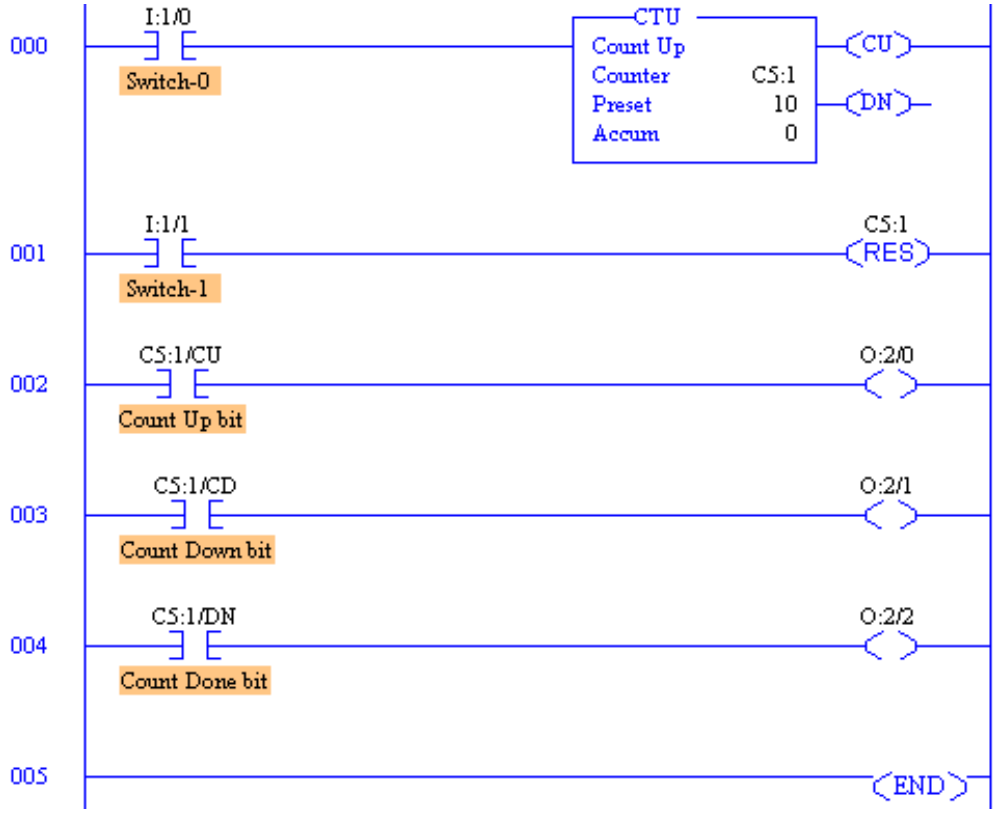
C5:0/15 or C5:0/CU : Count Up enable bit

8.05 Using the Reset (RES) Instruction

The RESet (RES) instruction can be used to reset the accumulated value of a counter. When enabled, the reset instruction resets the accumulated value, as well as the Count Up enable (CU) bit, the Count Down enable (CD) bit, the Overflow (OV) bit, and the Under flow (UN) bit. The RES instruction must have the same address as the associated counter.

8.06 The CTU and RES ..... Counter Instructions

- From the LogixPro Simulations Menu, select the I/O Simulation.
- Clear out any existing program by selecting the "New" entry in the File menu, and then select the "Clear Data Table" entry in the Simulations menu.
- Now enter the following program being careful to enter the addresses exactly as shown.
- Confirm that you have entered the number 10 as the counter's preset value. This value is optionally used to set the point at which the counter's Done Bit will be Set, indicating that the count is complete.



- Once you have your program entered, and have ensured that it is correct, download it to the PLC.

- Ensure that Switch I:1/0 and I:1/1 are Open, and then place the PLC into the Run mode.

- Right click on the CTU instruction, and select "GoTo DataTable" from the dropdown menu.

- Note the initial value of Counter C5:1's accumulator and preset in the spaces below. Also indicate the state of each of the Counter's primary control bits in the spaces provided:

Initial State (Switch I:1/0=Open):

C5:1.ACC = _____ C5:1.PRE = _____ C5:1/CU = _____ C5:1/CD = _____
C5:1/DN = _____

- Open and Close switch I:1/00 a number of times and carefully observe the incrementing of C5:1's accumulator and the operation of the enable and done bits.

- Close switch I:1/01 and observe the effect that the "RES" instruction has on the counter.

- Attempt to increment the counter while switch I:1/01 is closed. You should not be able to increment the counter while the "RES" instruction is held "True".

- Open switch I:1/01 to allow the "RES" instruction to go false, and then increment the counter until the accumulator matches the preset.

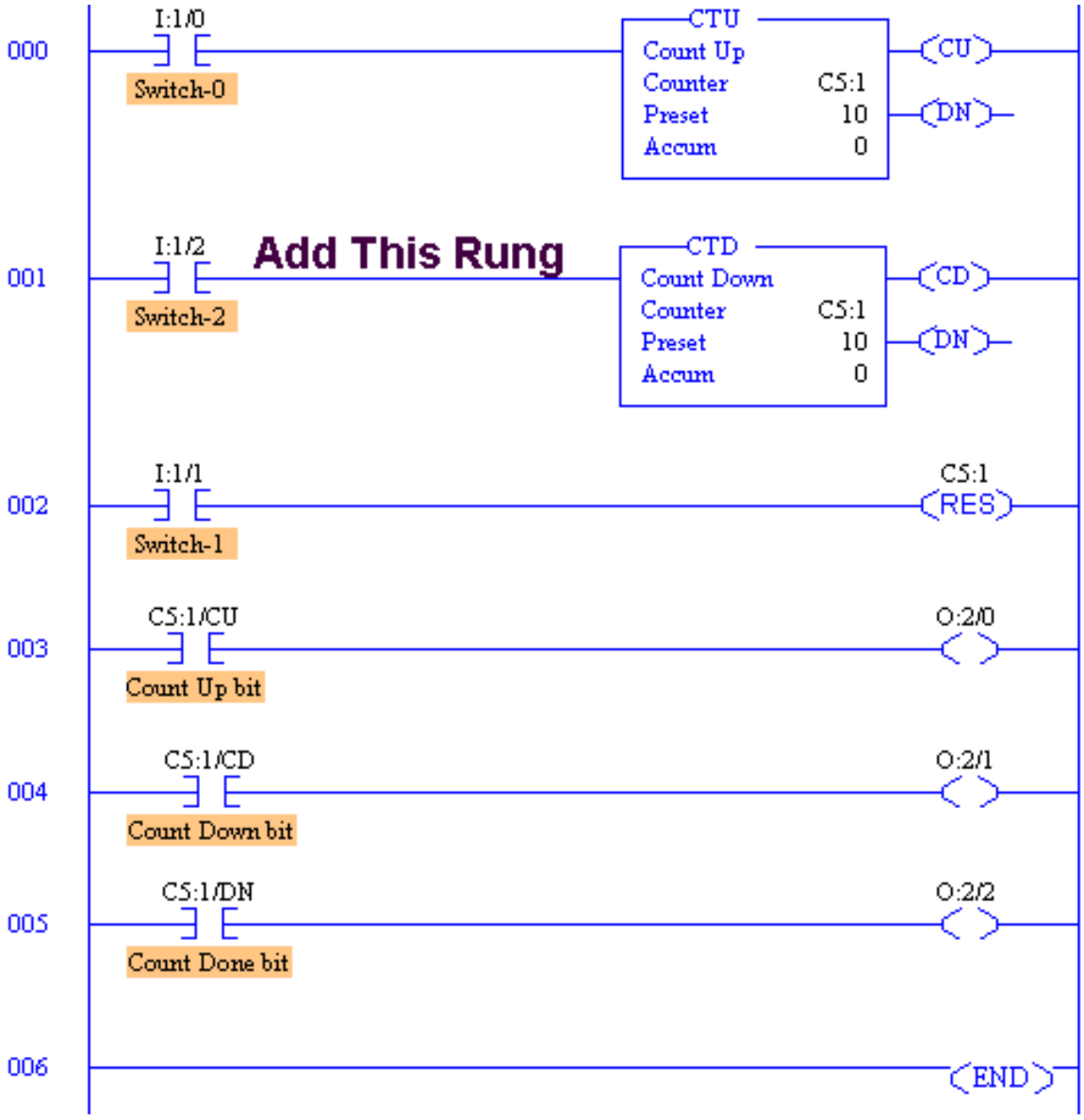
- Increment the counter 2 or 3 more times and note the final value of C5:1's accumulator, preset and status bits in the spaces below.

Final State (Switch I:1/0=Closed):

C5:1.ACC = _____ C5:1.PRE = _____ C5:1/CU = _____ C5:1/CD = _____
C5:1/DN = _____

8.07 The CTD ..... Count Down Instruction

Ensure that switch I:1/00 and I:1/01 are open; then place the PLC into the Program mode, and Insert a new rung containing a CTD instruction just below the first rung in your program.



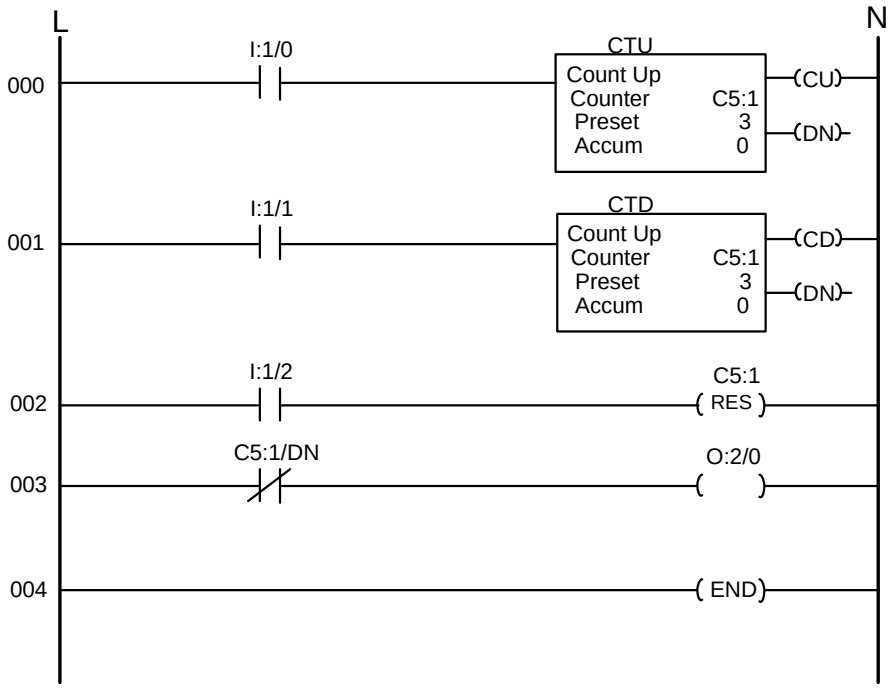
- Once you have completed this addition to your program, download your program to the PLC and select RUN.

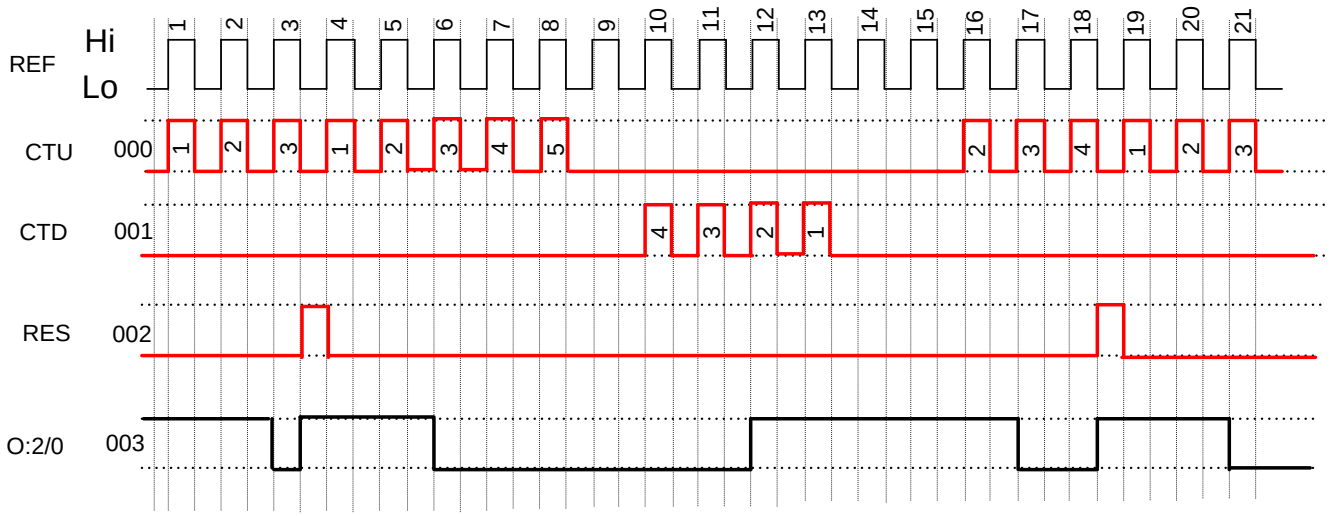
- Toggle the state of switch I:1/0 continuously until the accumulator of C5:1exceeds the preset.

- Now toggle switch I:1/02 and decrement counter C5:1 while carefully observing the status bits of the counter. Increment and decrement the counter from below zero to beyond the preset a number of times.

8.08 Timing Diagram for Up/Down Counter instruction.

In this example if you are given the Ladder Diagram and the status of rung 000, 001 and 002 as a timing diagram, you should be able to find the timing diagram of the rungs 003.





Test your knowledge:

Exercise 8.01:

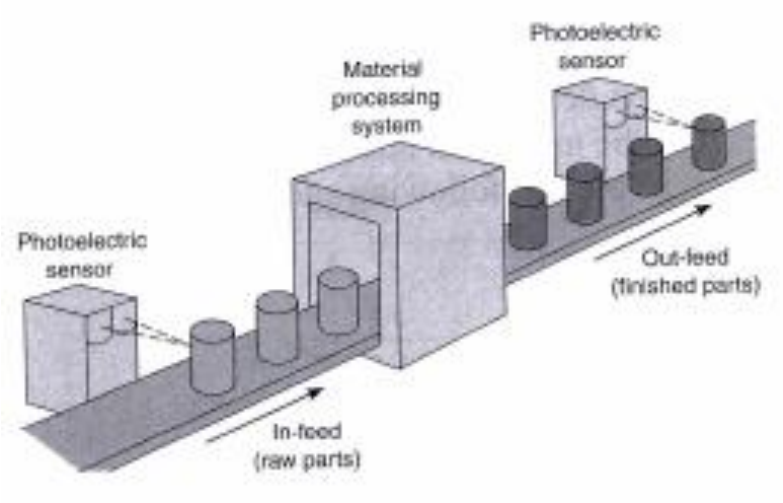
Up/Down counter instruction.

Implement the up/down counter program used in an in-process monitoring system. This program is designed to provide continuous motoring of items in-process and operates as follows:

1) An in-feed photoelectric sensor counts raw parts going into the system.
2) An out-fed photoelectric sensor counts finished parts leaving the machine.
3) The number of parts between the in-feed and out-feed is indicated by the accumulated count of the counter.
4) Before start-up, the system is completely empty of parts and the counter is reset manually to zero.
5) The counter preset value is irrelevant in this application since the output on or off logic is not used.

Use the PLC I/O simulator screen and the following addresses to simulate the program.

In-Feed Count_ I:1/0
Out-Feed Count_ I:1/1
Reset Button_I:1/2
Up/Down Counter_C5:1

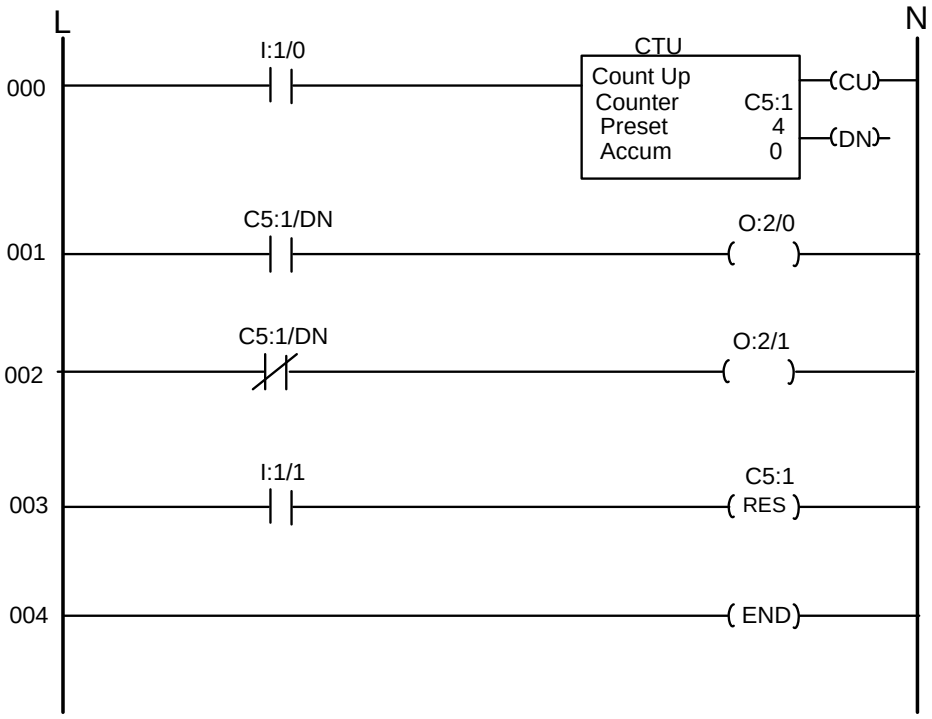


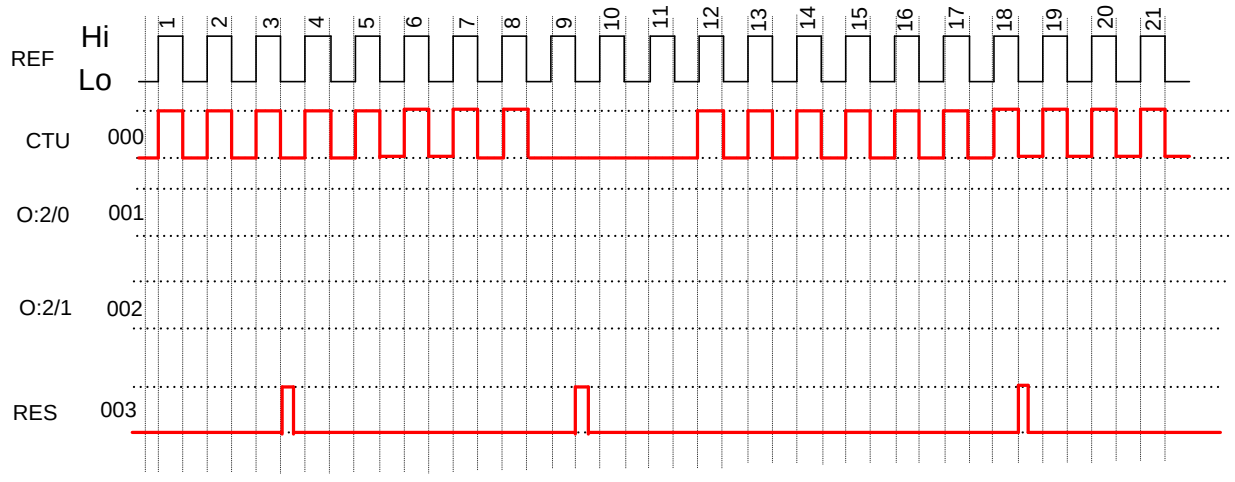
Show your work to the instructor after completing the exercise.

Exercise 8.02:

Timing Diagram for Up Counter instruction.

In this example if you are given the Ladder Diagram and the status of rung 000, and RESET as a timing diagram, you should be able to find the timing diagram of the rungs 001 and 002.





Show your work to the instructor after completing the exercise.

Chapter-9: Comparison Instructions

Objectives: Upon completion of this chapter, the student should be able to:

1- Understand comparison instructions.
2- Know the logic of LIM comparison instruction.
3- Determine the effect of MEQ comparison instruction.
4- Use comparison instructions in a PLC control program.

9.01 The basic Comparison instructions

The basic Comparison instructions compare the values stored in two memory locations. These two values can be the data stored in two different word locations, or one can be the data stored in a word and the other can be a constant value. The basic comparison instructions are listed in the following table:

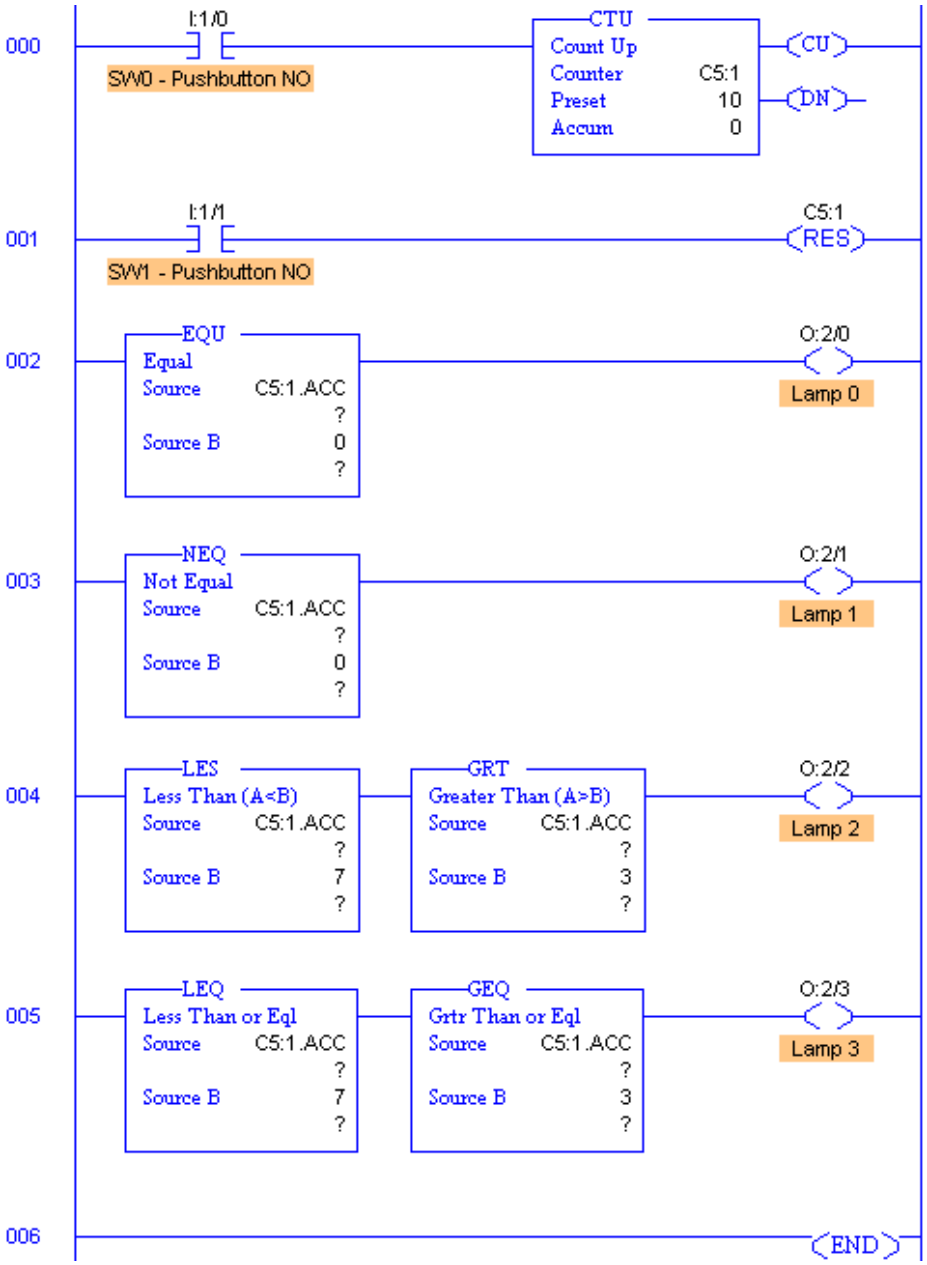
Instruction	Brief Description
EQU	The Equal instruction goes true if the source A and B values are Equal to each other.
NEQ	The Not Equal instruction goes true if the source A and B values are Not Equal to each other.
LES	The Less Than instruction goes true if the value in source A is Less Than the value in source B.
GRT	The Greater Than instruction goes true if the value in source A is Greater Than the value in source B.
LEQ	The Less Than OR Equal instruction goes true if the value in source A is Less Than or Equal to the value in source B.
GEQ	**GEQ** .... The Greater Than OR Equal instruction goes true if the value in source A is Greater Than or Equal to the value in source B

Since any PLC word including Timer and Counter accumulators and presets can be used as the source value in any of the basic comparison instructions, these instructions prove extremely versatile and are widely used in RSLogix programs.

9.02 The RSLogix basic Comparison Instructions

- From the LogixPro Simulations Menu, select the I/O Simulation. Clear out any existing program by selecting the "New" entry in the File menu, and then select the "Clear Data Table" entry in the Simulations menu.

- Now enter the following program being careful to enter the addresses and values exactly as shown.



- Once you have completed entering your program, download your program to the PLC. Ensure that SW0 and SW1 are configured as Normally Open pushbuttons then place the PLC into the Run mode.

- Toggle the state of switch SW0 (I:1/0) continuously while observing the truth of each of rungs as indicated by the lamps.

- Once the count exceeds nine or ten, reset the counter and repeat the above sequence. Keep doing this until you are convinced that the instructions are operating as described in the RSLogix documentation.

Finally, indicate the observed state of the lamps, by circling the appropriate numbers below:

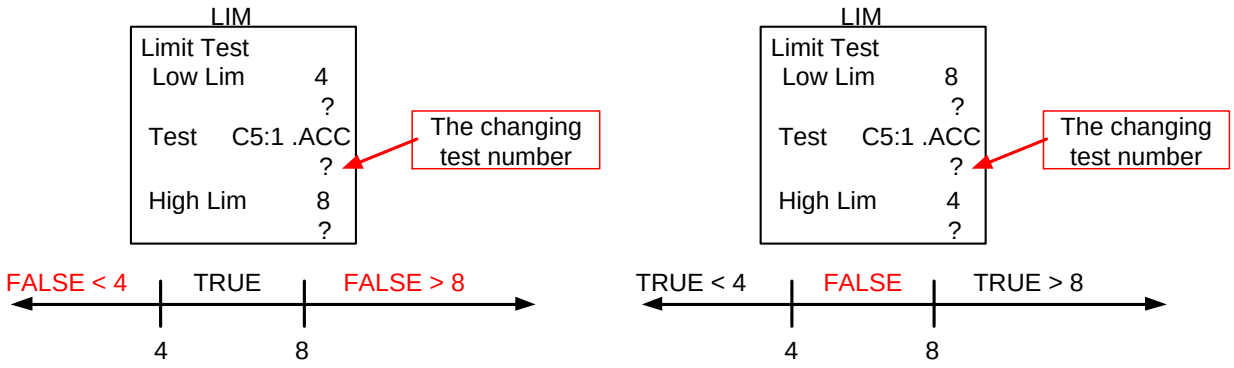
Lamp 0 is On during counts: 1...2...3...4...5...6...7...8...9...10
Lamp 1 is On during counts: 1...2...3...4...5...6...7...8...9...10
Lamp 2 is On during counts: 1...2...3...4...5...6...7...8...9...10
Lamp 3 is On during counts: 1...2...3...4...5...6...7...8...9...10

9.03 The LIM Instruction (Limit Comparison)

Programming the LIM instruction consists of entering 3 parameters: 1) Low limit, 2) Test, and 3) High limit. The limit test instruction functions in the following two ways:

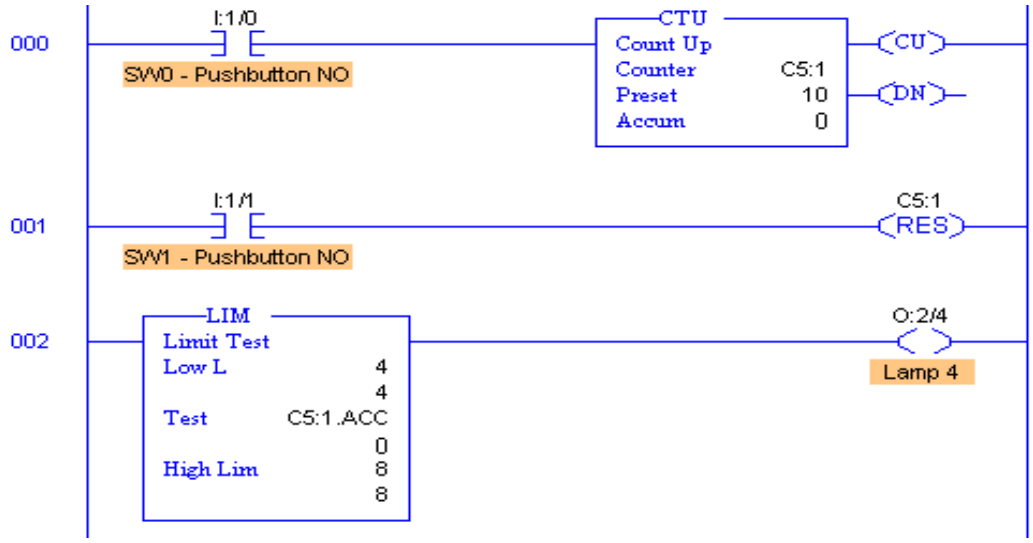
• ***The instruction is true if*** —The lower limit is equal to or less than the higher limit, and the test parameter value is equal to or inside the limits. Otherwise the instruction is false.
• ***The instruction is true if*** —The lower limit has a value greater than the higher limit, and the instruction is equal to or outside the limits. Otherwise the instruction is false.

The two diagrams below explain the operation of the LIM instruction.



- Write the PLC 3 rungs program shown below

- Ensure that addresses and values are exactly as shown.



- Once you have completed your program, download your program and place the PLC into the Run mode.

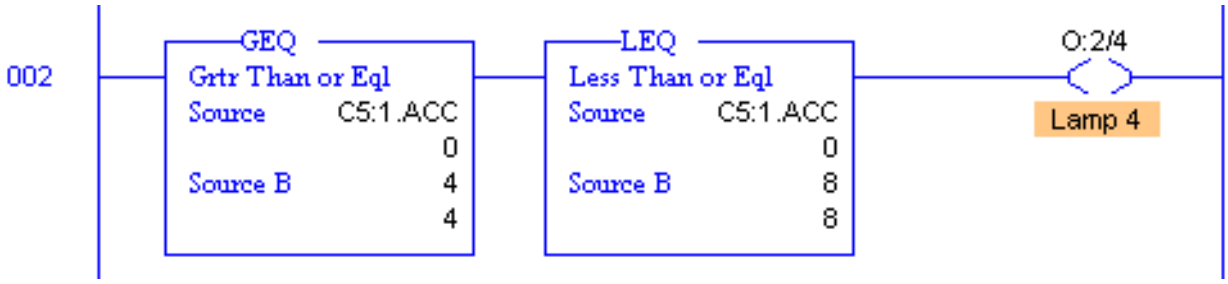
- Toggle the state of switch SW0 (I:1/0) continuously while observing the truth of Lamp 4.

- Once the count exceeds ten, reset the counter and repeat the above sequence. Keep doing this until you are convinced that the LIM instruction is operating as described in the RSLogix documentation.

Finally, indicate the observed state of Lamp 4, by circling the appropriate numbers below:

Lamp 4 is On during counts: 1...2...3...4...5...6...7...8...9...10

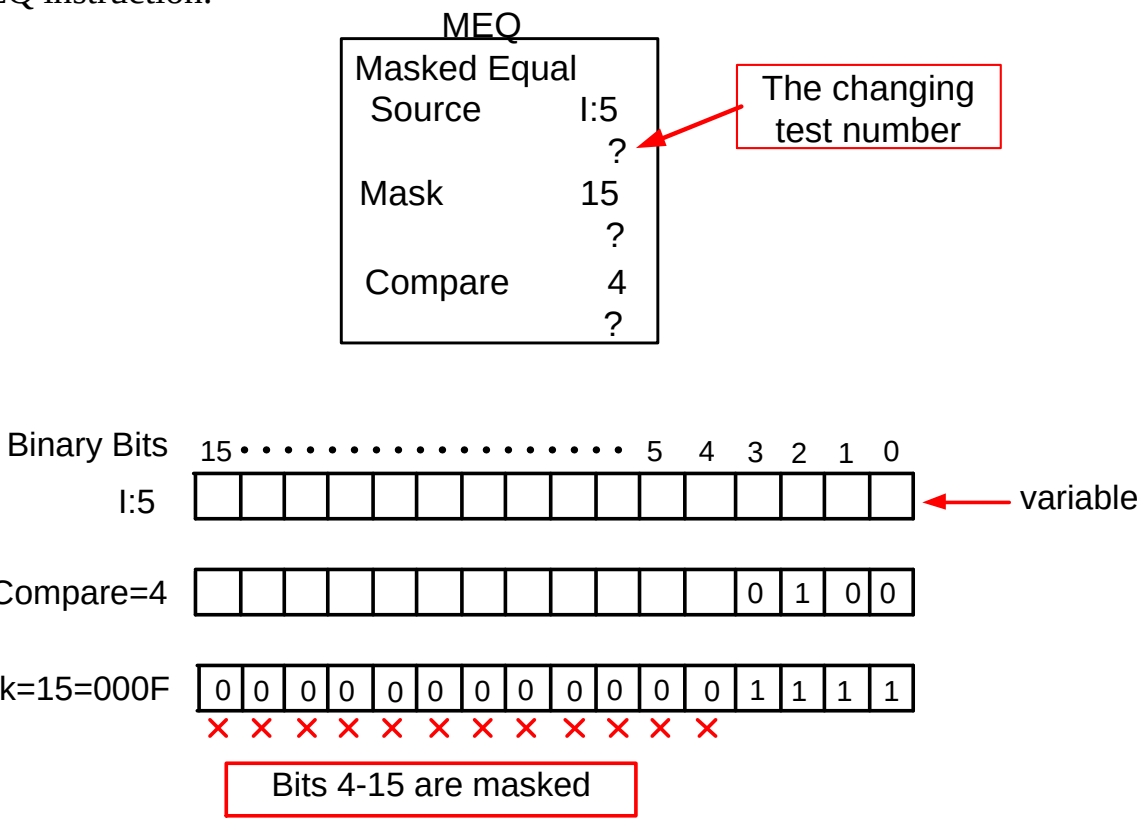
The LIM (Limit Comparison) instruction compares the Test value to the value of the Low Limit and the value of the High Limit. The instruction goes true if Test is Equal to or Greater than the Low Limit and Test is Less Than or Equal to the High Limit. A logical equivalent to rung 2 is shown below:



The LIM instruction provides in a single package the same functionality that would normally necessitate the utilization of 2 basic comparison instructions.

9.03 The MEQ Instruction (Masked, Equal Comparison)

The MEQ instruction has the same functionality of an EQU instruction, but it allows you to first mask out any extraneous information or bits prior to doing the actual test for equality. The diagram below show how the data is stored in memory for MEQ instruction.

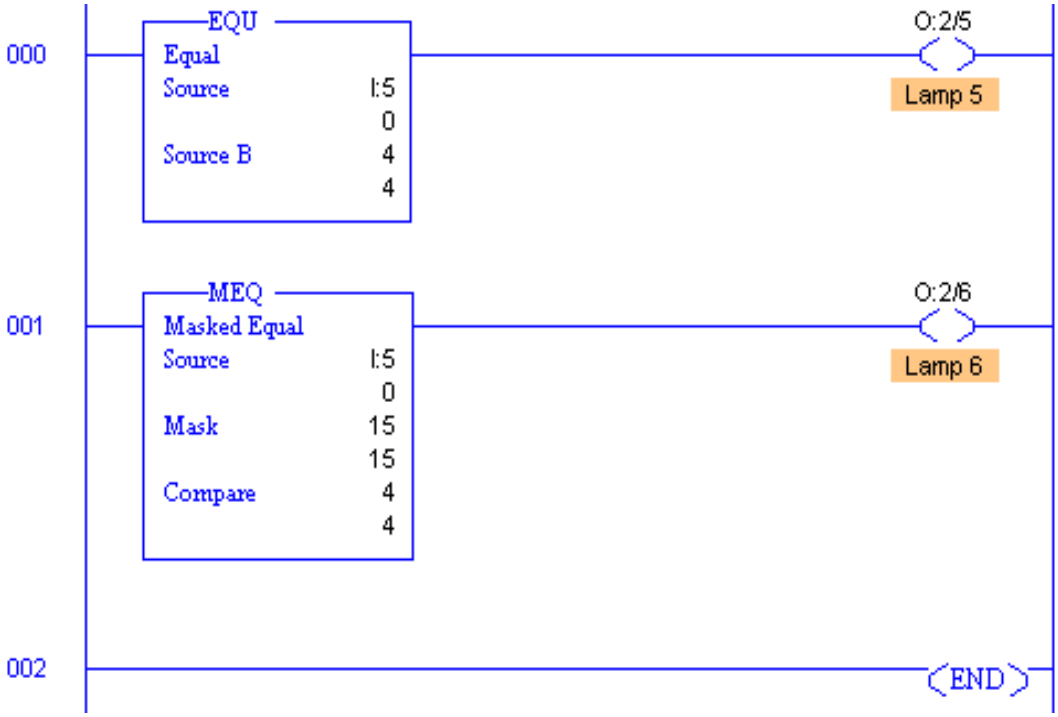


In the example above MEQ instruction is TRUE when the input is equal 4 that is (I:5) = 4. Bits 4-to-15 are masked which means they will not effect the output.

The MEQ instruction is sometimes considered an advanced level instruction which many might deem inappropriate for inclusion in a basic level exercise. The MEQ is however listed with the other comparison instructions, so a quick peek at it's functionality should not cause major harm. The subject of "Masking" relates to the act of controlling which bits within a binary value or word are passed through to a destination.

The following program is a very simple demonstration of how extraneous information can be selectively ignored with the judicious use of masking. There are several other instructions in the RSLogix instruction set that employ masking and work similarly, but only the MEQ instruction will be reviewed here.

- Clear your existing program by selecting the "New" entry in the File menu.
- Now enter the following program being careful to enter the addresses and values exactly as shown.
- Note: address I:5 is the address of the I/O simulator's input card which has the thumbwheel switches connected to it.



The MEQ and EQU instructions are almost identical in operation. The only difference is that with the MEQ instruction, selected bits within the Source value can be "masked out" or deleted prior to doing the comparison for Equality. In the above MEQ example we are going to mask out all bits other than the 4 which contain the data from the first thumbwheel switch.

- Once you have completed your program, download your program and place the PLC into the Run mode.

- Starting with the right hand (units) thumbwheel only, increment the displayed value up and down and note how both lamps energize when the value is set to 4.

- Now set the first thumbwheel to 4 and start incrementing the second (tens) thumbwheel. If your program is operating correctly only lamp 6 should remain lighted.

- Finally, set the thumbwheel values to match those listed below, and circle the appropriate number if the corresponding lamp is On.

  Lamp 5 is On when the thumbwheel value is: 1...4...14...34...54...94...104
  Lamp 6 is On when the thumbwheel value is: 1...4...14...34...54...94...104

Test your knowledge:

Exercise 9.01:

Write a documented program that will turn 3 lights ON when you press a pushbutton 6 times and the following conditions. Show your work to the instructor after completing the exercise.

1) Light L1 will turn ON when counter DONE bit close.
2) Light L2 will turn ON when EQU compare instruction is equal 6.
3) Light L3 will turn ON when MEQ compare instruction is equal 6.

Use the following addresses:

S1_I:1/0 to increment the count.
S2_I:1/1 to reset the count.
L1_O:2/0
L2_O:2/1
L3_O:2/2
C5:1 for counter

Exercise 9.02:

Write a documented program that will turn 3 lights ON when you press a pushbutton according to the following conditions. Show your work to the instructor after completing the exercise.

1) Light L1 will turn ON when NEQ Compare instruction not equal 5.
2) Light L2 will turn ON when LES compare instruction is less than 8.
3) Light L3 will turn ON when GRT compare instruction is greater than 10.
4) Light L4 will turn ON when LEQ compare instruction is less or equal to11.
5) Light L5 will turn ON when GEQ compare instruction is Greater or equal to13.

Use the following addresses:

S1_I:1/0 to increment the count.
S2_I:1/1 to reset the count.
L1_O:2/0
L2_O:2/1
L3_O:2/2
L4_O:2/0
L5_O:2/1
C5:1 for counter

This Page is Left Blank

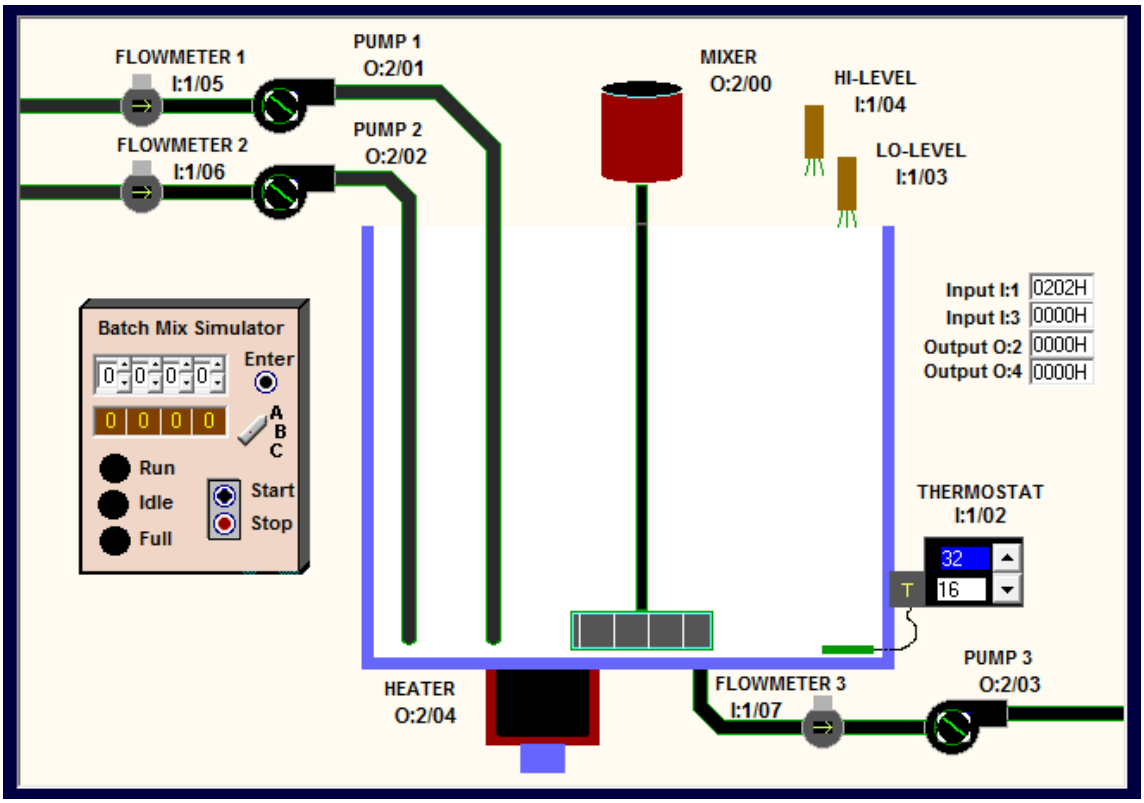
Chapter-10: Industrial Control Projects

Objectives: Upon completion of this chapter, the student should be able to:

1- Convert logic control statement to PLC control program.
2- Collect inputs and outputs needed for industrial control project.
3- Complete batch mixing industrial PLC control project.
4- Complete conveyor belt and product line industrial PLC control project.

10.01 Filling the Batch Mixing Tank Project

From the Simulations Menu at the top of the screen, Select the Batch Mixing Simulation to see the following industrial control exercise.



- Using your knowledge of PLC instructions, design a program to meet the following requirements:

- When the Start switch (I:1/0) is pressed, pump P1 will be energized and the tank will start to fill. The pulses generated by Flowmeter 1 should be used to increment a counter.

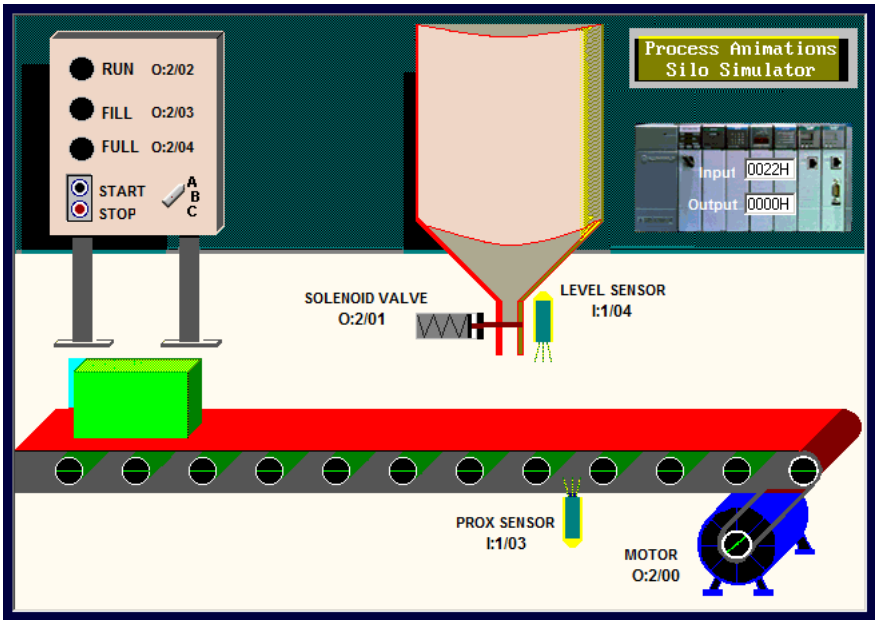
- When the count reaches a value where the tank is approximately 90% full, the pump is to be shut-off and the control panels FULL light is to be energized.

- The filling operation is to halt immediately if the stop switch is pressed.

- While testing, utilize the "Reset Simulation" and the "Reset Timers and Counters" entries in the Simulations menu to re-start your program.

10.02 Conveyor Belt and Product Line Project

From the Simulations Menu at the top of the screen, Select the Silo Simulation



Completely design and de-bug a ladder control circuit which will automatically position and fill the boxes which are continuously sequenced along the conveyor. Ensure that the following details are also met:

- The sequence can be stopped and re-started at any time using the panel mounted Stop and Start switches.
- The RUN light will remain energized as long as the system is operating automatically.
- The RUN light, Conveyor Motor and Solenoid will de-energize whenever the system is halted via the STOP switch.
- The FILL light will be energized while the box is filling.
- The FULL light will energize when the box is full and will remain that way until the box has moved clear of the prox-sensor.

Test your knowledge:

Exercise 10.01:

For the Batch Mixing project add the following:

1) Add a counter to count the number of tank heated and transferred out.
2) Add a timer to measure the time it takes to complete one tank.

Exercise 10.02:

For the Batch Mixing project add the following:

1) Add a counter to count the number of boxes filled .

Chapter-11: PLC and Troubleshooting

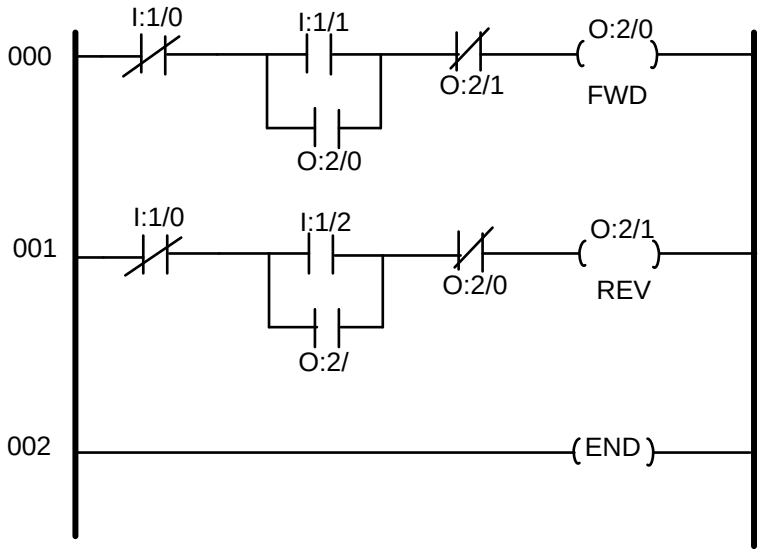
Objectives: Upon completion of this chapter, the student should be able to:

1- Know how the PLC software can find errors.
2- Determine and troubleshoot a software errors.
3- Determine and troubleshoot input hardware connections.
4- Determine and troubleshoot outputs hardware and voltage connections.

Working with PLC requires troubleshooting skills. This can be trouble shooting of the PLC program (software) or troubleshooting the wiring connection (hardware).

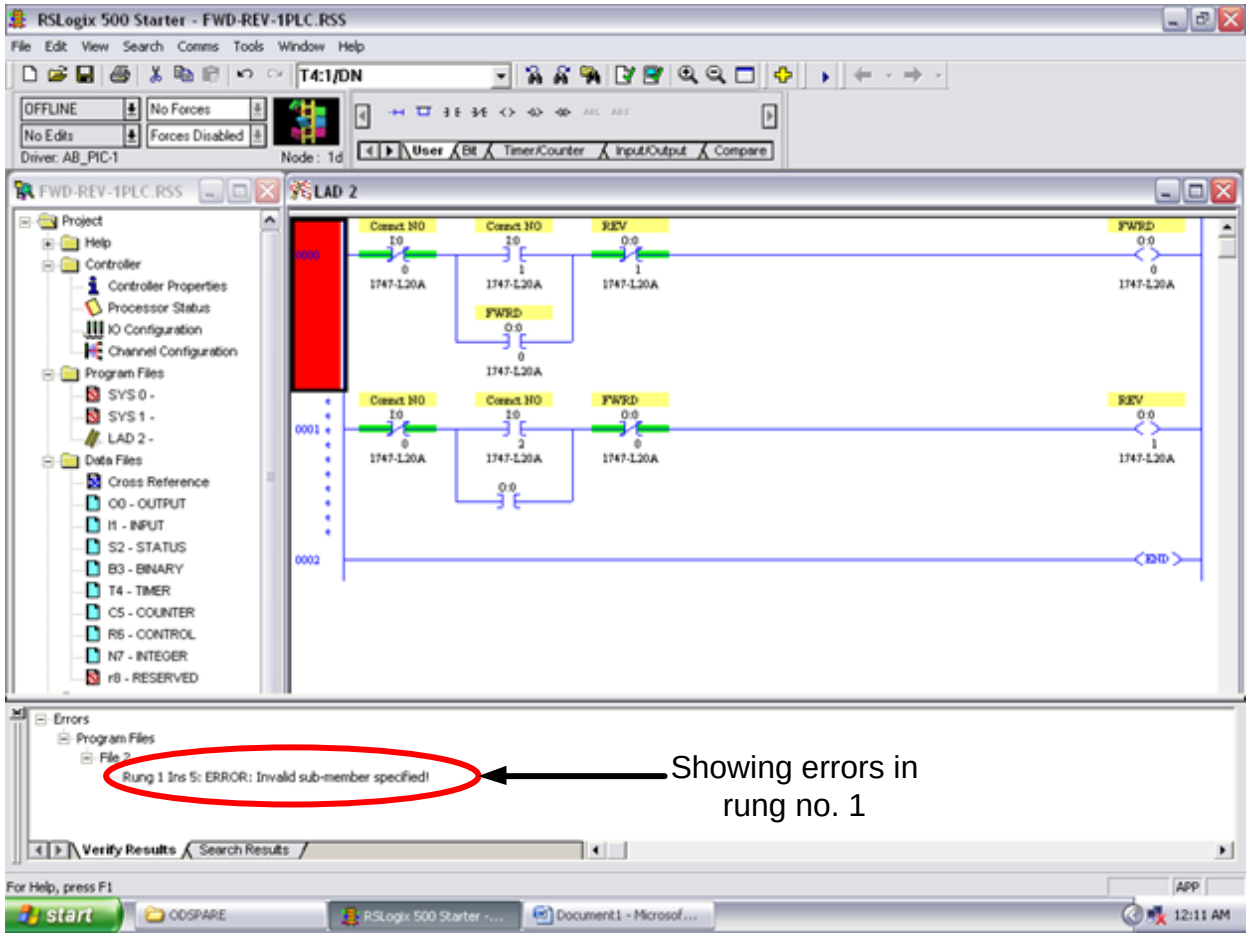
11.01 Troubleshooting PLC Program (software)

After writing the PLC program, we can use the PLC editing program to verify the program to see if there are any errors. Let's try this procedure. Using Rockwell RSLogix software, enter the following PLC program.



Save the program. Verify the project and the file by pressing on the verify icon. You should now see the figure below.

Look at the bottom of the screen you will find errors. The program will not run until you correct the errors.



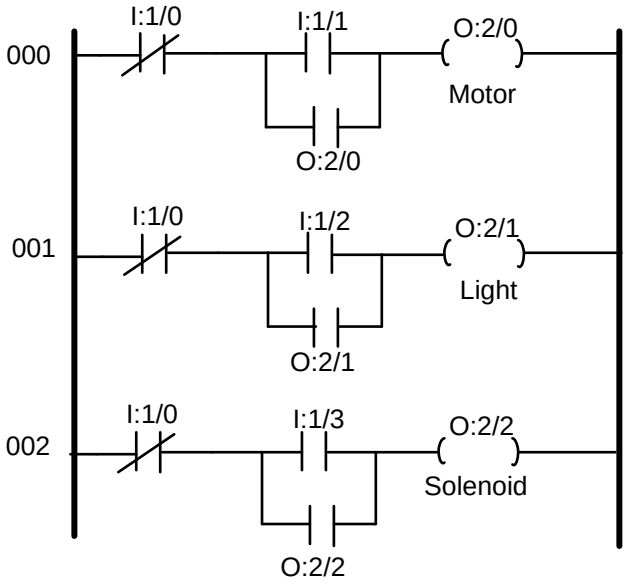
Using your knowledge about instructions and programing, correct the errors and run the program. In this program the error is in the address of the holding contact in rung no. 1.

11.02 Troubleshooting PLC (Hardware)

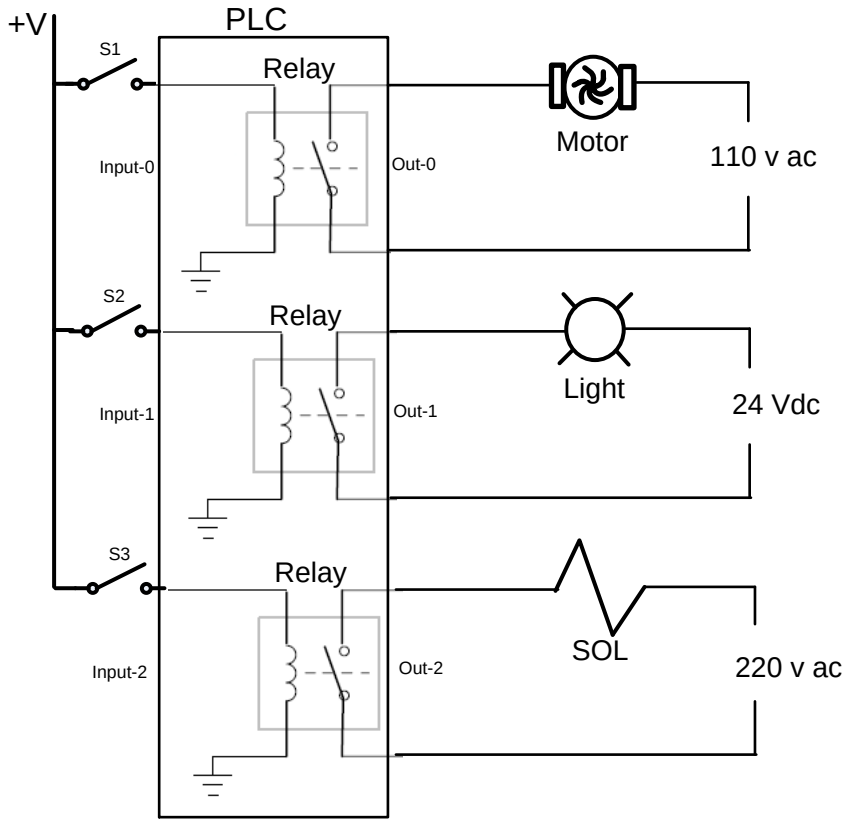
To troubleshoot errors in the PLC wiring connections, you need to use your knowledge in programing, logic, and circuits. We will try now one scenario. Let's say you want to control 3 components: 1) Motor, 2) Light, 3) Solenoid given the following data

1	S1 will control a motor requiring 220 volts ac
2	S2 will control a light requiring 110 volts ac
3	S3 will control a solenoid requiring 24 volts dc

The PLC control program is shown below. Using Rockwell RSLogix software, enter the following PLC program.

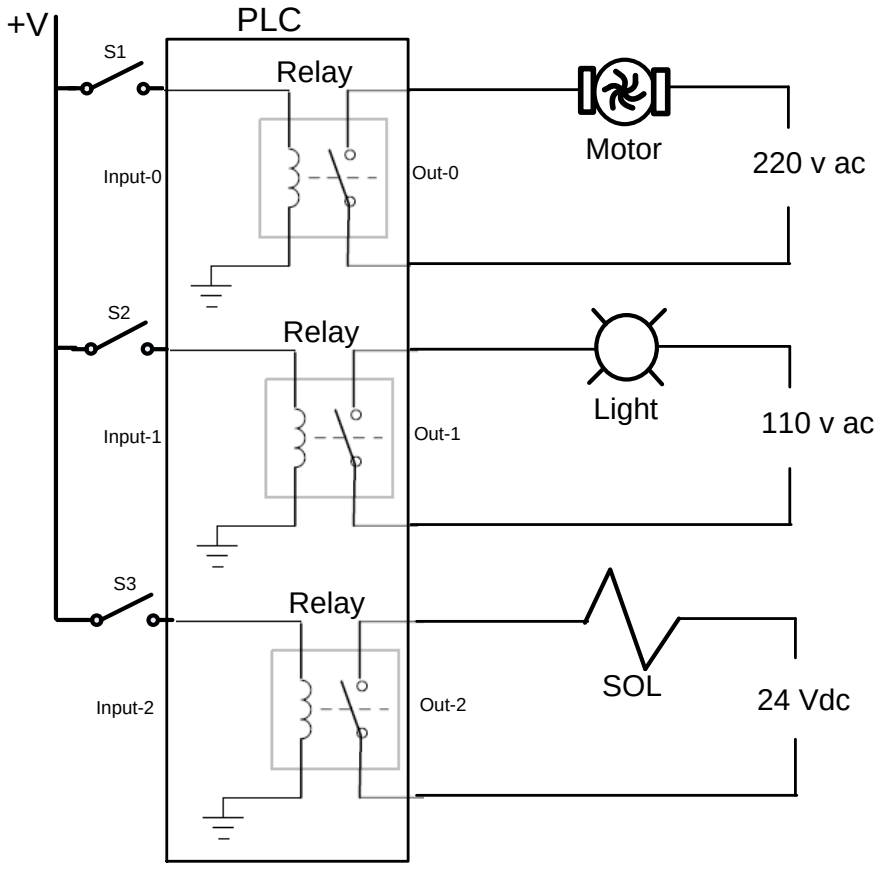


After entering the program, you need to connect the circuit. Let's say you connect the circuit as shown in the diagram below.



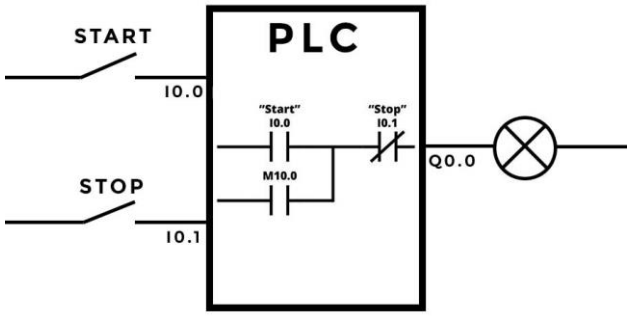
Look at the diagram above and troubleshoot the wiring connection.

The correct wiring is shown below.



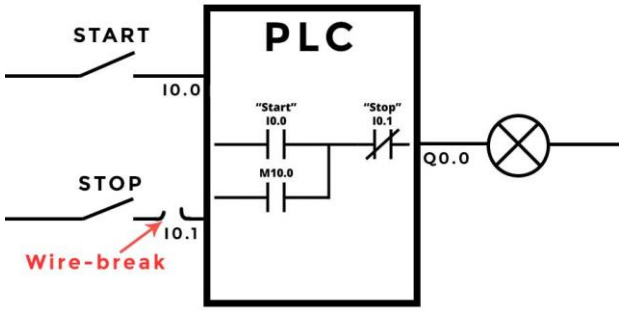
11.03 Fail-Safe (Trouble Preventive Consideration) :

Choosing to use normally open (N.O.) contact or normally closed (N.C.) contact can be critical in some cases. Let's look at the following example where we used N.O. contact for STOP switch.



Using normally open contacts as PLC inputs is good. But for stop functions it can be bad. This is because normally open contacts can create dangerous situations when they fail…

What if one of the wires broke as shown below.



Now, the stop button (the normally open contact) will have no function when the system is failing (wire-break). The wire break is one fail, but that produces another fail: The stop button isn't working. And since the stop button is a critical function, this is why this solution is not good practice.

How could this solution be good practice?

By using a normally closed contact as stop actuator. This is because the normally closed contact as an input actuator won't create dangerous situations under failure. Meaning that when a fail occurs (the wire-break), the input will act as the normally closed contact has been activated. So, if the wire to the stop button breaks, the same will happen as if someone activated the stop button. The latch will break.

At last you might wonder why we didn't do the same thing with the **start** button. That input is also a normally open contact, but with examine if closed (normally open logic) in the software.

Just like that stop button, the start button will not work if the wire breaks. Remember that wire-breaks and other failures shouldn't produce dangerous situations. Well, even though the start button will not work under a wire-break, the start button is not a critical function.

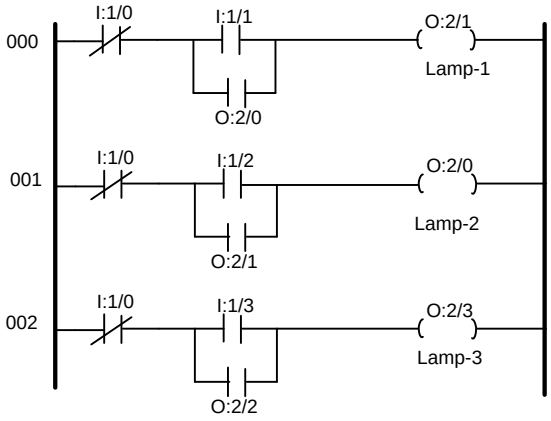
Why is the start function not a critical function?

Because it is not dangerous if the machine, motor or another movable part cannot start.

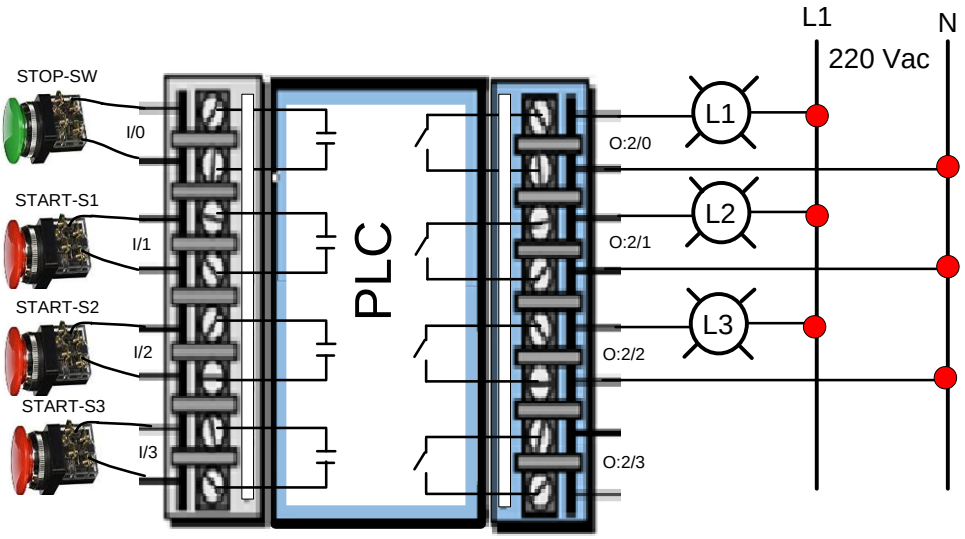
Test your knowledge:

Exercise 11-1:

Write a documented program that will turn 3 lights ON as shown in the plc program below. Enter plc program as is.



Connect the input and output as shown below (as is). Run the program and test it.



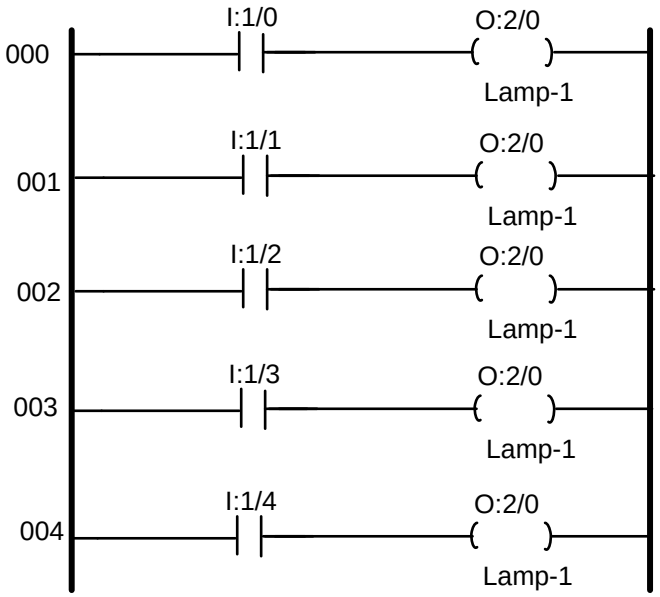
Troubleshoot the problem in the connections and fix it.

What was the problem? _____

How did you fix it? _____

Exercise 11-2:

A student was asked to write a plc program that will turn one light ON by using 5 different switches. He wrote the program as shown below.



Enter the program and test it.

Troubleshoot the problem in the connections and fix it.

What was the problem? _____

Troubleshoot the problem in the connections and fix it.

How did you fix it? _____

This Page is Left Blank

SECTION-2

MicroController

This Page is Left Blank

Chapter-1: Introduction to Microcontroller Architecture

Objectives: Upon completion of this chapter, the student should be able to:

1-Know the history of microcontroller.
2-Understand internal organization of computer.
3-Know the difference between microprocessor and microcontroller.
4-Know applications of microcontroller.

1.01 Introduction to Microcontroller:

A microcontroller (or MCU for microcontroller unit) is a small computer on a single integrated circuit. A microcontroller contains one or more CPUs (processor cores) along with memory and programmable input/output peripherals. Program memory in the form of RAM and ROM. Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, medical instruments, robotics and more.

The first microcontroller 8051 was developed by Intel Corporation in the year 1981. It was called as a "System on a chip". Intel refers to it as MCS-51 now.

The microcontroller incorporates all the features that are found in microprocessor. The microcontroller has built in ROM, RAM, Input Output ports, Serial Port, timers, interrupts and clock circuit.

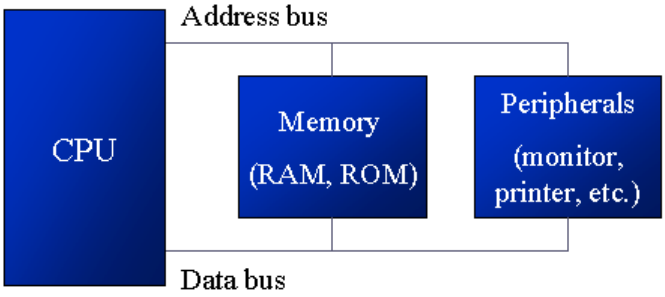
1.02 Computer Terminology

- The unit of data size
  - Bit : a binary digit that can have the value 0 or 1
  - Byte : 8 bits
  - Nibble : half of a bye, or 4 bits
  - Word : two bytes, or 16 bits

- The terms used to describe amounts of memory in IBM PCs and compatibles
  - Kilobyte (K): 1024 bytes = 1k bytes
  - Megabyte (M) : 1024k bytes = 1M bytes, over 1 million
  - Gigabyte (G) : 1024M bytes = 1G bytes, over 1 billion
  - Terabyte (T) : 1024G bytes = 1T Bytes, over 1 trillion

1.03 Internal Organization of Computers

Computer consists of 3 major parts:



1- CPU (Central Processing Unit)
   - Execute information stored in memory
2- I/O (Input/output) devices
   - Provide a means of communicating with CPU
3- Memory
   - **RAM** (Random Access Memory) –temporary storage of programs that computer is running. The data is lost when computer is off.
   - **ROM** (Read Only Memory) – contains programs and information essential to operation of the computer. The information cannot be changed by use, and is not lost when power is off. It is called nonvolatile memory

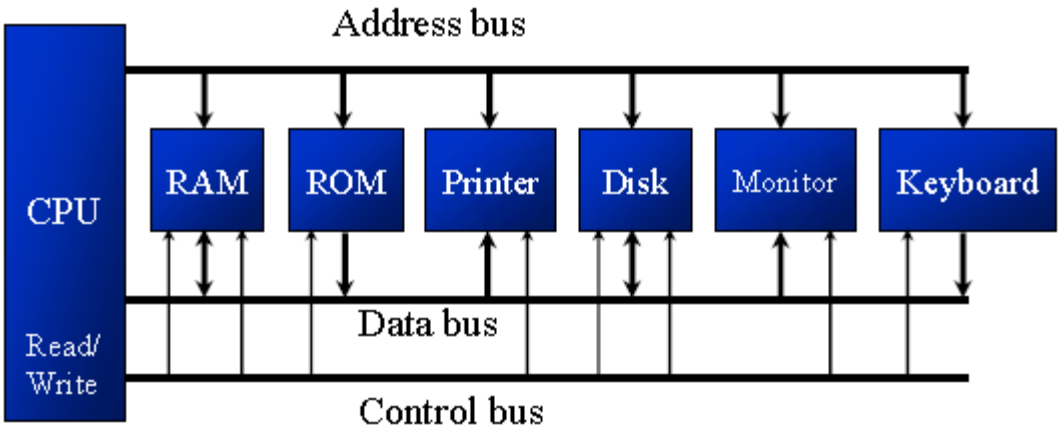
1.04 Busses

The CPU is connected to memory and I/O through strips of wire called a bus. The busses carry information from place to place. There are three types of busses:

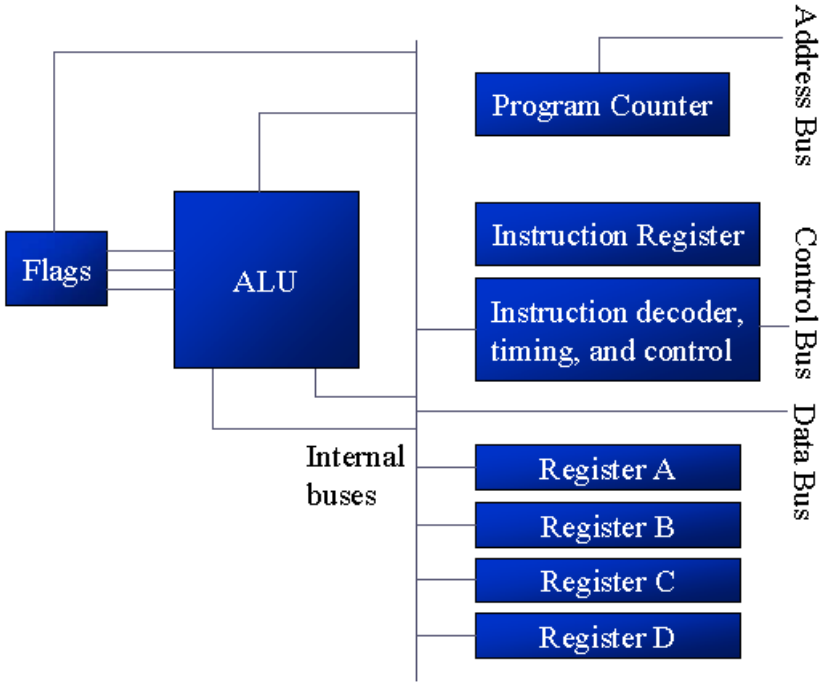
- **Address bus**: For a device (memory or I/O) to be recognized by the CPU, it must be assigned an address. The address assigned to a given device must be unique. The CPU puts the address on the address bus, and the decoding circuitry finds the device.

- **Data bus:** The CPU either gets data from the device or sends data to it**.**

- **Control bus:** Provides read or write signals to the device to indicate if the CPU is asking for information or sending it information.



1.05 Inside CPUs



The CPU consists of:

1- Registers

- The CPU uses registers to store information temporarily
- Address of value to be fetched from memory
- In general, the more and bigger the registers, the better the CPU
- Registers can be 8-, 16-, 32-, or 64-bit
- The disadvantage of more and bigger registers is the increased cost.

2- ALU (arithmetic/logic unit):

- Performs arithmetic functions such as add, subtract, multiply, and divide, and logic functions such as AND, OR, and NOT

3- Program counter:

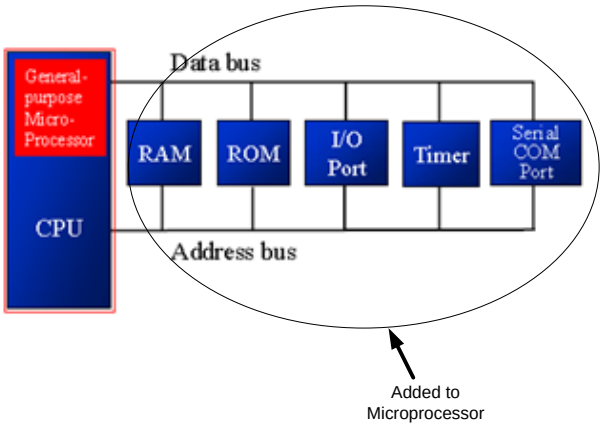
- Points to the address of the next instruction to be executed.
- As each instruction is executed, the program counter is incremented to point to the address of the next instruction to be executed

4- Instruction decoder:

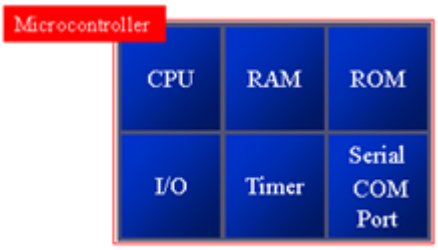
- Interprets the instruction fetched into the CPU.
- A CPU capable of understanding more instructions requires more transistors to design.

1.06 Microcontroller vs. General-Purpose Microprocessor

- General-purpose **microprocessors** has
  - No RAM
  - No ROM
  - No I/O ports



- **Microcontroller** has
  - CPU (microprocessor)
  - RAM
  - ROM
  - I/O ports
  - Timer
  - ADC and other peripherals



- **General-purpose microprocessors:**
  - Must add RAM, ROM, I/O ports, and timers must be added externally to make them functional.
  - Make the system bulkier and much more expensive.
  - Have the advantage of versatility on the amount of RAM, ROM, and I/O ports.

- **Microcontroller:**
  - The fixed amount of on-chip ROM, RAM, and number of I/O ports makes them ideal for many applications in which cost and space are critical.
  - In many applications, the space it takes, the power it consumes, and the price per unit are much more critical considerations than the computing power

1.07 Choosing a Microcontroller:

Choosing a microcontroller depends on the equipment or instruments we need to control, the necessary speed and the amount of data involved. For teaching purpose an eight bits microcontroller is ideal. Below is a list of 8 bit microcontroller.

- Motorola's 6811
- Intel's 8051
- Zilog's Z8

There are also 16-bit and 32-bit microcontrollers made by various chip makers

The microcontroller which we will use in this course is the Intel's 8051. It is an 8 bit microcontroller. Basically 8 bit specifies the size of data bus. 8 bit microcontroller means 8 bit data can travel on the data bus or we can read, write process 8 bit data.

Test your knowledge:

1. How many bytes are in 24 kilobytes?_____

2. What does RAM stand for?

   _____

3. What does ROM stand for?

   _____

4. List three major components of a computer system.

   _____

5. What does CPU stand for?

   _____

6. List three types of busses found in computer system.

   _____

7. A microcontroller normally has which of the following devices on-chip?

   a) RAM
   b) ROM
   c) I/O
   d) All the above

8. A general purpose microprocessor normally needs which of the following devices to be attached to it?

   e) RAM
   f) ROM
   g) I/O
   h) All the above

This Page is Left Blank

Chapter-2: Intel 8051 Microcontroller Overview

Objectives: Upon completion of this chapter, the student should be able to:

1- Describe Features of 8051 Microcontroller.
2- Know Architecture of 8051 Microcontroller.
3- Understand Pin configuration/diagram of 8051 Microcontroller.

2.01 Overview

The 8051 is a high performance single chip computer intended for use in sophisticated real time applications such as instrumentation, industrial control and computer peripherals. It provides extra features like interrupts, bit address ability and an enhanced set of instructions, which makes the chip very powerful and cost effective.

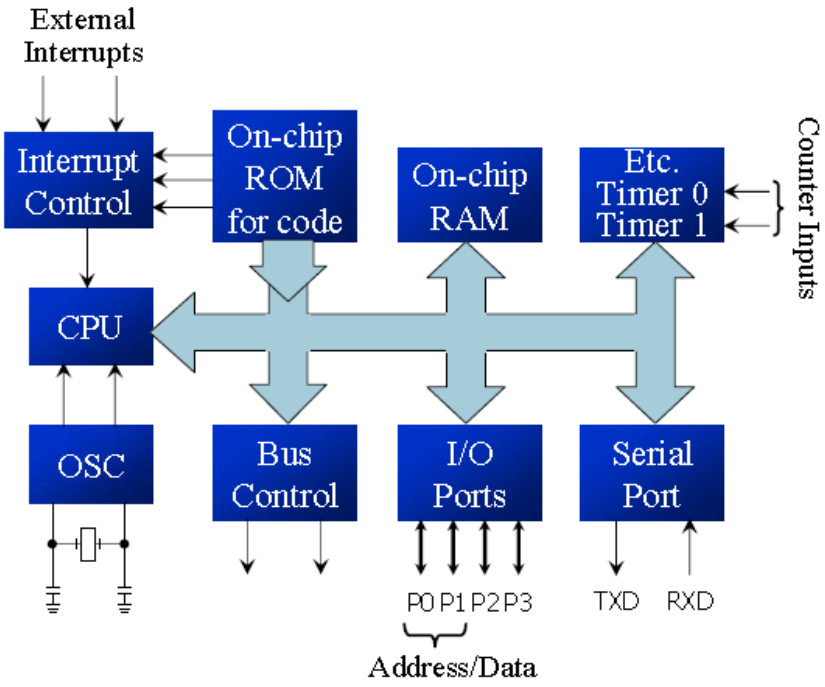
2.02 Standard features of the 8051

- The 8051 is an 8-bit processor
  - The CPU can work on only 8 bits of data at a time

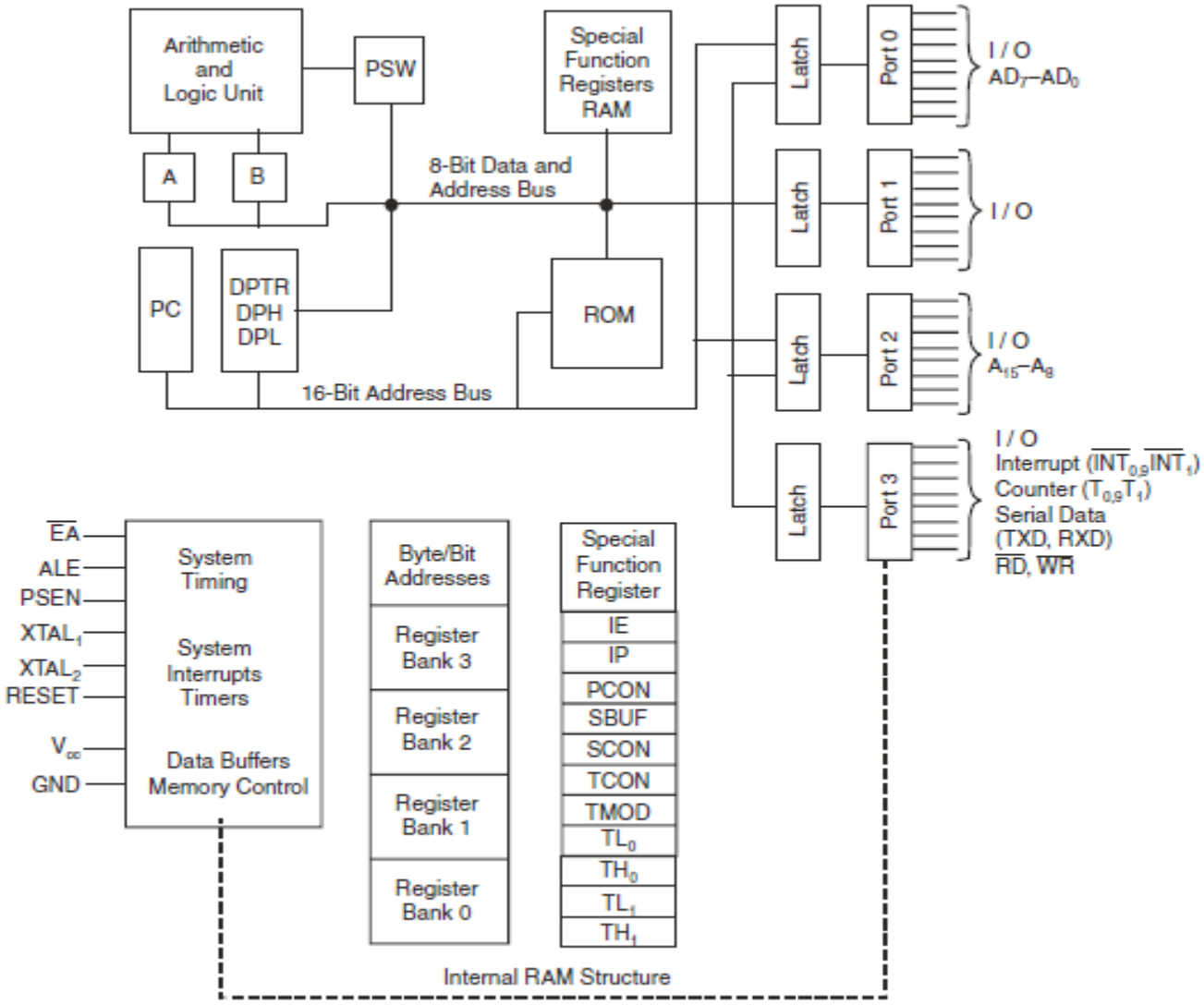
- The 8051 has
  - 128 bytes of RAM
  - 4K bytes of on-chip ROM
  - Two timers
  - One serial port
  - Four I/O ports, each 8 bits wide
  - 6 interrupt sources

2.03 ARCHITECTURE OF 8051

The diagram below shows major components of the Intel 8051 microcontroller

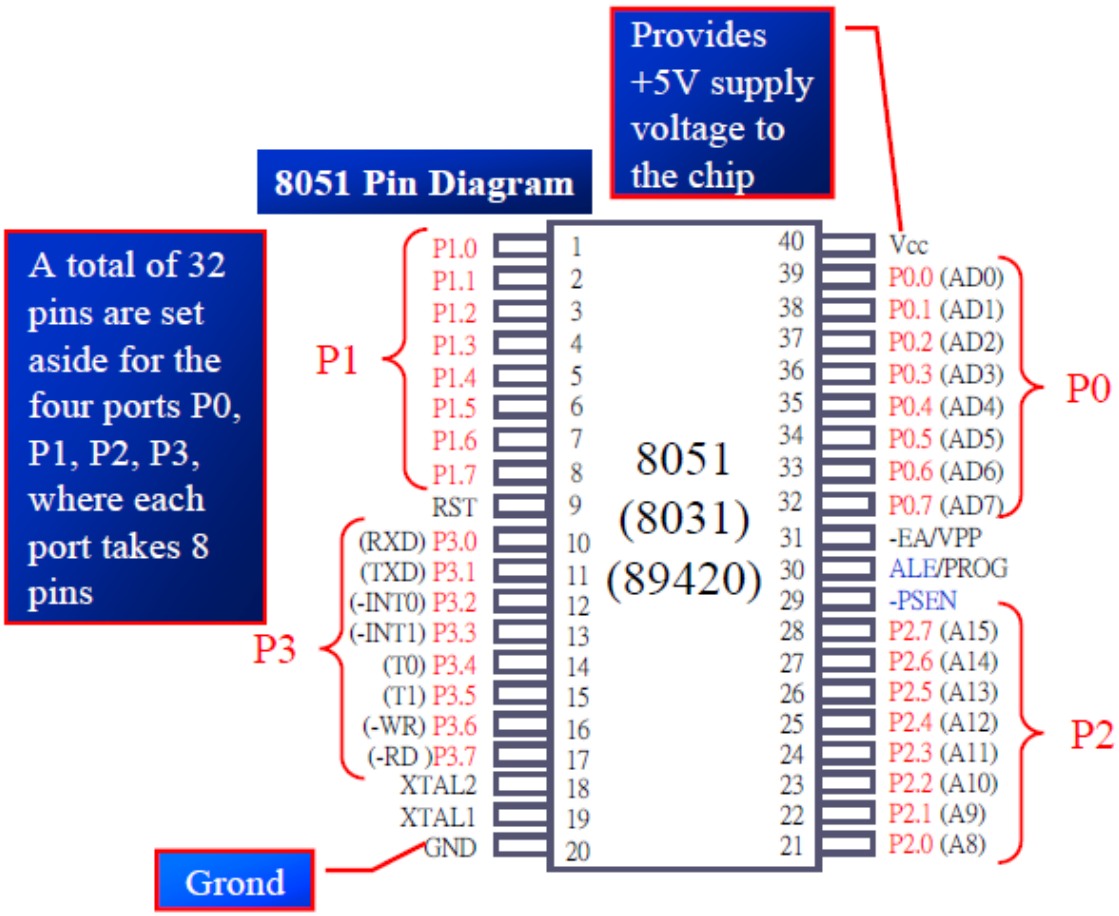


The diagram below shows more details of components inside the Intel 8051 microcontroller



2.04 PIN DIAGRAM OF 8051

The diagram below shows more the pin diagram of the Intel 8051 microcontroller. It has 40 pins.



2.04.01 Description of each pin is discussed here:

- VCC → 5V supply
- GND → Ground
- XTAL2/XTAL1 are for oscillator input
- Port 0 – 32 to 39 – AD0/AD7 and P0.0 to P0.7
- Port 1 – 1 to 8 – P1.0 to P1.7
- Port 2 – 21 to 28 – P2.0 to P2.7 and A 8 to A15
- Port 3 – 10 to 17 – P3.0 to P3.7

• P 3.0 – RXD – Serial data input – SBUF
• P 3.1 – TXD – Serial data output – SBUF
• P 3.2 – INT0 – External interrupt 0 – TCON 0.1
• P 3.3 – INT1 – External interrupt 1 – TCON 0.3
• P 3.4 – T0 – External timer 0 input – TMOD
• P 3.5 – T1 – External timer 1 input – TMOD
• P 3.6 –WR – External memory write cycle – Active LOW
• P 3.7 – RD – External memory read cycle – Active LOW
• RST – for Restarting 8051
• ALE – Address latch enable
1 – Address on AD 0 to AD 7
0 – Data on AD 0 to AD 7
• PSEN – Program store enable

Test your knowledge:

1. Name three features of the 8051.

   _____

   _____

   _____

2. What is the size of the on-chip ROM in 8051?

   _____

3. True or False, A general purpose microprocessor has on-chip ROM? ( )

4. True or False, A microcontroller has on-chip ROM? ( )

5. True or False, A microcontroller has on-chip I/O ports? ( )

6. What components are normally put together with the microcontroller into a single chip?

   _____

   _____

Chapter-3: Assembly Language Programing

Objectives: Upon completion of this chapter, the student should be able to:

1- Know the types and structure of 8051 Registers.
2- Understand structure of assembly language.
3- Use different instructions of assembly language programming.
4- Understand how to program and use I/O Ports as an Input or Output.

3.01 Inside the 8051

Registers

- Register are used to store information temporarily, while the information could be
  - a byte of data to be processed, or
  - an address pointing to the data to be fetched

- The vast majority of 8051 register are 8-bit registers
  - There is only one data type, 8 bits

- The 8 bits of a register are shown from MSB D7 to the LSB D0

- With an 8-bit data type, any data larger than 8 bits must be broken into 8-bit chunks before it is processed.

8 bit Registers

- The most widely used registers:

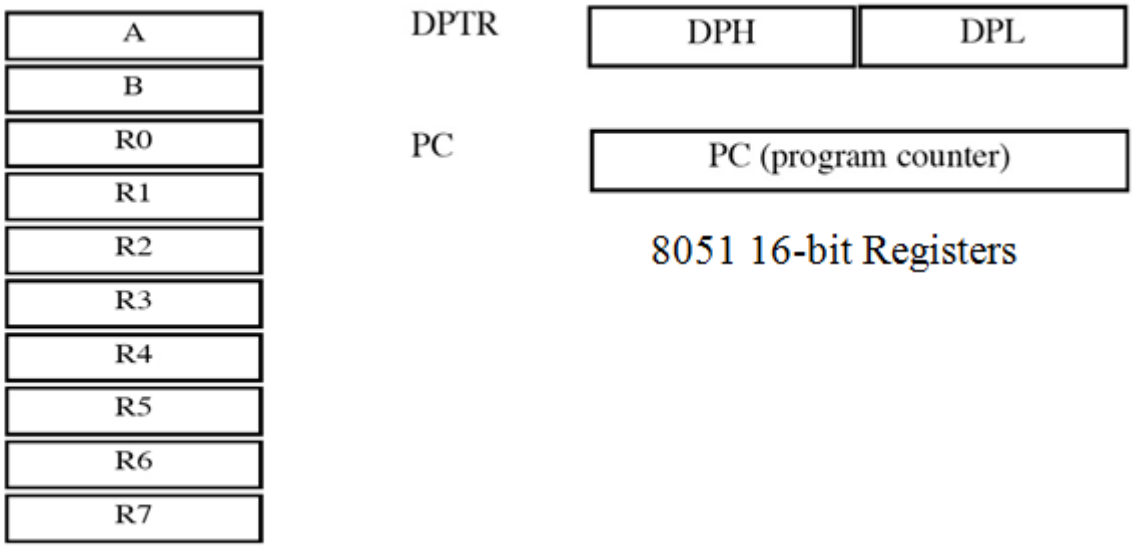
  - A (Accumulator): As its name suggests, is used as a general register to accumulate the results of a large number of instructions. It can hold an 8-bit (1-byte) value and is specifically used for all arithmetic and logic instructions.

  - B Register: The "B" register is very similar to the Accumulator in the sense that it may hold an 8-bit (1-byte) value.

  - The R Registers: The "R" registers are a set of eight registers these "R" registers are numbered from 0 through 7 (R0, R1, R2, R3, R4, R5, R6, and R7). These registers are used as auxiliary registers in many operations.

  - DPTR (data pointer): This is 16-bits (2-Bytes) register. The Data Pointer DPTR, as the name suggests, is used to point to data. It is used by a number of commands which allow the 8051 to access the correct memory.

  - PC (program counter): program counter tells the 8051 where the next instruction to execute is found in memory. When the 8051 is initialized PC always starts at 0000h and is incremented each time an instruction is executed.



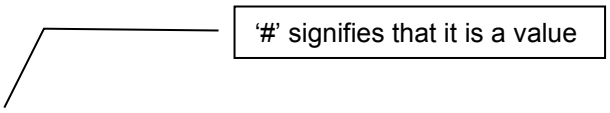
8-bit Registers of the 8051

8051 16-bit Registers

3.02 MOV Instruction:

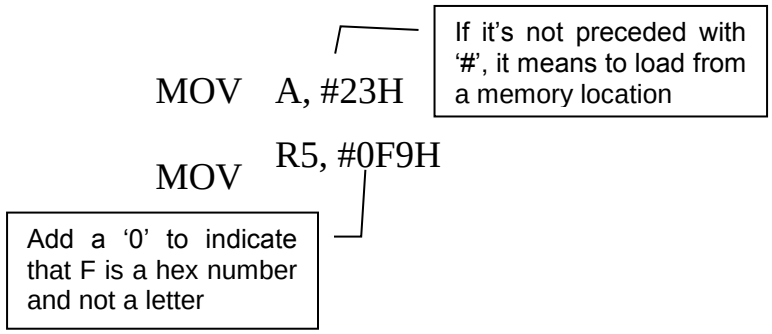
MOV destination, source ; copy source to destination.

The instruction tells the CPU to move (in reality, COPY) the source operand to the destination operand




```
MOV   A, #55H     ; Load value 55H into reg. A
MOV   R0, A       ; copy contents of A into R0,  (now A=R0=55H)
MOV   R1, A       ; copy contents of A into R1,  (now A=R0=R1=55H)
MOV   R2, A       ; copy contents of A into R2, (now A=R0=R1=R2=55H)
MOV   R3, #95H    ; load value 95H into R3, (now R3=95H)
MOV   A, R3       ; copy contents of R3 into A,  now A=R3=95H
```

- Notes on programming
  - Value (preceded with #) can be loaded directly to registers A, B, or R0 – R7.



  - If values 0 to F moved into an 8-bit register, the rest of the bits are assumed all zeros.

```
MOV   A, #5H      ; the result will be A=05
                  ; i.e., A = 00000101 in binary
```

  - Moving a value that is too large into a register will cause an error.

```
MOV   A, #7F2H    ; ILLEGAL: 7F2H> 8 bits (FFH)
                  ; i.e., A = 00000101 in binary
```

3.03 ADD Instruction

ADD A, source ; ADD the source operand to the Accumulator

- The ADD instruction tells the CPU to add the source byte to register A and put the result in register A.
- Source operand can be either a register or immediate data, but the destination must always be register A.
- "ADD R4, A" and "ADD R2, #12H" are invalid since A must be the destination of any arithmetic operation

MOV	A, #25H	; load 25H into A
MOV	R2, #34H	; load 34H into R2
ADD	A, R2	; add R2 to accumulator (A = A + R2)

There are always many ways to write the same program, depending on the registers used

```
MOV     A, #25H     ; load one operand
                    ; into A (A = 25H)
ADD     A, #34H     ; add the second
                    ; operand 34H to A
```

Test your knowledge:

1. Write the instruction to move value 34H into register A and value 3FH into register B, then add them together.

2. Write the instruction to add the values 16H and CDH. Place the results in register R2.

3. True or False, No values can be moved directly into registers R0 – R7. ( )

4. The vast majority of registers in 8051 are _____ bits.

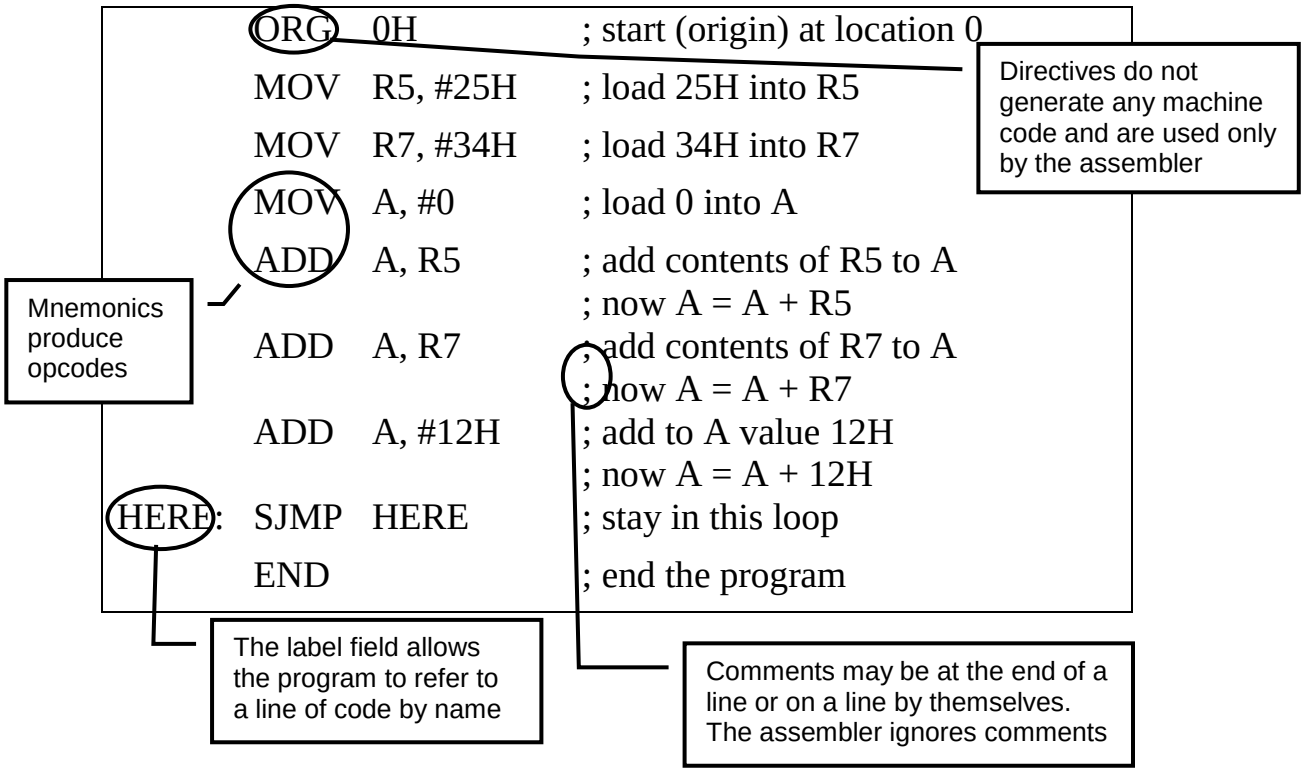
3.04 Structure of Assembly Language

- Assembly language instruction includes
  - **A mnemonic** (abbreviation easy to remember) the commands to the CPU, telling it what to do.
  - **Operands:** the data items being manipulated

- A given Assembly language program is a series of statements, or lines which are either:
  - **Assembly language instructions** such as ADD and MOV to tell the CPU what to do or
  - **Directives** such as ORG and END that gives directions to the assembler

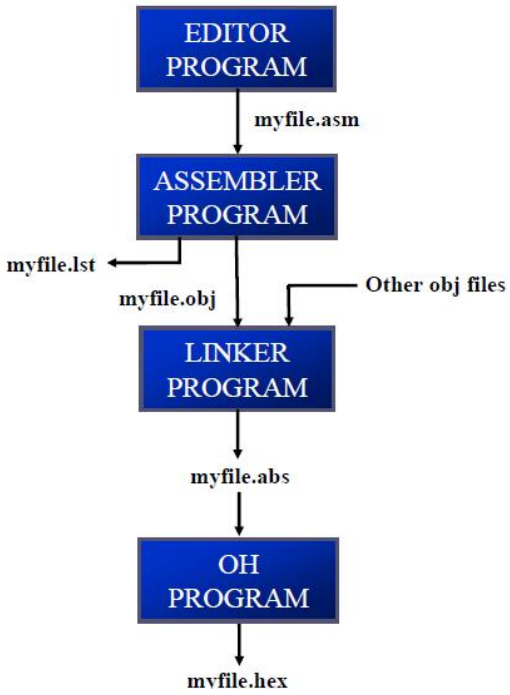
- An Assembly language instruction consists of four fields:

[label:] Mnemonic [operands] [; comment]



3.05 Assembling and Running an 8051 Program

Steps to create an assembly language program are outlined as follows:



1. First we use an editor to type a program such as MS-DOS Edit program or Notepad in Windows. The source file has an extension "asm" or "src", depending on which assembler you are using.

2. The "asm" source file containing the program code created in step 1 is fed to an 8051 assembler to convert the instruction into machine code.

3. Assembler require a third step called linking

- The linker program takes one or more object code files and produce an absolute object file with the extension "abs".
- This "abs" file is used by 8051 trainers that have a monitor program.

4. Next the "abs" file is fed into a program called "OH" (object to hex converter) which creates a file with extension "hex" that is ready to burn into ROM

- This program comes with all 8051 assemblers
- Recent Windows-based assemblers combine step 2 through 4 into one step.

Test your knowledge:

1. What is the purpose of the directives?

   _____

2. True or False, Generally, the extension of the source file is “asm” or “src”?

3. Which of the following files can be produced by Notepad in windows?

a) Myprog.asm
b) Myprog.obj
c) Myprog.exe
d) Myprog.lst

3.06 Program Counter and ROM Space

3.06.01 Program Counter

The program counter points to the address of the next instruction to be executed. As the CPU fetches the opcode from the program ROM, the program counter is increasing to point to the next instruction.

The program counter is 16 bits wide .This means that it can access program addresses 0000 to FFFFH, a total of 64K bytes of code

3.06.02 Power Up

All 8051 members start at memory address 0000 when they’re powered up.

- Program Counter has the value of 0000.
- The first opcode is burned into ROM address 0000H, since this is where the 8051 looks for the first instruction when it is booted.
- We achieve this by the ORG statement in the source program.

3.06.03 Placing Code in ROM

Examine the list file and how the code is placed in ROM

1	0000			ORG	0H	;start (origin) at 0
2	0000	7D25		MOV	R5, #25H	;load 25H into R5
3	0002	7F34		MOV	R7, #34H	;load 34H into R7
4	0004	7400		MOV	A, #0	;load 0 into A
5	0006	2D		ADD	A, R5	;add contents of R5 to A ;now A = A + R5
6	0007	2F		ADD	A, R7	;add contents of R7 to A ;now A = A + R7
7	0008	2412		ADD	A, #12H	;add to A value 12H ;now A = A + 12H
8	000A	80EF	HERE:	SJMP	HERE	;stay in this loop
9	000C			END		

ROM Address	*Machine Language*	*Assembly Language*
0000	7D25	MOV R5, #25H
0002	7F34	MOV R7, #34H
0004	7400	MOV A, #0
0006	2D	ADD A, R5
0007	2F	ADD A, R7
0008	2412	ADD A, #12H
000A	80EF	HERE: SJMP HERE

After the program is burned into ROM, the opcode and operand are placed in ROM memory location starting at 0000.

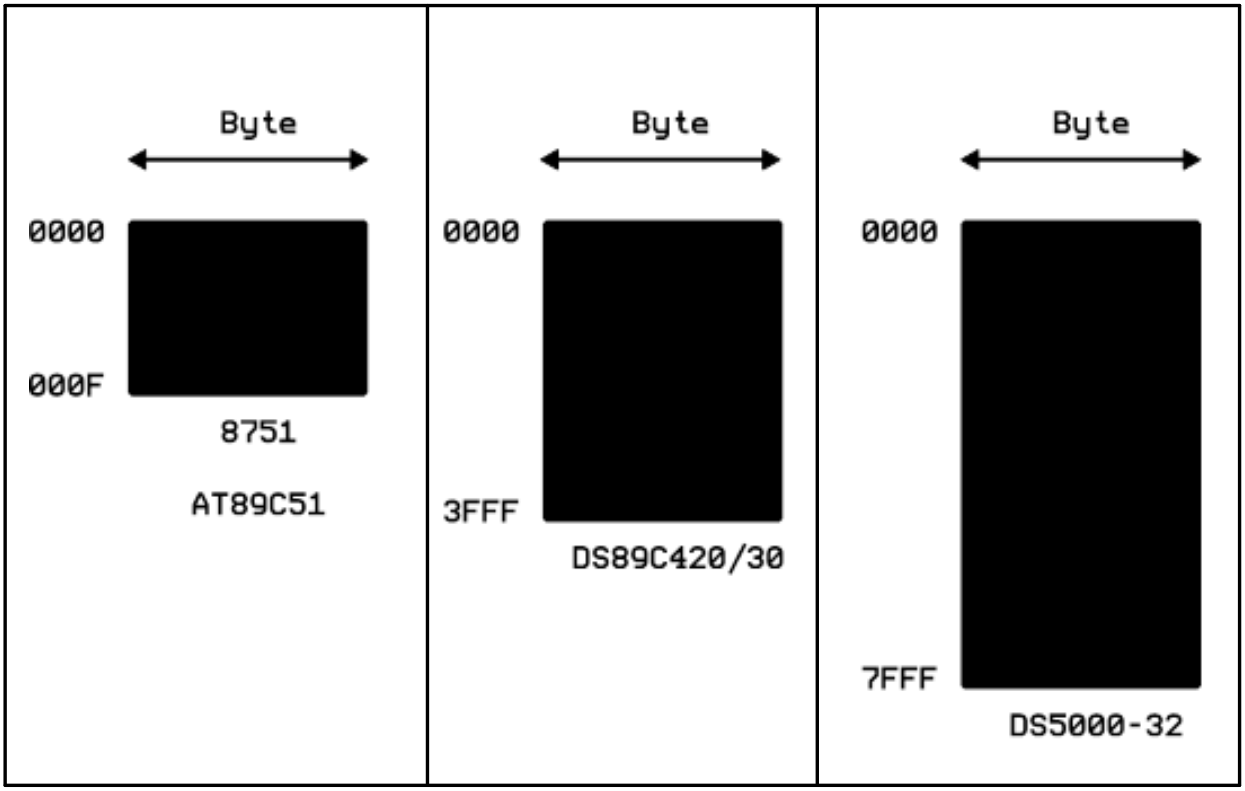
ROM contents

Address	*Code*
0000	7D
0001	25
0002	7F
0003	34
0004	74
0005	00

Address	*Code*
0006	2D
0007	2F
0008	24
0009	12
000A	80
000B	FE

3.06.04 ROM Memory Map in 8051 Family

The diagram below shows the ROM memory address maps for 3 different microcontrollers of the 8051 family.



Test your knowledge:

1. In the 8051, the program counter is _____ bits wide?

2. The instruction “MOV A, #44” is a ___-byte instruction?

3.07 Assembler Directives

The following are some more widely used directives of the 8051

ORG (origin)

- The ORG directive is used to indicate the beginning of the address
- The number that comes after ORG can be either in hex or in decimal
- If the number is not followed by H, it is decimal and the assembler will convert it to hex.

END

- This indicates to the assembler the end of the source (asm) file.
- The END directive is the last line of an 8051 program.
- Anything in the code after the END directive is ignored by the assembler.

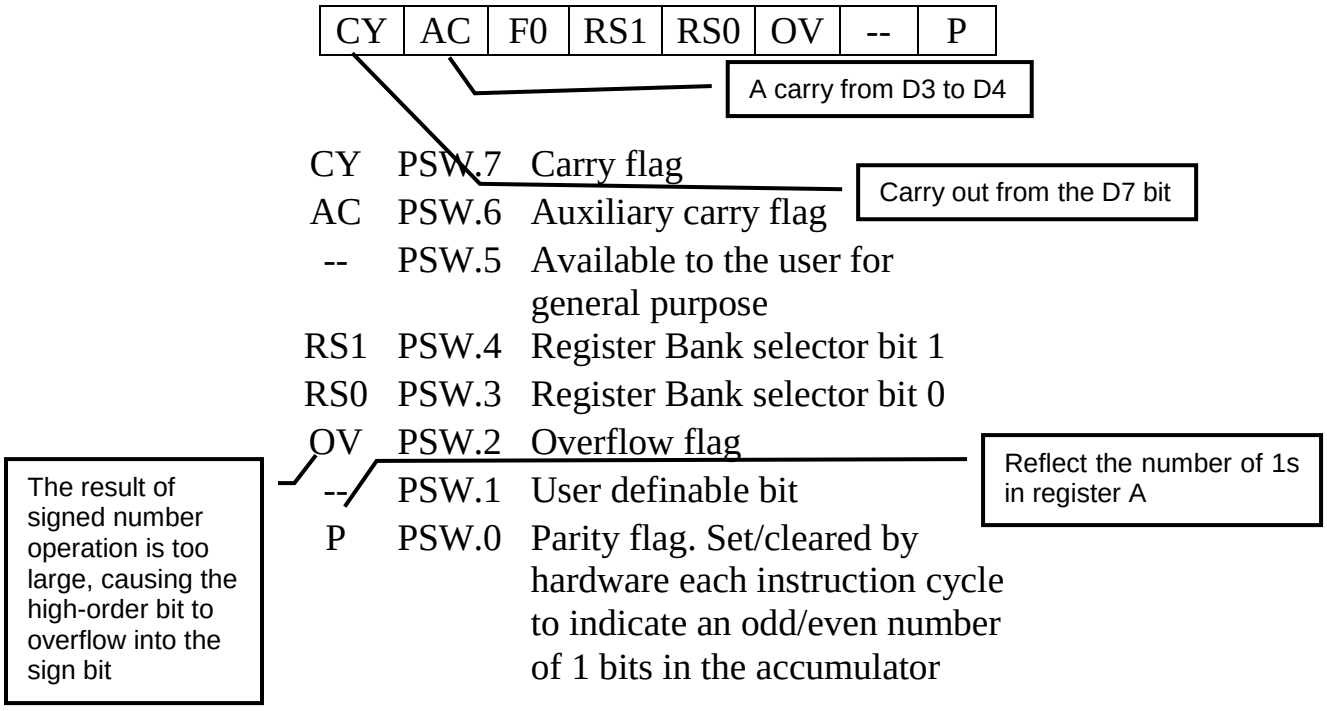
EQU (Equate)

- This is used to define a constant without occupying a memory location.
- The EQU directive does not set aside storage for a data item but associates a constant value with a data label.
- When the label appears in the program, its constant value will be substituted for the label.

3.08 Flag Bits and PSW Register

The program status word (PSW) register, also referred to as the flag register is an 8 bit register but only 6 bits are used.

- CY (carry flag), AC (auxiliary carry flag), P (parity flag), and OV (overflow flag) are called conditional flags, meaning that they indicate some conditions that resulted after an instruction was executed.
- The PSW3 and PSW4 are designed as RS0 and RS1, and are used to change the bank.
- The two unused bits PSW1 and PSW 5 are user-definable.



RS1	RS0	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

Example 3.1:

Show the status of the CY, AC and P flag after the addition of 38H and 2FH in the following instructions.

```
MOV   A, #38H
ADD   A, #2FH      ;after the addition A=67H, CY=0
```

Solution:

```
    38   0 0 1 1 1 0 0 0
+   2F   0 0 1 0 1 1 1 1
    67   0 1 1 0 0 1 1 1
```

CY = 0 since there is no carry beyond the D7 bit
AC = 1 since there is a carry from the D3 to the D4 bit
P = 1 since the accumulator has an odd number of 1s (it has five 1s)

Example 3.2:

Show the status of the CY, AC and P flags after the addition of 9CH and 64H in the following instructions:

```
MOV   A, #9CH
ADD   A, #64H      ;after the addition A=00H, CY=1
```

Solution:

```
    9C   1 0 0 1 1 1 0 0
+   64   0 1 1 0 0 1 0 0
   100   0 0 0 0 0 0 0 0
```

CY = 1 since there is a carry beyond the D7 bit
AC = 1 since there is a carry from the D3 to the D4 bit
P = 0 since the accumulator has an even number of 1s (it has zero 1s)

Test your knowledge:

1. The flag register in 8051 is also called _____.
2. What is the size of the flag register in the 8051?
3. Which bits of the PSW register are user-definable?
4. Find the CY and AC flag bits for the following code:

```
MOV   A, #0FFH
ADD   A, #01
```

5. The instruction "MOV A, #44" is a ___-byte instruction?

3.09 JUMP, LOOP and CALL Instructions

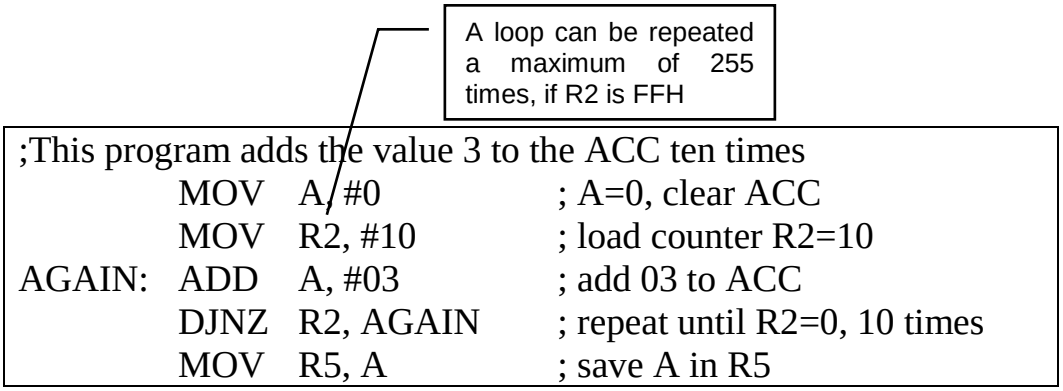
3.09.01 LOOP and JUMP Instructions

Looping in 8051
Repeating a sequence of instructions a certain number of times is called **a loop**
In the 8051, the loop action is performed by the instruction JUMP.

DJNZ reg, Label

This is a JUMP if NOT ZERO instruction. In this instruction, the register is decremented, If it is not zero, it jumps to the target address referred to by the label. Prior to the start of loop the register is loaded with the counter for the number of repetitions. Counter can be R0 – R7 or RAM location.

Example 3.3:




```
;This program adds the value 3 to the ACC ten times
        MOV   A, #0          ; A=0, clear ACC
        MOV   R2, #10        ; load counter R2=10
AGAIN:  ADD   A, #03         ; add 03 to ACC
        DJNZ  R2, AGAIN      ; repeat until R2=0, 10 times
        MOV   R5, A          ; save A in R5
```

What is the maximum number of times that the loop in the example above can be repeated? Answer is 10 times.

Since R2 holds the count and the R2 is an 8-bit register, it can hold a maximum of FFH (255 Decimal), therefore, the loop can be repeated a maximum of 256 times.

3.09.02 Nested Loop

If we want to repeat an action more times than 256, we use a loop inside a loop, which is called **nested loop. To do that** we use two registers to hold the count.

Write a program to (a) load the accumulator with the value 55H, and (b) complement the ACC 700 times

```
        MOV   A, #55H      ;A=55H
        MOV   R3, #10      ;R3=10, outer loop count
NEXT:   MOV   R2, #70      ;R2=70, inner loop count
AGAIN:  CPL   A            ;complement register A
        DJNZ  R2, AGAIN    ;repeat it 70 times
        DJNZ  R3, NEXT
```

3.09.03 Conditional Jumps (Jump only if a certain condition is met)

JZ label ;jump if A=0

```
        MOV   A, R0        ;A=R0
        JZ    OVER         ;jump if A = 0
        MOV   A, R1        ;A = R1
        JZ    OVER         ;jump if A = 0
        ...
OVER:
```

Can be used only for register A. Not any other register

Determine if R5 contains the value 0. If so replace it with 55H

```
        MOV   A, R5        ;copy R5 to A
        JNZ   NEXT         ;jump if A is not zero
        MOV   R5, #55H
NEXT:   ...
```

JNC label ; jump if no carry, CY=0

- ✓ If CY = 0, the CPU starts to fetch and execute instruction from the address of the label.
- ✓ If CY = 1, it will not jump but will execute the next instruction below JNC.

Find the sum of the values 79H, F5H, E2H. Put the sum in registers R0 (low byte) and R5 (high byte).

```
                        MOV R5, #0
        MOV   A, #0     ;A=0
        MOV   R5, A     ;clear R5
        ADD   A, #79H   ;A=0+79H=79H
        JNC   N_1       ;if CY=0, add next number
        INC   R5        ;if CY=1, increment R5
N_1:    ADD   A, #0F5H  ;A=79+F5=6E and CY=1
        JNC   N_2       ;jump if CY=0
        INC   R5        ;if CY=1, increment R5 (R5=1)
N_2:    ADD   A, #0E2H  ;A=6E+E2=50 AND CY=1
        JNC   OVER      ;jump if CY=0
        INC   R5        ;if CY=1, increment 5
OVER:   MOV   R0, A     ;now R0=50H and R5=02
```

8051 conditional jump instructions	
JZ	Jump if A = 0
JNZ	Jump if A ≠ 0
DJNZ	Decrement and Jump if A ≠ 0
CJNE A, byte	Jump if A ≠ byte
CJNE reg, #data	Jump if byte ≠ #data
JC	Jump if CY = 1
JNC	Jump if CY = 0
JB	Jump if bit = 1
JNB	Jump if bit = 0
JBC	Jump if bit = 1 and clear bit

Unconditional Jumps

The unconditional jump is a jump in which control is transferred unconditionally to the target location.

Call Instructions

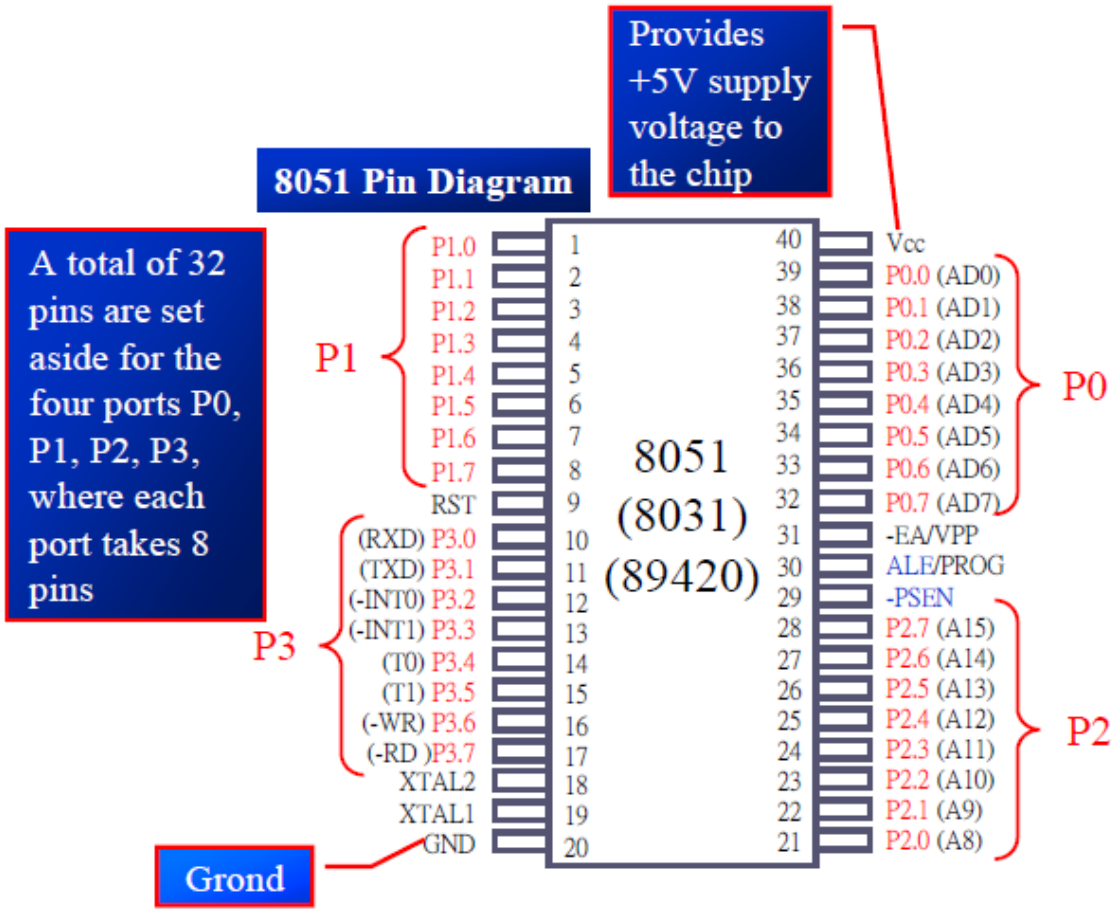
The Call instruction is used to call subroutine. Subroutines are often used to perform tasks that need to be performed frequently. This makes a program more structured in addition to saving memory space.

Example 3.4:

```
        ORG   0
BACK:   MOV   A, #55H          ;load A with 55H
        MOV   P1, A            ;send 55H to port 1
        CALL  DELAY            ;time delay
        MOV   A, #0AAH         ;load A with AA (in hex)
        MOV   P1, A            ;send AAH to port 1
        CALL  DELAY
        SJMP  BACK             ;keep doing this indefinitely
;------- this is the delay subroutine -------
        ORG   300H             ;put DELAY at address 300H
DELAY:  MOV   R5, #0FFH        ;R5=255 (FF in hex), counter
AGAIN:  DJNZ  R5, AGAIN        ;stay here until R5 becomes 0
        RET                    ;return to caller (when R5=0)
        END
```

3.10 8051 I/O Port Programming

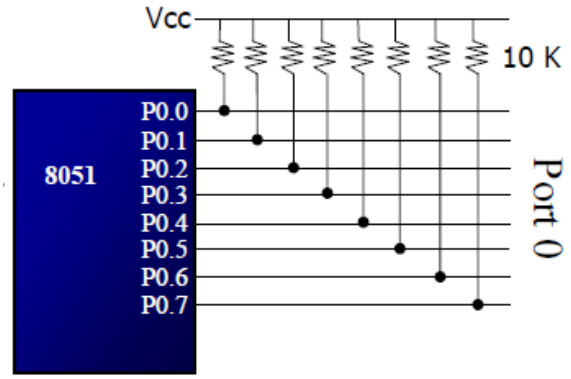
3.10.01 I/O Port Pins



- The four 8-bit I/O ports P0, P1, P2 and P3 each uses 8 pins.
- All the ports upon RESET are configured as input, ready to be used as input ports. When the first 0 is written to a port, it becomes an output.
- To reconfigure it as an input, a 1 must be sent to the port.
- To use any of these ports as an input port, it must be programmed.

3.10.02 Port 0

It can be used for input or output, each pin must be connected externally to a 10K ohm pull-up resistor.



The following code will continuously send out to port 0 the alternating value 55H and AAH

```
BACK:   MOV     A, #55H
        MOV     P0, A
        ACALL   DELAY
        MOV     A, #0AAH
        MOV     P0, A
        ACALL   DELAY
        SJMP    BACK
```

Port 0 as Input

In order to make port 0 an input port, the port must be programmed by writing 1 to all the bits.

In the following example Port 0 is configured first as an input port by writing 1s to it, and then data is received from that port and sent to P1.

```
        MOV   A, #FFH     ;A=FF hex
        MOV   P0, A       ;make P0 an i/p port by writing it all 1s
BACK:   MOV   A, P0       ;get data from P0
        MOV   P1, A       ;send it to port 1
        SJMP  BACK        ;keep doing it
```

3.10.03 Port 1

Port 1 can be used as input or output. In contrast to port 0, this port does not need any pull-up resistors since it already has pull-up resistors internally. Upon reset, port 1 is configured as an input port.

The following code will continuously send out to port 1 the alternating value 55H and AAH.

```
        MOV     A, #55H
BACK:   MOV     P1, A
        ACALL   DELAY
        CPL     A
        SJMP    BACK
```

Port 1 as Input

To make port 1 an input port, it must be programmed as such by writing 1 to all its bits.

```
MOV     A, #0FFH    ;A=FF hex
MOV     P1, A       ;make P1 an input port
                    ;by writing it all 1s
MOV     A, P1       ;get data from P1
MOV     R7, A       ;save it into R7
ACALL   DELAY       ;wait
MOV     A, P1       ;another data from P1
MOV     R5, A       ;save it into R5
```

3.10.04 Port 2

Port 2 can be used as input or output. Just like P1, port 2 does not need any pull-up resistors since it already has pull-up resistors internally. Upon reset, port 2 is configured as an input port.

Port 2 as Input

To make port 2 an input port, it must be programmed as such by writing 1 to all its bits.

3.10.05 Port 3

Port 3 can be used as input or output. It does not need any pull-up resistors and it is configured as an input port upon reset, this is not the way it is most commonly used as it has additional function of providing some extremely important signals.

P3 Bit	Function	Pin
P3.0	RxD	10
P3.1	TxD	11
P3.2	$\overline{\text{INT0}}$	12
P3.3	$\overline{\text{INT1}}$	13
P3.4	T0	14
P3.5	T1	15
P3.6	$\overline{\text{WR}}$	16
P3.7	$\overline{\text{RD}}$	17

Serial communications

External interrupts

Timers

Read/Write signals of external memories

In systems based on 8751, 89C51 or DS89C4x0, pins 3.6 and 3.7 are used for I/O while the rest of the pins in port 3 are normally used in the alternate functions role

3.10.06 I/O Bit Manipulation Programming

I/O Ports and Bit Addressability

Sometimes we need to access only 1 or 2 bits of the port.

```
BACK:   CPL     P1.2        ;complement P1.2
        ACALL   DELAY
        SJMP    BACK

;another variation of the above program
AGAIN:  SETB    P1.2        ;set only P1.2
        ACALL   DELAY
        CLR     P1.2        ;clear only P1.2
        ACALL   DELAY
        SJMP    AGAIN
```

Single Bit addressability of Ports

P0	P1	P2	P3	Port Bit
P0.0	P1.0	P2.0	P3.0	D0
P0.1	P1.1	P2.1	P3.1	D1
P0.2	P1.2	P2.2	P3.2	D2
P0.3	P1.3	P2.3	P3.3	D3
P0.4	P1.4	P2.4	P3.4	D4
P0.5	P1.5	P2.5	P3.5	D5
P0.6	P1.6	P2.6	P3.6	D6
P0.7	P1.7	P2.7	P3.7	D7

Instructions that are used for signal-bit operations are as following

Instruction		Function
SETB	bit	Set the bit (bit = 1)
CLR	bit	Clear the bit (bit = 0)
CPL	bit	Complement the bit (bit = NOT bit)
JB	bit, target	Jump to target if bit = 1 (jump if bit)
JNB	bit, target	Jump to target if bit = 0 (jump if NOT bit)
JBC	bit, target	Jump to target if bit = 1, clear bit (jump if bit, then clear)

Example 3.5:

Write a program to perform the following:
(a) Keep monitoring the P1.2 bit until it becomes high
(b) When P1.2 becomes high, write value 45H to port 0
(c) Send a high-to-low (H-to-L) pulse to P2.3

Solution:

```
        SETB   P1.2          ;make P1.2 an input
        MOV    A, #45H       ;A=45H
AGAIN:  JNB    P1.2, AGAIN   ;get out when P1.2=1
        MOV    P0, A         ;issue A to P0
        SETB   P2.3          ;make P2.3 high
        CLR    P2.3          ;make P2.3 low for H-to-L
```

Example 3.6:

Assume that bit P2.3 is an input and represents the condition of an oven. If it goes high, it means that the oven is hot. Monitor the bit continuously. Whenever it goes high, send a high-to-low pulse to port P1.5 to turn on a buzzer.

Solution:

```
HERE:   JNB    P2.3, HERE    ;keep monitoring for high
        SETB   P1.5          ;set bit P1.5=1
        CLR    P1.5          ;make high-to-low
        SJMP   HERE          ;keep repeating
```

Example 3.7:

A switch is connected to pin P1.7. Write a program to check the status of SW and perform the following:
(a) If SW=0, send letter 'N' to P2
(b) If SW=1, send letter 'Y' to P2
Use the carry flag to check the switch status.

Solution:

```
        SETB   P1.7        ;make P1.7 an input
AGAIN:  MOV    C, P1.2     ;read SW status into CF
        JC     OVER        ;jump if SW=1
        MOV    P2, #'N'    ;SW=0, issue 'N' to P2
        SJMP   AGAIN       ;keep monitoring
OVER:   MOV    P2, #'Y'    ;SW=1, issue 'Y' to P2
        SJMP   AGAIN       ;keep monitoring
```

3.10.07 Arithmetic & Logic Instructions and Programs

3.10.07.01 Arithmetic Instructions

Addition

```
ADD A, source          ;A = A + source
```

The instruction ADD is used to add two operands. Destination operand is always in register A. Source operand can be a register, immediate data, or in memory. Memory-to-memory arithmetic operations are never allowed in 8051 Assembly Language.

Example 3.8:

Show how the flag register is affected by the following instruction.

```
        MOV   A, #0F5H      ;A=F5 hex
        ADD   A, #0BH       ;A=F5+0B=00
```

Solution:

```
        F5H   1 1 1 1  0 1 0 1
    +   0BH   0 0 0 0  1 0 1 1
        ------------------------
        100H  0 0 0 0  0 0 0 0
```

CY=1, since there is a carry out from D7
PF=1, because the number of 1s is zero (an even number), PF is set to 1.
AC=1, since there is a carry from D3 to D4

3.10.07.02 Logic Instructions

AND LOGIC

ANL destination, source ;dest = dest AND source

This instruction will perform a logic AND on the two operands and place the result in the destination. The destination is normally the accumulator. The source operand can be a register, in memory, or immediate.

Show the results of the following.

```
        MOV   A, #35H       ;A = 35H
        ANL   A, #0FH       ;A = A AND OFH
35H     0 0 1 1 0 1 0 1
0FH     0 0 0 0 1 1 1 1
05H     0 0 0 0 0 1 0 1
```

ANL is often used to mask (set to 0) certain bits of an operand

OR LOGIC

ORL destination, source ;dest = dest OR source

The destination and source operands are ORed and the result is placed in the destination. The destination is normally the accumulator. The source operand can be a register, in memory, or immediate

Show the results of the following:

```
        MOV   A, #04H       ;A = 04
        ORL   A, #68H       ;A = 6C
04H     0 0 0 0 0 1 0 0
68H     0 1 1 0 1 0 0 0
6CH     0 1 1 0 1 1 0 0
```

ORL instruction can be used to set certain bits of an operand to 1

Test your knowledge:

1. If we want to repeat an action more times than 256, we use a loop inside a loop, which is called _____.

2. The Call instruction is used to call _____.

3. Subroutines are often used to perform tasks that need to be performed _____.

4. The unconditional jump is a jump in which control is transferred unconditionally to the _____.

5. How to make any port as an Input port:

   _____

   _____

   _____

6. In both OR and AND logics the destination is normally the accumulator ( ).

7. The source operand in both OR and AND logics can be a register ( ).

Chapter-4: Introduction to MCU 8051 IDE Software

Objectives: Upon completion of this chapter, the student should be able to:

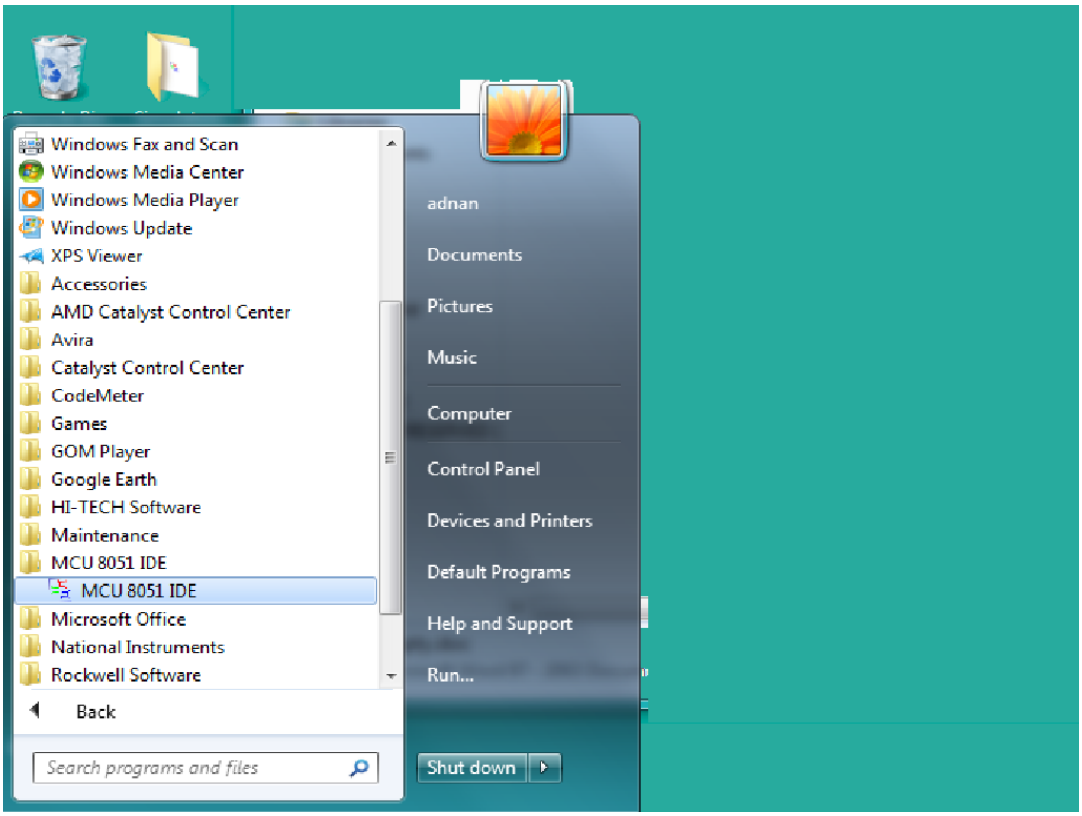
1- Know how to use MCU 8051 IDE Software.
2- Understand how to create new project.
3- Learn how write, save, compile and run the program.

4.01 Guide to Using MCU-8051-IDE Simulator Software

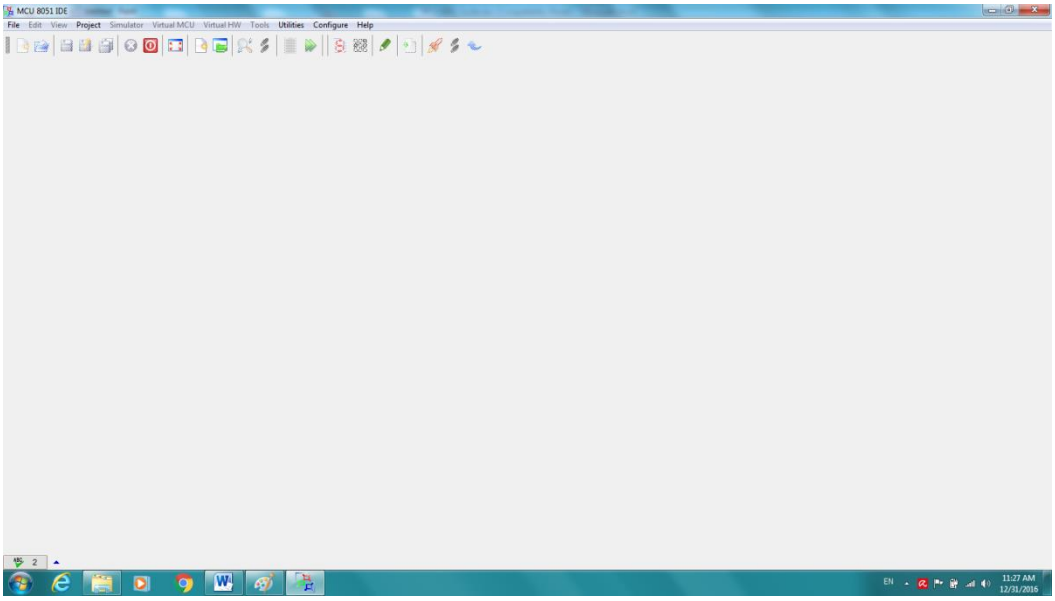
This tutorial shows step-by-step procedure on how to:
1-**Start** → **2-Write** → **3-Compile**→ **4-Run** an assembly language program using **MCU-8051-IDE software**.

1- Start:

1a) Go to the start menu and start the program **MCU 8051 IDE** as shown in the screen below.

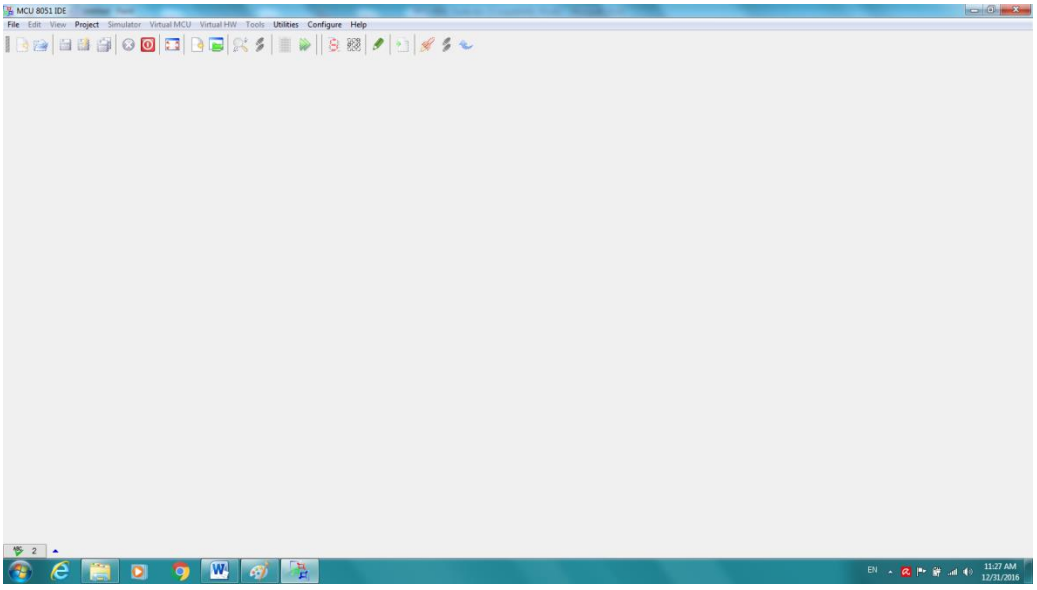


1b) After starting the program you will get the following screen.

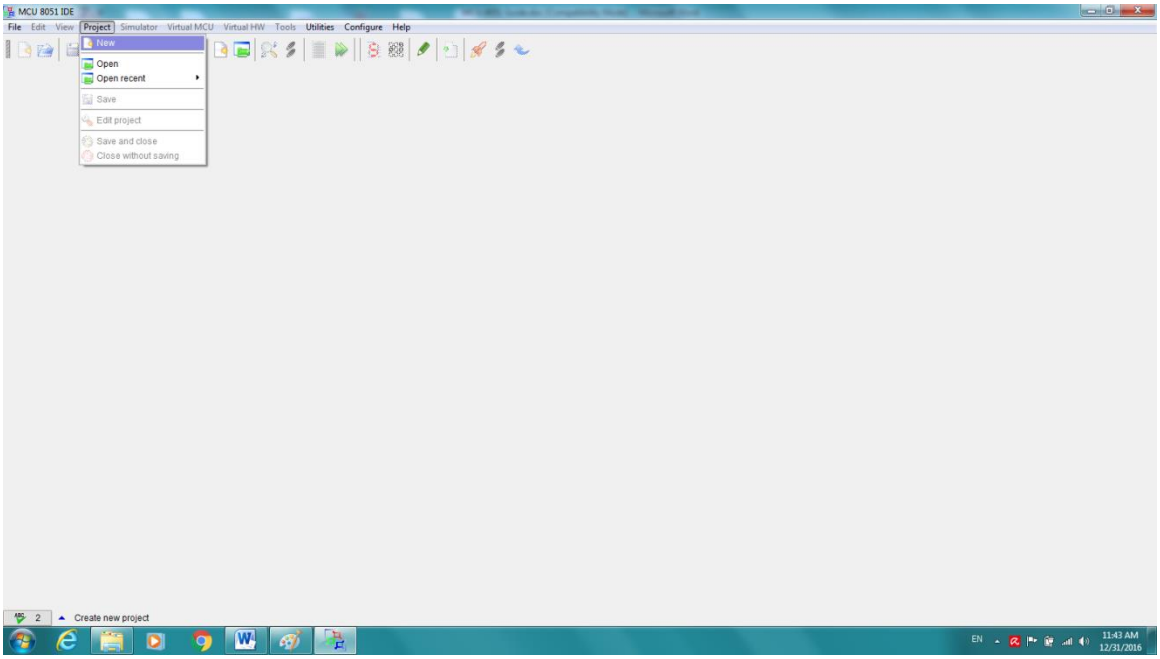


1c) Now we want to start a new project. You should create a new folder (directory) to store your project. Minimize the program screen and go to the desktop and create a new folder (Directory), name it: **Project-1 DIR**

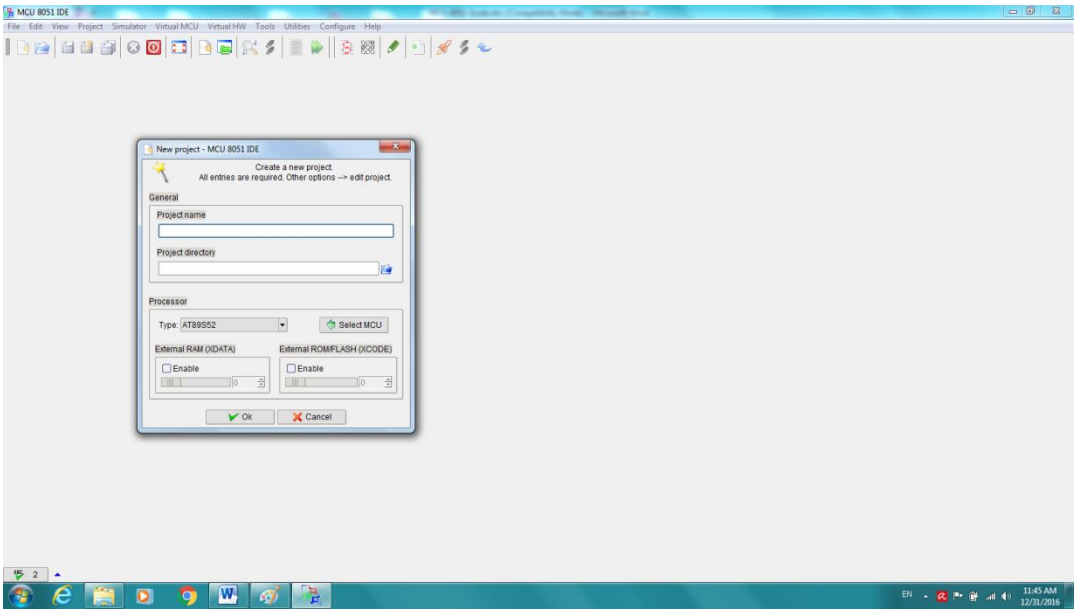
1d) Once you created the new folder **Project-1 DIR**, go back and maximize **MCU 8051 IDE** now you are back in the software with the following screen.



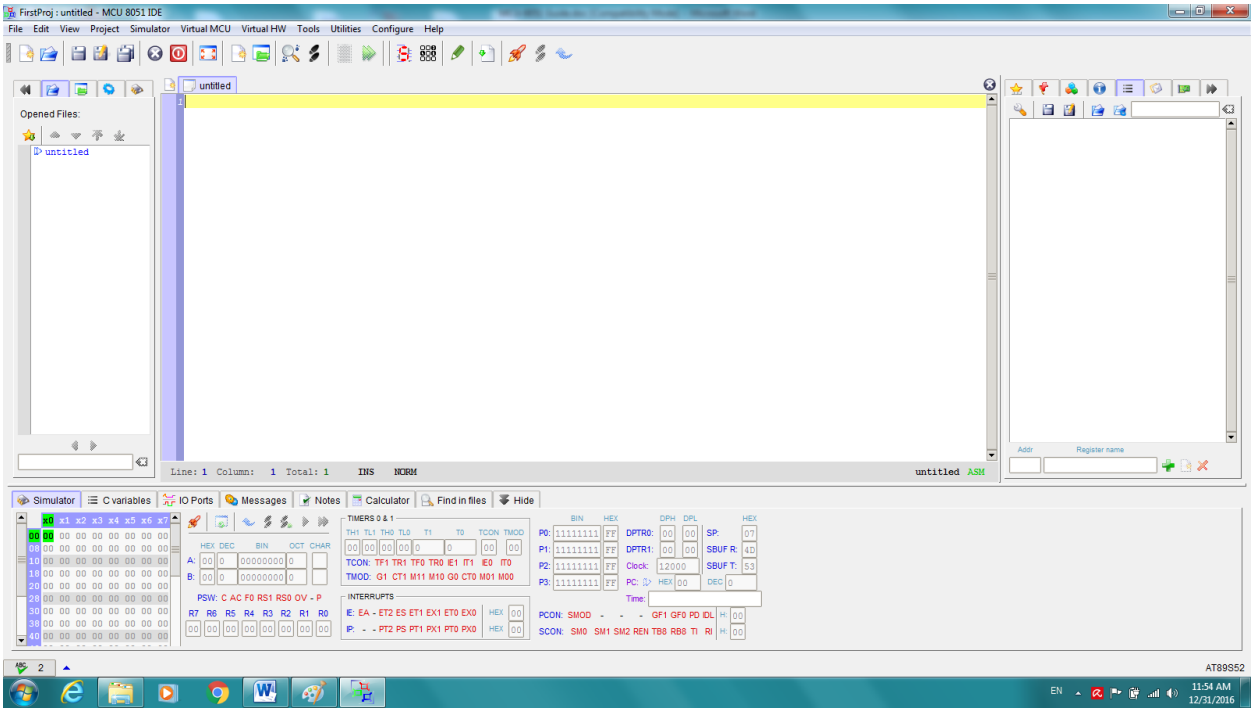
1e) Now we want to start a new project. Go to **Project** menu and select **New** as shown in the following screen.



1f) After selecting **New** from the **Project** menu, you will see the following screen with a dialog box to complete.



1g) For *project name* write: **Project1.** For *project directory*, go to the right side and select the folder which you created earlier: **Project-1 DIR,** for the *processor type* select: **8051** and click **OK**. You will get the following screen.

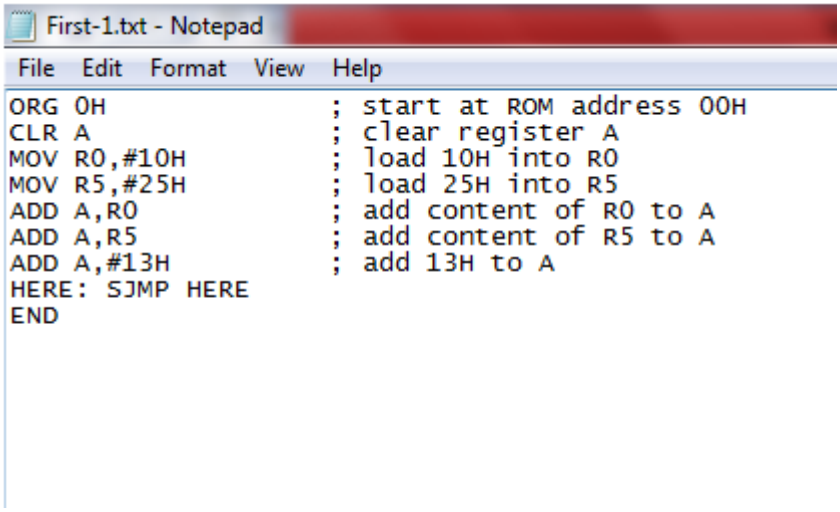


4.02 Write:

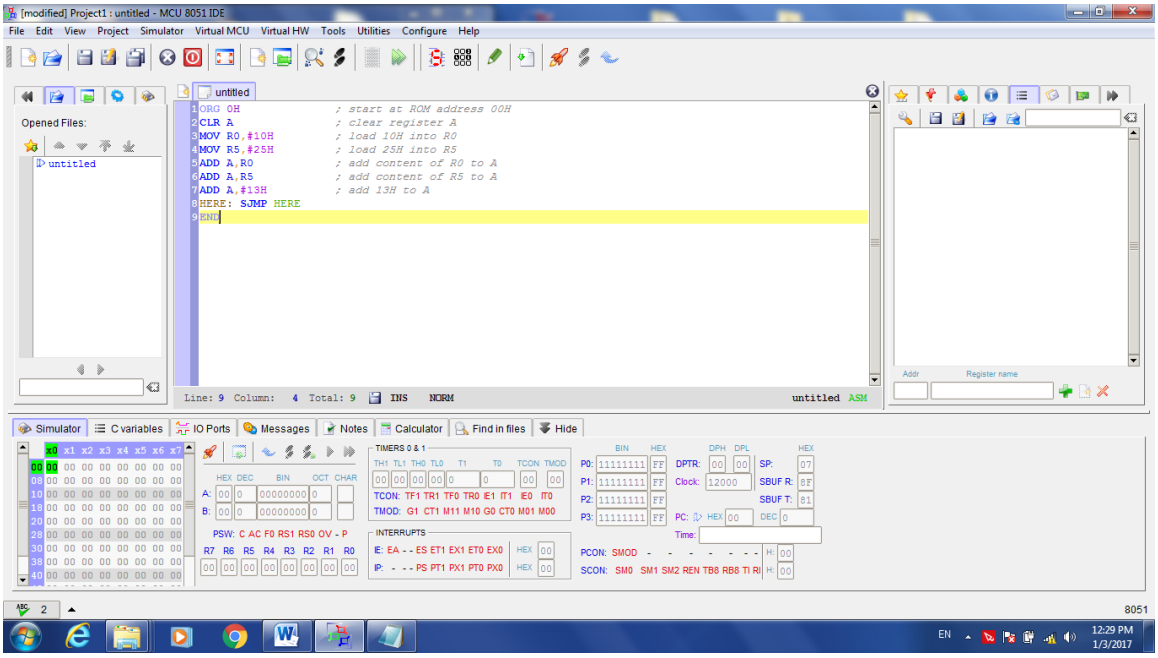
2a) Now we want to write the program. Minimize the **MUC 8051 IDE** software. Go to the start menu and start **Note Pad** the program and write the following program. Save the program with the name: Prog1

```
ORG 0H              ; Start at ROM address 00H
CLR A               ; clear register A
MOV R0,#10H         ; load 10H into R0
MOV R5,#25H         ; load 25H into R5
ADD A,R0            ; add content of R0 to A
ADD A,R5            ; add content of R5 to A
ADD A,#13H          ; add 13H to A
HERE: SJMP HERE
END
```

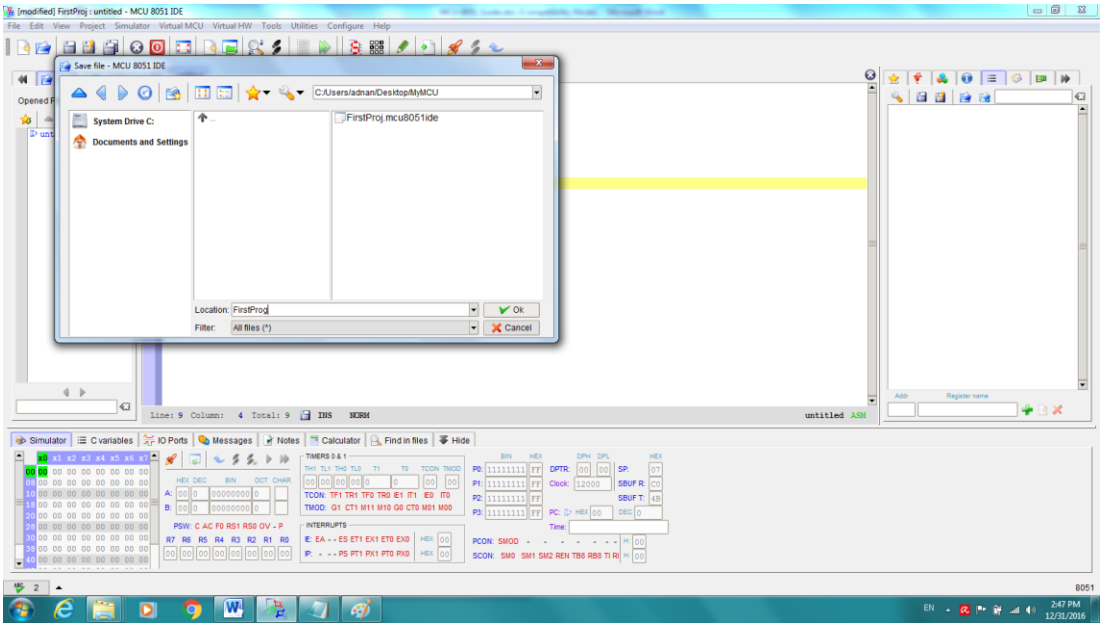
2b) You should have now the following screen (In the Note Pad screen). Save the program in the Notepad with the name: **Prog1**



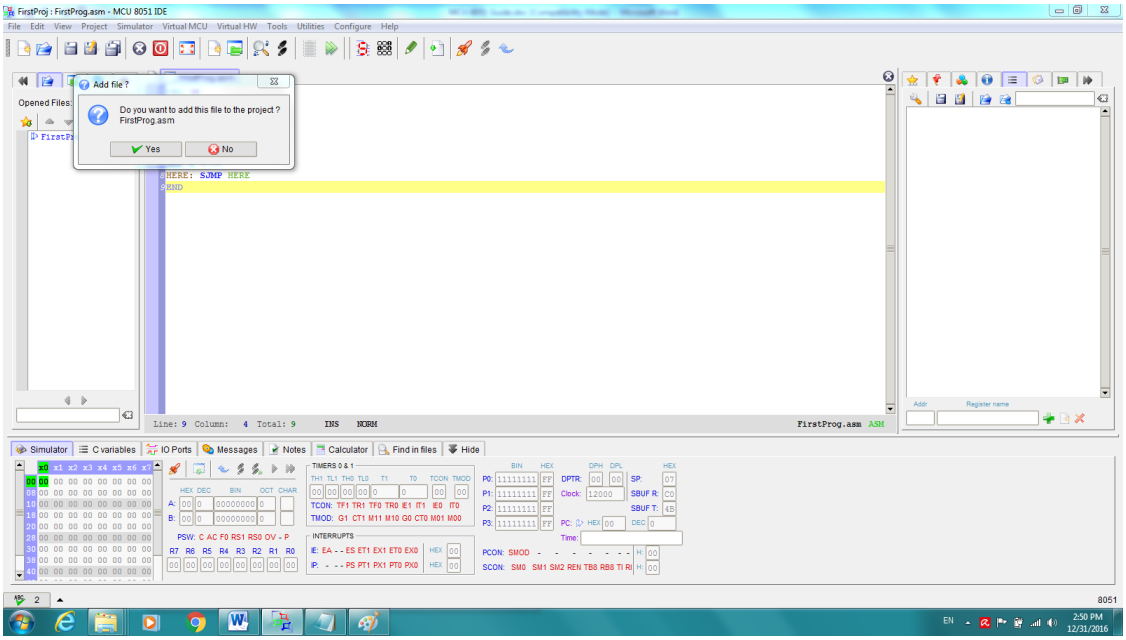
2c) **Copy** the program from the Note Pad and **Paste** it in the program area of **MUC 8051 IDE** screen. You should have now the following screen.



2d) Now we need to save the program. Go to **File** menu and select **Save as**. You should have now the following screen.

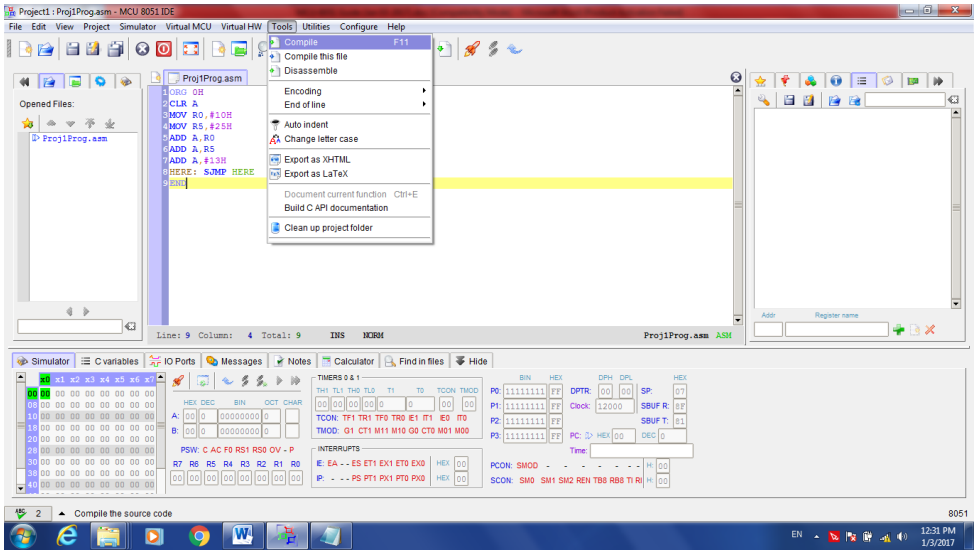


2e) For the *Location* write the name of the program: **Project1Prog1** and click **OK**. You will get the following screen click **yes**.

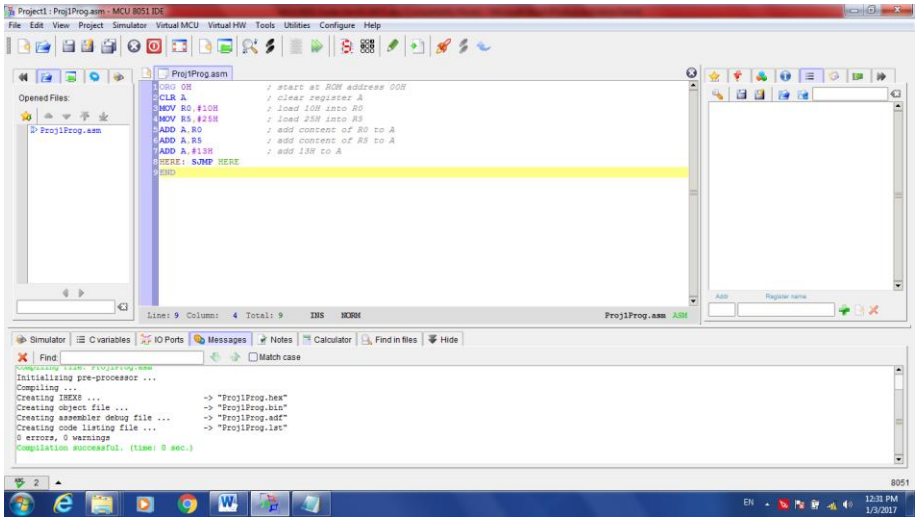


4.03 Compile:

3a) Now to be able to test the program we must compile the program then run it. To compile the program, go to **Tools** menu and select **Compile** as shown in the picture below. Compiling the program converts the program to be compatible with the 8051 and starting the assembler.

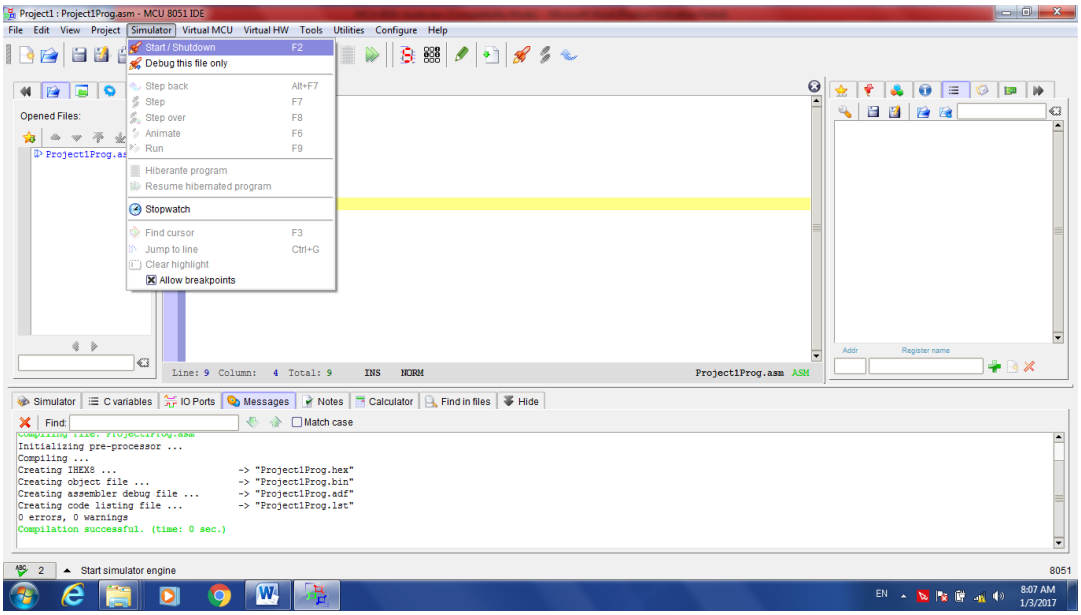


3b) When you compile the program you will get the following screen, showing the necessary files created by the compiler in the bottom window part of the simulator screen. (If there are errors, you will see the errors in that window. You must correct the errors before preceding)

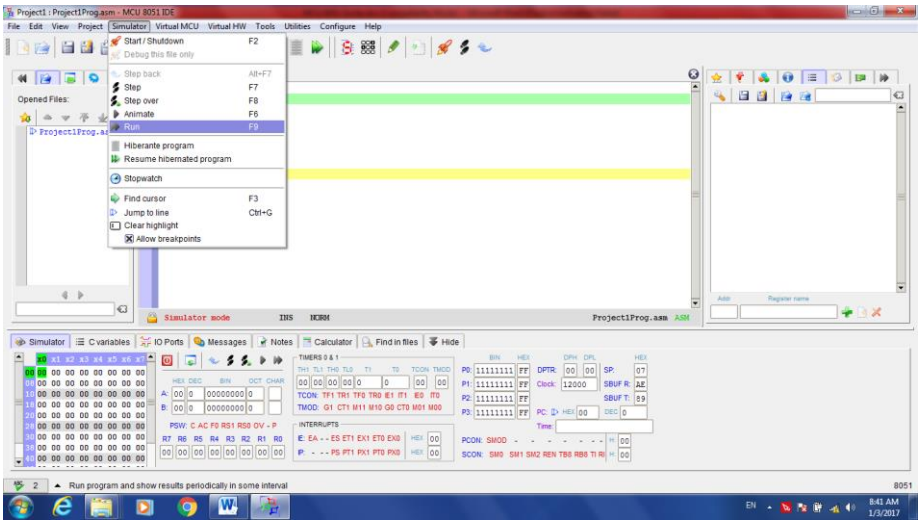


4.04 Run:

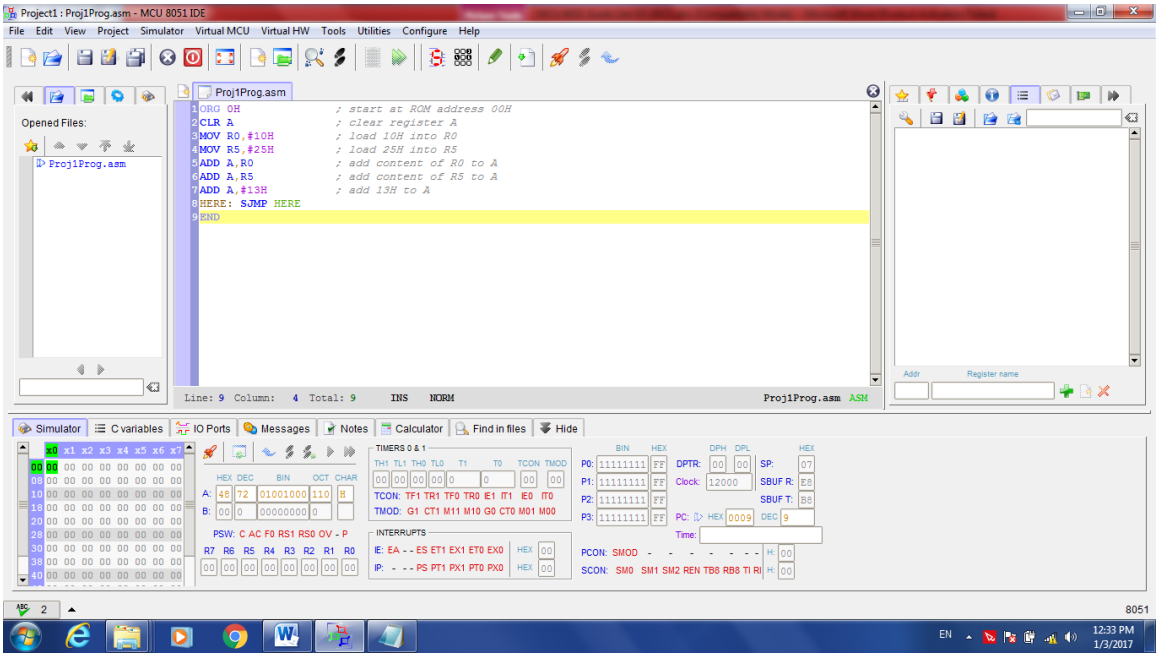
4a) Now we want to run the program. First we have to switch the **MCU 8051 IDE** software to **Simulator mode**. To switch the software to **Simulator mode**, go to the **Simulator** menu and select **Start/Shutdown** as shown in the picture below.



4b) When you place the software in the simulator mode, you will see a picture of a lock and **Simulator mode** showing in the middle of the page Now to run the program, go to the **simulator** menu and select **Run** as shown in the picture below.



4c) Now you need to start the simulator. In the bottom window of the simulator software there is a red icon. This icon will start and stop the simulator from running. Press this icon button as shown below and the program now is running. The values in registers A and B are changed according to the program



4d) To save and close the program file in the project. Go **Project** menu and select **Save and close.**

4e) To close and exit the program and the simulator software. Go **File** menu and select **Close all** and again from the **File** menu and select **Quit**.

4.05 Procedure Summary:

This is a summary of the tutorial showing step-by-step procedure on how to:
1-**Start** → **2-Write** → **3-Compile**→ **4-Run** an assembly language program using **MCU-8051-IDE software**.

1- Create a folder to store the project called it: **Project-1 DIR**
2- (From Start Menu) Start the program **MCU 8051 IDE** → (From Project Menu) select **New** → (complete the dialog box) For project name: **Project1** → For project directory select: **Project-1 DIR** → Select type of processor: **8051** → click: **OK**

3- **Write** the program in **Note Pad** → **copy** the program and insert it in the program area of the simulator → (From File Menu) save the program **Save as** name it: **Project1Prog1** → Click: **OK** → Click: **yes**

4- (From Tools Menu) select: **Compile** → (From Simulator Menu) select: **Start/Shutdown** → (From Simulator Menu) select: **Run** → Press the red icon to start simulator.

5) To save and close the program file in the project. Go **Project** menu and select **Save and close.**

6) To close and exit the program and the simulator software. Go **File** menu and select **Close all** and again from the **File** menu and select **Quit**.

END

Chapter-5: Input-Output Interface

Objectives: Upon completion of this chapter, the student should be able to:

1- Learn about virtual hardware of MCU 8051 IDE Software.
2- Simulate program using virtual hardware.

5.01 Interfacing with Hardware

At this time we are going to learn how to interface the **MCU-8051-IDE software** with **Virtual HW** hardware namely **LED Panel**.

First we have to go through the steps which we have learned earlier. The steps are summarized below:

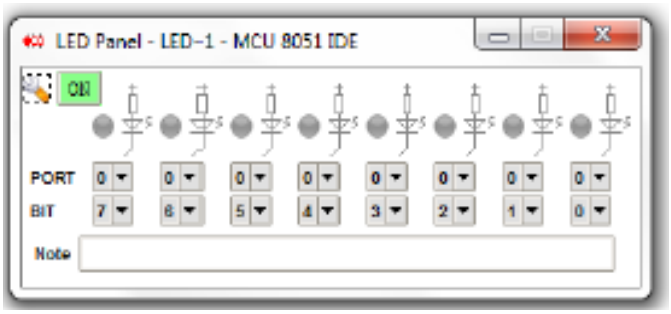
1- Create a folder to store the project called it: **Project-1 DIR**
2- (From Start Menu) Start the program **MCU 8051 IDE** → (From Project Menu) select **New** → (complete the dialog box) For project name: **Project1** → For project directory select: **Project-1 DIR** → Select type of processor: **8051** → click: **OK**

3- **Write** the program in **Note Pad** → **copy** the program and insert it in the program area of the simulator → (From File Menu) save the program **Save as** name it: **Project1Prog1** → Click: **OK** → Click: **yes**

Next we will attach the **Virtual HW** hardware namely **LED Panel** to the project using the following steps:

5.01.01 Attached Virtual HW hardware:

1) After writing and saving the program, you go to the **Virtual HW** pull down menu and select the **LED Panel**. You will get the following interface LED Panel on the top of the existing screen.



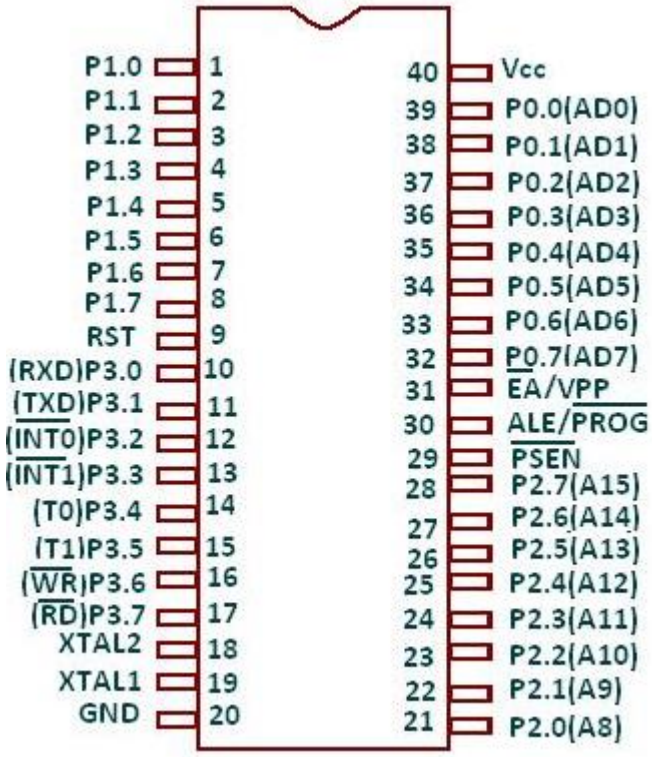
2) Now you need to configure the **port** and the **bits**.
3) Once it is configured. Save the configuration.
4) Now you can Compile and Run the program. You should see the simulation on the LED Panel working according to the program.

END

Chapter-6: MicroController Projects

Objectives: Upon completion of this chapter, the student should be able to:

1- Interface a circuit to the 8051 microcontroller.
2- Know the necessary pins for the operation of the 8051 microcontroller such as power supply, ground, reset, clock input
3- Write and simulate assembly language program.
4- Burn (download) assembly language program in the 8051 microcontroller.
5- Run and test 8051 micro-controlled circuit.



This Page is Left Blank

Project no. 1: Sending Outputs to LEDs Using Microcontroller Port P1.

Equipment & Software:

Microcontroller trainer	1
PC	1
Power Supply	1

Procedure:

Below is the summary of procedure steps:

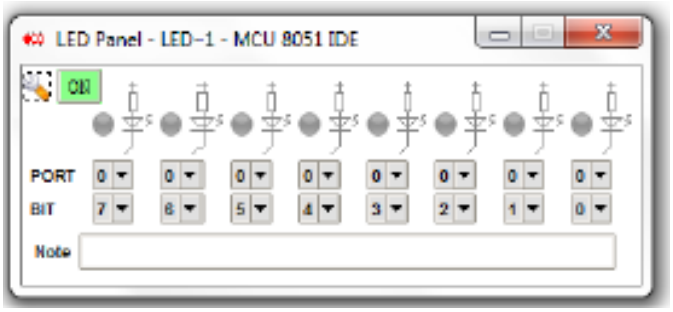
1) Write the program.
2) Compile the program and test it in the simulator.
3) Connect the circuit.
4) Burn the program to the 8051 microcontroller.
5) Place the 8051 microcontroller in the circuit.
6) Run (turn on) the circuit.

Below is the detailed of procedure steps:

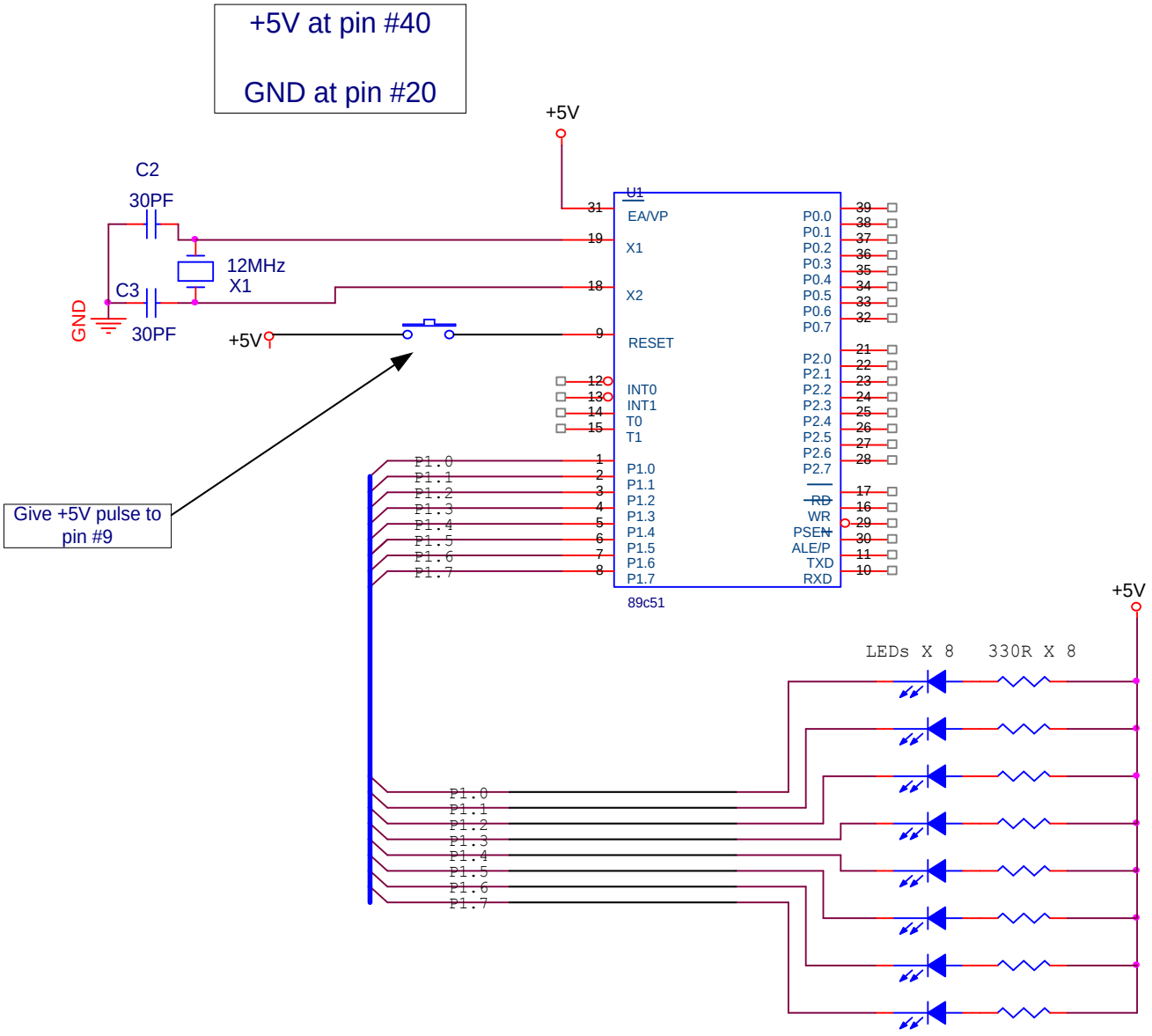
1) Using Notepad, write the program shown below and save it with the name:"Proj-1".

```
;File Name: Proj-1.ASM
            ORG     0H
            JMP     MAIN
MAIN:       MOV     A, #0H
AGAIN:      MOV     P1, A
            INC     A
            CALL    DLY
            CJNE    A, #0FFH, AGAIN
            MOV     A, #0H
            JMP     AGAIN
DLY:        MOV     R0, #0FFH
LP2:        MOV     R1, #0FFH
LP1:        DJNZ    R1, LP1
            DJNZ    R0, LP2
            RET
```

2) Copy the program to the simulator and use the same procedure for compilation of program as discussed in **chapter-4** and test the program using the simulator as described in **chapter-5**.



3) Connect the circuit shown below.

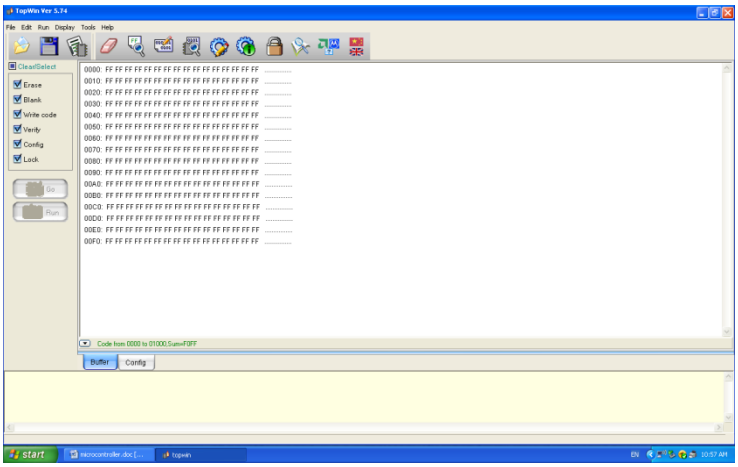


4) Now we want to burn (Download) the program to the 8051 microcontroller. Follow the steps below to do this task.

Burn the Program in the IC

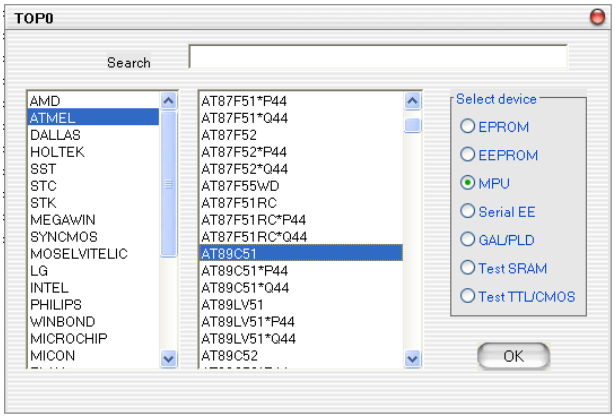
1- Plug in the IC programmer into the PC via USB port.

2- Run the software TOPWIN.



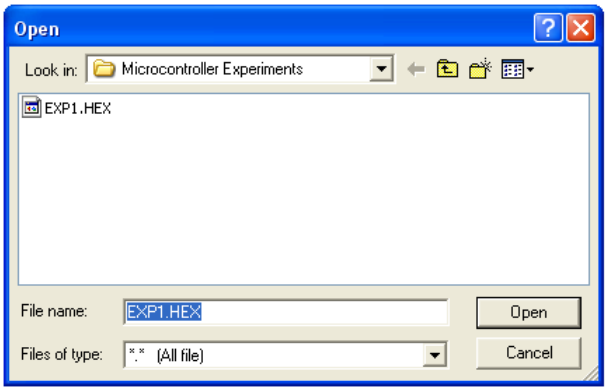
3- Select chip in the run menu. 

4- Select ATMEL → Select AT89C51 → Check ◙ MPU → OK



Burn the Program in the IC (Continue)

5- File → Open → Load Hex File from location of the program → Open.



You should see a dialogue box

.Hex Intel Hex

© NO Clear
© Fill by 00

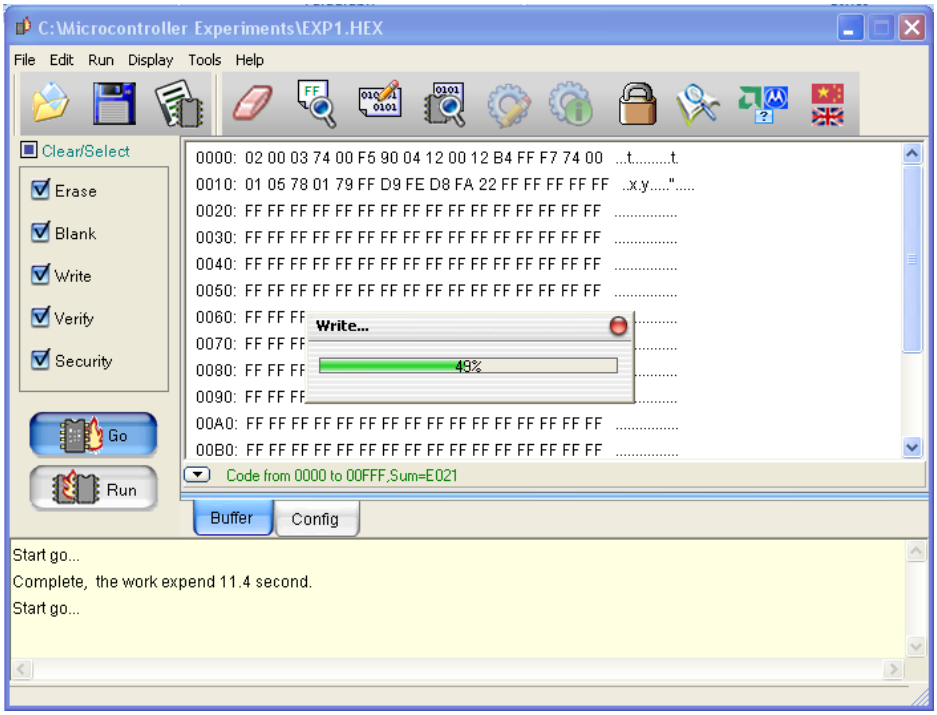
◙ Fill by 'EF'

Select Fill by 'EF' and press OK

Burn the Program in the IC (Continue)

6- Insert the 8051 microcontroller in the IC programmer.

7- Program the microcontroller by selecting Erase, Blank Write, Verify, Security and then press GO.



8- Remove the Microcontroller from the programmer.

Note: You can burn another IC just by placing another IC in the machine and (press run first???)

5) Place the 8051 microcontroller in the circuit.

6) Run (turn on) the circuit.

Troubleshooting:

The steps to troubleshoot the hardware if the program does not run properly are as follows:

i) Check the power to the card.

PIN Number	Expected Voltage	Measured Voltage
40	5 V	
20	0 V	

ii) Check the RST signal. It should be LOW & when switch is pressed, it should be HIGH & should go LOW again when switch is released.

RST Switch	PIN Number	Expected Voltage	Measured Voltage
Released	18	0 V	
	19	0 V	
Pressed	18	5 V	
	19	5 V	

iii) Check the EA signal; it must be high if using the Internal Memory of Microcontroller.

PIN Number	Expected Voltage	Measured Voltage
31	5 V	

Test your knowledge:

Modify the program to Decrement from FFH to 00H at the same port.

Note: Write down the Observation & Conclusion.

_____

_____

_____

_____

_____

_____

Project no. 2: Rotating LEDs Left-to-Right and Right-to-Left.

Equipment & Software:

Microcontroller trainer	1
PC	1
Power Supply	1

Procedure:

Below is the summary of procedure steps:

Note: (It is the same procedure used earlier in project no. 1)

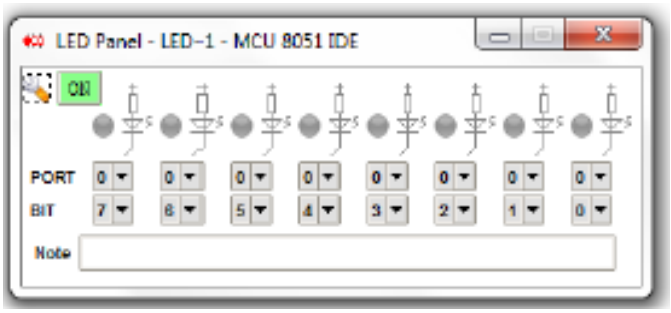
1) Write the program.
2) Compile the program and test it in the simulator.
3) Connect the circuit.
4) Burn the program to the 8051 microcontroller.
5) Place the 8051 microcontroller in the circuit.
6) Run (turn on) the circuit.

Below is the detailed of procedure steps:

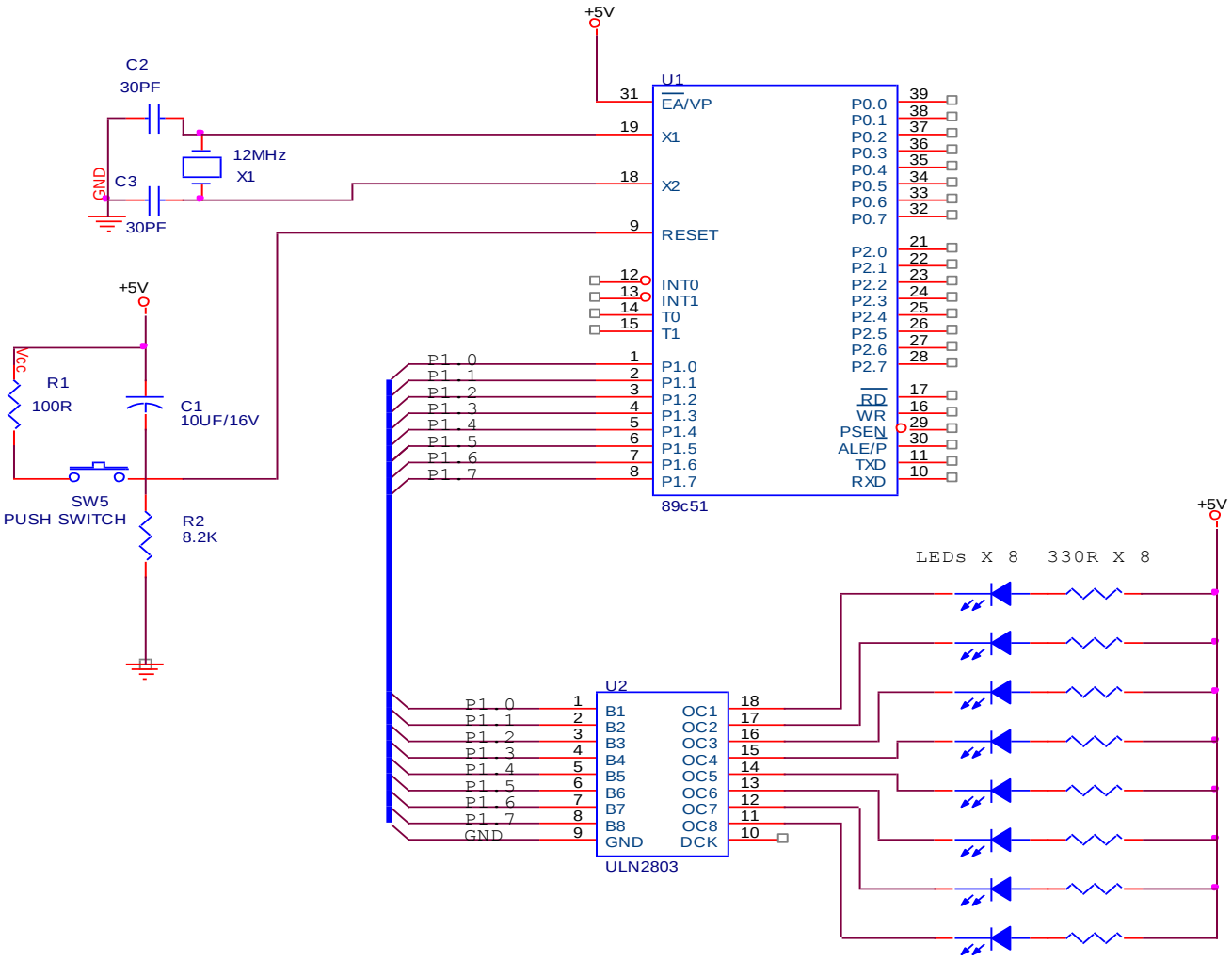
1) Using Notepad, write the program shown below and save it with the name:"Proj-2".

```
;File Name: Proj-2.ASM
            ORG     0H
            JMP     MAIN
MAIN:       MOV     A, #01H
AGAIN:      MOV     P1, A
            RL      A
            CALL    DLY
            CJNE    A, #80H, AGAIN
            MOV     P1, A
AGAIN2:     RR      A
            CALL    DLY
            CJNE    A, #01H, AGAIN2
            MOV     P1, A
            JMP     AGAIN
DLY:        MOV     R0, #0FFH
LP2:        MOV     R1, #0FFH
LP1:        DJNZ    R1, LP1
            DJNZ    R1, LP2
            RET
```

2) Copy the program to the simulator and use the same procedure for compilation of program as discussed in **chapter-4** and test the program using the simulator as described in **chapter-5**.



3) Connect the circuit shown below.



4) Now we want to burn (Download) the program to the 8051 microcontroller. Follow the same steps done in **project no.1** to do this task.

5) Place the 8051 microcontroller in the circuit.

6) Run (turn on) the circuit.

Troubleshooting:

The steps to troubleshoot the hardware if the program does not run properly are as follows:

i) Check the power to the card.

PIN Number	Expected Voltage	Measured Voltage
40	5 V	
20	0 V	

ii) Check the RST signal. It should be LOW & when switch is pressed, it should be HIGH & should go LOW again when switch is released.

RST Switch	PIN Number	Expected Voltage	Measured Voltage
Released	18	0 V	
	19	0 V	
Pressed	18	5 V	
	19	5 V	

iii) Check the EA signal; it must be high if using the Internal Memory of Microcontroller.

PIN Number	Expected Voltage	Measured Voltage
31	5 V	

Test your knowledge:

Write a program to rotate a Stepper Motor using RL/RR commands.

Note: Write down the Observation & Conclusion.

_____

_____

_____

_____

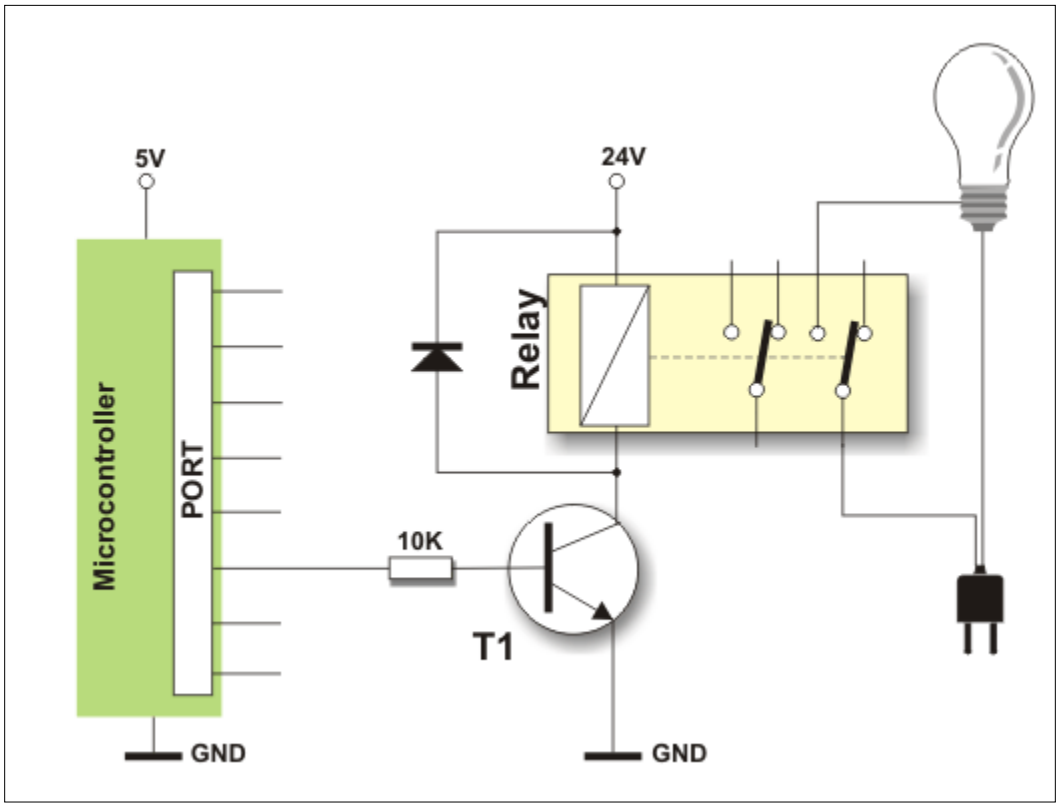
_____

_____

This Page is Left Blank

Project no. 3: Input from Pushbutton & Activating Relays

In this project we will learn the interfacing of Push buttons with a port bit.To eliminate the debouncing effect in mechanical switches by using the Delay function.



Equipment & Software:

Microcontroller trainer	1
PC	1
Power Supply	1

Procedure:

Do the same procedure 1-to-6 used earlier in **project no. 1**

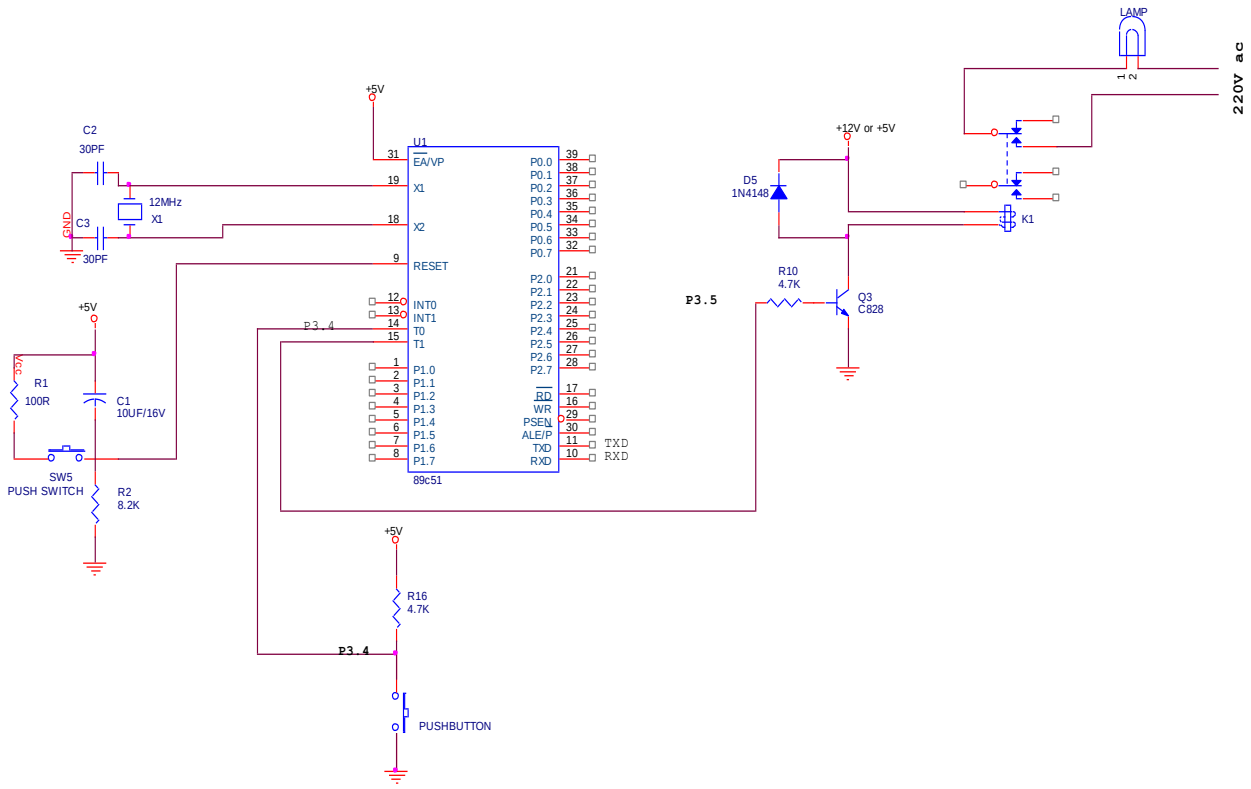
1- Write the program.
2- Compile the program and test it in the simulator.
3- Connect the circuit.
4- Burn the program to the 8051 microcontroller.
5- Place the 8051 microcontroller in the circuit.
6- Run (turn on) the circuit.

Below is the detailed of procedure steps:

1) Using Notepad, write the program shown below and save it with the name:"Proj-3".

```
;File Name: Proj-3.ASM
         ORG    0H
         JMP    MAIN          ;Jump to Main on Power On or Reset
MAIN:    CLR    P3.5          ;Turn OFF the relay on Start up
WAIT:    JB     P3.4, WAIT
         CALL   DLY
         CPL    P3.5
         JMP    WAIT
DLY:     MOV    R0, #0FFH     ;Delay can be increased/decreased
LP1:     DJNZ   R0, LP1       ;depending upon the Switch quality
         RET
```

2) Connect the circuit shown below.



3) Now we want to burn (Download) the program to the 8051 microcontroller. Follow the same steps done in **project no.1** to do this task.

4) Place the 8051 microcontroller in the circuit.

5) Run (turn on) the circuit.

Troubleshooting:

The steps to troubleshoot the hardware if the program does not run properly are as follows:

i) Check the power to the card.

PIN Number	Expected Voltage	Measured Voltage
40	5 V	
20	0 V	

ii) Check the RST signal. It should be LOW & when switch is pressed, it should be HIGH & should go LOW again when switch is released.

RST Switch	PIN Number	Expected Voltage	Measured Voltage
Released	18	0 V	
	19	0 V	
Pressed	18	5 V	
	19	5 V	

iii) Check the EA signal; it must be high if using the Internal Memory of Microcontroller.

PIN Number	Expected Voltage	Measured Voltage
31	5 V	

Note: Write down the Observation & Conclusion.

_____

_____

_____

_____

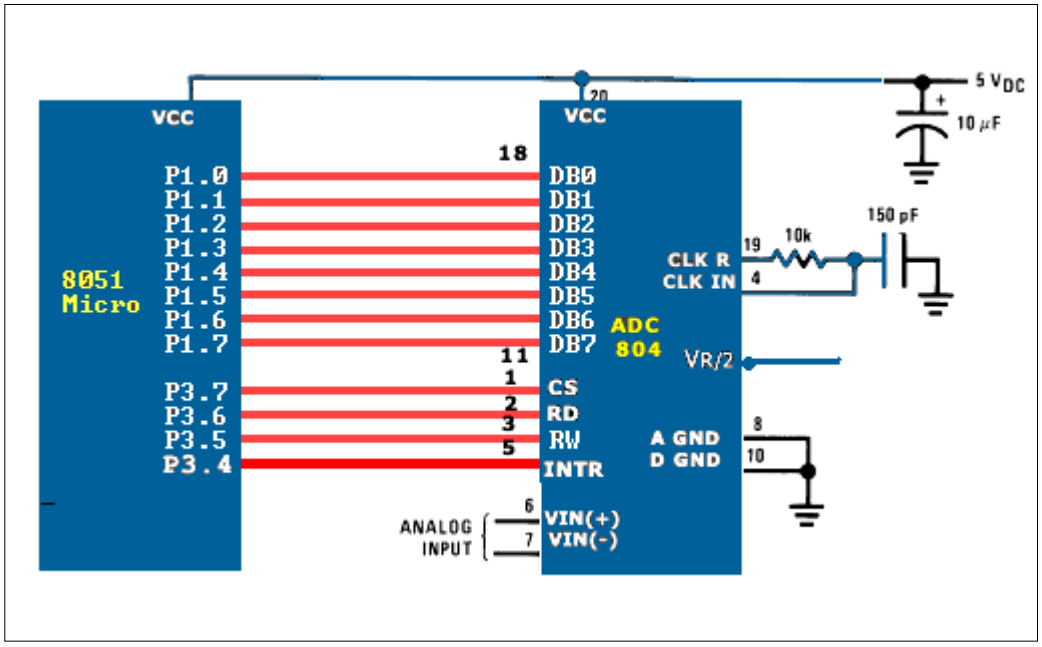
_____

_____

This Page is Left Blank

Project no. 4: Interfacing ADC with Microcontroller

In this project we will learn the operation and interfacing technique of ADC with microcontroller.



Equipment & Software:

Microcontroller trainer	1
PC	1
Power Supply	1

Procedure:

Do the same procedure 1-to-6 used earlier in **project no. 1**

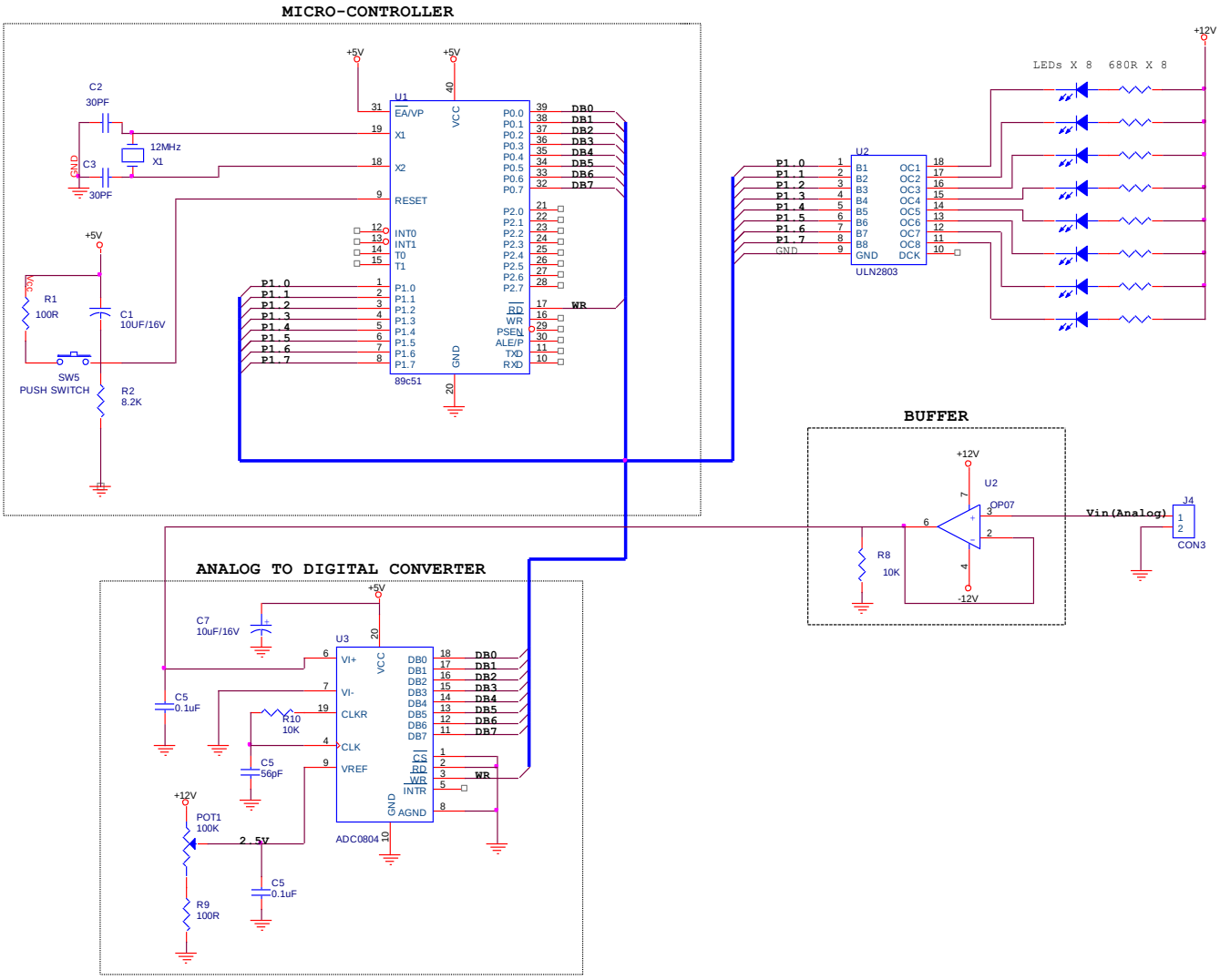
1- Write the program.
2- Compile the program and test it in the simulator.
3- Connect the circuit.
4- Burn the program to the 8051 microcontroller.
5- Place the 8051 microcontroller in the circuit.
6- Run (turn on) the circuit.

Below is the detailed of procedure steps:

1) Using Notepad, write the program shown below and save it with the name:"Proj-4".

```
;File Name: Proj-4.ASM
          ORG    0H
          JMP    MAIN
MAIN:     MOV    P0, #FFH
          MOV    P1, #FFH
AGAIN:    CALL   PULSE     ;WR pulse to ADC to start conversion
          CALL   DLY       ;Wait for conversion to complete
          MOV    A, P0     ;Read ADC value to P0
          MOV    P1, A     ;Output that value to P1 (LEDs) to
          JMP    AGAIN     ;observe change in voltage
PULSE:    SETB   P1.7
          NOP              ;Delay for 1uS
          CLR    P1.7
          NOP
          SETB   P1.7
          RET
DLY:      MOV    R0, #FFH
LP1:      DJNZ   R1, LP1
          RET
```

2) Connect the circuit shown below.



3) Now we want to burn (Download) the program to the 8051 microcontroller. Follow the same steps done in **project no.1** to do this task.

4) Place the 8051 microcontroller in the circuit.

5) Run (turn on) the circuit.

Troubleshooting:

The steps to troubleshoot the hardware if the program does not run properly are as follows:

i) Check the power to the card. AT89C51

PIN Number	Expected Voltage	Measured Voltage
40	5 V	
20	0 V	

ii) Check the power to the card. ADC0804

PIN Number	Expected Voltage	Measured Voltage
20	5 V	
10	0 V	

iii) Check the RST signal. It should be LOW & when switch is pressed, it should be HIGH & should go LOW again when switch is released.

RST Switch	PIN No.	Expected Voltage	Measured Voltage
Released	18	0 V	
	19	0 V	
Pressed	18	5 V	
	19	5 V	

iv) Check the EA signal; it must be high if using the Internal Memory of Microcontroller.

PIN Number	Expected Voltage	Measured Voltage
31	5 V	

Check the result to P1 (LEDs will show the change when analog voltage is varied).

Note: Write down the Observation & Conclusion.

_____

_____

_____

_____

_____

_____

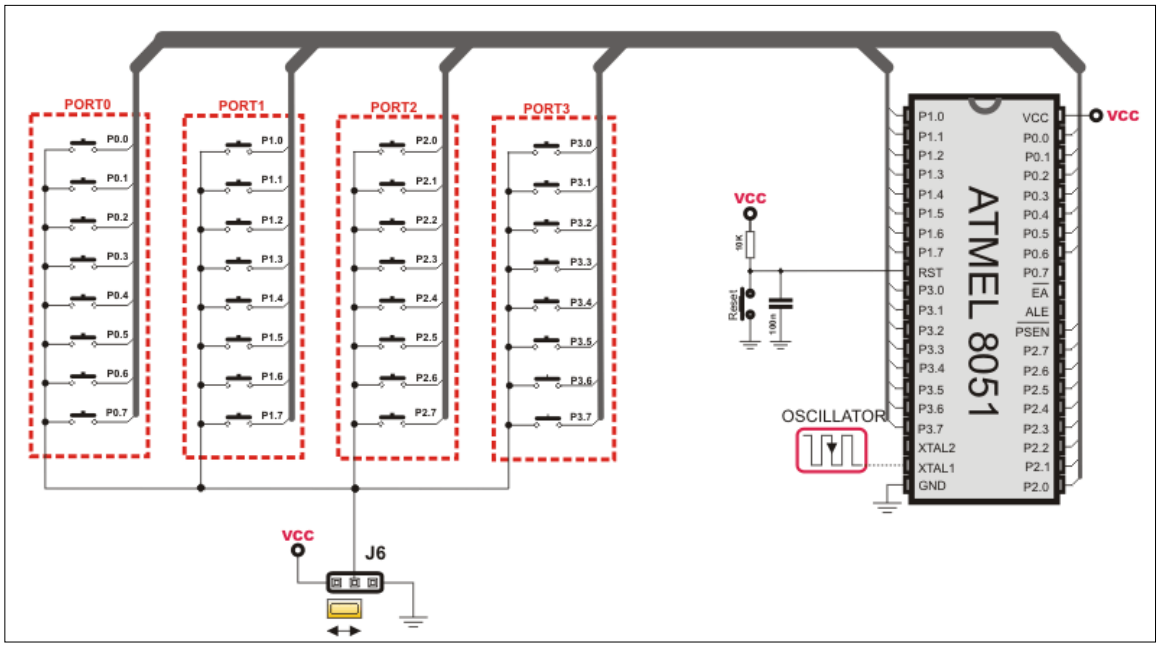
This Page is Left Blank

Project no. 5: Incrementing / Decrementing and Rotating Output Port 2

In this project we will be sending output to port 2 (P2) in increment / decrement orders and rotate left / right orders using Push Switches.

Program Description:

Part A of this experiment is similar to experiment 1. For part A, we are using the same program as we used in the experiment 1 to test our hardware. For part B, we will see how we can interface push switches with the Microcontroller.



Equipment & Software:

Microcontroller trainer	1
PC	1
Power Supply	1

Procedure:

Do the same procedure 1-to-6 used earlier in **project no. 1**

1- Write the program.
2- Compile the program and test it in the simulator.
3- Connect the circuit.
4- Burn the program to the 8051 microcontroller.
5- Place the 8051 microcontroller in the circuit.
6- Run (turn on) the circuit.

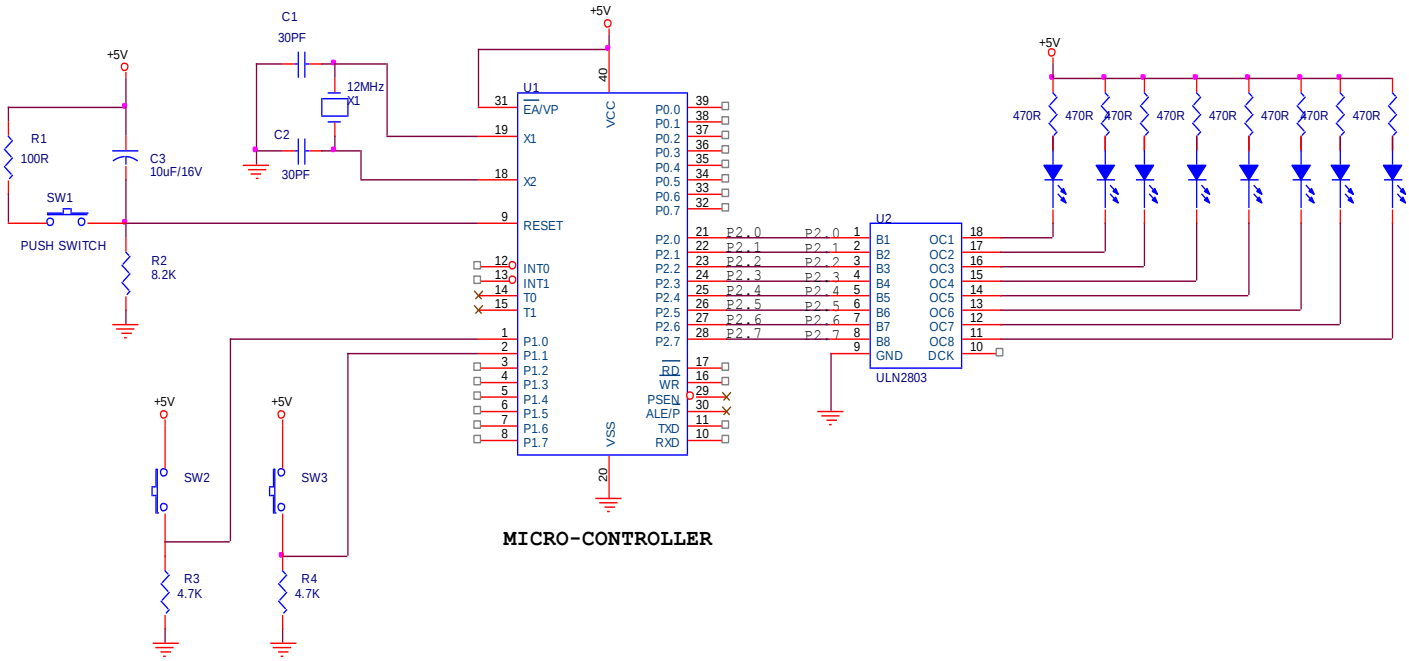
Below is the detailed of procedure steps:

1) Using Notepad, write the program shown below and save it with the name:"Proj-5".

Part A: Program:

```
;File Name: Proj-5.ASM
;Program to send outputs to port 2 (P2) in incremental order
        ORG   0H              ;Set the origin
        JMP   MAIN
MAIN:   MOV   A, #0H          ;Clear A
AGAIN:  MOV   P2, A           ;output to P2
        CALL  DLY             ;small delay (approx.. 0.1 sec)
        INC   A               ;increment A
        CJNE  A,      #0FFH,  ;compare A until 255 & jump to
              AGAIN           ;AGAIN
        JMP   MAIN            ;jump to MAIN
DLY:    MOV   R1, #0FFH       ;outer loop
LP2:    MOV   R2, #0FFH       ;inner loop
LP1:    DJNZ  R2, LP1         ;dec R2 & jump to LP1 until R2=0
                              ;(inner loop)
        DJNZ  R1, LP2         ;dec R1 & jump to LP2 until R1=0
                              ;(outer loop)
        RET                   ;return from subroutine
        END
```

2) Connect the circuit shown below.



3) Now we want to burn (Download) the program to the 8051 microcontroller. Follow the same steps done in **project no.1** to do this task.

4) Place the 8051 microcontroller in the circuit.

5) Run (turn on) the circuit.

Troubleshooting:

The steps to troubleshoot the hardware if the program does not run properly are as follows:

i) Check the power to the card.

PIN Number	Expected Voltage	Measured Voltage
40	5 V	
20	0 V	

ii) Check the RST signal. It should be LOW & when switch is pressed, it should be HIGH & should go LOW again when switch is released.

RST Switch	PIN Number	Expected Voltage	Measured Voltage
Released	18	0 V	
	19	0 V	
Pressed	18	5 V	
	19	5 V	

iii) Check the EA signal; it must be high if using the Internal Memory of Microcontroller.

PIN Number	Expected Voltage	Measured Voltage
31	5 V	

Assignment:

Modify the program to send outputs to port2 (P2) in the rotate left & rotate right orders (continuously), show to the instructor & run it.

Part B:

The purpose of this part is to use the push switches to increment & decrement on port 2 (P2) as follows:

i. When P1.0 is pressed, there should be increment to P2,
ii. When P1.1 is pressed, there should be decrement to P2.

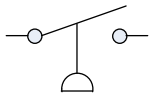
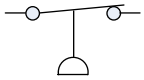
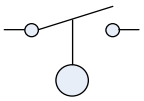
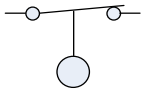
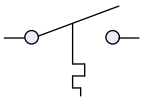
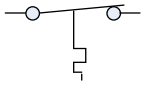
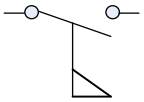
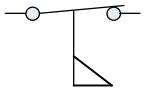
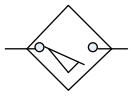
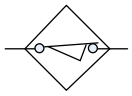
Program:

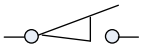
;File Name: Proj-5C.ASM			
	ORG	0H	;Set the origin
	JMP	MAIN	
MAIN:	MOV	A, #0H	;clear A
	MOV	P2, A	
AGAIN:	JB	P1.0, INCC	;if P1.0 is pressed
	JB	P1.1, DECC	;if P1.1 is pressed
	JMP	AGAIN	
INCC:	CALL	DLY	
	INC	A	
	MOV	P2, A	
	JMP	AGAIN	
DECC:	CALL	DLY	
	DEC	A	
	MOV	P2, A	
	JMP	AGAIN	
DLY:	MOV	R1, #3FH	;outer loop
LP2:	MOV	R2, #0FFH	;inner loop
LP1:	DJNZ	R2, LP1	;dec R2 & jump to LP1 until R2=0 (inner loop)
	DJNZ	R1, LP2	;dec R1 & jump to LP2 until R1=0 (outer loop)
	RET		
	END		

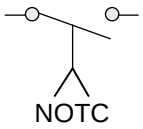
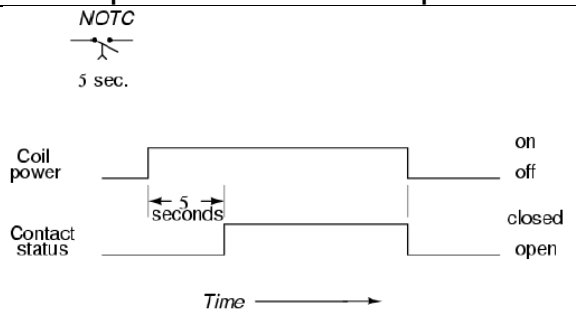
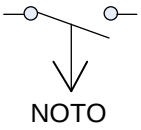
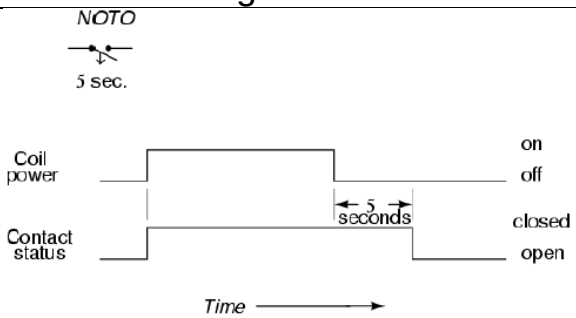
This Page is Left Blank

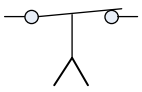
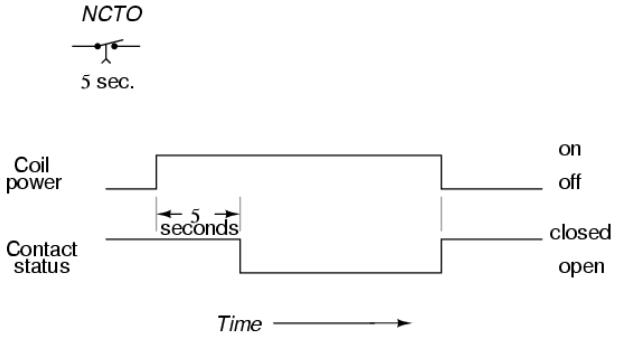
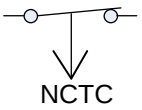
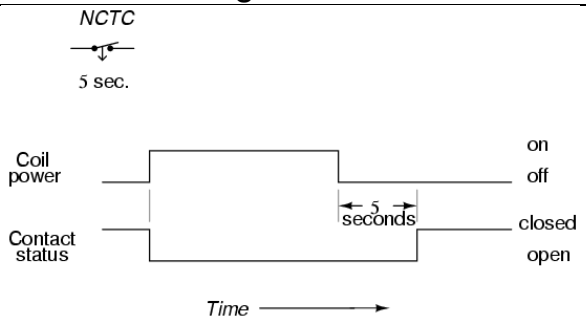
Appendix-A:

Type of Switches

Type of Switch	Symbol & Status	
	NO	NC
Pressure Switch		
Level Switch		
Temperature Switch		
Flow Switch		
Proximity Switch		

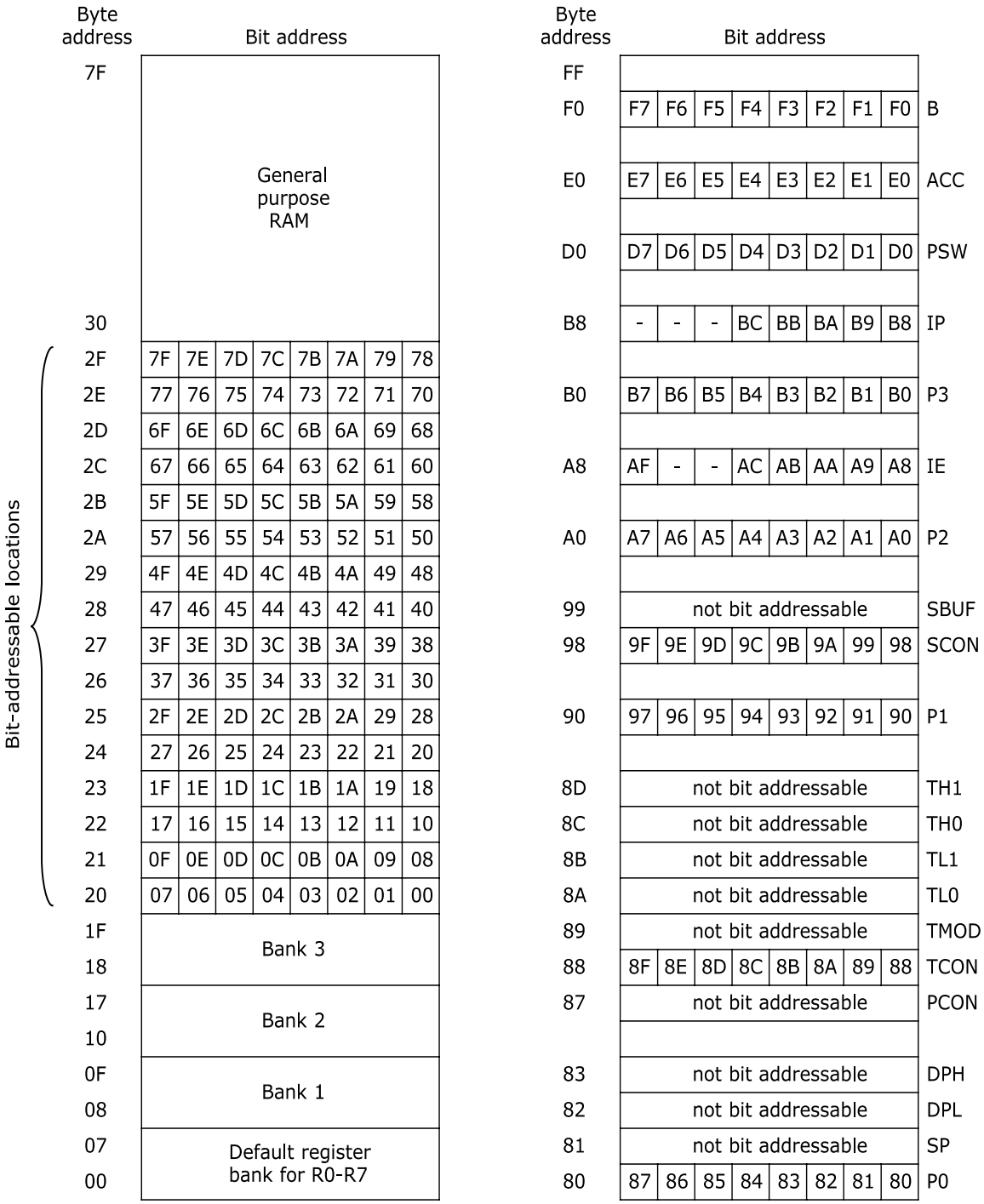
Type of Switch	Symbol	Description
Limit Switch	 NO	Normally Open
Limit Switch	 NOHC	Normally Open Held Closed
Limit Switch	 NC	Normally Closed
Limit Switch	 NCHO	Normally Closed Held Open

Type of Switch	Symbol	Description with an example
Timed Switch		 NOTC: Normally Open, Timed Closed Closes 5 seconds after coil is energized. Open immediately when coil is de-energized
Timed Switch		 NOTO: Normally Open, Timed Open Closes immediately after coil is energized. Open 5 seconds after coil is de-energized

Timed Switch	 NCTO	 NCTC: Normally Closed, Timed Open Open 5 seconds after coil is energized. Closes immediately when coil is de-energized
Timed Switch	 NCTC	 NCTC: Normally Closed, Timed Closed Opens immediately after coil is energized. Closes 5 seconds after coil is de-energized

Appendix-A:

Appendix: The 8051 RAM Memory Map



Appendix: 8051 Instruction Set

Hex Code	Bytes	Cycles	Instruction	
00	1	12	NOP	
01	2	2	AJMP	addr11
02	3	2	LJMP	addr16
03	1	1	RR	A
04	1	1	INC	A
05	2	1	INC	direct
06	1	1	INC	@R0
07	1	1	INC	@R1
08	1	1	INC	R0
09	1	1	INC	R1
0A	1	1	INC	R2
0B	1	1	INC	R3
0C	1	1	INC	R4
0D	1	1	INC	R5
0E	1	1	INC	R6
0F	1	1	INC	R7
10	3	2	JBC	bit, offset
11	2	2	ACALL	addr11
12	3	2	LCALL	addr16
13	1	1	RRC	A
14	1	1	DEC	A
15	2	1	DEC	direct
16	1	1	DEC	@R0
17	1	1	DEC	@R1
18	1	1	DEC	R0
19	1	1	DEC	R1
1A	1	1	DEC	R2
1B	1	1	DEC	R3
1C	1	1	DEC	R4
1D	1	1	DEC	R5
1E	1	1	DEC	R6
1F	1	1	DEC	R7
20	3	2	JB	bit, offset
21	2	2	AJMP	addr11
22	1	2	RET	
23	1	1	RL	A
24	2	1	ADD	A, #immed
25	2	1	ADD	A, direct
26	1	1	ADD	A, @R0
27	1	1	ADD	A, @R1
28	1	1	ADD	A, R0
29	1	1	ADD	A, R1
2A	1	1	ADD	A, R2
2B	1	1	ADD	A, R3

Hex Code	Bytes	Cycles	Instruction	
80	2	2	SJMP	offset
81	2	2	AJMP	addr11
82	2	2	ANL	C, bit
83	1	2	MOVC	A, @A+PC
84	1	4	DIV	AB
85	3	2	MOV	direct, direct
86	2	2	MOV	direct, @R0
87	2	2	MOV	direct, @R1
88	2	2	MOV	direct, R0
89	2	2	MOV	direct, R1
8A	2	2	MOV	direct, R2
8B	2	2	MOV	direct, R3
8C	2	2	MOV	direct, R4
8D	2	2	MOV	direct, R5
8E	2	2	MOV	direct, R6
8F	2	2	MOV	direct, R7
90	3	2	MOV	DPTR, #immed
91	2	2	ACALL	addr11
92	2	2	MOV	bit, C
93	1	2	MOVC	A, @A+DPTR
94	2	1	SUBB	A, #immed
95	2	1	SUBB	A, direct
96	1	1	SUBB	A, @R0
97	1	1	SUBB	A, @R1
98	1	1	SUBB	A, R0
99	1	1	SUBB	A, R1
9A	1	1	SUBB	A, R2
9B	1	1	SUBB	A, R3
9C	1	1	SUBB	A, R4
9D	1	1	SUBB	A, R5
9E	1	1	SUBB	A, R6
9F	1	1	SUBB	A, R7
A0	2	2	ORL	C, /bit
A1	2	2	AJMP	addr11
A2	2	1	MOV	C, bit
A3	1	2	INC	DPTR
A4	1	4	MUL	AB
A5				reserved
A6	2	2	MOV	@R0, direct
A7	2	2	MOV	@R1, direct
A8	2	2	MOV	R0, direct
A9	2	2	MOV	R1, direct
AA	2	2	MOV	R2, direct
AB	2	2	MOV	R3, direct

Hex Code	Bytes	Cycles	Instruction	
2C	1	1	ADD	A, R4
2D	1	1	ADD	A, R5
2E	1	1	ADD	A, R6
2F	1	1	ADD	A, R7
30	3	2	JNB	bit, offset
31	2	2	ACALL	addr11
32	1	2	RETI	
33	1	1	RLC	A
34	2	1	ADDC	A, #immed
35	2	1	ADDC	A, direct
36	1	1	ADDC	A, @R0
37	1	1	ADDC	A, @R1
38	1	1	ADDC	A, R0
39	1	1	ADDC	A, R1
3A	1	1	ADDC	A, R2
3B	1	1	ADDC	A, R3
3C	1	1	ADDC	A, R4
3D	1	1	ADDC	A, R5
3E	1	1	ADDC	A, R6
3F	1	1	ADDC	A, R7
40	2	2	JC	offset
41	2	2	AJMP	addr11
42	2	1	ORL	direct, A
43	3	2	ORL	direct, #immed
44	2	1	ORL	A, #immed
45	2	1	ORL	A, direct
46	1	1	ORL	A, @R0
47	1	1	ORL	A, @R1
48	1	1	ORL	A, R0
49	1	1	ORL	A, R1
4A	1	1	ORL	A, R2
4B	1	1	ORL	A, R3
4C	1	1	ORL	A, R4
4D	1	1	ORL	A, R5
4E	1	1	ORL	A, R6
4F	1	1	ORL	A, R7
50	2	2	JNC	offset
51	2	2	ACALL	addr11
52	2	1	ANL	direct, A
53	3	2	ANL	direct, #immed
54	2	1	ANL	A, #immed
55	2	1	ANL	A, direct
56	1	1	ANL	A, @R0
57	1	1	ANL	A, @R1
58	1	1	ANL	A, R0
59	1	1	ANL	A, R1

Hex Code	Bytes	Cycles	Instruction	
AC	2	2	MOV	R4, direct
AD	2	2	MOV	R5, direct
AE	2	2	MOV	R6, direct
AF	2	2	MOV	R7, direct
B0	2	2	ANL	C, /bit
B1	2	2	ACALL	addr11
B2	2	1	CPL	bit
B3	1	1	CPL	C
B4	3	2	CJNE	A, #immed, offset
B5	3	2	CJNE	A, direct, offset
B6	3	2	CJNE	@R0, #immed, offset
B7	3	2	CJNE	@R1, #immed, offset
B8	3	2	CJNE	R0, #immed, offset
B9	3	2	CJNE	R1, #immed, offset
BA	3	2	CJNE	R2, #immed, offset
BB	3	2	CJNE	R3, #immed, offset
BC	3	2	CJNE	R4, #immed, offset
BD	3	2	CJNE	R5, #immed, offset
BE	3	2	CJNE	R6, #immed, offset
BF	3	2	CJNE	R7, #immed, offset
C0	2	2	PUSH	direct
C1	2	2	AJMP	addr11
C2	2	1	CLR	bit
C3	1	1	CLR	C
C4	1	1	SWAP	A
C5	2	1	XCH	A, direct
C6	1	1	XCH	A, @R0
C7	1	1	XCH	A, @R1
C8	1	1	XCH	A, R0
C9	1	1	XCH	A, R1
CA	1	1	XCH	A, R2
CB	1	1	XCH	A, R3
CC	1	1	XCH	A, R4
CD	1	1	XCH	A, R5
CE	1	1	XCH	A, R6
CF	1	1	XCH	A, R7
D0	2	2	POP	direct
D1	2	2	ACALL	addr11
D2	2	1	SETB	bit
D3	1	1	SETB	C
D4	1	1	DA	A
D5	3	2	DJNZ	direct, offset
D6	1	1	XCHD	A, @R0
D7	1	1	XCHD	A, @R1
D8	2	2	DJNZ	R0, offset
D9	2	2	DJNZ	R1, offset

Hex Code	Bytes	Cycles	Instruction	
5A	1	1	ANL	A, R2
5B	1	1	ANL	A, R3
5C	1	1	ANL	A, R4
5D	1	1	ANL	A, R5
5E	1	1	ANL	A, R6
5F	1	1	ANL	A, R7
60	2	2	JZ	offset
61	2	2	AJMP	addr11
62	2	1	XRL	direct, A
63	3	2	XRL	direct, #immed
64	2	1	XRL	A, #immed
65	2	1	XRL	A, direct
66	1	1	XRL	A, @R0
67	1	1	XRL	A, @R1
68	1	1	XRL	A, R0
69	1	1	XRL	A, R1
6A	1	1	XRL	A, R2
6B	1	1	XRL	A, R3
6C	1	1	XRL	A, R4
6D	1	1	XRL	A, R5
6E	1	1	XRL	A, R6
6F	1	1	XRL	A, R7
70	2	2	JNZ	offset
71	2	2	ACALL	addr11
72	2	2	ORL	C, bit
73	1	2	JMP	@A+DPTR
74	2	1	MOV	A, #immed
75	3	2	MOV	direct, #immed
76	2	1	MOV	@R0, #immed
77	2	1	MOV	@R1, #immed
78	2	1	MOV	R0, #immed
79	2	1	MOV	R1, #immed
7A	2	1	MOV	R2, #immed
7B	2	1	MOV	R3, #immed
7C	2	1	MOV	R4, #immed
7D	2	1	MOV	R5, #immed
7E	2	1	MOV	R6, #immed
7F	2	1	MOV	R7, #immed

Hex Code	Bytes	Cycles	Instruction	
DA	2	2	DJNZ	R2, offset
DB	2	2	DJNZ	R3, offset
DC	2	2	DJNZ	R4, offset
DD	2	2	DJNZ	R5, offset
DE	2	2	DJNZ	R6, offset
DF	2	2	DJNZ	R7, offset
E0	1	2	MOVX	A, @DPTR
E1	2	2	AJMP	addr11
E2	1	2	MOVX	A, @R0
E3	1	2	MOVX	A, @R1
E4	1	1	CLR	A
E5	2	1	MOV	A, direct
E6	1	1	MOV	A, @R0
E7	1	1	MOV	A, @R1
E8	1	1	MOV	A, R0
E9	1	1	MOV	A, R1
EA	1	1	MOV	A, R2
EB	1	1	MOV	A, R3
EC	1	1	MOV	A, R4
ED	1	1	MOV	A, R5
EE	1	1	MOV	A, R6
EF	1	1	MOV	A, R7
F0	1	2	MOVX	@DPTR, A
F1	2	2	ACALL	addr11
F2	1	2	MOVX	@R0, A
F3	1	2	MOVX	@R1, A
F4	1	1	CPL	A
F5	2	1	MOV	direct, A
F6	1	1	MOV	@R0, A
F7	1	1	MOV	@R1, A
F8	1	1	MOV	R0, A
F9	1	1	MOV	R1, A
FA	1	1	MOV	R2, A
FB	1	1	MOV	R3, A
FC	1	1	MOV	R4, A
FD	1	1	MOV	R5, A
FE	1	1	MOV	R6, A
FF	1	1	MOV	R7, A