جامعة الملك فهد للبترول والمعادن
King Fahd University of Petroleum & Minerals

Department of Electrical Engineering

King Fahd University of Petroleum and Minerals
Dhahran - Saudi Arabia

Laboratory Manual

EE 390

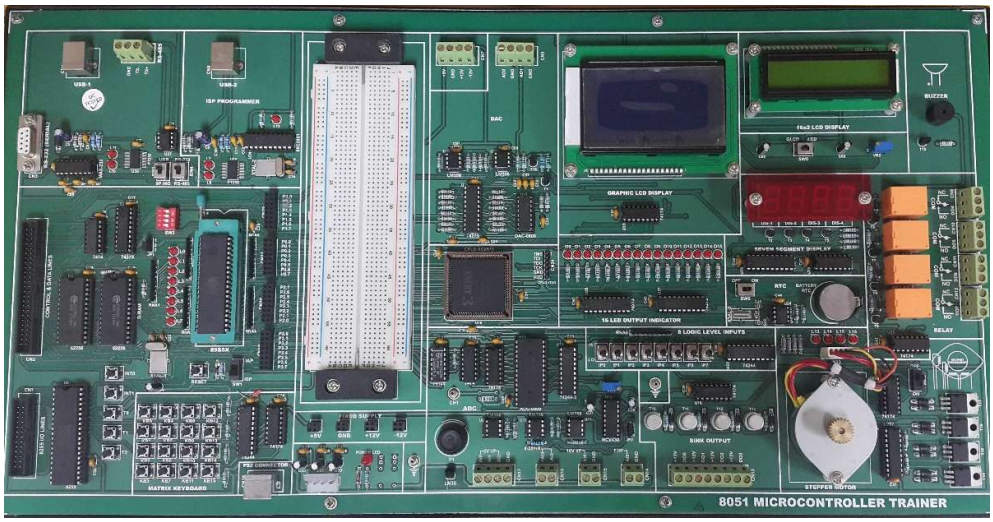


Table of Contents

Experiment #1 Introduction to ATMEGA Microcontroller (Arduino) 3

Experiment #2 Temperature Sensing System Using Arduino 7

Experiment #3 Introduction to Microcontroller 8051 and MIDE-51 ..... **Error! Bookmark not defined.**

Experiment #4 Data Transfer Instructions for Registers and Memory 30

Experiment #5 The 8051 Jump, Loop, and Call Instructions 40

Experiment #6 The 8051 Microcontroller Addressing Modes 45

Experiment #7 The 8051 Arithmetic Instructions 53

Experiment #8 Using Timers of 89S52 Microcontroller 57

Experiment #9 Introduction to C Compiler for 8051 Microcontroller 63

Experiment #10 Introduction to Interrupts of 89S52 73

Appendix A – 8051 Instruction Set 79

Appendix B – 8051 Special Function Registers 84

Appendix C – ASCII Table 85

Experiment #1:
Introduction to ATmega Microcontroller (Arduino)

1.0 Objective:

Write program in C-language, compile, download the program into Arduino memory, Execute to implement specific task.

1.1 Introduction:

In this DEMO experiment, you will be programming and using an "Arduino Uno" microcontroller board based on the ATmega328P microcontroller (8-bit AVR RISC-based microcontroller combines 32KB ISP flash memory, data sheet: http://www.atmel.com/devices/atmega328p.aspx). Arduino Uno board is shown in figure 1. Note that it has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

In this simple DEMO experiment, you will control LED's according to the input from a push-button switch. The programming includes using functions to assign input and output ports, checking the state of the input switch, sending logic-low and logic-high signals to LED's and implementing a DELAY.

Equipment:

Three 220 Ω resister, One 10 kΩ resister, Three LEDs, One push-button switch, Arduino board and connecting USB cable, one millimeter to check LEDs, switch and resistors.

1.2 Lab Work:

Part I: **Test Circuit**: Build the circuit given on Figure 2 using the Arduino Uno board. Use a 220-Ω resister, a push-button switch and an LED. Verify the output using the switch.

Part II: Program

- Use Arduino software to write the code given below.
- Compile the code and check for errors using "Verify" option.
- Implement the circuit given on Figure 3. Use 220-Ω resisters for pins 3, 4 and 5 and 10-KΩ resister for pin2 (of pin-7 if pin-2 is not usable)
- Now program the microcontroller using "Upload" option.
- Verify the output.

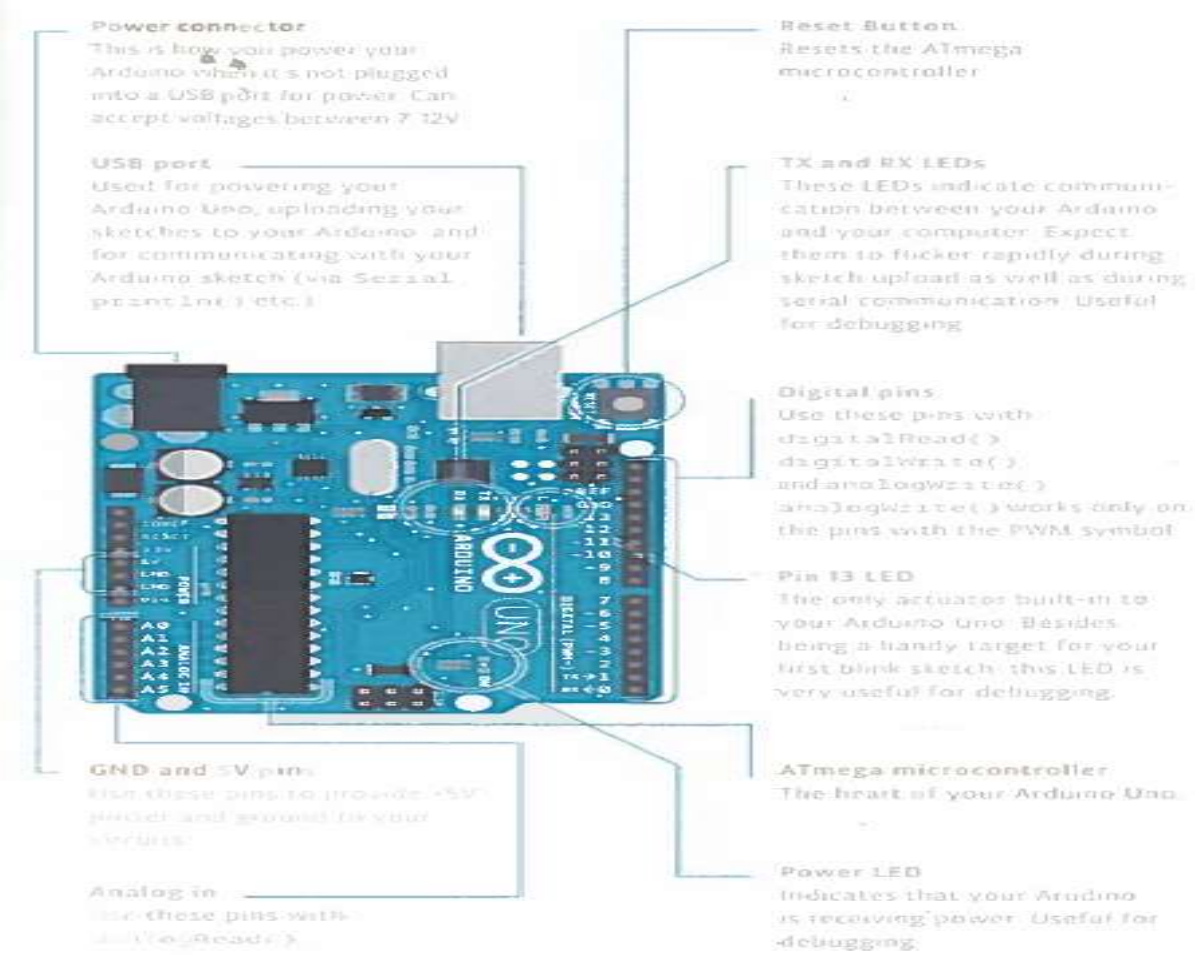


Figure 1: Arduino Uno Board

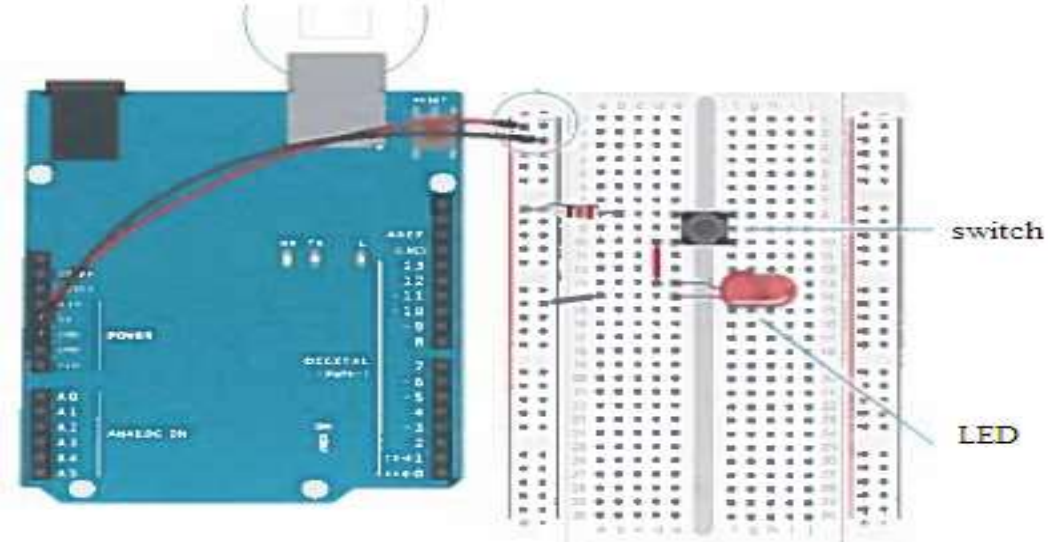


Figure 2: Validation circuit (interactive)

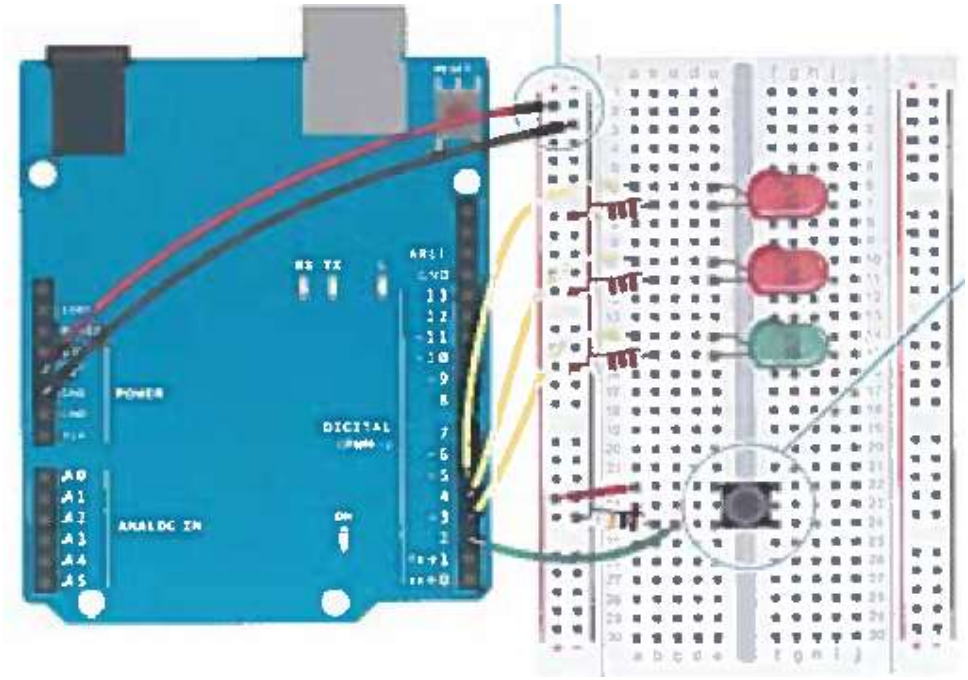


Figure 3: Circuit that will be used with following programs

Program:

```
          int switchState = 0;
          void setup(){
            pinMode(3,OUTPUT);
            pinMode(4,OUTPUT);
            pinMode(5,OUTPUT);
            pinMode(2,INPUT);  }
void loop(){
            switchState = digitalRead(2);
            if(switchState == LOW) {
              digitalWrite(3,HIGH);
              digitalWrite(4,HIGH);
              digitalWrite(5,LOW);  }
            else{
              digitalWrite(3,LOW);
              digitalWrite(4,LOW);
              digitalWrite(5,HIGH);
              delay(250);
              digitalWrite(4,HIGH);
              digitalWrite(5,LOW);
              delay(250);  }
              }
```

Part III: Using the circuit given on figure 3, execute the following code to implement a flashing light system for three lights. Use the steps defined in the previous part and verify the output using Arduino kit and three LED's.

```
int switchState = 0;
void setup(){
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(7,INPUT);
}
void loop(){
  switchState = digitalRead(7);
  if(switchState == LOW) {
    digitalWrite(3,HIGH);
    digitalWrite(4,LOW);
    digitalWrite(5,LOW);
    delay(200);
    digitalWrite(3,LOW);
    digitalWrite(4,HIGH);
    digitalWrite(5,LOW);
    delay(200);
    digitalWrite(3,LOW);
    digitalWrite(4,LOW);
    digitalWrite(5,HIGH);
    delay(200);
  }
  else{
digitalWrite(3,LOW);
digitalWrite(4,LOW);
digitalWrite(5,LOW);  } }
```

Part IV: Using the circuit given on figure 3 [page 34 (Fig. 1)], modify the code to: Define a variable "switchst" and initialize it with' LOW'. Replace all the 'LOW' in previous code by "switchst" and high by "~switchst". Verify output

```
int switchst = LOW;
digitalWrite(4,switchst) ……
```

Part V: Using the circuit given on figure 3, modify the code to: use the switch to change the direction of flashing light system. Your system should reverse the direction whenever it receives a HIGH from the switch.

1.3 Lab Report:

1) Objective, Introduction
2) Lab-work description with results
   i. Use the codes/programs to explain the function of the programs.
   ii. Application of such programs in real life devices.
3) Conclusion

Experiment #2:
Temperature Sensing System using Arduino

2.0 Objective:

Write program in C-language, compile, download the program into Arduino memory, execute to implement specific task.

2.1 Introduction:

In this experiment will use an "Arduino Uno" microcontroller board shown in figure 1. Arduino Uno board is based on the ATmega328P microcontroller (8-bit AVR RISC-based microcontroller combines 32KB ISP flash memory, data sheet: http://www.atmel.com/devices/atmega328p.aspx. Note that it has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with AC-to-DC adapter or battery to get started.
In this experiment, you will measure temperature of the room using temperature sensor (LM 35) and display it on computer screen. The programming includes functions to assign input and output ports, checking the state of the input switch, sending data to computer through serial communication and using if-else and for loops.

Equipment:

Three 220 Ω resister, One 10 kΩ resister, Three LEDs, One push-button switch, Temperature sensor (LM 35), Arduino board and connecting USB cable, One multi-meter to check LEDs, switch and resistors.

2.2 Lab Work:

Part I: Program

The following code implements a temperature sensing system.

- Use Arduino software to write the code given below.
- Compile the code and check for errors using "Verify" option.
- Implement the circuit given on Figure 2.
- Now program the microcontroller using "Upload" option.
- Verify the output. Use "TOOL➔_____" to see the display screen

Code:

```
const int sensorPin = A0;
const float baselineTemp = 24.0;          ➔ slightly change it to see effects
void setup() {
    Serial.begin(9600);
```

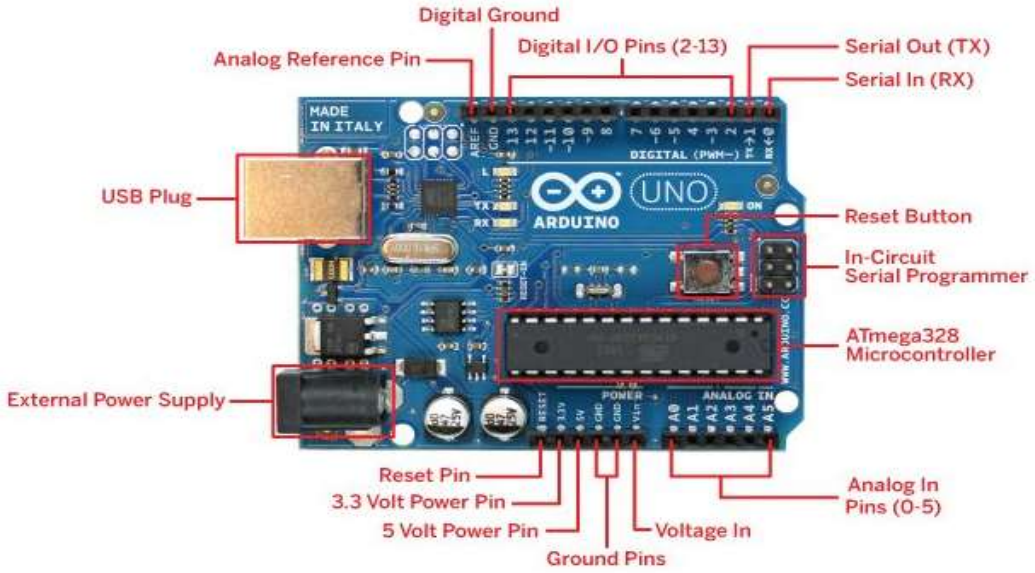


Figure 1: Arduino Uno Board

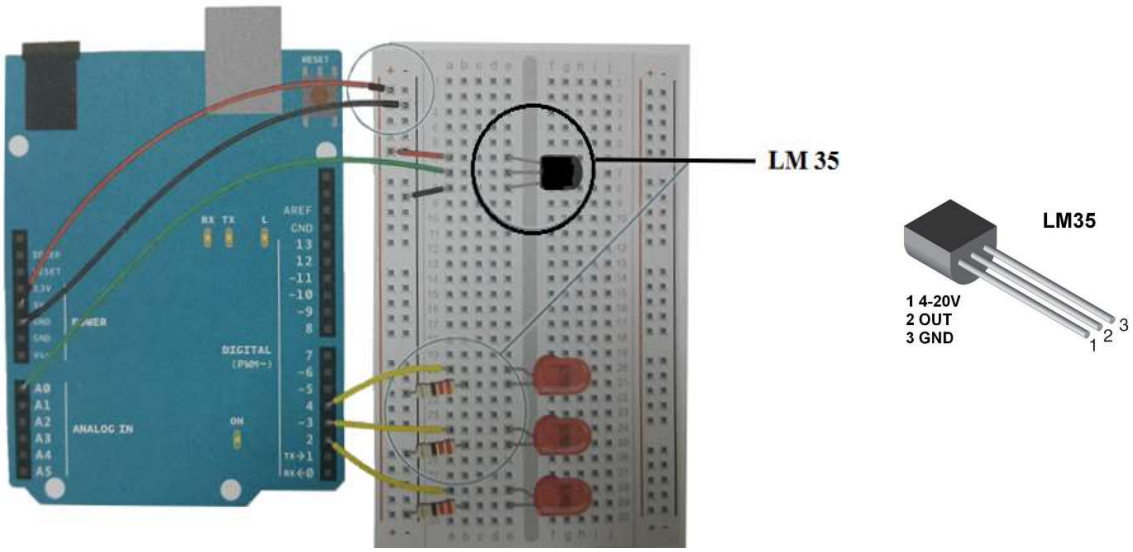


Figure 2: Validation circuit (interactive)

```
  for (int pinNumber = 2; pinNumber < 5; pinNumber++) {
      pinMode(pinNumber,OUTPUT);
      digitalWrite(pinNumber,LOW);  }
    }
void loop(){
      int sensorVal=analogRead(sensorPin);
      Serial.print("Sensor Value: ");
      Serial.print(sensorVal);
      float voltage = (sensorVal/1024.0)*5.0;
      Serial.print(",Volts: ");
```

```
        Serial.print(voltage);
        Serial.print(", degrees C: ");
        float temperature = (voltage)*100;
        Serial.println(temperature);

    if (temperature < baselineTemp){
        digitalWrite(2,LOW);
        digitalWrite(3,LOW);
        digitalWrite(4,LOW);

} else if (temperature >= baselineTemp && temperature < baselineTemp+2) {
        digitalWrite(2,HIGH);
        digitalWrite(3,LOW);
        digitalWrite(4,LOW);

} else if (temperature >= baselineTemp+2 && temperature < baselineTemp+4) {
        digitalWrite(2,HIGH);
        digitalWrite(3,HIGH);
        digitalWrite(4,LOW);
} else if (temperature >= baselineTemp+4) {
        digitalWrite(2,HIGH);
        digitalWrite(3,HIGH);
        digitalWrite(4,HIGH); }
        delay(100); }
```

Part II: Modify the above program to display the temperature in Fahrenheit scale. Use the following formula to convert Celsius to Fahrenheit. Use "TOOL➔_____" to see the display screen with the scrolling data.

$$F = \frac{9.0}{5.0} * C + 32$$

Part III: Modify the previous part. Connect a switch to port 5. The switch should perform the function of STOP/START. The temperature sensing system should stop once you press the switch. The system should restart if you press the switch again and so on. Display the message "SYSTEM STOPPED" when the system is in OFF state (***Hint:*** *you can use codes from last experiment to implement this part*)

2.3 Lab Report:

1) Objective, Introduction
2) Lab-work description with results
   i. Use the codes/programs to explain the function of the programs.
   ii. Application of such programs in real life devices.
3) Conclusion

Experiment #3:
Introduction to Microcontroller 8051 and MIDE-51

3.0 Objectives:

- Introducing the microcontroller 8051 family.
- Familiarization with the parts of basic module of 8051 trainer.
- Introducing the 8051 microcontroller trainer.
- Learning how to use 8051 PC-based debugger/simulator.
- Learning how to program the 8051 microcontroller trainer.

3.1 Material Required:

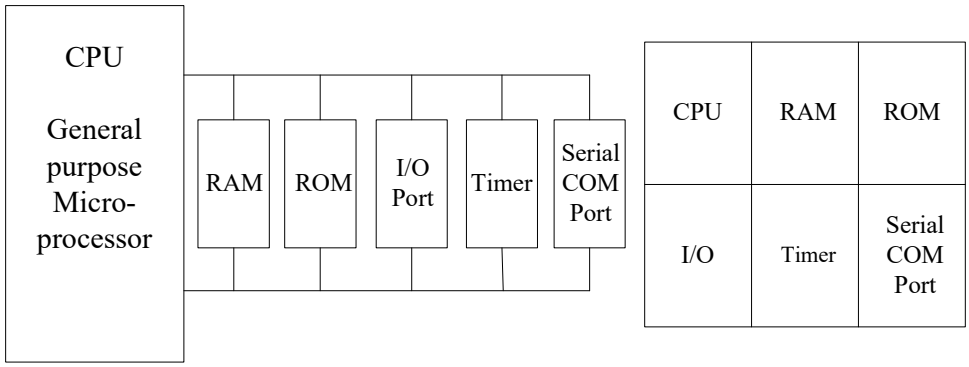
- 8051 microcontroller trainer (power supply and a serial data cable).
- 8051 PC-based debugger/simulator and programmer.

3.2 Introduction:

Microprocessor is a general-purpose CPU such as Intel's x86 family. Microprocessors contain no RAM, ROM or I/O ports on the chip itself. Although the addition of external RAM, ROM, and I/O ports makes these systems bulkier and much more expensive, they have the advantage of versatility such that the designer can decide on the amount of RAM, ROM, and I/O ports needed to fit the task at hand. A microcontroller is a special purpose processor. It has a CPU in addition to a fixed amount of RAM, ROM, I/O ports, and a timer all on one chip. This makes microcontroller ideal for many applications in which cost and space are critical. In many applications, like a TV remote control, there is no need for computing power of even an 8086 microprocessor. In such applications, the space it takes, the power it consumes, and the price per unit are much more critical considerations than the computing power. Figure 3.1 compares a microprocessor system with a microcontroller system.

Other microcontrollers integrate an ADC and other peripherals as well. Microcontrollers are frequently found in home appliances (remote control,

microwave oven, refrigerators, television and VCRs, stereos), computers and computer equipment (laser printers, modems, disk drives), cars (engine control, diagnostics, climate control), environmental control (greenhouse, factory, home), instrumentation, aerospace, and thousands of other uses. In many items, more than one microcontroller can be found. Microcontrollers come in many varieties. Depending on the power and features that are needed, one might choose a 4, 8, 16, or 32 bit microcontroller.



a. Microprocessor System b. Microcontroller System

Figure 3.1: Microprocessor System Contrasted with Microcontroller System

In 1981, Intel Corporation introduced the 8051 microcontroller. It is an 8-bit that has a total of four I/O ports, each 8 bits wide. It became widely popular after Intel allowed other manufacturers to make and market any flavor of the 8051 provided they remain code-compatible with the 8051. This has led to many versions of the 8051 with different speeds and amounts of resources however; a program written for one will run on any regardless of the manufacturer. Figure 3.2 shows a block diagram of the core 8051.

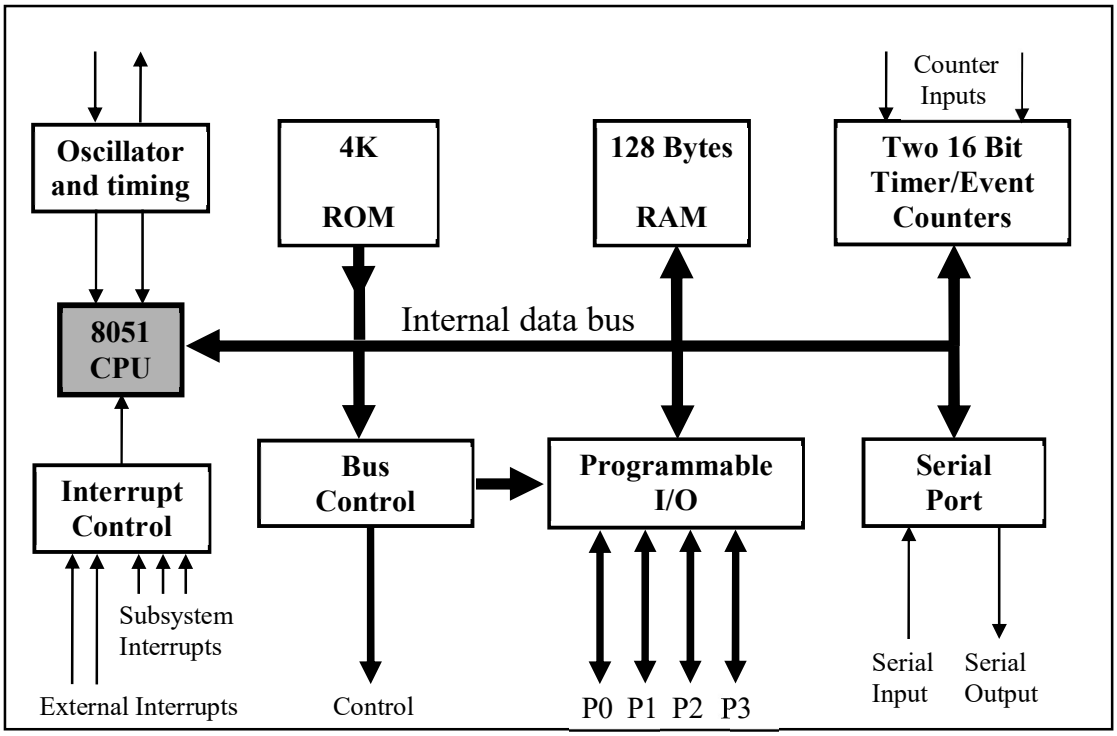


Figure 3.2: Block Diagram of the 8051 Core

3.3.1 Data Storage

The 8051 has 256 bytes of RAM on-chip. The lower 128 bytes are intended for internal data storage. The upper 128 bytes are the Special Function Registers (SFR). The lower 128 bytes are not to be used as standard RAM. They house the 8051's registers, its default stack area, and other features.

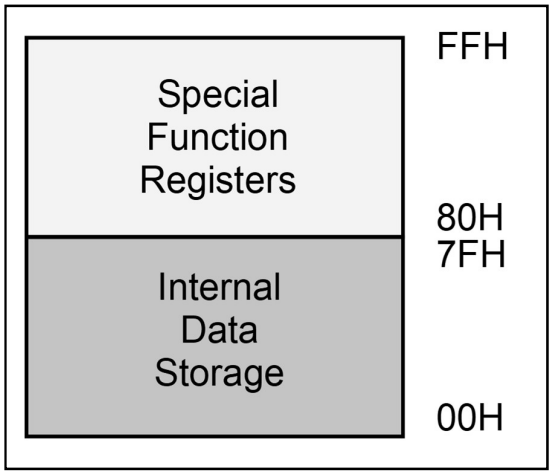


Figure 3.3: The 8051 RAM Memory

Register Banks

- The lowest 32 bytes of the on-chip RAM form 4 banks of 8 registers each.
- Only one of these banks can be active at any time.
- Bank is chosen by setting 2 bits in PSW
- Default bank (at power up) is bank 0 (locations 00 – 07).
- The 8 registers in any active bank are referred to as R0 through R7.

Given that each register has a specific address; it can be accessed directly using that address even if its bank is not the active one.

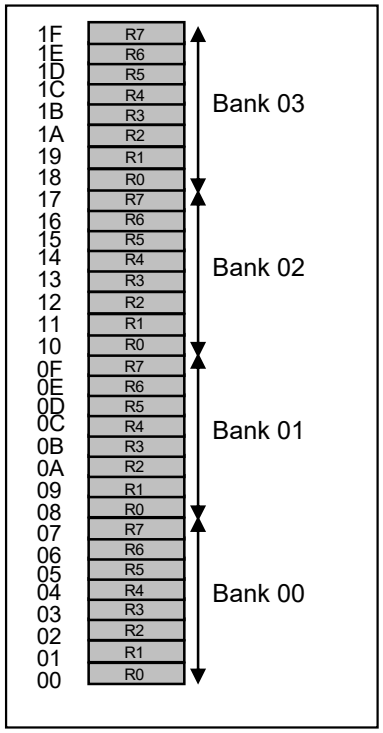


Figure 3.4: The 8051 Register Banks

3.3.2 Special Function Registers

The upper 128 bytes of the on-chip RAM are used to house special function registers. In reality, only about 25 of these bytes are actually used. The others are reserved for future versions of the 8051. These are registers associated with important functions in the operation of the 8051. Some of these registers are *bit-addressable* as well as *byte-addressable*. The address of bit 0 of the register will be the same as the address of the register.

- ACC and B registers – 8 bit each
- DPTR : [DPH:DPL] – 16 bit combined
- PC : Program Counter – 16 bits
- Stack pointer SP – 8 bit
- PSW : Program Status Word
- Port Latches
- Serial Data Buffer
- Timer Registers
- Control Registers

See the textbook a complete list of Special Function Registers and their addresses.

3.3.3 8051 Instructions

The 8051 has 255 instructions. Every 8-bit op-code from 00 to FF is used except for A5. The instructions are grouped into 5 groups:

- Arithmetic
- Logic
- Data Transfer
- Boolean
- Branching

The following table lists some of the commonly used instructions. See the textbook for a complete list of the 8051 instructions.

Instruction	Description	Execution Cycle
MOV A, B	Move the content from register B to register A	1-2
SETB bit-address	If bit-address=P0.1, the "SETB P0.1" will make 8051 to send 5 volt to "pin-0 of port 1" (or make it set)	1
CLR bit-address	Reset the 'bit-address' to 0	1
SJMP address	Jump to the physical address of memory location	2

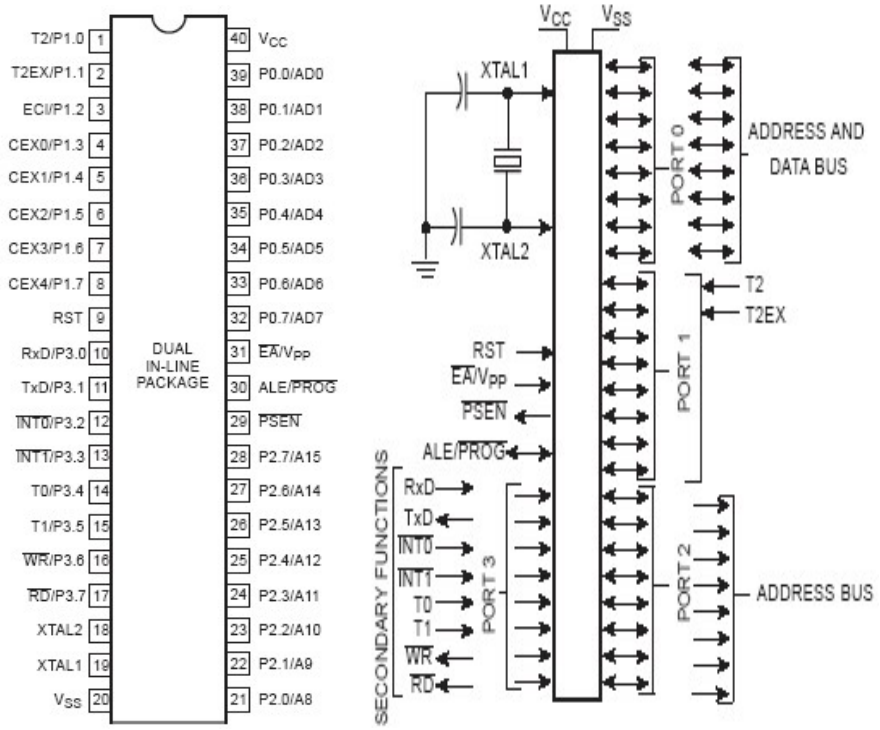
3.3.4 The 89S52 Microcontroller

The 89S52 is a low-power, high-performance CMOS 8-bit microcomputer with 8K bytes of Flash Memory. The device is compatible with the industry-standard MCS-51 instruction set and pin-out. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the 89S52 is a powerful microcomputer which provides a highly-flexible and cost-effective solution to many embedded control applications.

Features

- Compatible with MCS-51™ Products
- 8K Bytes of In-System Flash Memory
- Three-level Program Memory Lock

- 256 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Three 16-bit Timer/Counters
- Eight Interrupt Sources
- Full Duplex UART Serial Channel
- Low-power Idle and Power-down Modes
- Interrupt Recovery from Power-down Mode
- Watchdog Timer
- Dual Data Pointer
- Power-off Flag



Pin Configuration of 89S52 Logic Symbol of 89S52

Figure 3.5: Configurations of 89S52

The 89S52 comes in a 40 pin DIP package. The Pin Configuration of the 89S52 microcontroller is shown in Figure 3.5.

3.3 Lab Work:

In this lab, we will use the 89S52 microcontroller system designer board shown in Figure 3.6:

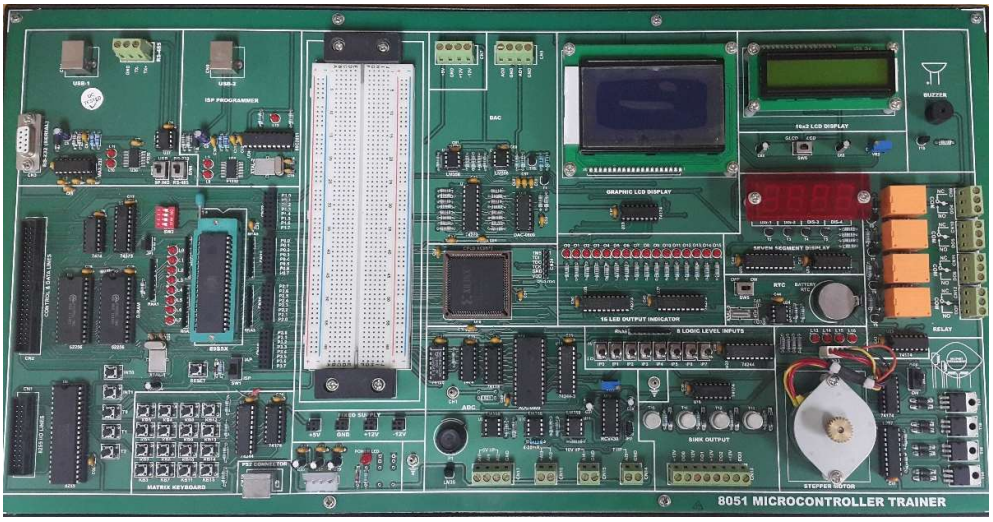


Figure 3.6: 89S52 Designer Board

This experiment will involve the following tasks:

1. Identify the different parts on the 89S52 microcontroller system development board. E.g.,
    a. Microcontroller chip
    b. Oscillator (also see frequency written on it)
    c. Power socket
    d. Temperature sensor (LM35)
    e. USB-2 ISP Programmer Connector
    f. Stepper motor
    g. Graphic LCD Display
    h. LCD Display

2. MIDE-51 is the 89S52 PC-based debugger/simulator that allows you to edit the program, assemble and debug it, and then you simulate the microcontroller operation. Follow the steps below to use the MIDE-51:

a. Open MIDE-51 program

b. Click File → New and write the following small test code in the window as shown in Figure 3.7:

```
        ORG 0000H
LOOP:   SETB P1.0      ; Set Port 1.0 high (1 m\c cycle)
        CLR P1.0       ; Clear Port 1.0 low (1 m\c cycle)
        SJMP LOOP      ; Jump to beginning (2 m\c cycles)
```

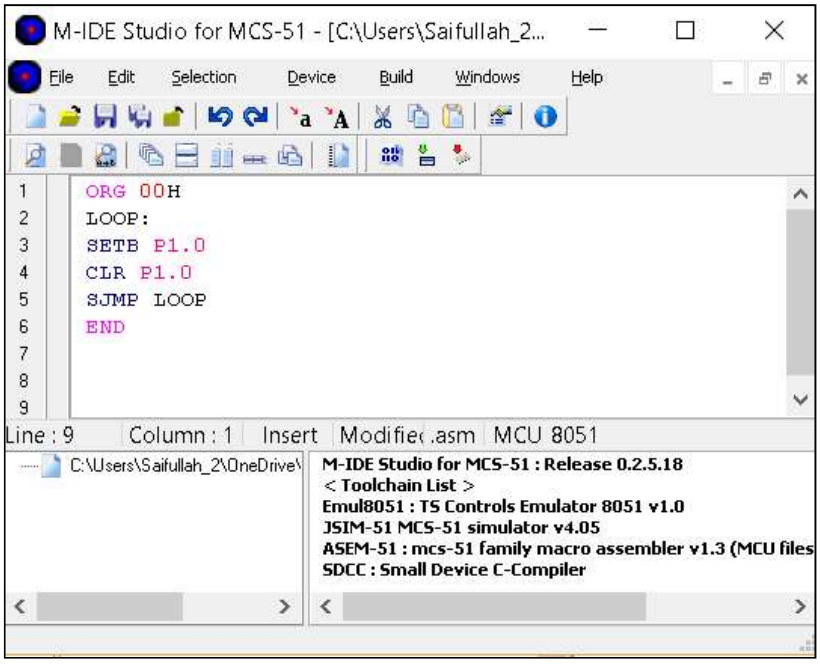


Figure 3.7: Writing Code in MIDE-51

c. Save the file as **EXP3.asm**. Then build and simulate as shown in Figure 3.8. This will generate **EXP3.hex** file.

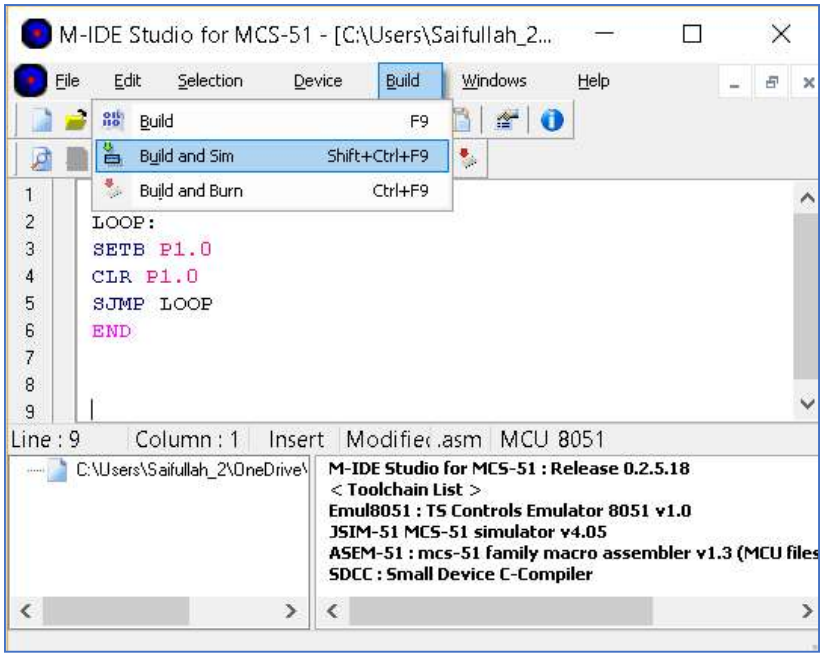


Figure 3.8: Building and Simulating the Code in MIDE-51

d. If there will be **0 error(s)** following window will open as shown in Figure 3.9.

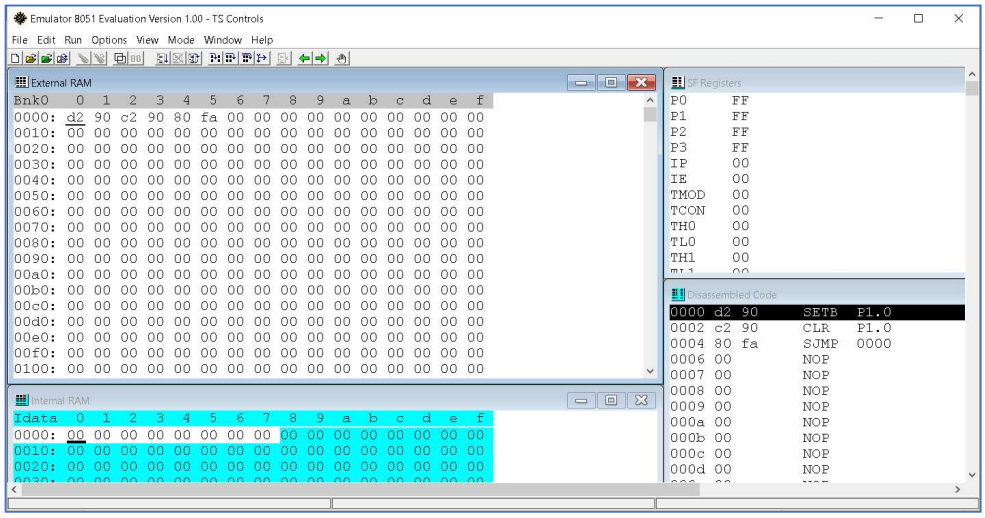


Figure 3.9: New window opening after building and simulating the Code in MIDE-51

e. Now click file and then go to configure memory system as shown below in Figure 3.10.

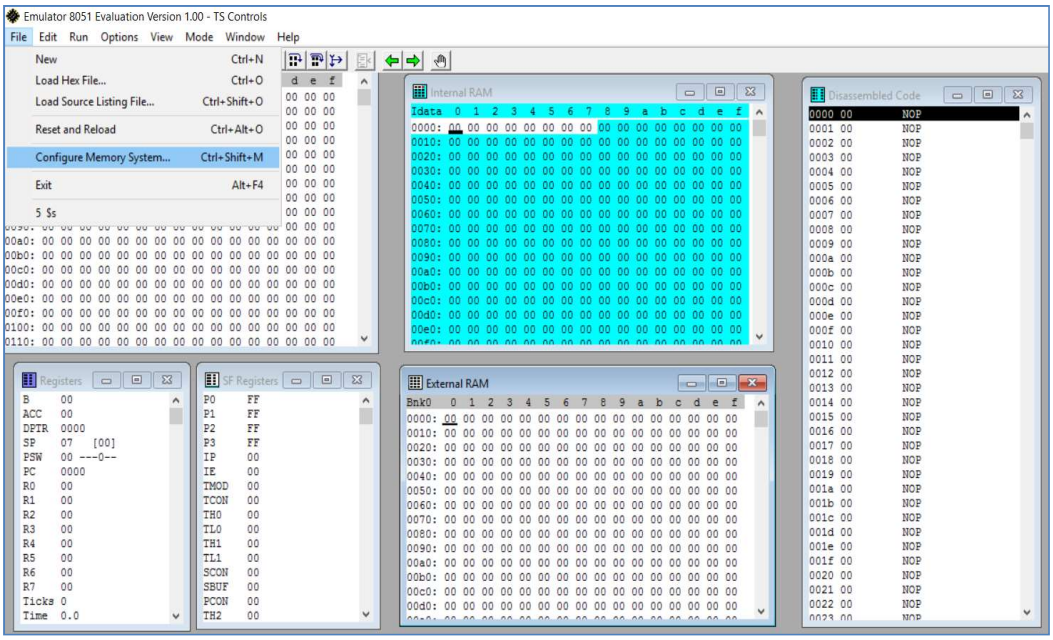


Figure 3.10: Configuring memory system in Emulator 8051 Evaluation

f. After pressing configure memory system in MIDE-51, following window will open as shown in Figure 3.11. Select tiny mode and then press OK to synchronize DB with external RAM.

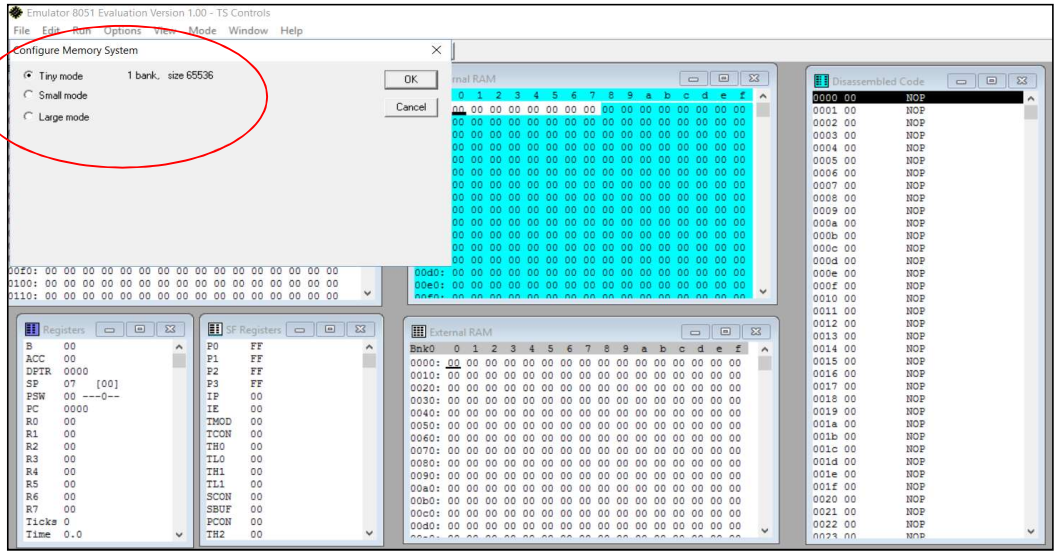


Figure 3.11: Selecting *'tiny mode'*

g. Now open the various windows by clicking “View” from the menu bar as shown in Figure 3.12 to get familiar with them. You may need these windows in future experiments.

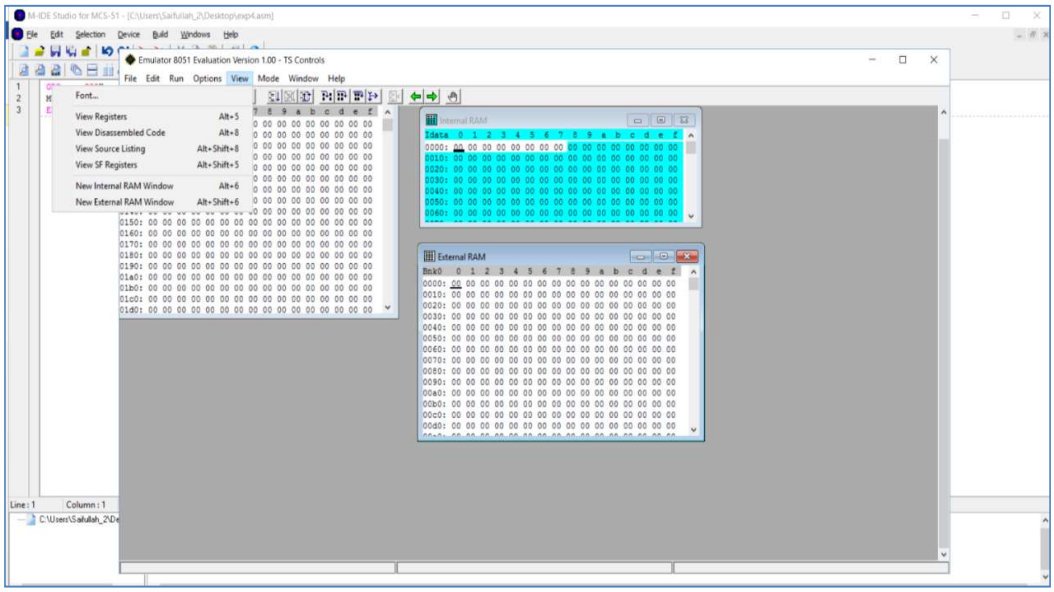


Figure 3.12: Opening Different Windows

h. An example of the functions of windows is shown in Figure 3.13. Identify different windows in the simulator. E.g., Windows for:

1. Registers
2. SF Registers
3. Internal Memory
4. External Memory
5. Disassembled Code

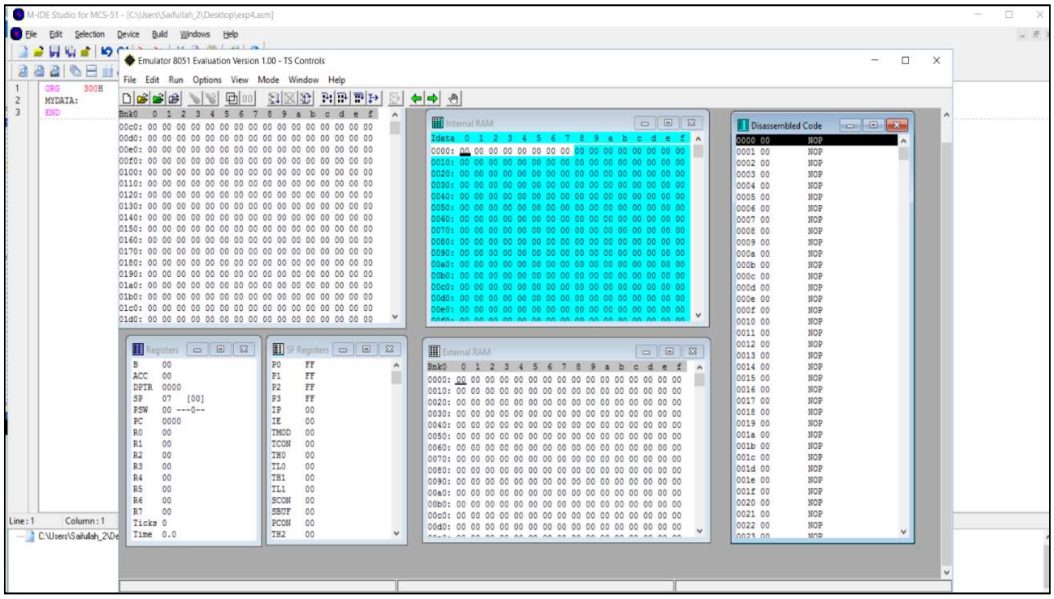


Figure 3.13: Windows Showing the Contents of Different Resources of the 8051

i. Run the program "line by line" by clicking "Run → Step" or directly pressing "F11" key.

j. Run the whole program by clicking "Run → Run" or directly by pressing "F5" key.

k. Program can be stopped by clicking "Run → **Stop**" or by pressing "Shift + F5".

3. Now if you are satisfied with the simulated results, then you can download and execute the code in the 89S52 microcontroller. This can be done by connecting the 89S52 board (power and data cords) and downloading the EXP3.hex code to the 89S52 microcontroller. Steps to follow for this process are:

a. Connect the USB cable at port USB-2 (ISP Programmer) on 89S52 Microcontroller Board as shown in Figure 3.14.

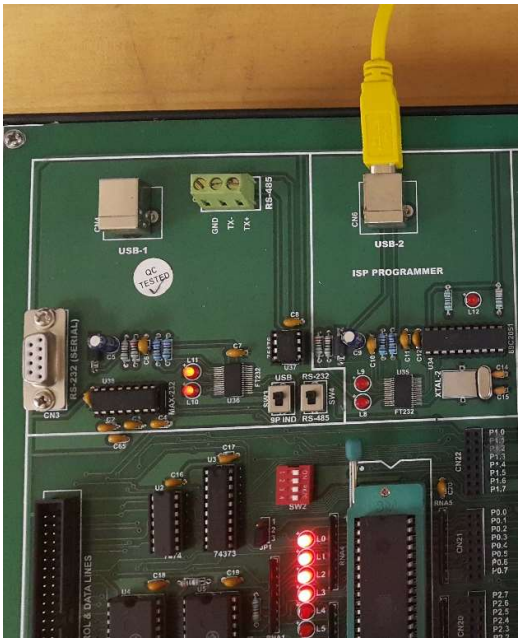


Figure 3.14: Connecting Cable at USB-2 Port (ISP Programmer)

b. Set the switch 1 (SW1) to ISP as shown in Figure 3.15.

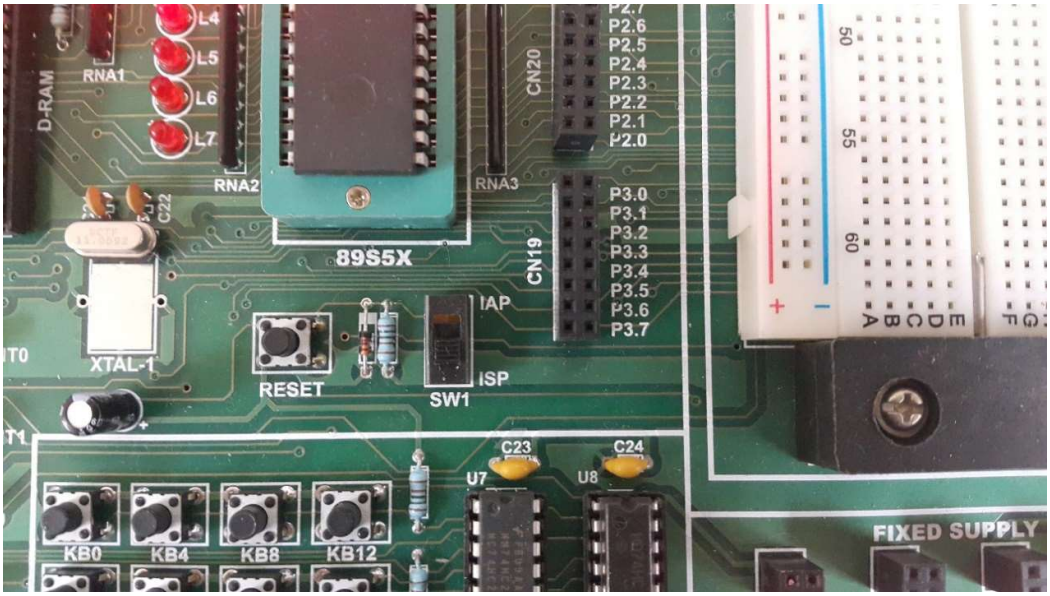


Figure 3.15: Setting SW1 to ISP

c. Open ISP_PROG v1.4 as shown in Figure 3.16.

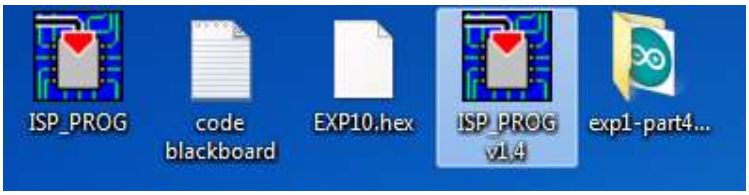


Figure 3.16: Open ISP_PROGv1.4

d. After opening ISP_PROGv1.4 a new window will open as shown in Figure 3.17. Then Press OK.

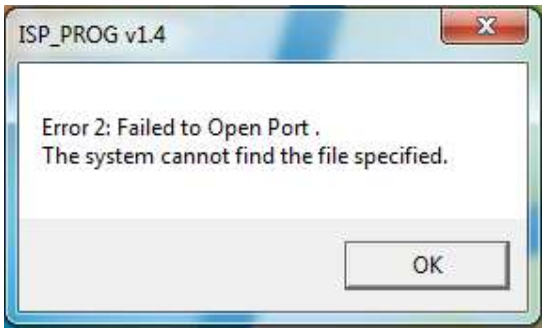


Figure 3.17: New window after opening ISP_PROGv1.4

e. A new window will open as shown in Figure 3.18. Set Target Clock to 11.0592.

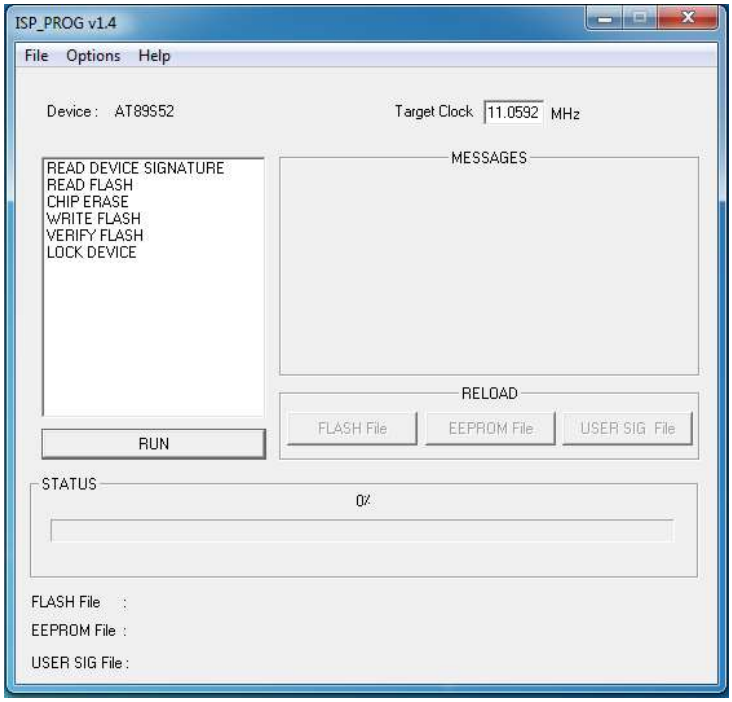


Figure 3.18: ISP_PROGv1.4 window

f. Go to Options ➔ Settings, a new window will open as shown in Figure 3.19.

Figure 3.19: PORT Settings window

g. Set Port to COM 2 (other ports can be selected).

h. Go to Options ➔ Device, a new window will open as shown in Figure 3.20. Select AT89S52.

Figure 3.20: Device Selection

i. Select "READ DEVICE SIGNATURE" and press RUN. If the device is not connected to correct port, an error window will open as shown in Figure 3.21. Then press OK. If no errors are found go to Step "n".

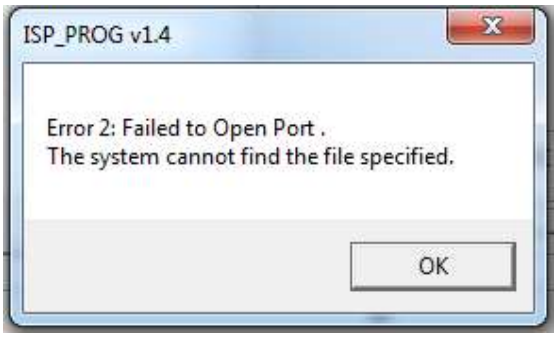


Figure 3.21: Error Window

j. Now, go to Computer ➔ Manage as shown in Figure 3.22. A new window will open as shown in Figure 3.23.

Figure 3.22: Manage settings of Computer

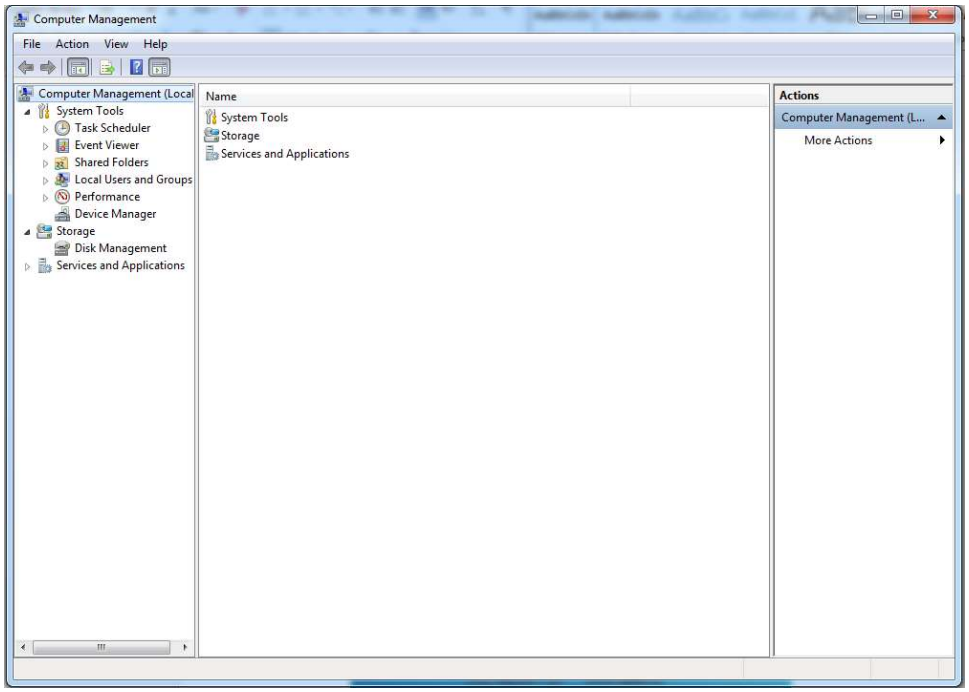


Figure 3.23: Computer Management Window

k. Go to Device Manager ➔ Ports (COM & LPT) ➔ USB Serial Port (…) as shown in Figure 3.24. Then Go to Port Settings ➔ Advanced as shown in Figure 3.25.

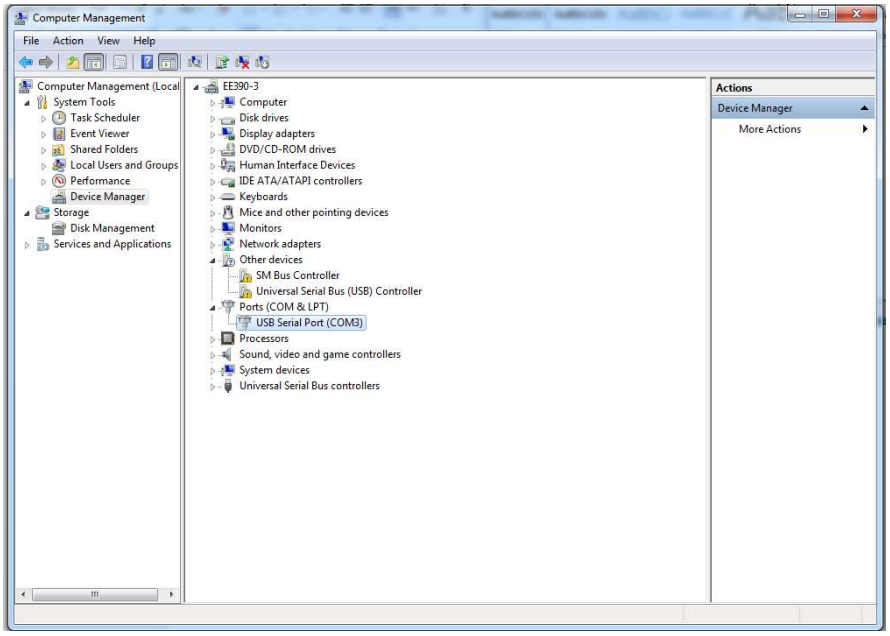


Figure 3.24: USB Serial Port Settings

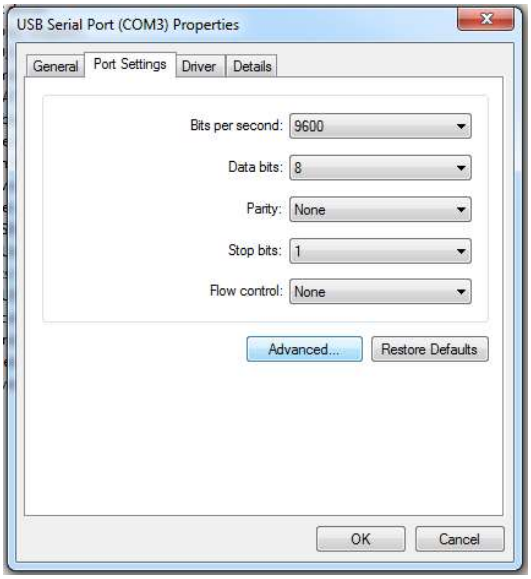


Figure 3.25: USB Serial Port Properties

l. Select COM2 and press OK as shown in Figure 3.26.

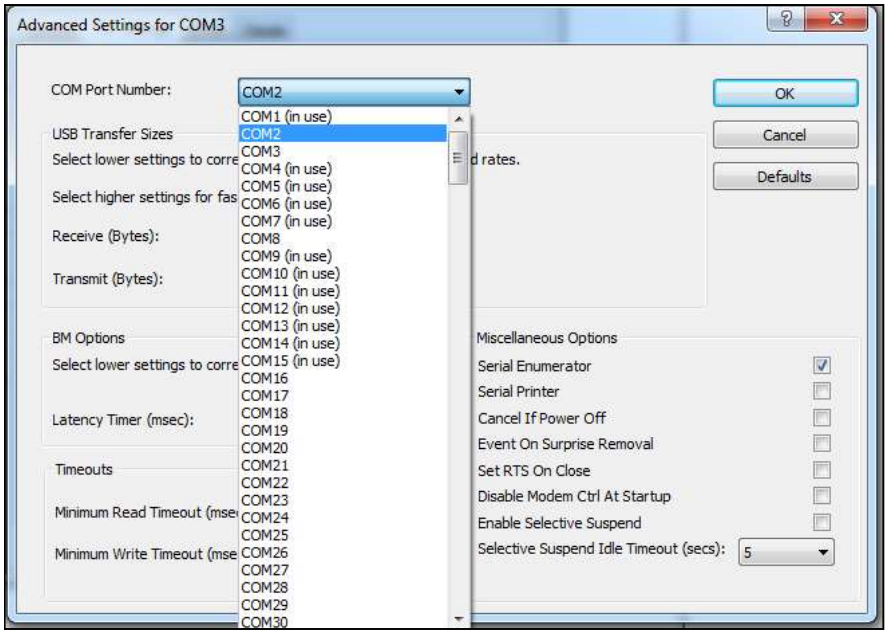


Figure 3.26: Selecting COM2

m. Now, repeat Step "i". Signature: 0x1E 0x52 message will be printed as shown in Figure 3.27.

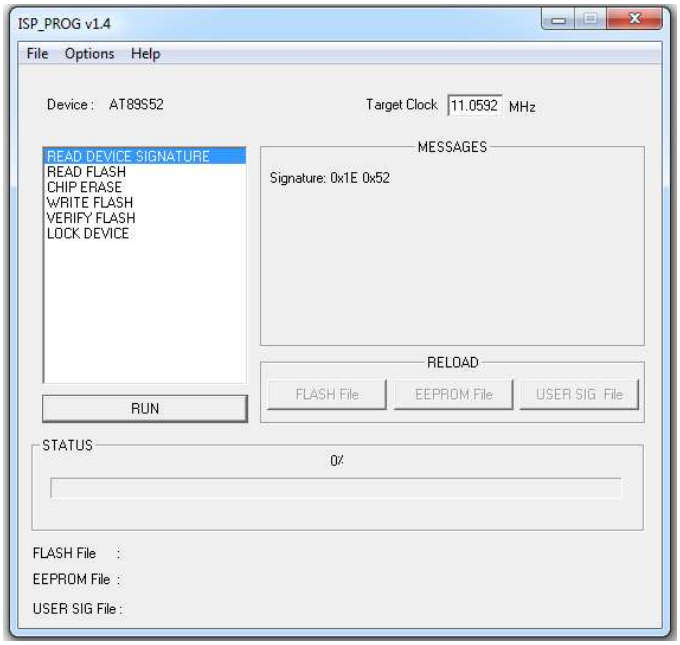


Figure 3.27: Reading Device Signature

n. Now go to File ➔ Load Flash File. Then select EXP3.hex file.

o. Select options and press run as shown in Figure 3.28.

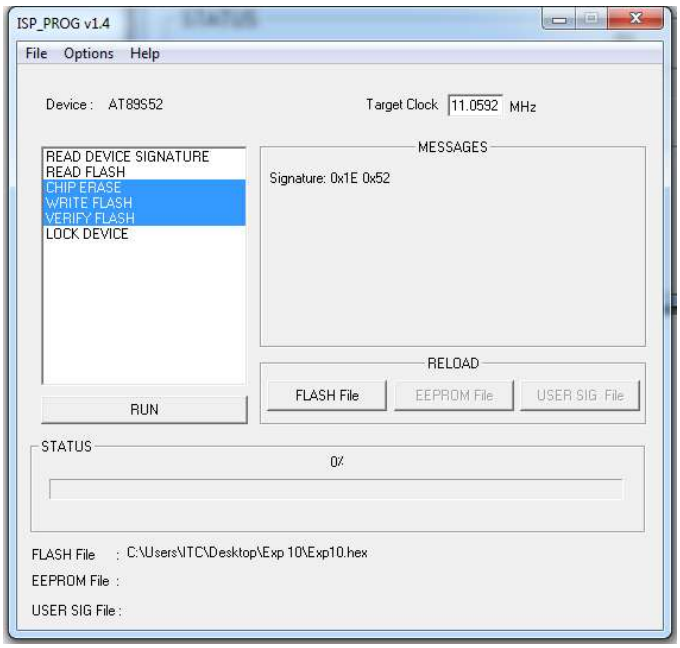


Figure 3.28: Downloading Hex File

p. Check the output using oscilloscope after status becomes 100% as shown in Figure 3.29.

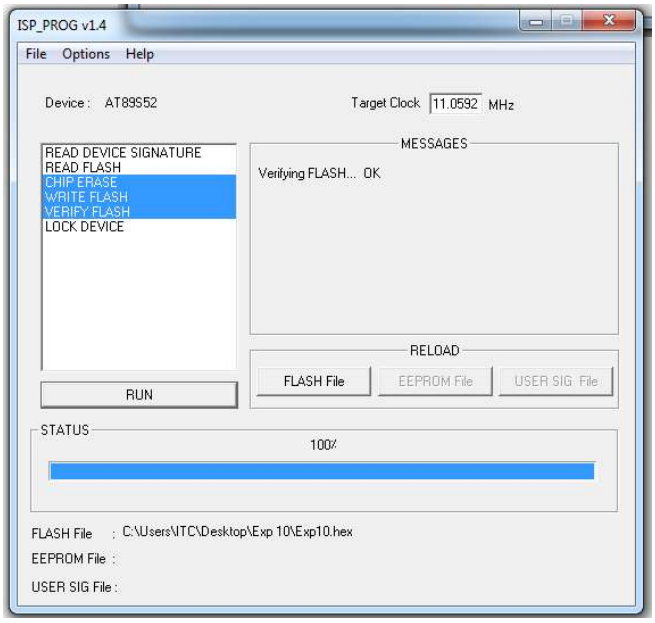


Figure 3.29: Download Completion Window

4. Connect the probe of oscilloscope at the proper output port (P1.0 in the above example). Don't forget to connect the black connector of oscilloscope's probe to GND of the 89S52 trainer board.

5. You should get a waveform like the one shown in Figure 3.30. Use the oscilloscope to measure the period of the square wave (T). Also measure the time related to "logic 1". These two values will allow you to compute duty-cycle. (Note that all these values can be directly recorded from oscilloscope)

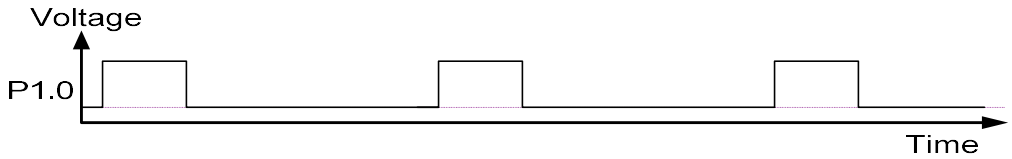
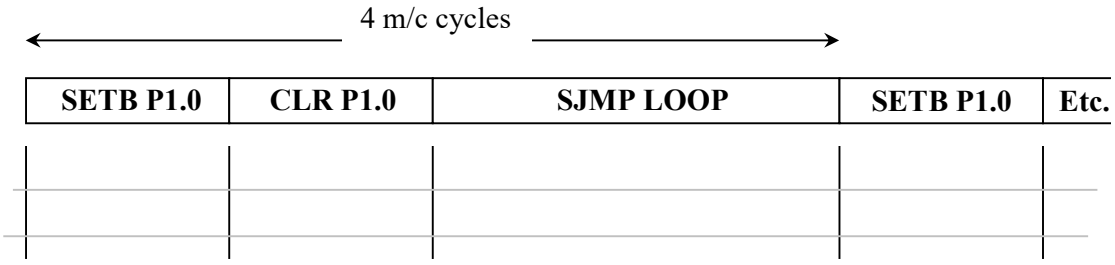


Figure 3.30: Output Waveform of the Microcontroller on the Oscilloscope

3. 4 Lab Report:

1. Abstract, Introduction, lab programs/work with results and conclusion
2. Calculate the frequency and duty cycle for the waveform you have obtained.
3. Draw the waveform as shown below according to the execution of instructions.



Experiment #4:
Data Transfer Instructions for Registers and Memory

4.0 Objective:

To study the data transfer instructions of the 8051 microcontroller, and write simple assembly programs in MIDE-51.

4.1 Introduction:

The process of writing program for the microcontroller mainly consists of giving instructions (commands) in the specific order in which they should be executed in order to carry out a specific task. All commands are known as INSTRUCTION SET. All microcontrollers compatible with the 8051 have in total of 255 instructions, i.e. 255 different words available for program writing.

4.2 Registers:

In the CPU, registers are used to store information temporarily. That information could be of a byte of data to be processed, or an address pointing to the data to be fetched. The vast majority of 8051 registers are 8-bit registers. The most widely used registers of the 8051 are A (accumulator), B, R0, R1, R2, R3, R4, R5, R6, R7, DPTR (data pointer), and PC (program counter). All of the above registers are 8-bit, except DPTR and the program counter. The accumulator, register A, is used for all arithmetic and logic instructions. To understand the use of these registers, we will use simple instructions, MOV or MOVX and ADD.

1. The MOV Instruction

The MOV instruction is a data transfer instruction, and copies data from one location to another. It has the following format:

```
MOV destination, source      ; copy source to destination.
```

This instruction tells the CPU to move (in reality, copy) the source operand to the destination operand.

```
MOV A, #66H        ; load value 66H into reg. A
MOV R0, A          ; copy contents of A into R0
MOV R2, A          ; copy contents of A into R2
MOV R1, #99H       ; load value 99H into R1
MOV A, R3          ; copy contents of R3 into A
```

When programming the 8051 microcontroller, the following points should be noted:

1. Values can be loaded directly into any of registers A, B, or R0 – R7. However, to indicate that it is an immediate value it must be preceded with a pound sign (#). For example the instruction "MOV A, #23H" and "MOV R5, #0F9H". Notice in instruction "MOV R5, #0F9H" that a 0 is used between the # and F to indicate that F is a hex number and not a letter. In other words "MOV R5, #F9H" will cause an error.

2. If values 0 to F are moved into an 8-bit register, the rest of the bits are assumed to be all zeros. For example, in the instruction "MOV A, #5″ the result will be A = 05; that is, A = 00000101 in binary.

3. Moving a value that is too large into a register will cause an error. For example the instruction "MOV A, #7F2H" is illegal.

2. The ADD Instruction

The ADD instruction is an arithmetic instruction. It has the following format: 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. The source operand can be either a register or immediate data, but the destination must always be the accumulator (Register A).

For example the following instructions add the contents of register A and register R2 and put the result in the accumulator.

```
MOV A, #25H      ; load 25H into A
MOV R2, #34H     ; load 34H into R2
ADD A, R2        ; add R2 to accumulator
```

To add two numbers, such as 20H and 30H, each can be moved to a register and then added together

```
MOV A, #20H
MOV R3, #30H
ADD A, R3
```

4.3 8051 Data types and Directives:

Pseudo instructions are not translated to machine code, and therefore they have no operation code. They are used by the assembler to organize the source code file (.asm file). There are several pseudo instructions supported by the 8051 microcontroller. Some of the widely used pseudo instructions are discussed below.

1. ORG (Origin)

The ORG pseudo instruction 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.

2. DB (Define Byte)

The DB pseudo instruction is the most widely used data instruction in the assembler. It is used to define the 8-bit data. When DB is used to define data, the numbers can be in decimal, binary, hex, or ASCII formats. For decimal, the "D" after the decimal number is optional, but using "B" (binary) and "H" (hexadecimal) for the others is required. Regardless of which is used, the assembler will convert the numbers into hex. To indicate ASCII, simply place the characters in quotation marks ('like this'). The assembler will assign the ASCII code for the numbers or characters automatically. Either single or double quotation marks can be used around the ASCII strings. Following are some DB examples:

```
        ORG   200H
DATA1:  DB    28              ; DECIMAL
DATA2:  DB    00110101B       ; BINARY
DATA3:  DB    39H             ; HEX
DATA4:  DB    218H            ; ASCII NUMBERS
DATA6:  DB    'KFUPM'         ; ASCII CHARACTERS
```

3. EQU (Equate)

This is used to define a constant without occupying a memory location. The EQU pseudo instruction does not set aside storage for a data item but associates a constant value with a data label so that when the label appears in the program, its constant value will be substituted for the label. The following uses EQU for the counter constant and then the constant is used to load the R3 register.

```
COUNT   EQU 20
MOV     R3, #COUNT
```

When executing the instruction "MOV R3, #COUNT", the register R3 will be loaded with the value 20 (notice the # sign). What is the advantage of using EQU? Assume that there is a constant (a fixed value) used in many different places in the program, and the programmer wants to change its value throughout. By the use of EQU, the programmer can change it once and the assembler will change all of its occurrences, rather than search the entire program trying to find every occurrence.

4. END

Another important pseudo instruction is the END directive. This indicates to the assembler the end of the source (.asm) file. The END pseudo instruction is the last line

of an 8051 program, meaning that in the source code anything after the END directive is ignored by the assembler.

4.4 Structure of the 8051 Assembly Language Programs:

An Assembly language program consists of, among other things, a series of lines of Assembly language instructions. An Assembly language instruction consists of a mnemonic, optionally followed by one or two operands. The operands are the data items being manipulated, and the mnemonics are the commands to the CPU, telling it what to do with those items.

```
ORG 0H
MOV R4, #20H
MOV R6, #30H
MOV A, #0
ADD A, R4
ADD A, R7
ADD A, #12H
END
```

A given Assembly language program is a series of statements, or lines, which are either Assembly language instructions such as ADD and MOV, or statements called pseudo instructions. While instructions tell the CPU what to do, pseudo-instructions (also called directives) give directions to the assembler. For example, in the above program while the MOV and ADD instructions are commands to the CPU, ORG and END are directives to the assembler. ORG tells the assembler to place the opcode at memory location 0 while END indicates to the assembler the end of the source code. In other words, one is for the start of the program and the other one for the end of the program.

4.5 Program Status Word register:

The program status word (PSW) register is an 8-bit register. It is also referred to as the flag register. Although the PSW register is 8 bit wide, only 6 bits of it are used by the 8051. The two unused bits are user definable flags. Four of the flags are called conditional flags, meaning that they indicate some conditions that resulted after an instruction was executed. These four are CY (Carry), AC (auxiliary carry), P (parity), and OV (Overflow).

CY	AC	F0	RS1	RS0	OV	--	P

CY PSW.7 Carry flag
AC PSW.6 Auxiliary carry flag
-- PSW.5 Available to the user for general purpose.
RS1 PSW.4 Register bank selector bit 1
RS0 PSW.3 Register bank selector bit 0
OV PSW.2 Overflow flag
-- PSW.1 User definable bit
P PSW.0 Parity flag

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

Figure 4.1: Program Status Word Register

4.6 Pre Lab:

1. Write a program for 8051 microcontroller to add the values 9CH and 64H and put the result in register A. Show the status of the carry flag (C) , auxiliary carry flag (AC), and the parity flag (P) after program execution.

2. State the contents of memory locations 300H-305H for the following

```
        ORG   300H
MYDATA: DB    "ADE123"
        END
```

4.7 Lab Work:

1. Write a program using 'DB' directive to reserve 10_D bytes of memory space with variable name 'num1' in ROM. Then initializes these memory locations with the hex number: B_H. Finaly, the program should equate variable 'num2' to data: 23H.

2. Write a program for 8051 that reserves in RAM a memory space (10H-1FH) of 16_D bytes and initializes it with the hex number: A_H. The program should copy the 16_D bytes of data into another memory location 30H-3F_H using MOV instruction. Verify the program using MIDE-51

3. Initialize the memory locations 30H-36H with the decimal number: 10_D. Use ADD command to add the contents stored at these RAM memory locations (30 H -36H). Store the result in R2. Show the status of the carry flag (C), auxiliary carry flag (AC), and the parity flag (P) after program execution.

In these experiments, MIDE-51 simulator will be used that allows you to write the program, assemble and debug it. Follow the steps below to use the MIDE-51:

a. Open MIDE-51 program (you can download it from Blackboard)

b. Click File → New. Write the following small test code in the window as shown in Fig 4.2:

```
        ORG   300H
MYDATA: DB    "ADE123"
        END
```

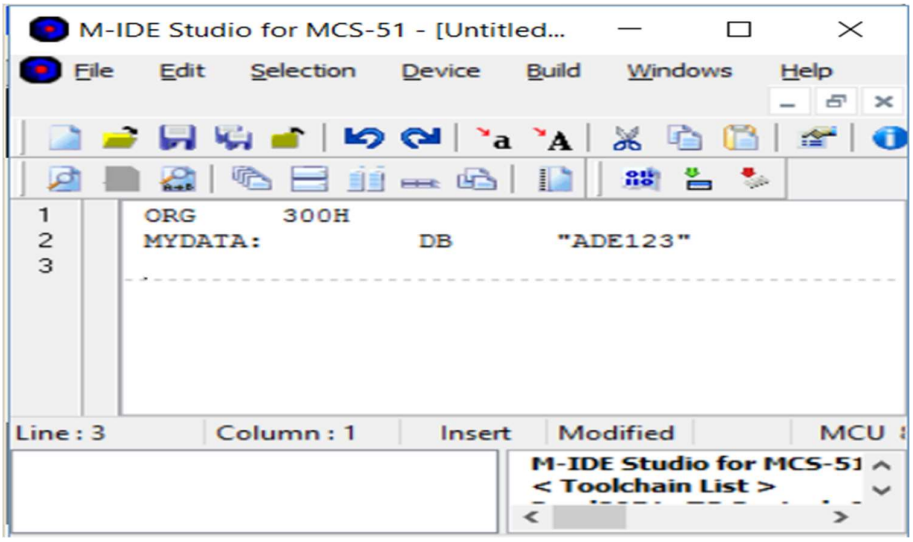


Figure 4.2: Writing Code in MIDE-51

c. Save the file as **exp4.asm**. Then build and simulate as shown in Figure 4.3.

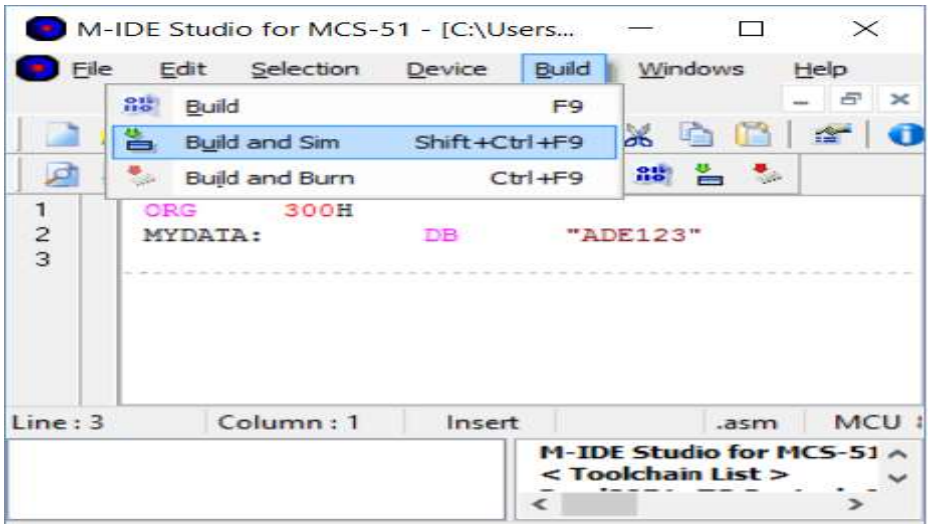


Figure 4.3: Building and Simulating the Code in MIDE-51

d. Building and Simulating the Code in MIDE-51

e. If there will be 0 error(s) following window will open as shown in Figure 4.4.

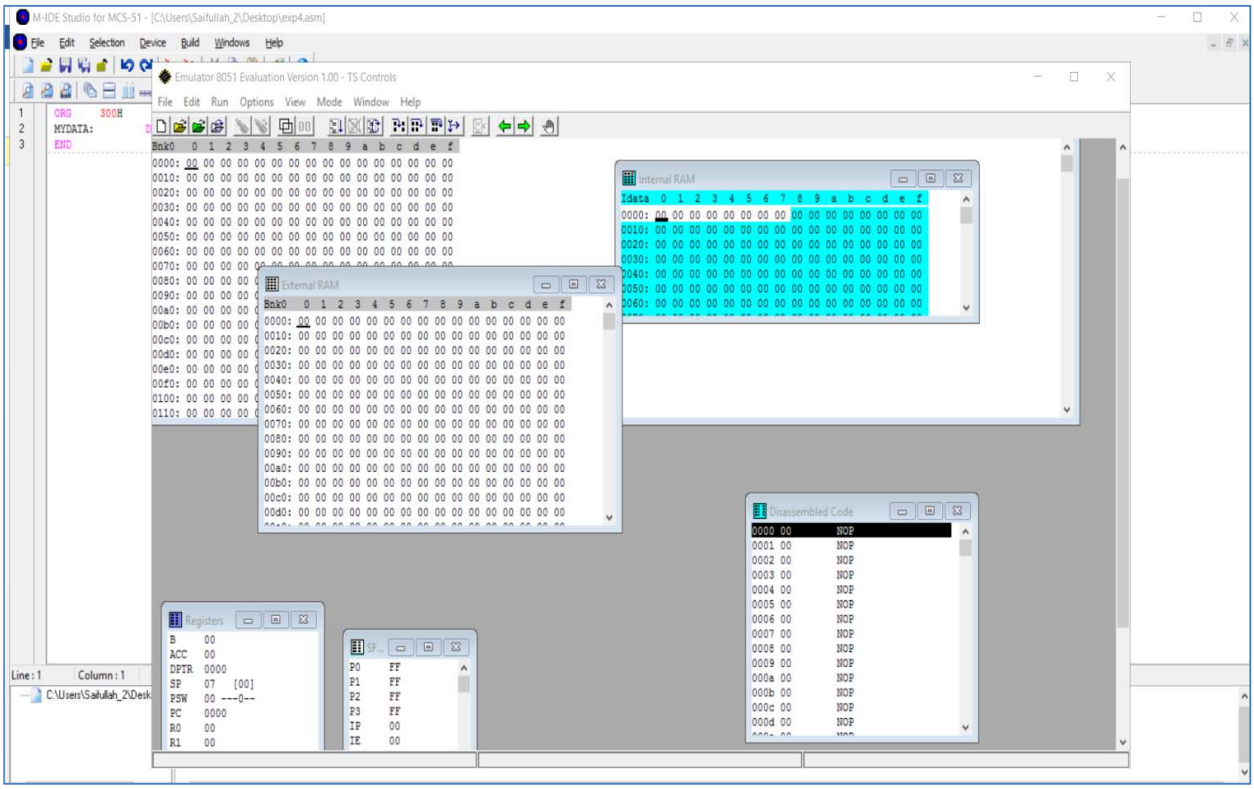


Figure 4.4: New window opening after building and simulating the Code in MIDE-51

f. Now click file and then go to configure memory system as shown below in Figure 4.5.

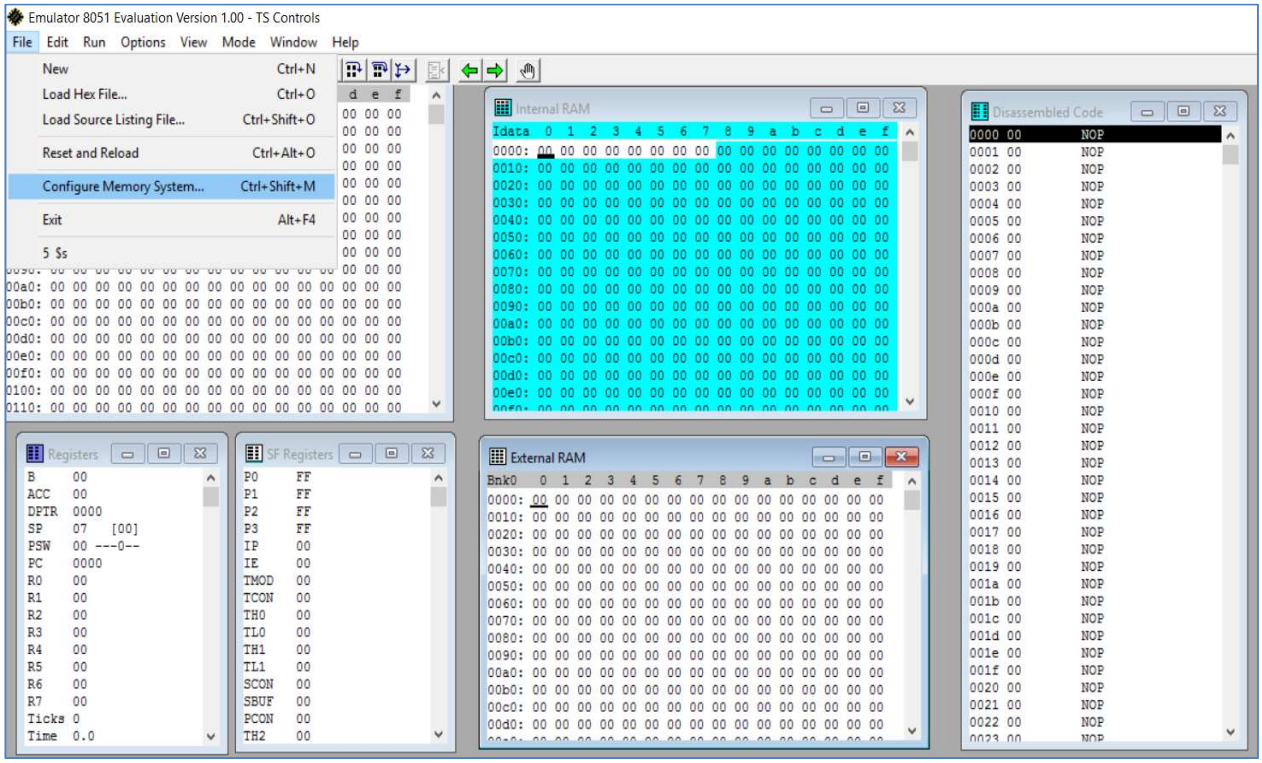


Figure 4.5: Configuring memory system in Emulator 8051 Evaluation

g. After pressing configure memory system **in MIDE-51**, following window will open as shown in Figure 4.6. Select **tiny mode** and then press OK to synchronize DB with external RAM (ROM).

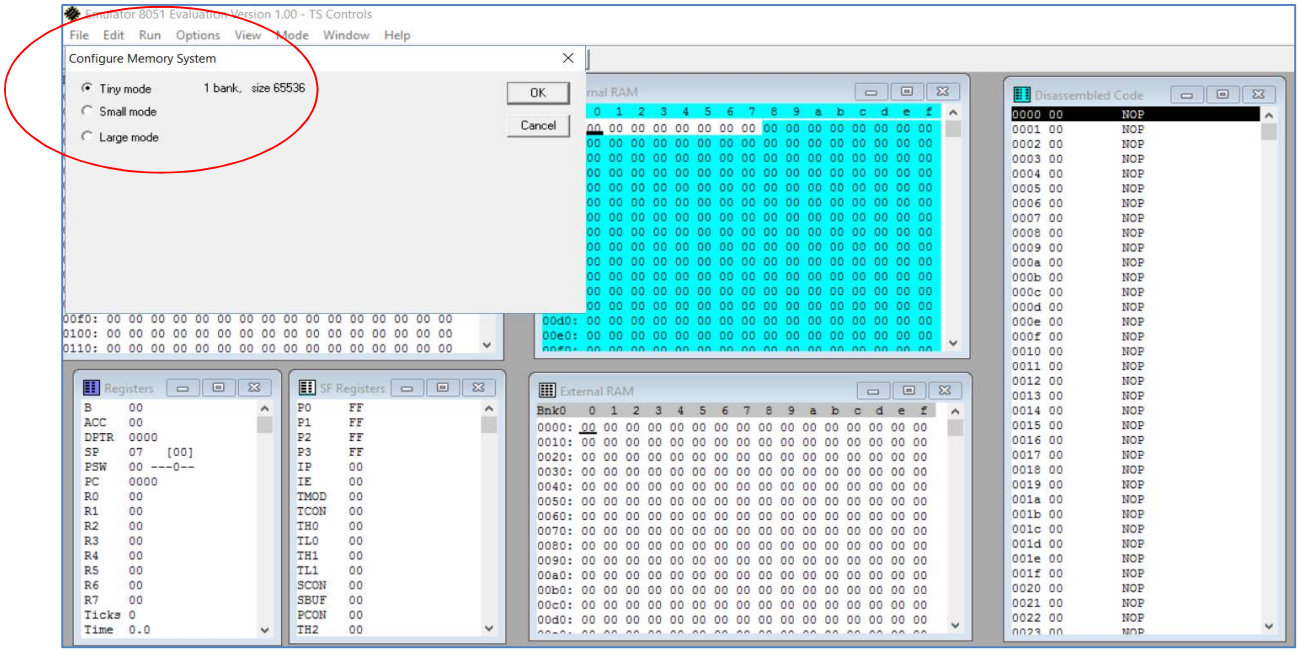


Figure 4.6: Selecting *'tiny mode'*

h. Now open the various windows by clicking "View" from the menu bar as shown in Figure 4.7 to get familiar with them. You may need these windows in future experiments.

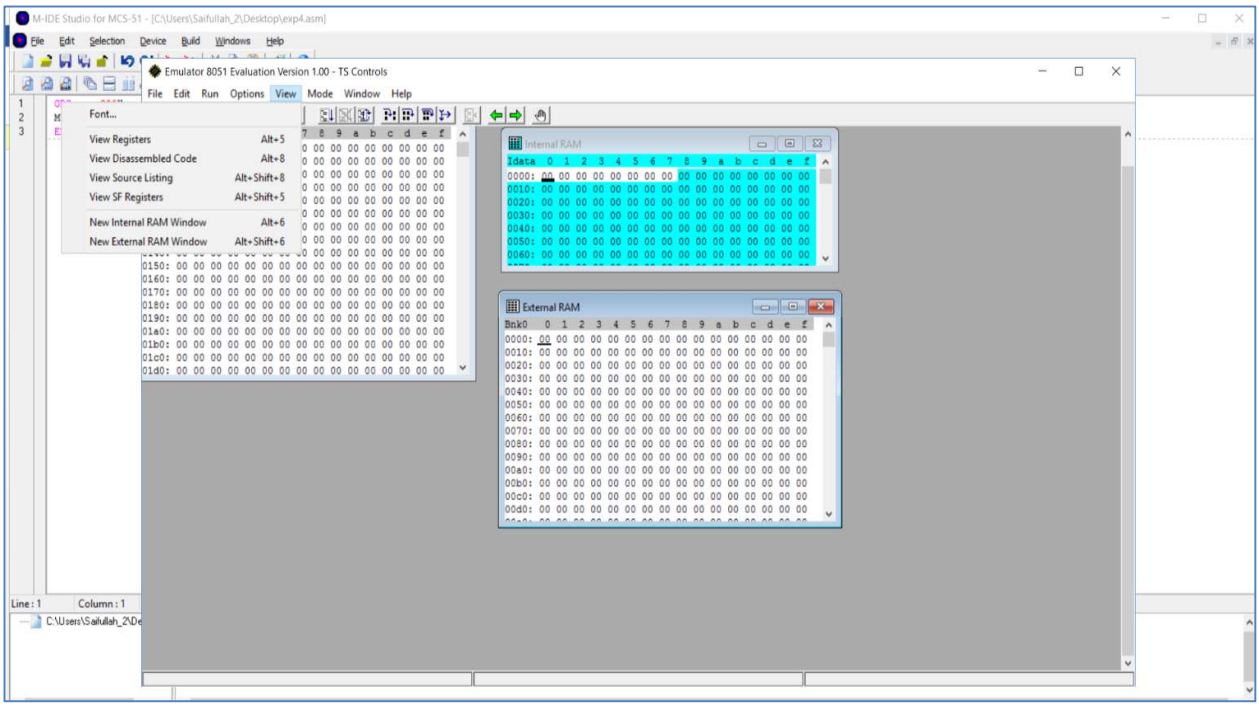


Figure 4.7: Opening Different Windows

i. An example of the functions of windows is shown in Figure 4.8. Identify different windows in the simulator. E.g., Windows for:

1. Registers
2. SF Registers
3. Internal Memory
4. External Memory
5. Disassembled Code

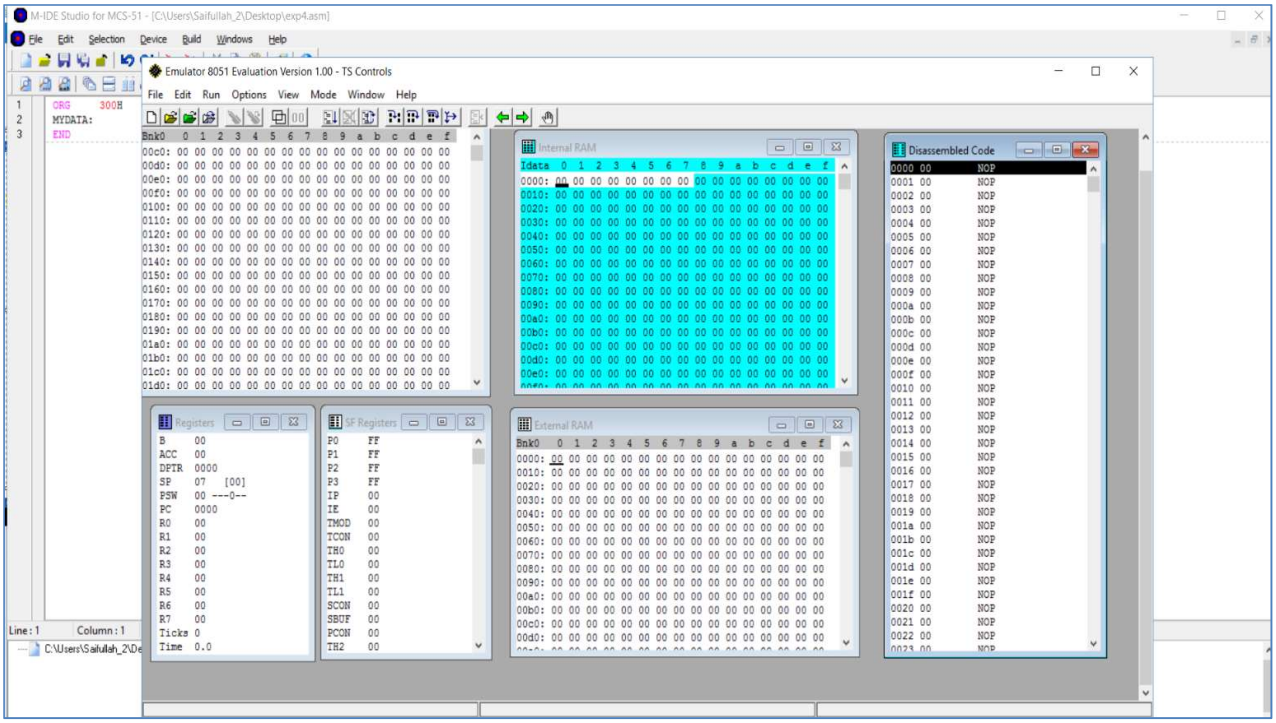


Figure 4.8: Windows Showing the Contents of Different Resources of the 8051

j. Run the program "line by line" by clicking "Run → Step" or directly pressing "F11" key.

k. Run the whole program by clicking "Run → Run" or directly by pressing "F5" key.

l. Program can be stopped by clicking "Run → **Stop**" or by pressing "Shift + F5".

2. Execute the prelab programs and verify.

4.8 Lab Report should contain:

(i) Prelab
(ii) Report with objective and lab-programs/lab-work with results and conclusion

Experiment #5:
Jump, Loop, and Call Instructions

5.0 Objective:

The objective of this experiment is to learn the jump, loop, and call instructions and write simple programs for 8051. The instructions will be first executed using 8051 simulation software MIDE-51. Finally, the program will be downloaded and executed using 8051 microcontroller trainer.

5.1 Introduction:

In the sequence of instructions to be executed, it is often necessary to transfer program control to a different location or to call a time delay subroutine. There are many instructions in the 8051 to achieve this. In this lab, we discuss instructions used for looping, as well as instructions for conditional jump, unconditional jump and call.

1. The SJMP (*Short Jump*) Instruction

The SJMP instruction is a 2-byte unconditional jump instruction which is used to make jumps in the program. The jump to the target address will be within -128 to +127 bytes of memory relative to the address of the current PC (program counter). The syntax of the SJMP instruction is:

SJMP *label*

```
      MOV  R3,#2
AG:   ADD  A,R3
      SJMP AG
```

2. The LJMP (Long Jump) Instruction

The LJMP instruction is a 3-byte unconditional jump instruction which is used to make jumps in the program. The target address allows a jump to any memory location from 0000 to FFFFH within the 64KB memory address space. The syntax of the LJMP instruction is:

LJMP *label*

The target address is referred to by the label.

3. The DJNZ Instruction

The DJNZ is an important conditional jump instruction that is widely used for making loops in the 8051 programs. This instruction takes the following syntax:

DJNZ register, label

```
        MOV  R3,#10
AG:     ADD  A,R3
        DJNZ R3,AG          ; Repeat until R3 is zero (10 times)
```

4. The JNZ Instruction (Jump if A ≠ 0)

In this instruction the content of register A is checked. If it is not zero, it jumps to the target address. Notice that the JNZ instruction can be used only for register A. It can only check to see whether the accumulator is zero or not, and it does not apply to any other register.

```
        MOV  A,R0
        JNZ  AG     ; Jump if A is not zero
AG:     ...
```

5. The JZ Instruction (Jump if A = 0)

In this instruction the content of register A is checked. If it is zero, it jumps to the target address. It is applied only to the accumulator register.

```
        MOV  A,R0
        JZ   AG     ; Jump if A is zero
AG:     ...
```

6. The CJNE Instruction

The CJNE (Compare and jump if not equal) instruction is used to compare the content of a general purpose register with a certain value or direct memory location and jump to a certain address if the two value are not equal. The syntax of the instruction is:

CJNE destination, source, label

The destination can be register A or the bank registers (R0-R7), and the source may be an immediate value or a direct memory address.

7. JNC (jump if no carry)

In this instruction, the carry flag bit in the flag (PSW) register is used to make the decision whether to jump. In executing "JNC label", the processor looks at the carry flag to see if it is raised (C ='1'). If it is not, the CPU starts to fetch and execute instructions from the address of the label. If C = '1', it will not jump but will execute the next instruction below JNC.

8. JC (Jump if carry)

In the JC instruction, if C = '1' it jumps to the target address. On the other hand if no carry occurs, the processor will execute the next instruction below JC.

9. The JNB and JB Instructions

The JNB (jump if no bit) and JB (jump if bit = 1) instructions are also widely used single-bit operations. They allow us to monitor a bit and make a decision depending on whether it is '0' or '1'.

10. CALL Instructions

CALL instruction is used to call a 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. In the 8051, there are two instructions for call: LCALL (3 bytes) and ACALL (2 bytes)

```
        ORG 00H         ; After reset, start fetching instructions from 00H
        MOV A, #55H     ; load 0101 0101 in A
AGAIN:
        ACALL DELAY     ; call delay routine
        CPL A           ; invert A to 1010 1010
        SJMP AGAIN      ; repeat

DELAY:  MOV R4, #05     ; move 5 in R4
AG:     DJNZ R4,AG      ; decrement R4 until 0
        RET
END
```

5.2 PRE LAB:

1. Write an 8051 program to add 3 to the accumulator ten times and save the result in register R5.

2. Write a program to (a) load the accumulator with the value 55H, and (b) complement the accumulator 500 times.

3. Write an 8051 program to generate a delay of 10 msec (Assume crystal frequency of 11.0 592 MHz).

5.3 LAB WORK:

1. The given program generates a square wave of duty cycle 50% and frequency of 1 KHz at P1.0. Verify the output using MIDE-51 program as well as 8051 trainer board with the help of oscilloscope. To do this follow the steps below:

a. In this experiment, we will use 8051 trainer board 8 logic level switches for input (associated with FF02H memory address) and LEDs for outputs (associated with FF00H and FF01H memory locations) as shown in Figure 5.1.

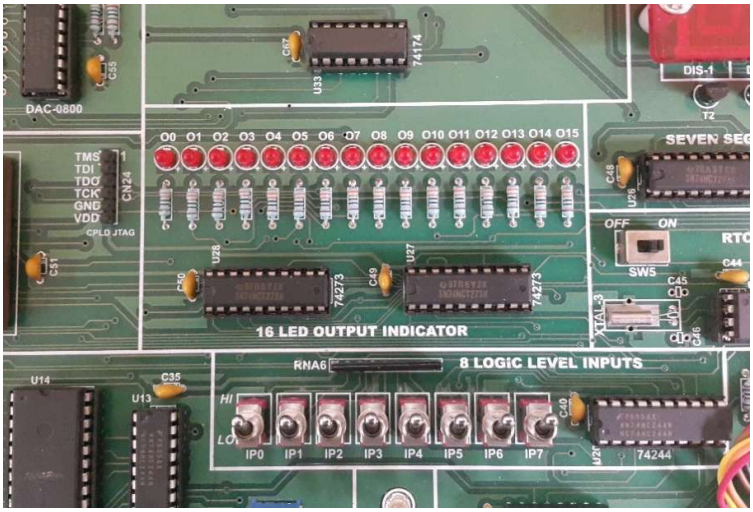


Figure 5.1: LEDs and Input Switches Associated with Memory Locations.

b. To start with, use MIDE-51 program to write the program below. Name your file as **led_blink.asm**.

c. Build and simulate the program. This will generate the HEX file ***led_blink.hex***.

d. Execute the program and see the output using MIDE-51.

e. Connect the 8051 trainer board (power and data cords) with PC.

f. Download the "HEX" file to 89S52 microcontroller available on trainer board. See detailed process in Experiment-3 manual.

g. Run the program to verify the execution of the program using 8051 board.

This program blinks LEDs every half second (500 msec approx.)

```
        ORG 00H          ; After reset, start fetching instructions from 00H
        MOV A, #55H      ; load 0101 0101 in A
        MOV DPTR, #0FF00H; LEDs O0-O7 are controlled by FF00H memory address
AGAIN:
        MOVX @DPTR, A    ; move A to memory address
        ACALL DELAY      ; call delay routine
        CPL A            ; invert A to 1010 1010
        SJMP AGAIN       ; repeat blinking
DELAY:                   ; this routine implements a delay of 500ms
        MOV R4, #05      ; move 5 in R4    ➔ 1 MC (machine cycle)
OUTER2: MOV R3, #200     ; move 200 in R3 ➔ 1 MC
```

50% Duty Cycle
500ms ➔ ON
500ms ➔ OFF

$$f = \frac{1}{T} = \frac{1}{1000ms} = 1\ Hz$$

```
OUTER1: MOV R2, #255        ; move 255 in R2 ➔ 1 MC
INNER:  DJNZ R2, INNER      ; decrement R2 until 0 ➔ 2 MC
        DJNZ R3, OUTER1     ; decrement R3 until 0 ➔ 2 MC
        DJNZ R4, OUTER2     ; decrement R4 until 0 ➔ 2 MC
RET
END
```

2. Write a new program to generate a square wave of duty cycle 25% and frequency of 1 KHz at P2.5. Verify the output using MIDE-51 program, 8051 board and the oscilloscope.
3. Write a program for 8051 that takes input from the switch and controls the speed of a continuous flashing light system. Your program must fulfill the following conditions.
   i. The pattern of the LED's should be from O0 to O7 controlled by FF00H memory address.
   ii. Three input switches must be used. Say IP2 = '1' is used to activate the 1st speed, IP1 = '1' is used to activate the 2nd speed and IP0= '1' is used to activate the 3rd speed, tabulated below.
   iii. To start, the flashing light system should only start if it receives an input from the switch.
   iv. Once the system has started, the user must be given the option to change the speed of the light system according to the following table sequence:

Switch (IP0 IP1 IP2)	**Flashing Light Speed**
Input 1 (0 0 1)	Slow (Delay = 2000 msec)
Input 2 (0 1 0)	Medium (Delay = 1250 msec)
Input 3 (1 0 0)	Fast (Delay = 750 msec)

5.4 Lab Report should contain:

(i) Prelab
(ii) Report with objective and lab-programs/lab-work with results and conclusion

Experiment #6:
The 8051 Microcontroller Addressing Modes

6.0 Objective:

To study the different addressing modes of the 8051 microcontroller and to verify the actions of data transfer in MIDE-51.

6.1 Introduction:

The CPU can access data in various ways. The data could be in a register, or in memory, or be provided as an immediate value. These various ways of accessing data are called *addressing modes.* The 8051 provides six distinct addressing modes. The addressing modes 3-5 are used to access the memory. These modes are as follows:

1. Immediate Addressing Mode
2. Register Addressing Mode
3. Direct Addressing Mode
4. Indirect Addressing Mode
5. External Indirect Addressing Mode
6. Index Addressing Mode

1. Immediate Addressing Mode

Immediate addressing is so-named because the value to be stored in memory immediately follows the operation code in memory. In immediate addressing mode, the source operand is constant.

For example, the instruction:

MOV ,#20H

This instruction uses Immediate Addressing because the Accumulator will be loaded with the value that immediately follows; in this case 20 (hexadecimal). Notice that the immediate data must be preceded by the pound sign "#." This addressing mode can be used to load information into any of the registers, including the DPTR register as shown in the following examples:
DPTR can also be accessed as two 8-bit registers, the high byte DPH and low byte DPL

```
MOV A,#25H          ;load 25H into A
MOV R4,#62          ;load 62 into R4
MOV B,#40H          ;load 40H into B
```

```
MOV DPTR,#4521H      ;DPTR=4512H
MOV DPL,#21H         ;This is the same
MOV DPH,#45H         ;as above

;illegal!! Value > 65535 (FFFFH)
MOV DPTR,#68975
```

Notice that we can also use immediate addressing mode to send data to 8051 ports. For example, "`MOV P1,#55H`" is a valid instruction.

Immediate addressing is very fast since the value to be loaded is included in the instruction. However, since the value to be loaded is fixed at compile-time it is not very flexible.

2. Register Addressing Mode

Register addressing mode involves the use of registers to hold the data to be manipulated. Examples of register addressing mode are shown below:

```
MOV A ,R0        ;copy contents of R0 into A
MOV R2,A         ;copy contents of A into R2
ADD A ,R5        ;add contents of R5 to A
ADD A ,R7        ;add contents of R7 to A
MOV R6,A         ;save accumulator in R6
```

It should be noted that the source and destination registers must match in size. In other words, coding "`MOV DPTR,A`" will give an error, since the source is an 8-bit register and the destination is a 16-bit register.

Notice that we can move data between the accumulator and Rn (for n = 0 to 7) but movement of data between "Rn" registers is not allowed. For example, the instruction "MOV R4, R7″ is invalid.

3. Direct Addressing Mode

Direct addressing is so-named because the value to be stored in memory is obtained by directly retrieving it from another memory location. For example:

MOV A, 30H

This instruction will read the data out of Internal RAM address 30 (hexadecimal) and store it in the Accumulator. Further examples of the direct addressing mode are shown below:

```
MOV R0,40H    ;save content of RAM location 40H in R0
MOV 56H,A     ;save content of A in RAM location 56H
MOV R4,7FH    ;move contents of RAM location7FH to R4
```

As discussed earlier, RAM locations 0 to 7 are allocated to bank 0 registers R0 – R7. The registers can be accessed in two ways, as shown below.

```
MOV A, 4          ; is same as (direct addressing mode)
MOV A,R4          ;which means copy R4 into A
                  ;(register addressing mode)
MOV A,2           ;copy R2 to A (A= R2=05)
MOV B,2           ;copy R2 to B (B=R2=05)

MOV 7,2           ;copy R2 to R7
                  ;since "MOV R7, R2" is invalid
```

Contrast this with immediate addressing mode, there is no "#" sign in the operand

```
MOV R0,40H        ;save content of 40H in R0
MOV 56H,A         ;save content of A in 56H
```

Direct addressing is generally fast since, although the value to be loaded isn't included in the instruction, it is quickly accessible since it is stored in the 8051s Internal RAM. It is also much more flexible than Immediate Addressing Mode since the value to be loaded is whatever is found at the given address. Also, it is important to note that when using direct addressing any instruction which refers to an address between 00H and 7FH is referring to Internal Memory. Any instruction which refers to an address between 80H and FFH is referring to the SFR control registers that control the 8051 microcontroller itself. For example, the instruction `MOV 0E0,#30H` is equivalent to `MOV A,#30H`.

4. Indirect Addressing Mode

Indirect addressing is a very powerful addressing mode which in many cases provides an exceptional level of flexibility. Indirect addressing is also the only way to access the extra 128 bytes of Internal RAM found on an 8052.
Indirect addressing appears as follows:

`MOV A,@R0`

This instruction causes the 8051 to analyze the value of the R0 register. The 8051 will then load the accumulator with the value from Internal RAM which is found at the address indicated by R0.

If the data is inside the on-chip RAM, only registers R0 and R1 are used as pointers. In other words, R2 – R7 cannot be used to hold the address of an operand located in RAM when using this addressing mode. When R0 and R1 hold the addresses of RAM locations, they must be preceded by the "@" sign.

```
MOV A,@R0       ;move contents of RAM whose
                ;address is held by R0 into A
MOV @R1,B       ;move contents of B into RAM
                ;whose address is held by R1
```

Indirect addressing mode always refers to Internal RAM; it never refers to an SFR. Another advantage of indirect addressing mode is that it makes accessing data dynamic rather than static as in direct addressing mode, as looping is not possible in direct addressing mode.

5. External Indirect Addressing Mode

In addition to its code memory, the 8051 family also has 64K bytes of data memory space. In other words, the 8051 has 128K bytes of address space of which 64K bytes are set aside for program code and the other 64K bytes are set aside for data. Program space is accessed using the program counter (PC) to locate and fetch instructions, but the data memory space is accessed using the DPTR register and an instruction called MOVX, where X stands for external (meaning that the data memory space must be implemented externally).

This method of addressing is used to access data stored in external data memory. There are only two commands that use external indirect addressing mode:

```
MOVX A,@DPTR
MOVX @DPTR, A
```

As you can see, both commands utilize DPTR. In these instructions, DPTR must first be loaded with the address of external memory that you wish to read or write. Once DPTR holds the correct external memory address, the first command will move the contents of that external memory address into the Accumulator. The second command will do the opposite: it will allow you to write the value of the Accumulator to the external memory address pointed to by DPTR.

6. Indexed Addressing Mode

Indexed addressing mode is widely used in accessing data elements of look-up table entries located in the program ROM space of the 8051. The instruction used for this purpose is:

```
MOVC A, @A+DPTR
```

The 16-bit register DPTR and register A are used to form the address of the data element stored in on-chip ROM. Because the data elements are stored in the program (code) space ROM of the 8051, the instruction MOVC is used instead of MOV. The “C” means code. In this instruction the contents of A are added to the 16-bit register DPTR to form the 16-bit address of the needed data.

One major difference between the code space and data space is that, unlike code space, the data space cannot be shared between code and data.

Example 1:
Write a program to send 55H to ports P1 and P2, using (a) their names (b) their addresses

Solution :
(a)

```
MOV A,#55H          ;A=55H
MOV P1,A            ;P1=55H
MOV P2,A            ;P2=55H
```

(b) From SFR table, P1 address=80H; P2 address=A0H

```
MOV A,#55H          ;A=55H
MOV 80H,A           ;P1=55H
MOV 0A0H,A          ;P2=55H
```

The look-up table allows access to elements of a frequently used table with minimum operations

Example 2:
Write a program to copy the value 55H into RAM memory locations 40H to 41H using (a) direct addressing mode, (b) register indirect addressing mode without a loop, and (c) with a loop

Solution:
(a)

```
        MOV A,#55H      ;load A with value 55H
        MOV 40H,A       ;copy A to RAM location 40H
        MOV 41H,A       ;copy A to RAM location 41H
(b)
        MOV A,#55H      ;load A with value 55H
        MOV R0,#40H     ;load the pointer. R0=40H
        MOV @R0,A       ;copy A to RAM R0 points to
        INC R0          ;increment pointer. Now R0=41h
        MOV @R0,A       ;copy A to RAM R0 points to
(c)
        MOV A,#55H      ;A=55H
        MOV R0,#40H     ;load pointer. R0=40H,
        MOV R2,#02      ;load counter, R2=3
 AGAIN: MOV @R0,A       ;copy 55 to RAM R0 points to
        INC R0          ;increment R0 pointer
        DJNZ R2,AGAIN   ;loop until counter = zero
```

Example 3:
Write a program to clear 16 RAM locations starting at RAM address 60H

Solution:

```
        CLR A          ;A=0
        MOV R1,#60H    ;load pointer. R1=60H
        MOV R7,#16     ;load counter, R7=16
AGAIN:  MOV @R1,A      ;clear RAM R1 points to
        INC R1         ;increment R1 pointer
        DJNZ R7,AGAIN  ;loop until counter=zero
```

Example 4:
Write a program to get the x value from P1 and send x2 to P2, continuously

Solution:

```
        ORG 0
        MOV DPTR,#300H    ;LOAD TABLE ADDRESS
        MOV A,#0FFH       ;A=FF
        MOV P1,A          ;CONFIGURE P1 INPUT PORT
BACK:   MOV A,P1          ;GET X
        MOV A,@A+DPTR     ;GET X SQAURE FROM TABLE
        MOV P2,A          ;ISSUE IT TO P2
        SJMP BACK         ;KEEP DOING IT

        ORG 300H

XSQR_TABLE:
        DB 0,1,4,9,16,25,36,49,64,81
        END
```

Example 5
Write a program to copy a block of 10 bytes of data from 35H to 60Hin the internal RAM

Solution:

```
        MOV R0, #35H ; source pointer
        MOV R1, #60H ; destination pointer
        MOV R3, #10  ; counter
BACK:   MOV  A, @R0  ; get a byte from source
        MOV @R1, A   ; copy it to destination
        INC R0       ; increment source pointer
        INC R1       ;increment destination pointer
        DJNZ R3,BACK ; keep doing for ten bytes
```

6.2 Pre Lab:

Identify the addressing modes for the following instructions.

	Statement	Addressing Mode	what will happen after execution
a.	MOV B,#34H		
b.	MOV A,50H		
c.	MOV R2,07		
d.	MOV R3, #0		
e.	MOV R7,0		
f.	MOV R6,#7FH		
g.	MOV R0,A		
h.	MOV B,A		
i.	MOV A,@R0		
j.	MOV R7, A		

6.3 Lab Work:

1. Write a program to add the following data and store the result in RAM location 30H.

```
          ORG 200H
MYDATA:   DB 06, 09, 02, 05, 07
```

2. Write a program to copy the contents of internal RAM locations 20-2FH to external RAM (ROM) locations D0-DFH.

   To verify your program, first initialize internal RAM locations 20-2FH with hex value A.

3. Write a program to copy the contents of external RAM (ROM) locations 20-2FH to internal RAM locations D0-DFH.

   To verify your program, first initialize upper/external RAM locations 20-2FH with hex value B.

4. Write a program to copy the contents of external RAM (ROM) locations 10-1FH to external RAM (ROM) locations 30-3FH.

   To verify your program, first initialize external RAM (ROM) locations 10-1FH with hex value C.

6.4 Lab Report should contain:

(i) Prelab
(ii) Report with objective and lab-programs/lab-work with results and conclusion

Experiment #7:
The 8051 Arithmetic Instructions

7.0 Objective:

To study the different arithmetic instructions in 8051 microcontroller.

7.1 Introduction:

Arithmetic instructions provide the 8051 microcontroller with its basic integer math skills. The 8051 family provides several instructions to perform addition, subtraction, multiplication, and division on different sizes and types of numbers. The basic set of assembly language instructions is as follows:

Addition: ADD, ADDC, INC
Subtraction: SUBB, DEC
Multiplication: MUL
Division: DIV

ADD Instruction:

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

The instruction ADD is used to add two operands. In add instruction, the destination operand is always in register A and source operand can be a register, immediate data, or in memory. Memory-to-memory arithmetic operations are never allowed in 8051 Assembly language

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

ADDC Instruction:
When adding two 16-bit data operands, the propagation of a carry from lower byte to higher byte is concerned

Example:
Write a program to add two 16-bit numbers. Place the sum in R7 and R6; R6 should have the lower byte.

Solution:

```
CLR C           ;make CY=0
MOV A, #0E7H    ;load the low byte now A=E7H
ADD A, #8DH     ;add the low byte
MOV R6, A       ;save the low byte sum in R6
```

```
        MOV A, #3CH    ;load the high byte
        ADDC A, #3BH   ;add with the carry
        MOV R7, A      ;save the high byte sum
```

SUBB Instruction:
In many microprocessor there are two different instructions for subtraction: SUB and SUBB (subtract with borrow). In the 8051 we have only SUBB. The 8051 uses adder circuitry to perform the subtraction

```
        SUBB A,source  ;A = A – source – CY
```

To make SUB out of SUBB, we have to make CY=0 prior to the execution of the instruction. Notice that we use the CY flag for the borrow

SUBB when CY = 0

1. Take the 2's complement of the subtrahend (source operand)
2. 2. Add it to the minuend (A)
3. 3. Invert the carry

```
     CLR C
     MOV A,#4C      ;load A with value 4CH
     SUB  ;if CY=1, take 1's complement
     INC A          ;and increment to get 2's comp
NEXT:MOV R1,A       ;save A in R1
```

```
Solution:

     4C   0100 1100      0100 1100
   - 6E   0110 1110      1001 0010  (a) (2's complement)
   -  22 01101 1110      (b) add the numbers
     2's compliment of DE_H
(c) invert the carry CY = 1
```

If CY=0, the result is positive, and if CY=1, the result is negative and the destination has the 2's complement of the result.

MUL Instruction

The 8051 supports byte by byte multiplication only. The byte are assumed to be unsigned data

```
        MUL AB         ;AxB, 16-bit result in B, A
        MOV A,#25H     ;load 25H to reg. A
        MOV B,#65H     ;load 65H to reg. B
        MUL AB         ;25H * 65H = E99 where
                       ;B = OEH and A = 99H
```

Unsigned Multiplication Summary (MUL AB)

Multiplication	Operand1	Operand2	Result
Byte x Byte	A	B	B = high byte A = low byte

Unsigned DIV Instruction:

The 8051 supports byte over byte division only. The byte are assumed to be unsigned data

```
DIV AB          ;divide A by B, A/B

MOV A,#95       ;load 95 to reg. A
MOV B,#10       ;load 10 to reg. B
DIV AB          ;A = 09(quotient) and
                ;B = 05(remainder) A B A B
```

Unsigned Division Summary (DIV AB)

Division	Numerator	Denominator	Quotient	Remainder
Byte / byte	A	B	A	B

7.2 Pre Lab:

1. Write a program to add all digits of your ID number and save the result (in Dec) in R3.

2. Write a program to perform 77 x 34 and 77 ÷ 34 in 8051.

7.3 Lab Work:

1. Write a program to add the following data and store the result in R2, R3. The data is stored in upper/external RAM.

```
          ORG 250H
MYDATA:   DB 53H, 94 H, 56 H, 92 H, 74 H, 65 H, 43 H, 23
H, 83H
```

2. Write a program with three subroutines. (a) Transfer the following data from upper/external RAM to lower/internal RAM locations starting at 30H, (b) add them and save the result in 70H, and (c) find the average of the data and store the result in R7.

```
          ORG 250H
MYDATA:   DB 3 H, 9 H, 6 H, 9 H, 7 H, 6 H, 4 H, 2 H, 8 H
```

3. Write a program in MIDE that calculates the factorial of number 5 and stores the result in a memory location. Verify the program using debugger [Hint: Since 5! = 5x4x3x2x1, use MUL instruction to find the multiplication. Store 5 in a register and decrement the register after every multiplication and then multiply the result with the decremented register. Repeat these steps using conditional jump instruction]

4. Count all the factors of decimal number 36 in the memory locations 260H-280H.

```
          ORG 260H
MYDATA:   DB 2, 3, 4, 6, 7, 6, 8, 9, 2, 3, 4, 6, 7, 6, 8,
9, 2, 3, 4, 6, 7, 6, 8, 9, 2, 3, 4, 6, 7, 6, 18, 9  (all
hex numbers)
```

7.4 Lab Report should contain:

(i) Prelab
(ii) Report with objective and lab-programs/lab-work with results and conclusion

Experiment #8:
Using timers of 89S52 microcontroller

8.0 Objectives:

- Familiarization with 89S52 microcontroller's Timers/Counters and special function registers (SFRs) related to timer usage.
- Learning how to initialize and use the SFRs related to timers and counters.
- Learning how to write delay subroutines using timers.

8.1 Material Required:

- 8051 microcontroller trainer (power supply and a serial data cable).
- 8051 PC-based debugger/simulator and programmer.

8.2 Introduction:

- 89S52 microcontroller has built-in three 16-bit timers which can be used for different applications like creating delays, baud rate generators, etc. These timers can also be used as event counters.
- When activated, these Timers/Counters increment once for each event. An event can be related to the system clock (Timer) or from an external logic signal transition (Counter).
- The "Timer" or "Counter" function is selected by control bits C/T in the Special Function Register (SFR) TMOD (Timer MODe). Timer 0 and Timer 1 have four operating modes, which are selected by bit-pairs (M1, M0) in TMOD. However, Timer 2 has three operating modes: capture, auto-reload (up or down counting), and baud rate generator which can be selected by using bits in T2CON register. In this experiment, Timer 0 and Timer 1 will be explained.
- During the transition from its maximum possible count of FFFFH (65535) to 0000H (rolls over from all 1s to all 0s), it sets the overflow flag TFx, where x is 0 or 1 for Timer 0 or Timer 1, respectively.
- **TMOD** register is not bit-addressable. It is loaded, generally, by the software at the beginning of a program to initialize the timer mode.

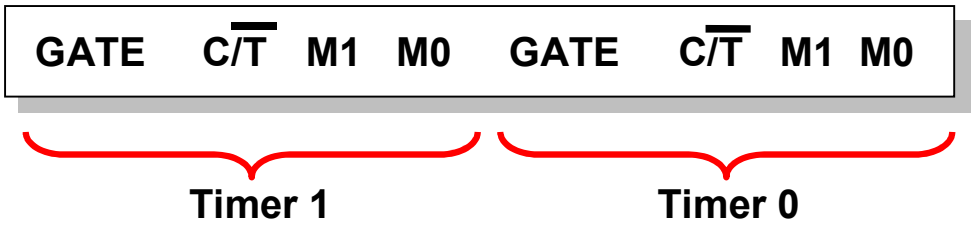


Figure 8.1: TMOD Register

- **C/T:** Set for counter operation, reset for timer operation.
- **GATE:** Permits INTx pin to enable/disable the counter. For normal operation, this should be 0.
- **M1, M0:** To Select the Mode
  - **0 0 : Mode 0 -** 13-bit timer mode (Emulates 8048).
  - **0 1 : Mode 1 -** 16-bit timer mode.
  - **1 0 : Mode 2 -** 8-bit auto-reload mode.
  - **1 1 : Mode 3 -** Split timer mode (only Timer 0 is used as two 8-bit timers while Timer 1 can be used for other purposes like Baud rate generation).

- As all the registers of 89S52 microcontroller are 8-bit. The 16-bit Timers/Counters are built by cascading two 8-bit registers. This cascading is done automatically by hardware; user has to just select the mode. These registers are called as TLx and THx where L stands for low and H stands for high byte of 16-bit register.
- These Timers/Counters can count up only and can be preset to any value so that any time or count limit can be set up. For example, to time or count 100 events, the registers THx and TLx should be loaded with the number 65535 – 100 + 1 = 65436 (FF9CH). So after 100 events, it will roll over from all 1s to all 0s and sets the overflag TFx.
- Mode 2 (8-bit auto-reload mode) is the best mode for counting less than 256 events. Only in this mode you don't have to load the timer register with the initial (preset) values every time when it starts a new loop. In other modes, the timer registers must be initialized with values every time when it starts a new loop.

- **TCON (**Timer CONtrol) register is another SFR related to Timers/Counters. It is bit-addressable register.

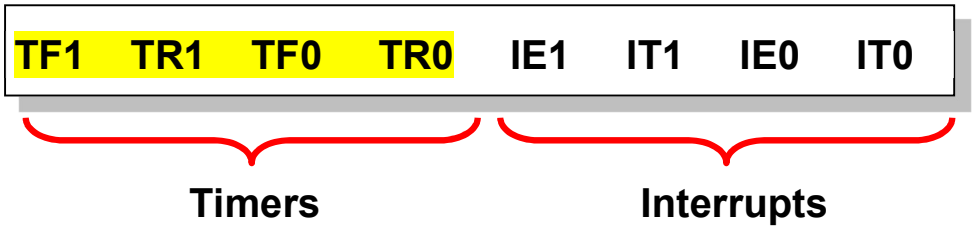


Figure 8.2: TCON Register

  - **TF1, TF0:** Overflow flags for Timer 1 and Timer 0.
  - **TR1, TR0:** Run control bits for Timer 1 and Timer 0 (Set to run, reset to hold).
  - **IE1, IE0:** Edge flag for external interrupts 1 and 0 (Set by interrupt edge, cleared when interrupt is processed)*.
  - **IT1, IT0:** Type bit for external interrupts (Set for falling edge interrupts, reset for 0 level interrupts)*.

  * Not related to counter/timer operation but used to detect and initiate external interrupts.

- Figure 8.3 elaborates more about the timer circuit of 89S52. In this figure, Timer 1 is operating in 16-bit (Mode 1).

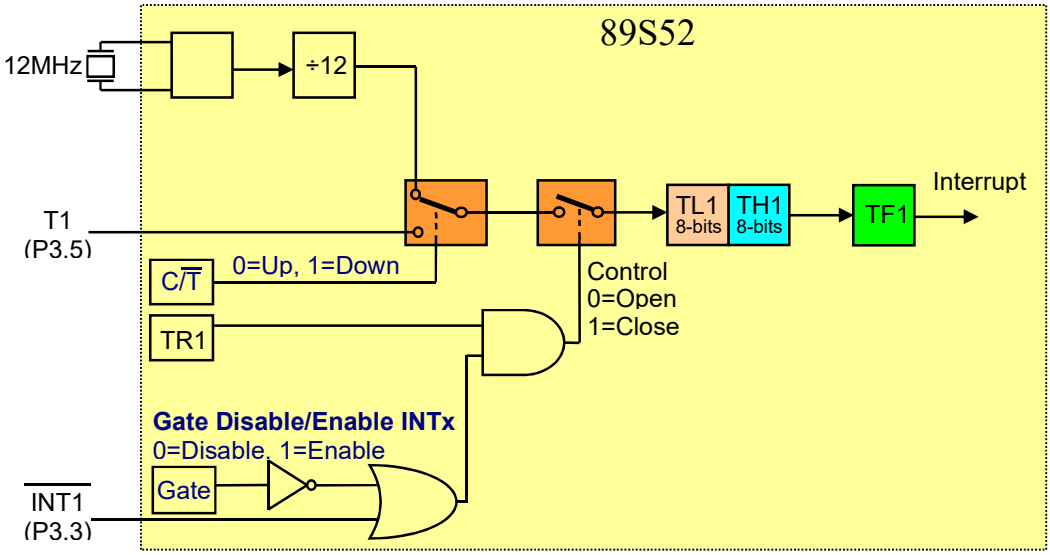


Figure 8.3: Timer 1 is Operating in 16-bit (Mode 1)

8.3 Pre-Lab:

- Assume that you have to count 450_D MC pulses of an original 8051 microcontroller using Timer 0. What should be loaded in the following registers with best suitable mode (Mode 0, 1, 2, or 3)? You can write the values either in Hexadecimal or in Binary.

TMOD

TCON

TH0

TL0

- Is there any other register not mentioned above is needed for this timer operation?
- What is the best suitable mode for the timer?

8.4 Lab Work:

Part I

In many applications there is a need for a specified time delay. The machine cycle is very small (in microseconds). Thus using 16-bit timers cannot generate even a 1 second delay. The following code creates a 1 second delay starting from the end of a high pulse from IP0 (bit 0 of FF02H memory address).

1) Type the following code using the debugger (MIDE-51) that you have used in earlier experiments. Observe how the SFRs related to timers are initialized.

2) Build and simulate the program. This will generate the HEX file.

3) Execute the program and see the output using MIDE-51.

4) Connect the 8051 trainer board (power and data cords) with PC.

5) Download the "HEX" file to 89S52 microcontroller available on trainer board. See detailed process in Experiment-3 manual.

6) Run the program to verify the execution of the program using 8051 board.

```
        ORG 0000H
        MOV TMOD, # 01H        ; Timer 0 in mode 1
        MOV DPTR, #0FF02H      ; 8 logic level switches for input
L1:     MOVX A,@DPTR
        JNB ACC.0, L1          ; Wait for '1' input from IP0 s/w
        MOV DPTR, #0FF00H      ; LEDs O0-O7
L2:     MOV A, #00000001B
        MOVX @DPTR, A
        CALL DELAY1S
        MOV A, #00000000B;
        MOVX @DPTR, A
        CALL DELAY1S
        SJMP L2

DELAY1S: MOV R7, #50           ; Loop for 50 times
AGAIN:   MOV TH0, #0B8H        ; initial value to be loaded
         MOV TL0, #00H         ; in Timer 0 registers
         SETB TR0              ; start Timer 0
WAIT:    JNB TF0, WAIT         ; check the overflow
         CLR TF0               ; clear overflow flag
         CLR TR0               ; stop Timer 0
         DJNZ R7, AGAIN        ; decrement R7 if not 0
         RET                   ; then jump
         END
```

Part II

With the help of above subroutine for DELAY1S, write an efficient main code for controlling 3 machines in a factory. All the three machines should turn ON after receiving input from IP0. Assume the first machine (m/c-1) is connected to bit 0 of memory location FF00H (i.e., LED O0) and should be kept ON for 1 sec. The second machine (m/c-2) is connected to bit 1 of memory location FF00H (i.e., LED O1) and should be kept ON for 3 sec. And the third machine (m/c-3) is connected to bit 2 of memory location FF00H (i.e., LED O2) and should be kept ON for 5 sec, like the following waveform.

Note: you can use just **a single Timer** (like Timer 0 in the above code) to do this.

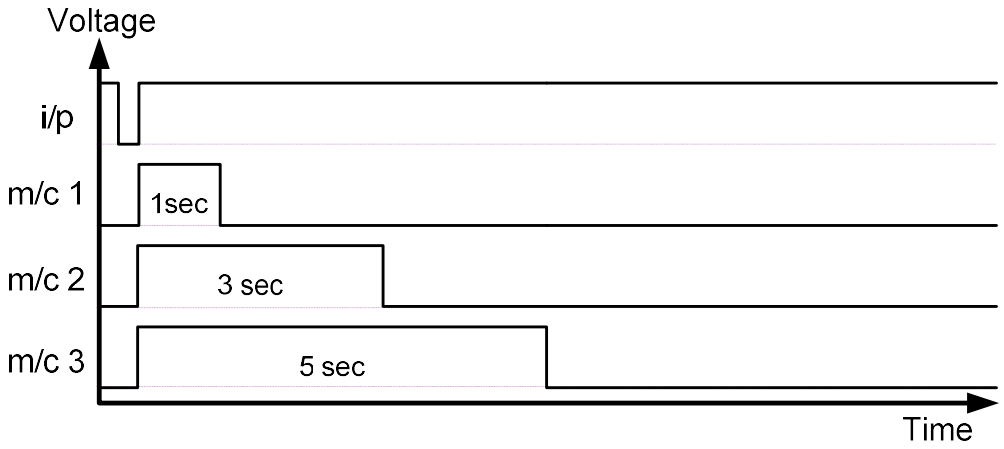


Figure 8.5: Required Waveform for the Three Machines

Part III

Now modify the code to control the 3 machines as follows. (Hint: For the 1st delay period, all three machines are ON …. :

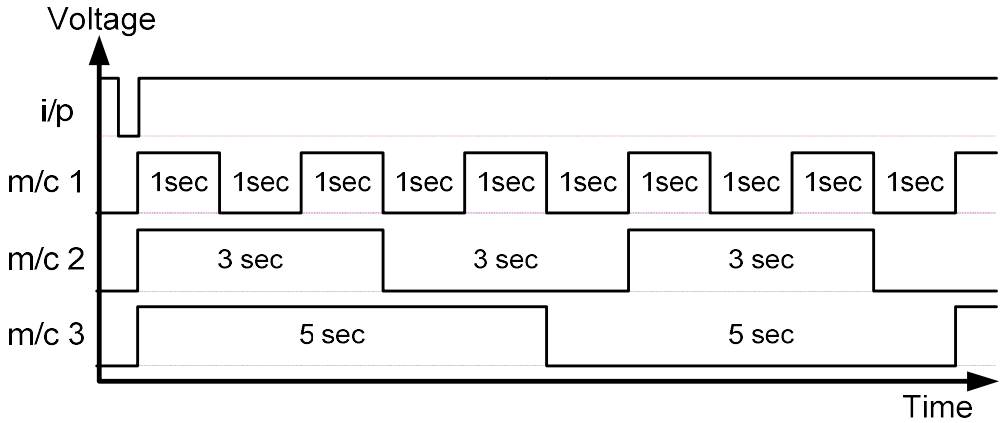


Figure 8.6: Required Modified Waveform for the Three Machines

8.5 Lab Report should contain:

(i) Prelab
(ii) Report with objective and lab-programs/lab-work with results and conclusion

Experiment #9:
Introduction to C Compiler for 8051 Microcontroller

9.0 Objectives:

- Learning how to use 8051 PC-based debugger/simulator.
- Learning how to program the 8051 microcontroller trainer using C language.
- Learning how to write loops using C language.

9.1 Material Required:

- 8051 microcontroller trainer (power supply and a serial data cable).
- 8051 PC-based debugger/simulator and programmer.
- External alarm horn.

9.2 Introduction:

When writing the program, it is often necessary to repeat some instructions again and again especially when generating a time delay. Also, sometimes the input is to be read continuously and some bits are to be controlled based on that input such as toggling the LEDs when input becomes high. In this lab, we discuss some data types and instructions used for looping, defining functions, and to store data in external RAM.

1. Unsigned char

Unsigned char is an 8-bit data type that takes a value in the range of 0-255 (0-FF_H). For example,

unsigned char a = 100;

2. Signed char

The signed char is an 8-bit data type that uses most significant bit (D7) to represent the sign. If D7 is 0 that means data is positive and if it is 1 then it represents negative value. The rest of the bits are used to represent the magnitude of the data providing us values from -128 to +127. For example,

signed char b = -10;

3. Unsigned int

Unsigned int is a 16-bit data type that takes a value in the range of 0-65535 (0-$FFFF_H$). For example,

unsigned int c = 5000;

4. Signed int

The signed int is a 16-bit data type that uses most significant bit (D15) to represent the sign. If D15 is 0 that means data is positive and if it is 1 then it represents negative value. The rest of the bits are used to represent the magnitude of the data providing us values from -32768 to +32767. For example,

signed int d = -500;

5. Sbit (single bit)

It is widely used 8051 data type designed specifically to access single-bit addressable registers. It allows access to the single bits of the SFR registers. For example,

sbit MYBIT = 1;

6. For loop instruction

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. Following is the syntax of '*for'* loops.

```
for ( init; condition; increment ) {
    statement(s); }
```

To write forever loop use following instruction.

```
for (;;)   // forever loop
```

7. Time delay function

Functions are defined as follows:

```
void MSDelay (unsigned int);  // To define at the beginning of the code
// Following is the function usually written at the end of the code
void MSDelay (unsigned int itime) {
unsigned int i,j;
for (i = 0 ; i < itime ; i++){
for (j = 0 ; j < 113 ; j++); }}
```

// end of the function

The function given above generates a delay in milliseconds. To use this function, inside the code, write the following instruction.

MSDelay (5000); // To generate 5 seconds delay

8. Read/Write in external RAM

The xbyte instruction is used to read or write data in external RAM. For example,

ACC = XBYTE[0xFF02]; // Reading input since switches IP0-IP7 are controlled
// by $FF02_H$ memory address.
XBYTE[0xFF00] = 3; // Turning on LEDs O0 and O1.

Example 1:

Write an 8051 C program to toggle bits of P1 continuously after each 500 msec.

```
# include <reg51.h>              // Include header file for the registers and SFRs of 8051
void MSDelay (unsigned int); // Defining MSDelay function

void main (void){              // Main function
for (;;){                      // Forever loop
P1 = 0x55;
MSDelay (500);
P1 = 0xAA;
MSDelay (500); }}

void MSDelay (unsigned int itime) {
unsigned int i,j;
for (i = 0 ; i < itime ; i++){
for (j = 0 ; j < 113 ; j++); }}
```

Example 2:

Write an 8051 C program to toggle LEDs O0-O7 continuously after each 200 msec if the input IP0 becomes high.

```
# include <reg51.h>              // Include header file for the registers and SFRs of 8051
# include <absacc.h>             // Include header file for XBYTE
void MSDelay (unsigned int); // Defining MSDelay function
sbit mybit = ACC^0;

void main (void){              // Main function
```

```
ACC = XBYTE[0xFF02];      // Reading input from switches IP0-IP7
if (mybit == 1) {
while (1) {                       // Forever loop
XBYTE[0xFF00]  = 0x55;    // Turning on odd number of LEDs from O0-O7
MSDelay (200);
XBYTE[0xFF00] = 0xAA;     // Turning on even number of LEDs from O0-O7
MSDelay (200); }}}

void MSDelay (unsigned int itime) {
unsigned int i,j;
for (i = 0 ; i < itime ; i++){
for (j = 0 ; j < 113 ; j++); }}
```

9.3 Pre-Lab:

- Write an 8051 C program to store ASCII letters 'A' to 'F' in external RAM addresses starting at $EE00_H$. Also, send the data to P3 one byte at a time.
- Write an 8051 C program to monitor bit 0 of external RAM location $FF02_H$ (switches are connected to this memory address). If it is high, send 55_H to external RAM location $FF01_H$ (connected with LEDs O8-O15); otherwise send AA_H to the same location. Check the input continuously.

9.4 Lab Work:

Part I

A simple security system needs to be built. In this system, if somebody opens a door or a window at night (when activated) should turn ON an alarm for 1 sec and OFF for 1 sec and so on as shown in Figure 9.1, unless the system is reset.

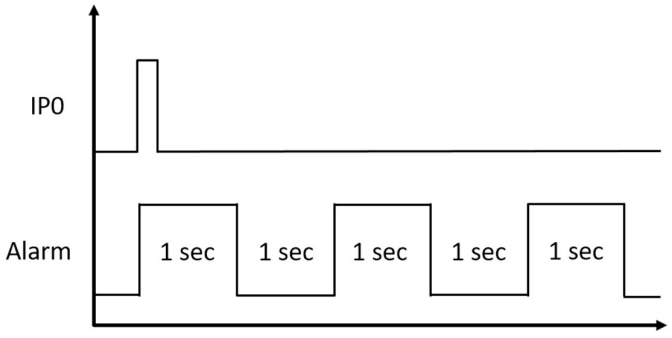


Figure 9.1: Required Waveform for the Alarm

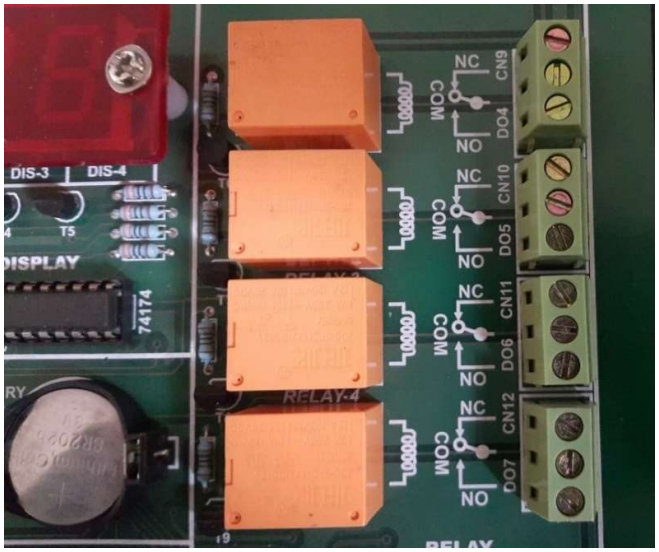


Figure 9.2: Relays available on the 8051 trainer board.

A magnetic reed switch with a magnet will be fitted on a door or a window. The alarm horn will be fitted in the bedroom of the owner of the house. Later on, this system can be extended by fitting more reed switches on all windows and doors of the house. If the door or window is closed, alarm will not be activated. But as soon as somebody opens a door or a window, the alarm will sound. IP0 will be used instead of a reed switch. Moreover, to connect an alarm we will use relay installed on 8051 trainer board. There are four relays on the board as shown in Figure 9.2. These relays can be operated by sending 1 to the corresponding bit at $FF07_H$ memory location in the external RAM as shown in Table 9.1.

Table 9.1

Location	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$FF07_H$	x	x	x	x	Relay 4	Relay 3	Relay 2	Relay 1

In the relays, there are usually three terminals commonly known as normally open (NO), normally closed (NC), and common (COM). There is a switch at COM terminal which is connected with NC terminal when the relay is not energized. However, when the relay is energized this switch gets connected with NO terminal.

1) Connect a speaker as shown in Figure 9.3.

2) Now, open Keil uvision4 program.

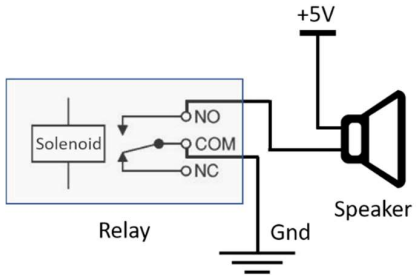


Figure 9.3: Circuit diagram to connect speaker.

3) Go to Project ➔ New uvision Project as shown in Figure 9.4.

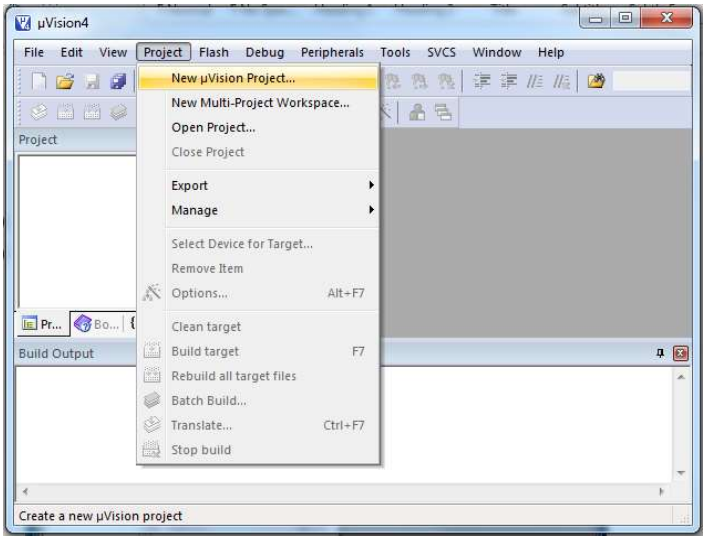


Figure 9.4: Creating a new project.

4) After entering a file name, e.g., 'Experiment9', a new window will open as shown in Figure 9.5.

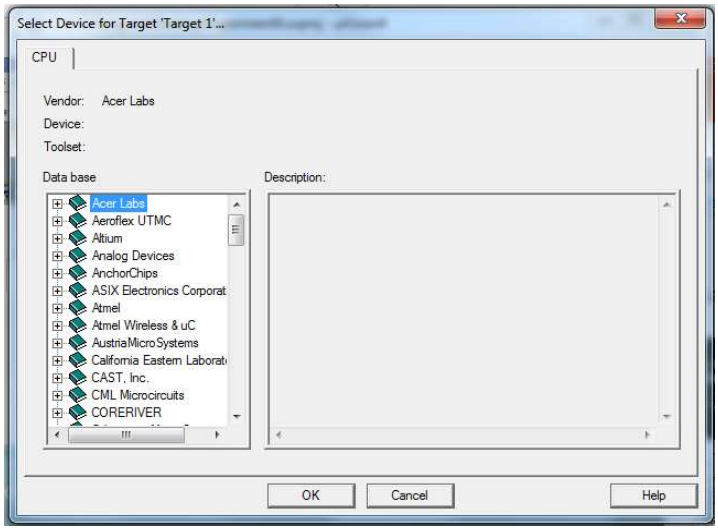


Figure 9.5: Selecting device.

5) Go to Atmel ➔Select AT89S52 and press OK. A pop-up will appear as shown in Figure 9.6. Press No.

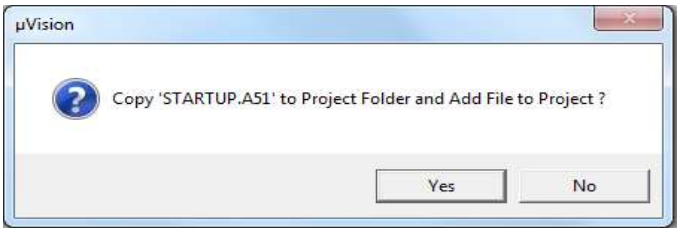


Figure 9.6: A pop-up window.

6) Go to File ➔ New as shown in Figure 9.7.

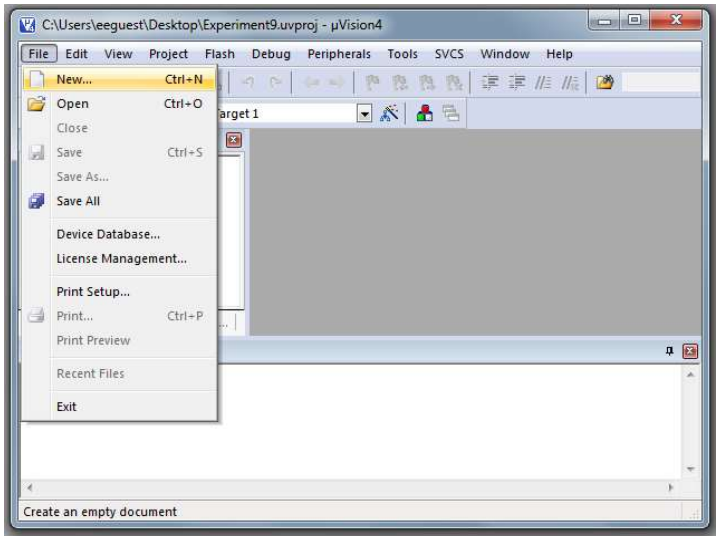


Figure 9.7: Opening a new file.

7) Write the complete code in the text space as shown in Figure 9.8.

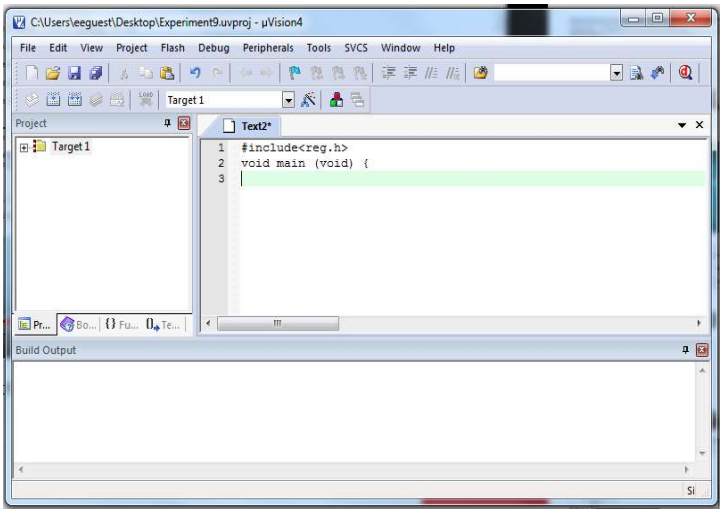


Figure 9.8: Space to write a code.

8) After writing a complete code go to File ➔ Save as and save the file, e.g., ‘exp9.c’.
9) Right click on Target 1 and go to “Options for Target…” as shown in Figure 9.9.

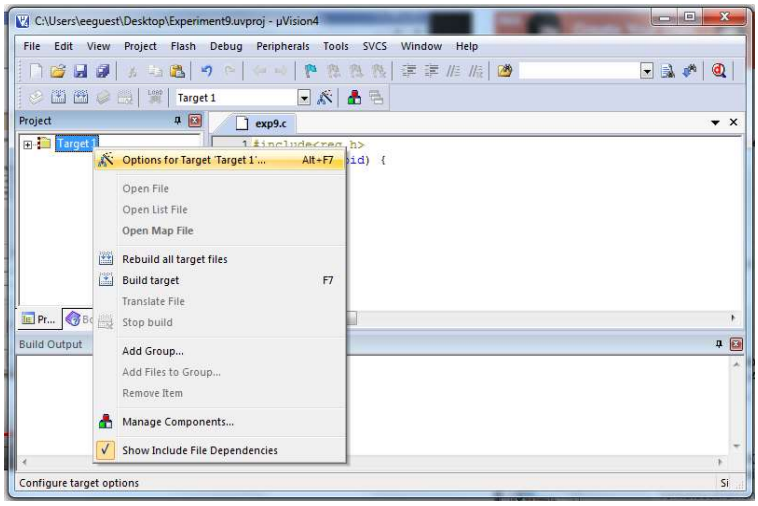


Figure 9.9: Options for Target dialogue box.

10) Set the XTAL to 11.0592 MHz.
11) Go to Output tab in the same window and check the box “Create HEX File” as shown in Figure 9.10. Then Press OK.

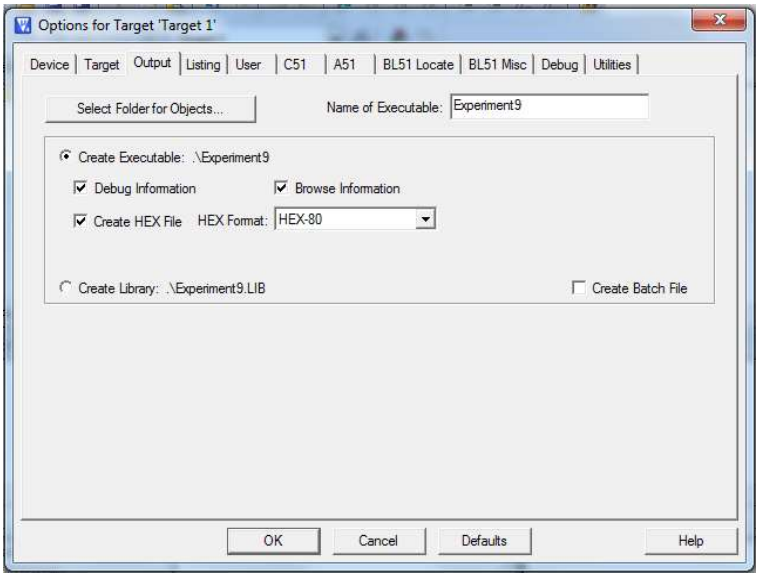


Figure 9.10: Selecting HEX file option.

12) Maximize Target 1 as shown in Figure 9.11. Here, you will find Source Group 1 tab.

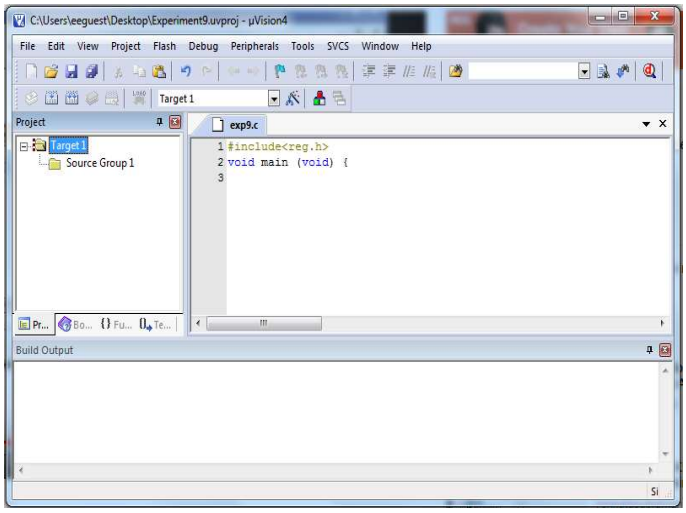


Figure 9.11: Maximizing Target1 tab.

13) Right click "Source Group 1" and press "Add Files to Group…". Now, select 'exp9.c' file and press Add.

14) Right click "Source Group 1" and press "Rebuild all Target Files" as shown in Figure 9.12. This will generate the hex files.

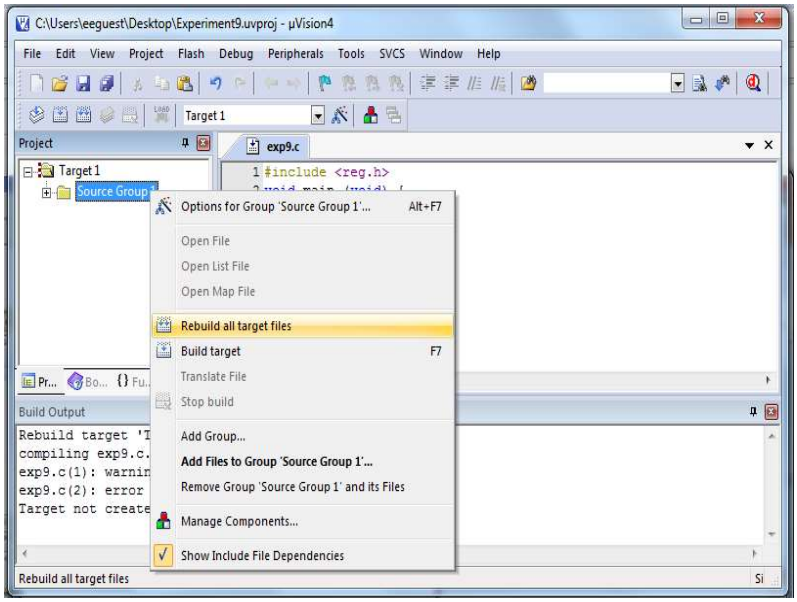


Figure 9.12: Rebuilding all target files.

15) Download the "HEX" file to 89S52 microcontroller available on trainer board. See detailed process in Experiment-3 of lab manual.

16) Run the program to verify the execution of the program using 8051 board.

Part II

In this part, the security system needs to be modified. Now, 3 input switches, i.e., IP0, IP1, and IP2 are required. Further details of the circuit are as follows:

1) Use IP0, IP1, and IP2 to read ALARM, LED, and ACCEPT signals respectively.

2) After having a pulse at IP0 (i.e., ALARM signal), alarm (use $FF07_H$ memory location) should turn ON as in first part of this experiment. Along with this alarm, LED (use $FF00_H$ or $FF01_H$ memory location) should also blink at a frequency of 2 Hz.

3) When there is an input signal at IP1 (i.e., ACCEPT signal), the alarm should be turned OFF and the LED should keep blinking to show that the system is alive.

4) When there is an input signal at IP2 (i.e., CLEAR signal), the system must turn OFF both alarm and LED.

9.5 Lab Report should contain:

(i) Prelab
(ii) Report with objective and lab-programs/lab-work with results and conclusion

Experiment #10
Introduction to interrupts of 89S52

10.0 Objectives:

- Familiarization with 89S52 microcontroller's Interrupts and special function registers (SFRs) related to interrupt usage.
- Learning how to initialize and use the SFRs related to interrupts.
- Learning how to build a simple air-conditioning system which can be extended easily.

10.1 Material Required:

- 8051 microcontroller trainer (power supply and a serial data cable).
- 8051 PC-based debugger/simulator and programmer.
- External fan.

10.2 Introduction:

- 89S52 microcontroller has a built-in 6 source two-level interrupt structure.
- These interrupts are asynchronous to the system clock.
- There are two external interrupts (INT0 and INT1), three timer interrupts (Timers 0, 1, and 2), and the serial port interrupt.
- **IE** (Interrupt Enable) register is an SFR to enable/disable the interrupts. It is bit-addressable as shown in Figure 10.1.

EA	--	ET2	ES	ET1	EX1	ET0	EX0

Figure 10.1: IE Register (Enable Bit = 1 enables the interrupt. Enable Bit = 0 disables it.)

The bits of the IE register are expressed as IE.x where x is the bit number. For example bit 7 of register IE is IE.7. The symbol and function of each bit are:

- **IE.7 – EA** Global disable bit. If EA = 0, all interrupts are disabled. If EA = 1, each interrupt can be individually enabled or disabled by setting or clearing its enable bit.
- **IE.6 (--)** Reserved
- **IE.5 (ET2)** Timer 2 interrupt enable bit.
- **IE.4 (ES)** Serial Port interrupt enable bit.
- **IE.3 (ET1)** Timer 1 interrupt enable bit.
- **IE.2 (EX1)** External interrupt 1 enable bit.
- **IE.1 (ET0)** Timer 0 interrupt enable bit.
- **IE.0 (EX0)** External interrupt 0 enable bit.

For example, Timer 1 interrupt can be enabled as follows:

```
SETB EA     ; Enable global interrupt bit
SETB ET1    ; Enable Timer 1 interrupt
```

- Alternatively, the instruction **MOV IE, #10001000B** sets both bits.
- The default priority polling sequence is given in Table 10.1, where X0 means External Interrupt 0 (highest priority), T0 for Timer 0, X1 for External Interrupt 1, T1 for Timer 1, SP for Serial Port, and T2 for Timer 2.

SOURCE	POLLING PRIORITY	VECTOR ADDRESS
X0	1 (Highest)	03H
T0	2	0BH
X1	3	13H
T1	4	1BH
SP	5	23H
T2	6 (Lowest)	2BH

Table 10.1: Interrupt Polling Priority Sequence

- **IP (Interrupt Priority)** register is an SFR for changing the default interrupt polling sequence priorities.

--	--	PT2	PS	PT1	PX1	PT0	PX0

Figure 10.2: IP Register (Priority Bit = 1 assigns high priority. Priority Bit = 0 assigns low priority.)

The symbol and function of each bit of the IP register are:

- **IP.7 (--)** Not used.
- **IP.6 (--)** Not used.
- **IP.5 (PT2)** Timer 2 interrupt priority bit.
- **IP.4 (PS)** Serial Port interrupt priority bit.
- **IP.3 (PT1)** Timer 1 interrupt priority bit.
- **IP.2 (PX1)** External interrupt 1 priority bit.
- **IP.1 (PT0)** Timer 0 interrupt priority bit.
- **IP.0 (PX0)** External interrupt 0 priority bit.

For example, Timer 0 interrupt priority is lower than External 0 interrupt as shown in Table 10.1 above. The following instruction sets Timer 0 to be the highest priority:

MOV IP, #00000010B

- **TCON (**Timer CONtrol) register is an SFR related to interrupts. It is bit-addressable register as shown in Figure 10.3.

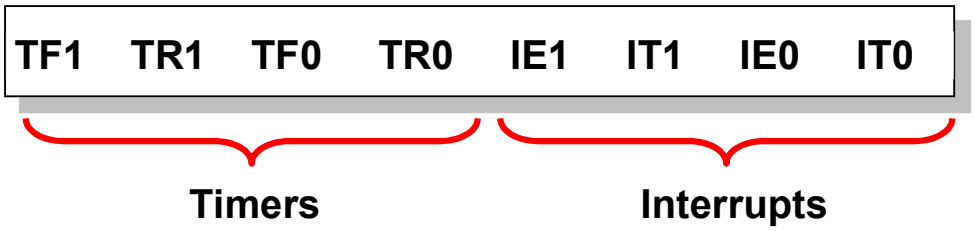


Figure 10.3: TCON Register

- **IE1, IE0 :** Edge flag for external interrupts 1 and 0. Set by interrupt edge, cleared when interrupt is processed.
- **IT1, IT0 :** Type bit for external interrupts. Set for falling edge interrupts, reset for 0 level interrupts.
- **TF1, TF0 :** Overflow flags for Timer 1 and Timer 0.*
- **TR1, TR0 :** Run control bits for Timer 1 and Timer 0. Set to run, reset to hold. *

* Not related to interrupts operation but used for counter/timer operations.

- An interrupt will be serviced as long as an interrupt of equal or higher priority is not already being serviced. If an interrupt of equal or higher level priority is

being serviced, the new interrupt will wait until it is finished before being serviced.

- If a lower priority level interrupt is being serviced, it will be stopped and the new interrupt serviced. When the new interrupt is finished, the lower priority level interrupt that was stopped will be completed.
- Following is an example of usage of interrupts assuming 12MHz crystal is used. This code generates two different frequencies, which are 500 Hz at P1.6 and 7 kHz at P1.7.

```
ORG     0000H               ;Reset entry point
        LJMP Main
        ORG  000BH          ;T0 interrupt vector
        LJMP T0ISR
        ORG  001BH          ;T1 interrupt vector
        LJMP T1ISR
        ORG  0030H
                            ;Main starts here
Main:   MOV  TMOD,#12H      ;T1 mode 1, T0 mode 2
        MOV  TH0,#-71       ;7-kHz using T0
        MOV  IE,#8AH        ;Enable T0 and T1 INT
        SETB TR0            ;Start Timer 0
        SETB TF1            ;Force T1 int
        SJMP $              ;Do nothing
T0ISR:  CPL  P1.7           ;7-kHz at P1.7
        RETI
T1ISR:  CLR  TR1            ;Stop T1
        MOV  TH1,#0FCH      ;HIGH(-1000)
        MOV  TL1,#18H       ;LOW(-1000), 1 ms for T1
        SETB TR1            ;Start Timer 1
        CPL  P1.6           ;500Hz at P1.6
        RETI
```

10.3 Pre Lab:

- Read carefully all the material given in the introduction and solve the following:
    - Write an instruction to enable the External 0 and Timer 1 interrupt.
    - Write an instruction sequence to make the External 1 interrupt the highest priority.
- If Timer 0 and External 1 interrupts occur at the same time, which interrupt will be serviced first?
- Are these interrupts synchronous to the system clock?

10.4 Lab Work:

A simple air-conditioning system needs to be built. In this system, two temperature sensors are connected with the two external interrupts of the 89S52 microcontroller, one for HOT and one for COLD. According to the input from these sensors, microcontroller will either turn ON or OFF the air-conditioning system and keep the temperature at a comfortable level.

When the temp increases, the sensor for HOT sends a signal to the external interrupt EX0, microcontroller will turn ON the A/C.

When the temp decreases, the sensor for COLD sends a signal to the external interrupt EX1, microcontroller will turn OFF the A/C.

A fan will be used to represent the air conditioning system and it will be connected through a relay installed on 8051 trainer board. There are four relays on the board as shown in Figure 10.4. These relays can be operated by sending 1 to the corresponding bit of $FF07_H$ memory location in the external RAM as shown in Table 10.2.

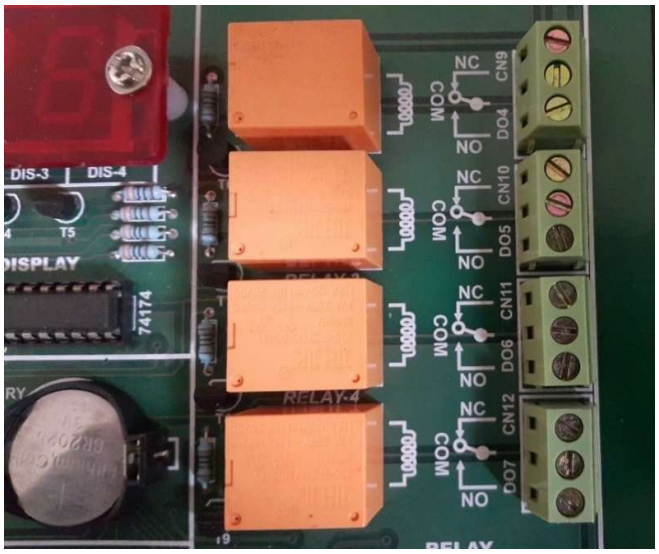


Figure 10.4: Relays available on the 8051 trainer board.

Location	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$FF07_H$	x	x	x	x	Relay 4	Relay 3	Relay 2	Relay 1

Table 10.2: Relay Selection.

In the relays, there are usually three terminals commonly known as normally open (NO), normally closed (NC), and common (COM). There is a switch at COM terminal which is connected with NC terminal when the relay is not energized. However, when the relay is energized this switch gets connected with NO terminal.

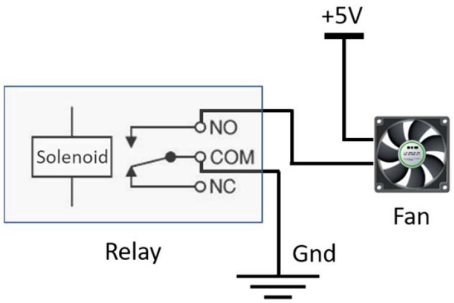


Figure 10.5: Circuit diagram to connect fan.

17) Connect a fan as shown in Figure 10.5.

18) Write a code for this system that when the HOT sensor sends a low pulse at EX0, it should turn ON the fan (A/C) and when the COLD sensor sends a low pulse at EX1, it should turn OFF the fan (A/C). Properly initialize the SFRs related to interrupts in the code.

19) Build and simulate the program in MIDE-51. This will generate the HEX file.

20) Execute the program and see the output using MIDE-51.

21) Connect the 8051 trainer board (power and data cords) with PC.

22) Download the "HEX" file to 89S52 microcontroller available on trainer board. See detailed process in Experiment-3 manual.

23) Run the program to verify the execution of the program using 8051 board.

10.5 Lab Report should contain:

(i) Prelab
(ii) Report with objective and lab-programs/lab-work with results and conclusion

Appendix A

8051 Instruction Set

The instructions are grouped into 5 groups

- Arithmetic
- Logic
- Data Transfer
- Boolean
- Branching

A complete list of all instructions in each of the above 5 groups is shown from the next page.

Notes on Data Addressing Modes

Rn - Working register R0-R7
direct - 128 internal RAM locations, any l/O port, control or status register
@Ri - Indirect internal or external RAM location addressed by register R0 or R1
#data - 8-bit constant included in instruction
#data 16 - 16-bit constant included as bytes 2 and 3 of instruction
bit - 128 software flags, any bitaddressable l/O pin, control or status bit
A – Accumulator

Notes on Program Addressing Modes

addr16 - Destination address for LCALL and LJMP may be anywhere within the 64-Kbyte program memory address space.
addr11 - Destination address for ACALL and AJMP will be within the same 2-Kbyte page of program memory as the first byte of the following instruction.
rel - SJMP and all conditional jumps include an 8 bit offset byte. Range is + 127/– 128 bytes relative to the first byte of the following instruction.

Instruction Set Summary

Mnemonic		Description	Byte	Cycle
Arithmetic Operations				
ADD	A,Rn	Add register to accumulator	1	1
ADD	A,direct	Add direct byte to accumulator	2	1
ADD	A, @Ri	Add indirect RAM to accumulator	1	1
ADD	A,#data	Add immediate data to accumulator	2	1
ADDC	A,Rn	Add register to accumulator with carry flag	1	1
ADDC	A,direct	Add direct byte to A with carry flag	2	1
ADDC	A, @Ri	Add indirect RAM to A with carry flag	1	1
ADDC	A, #data	Add immediate data to A with carry flag	2	1
SUBB	A,Rn	Subtract register from A with borrow	1	1
SUBB	A,direct	Subtract direct byte from A with borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with borrow	1	1
SUBB	A,#data	Subtract immediate data from A with borrow	2	1
INC	A	Increment accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
DEC	A	Decrement accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
INC	DPTR	Increment data pointer	1	2
MUL	AB	Multiply A and B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal adjust accumulator	1	1

Instruction Set Summary (cont'd)

Mnemonic		Description	Byte	Cycle
Logic Operations				
ANL	A,Rn	AND register to accumulator	1	1
ANL	A,direct	AND direct byte to accumulator	2	1
ANL	A,@Ri	AND indirect RAM to accumulator	1	1
ANL	A,#data	AND immediate data to accumulator	2	1
ANL	direct,A	AND accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to accumulator	1	1
ORL	A,direct	OR direct byte to accumulator	2	1
ORL	A,@Ri	OR indirect RAM to accumulator	1	1
ORL	A,#data	OR immediate data to accumulator	2	1
ORL	direct,A	OR accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive OR register to accumulator	1	1
XRL	A direct	Exclusive OR direct byte to accumulator	2	1
XRL	A,@Ri	Exclusive OR indirect RAM to accumulator	1	1
XRL	A,#data	Exclusive OR immediate data to accumulator	2	1
XRL	direct,A	Exclusive OR accumulator to direct byte	2	1
XRL	direct,#data	Exclusive OR immediate data to direct byte	3	2
CLR	A	Clear accumulator	1	1
CPL	A	Complement accumulator	1	1
RL	A	Rotate accumulator left	1	1
RLC	A	Rotate accumulator left through carry	1	1
RR	A	Rotate accumulator right	1	1
RRC	A	Rotate accumulator right through carry	1	1
SWAP	A	Swap nibbles within the accumulator	1	1

Instruction Set Summary (cont'd)

Mnemonic		Description	Byte	Cycle
Data Transfer				
MOV	A,Rn	Move register to accumulator	1	1
MOV	A,direct *)	Move direct byte to accumulator	2	1
MOV	A,@Ri	Move indirect RAM to accumulator	1	1
MOV	A,#data	Move immediate data to accumulator	2	1
MOV	Rn,A	Move accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct byte	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri, #data	Move immediate data to indirect RAM	2	1
MOV	DPTR, #data16	Load data pointer with a 16-bit constant	3	2
MOVC	A,@A + DPTR	Move code byte relative to DPTR to accumulator	1	2
MOVC	A,@A + PC	Move code byte relative to PC to accumulator	1	2
MOVX	A,@Ri	Move external RAM (8-bit addr.) to A	1	2
MOVX	A,@DPTR	Move external RAM (16-bit addr.) to A	1	2
MOVX	@Ri,A	Move A to external RAM (8-bit addr.)	1	2
MOVX	@DPTR,A	Move A to external RAM (16-bit addr.)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with accumulator	1	1
XCH	A,direct	Exchange direct byte with accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with accumulator	1	1
XCHD	A,@Ri	Exchange low-order nibble indir. RAM with A	1	1

*) MOV A,ACC is not a valid instruction

Instruction Set Summary (cont'd)

Mnemonic		Description	Byte	Cycle
Boolean Variable Manipulation				
CLR	C	Clear carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set carry flag	1	1
SETB	bit	Set direct bit	2	1
CPL	C	Complement carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to carry flag	2	2
ANL	C,/bit	AND complement of direct bit to carry	2	2
ORL	C,bit	OR direct bit to carry flag	2	2
ORL	C,/bit	OR complement of direct bit to carry	2	2
MOV	C,bit	Move direct bit to carry flag	2	1
MOV	bit,C	Move carry flag to direct bit	2	2
Program and Machine Control				
ACALL	addr11	Absolute subroutine call	2	2
LCALL	addr16	Long subroutine call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute jump	2	2
LJMP	addr16	Long iump	3	2
SJMP	rel	Short jump (relative addr.)	2	2
JMP	@A + DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if accumulator is zero	2	2
JNZ	rel	Jump if accumulator is not zero	2	2
JC	rel	Jump if carry flag is set	2	2
JNC	rel	Jump if carry flag is not set	2	2
JB	bit,rel	Jump if direct bit is set	3	2
JNB	bit,rel	Jump if direct bit is not set	3	2
JBC	bit,rel	Jump if direct bit is set and clear bit	3	2
CJNE	A,direct,rel	Compare direct byte to A and jump if not equal	3	2

Appendix B

8051 Special Function Registers

SYMBOL	DESCRIPTION	DIRECT ADDRESS	BIT ADDRESS, SYMBOL, OR ALTERNATIVE PORT FUNCTION MSB							LSB	RESET VALUE
ACC*	Accumulator	E0H	E7	E6	E5	E4	E3	E2	E1	E0	00H
AUXR#	Auxiliary	8EH	–	–	–	–	–	–	–	AO	xxxxxxx0B
AUXR1#	Auxiliary 1	A2H	–	–	–	LPEP[2]	WUPD	0	–	DPS	xxx000x0B
B*	B register	F0H	F7	F6	F5	F4	F3	F2	F1	F0	00H
DPTR:	Data Pointer (2 bytes)										
DPH	Data Pointer High	83H									00H
DPL	Data Pointer Low	82H									00H
			AF	AE	AD	AC	AB	AA	A9	A8	
IE*	Interrupt Enable	A8H	EA	–	ET2	ES	ET1	EX1	ET0	EX0	0x000000B
			BF	BE	BD	BC	BB	BA	B9	B8	
IP*	Interrupt Priority	B8H	–	–	PT2	PS	PT1	PX1	PT0	PX0	xx000000B
			B7	B6	B5	B4	B3	B2	B1	B0	
IPH#	Interrupt Priority High	B7H	–	–	PT2H	PSH	PT1H	PX1H	PT0H	PX0H	xx000000B
			87	86	85	84	83	82	81	80	
P0*	Port 0	80H	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	FFH
			97	96	95	94	93	92	91	90	
P1*	Port 1	90H	–	–	–	–	–	–	T2EX	T2	FFH
			A7	A6	A5	A4	A3	A2	A1	A0	
P2*	Port 2	A0H	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	FFH
			B7	B6	B5	B4	B3	B2	B1	B0	
P3*	Port 3	B0H	$\overline{RD}$	$\overline{WR}$	T1	T0	$\overline{INT1}$	$\overline{INT0}$	TxD	RxD	FFH
PCON#[1]	Power Control	87H	SMOD1	SMOD0	–	POF	GF1	GF0	PD	IDL	00xx0000B
			D7	D6	D5	D4	D3	D2	D1	D0	
PSW*	Program Status Word	D0H	CY	AC	F0	RS1	RS0	OV	–	P	000000x0B
RACAP2H#	Timer 2 Capture High	CBH									00H
RACAP2L#	Timer 2 Capture Low	CAH									00H
SADDR#	Slave Address	A9H									00H
SADEN#	Slave Address Mask	B9H									00H
SBUF	Serial Data Buffer	99H									xxxxxxxxB
			9F	9E	9D	9C	9B	9A	99	98	
SCON*	Serial Control	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	00H
SP	Stack Pointer	81H									07H
			8F	8E	8D	8C	8B	8A	89	88	
TCON*	Timer Control	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	00H
			CF	CE	CD	CC	CB	CA	C9	C8	
T2CON*	Timer 2 Control	C8H	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$	00H
T2MOD#	Timer 2 Mode Control	C9H	–	–	–	–	–	–	T2OE	DCEN	xxxxxx00B
TH0	Timer High 0	8CH									00H
TH1	Timer High 1	8DH									00H
TH2#	Timer High 2	CDH									00H
TL0	Timer Low 0	8AH									00H
TL1	Timer Low 1	8BH									00H
TL2#	Timer Low 2	CCH									00H
TMOD	Timer Mode	89H	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0	00H

* SFRs are bit addressable.
\# SFRs are modified from or added to the 80C51 SFRs.
– Reserved bits.
1. Reset value depends on reset source.
2. LPEP – Low Power EPROM operation (OTP/EPROM only)

Appendix C

ASCII Table (7-bit)

(ASCII = American Standard Code for Information Interchange)

Decimal	Octal	Hex	Binary	Value	
000	000	000	00000000	NUL	(Null char.)
001	001	001	00000001	SOH	(Start of Header)
002	002	002	00000010	STX	(Start of Text)
003	003	003	00000011	ETX	(End of Text)
004	004	004	00000100	EOT	(End of Transmission)
005	005	005	00000101	ENQ	(Enquiry)
006	006	006	00000110	ACK	(Acknowledgment)
007	007	007	00000111	BEL	(Bell)
008	010	008	00001000	BS	(Backspace)
009	011	009	00001001	HT	(Horizontal Tab)
010	012	00A	00001010	LF	(Line Feed)
011	013	00B	00001011	VT	(Vertical Tab)
012	014	00C	00001100	FF	(Form Feed)
013	015	00D	00001101	CR	(Carriage Return)
014	016	00E	00001110	SO	(Shift Out)
015	017	00F	00001111	SI	(Shift In)
016	020	010	00010000	DLE	(Data Link Escape)
017	021	011	00010001	DC1	(XON) (Device Control 1)
018	022	012	00010010	DC2	(Device Control 2)
019	023	013	00010011	DC3	(XOFF) (Device Control 3)
020	024	014	00010100	DC4	(Device Control 4)
021	025	015	00010101	NAK	(Negative Acknowledgement)
022	026	016	00010110	SYN	(Synchronous Idle)
023	027	017	00010111	ETB	(End of Trans. Block)
024	030	018	00011000	CAN	(Cancel)
025	031	019	00011001	EM	(End of Medium)
026	032	01A	00011010	SUB	(Substitute)
027	033	01B	00011011	ESC	(Escape)
028	034	01C	00011100	FS	(File Separator)
029	035	01D	00011101	GS	(Group Separator)
030	036	01E	00011110	RS	(Request to Send)(Record Separator)
031	037	01F	00011111	US	(Unit Separator)
032	040	020	00100000	SP	(Space)
033	041	021	00100001	!	(exclamation mark)
034	042	022	00100010	"	(double quote)
035	043	023	00100011	#	(number sign)
036	044	024	00100100	$	(dollar sign)
037	045	025	00100101	%	(percent)
038	046	026	00100110	&	(ampersand)

039	047	027	00100111	'	(single quote)
040	050	028	00101000	(	(left/opening parenthesis)
041	051	029	00101001	)	(right/closing parenthesis)
042	052	02A	00101010	*	(asterisk)
043	053	02B	00101011	+	(plus)
044	054	02C	00101100	,	(comma)
045	055	02D	00101101	-	(minus or dash)
046	056	02E	00101110	.	(dot)
047	057	02F	00101111	/	(forward slash)
048	060	030	00110000	0	
049	061	031	00110001	1	
050	062	032	00110010	2	
051	063	033	00110011	3	
052	064	034	00110100	4	
053	065	035	00110101	5	
054	066	036	00110110	6	
055	067	037	00110111	7	
056	070	038	00111000	8	
057	071	039	00111001	9	
058	072	03A	00111010	:	(colon)
059	073	03B	00111011	;	(semi-colon)
060	074	03C	00111100	<	(less than)
061	075	03D	00111101	=	(equal sign)
062	076	03E	00111110	>	(greater than)
063	077	03F	00111111	?	(question mark)
064	100	040	01000000	@	(AT symbol)
065	101	041	01000001	A	
066	102	042	01000010	B	
067	103	043	01000011	C	
068	104	044	01000100	D	
069	105	045	01000101	E	
070	106	046	01000110	F	
071	107	047	01000111	G	
072	110	048	01001000	H	
073	111	049	01001001	I	
074	112	04A	01001010	J	
075	113	04B	01001011	K	
076	114	04C	01001100	L	
077	115	04D	01001101	M	
078	116	04E	01001110	N	
079	117	04F	01001111	O	
080	120	050	01010000	P	
081	121	051	01010001	Q	
082	122	052	01010010	R	
083	123	053	01010011	S	
084	124	054	01010100	T	
085	125	055	01010101	U	
086	126	056	01010110	V	

087	127	057	01010111	W	
088	130	058	01011000	X	
089	131	059	01011001	Y	
090	132	05A	01011010	Z	
091	133	05B	01011011	[	(left/opening bracket)
092	134	05C	01011100	\	(back slash)
093	135	05D	01011101	]	(right/closing bracket)
094	136	05E	01011110	^	(caret/circumflex)
095	137	05F	01011111	_	(underscore)
096	140	060	01100000	`	
097	141	061	01100001	a	
098	142	062	01100010	b	
099	143	063	01100011	c	
100	144	064	01100100	d	
101	145	065	01100101	e	
102	146	066	01100110	f	
103	147	067	01100111	g	
104	150	068	01101000	h	
105	151	069	01101001	i	
106	152	06A	01101010	j	
107	153	06B	01101011	k	
108	154	06C	01101100	l	
109	155	06D	01101101	m	
110	156	06E	01101110	n	
111	157	06F	01101111	o	
112	160	070	01110000	p	
113	161	071	01110001	q	
114	162	072	01110010	r	
115	163	073	01110011	s	
116	164	074	01110100	t	
117	165	075	01110101	u	
118	166	076	01110110	v	
119	167	077	01110111	w	
120	170	078	01111000	x	
121	171	079	01111001	y	
122	172	07A	01111010	z	
123	173	07B	01111011	{	(left/opening brace)
124	174	07C	01111100	|	(vertical bar)
125	175	07D	01111101	}	(right/closing brace)
126	176	07E	01111110	~	(tilde)
127	177	07F	01111111	DEL	(delete)

Equipment Needed for Microcontroller Lab

The following items are needed for each station in the Lab.

- A PC with serial port and windows operating system
- 89C51 designer board LAB PRO-51
- 89C51 PC-based debugger/simulator (Pinnacle 52)
- 74245 Buffer chip
- Relay (SPDT or DPDT)
- Reed switch, and magnet
- 12 V Alarm horn.
- Diode 1N4001,
- Power transistor NPN type TIP120.
- 6 V DC small fan
- 6 V DC motor
- Resistors 10K, 1K, 2K, and 100Ω.