EL303-MICROPROCESSOR INTERFACING & PROGRAMMING

LAB MANUAL

DEPARTMENT OF ELECTRICAL ENGINEERING,

FAST-NU, LAHORE

Created by: Mr. Muhammad Ahmad Raza

Date: 10th Dec, 2013

Last Updated by: Maimoona Akram

Date: January, 2020

Approved by the HoD: Dr. Usman Shahid

Date: January, 2020

Table of Contents

List of Equipment

Sr. No.	Description
1	AT89C52 Microcontrollers
2	RIMS 8051 Based Microcontroller Trainers
3	Computer System
4	Keil Integrated Development Environment Version-3
5	EZDL4.EXE Programmer
6	Proteus Simulator
7	7-Segment Displays
8	Telephone Keypads
9	6V DC Motors

EXPERIMENT -1

KEIL µVISION-3 IDE & TRAINER BOARD

OBJECTIVES:

- To learn to compile and debug a program in Keil µVision 3IDE.
- To be able to download the program to the Flash memory of 89C52 microcontroller using the Trainer Board and the PC usb port.
- To run a simple code using the trainer board.

EQUIPMENT/TOOLS:

- ATMEL ® 89C52 Microcontroller.
- Microprocessor Trainer Board.

INTRODUCTION TO THE UNIVERSAL MICROCONTROLLER TRAINER: This section would give you a brief introduction of the IT-4310 Trainer as shown in Figure 1.1 below.

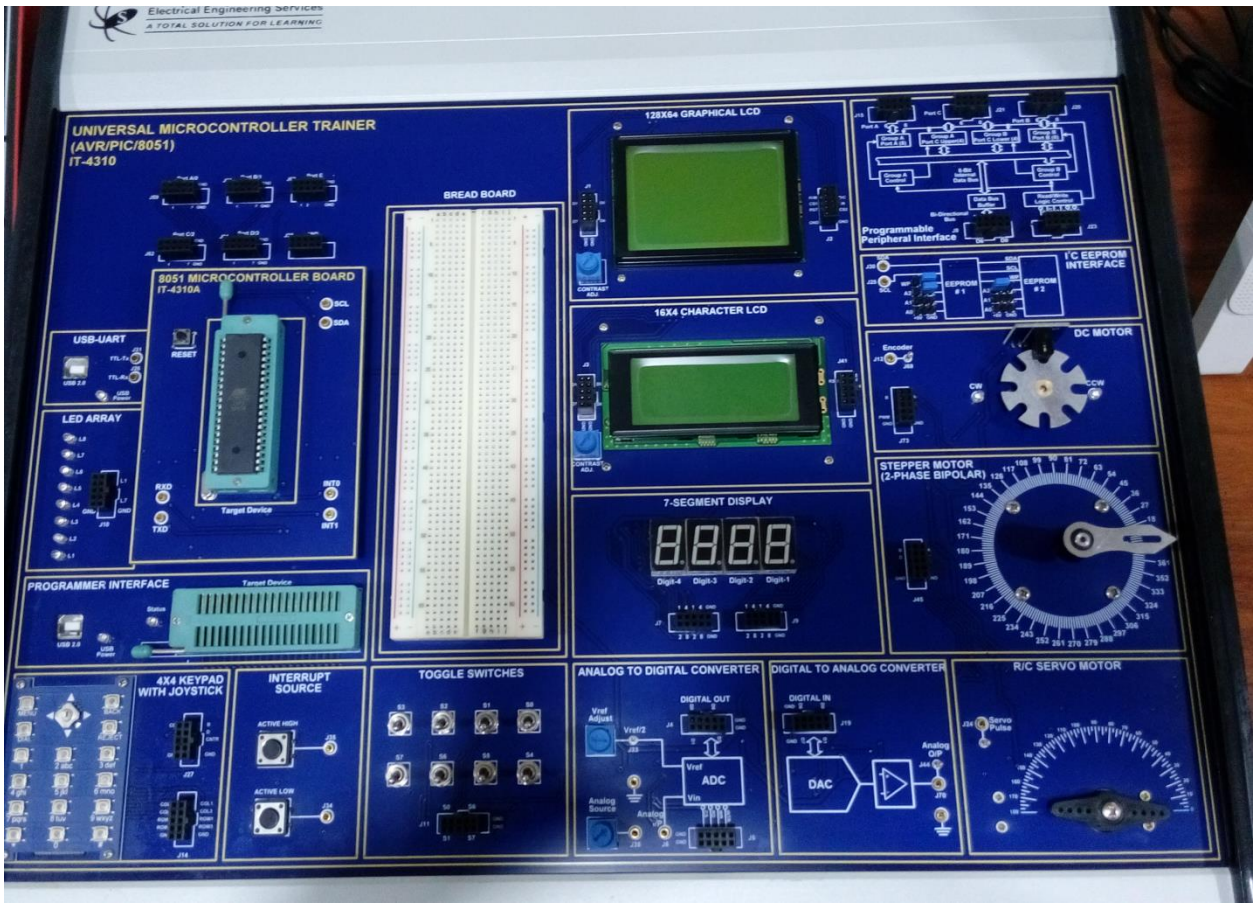


Figure 1.1 Introduction of the IT-4310 trainer

The IT-4310is explained below:

1. **LCD DISPLAY:** Character LCD Display for 16x4and 128x64 Graphical LCD
2. **LED ARRAY:** There are four LEDs L1-L8
3. **TOGGLE SWITCHES:** There are eight toggle switches S1-S8
4. **ANALOG TO DIGITAL CONVERTER:** It has an analog input source and a digital output and Voltage Reference Adjust and ADC chip mounted inside the trainer
5. **DIGITAL TO ANALOG CONVERTER:** It has A digital Input part and analog output part and a DAC mounted inside the trainer.

KEIL µVISION-3IDE:

STEPS FOR CREATING THE FIRST PROJECT:

- Keil µVision3 is a standard Windows application and started by clicking on the program icon. To create a new project file select from the µVision3 menu **Project – New Project…** Create a new folder and enter the file name for the new project, e.g. **Project1**. µVision3 creates a new project file with the name **PROJECT1.UV2** that contains a default target and file group name. You can see these names in the **Project Window – Files**.
- Now use from the menu **Project – Select Device for Target** and select a CPU for your project. The **Select Device** dialog box shows the µVision3 device database. Just select the microcontroller you use. We are using for our examples the **AT89C52/ AT89S52**. This selection sets necessary tool options for the 89C52 device and simplifies in this way the tool configuration.
- You may create a new source file with the menu option **File – New**. This opens an empty editor window where you can enter your source code. Create a new file called **Project1.asm.**
- Once you have created your source file you can add this file to your project. µVision3 offers several ways to add source files to a project. For example, you can select the file group in the **Project Window – Files** page and click with the right mouse key to open a local menu. The option **Add Files** opens the standard files dialog. Select the file **Project1.asm** that you have just created.
- Typically, the tool settings under **Options – Target** are all you need to start a new application. When you build an application with syntax errors, µVision3 will display errors and warning messages in the **Output Window – Build** page. A double click on a message line opens the source file on the correct location in a µVision3 editor window. • Once you have successfully generated your application you can start debugging.
- After you have tested your application, it is required to create an Intel HEX file to download the software into an EPROM programmer or simulator. µVision3 creates HEX files with each build process when **Create HEX file** under **Options for Target – Output** is enabled (as shown in the figure below).

STEPS TO DOWNLOAD THE PROJECT TO THE MICRO-CONTROLLER MEMORY:

- Connect the usb cable between the PC usb port and the Trainer Board. If the usb port is connected then the USB Power LED will glow.
- Insert the micro controller in the Programmer Interface.
- Run the program "Universal Programmer" from the PC.
- Follow the instructions below to download the HEX file of the project to the microcontroller.

I. Click on the "LOAD" icon in the top bar as shown in the figure 1.2 below:

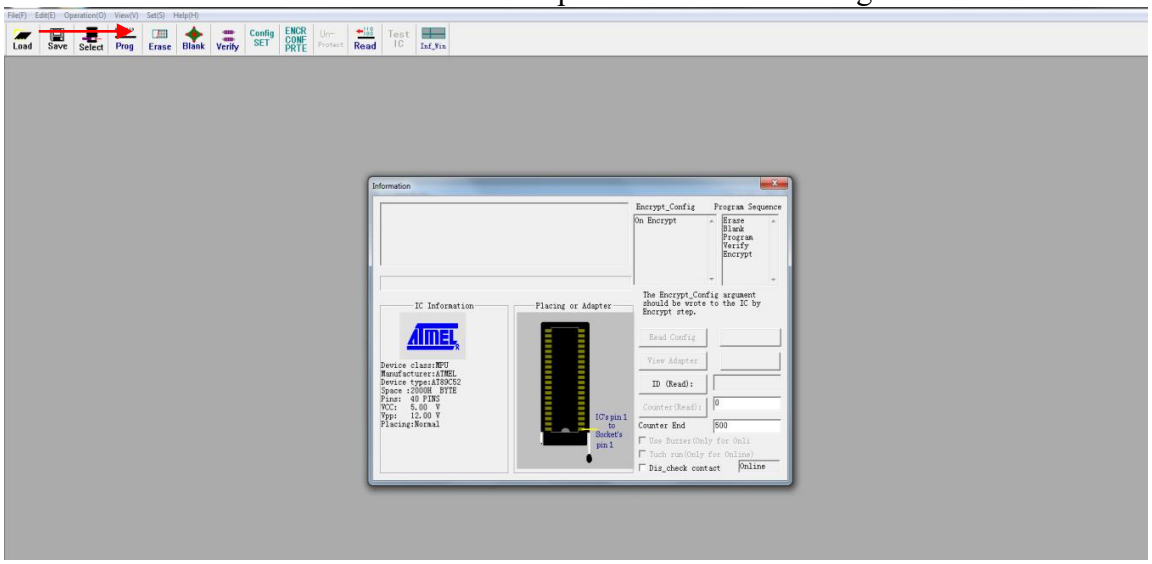


Figure 1.2 Click on LOAD icon

II. Select the ".hex" file that you want to load on the microcontroller as shown in figure 1.3 below:

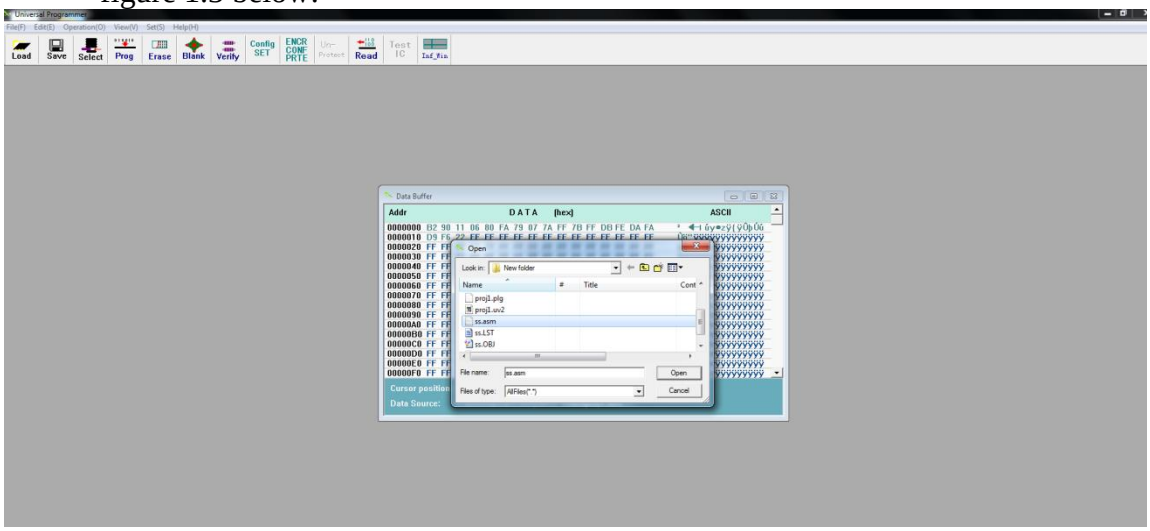


Figure 1.3 Load Hex file on microcontroller

III. Check the file information section of "Hex (intel)" as shown in Figure 1.4 below.

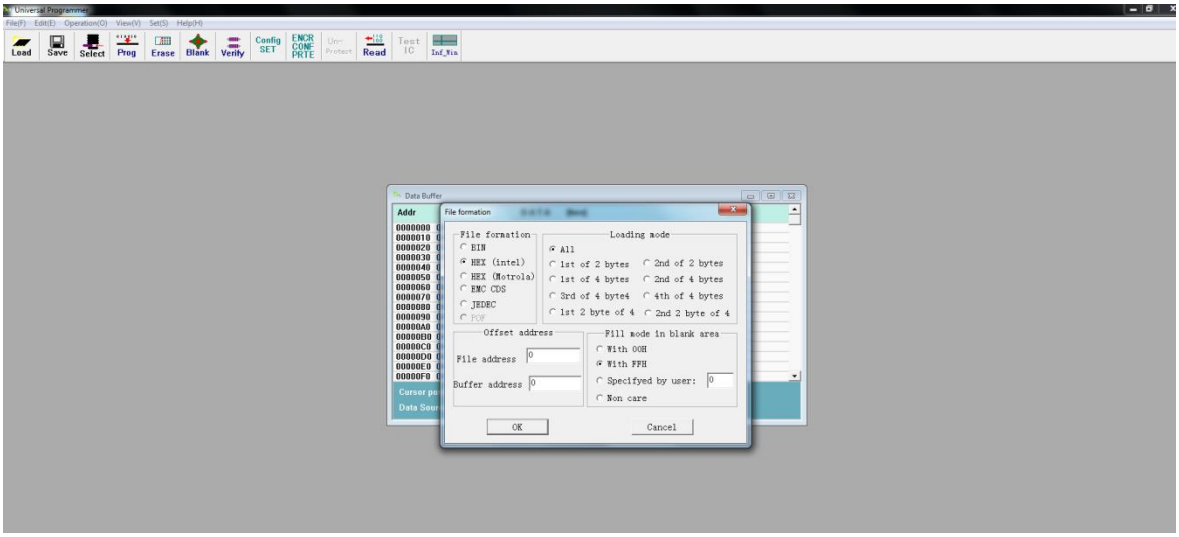


Figure 1.4 Checking file information

IV. Click on the "Prog" icon to finish the process of loading the program on microcontroller as shown in Figure 1.5 below.

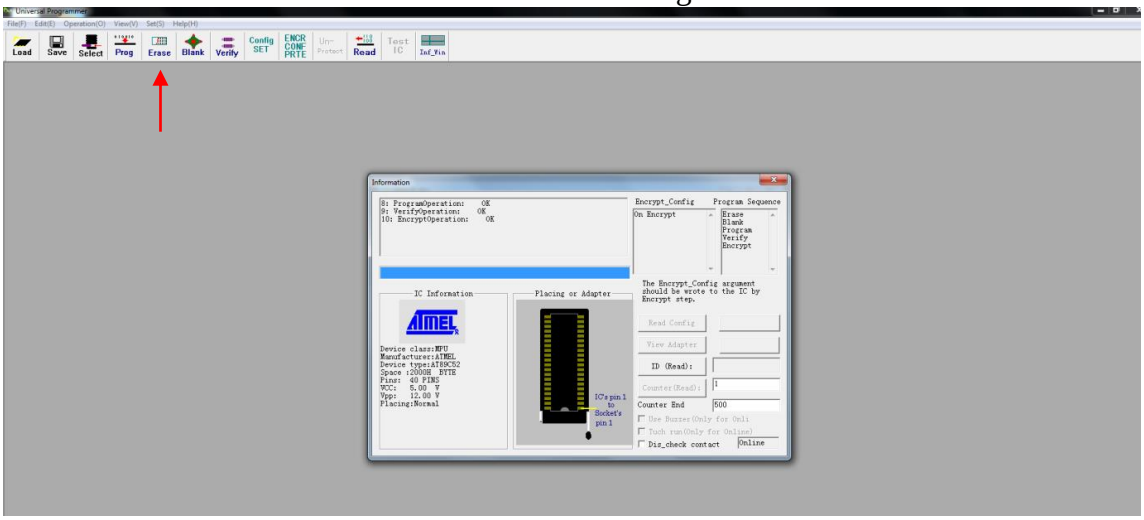


Figure 1.5 Clicking prog to finish the process of loading

PROCEDURE:

Type the following code in a *.asm* file and follow the above procedure to compile yourfirst project, and then program the 89C52 micro controller.

```
ORG 0H
STARTLOOP:  CPL P1.0
            CALL DELAY
            JMP STARTLOOP
DELAY:
            MOV R0, #7
      LOOP1: MOV R1, #255
      LOOP2: MOV R2, #255
            DJNZ R2, $
            DJNZ R1, LOOP2
            DJNZ R0, LOOP1
      RET
END
```

HARDWARE SETUP

- Connect the Pin of P1.0 to an LED socket on the board.
- Insert the microcontroller into the proper socket (28 in **Figure-1**).
- Power up the board.
- The LED should blink once a second.
- You can test the output on the oscilloscope as well.

POST LAB QUESTIONS:

Answer the following questions:

1. What is the difference between microprocessor and micro-controller?
2. What is an instruction?
3. What is an instruction set?
4. Define machine cycle and explain in the context of 8051
5. What are the different groups of instructions supported by 8051?

EXPERIMENT -2

DATA MOVEMENT WITHIN THE REGISTERS BANKS USING DIRECT AND REGISTER ADDRESSING MODES

OBJECTIVES:

- To understand the RAM organization of 8051 microcontroller.
- To be able to use direct and register addressing modes to store and retrieve data in the RAM.

EQUIPMENT/TOOLS:

- KeilµVision3IDE

BACKGROUND:

8051 RAM ORGANIZATION:

There are 128 bytes of RAM in the8051 assigned addresses 00 to 7FH. The 128 bytes are divided into three different groups as follows:

1. **REGISTER BANKS:**

A total of 32 bytes from locations 00 to 1F hex are set aside for register banks and the stack. These 32 bytes are divided into 4 banks of registers in which each bank has 8 registers, R0-R7 RAM location from 0 to 7 are set aside for bank 0 of R0-R7 where R0 is RAM location 0, R1 is RAM location 1, R2 is RAM location 2, and so on, until memory location 7 which belongs to R7 of bank 0. It is much easier to refer to these RAM locations with names such as R0, R1, and so on, than by their memory locations. Register bank 0 is the default when 8051 is powered up We can switch to other banks by use of the PSW register. Bits D4 and D3 of the PSW are used to select the desired register bank as shown in Figure 2.1. Use the bit-addressable instructions SETB and CLR to access PSW.4 and PSW.3

PSW bank selection

	RS1(PSW.4)	RS0(PSW.3)
Bank 0	0	0
Bank 1	0	1
Bank 2	1	0
Bank 3	1	1

Figure 2.1 Register banks

2. **BIT-ADDRESSABLE LOCATIONS:**

The 8051 contains a total of 210 bit-addressable locations of which 128 are at locations 20H to 2FH while the rest are in the SFRs. Each of the 128 bits from 20H to 2FH have a unique number (address) attached to them, as shown in the table above. The 8051 instruction set allows you to set or reset any single bit in this section of RAM. With the general purpose RAM from 30H to 7FH and the register banks from 00H to 1FH, you may only read or write a full byte (8 bits) at these locations. However, with bit-addressable RAM (20H to 2FH) you can read or write any single

bit in this region by using the unique address for that bit. We will later see that this is a very powerful feature.

3. **SCRATCH PAD:**

A total of 80 bytes from locations 30H to 7FH are used for read and write storage, called scratch pad.

4. **SPECIAL FUNCTION REGISTERS (SFRs):**

Locations 80H to FFH contain the special function registers as shown in Figure 2.2. As you can see from the diagram above, not all locations are used by the 8051. These extra locations are used by other family members (8052, etc.) for the extra features these microcontrollers possess. Also note that not all SFRs are bit-addressable. Those that are have a unique address for each bit.

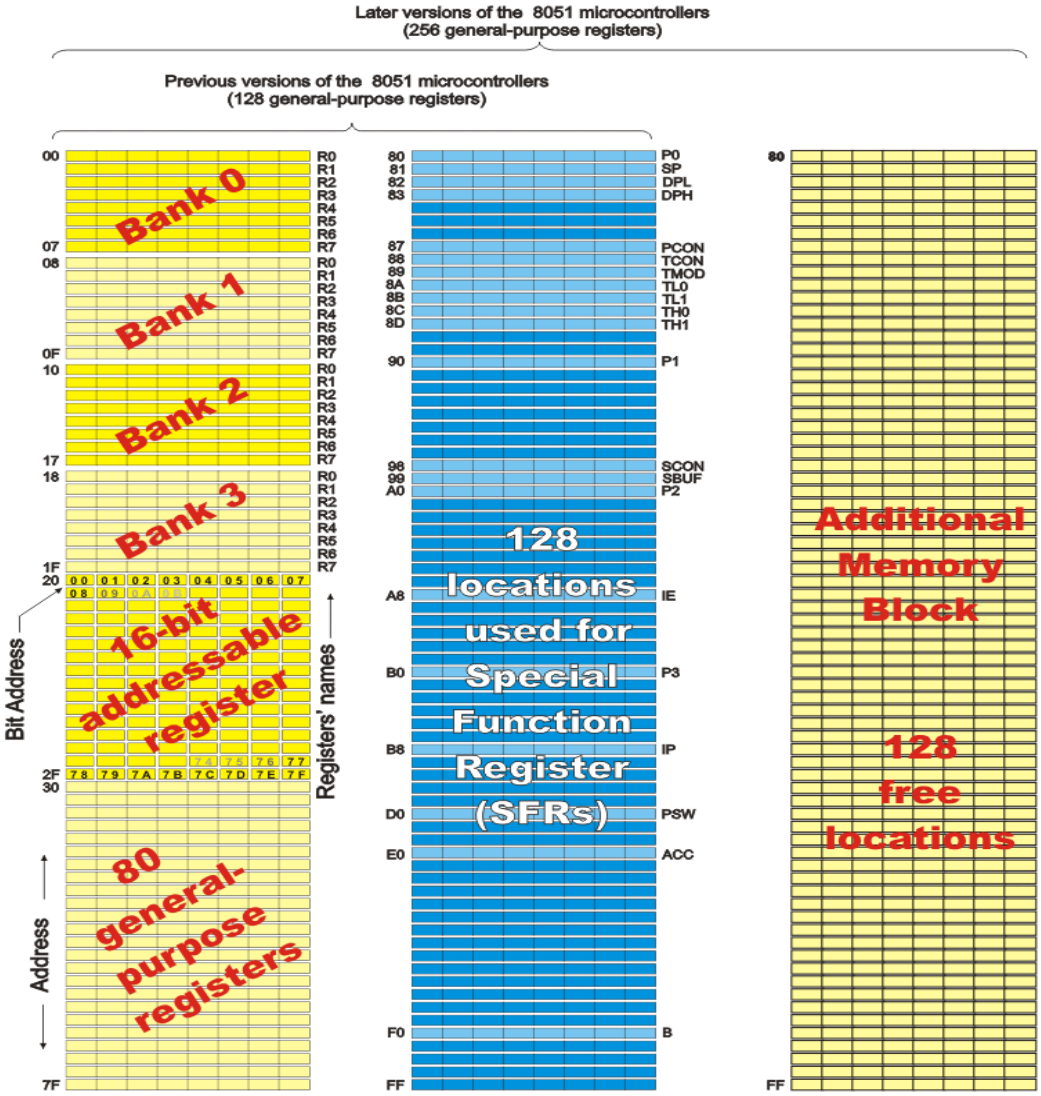


Figure 2.2 8051 RAM organization

How to view RAM in KEIL

1. Build your program by pressing F5, it should be error free.
2. Go to debug mode by pressing Ctrl+F5
3. Go to View tab, and select Memory Window
4. On bottom right of the complier a memory window will appear as shown in Figure 2.3. Replace C:0x0000 with D:0x(desired address). Values are shown from left to right with increasing addresses, starting with prescribed address

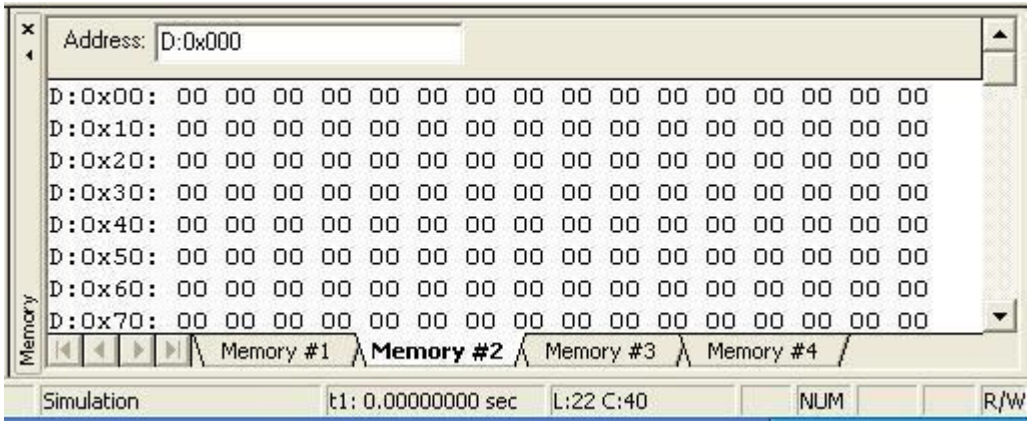


Figure 2.3 Memory window

EXERCISES:

1. Write an assembly language program to store eight numbers in Bank-0 (Immediate Addressing Mode).

2. Write an assembly language program to read eight numbers from Bank0 and copy the content in Bank1, Bank-2, and Bank-3, using register addressing mode.

3. Write an assembly language program to read eight numbers from Bank0 and copy the content in Bank1, Bank-2, and Bank-3, using direct addressing mode.

POST LAB QUESTIONS:

Answer the following questions:

1. What is the significance of SFRs?
2. What is Accumulator Register?
3. What are the flags supported by 8051 controller?
4. Write a piece of code to demonstrate bank switching that shows switching among all banks.

EXPERIMENT -3

DATA MOVEMENT WITHIN THE ON CHIP RAMUSING INDRIECT ADDRESSING MODE AND STACK

OBJECTIVES:

- To be able to use stack and indirect addressing mode to store and retrieve data in the RAM.
- To understand the use of indirect addressing mode to access the extra 128 bytes of RAM available in 89C52.

EQUIPMENT/TOOLS:

- KeilµVision3IDE

BACKGROUND:

89C52 has additional 128 bytes of RAM as compared to 89C51 as shown in Figure 3.1 below. The additional 128 RAM locations block shares same address range (80H - FFH) as that of SFR block in RAM as shown in figure below. Therefore, in order to differentiate that either we want to store and/or retrieve data from SFR block or extra 128-byte RAM, addressing mode plays an important role. If it is required to access memory location 80H in extra 128-byte RAM block, then register indirect addressing mode will be used. However, if it is required to access SFR at 80H in SFR block then direct, or register addressing mode will be used. R0 and R1 are the only registers that can be used as pointers in register indirect addressing mode.

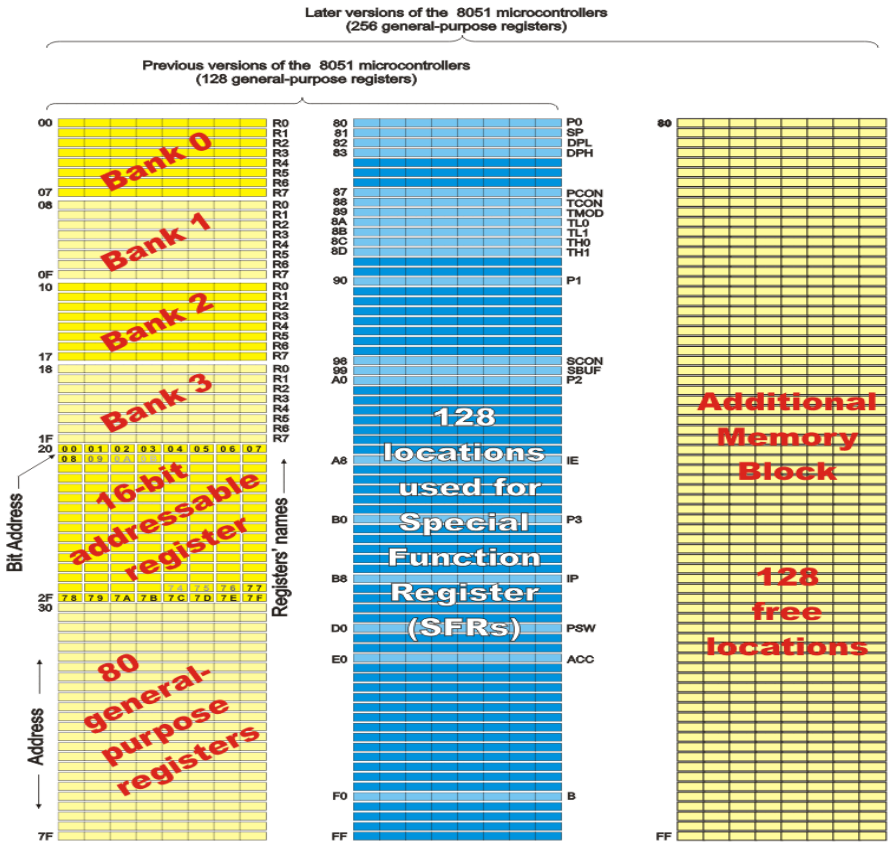


Figure 3.1 8051 RAM organization

STACK:

- The stack is a section of RAM used by the CPU to store information temporarily. This information could be data or an address. The register used to access the stack is called the SP (stack pointer) register. The stack pointer in the 8051 is only 8 bit wide, which means that it can take value of 00 to FFH.
- The storing of a CPU register in the stack is called a **PUSH**. The register SP is pointing to the last used location of the stack. As we push data onto the stack, the SP is incremented by one. This is different from many microprocessors.
- Loading the contents of the stack back into a CPU register is called a **POP**. With every pop, the top byte of the stack is copied to the register specified by the instruction and the stack pointer is decremented once.
- The CPU also uses the stack to save the address of the instruction just below the CALL instruction. This is how the CPU knows where to resume when it returns from the called subroutine.

EXERCISES:

1. Write an assembly language program that uses stack to store following values on the designated addresses:

Address	Value to Stored
0x30	0x12
0x35	0x34
0x3A	0x56
0x3F	0x78
0x40	0x1A
0x46	0x2B
0x4B	0x3C
0x51	0x4D
0x5D	0x5E
0x5F	0x6F

2. Now read the values stored in stack and copy them in new RAM locations using stack as directed in the following table:

Address	Value to Stored
0x60	0x12
0x61	0x34
0x62	0x56
0x63	0x78
0x64	0x1A
0x65	0x2B
0x66	0x3C
0x67	0x4D
0x68	0x5E
0x69	0x6F

3. Perform first two exercises using register indirect addressing mode.

POST LAB QUESTIONS:

1. What is the use of scratch pad area of internal RAM of 8051?
2. Write a program to copy the content of lower RAM locations 60H - 6FH to upper RAM locations C8H – D7H in 89C52.

.

EXPERIMENT -4

DELAY GENERATION AND USE OF KIEL DEBUGGER

OBJECTIVES:

- To understand the use of branch instructions available in 8051 assembly language.
- To be able to generate a given amount of delay using nested loops such that error is minimum in the delay

EQUIPMENT/TOOLS:

- KeilµVision.3IDE

BACKGROUND:

Every microprocessor needs a clock to execute the program. CPU executing an instruction takes a certain number of clock cycles. These are referred to as machine cycles. The length of machine cycle depends on the frequency of the crystal oscillator connected to 8051. In original 8051, one machine cycle lasts 12 oscillator periods. We may use DJNZ instructions to make nested loops to generate desired amount of delay. Each DJNZ takes 2 machine cycles.

$$Oscillator\ frequency\ = f_{os}$$
$$Amount\ of\ delay\ required = t_0$$
$$No.of\ machine\ cycles\ required\ to\ generate\ to\ delay\ = \frac{f_{os} \times t_0}{12}$$
$$No.of\ DJNZs\ required\ to\ generate\ to\ delay\ = \frac{f_{os} \times t_0}{12 \times 2}$$

Following examples demonstrate how to compute the execution time in of a program:

Find the size of the delay in following program, if the crystal frequency is 11.0592MHz.

```
        MOV   A,#55H
AGAIN:  MOV   P1,A
        ACALL DELAY
        CPL   A
        SJMP  AGAIN
;---time delay-------
DELAY:  MOV   R3,#200
HERE:   DJNZ  R3,HERE
        RET
```

A simple way to short jump to itself in order to keep the microcontroller busy

```
HERE:  SJMP  HERE
```

We can use the following:

```
SJMP  $
```

Solution:

```
                          Machine cycle
DELAY:  MOV   R3,#200          1
HERE:   DJNZ  R3,HERE          2
        RET                    2
```

Therefore, [(200x2)+1+2]x1.085 μs = 436.255 μs.

Find the size of the delay in following program, if the crystal frequency is 11.0592MHz.

```
                          Machine Cycle
DELAY:  MOV   R2,#200          1
AGAIN:  MOV   R3,#250          1
HERE:   NOP                    1
        NOP                    1
        DJNZ  R3,HERE          2
        DJNZ  R2,AGAIN         2
        RET                    2
```

Notice in nested loop, as in all other time delay loops, the time is approximate since we have ignored the first and last instructions in the subroutine.

Solution:

For HERE loop, we have (4x250)x1.085 μs=1085 μs.
For AGAIN loop repeats HERE loop 200 times, so we have 200x1085 μs=217000 μs. But "MOV R3,#250" and "DJNZ R2,AGAIN" at the start and end of the AGAIN loop add (3x200x1.805)=651 μs.
As a result we have 217000+651=217651 μs.

Two factors can affect the accuracy ofthe delay:

1. Crystal frequency: The duration of the machine cycle is a function of this crystal frequency.
2. 8051 design: The original machine cycle duration was set at 12 clocks. Advances in both IC technology and CPU design in recent years have made

Clocks per machine cycle for various 8051 versions

Chip/Maker	Clocks per Machine Cycle
AT89C51 Atmel	12
P89C54X2 Philips	6
DS5000 Dallas Semi	4
DS89C420/30/40/50 Dallas Semi	1

Figure 4.1 Clocks per machine cycle for various 8051 versions

We can use debugger of our IDE to find the time taken by the processor to execute a program. Following figure 4.2 demonstrates

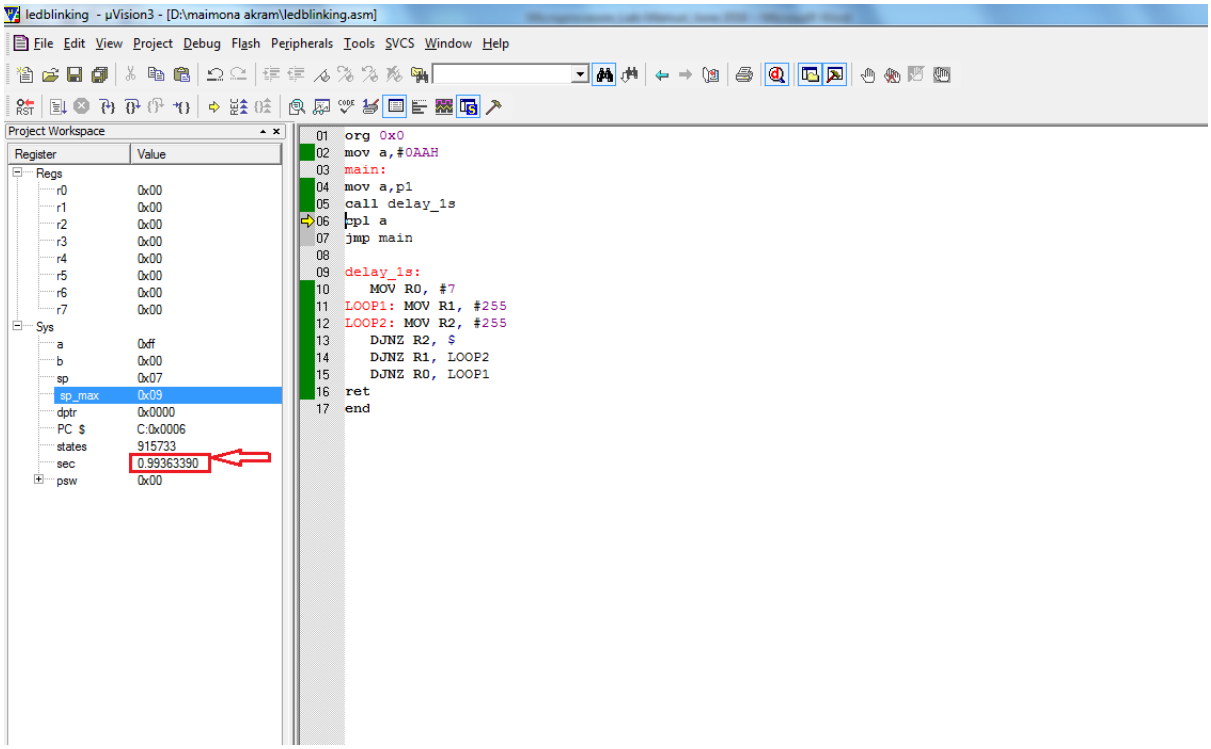


Figure 4.2 Debugger of IDE to find time taken by processor

EXERCISES:

1.Write an assembly language program to generate a delay of 5 seconds. Use debugger to measure actual delay and then compute the percentage error in your program. Now optimize your program to minimize the error in delay. The percentage error should be less than 10^{-3}.

2.Eight LEDs are required to glow according to the patterns mentioned in the table below. There are two groups of patterns. Each group consists of two patterns. You are required to design and implement an algorithm that switch between the two patterns every 2 seconds. Moreover, groups are switched every 8 seconds.

Note: The red color of LED represents HIGH logic and green color represents LOW logic.

Group#1	
Group#2	

EXPERIMENT- 5

ARITHMETIC INSTRUCTIONS

OBJECTIVES:

- To understand the operation of different arithmetic instructions available in 8051 assembly language.
- To be able to solve arithmetic expressions in assembly language of 8051 microcontroller.

EQUIPMENT/TOOLS:

- KeilµVision3IDE

BACKGROUND:

Following is a brief overview of the arithmetic instructions in 8051 **architecture.**

1. ADD INSTRUCTION:

Syntax: ADD A , source
Effect: A = A + source

The instruction ADD is used to add twooperands. Destination operand is always in register A Source operand can be a register, immediate data, or in memory.

2. SUBB INSTRUCTION:

Syntax: SUBB A , source
Effect: A = A – source – CY

In many microprocessors there are twodifferent instructions for subtraction:SUB and SUBB (subtract with borrow). In the 8051 we have only SUBB. The 8051 microcontroller uses adder circuitry to performthe subtraction. To make SUB out of SUBB, we have tomake CY=0 prior to the execution ofthe instruction. Notice that we use the CY flag for theborrowing.

SUBB when CY = 0

1. Take the 2's complement of the subtrahend (source operand)
2. Add it to the minuend (A)
3. Ignore the carry, if there is no overflow

If answer is negative it is stored in the form of 2's complement. For 8-bit 2's complement system following table 5.1 shows the range of +ev and –ev numbers:

Decimal	Binary	Hex
-128	1000 0000	80
-127	1000 0001	81
-126	1000 0010	82
...	... ...	...
-2	1111 1110	FE
-1	1111 1111	FF
0	0000 0000	00
+1	0000 0001	01
+2	0000 0010	02
...	... ...	...
+127	0111 1111	7F

Table 5.1 Ranges of +ve and –ve numbers

3. MUL INSTRUCTION:

Syntax: MUL AB
Effect: AxB, 16-bit result in B, A

The 8051 supports byte by byte multiplication only. The bytes are assumed to be unsigned data as shown in Figure 5.1.

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

Figure 5.1 Byte by byte multiplication

4. DIV INSTRUCTION:

Syntax: DIV AB
Effect: Divide A by B, A/B

The 8051 supports bytes over byte division only, the byte are assumed to be unsigned data as shown in Figure 5.2.

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

Figure 5.2 Byte by byte division

EXERCISE:

Write an assembly language program to solve following arithmetic expression. Test and use debugger of the KEIL µVision3 IDE to verify your program for given values of the variables. If verification fails, then give proper reason why your program gave wrong answer.

$$\{(2x + 3y + 3z)/(x + 5y - 3)\} * \{(4z + 3) - (3x + 1)\}$$

TEST VALUES:

1) $x = 2,\ y = 3,\ z = 5$
2) $x = 200,\ y = 3,\ z = 5$
3) $x = 100,\ y = 3,\ z = 5$

EXPERIMENT - 6

LOGICAL & BRANCH INSTRUCTIONS

OBJECTIVES:

- To understand BCD addition of packed BCD numbers.
- To understand usage of masking using logical instructions.
- To understand usage of **CJNE** comparison branch instruction.

EQUIPMENT/TOOLS:

- KeilμVision3IDE

BACKGROUND:

Following is a brief overview of the logical instructions in 8051 **architecture.**

1. **ANL INSTRUCTION:**
   Syntax: ANL A, source
   Effect: A = A AND source

The instruction ANL is used to perform bitwise logical AND on twooperands. Destination operand is always in either register A or direct RAM location. If A is used as destination, thensource operand can be a register, immediate data, or in memory (direct or indirect). If direct RAM location is used as destination, then source operand can be A or immediate data.

2. **ORL INSTRUCTION:**
   Syntax: ORL A, source
   Effect: A = A OR source

The instruction ORL is used to perform bitwise logical OR on twooperands. Destination operand is always in either register A or direct RAM location. If A is used as destination, thensource operand can be a register, immediate data, or in memory (direct or indirect). If direct RAM location is used as destination, then source operand can be A or immediate data.

3. **XRL INSTRUCTION:**
   Syntax: XRL A, source
   Effect: A = A XOR source

The instruction ANL is used to perform bitwise logical XOR on twooperands. Destination operand is always in either register A or direct RAM location. If A is used as destination, thensource operand can be a register, immediate data, or in memory (direct or indirect). If direct RAM location is used as destination, then source operand can be A or immediate data.

There is only one comparison branch instruction **CJNE** that compares the magnitudes of two operands and branches if their values are not equal.
Syntax: CJNE<destination byte>, <source byte>, rel
The carry flag is set if the unsigned integer value of destination is less than unsigned integer value of source. Otherwise, the carry flag is cleared. Neither operand is affected. If A is used as destination, thensource operand can be a register B, immediate data, or in memory(direct). If register (R0 – R7) or an indirect RAM location is used as destination, then source operand can be immediate data only.

EXERCISE:

Write an assembly language program that adds two packed BCD numbers. If the result is not BCD, then correct this problem. Summary of the algorithm for correction is given below:

a) If the lower nibble (4 bits) is greater than 9, or if AC=1, add 6 to the lower 4 bits.
b) If the upper nibble (4 bits) is greater than 9, or if C=1, add 6 to the upper 4 bits.
c) It is possible that 6 is added to both the nibbles.

Test and use debugger of the KEIL µVision3 IDE to verify your program for given values.

TEST VALUES:

1) $x = 47H,\ y = 38H,\ Result = 85H$
2) $x = 29H,\ y = 18H,\ Result = 47H$
3) $x = 52H,\ y = 91H,\ (Result = 143H \rightarrow register = 43H)$
4) $x = 94H,\ y = 91H,\ (Result = 185H \rightarrow register = 85H)$
5) $x = 54H,\ y = 87H,\ (Result = 141H \rightarrow register = 41H)$
6) $x = 98H,\ y = 99H,\ (Result = 197H \rightarrow register = 97H)$
7) $x = 93H,\ y = 67H,\ (Result = 160H \rightarrow register = 60H)$
8) $x = 23H,\ y = 35H,\ (Result = 58H \rightarrow register = 58H)$
9) $x = 63H,\ y = 36H,\ (Result = 99H \rightarrow register = 99H)$
10) $x = 63H,\ y = 39H,\ (Result = 102H \rightarrow register = 02H)$
11) $x = 39H,\ y = 63H,\ (Result = 102H \rightarrow register = 02H)$

Note: Use of DA instruction is not allowed.

POST LAB QUESTION:

Write an assembly language program that adds two packed BCD numbers and check that whether the resultant BCD number is even or odd.If the resultant BCD number is even, then set **P2.3**. The result must be a valid BCD number.

EXPERIMENT - 7

DEBOUNCING OF MECHANICAL SWITCHES: 4-BIT SIMPLE CALCULATOR

OBJECTIVES:

- To understand the phenomena of bouncing in mechanical switches and its solution
- To be able to use mechanical switches in making a simple application

EQUIPMENT/TOOLS:

- KeilµVision.3IDE
- ATMEL ®89C52 Microcontroller
- Microprocessor Trainer Board

BACKGROUND:

Push-button switches, toggle switches, and electro-mechanical relays all have one thing in common: contacts. It's the metal contacts that make and break the circuit and carry the current in switches and relays. Because they are metal, contacts have mass. And since at least one of the contacts is on a movable strip of metal, it has springiness. Since contacts are designed to open and close quickly, there is little resistance (damping) to their movement.

Because the moving contacts have mass and springiness with low damping they will be "bouncy" as they make and break. That is, when a normally open (N.O.) pair of contacts is closed, the contacts will come together and bounce off each other several times before finally coming to rest in a closed position. The effect is called "contact bounce" or, in a switch, "switch bounce" see Figure 7.1. Note that contacts can bounce on opening as well as on closing.

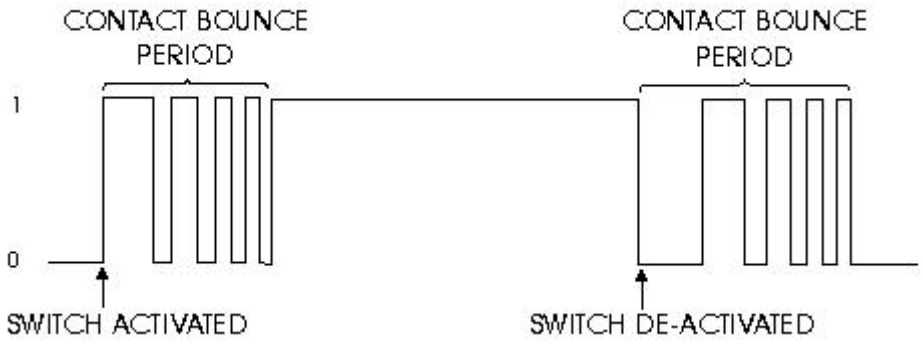


Figure 7.1 Bouncing of mechanical switches

If all you want your switch or relay to do is turn on a lamp or start a fan motor, then contact bounce is not a problem. But if you are using a switch or relay as input to a digital counter, a personal computer, or a micro-processor based piece of equipment, then you must consider contact bounce. The reason for concern is that the time it takes for contacts to stop bouncing is measured in milliseconds. Digital circuits can respond in microseconds.

As an example, suppose you want to count widgets as they go by on a conveyor belt. You could set up a sensitive switch and a digital counter so that as the widgets go by they activate

the switch and increment the counter. But what you might see is that the first widget produces a count of 47, the second widget causes a count of 113, and so forth. What's going on? The answer is you're not counting widgets; you're counting how many times the contacts bounced each time the switch is activated!

There are several ways to solve the problem of contact bounce (that is, to "de-bounce" the input signal). Often the easiest way is to simply get a piece of equipment that is designed to accept "bouncy" input. In the widget example above, you can buy special digital counters that are designed to accept switch input signals. They do the de-bouncing internally. If that is not an option, then you will have to do the debouncing yourself using either hardware or software.

DEBOUNCING USING HARDWARE:

A simple hardware debounce circuit for a momentary N.O. push-button switch is show in Figure 7.2. As you can see, it uses an RC time constant to swamp out the bounce. If you multiply the resistance value by the capacitance value you get the RC time constant. You pick R and C so that RC is longer than the expected bounce time. An RC value of about 0.1 seconds is typical. Note the use of a buffer after the switch to produce a sharp high-to-low transition. And remember that the time delay also means that you have to wait before you push the switch again. If you press it again too soon it will not generate another signal.

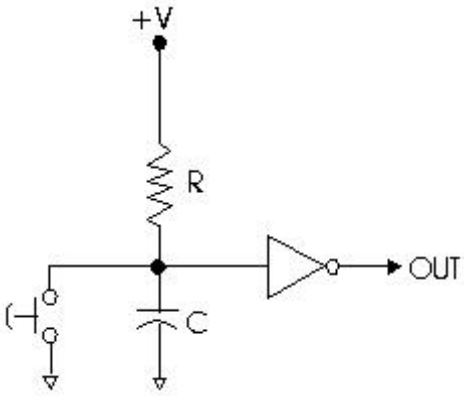


Figure 7.2 Hardware debounce circuit

DEBOUNCING USING SOFTWARE:

If you're the one developing the digital "box", then you can debounce in software. Usually, the switch or relay connected to the computer will generate an interrupt when the contacts are activated. The interrupt will cause a subroutine (interrupt service routine) to be called. A typical debounce routine is given below in a sort of generic assembly language.

The idea is that as soon as the switch is activated the Debounce Routine (DR) is called. The DR calls another subroutine called DELAY which just kills time long enough to allow the contacts to stop bouncing. At that point the DR checks to see if the contacts are still activated (maybe the user kept a finger on the switch). If so, the DR waits for the contacts to clear. If the contacts are clear, DR calls DELAY one more time to allow for bounce on contact-release before finishing.

A debounce routine must be tuned to your application; the one above may not work for everything. Also, the programmer should be aware that switches and relays can lose some of

their springiness as they age. That can cause the time it takes for contacts to stop bouncing to increase with time. So the debounce code that worked fine when the keyboard was new might not work a year or two later. Consult the switch manufacturer for data on worst-case bounce times

EXERCISE:

Design and implement a 4-bit simple, fixed point, calculator that reads data from SPDT's connected at P1 and sends output to LEDs connected at P2. First, simulate the code in Proteus and then check your algorithm on hardware as well.

PROCEDURE:

HARDWARE SETUP

- There are 8-SPDT switches on the trainer. Make initial state of all switches at 0V.
- Input is given through these SPDT switches. There are 4 data switches and 1 control, acting like an enter key, connected at P1.
- User prepares a 4-bit data over the binary switches, and then sets the control switch to logic 1 and then back to logic 0 (**0-1-0**).
- After this action controller reads the data at input port P1, performs desired operation and outputs the result to port P2.

SOFTWARE

Your program must be capable of doing the following jobs:

- Configure port P1 as input port
- Write a sub routine to debounce the switch connected at P1.4
- Read data at P1 when SPDT at P1.4 is made 1 from zero and then back to zero
- Repeat the previous step three times, twice for the operands and once for the operator
- Decode the operator, perform desired operation and show result at LEDs connected at P2
- Get ready to take input again

POST LAB QUESTION:

Write a program that generates one cycle of a square wave with frequency of **100mHz** on port pin **P3.3** with duty cycle depending on the 4-bit input value mentioned on port **P2** by the user. User prepares a 4-bit data over the binary switches, and then sets the control switch to logic 0 and then back to logic 1 (**1-0-1**). The user is allowed to mention the value from 0 – 10. For example, if input value is 1 then duty cycle is 10%, if the value is 2 duty cycle is 20% and so on. If value is 0 then **P3.3** is **low** and if value is greater than or equal to 10 then **P3.3** is **high**.

PROCEDURE:

EXPERIMENT - 8

KEYPAD INTERFACING

OBJECTIVES:

- To learn scanning of a simple keypad for input
- To learn a method of taking input from keypad and process it using the 89C52 micro-controller

EQUIPMENT/TOOLS:

- ATMEL 89C52 Microcontroller
- Microprocessor Trainer Board

BACKGROUND:

The keypad is a matrix of push buttons. Figure 8.1 shows the internal schematic of 4x3 matrix keypad. It can be used in various ways to take an input. Keypad can be scanned using two methods i.e., either ground each row at a time and read the columns or ground each column at a time and read rows, in order to scan the whole keypad for detecting the key pressed.

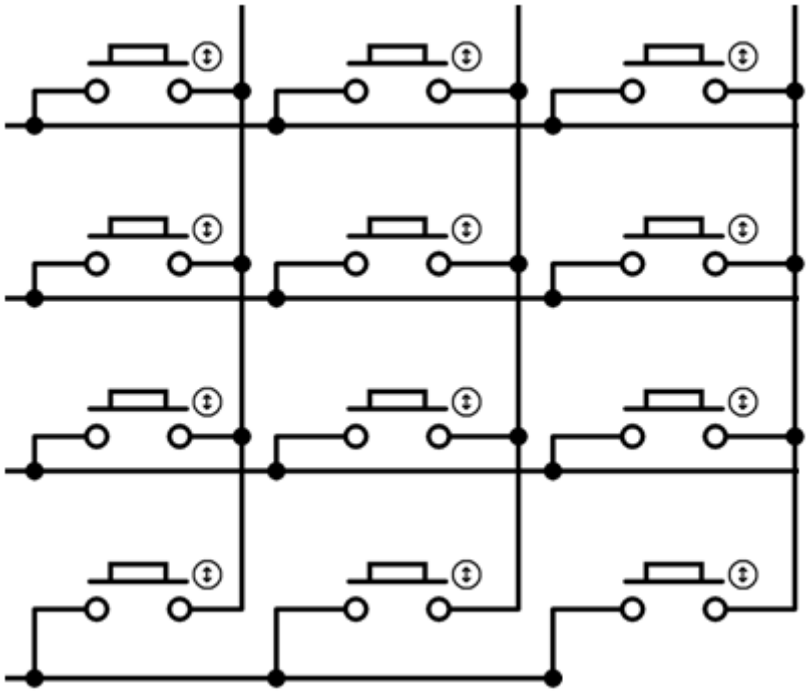


Figure 8.1 Internal schematic of 4x3 matrix keypad

Figure 8.2 shows the row column configuration of keypad available on trainer.

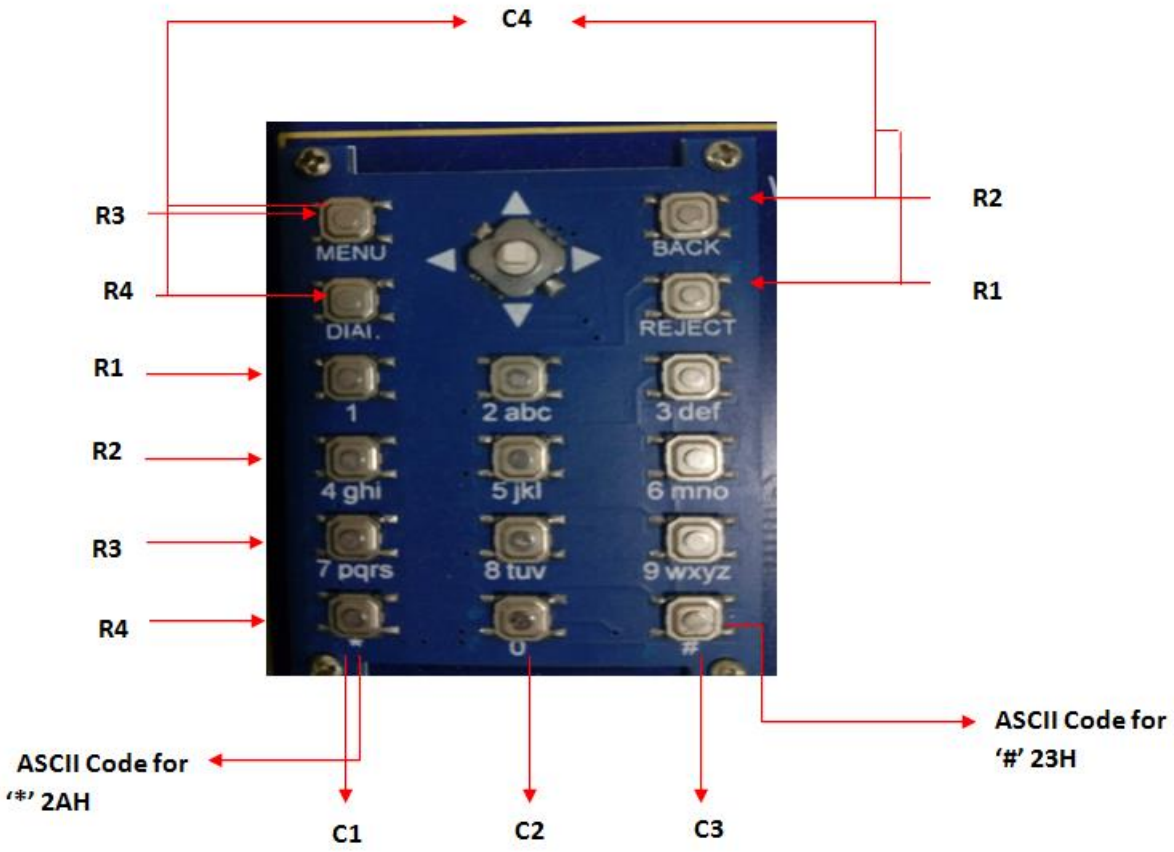


Figure 8.2 Row column configuration of keypad

EXERCISES:

1. Write a program that takes input of one digit from 4x3 keypad. Assign ten decimal value to '*' button and eleven decimal value to '#' button. After getting the input, display the square of the input number on LEDs.

2. Write a program that takes input of 2-digit number from 4x3 keypad and set **P1.3** pin if the input number is even otherwise clear **P1.3**pin. Assign ten decimal value to '*' button and eleven decimal value to '#' button. If '*' or '#' button is pressed then use their corresponding decimal value as 2-digit number.

POST LAB QUESTION:

Write a program that generates one cycle of a square wave with frequency of 100mHz on port pin P1.0 with duty cycle depending on the one digit input value pressed by the user on 4x3 keypad. The user is allowed to mention the value from 1 – 9. For example, if input value is 1 then duty cycle is 10%, if the value is 2 duty cycle is 20% and so on. Use crystal frequency **f=11.0592MHz.**

Note: Key presses on keypad are ignored during square wave generation.

EXPERIMENT- 9

4-DIGIT BCD TOSEVEN SEGMENT DISPLAY

OBJECTIVES:

- To learn a method of displaying a 4-digit number on an output device using the 89C52 micro-controller

EQUIPMENT/TOOLS:

- ATMEL 89C52 Microcontroller
- Microprocessor Trainer Board

A seven-segment display as shown in Figure 9.1 is an electronic display device for displaying decimal numerals. Seven-segment displays are widely used in digital clocks, electronic meters and other electronic devices that display numerical information. A BCD to Seven Segment decoder is a combinational logic circuit that accepts a decimal digit in BCD (input) and generates appropriate outputs for the segments to display the input decimal digit.
The figure below shows the seven segment display of the trainer

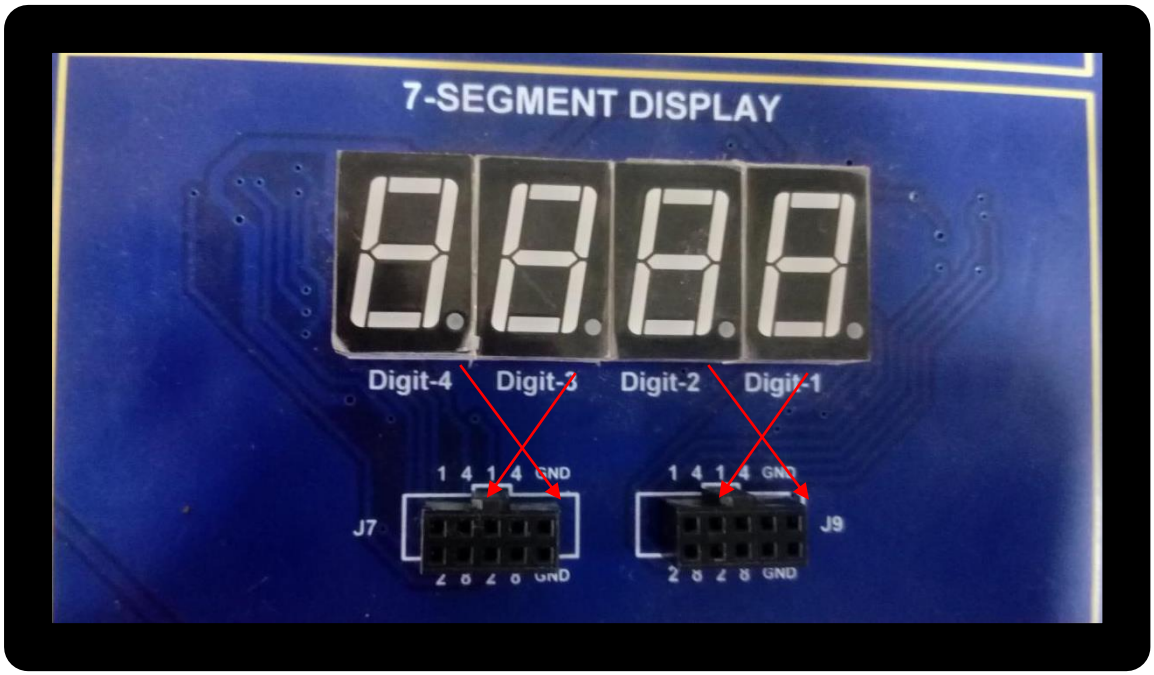


Figure 9.1 Seven segment display

EXERCISE:

Write a program that takes two 4-bit numbers on Port 3 as input (Lower nibble is A, upper nibble is B) and performs the required arithmetic operation according to the selection value in register R0. If R0 = 1, then perform A+B. If R0 = 2, then perform A*B. The result is to be shown on BCD to seven segment display in decimal base.
HINT: Use port 1 and 2

EXAMPLE 1:

Figure 9.2 below shows the output value 255 shown on 7-segment display after FF is given from the switches shown in Figure 9.3 to port 3

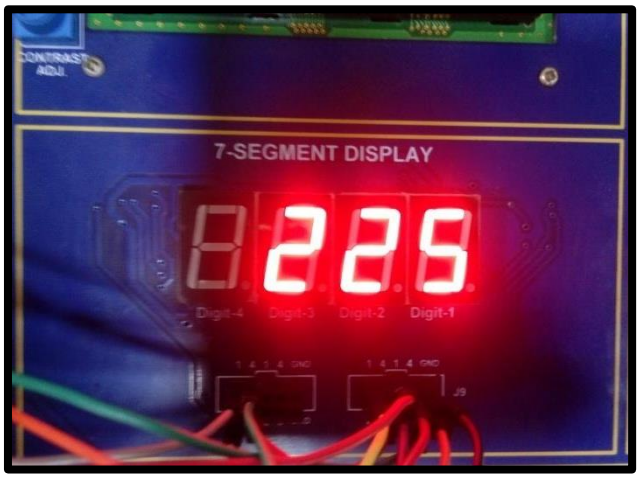


Figure 9.2 The output value 255 shown on 7-segment display

Figure 9.3 Toggle switches

POST LAB QUESTION:

We have a list of ten 4-digits numbers stored in the ROM in the form of comma-separated strings. Like "list db: “1234”, “6789”, ...,”3458”.
Write a program to display first complete number on 7-segment display for one second, then display the next number for one second, and so on.

EXPERIMENT-10

TIMERS and PULSE WIDTH MODULATION

OBJECTIVES:

- To learn to apply Pulse Width Modulation on a square wave.
- To learn to use PWM for speed control of a DC motor.

EQUIPMENT/TOOLS:

- KeilµVision.3IDE
- ATMEL 89S52 Microcontroller
- Microprocessor Trainer Board

BACKGROUND:

Pulse Width Modulation (also called Pulse Duration Modulation) is a method of regulating the output voltage of a switching power supply by varying the width, but not the height, of a train of pulses that drives a power switch .A good analogy is bicycle riding. You peddle (exert energy) and then coast (relax) using your momentum to carry you forward. As you slow down (due to wind resistance, friction, road shape) you peddle to speed up and then coast again. The 'duty cycle' is the ratio of peddling time to the total time (peddle + coast time). A 100% duty cycle means you are peddling all the time, and a 50% duty cycle means you are peddling only half the time.

PWM for motor speed control works in a very similar way. Instead of supplying a varying voltage to a motor, it is supplied with a fixed voltage value (such as 5V), which starts it spinning immediately. The voltage is then removed and the motor 'coasts'. By continuing this voltage on/off cycle with a varying duty cycle, the motor speed can be controlled.

HARDWARE SCHEMATICS

To setup the hardware, refer to following figure 10.1.

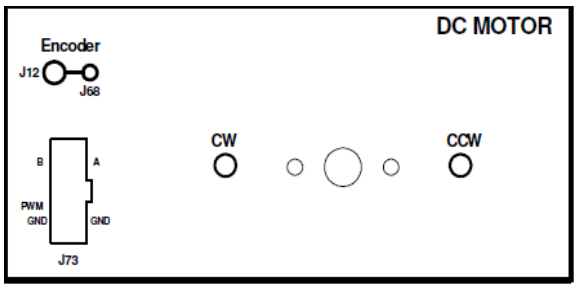


Figure 10.1 Hardware setup

To select the direction use pins “A” and “B” as shown in Table 10.1 on connector J73. Make sure “A” and “B” get complementary inputs and never go high simultaneously as it will damage the control circuitry.

STATUS OF “A”	STATUS OF “B”	DIRECTION OF MOTOR
1	0	Clock Wise
0	1	Counter Clock Wise

Table 10.1 Direction of motor

- PWN pin is used to control the speed of motor, which should be connected to Port’s 7^{th} pin.

- A should be connected to Port's 0^{th} pin.
- B should be connected to Port's 1^{st} pin.

EXERCISE:

Write a program to generate a 10 kHz square wave. Use the program to control the speed of a DC motor using PWM. Use timer for delay generation. The ON-OFF ratio of the wave must vary in 3 steps: Assume crystal frequency to be 11.0592MHz.

a. Fully ON
b. 80% speed
c. 60% speed

POST LAB QUESTIONS:

Answer the following questions:

1. What is meant by PWM?
2. What is the significance of value of frequency in PWM wave?
3. Can we use PWM in digital to analog conversion process? If yes how?
4. How can we make two soft PWM channels in 8051?

EXPERIMENT- 11

SIMULTANEOUS USE OF EXTERNAL INTERRUPT, SERIAL COMMUNICATION AND LCD

OBJECTIVES:

- To understand use of timers to generate a specific amount of delay
- To understand UART and to establish RS-232 based communication between two micro-controllers
- To learn to use the UART in 8051 based micro-controllers by manipulating Timers, and the TCON and SCON registers

EQUIPMENT/ TOOLS:

- ATMEL ®89C52 Microcontrollers
- Microprocessor Trainer Board
- 2-Lines × 16-Characters LCD

BACKGROUND:

Serial port is a three-wire full duplex communication channel. It is asynchronous communication protocol so the two entities taking part in communication must be programmed for the same bit rate, parity, start and stop bits. Students should read the theory of serial communication from the reference book. For our purposes we are going to do the following:

- Program TIMER-1 as a baud rate generator (8-bit auto reload mode) and program TMOD accordingly.
- Program SCON for particular mode of operation of serial port.
- Start communication.

The serial port has four modes of operation. At this moment, we are interested in mode-3(variable baud rate 8 bit transmission). The structure of SCON is as follows in Figure 11.1 below:

7	6	5	4	3	2	1	0
SM0	SM1	SM2	REN	TB8	RB8	T1	R1
0	1	0	1	0	0	1	0

SM0	SM1	Mode
0	0	Shift Register Mode Fixed Frequencey (OSC/12)
0	1	8-bit UART variable baud rate set by timer
1	0	9-bit UART fixed OSC/12 or OSC/64
1	1	9-bit UART variable set by timer

Figure 11.1 Structure of SCON

- SM0-SM2: Serial port mode control bits
- SM2 is used for multiprocessor communication we will set it to 0.
- As we are using 9-bit variable baud rate SCON = 0x52
- Select 8-bit auto reload mode for TIMER-1 i.e. TMOD = 0x20
- REN is receiver enable and should be set to 1 for receiver to function.

EXERCISES:

1. Write a program to receive in 8051 an ASCII character transmitted from PC and then transmit back to the PC the next ASCII character. This is so called the modified ECHO and is used to check the link between two communicating. Assume oscillator frequency to be **11.0592MHz**.

2. You are required to write a program to communicate two 89C52 microcontrollers serially at a **9600 baud rate**. Store five strings in microcontroller-1. Connect five SPDT switches with port P1 and a push button with P3.2 of microcontroller-1. Each SPDT switch represents a string stored in ROM. Whenever push button is pressed and then released, corresponding string is transmitted to the 2nd micro-controller. The microcontroller-2 displays the received string on 2x16 LCD. The push button introduces bouncing for **25msec**. Assume oscillator frequency to be **11.0592MHz**.

   Note: While displaying the string on LCD, microcontroller-2 does not receive anything. So after displaying the string go back to receive the next one.

POST LAB QUESTION:

Write an assembly language program to measure the time period of a rectangular wave coming at pin P3.2. The ON time is measured in milliseconds and stored in register R0. Oscillator frequency is 12 MHz
Hint: Along with external interrupt, use a timer that rolls over after every millisecond during the ON time.

EXPERIMENT-12

REAL TIME CLOCK

OBJECTIVE:

- To consolidate the knowledge of previous experiments to understand and program a complex problem.

EQUIPMENT/TOOLS:

- ATMEL ®89C52 Microcontroller
- Microprocessor Trainer Board
- PC

BACKGROUND:

In this experiment, you are required to use your knowledge of the timer and serial port (you can use the programs that you have developed in the previous experiments) to program a clock which will display time in the HyperTerminal program on the PC.

PROCEDURE:

The steps are as follows:

- Write an interrupt service routine for Timer 0 to run a clock routine that increments seconds, minutes and hours.
- Write an interrupt service routine for the serial port to communicate with the HyperTerminal
- After every second, print HH:MM:SS on the hyper terminal using the *printf()* routine of *stdio* library.

EXERCISE:

Program a clock which displays time in the Hyper Terminal. Your clock must haveHours, Minutes and Seconds (HH:MM:SS).

POST LAB QUESTIONS:

Answer the following questions:

1. What is the main advantage of using Timers to generate delays instead of using DJNZ nested loops?
2. When and why do we need to clear timer flag TFx?
3. What is meant by auto reload of a timer?
4. How can we use two timers to generate two square waves of frequency 2KHz and 3KHz with 50% duty cycle?

Appendix-A: Lab Evaluation Criteria

Labs with projects

1. Experiments and their report 50%
    a. Experiment 60%
    b. Lab report 40%
2. Quizzes (3-4) 15%
3. Final evaluation 35%
    a. ProjectImplementation 60%
    b. Project report and quiz 40%

Labs without projects

1. Experiments and their report 50%
    a. Experiment 60%
    b. Lab report 40%
2. Quizzes (3-4) 20%
3. Final Evaluation 30%
    i. Experiment 60%
    ii. Lab report, pre and post experiment quiz 40%

Notice:

Copying and plagiarism of lab reports is a serious academic misconduct. First instance of copying may entail ZERO in that experiment. Second instance of copying may be reported to DC. This may result in awarding FAIL in the lab course.

Appendix-B: Safety around Electricity

In all the Electrical Engineering (EE) labs, with an aim to prevent any unforeseen accidents during conduct of lab experiments, following preventive measures and safe practices shall be adopted:

- Remember that the voltage of the electricity and the available electrical current in EE labs has enough power to cause death/injury by electrocution. It is around 50V/10 mA that the "cannot let go" level is reached. "The key to survival is to decrease our exposure to energized circuits."
- If a person touches an energized bare wire or faulty equipment while grounded, electricity will instantly pass through the body to the ground, causing a harmful, potentially fatal, shock.
- Each circuit must be protected by a fuse or circuit breaker that will blow or "trip" when its safe carrying capacity is surpassed. If a fuse blows or circuit breaker trips repeatedly while in normal use (not overloaded), check for shorts and other faults in the line or devices. Do not resume use until the trouble is fixed.
- It is hazardous to overload electrical circuits by using extension cords and multi-plug outlets. Use extension cords only when necessary and make sure they are heavy enough for the job. Avoid creating an "octopus" by inserting several plugs into a multi-plug outlet connected to a single wall outlet. Extension cords should ONLY be used on a temporary basis in situations where fixed wiring is not feasible.
- Dimmed lights, reduced output from heaters and poor monitor pictures are all symptoms of an overloaded circuit. Keep the total load at any one time safely below maximum capacity.
- If wires are exposed, they may cause a shock to a person who comes into contact with them. Cords should not be hung on nails, run over or wrapped around objects, knotted or twisted. This may break the wire or insulation. Short circuits are usually caused by bare wires touching due to breakdown of insulation. Electrical tape or any other kind of tape is not adequate for insulation!
- Electrical cords should be examined visually before use for external defects such as: Fraying (worn out) and exposed wiring, loose parts, deformed or missing parts, damage to outer jacket or insulation, evidence of internal damage such as pinched or crushed outer jacket. If any defects are found the electric cords should be removed from service immediately.
- Pull the plug not the cord. Pulling the cord could break a wire, causing a short circuit.
- Plug your heavy current consuming or any other large appliances into an outlet that is not shared with other appliances. Do not tamper with fuses as this is a potential fire hazard. Do not overload circuits as this may cause the wires to heat and ignite insulation or other combustibles.
- Keep lab equipment properly cleaned and maintained.
- Ensure lamps are free from contact with flammable material. Always use lights bulbs with the recommended wattage for your lamp and equipment.
- Be aware of the odor of burning plastic or wire.
- ALWAYS follow the manufacturer recommendations when using or installing new lab equipment. Wiring installations should always be made by a licensed electrician or other qualified person. All electrical lab equipment should have the label of a testing laboratory.

- Be aware of missing ground prong and outlet cover, pinched wires, damaged casings on electrical outlets.
- Inform Lab engineer / Lab assistant of any failure of safety preventive measures and safe practices as soon you notice it. Be alert and proceed with caution at all times in the laboratory.
- Conduct yourself in a responsible manner at all times in the EE Labs.
- Follow all written and verbal instructions carefully. If you do not understand a direction or part of a procedure, ASK YOUR LAB ENGINEER / LAB ASSISTANT BEFORE PROCEEDING WITH THE ACTIVITY.
- Never work alone in the laboratory. No student may work in EE Labs without the presence of the Lab engineer / Lab assistant.
- Perform only those experiments authorized by your teacher. Carefully follow all instructions, both written and oral. Unauthorized experiments are not allowed.
- Be prepared for your work in the EE Labs. Read all procedures thoroughly before entering the laboratory. Never fool around in the laboratory. Horseplay, practical jokes, and pranks are dangerous and prohibited.
- Always work in a well-ventilated area.
- Observe good housekeeping practices. Work areas should be kept clean and tidy at all times.
- Experiments must be personally monitored at all times. Do not wander around the room, distract other students, startle other students or interfere with the laboratory experiments of others.
- Dress properly during a laboratory activity. Long hair, dangling jewelry, and loose or baggy clothing are a hazard in the laboratory. Long hair must be tied back, and dangling jewelry and baggy clothing must be secured. Shoes must completely cover the foot.
- Know the locations and operating procedures of all safety equipment including fire extinguisher. Know what to do if there is a fire during a lab period; “Turn off equipment, if possible and exit EE lab immediately.”

Appendix-C: Guidelines on Preparing Lab Reports

Each student will maintain a lab notebook for each lab course. He will write a report for each experiment he performs in his notebook. A format has been developed for writing these lab reports.

Lab Report Format

1. **Introduction**: Introduce the new constructs/ commands being used, and their significance.
2. **Objective**: What are the learning goals of the experiment?
3. **Design**: How do the new constructs facilitate achievement of the objectives; if possible, a comparison in terms of efficacy and computational tractability with the alternate constructs? Include the circuit diagram and code with explanation.
4. **Issues**: The bugs encountered and the way they were removed.
5. **Conclusions**: What conclusions can be drawn from experiment?
6. **Application**: Suggest a real world application where this exercise may apply.
7. Answers to post lab questions (if any).

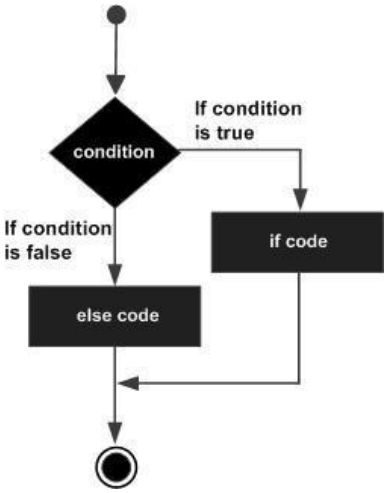
Sample Lab Report for Programming Labs

Introduction

The ability to control the flow of the program, letting it make decisions on what code to execute, is important to the programmer. The if-else statement allows the programmer to control if a program enters a section of code or not based on whether a given condition is true or false. If-else statements control *conditional branching*.

```
if ( expression )
  statement1
else
  statement2
```

If the value of *expression* is nonzero, *statement1* is executed. If the optional **else** is present, *statement2* is executed if the value of *expression* is zero. In this lab, we use this construct to select an action based upon the user's input, or a predefined parameter.



Objective:

To use if-else statements for facilitation of programming objectives: A palindrome is a number or a text phrase that reads the same backward as forward. For example, each of the following five-digit integers is a palindrome: 12321, 55555, 45554 and 11611. We have written a C++ program that reads in a five-digit integer and determines whether it is a palindrome.

Design:

A. Hardware Circuit Diagram:

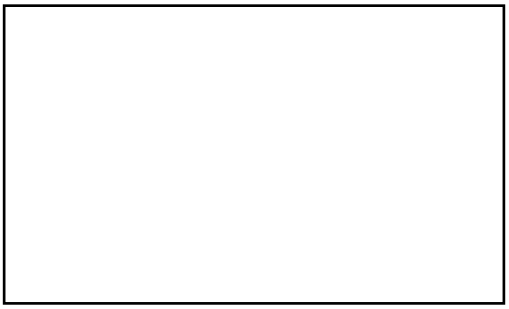


Fig.1. Circuit Diagram

B. Code with explanation:

The objective was achieved with the following code:

```
#include <iostream>

Using namespace std;
Int main()
{
    int i, temp, d, revrs=0;

    cout<<"enter the number to check :";
        cin>>i;
    temp=i;
    while(temp>0)
    {
```

```
        d=temp%10;
        temp/=10;
        revrs=revrs*10+d;

        }
    if(revrs==i)
    cout<<i<<" is palindorme";
    else
    cout<<i<<" is not palindrome";

    }
}
```

Screen shots of the output for various inputs are shown in Figure 2:

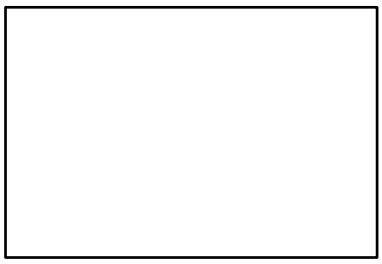


Fig.2. Screen shot of the output

The conditional statement made this implementation possible; without conditional branching, it is not possible to achieve this objective.

Issues:

Encountered bugs and issues; how were they identified and resolved.

Conclusions:

The output indicates correct execution of the code.

Applications:

If-else statements are a basic construct for programming to handle decisions.

Answers of Post Lab Questions