DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LABORATORY MANUAL

Course: Microcontroller Laboratory

Course Code: 4CSL02

Faculty: Smt. Kavitha M.,

Assistant Professor

SIDDAGANGA INSTITUTE OF TECHNOLOGY, TUMKUR-3

An Autonomous Institution, Affiliated to VTU, Belagavi & Recognized by AICTE and Accredited by NBA, New Delhi

MICROCONTROLLER LABORATORY

Lab Hours/ Week	**: 3 Hours**	**CIE Marks**	**: 50**
Sub. Code	**: 4CSL02**	**SEE Marks**	**: 50**
Credits	**: 1.5**	**SEE Duration**	**: 3 Hours**

Course Objectives:

1. Exhibit the knowledge of architecture and pin outs of 8051 microcontroller.
2. Analyze and Apply 8051 instruction set to develop assembly language programs for illustrating different types of operations.
3. Develop the ability to program 8051 microcontroller using embedded C.
4. Exhibit skills of developing embedded C code for I/O port programming and Timer/counter programming.
5. Interface 8051 to Logic controller, LCD, Seven segment display, Keyboard, DAC, Stepper motor and Elevator.

Course Outcomes (COs):

CO1: A**pply** different 8051 microcontroller instructions to **develop** assembly language code for illustrating data transfer and arithmetic operations using Keil tool.

CO2: Design and **develop** assembly language program for 8051 using different branch control instructions in Keil.

CO3: **Apply** Embedded C concepts to **develop** code for 8051 microcontroller using modern tool like Keil.

CO4: Design and **develop** code for interfacing different modules like Logic controller, Seven segment display, Keypad, DAC, Elevator with 8051 using embedded C with Keil and Flash Magic tools.

Vision of the Department

To work towards the vision of the institution by building a strong teaching and research environment that is capable of responding to the challenges of the 21st century.

Mission of the Department

To prepare under graduate, graduate and research students for productive careers in industry, academia and entrepreneurship through comprehensive educational programs, research in collaboration with industry & government, incubating innovative ideas, dissemination by scholarly publications and professional society /co-curricular activities.

Program Outcomes (POs):

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solution in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write

effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs):

1. **Computer based systems development:** Ability to apply the basic knowledge of database systems, computing, operating system, digital circuits, microcontroller, computer organization and architecture in the design of computer based systems.
2. **Software development:** Ability to specify, design and develop projects, application softwares and system softwares by using the knowledge of data structures, analysis and design of algorithm, programming languages, software engineering practices and open source tools.
3. **Computer communications and Internet applications:** Ability to design and develop network protocols and internet applications by incorporating the knowledge of computer networks, communication protocol engineering, cryptography and network security, distributed and cloud computing, data mining, big data analytics, ad hoc networks, storage area networks and wireless sensor networks.

SYLLABUS

PART – A

1. Write an ALP to exchange the block of data of length 'N' stored starting at RAM address 9000H and 9100H.

2. Write an ALP to add 'N' BCD numbers stored starting at RAM address 2000H. Store the result in the next consecutive locations.

3. Write an ALP to count the number of odd and even numbers in a block of 'N' numbers stored starting at RAM address 1000H. Store the result in the next consecutive locations.

4. Write an ALP to add two multi-byte numbers stored at RAM address 9000H and 9100H. Store the multi-byte result at RAM address 9200H.

5. Write an ALP to search for the key element in a block of 'N' bytes. If the number is present, show its position in the RAM location, 1050H. Otherwise, show FFH in 1050H location. Assume the key element is stored at RAM address 1000H and the data block starts at the RAM location 1001H.

6. Write an ALP to convert the binary number stored at RAM location 1000H into BCD and store the result in the next consecutive locations.

7. Write an ALP to compute the GCD and LCM of two 8-bit numbers stored at RAM locations 1000H and 1001H and store the result in the next consecutive locations.

8. Write an ALP to simulate BCD up counter.

9. Write an ALP to arrange the 'N' 8-bit numbers stored starting at RAM address 2000H in ascending order using Bubble Sort technique.

10. Write an ALP to multiply an 8-bit number stored at RAM location 1000H with a 16-bit number stored at RAM locations 1001H and 1002H. Store the result in the next consecutive locations.

PART – B

1. Write an 8051 C program to design a counter for counting the pulses of an input signal fed through pin P3.4. Display each count on the logic controller interface. (Use Counter 0 in mode2. The Count must be specified by the examiner).

2. Write an 8051 C program to read the status of 8 input bits from the Logic Controller Interface and display 'FF' if it is even parity bits otherwise display 00. Also display number of 1's in the input data.

3. Write an 8051 C program to compute x * y using Logic Controller Interface.

4. Write an 8051 C program to display the messages LIFE and HOPE alternately on a 4-digit seven-segment display Interface.

5. Write an 8051 C program to display the message "dEPt OF CSE" from right-to-left and left-to-right on a 4-digit seven-segment display Interface.

6. Write an 8051 C program to drive a Stepper motor Interface to rotate the motor by N steps in clockwise direction and N steps in anti-clockwise direction. Introduce suitable delay between successive steps.

7. Write an 8051 C program to display the strings on a 2x16 character LCD Interface.

8. Write an 8051 C program to scan a 4 x 4 keypad for key closure and display the code of the key pressed on LCD.

9. Write an 8051 C program to generate Half Rectified Sine wave, Fully Rectified Sine and sine waveform using the DAC Interface. (The output of the DAC is to be displayed on the CRO).

10. Write an 8051 C program to drive an elevator interface in the following way: Initially the elevator should be in the ground floor, with all requests in OFF state. When a request is made from a floor to any other floor, the elevator should move up or move down to that requested floor, service the request and stay in that floor waiting for any new request.

USING KEIL µVISION3 IDE

The Keil µVision IDE is a Windows-based software development platform that combines Project Management, Source Code Editing, Program Debugging, and Flash Programming in a single, powerful environment.

When you use the Keil µVision, the project development cycle is roughly the same as it is for any other software development project.

1. Create a project, select the target chip from the device database, and configure the tool settings.
2. Create source files in C or assembly.
3. Build your application with the project manager.
4. Correct errors in source files.
5. Test the linked application.

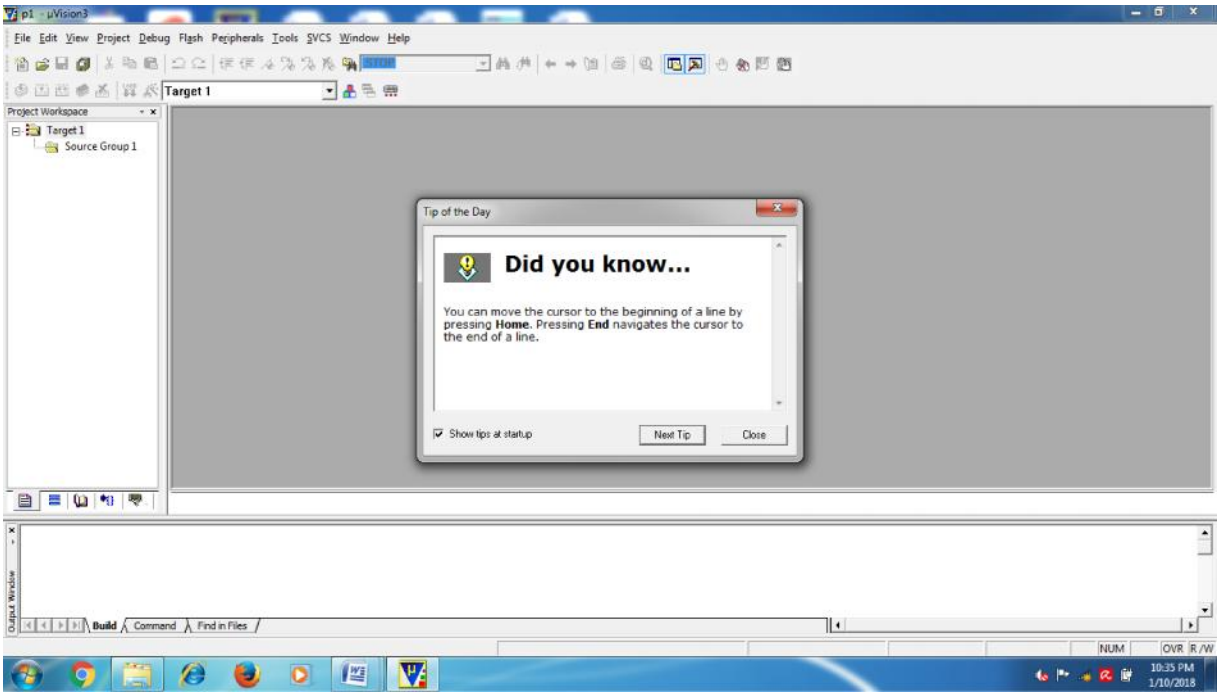
µVision3 has two operating modes:

- **Build Mode:** Allows you to translate all the application files and to generate executable programs.
- **Debug Mode:** Provides you with a powerful debugger for testing your application.

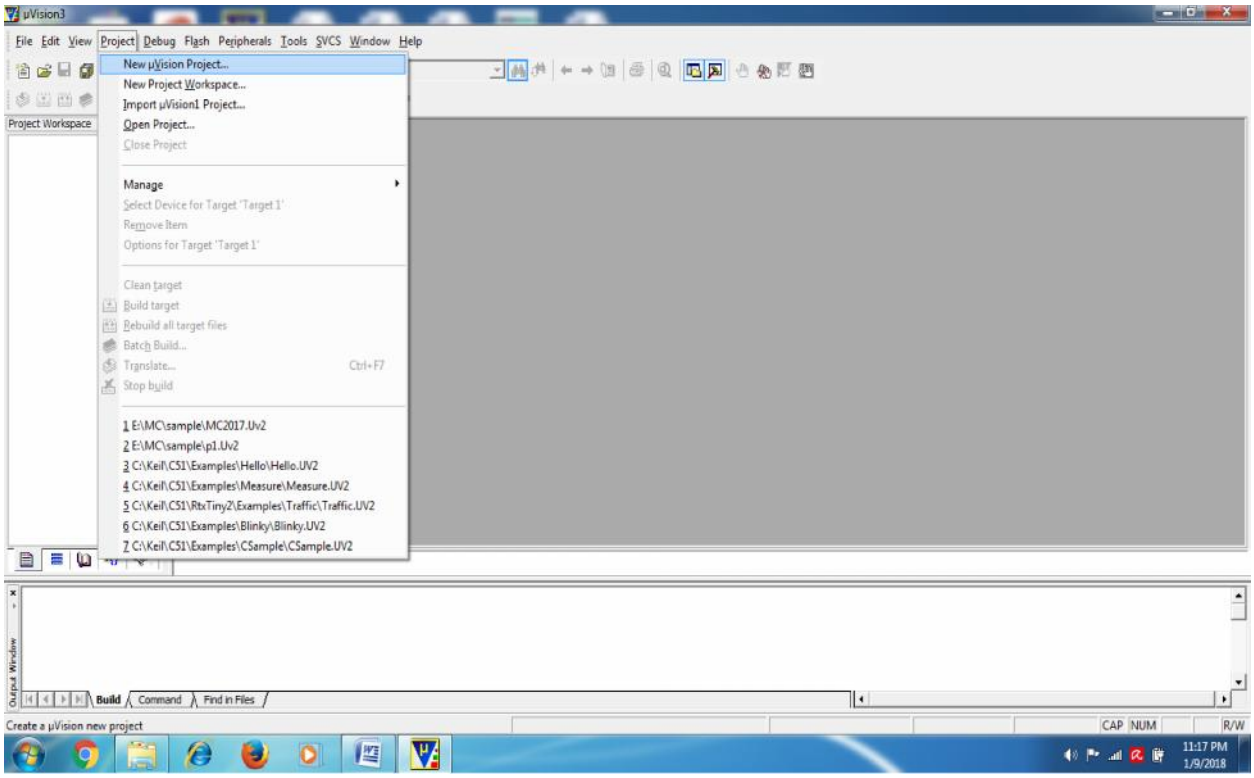
In both operating modes you may use the source editor of µVision3 to modify your source code. The Debug mode adds additional windows and stores an own screen layout.

To launch Keil µVision3 IDE goto Start menu and click on Keil µVision3.

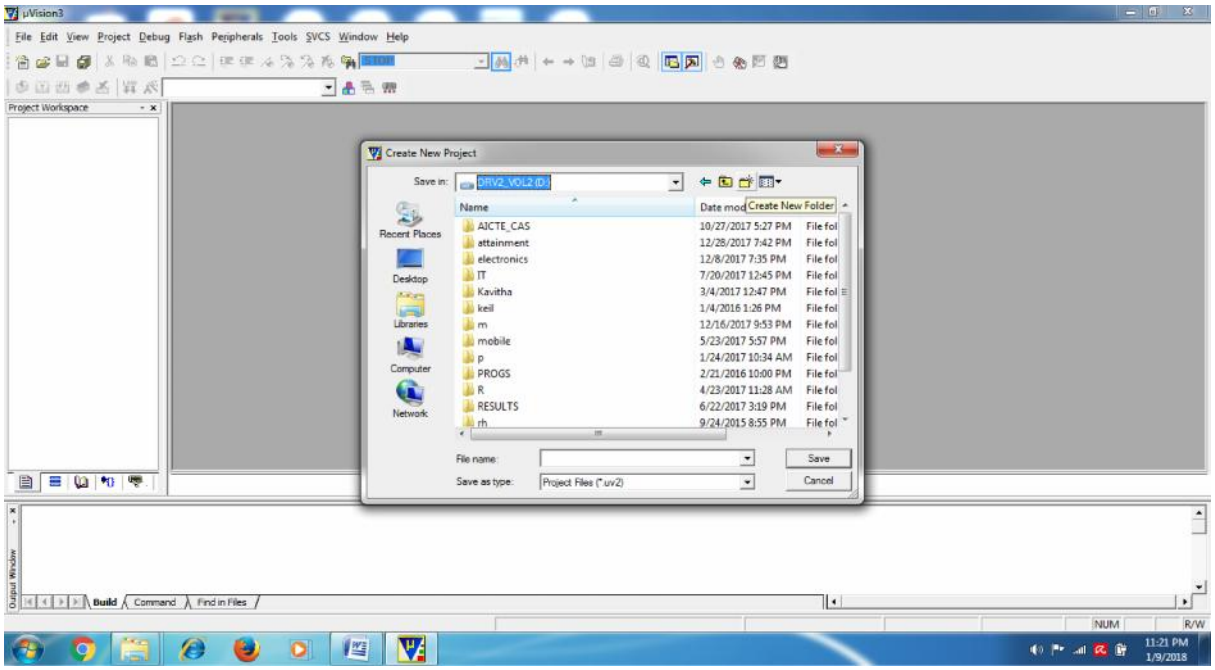
You will get a window as shown below.



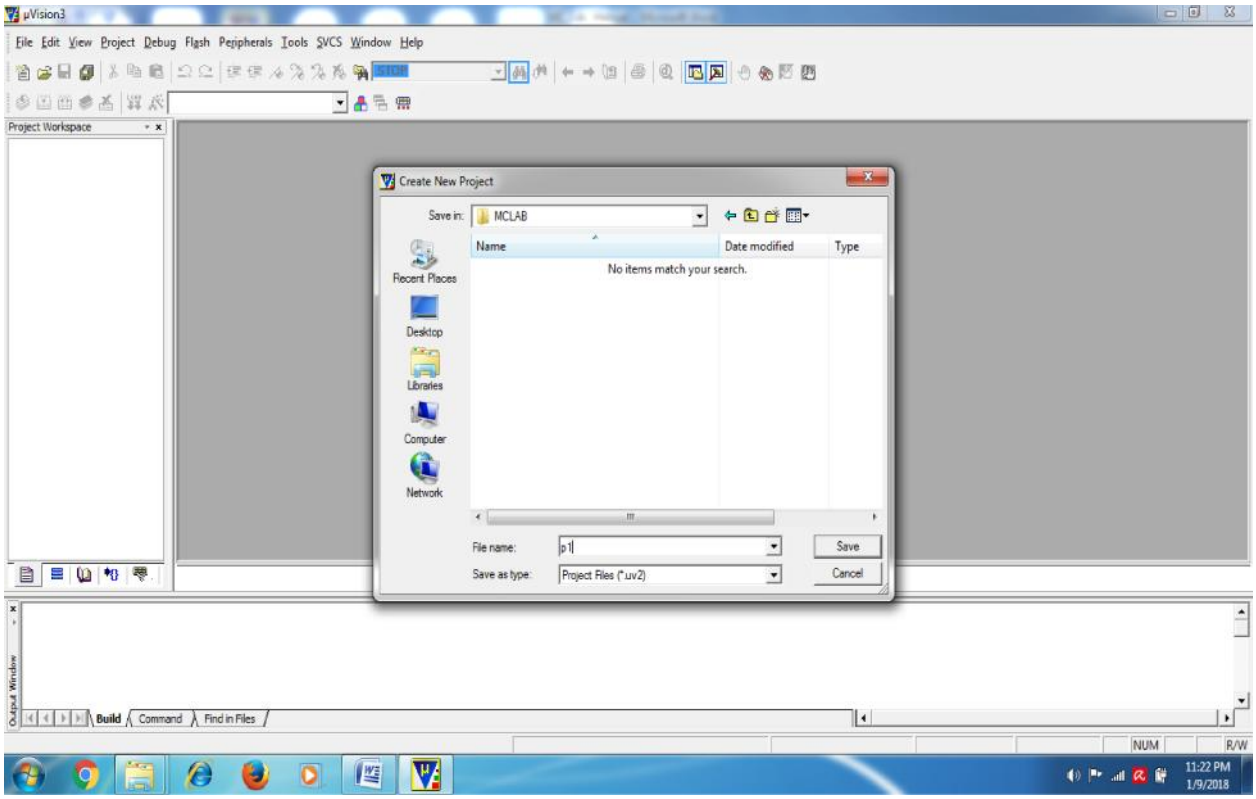
Next, Click on **Project** menu and select **New µVision Project** option as shown in below figure.



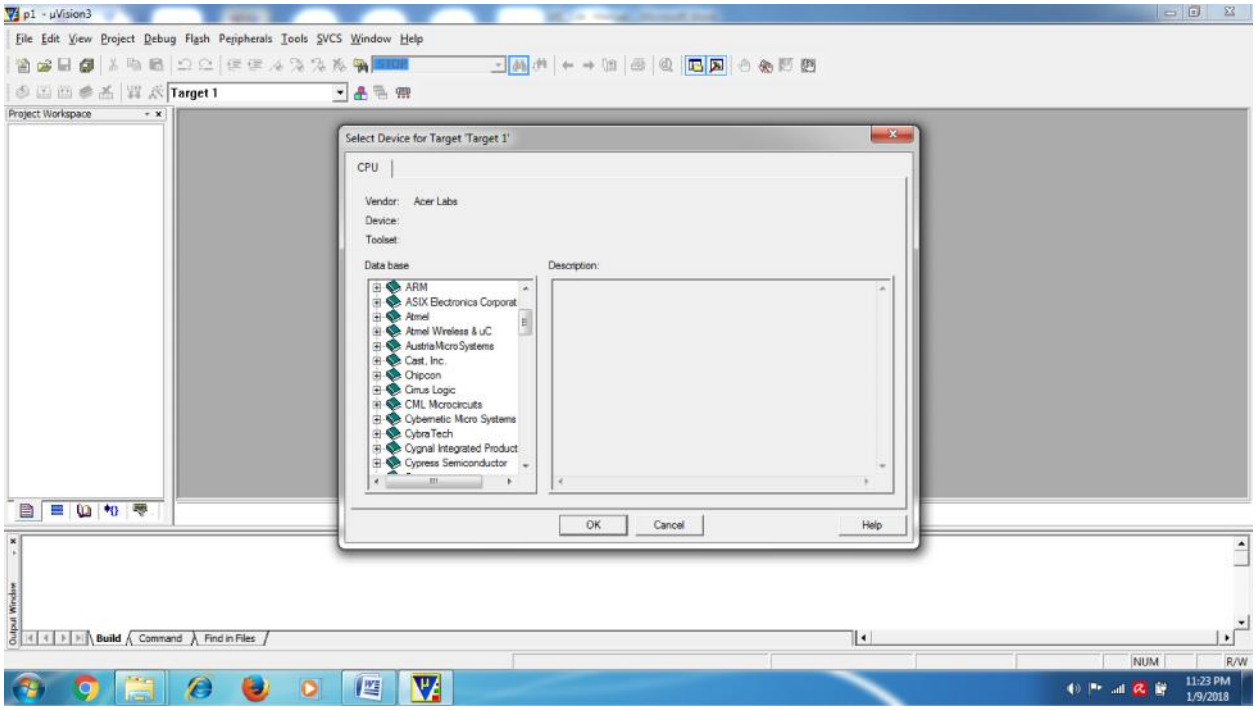
Now, create a new folder and go to that folder.



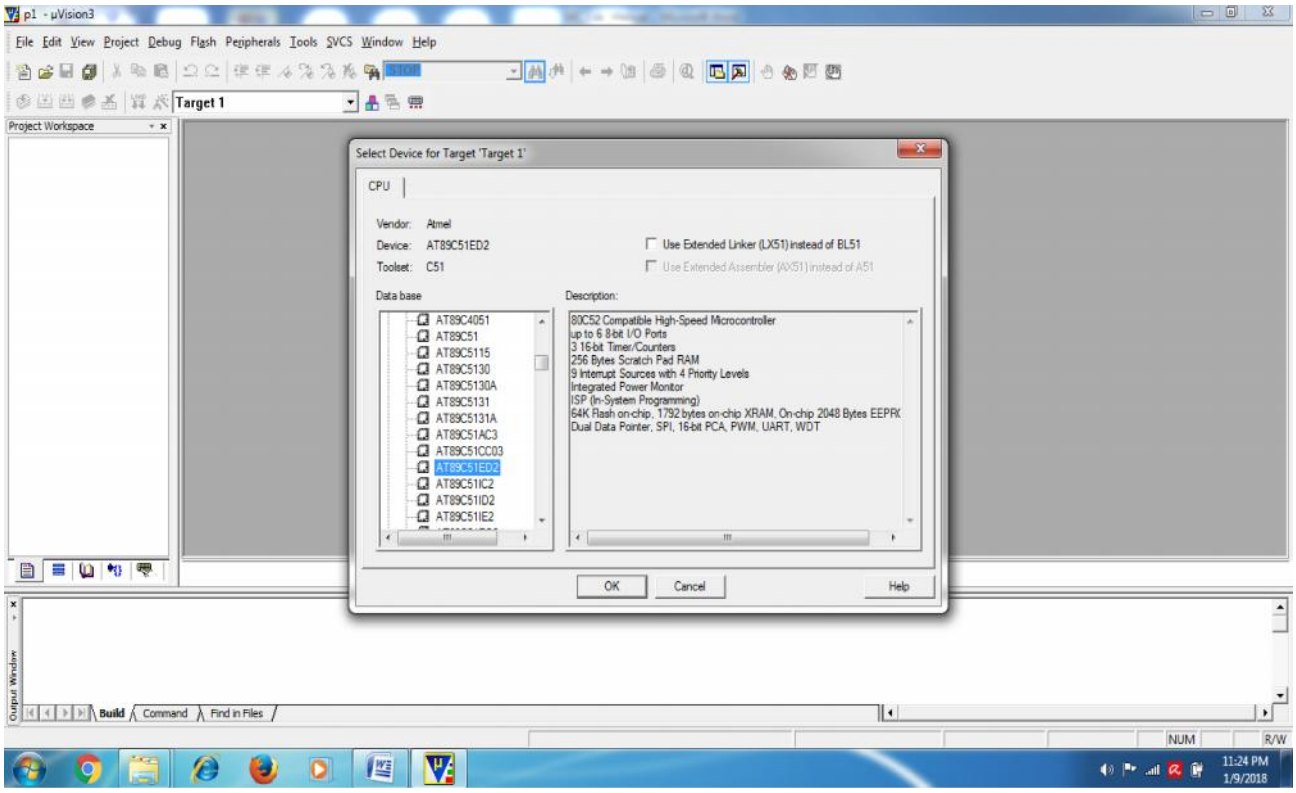
Give some valid Project file name and click on **Save** as shown in below figure.



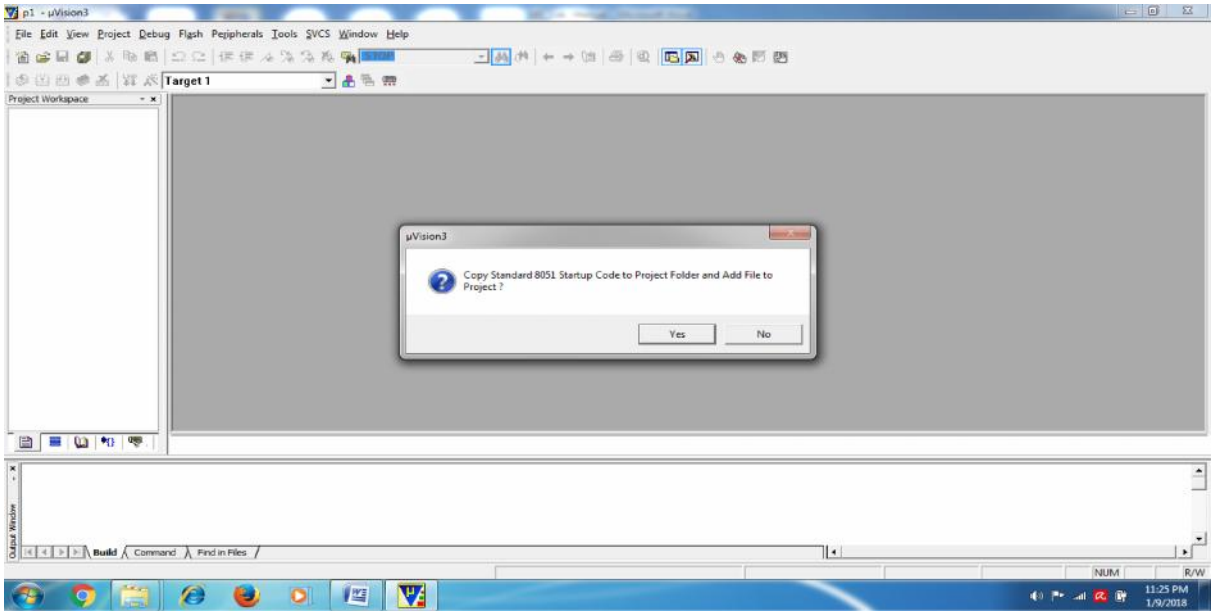
Now select the target chip as **Atmel** from the device data base.



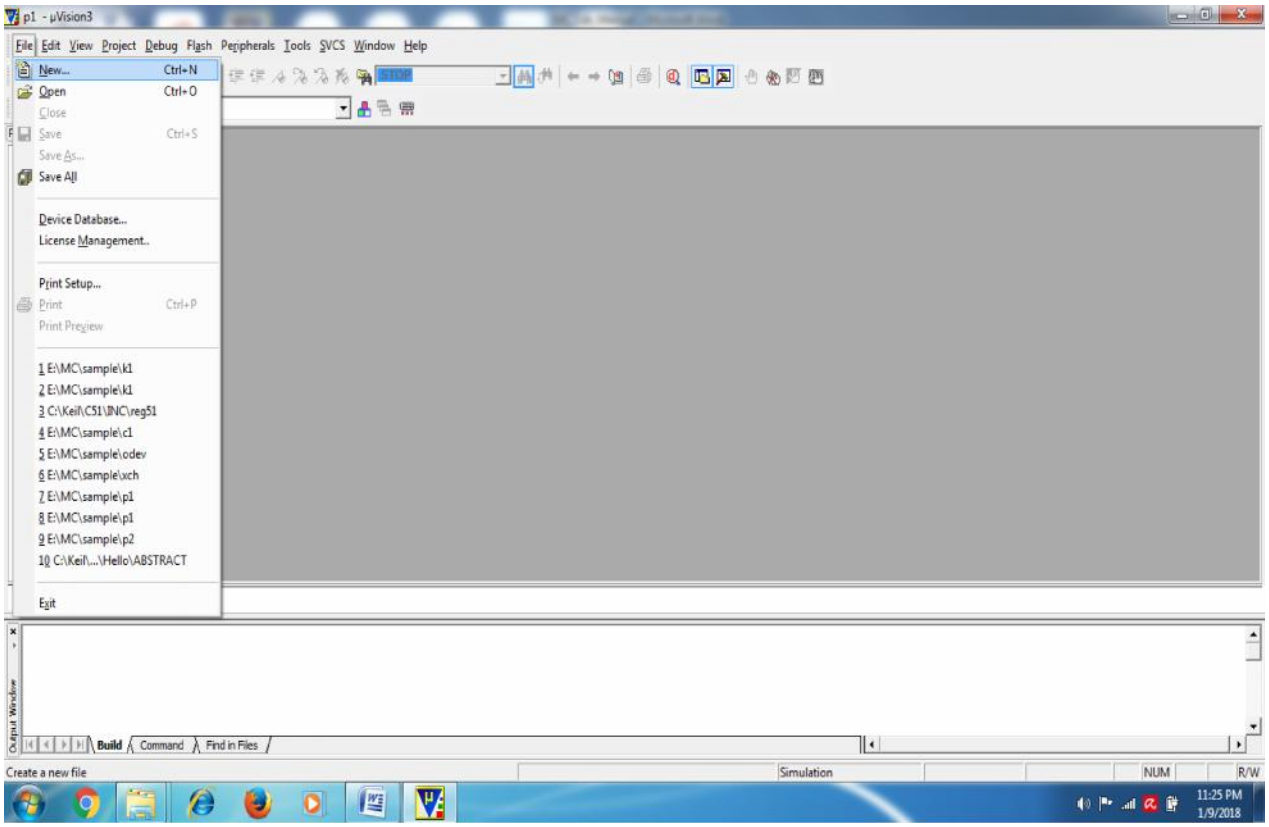
In Atmel, choose **AT89C51ED2** device and click on **OK** as shown in below figure.



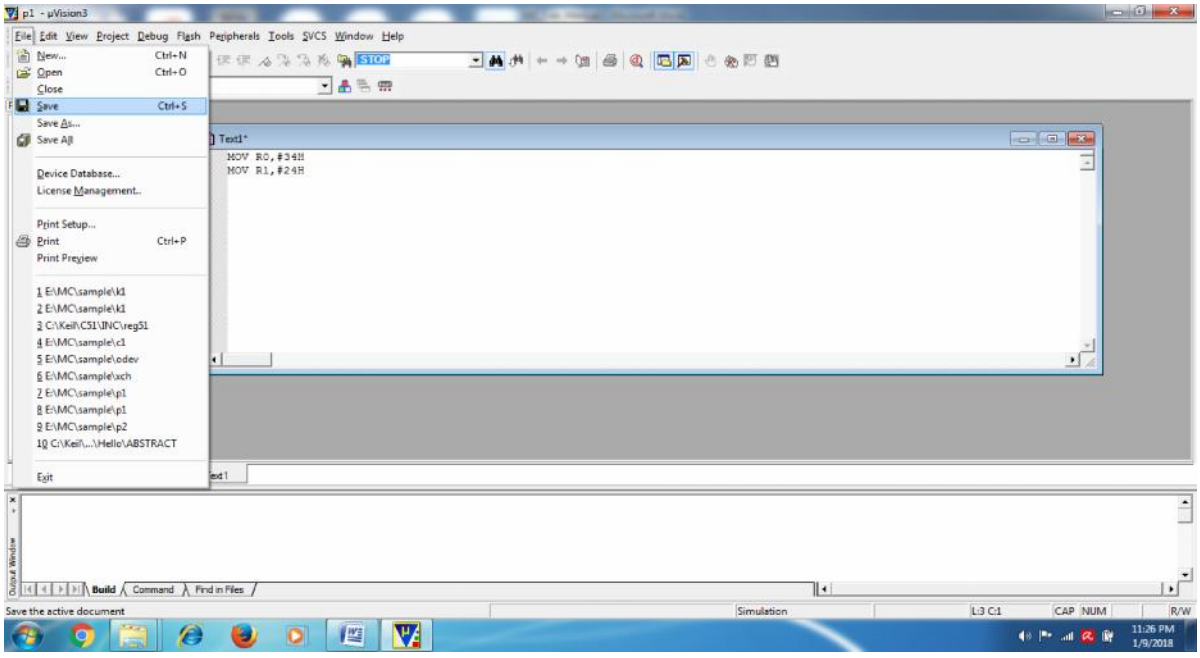
You need not add the Standard 8051 Startup Code to Project Folder. So, click on **No** as shown below.



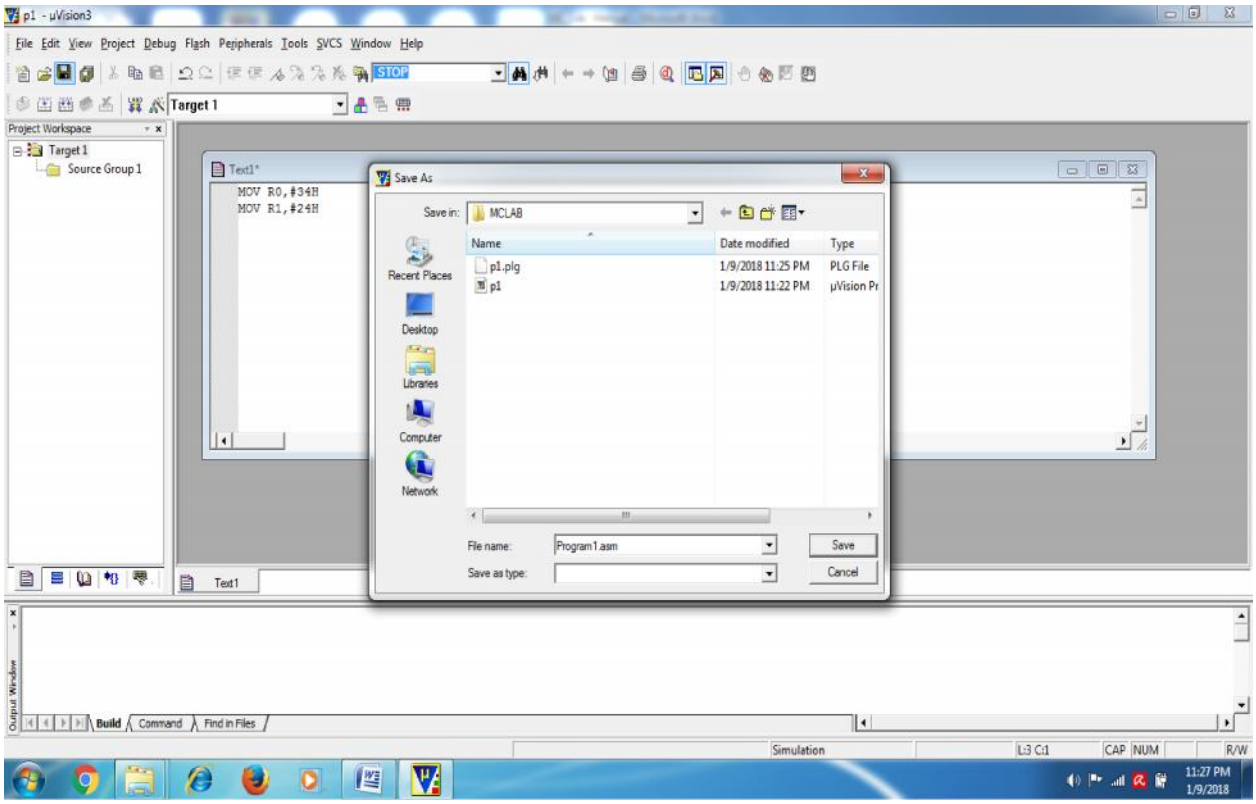
Now, go to **File** menu and select **New** option to create a new file.



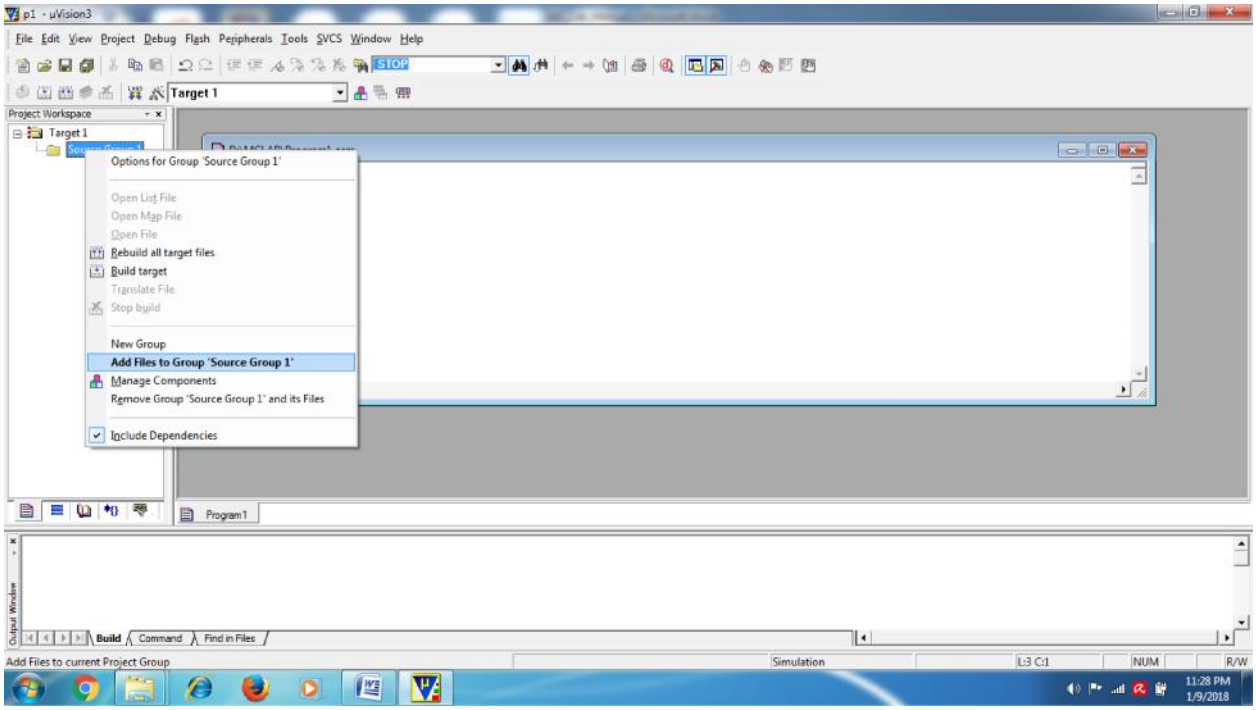
Now, edit your assembly language program and then click on **Save** as shown below.



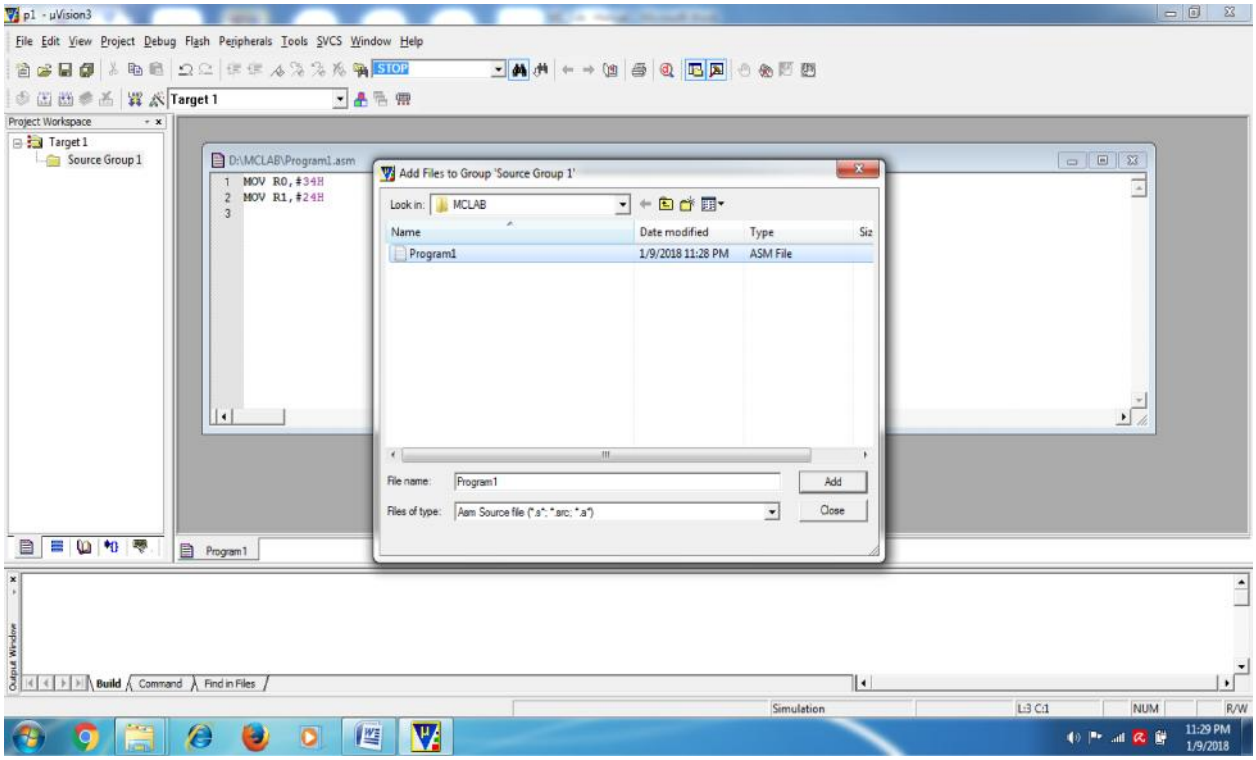
Now, save your file with .asm extension as shown in below figure and see that the file is saved in your folder.



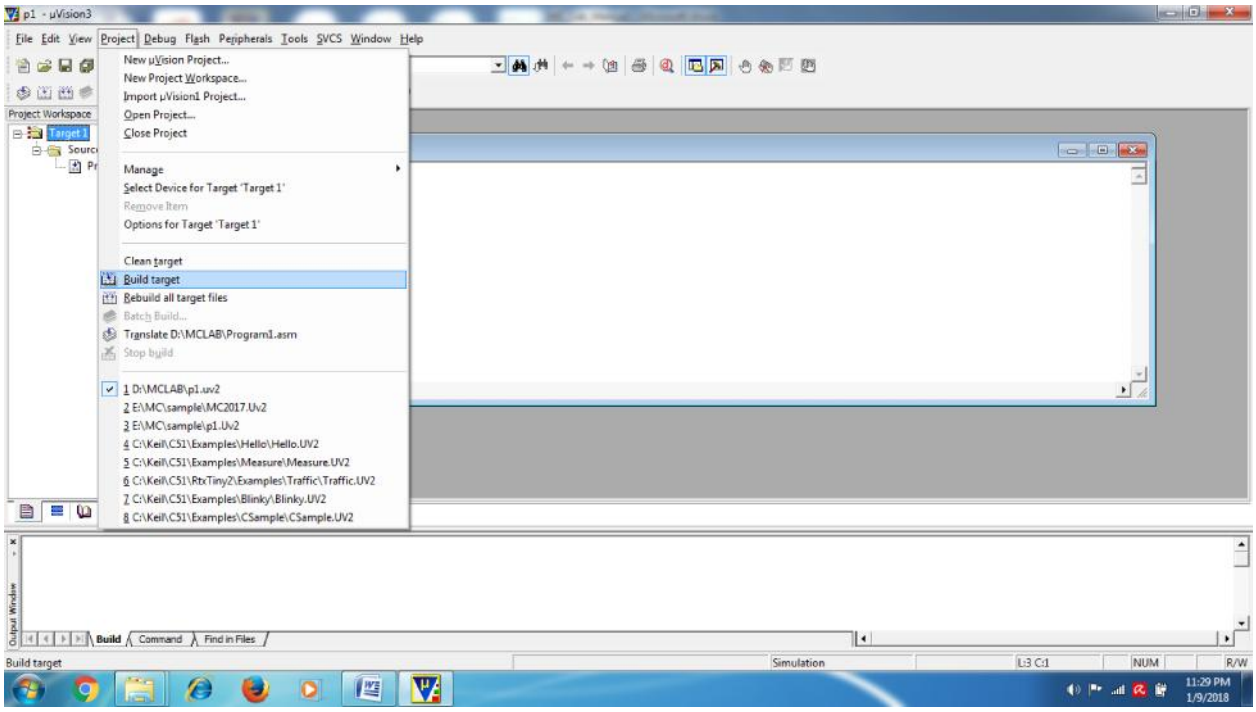
Now, right click on **Source Group1** in the **Project Workspace** and select **Add files to Group 'Source Group1'** as shown in below figure.



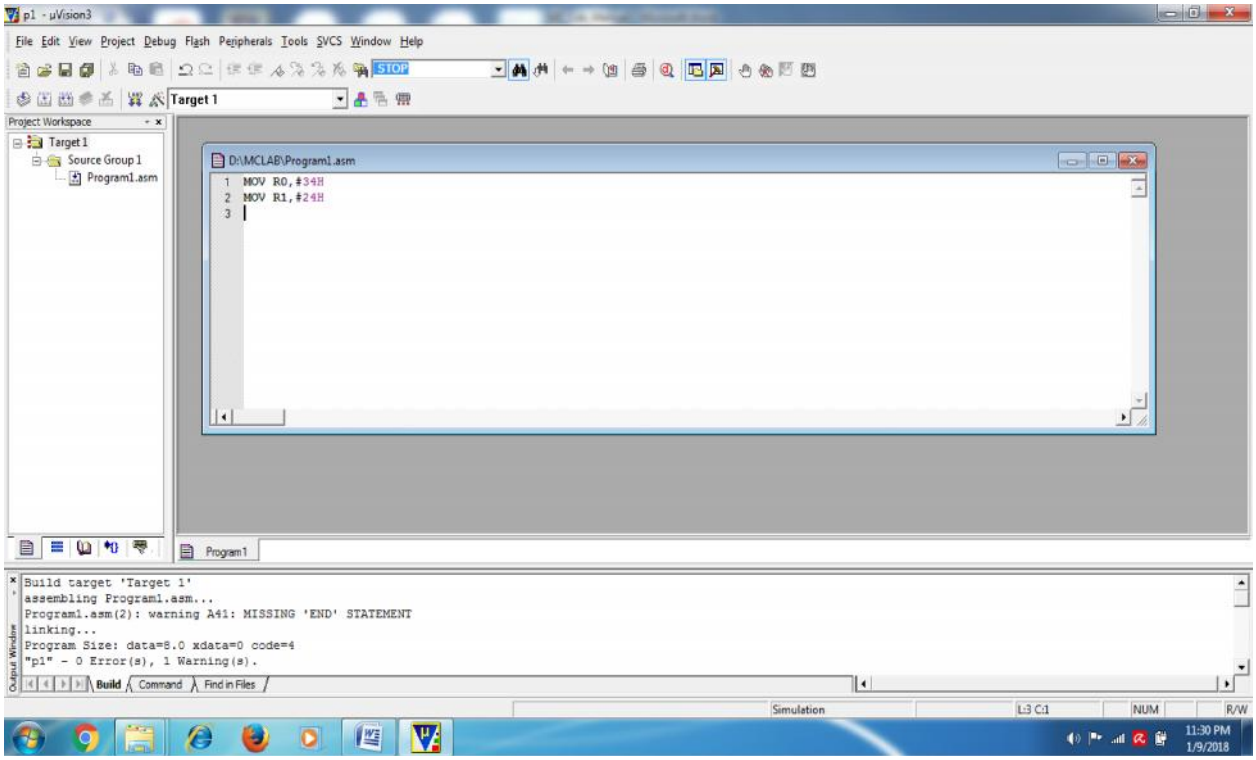
Now, browse your Asm Source file in your folder and click on Add as shown below.



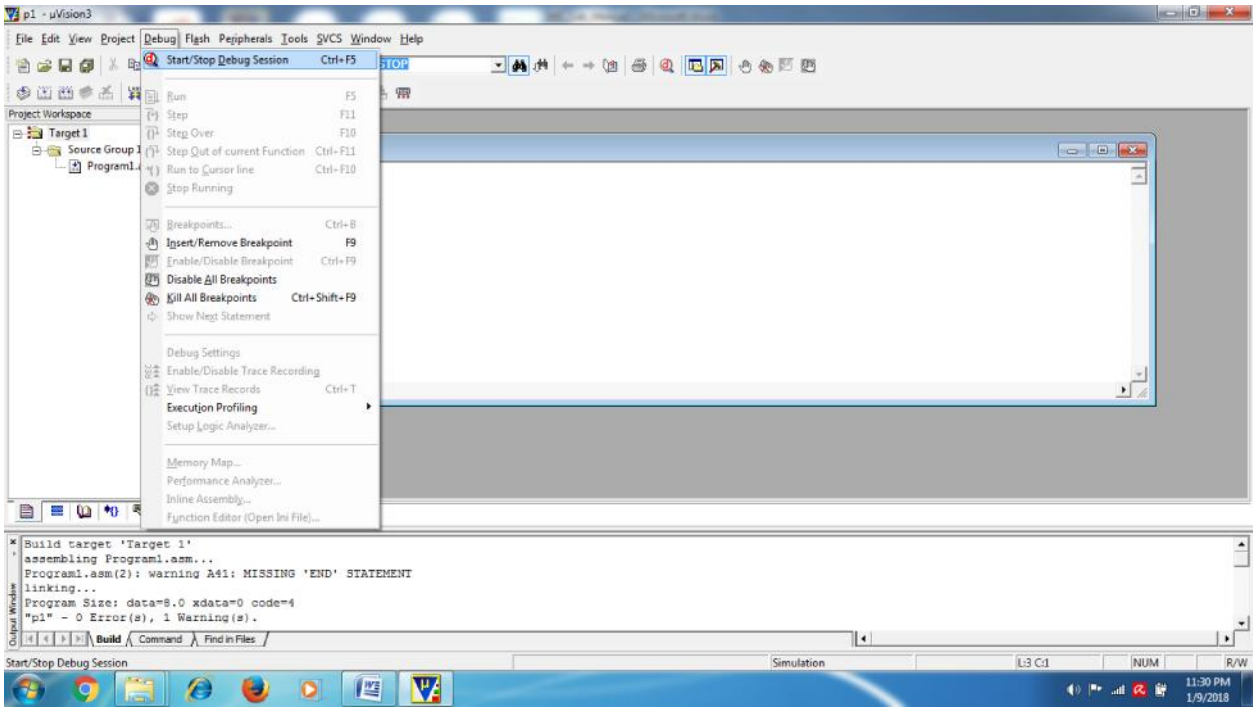
Once the file is added to the **Source Group1**, go to **Project** menu and click on **Build target**.



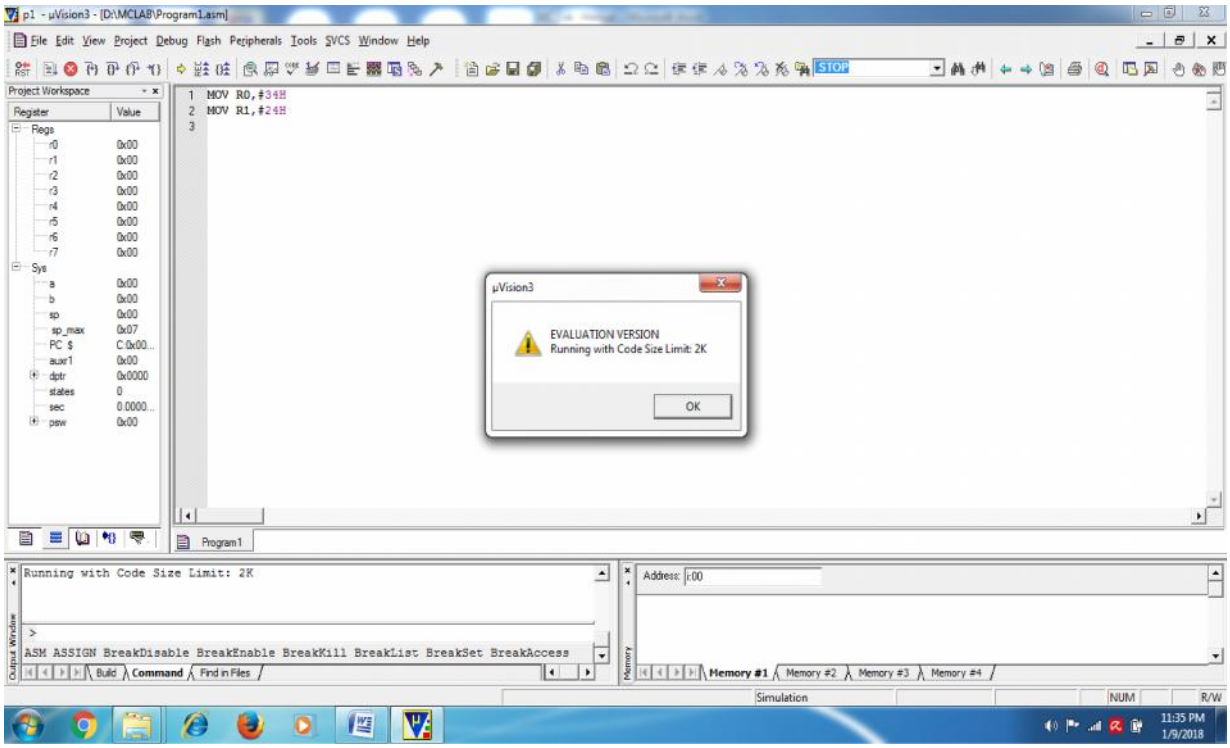
If there are no errors in your Source program then Build target is successful. If any errors, then go to the source program and debug the errors and again **Build target**.



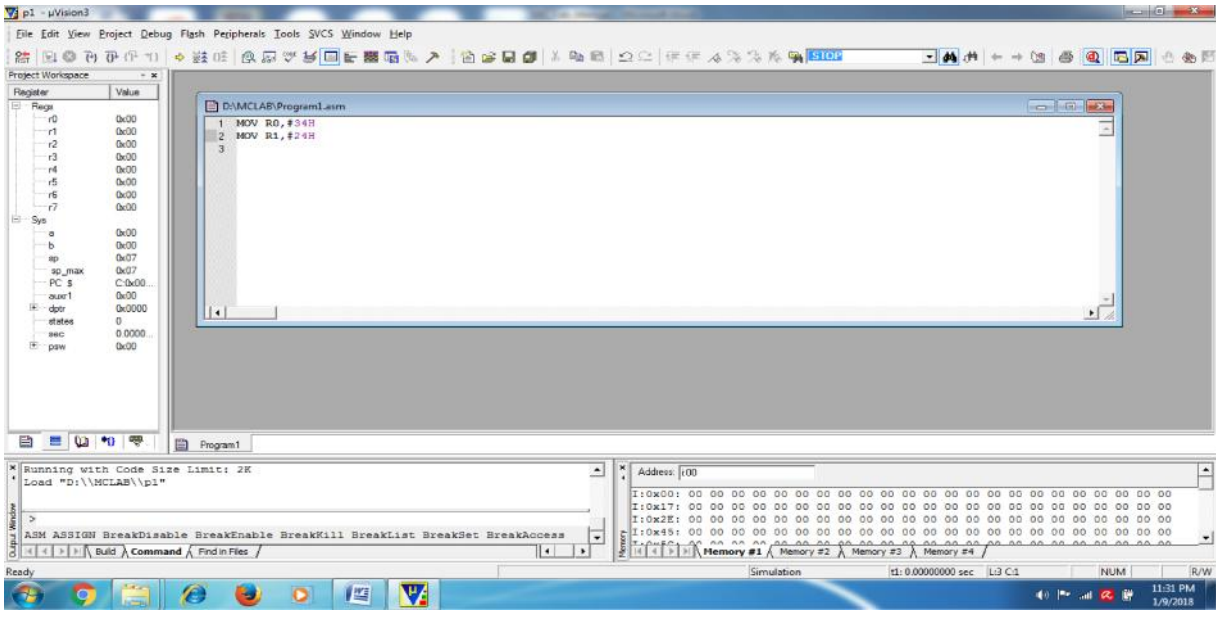
Now, go to **Debug** menu and click on **Start/Stop Debug Session** to start the debug session.



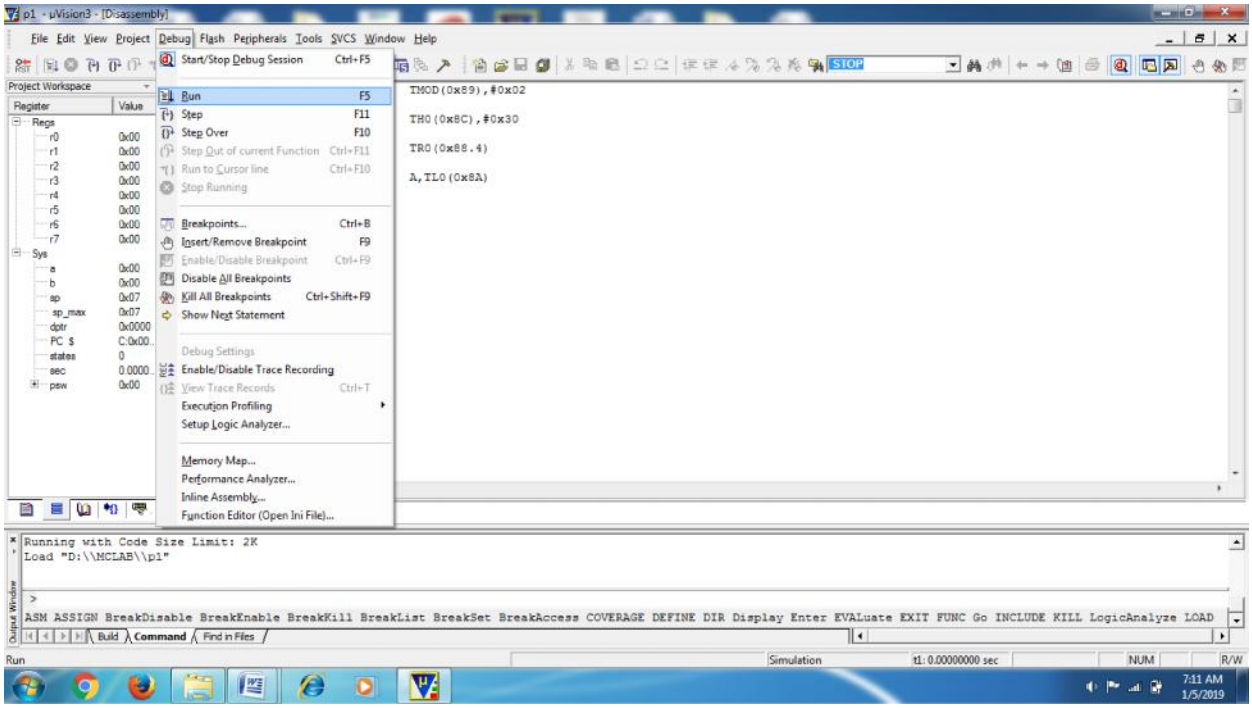
A dialog box appears indicating the code size limit in this open source IDE. Just click **OK**.



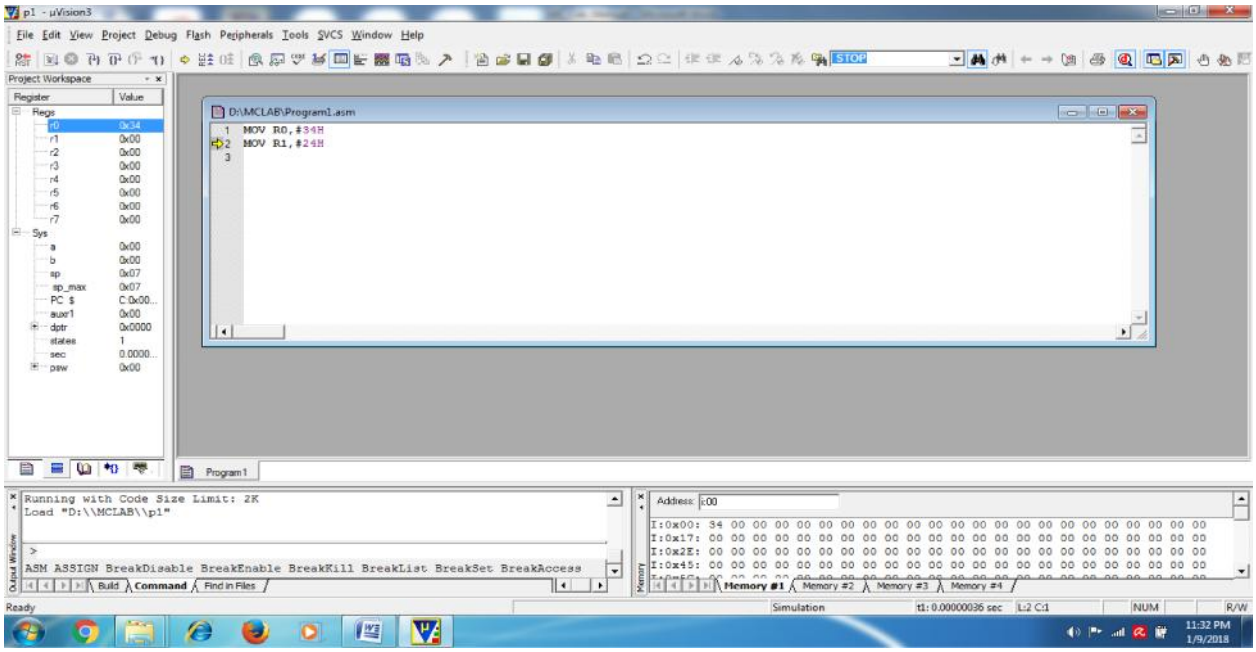
Now, you are in execution mode. You can view several windows in this mode such as Memory Window to see the data in internal or external RAM memory, Disassembly window to see the ROM address of each instruction and machine code generated for each instruction etc.



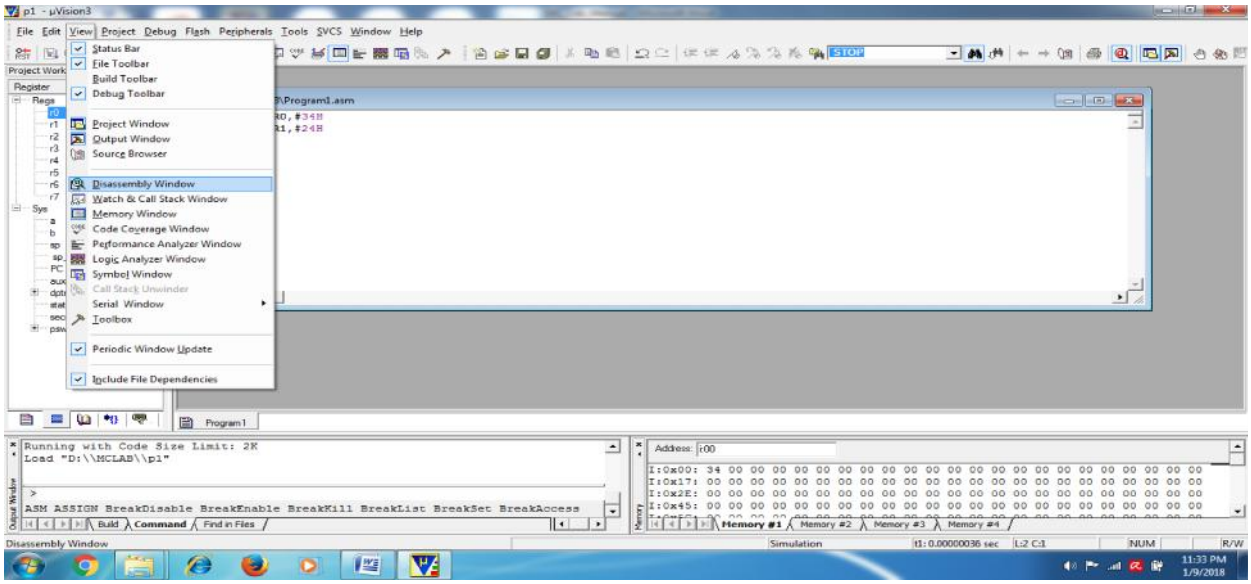
Now, to execute your program step by step, go to **Debug** menu and click on **Step** option as shown in below figure. For whole program execution, click on **Run** option in **Debug** menu.



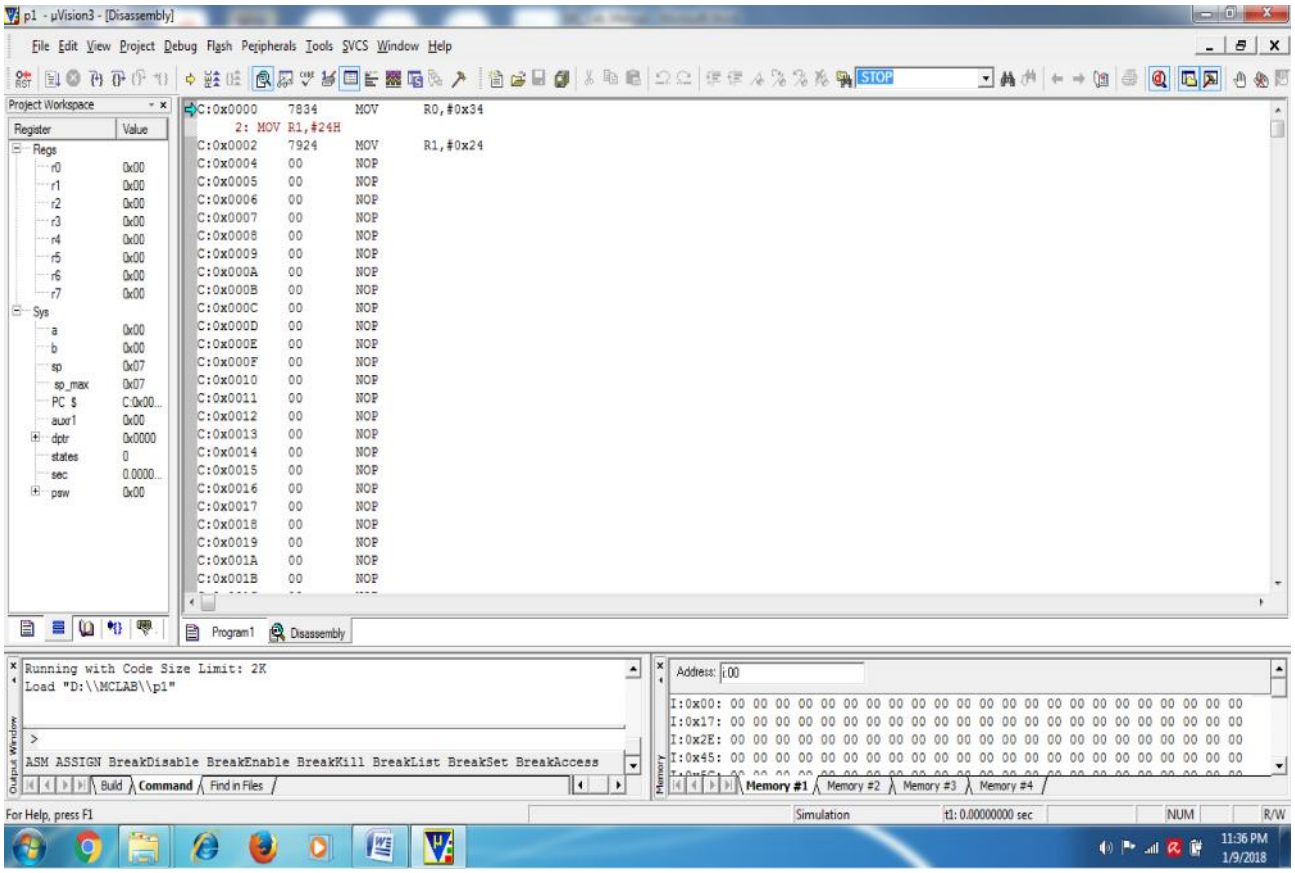
You can view the result of your program execution in appropriate window based on where you are storing the result. i.e., either in register or in memory. To view the result in memory, give **i:00** (i stands for internal RAM) in the text box provided in the memory window as shown in below figure. To view external RAM, give **x:00** in the memory window. You can edit the required memory locations.



If any window is not appearing then go to **View** menu and select required window in it. Below figure shows the selection of **Disassembly Window**.



The Disassembly Window shows the Code memory with each instruction address, the machine code generated for each instruction and also the instructions.



USING FLASH MAGIC TOOL

NXP Semiconductors produce a range of Microcontrollers that feature both on-chip Flash memory and the ability to be reprogrammed using In-System Programming technology.

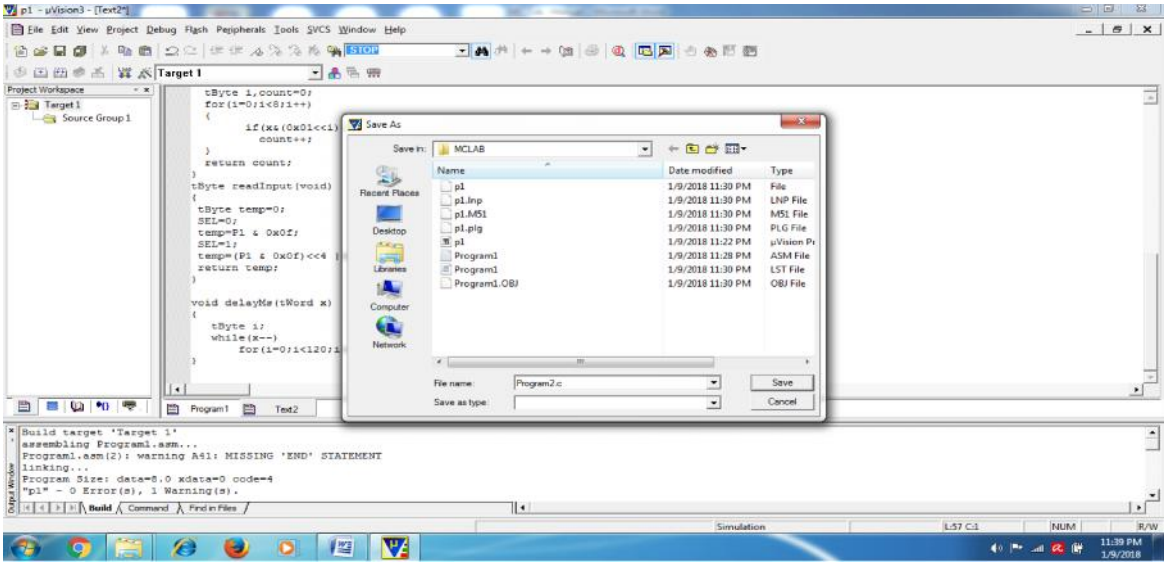
Flash Magic is Windows software from the Embedded Systems Academy that allows easy access to all the ISP features provided by the devices.

The 8051 Microcontroller board is interfaced with All-in-one I/O interface board for working with several I/O interfacing components like Logic controller, Seven segment display, keyboard, stepper motor, elevator, LCD, DAC etc.

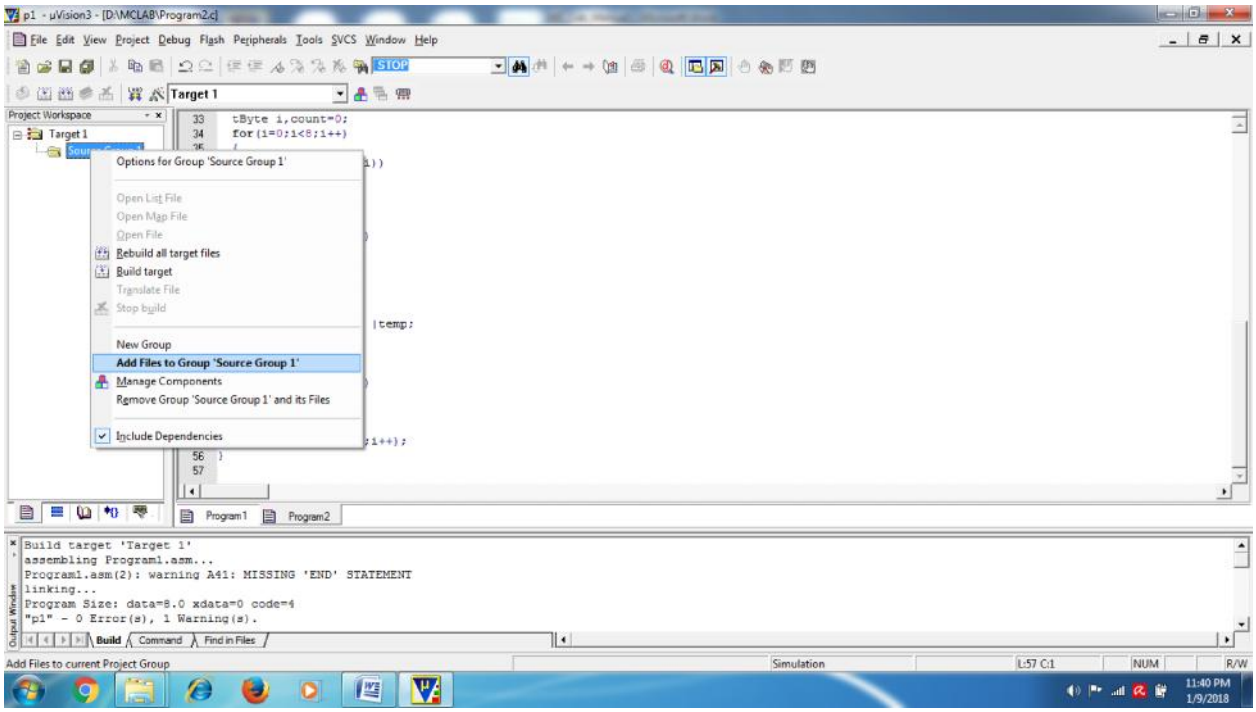
Flash Magic tool is used to dump the Hex file to the ROM of 8051 microcontroller. Before using this tool, the Hex file is to be generated using Keil IDE.

The following steps should be followed to generate the Hex file:

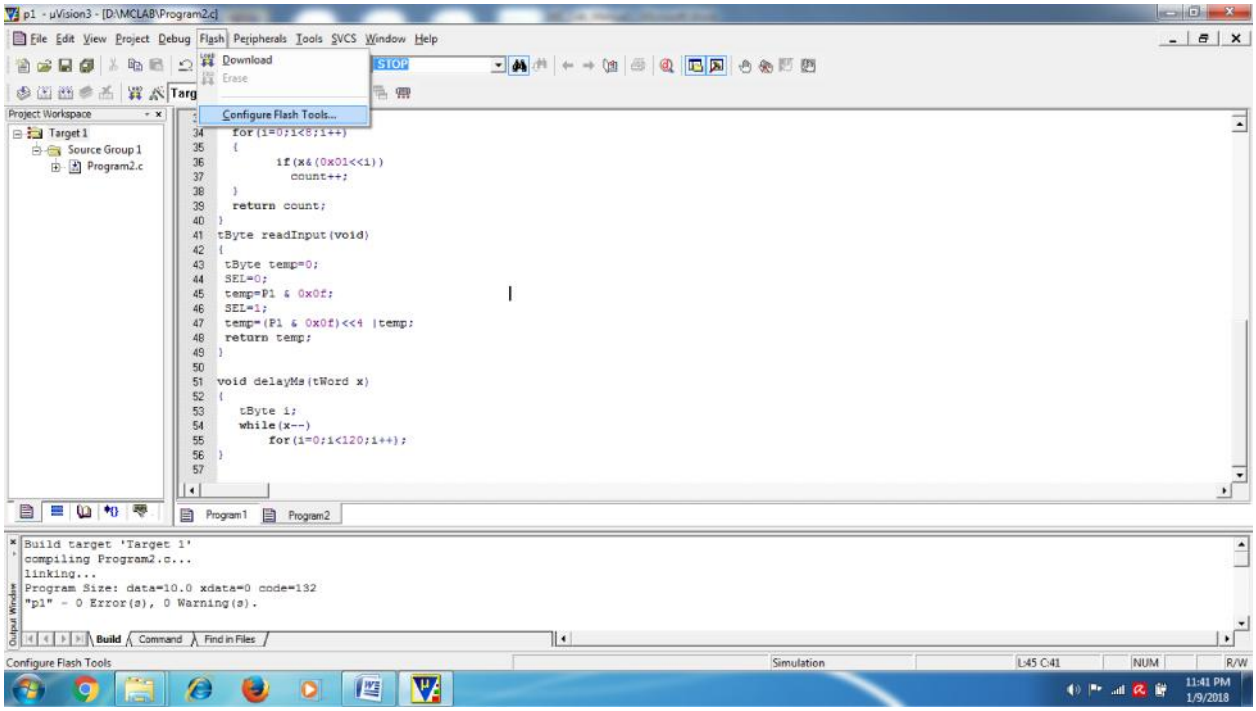
First, create a new file and edit your program and save it with .c extension in your folder as shown in below figure.



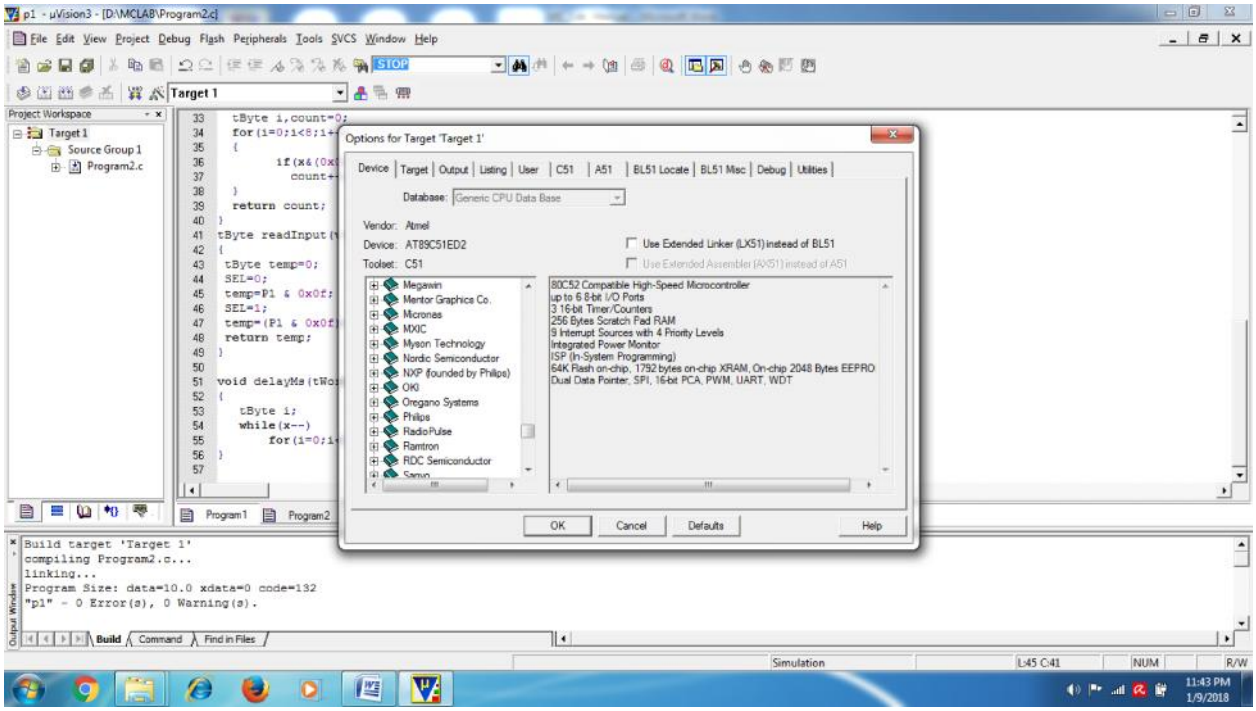
Now, right click on **Source Group1** in the **Project Workspace** and select **Add files to Group 'Source Group1'** as shown in below figure.



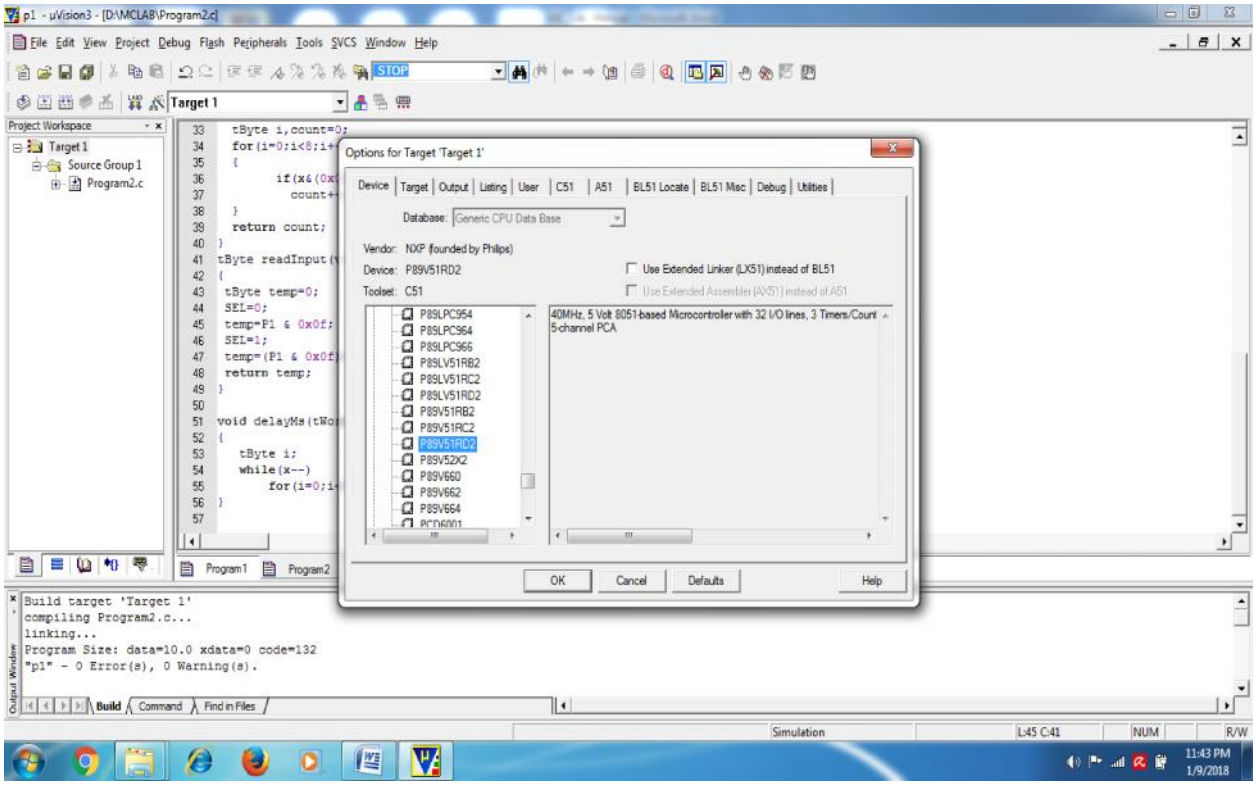
Now, go to **Flash** menu and click on **Configure Flash Tools** option as shown in below figure.



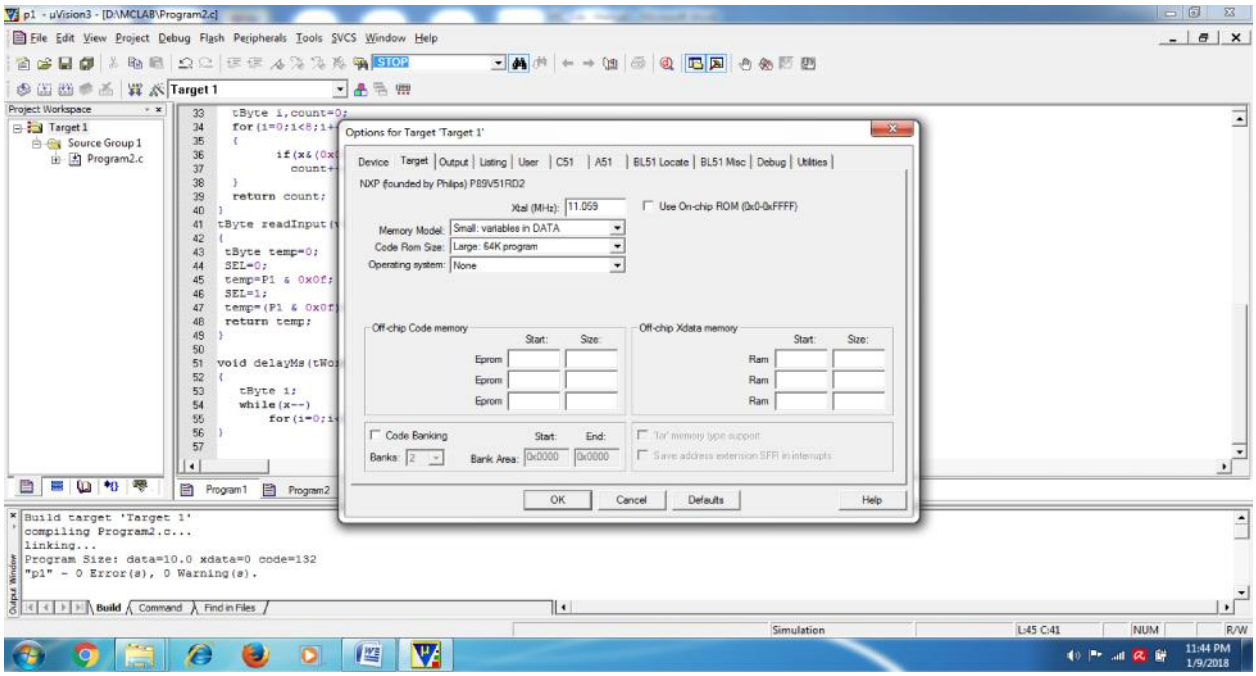
Now, click on Device option and then **NXP (founded by Philips)** to select the device.



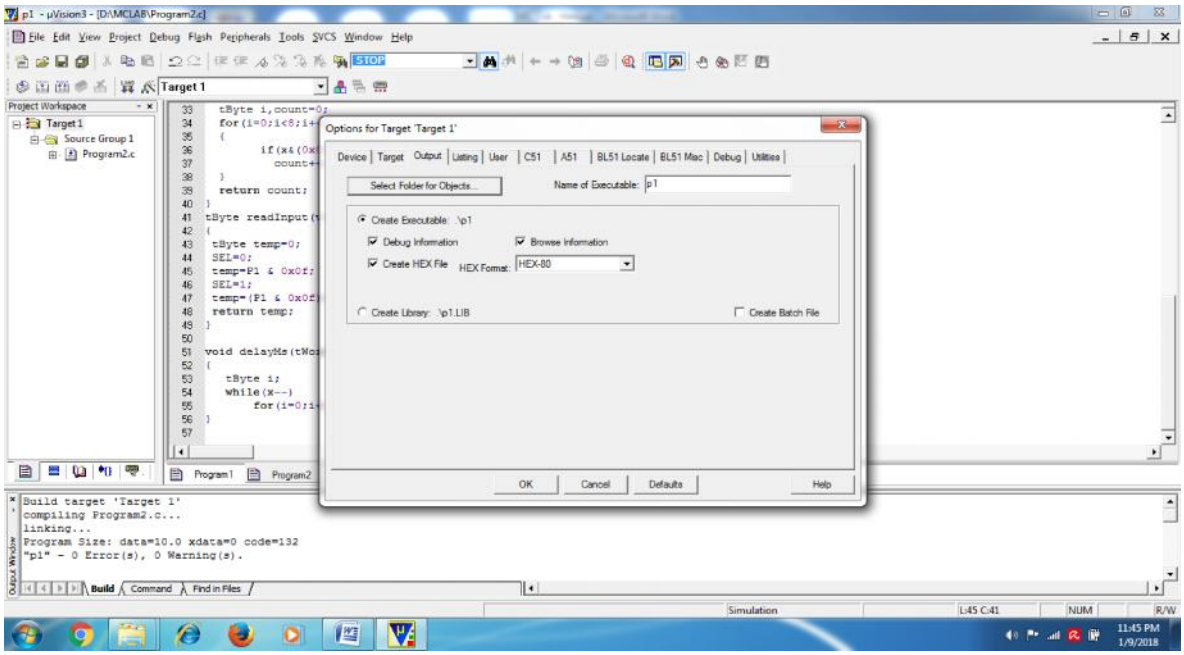
Next, select **P89V51RD2** device.



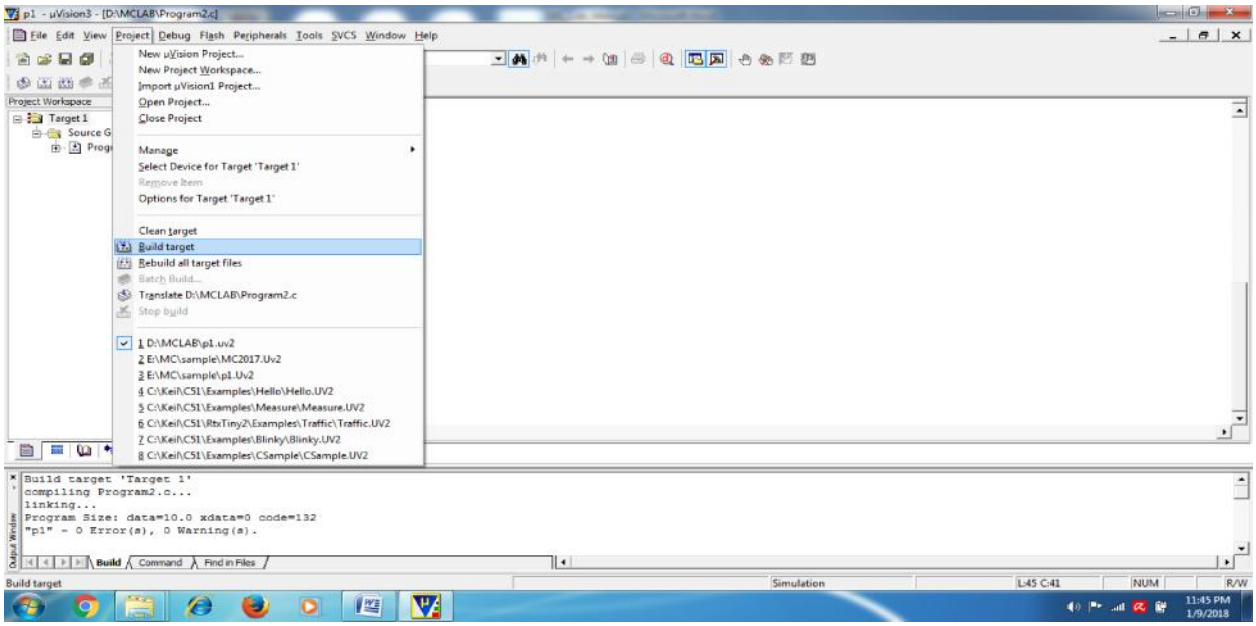
Next, click on **Target** option and give the **Xtal** frequency as **11.059 MHz** as shown in below figure.



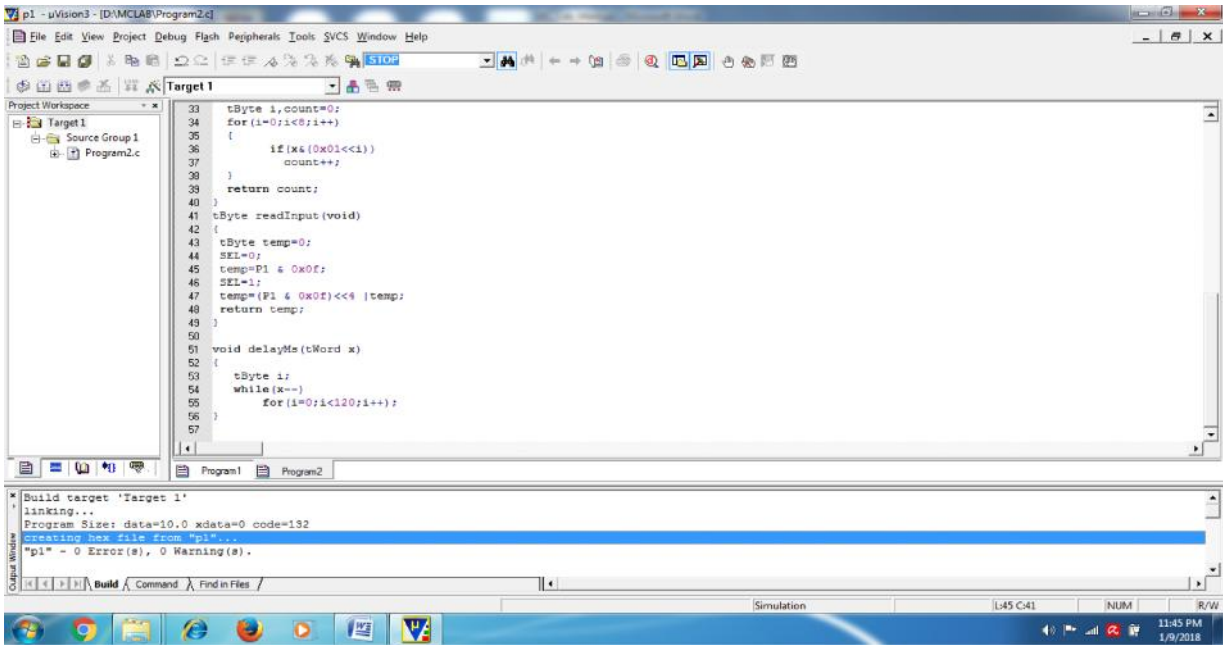
Next, click on **Output** option and select **Create Hex File**. If you want to give new name to the executable file then you can provide it in the textbox of Name of Executable and finally click on **OK** to complete configuration.



Now, go to Project menu and click on Build Target. If there are any errors, then go to the source file and debug it otherwise, target will not be created and the Hex file will not be generated.

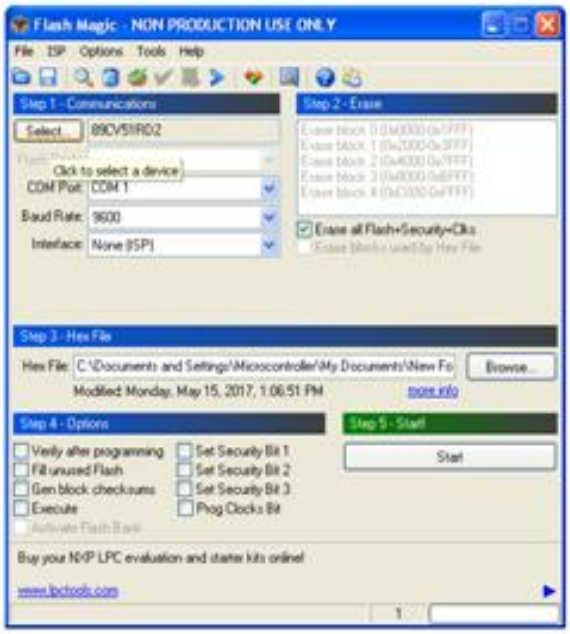


If Build Target is successful then you can see the message **creating hex file from "P1"** in the **output window** as shown in below figure. The Hex file will be created in the current working folder.

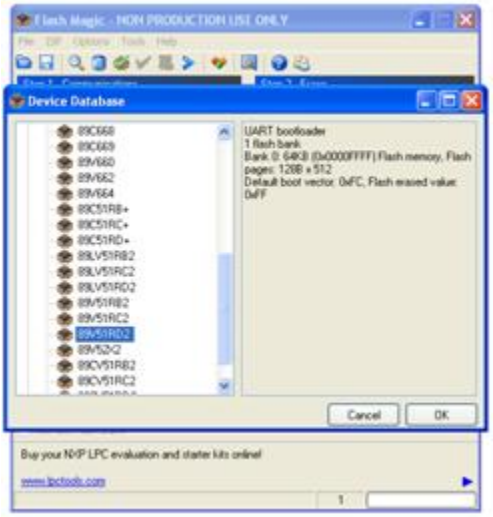


The Hex File generated is dumped into the ROM of actual 8051 Microcontroller using **Flash Magic Tool**. The following steps are to be followed:

To launch Flash Magic, go to **Start** menu and click on **Flash Magic**. The following window appears.



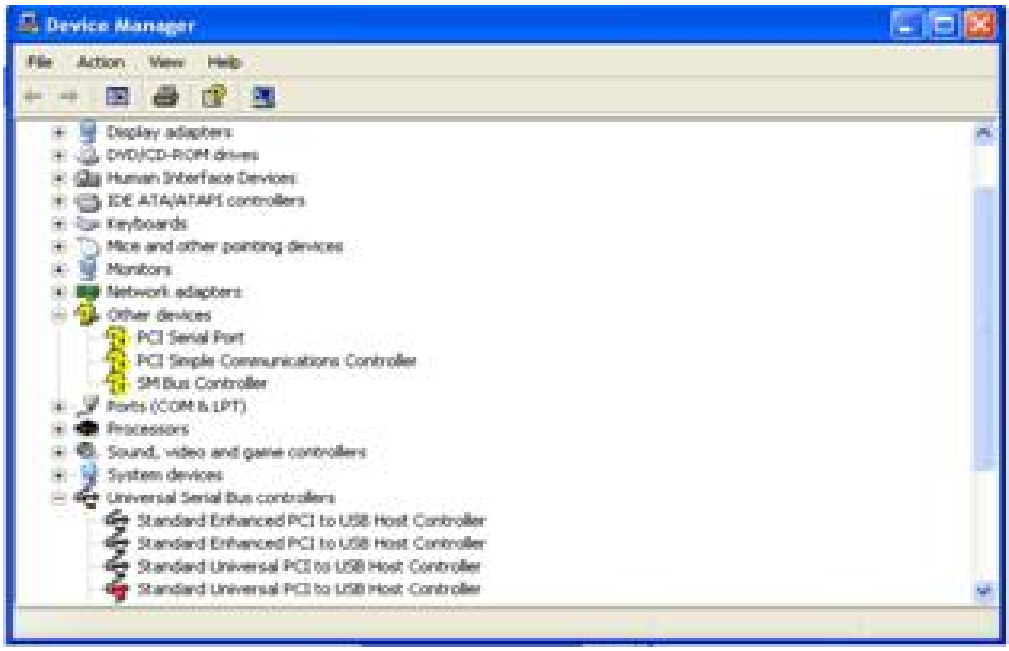
Now, click on **Select** to choose the device **89V51RD2** from the Device Database.



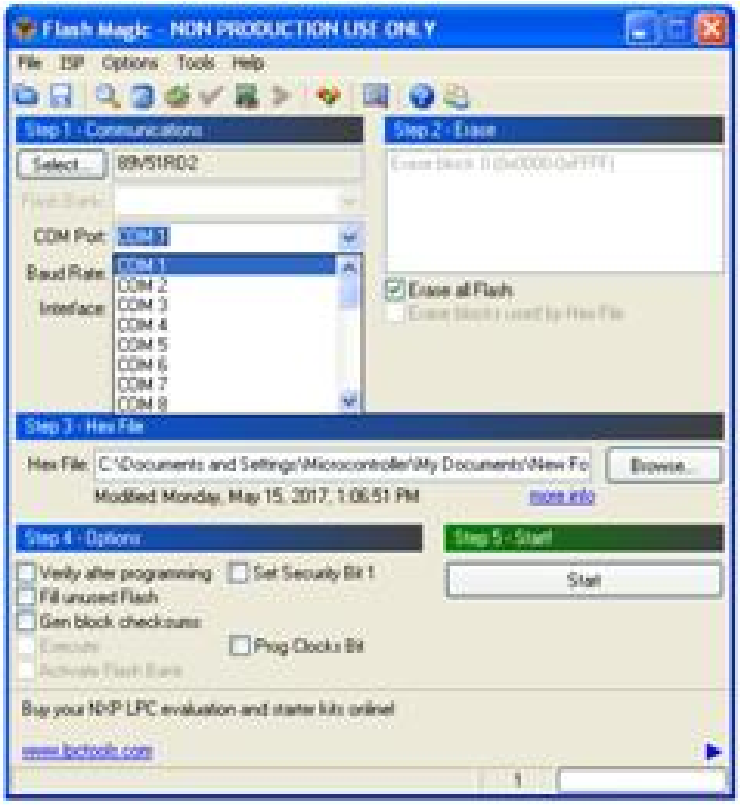
Next step is to choose correct **COM port** number. To do this, first find the COM port number to be used.

Right click on My Computer icon and go to Properties ->Hardware -> Device Manager

The following window appears. Click on **Ports** to find the USB to UART bridge COM port number.

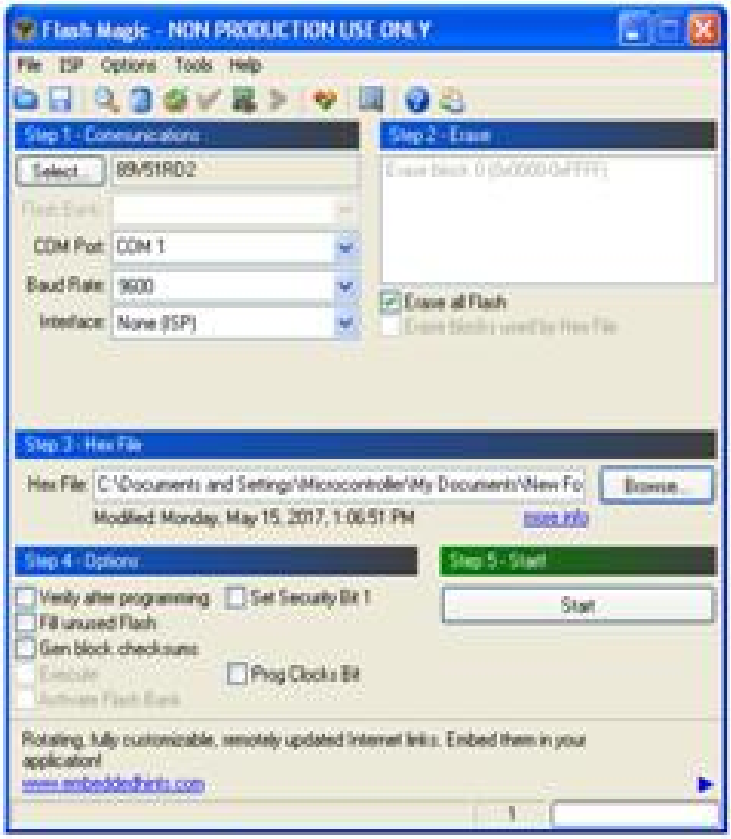


Now, select this **COM port** number from the drop down list as shown in below figure:



Next step is to browse the Hex file by clicking on **Browse** and specify its path where Hex file is created. Then click on Erase all Flash option. Select 9600 Baud rate.

Finally, click on Start to dump the Hex file selected to the ROM of 8051 Microcontroller. Once the process is completed, it shows **finished** in the progress bar.



PRACTICE PROGRAMS

Write an 8051 Assembly Language program for the following:

1. Put the number 34H in registers R5, R6 and R7.

Ans:

2. Put the number 8DH in RAM locations 30H and 31H.

Ans:

3. Copy the data at internal RAM location 7FH to the registers R0 and R3.

Ans:

4. Exchange the contents of the registers R0 and R1.

Ans:

5. Rotate the bytes in registers R0 to R3. i.e., copy the data in R0 to R1, R1 to R2, R2 to R3, and R3 to R0.

Ans:

6. Swap the nibbles of the register R0.

Ans:

7. Copy the data in external RAM locations 0A00H and 0A01H to the registers R0 and R1 respectively.

Ans:

8. Copy the byte at internal RAM address 4AH to external RAM address 0ABCH.

Ans:

9. Copy the data in register R5 to external RAM address 032FH.

Ans:

10. Store DPTR in external RAM locations 0123H (DPL) and 02BCH (DPH).
    (RAM address of DPL is 82H and DPH is 83H)

Ans:

11. Clear bit3 of RAM location 22H without affecting any other bits. (Assume 22H contains the data FFH)

Ans:

12. Set bit 5 of register R2 without affecting any other bits. (Assume R2 contains the data 57H)

Ans:

13. Complement the contents of register R4.

Ans:

14. Store the most significant nibble of A in both the nibbles of R5. (Assume A = A4H)

Ans:

15. Find a number that when XORed to the A register, results in the number 3FH in A.

Ans:

16. Assume register A contains 34H and register R0 contains 52H. Add the contents of register A and R0 and store the result in A.

Ans:

17. Make register bank3 active and store the data C6H in register R4 of bank3.

Ans:

18. Add the number 84H to RAM locations 17H and 18H. (Assume 17H contains 22H and 18H contains 33H)

Ans:

19. Add the byte in external RAM location 02CDH with internal RAM location 19H and store the result in internal RAM location 25H. (Assume 02CDH contains 65H and 19H contains 3FH)

Ans:

20. Subtract the contents of register R1 from R0 and put the result in R7.

Ans:

21. Subtract the contents of register R2 from the number F3H and put the result in external RAM location 028BH.

Ans:

22. Subtract the contents of RAM location 13H from RAM location 2BH and put the result in RAM location 3CH.

Ans:

23. Decrement the contents of external RAM locations 0123H and 01CDH.

Ans:

24. Multiply the data in A and B registers and put the result in R0 (LS byte) and R1 (MS byte).

Ans:

25. Multiply the data in RAM location 22H by the data in RAM location 15H and put the result in RAM locations 19H (LS byte) and 1AH (MS byte).

Ans:

26. Square the contents of R5 and put the result in R0 (LS byte) and R1 (MS byte).

Ans:

27. Divide the data in RAM location 3EH by the number 12H and put the quotient in R4 and the remainder in R5.

Ans:

28. Divide the number in RAM location 15H by the number in RAM location 16H and put the result in external RAM location 034AH (Quotient) and 034BH (Remainder).

Ans:

29. Put the number 85H in RAM locations 30H to 39H.

Ans:

30. Copy the data in internal RAM locations 12H to 15H to internal RAM locations 20H to 23H.

Ans:

31. Transfer a block of data of length 5 from 9000H to 9100H.

Ans:

32. Copy program bytes from 0000H-0009H to the internal RAM locations 20H-29H.

Ans:

33. Move bit4 of RAM location 30H to bit2 of B.

Ans:

34. Add the unsigned numbers found in internal RAM locations 25H, 26H, and 27H together and put the result in RAM locations 30H (LS byte) and 31H (MS byte).

Ans:

35. Check whether content of R7 is less than 75H. If yes, then set bit 30H. Otherwise reset bit 30H.

Ans:

36. Find the square root of a number stored in R0 register.

Ans:

37. Count the number of equal bytes between memory blocks 10H-19H and 30H-39H.

Ans:

38. Transfer the numbers in the location 10H-14H that are divisible by3 to the location starting at address 20H.

Ans:

39. Check whether the character stored in memory location 10H is in the range 30H-39H. If yes, then store the data 10H at location 11H. Otherwise, store 05H at location 11H.

Ans:

40. Count the number of times the data 25H is present in the location 11H-1AH. Store the count in location 1BH.

Ans:

Write an 8051 Embedded C program for the following:

1. Send 00H and FFH to port P1. Introduce some delay after each value.

Ans:

2. Toggle bits of P1 continuously forever with some delay.

Ans:

3. Send values 00H-FFH to port P1.

Ans:

4. Send the Hex values for ASCII characters 0, 1, 2, 3, 4, 5, A, B, C and D to port P1.

Ans:

5. Send values - 4 to +4 to port P1.

Ans:

PART – A

PROGRAM - 1: Write an ALP to exchange the block of data of length 'N' stored starting at RAM address 9000H and 9100H.

Description: This program exchanges two blocks of data where each block consists of 5 bytes of data. Below figure shows sample five bytes of data stored in each block before and after execution.

Before Execution

9000H	10H	9100H	60H
9001H	20H	9101H	70H
9002H	30H	9102H	80H
9003H	40H	9103H	90H
9004H	50H	9104H	A0H

After Execution

9000H	60H	9100H	10H
9001H	70H	9101H	20H
9002H	80H	9102H	30H
9003H	90H	9103H	40H
9004H	A0H	9104H	50H

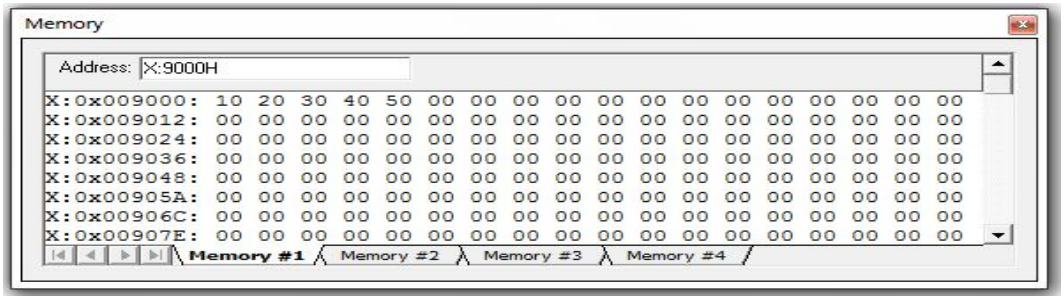
Program:

```
; TO EXCHANGE BLOCK OF DATA
        MOV DPTR,#9000H
        MOV R0,#5               ;block length

AGAIN:MOVX A,@DPTR
        MOV R1,A                ; Save the data in R1 register

        INC DPH
        MOVX A,@DPTR

        DEC DPH
        MOVX @DPTR,A

        MOV A,R1
        INC DPH
        MOVX @DPTR,A

        DEC DPH
        INC DPL
       DJNZ R0,AGAIN
HERE: SJMP HERE
        END
```

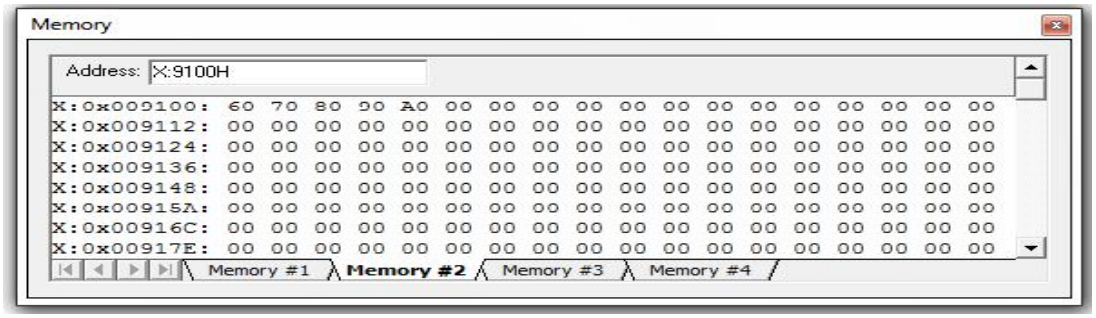
SAMPLE OUTPUT:

Before Execution:

The following five bytes of data are stored starting at external RAM address 9000H:

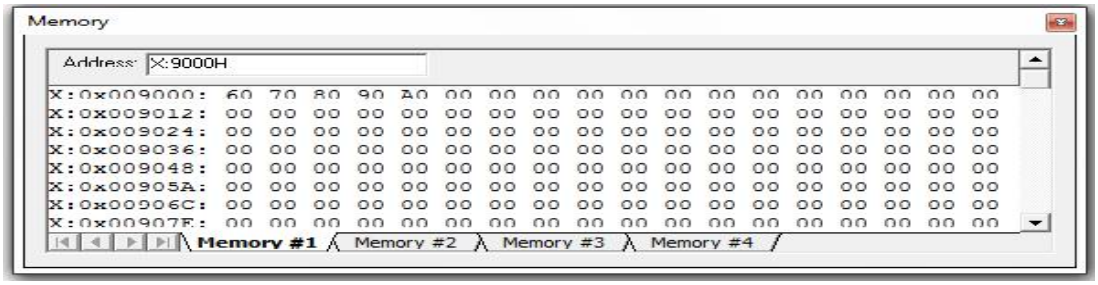


The following five bytes of data are stored starting at external RAM address 9100H:

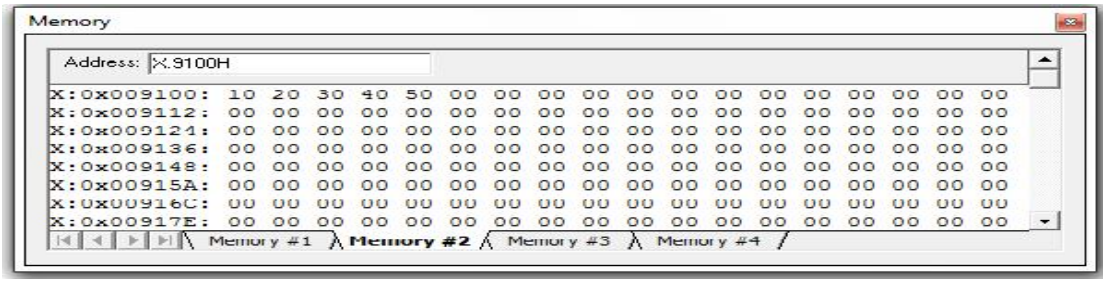


After Execution:

The following five bytes of data are stored starting at external RAM address 9000H:



The following five bytes of data are stored starting at external RAM address 9100H:



PROGRAM – 2: Write an ALP to add 'N' BCD numbers stored starting at RAM address 2000H. Store the result in the next consecutive locations.

Description: An 8-bit BCD number can contain two 4 bit BCD digits each ranging from 0 – 9. For example, 25 is a valid BCD number with two BCD digits 2 and 5 i.e., 0010_{BCD} $0101_{BCD.}$This program is to add N BCD numbers which are stored in external RAM. Here, we have considered 11 BCD numbers. Each time the addition operation is performed, we use DA A (Decimal Adjust Accumulator) instruction to adjust the result to correct BCD form. The higher byte result is held in R1 and the lower byte result is held in R2. These register contents are stored in consecutive memory locations after the 11 bytes of input data.

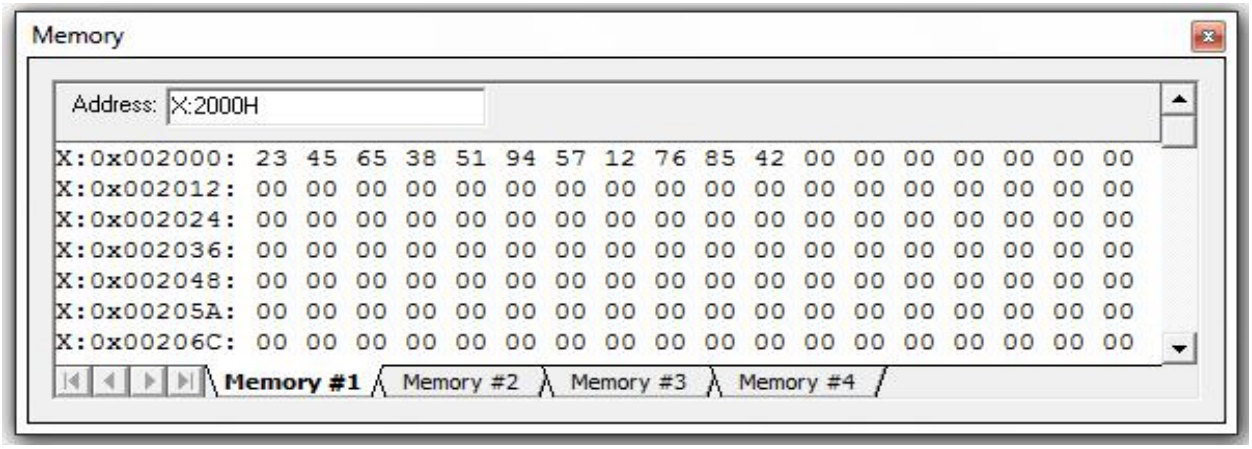
Program:

```
; TO ADD N BCD NUMBERS

          MOV DPTR,#2000H
          MOV R0,#11     ;Total number of BCD numbers
          MOV R1,#00     ;To hold higher byte of the result
          MOV R2,#00     ;To hold the lower byte of the result
AGAIN:    MOVX A,@DPTR
          ADD A,R2
          DA A
          MOV R2,A
          JNC NEXT

          CLR A
          MOV A,R1
          ADD A,#01
          DA A
          MOV R1,A

NEXT:     INC DPTR
          DJNZ R0,AGAIN
          MOV A,R1
          MOVX @DPTR,A

          INC DPTR
          MOV A,R2
          MOVX @DPTR,A
HERE: SJMP HERE
      END
```

SAMPLE OUTPUT:

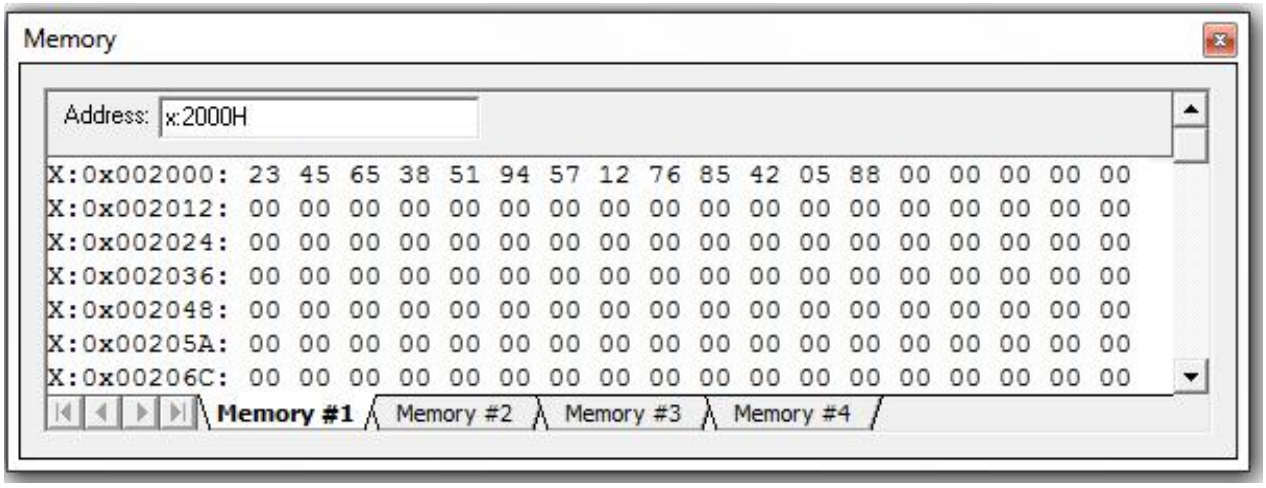
Before Execution:

The following eleven bytes of BCD data are stored starting at external RAM address 2000H:



After Execution:

The resultant sum 588 is stored in next consecutive two memory locations after eleven bytes of input BCD data, i.e., at locations 200BH and 200CH respectively.



PROGRAM – 3: Write an ALP to count the number of odd and even numbers in a block of 'N' numbers stored starting at RAM address 1000H. Store the result in the next consecutive locations.

Description: This program is to count the number of odd numbers and even numbers in a block of data containing N numbers. Here, we have considered 5 bytes of data. We have used two registers, R0 and R1 as counters to maintain the count of odd and even numbers respectively. Since the LSB (Least Significant Bit) of all even numbers is always 0 and for all odd numbers it is always 1, we have used RRC (Rotate Right through Carry) instruction to move the LSB to the carry flag. Then, we test the carry flag status to decide whether the input byte is odd or even. The counter values are stored in external RAM in next consecutive memory locations after the input bytes.

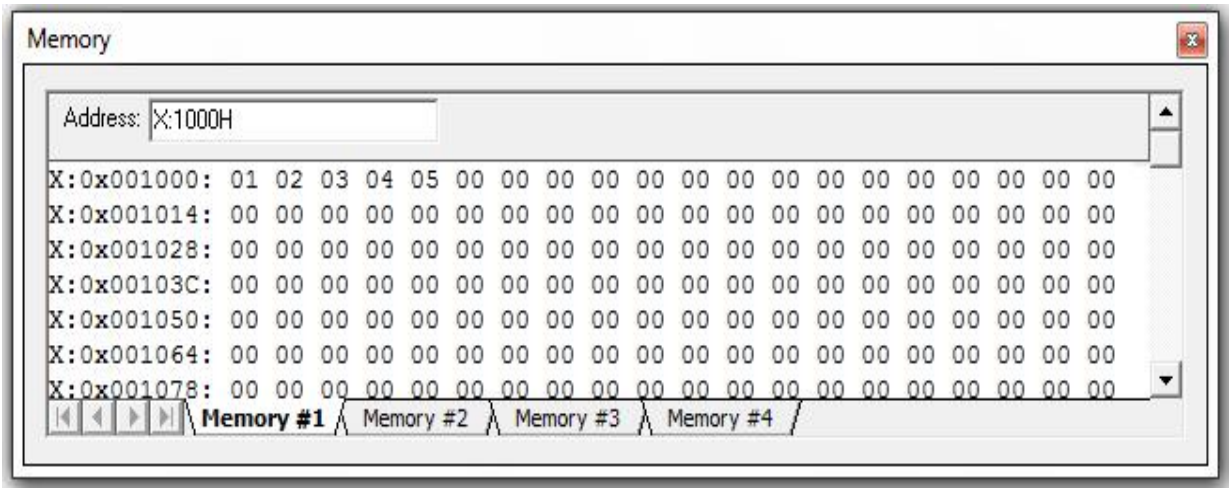
Program:

```
;TO COUNT ODD AND EVEN NUMBERS IN AN ARRAY

          MOV R0,#00H          ; Odd number counter
          MOV R1,#00H          ; Even number counter
          MOV R2,#5            ; No. of bytes in the block
          MOV DPTR,#1000H
AGAIN:    MOVX A,@DPTR
          RRC A
          JNC EVEN
          INC R0
          SJMP NEXT
EVEN:     INC R1
NEXT:     INC DPTR
          DJNZ R2,AGAIN

          MOV A,R0
          MOVX @DPTR,A  ;store odd no. count in memory

          INC DPTR
          MOV   A,R1
          MOVX @DPTR,A  ;store even no. count in memory
HERE: SJMP HERE
   END
```

SAMPLE OUTPUT:

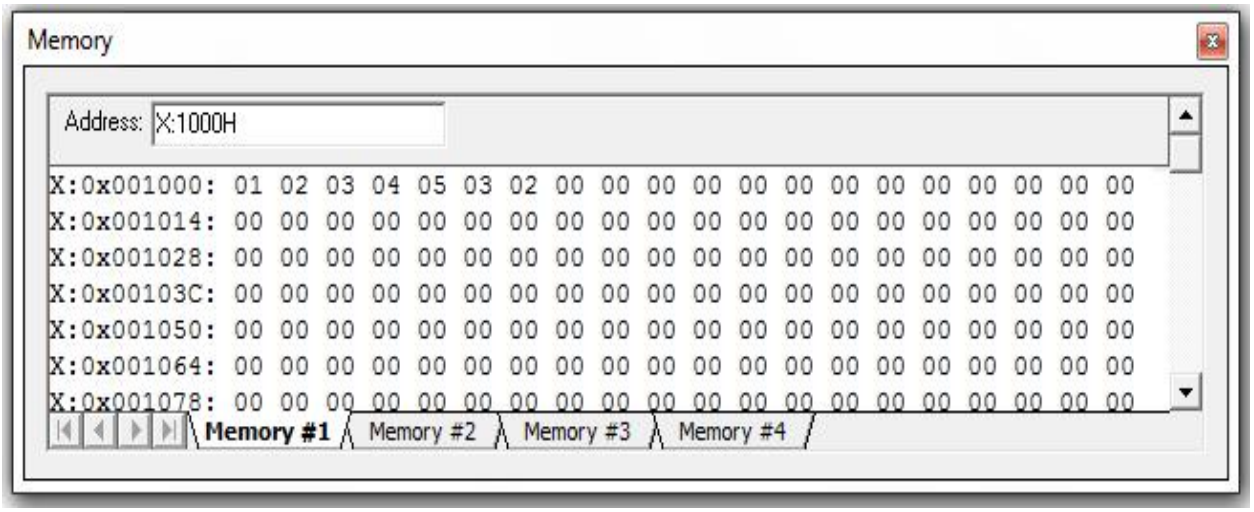
Before Execution:

The following five bytes of data are stored starting at external RAM address 1000H:



After Execution:

The odd numbers and even numbers count are stored in next consecutive two memory locations after five bytes of input data i.e., at 1005H and 1006H respectively.



PROGRAM – 4: Write an ALP to add two multi-byte numbers stored at RAM address 9000H and 9100H. Store the multi-byte result at RAM address 9200H.

Description: This program is to add two multi-byte numbers which are stored in external RAM. Here, we have considered two 5-byte numbers. Starting from the least significant bytes in both the numbers, we perform addition byte by byte and add the next consecutive bytes along with the carry using ADDC (ADD with Carry) instruction. The two 5-byte numbers and the resultant number are stored in Little Endian format in the memory.

Example: 3475514312H

+9247351454H

C6BC865766H

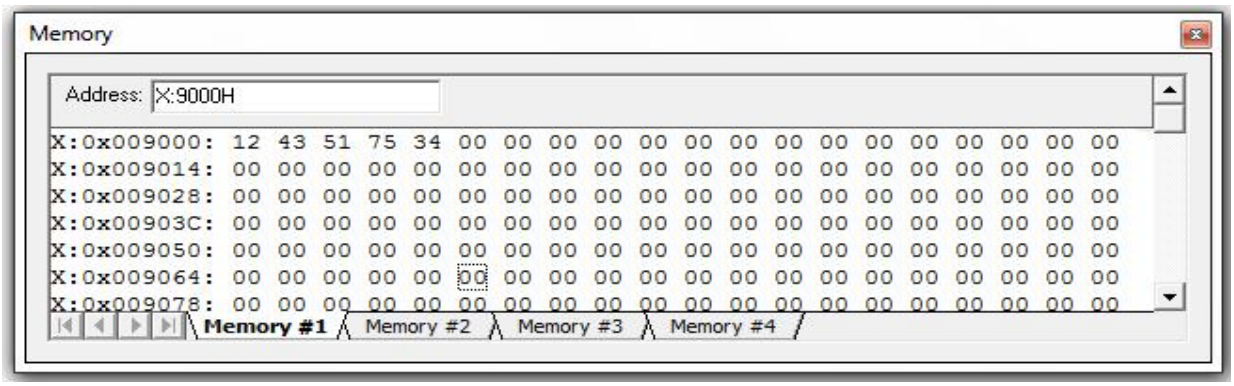
Program:

```
; TO ADD TWO MULTIBYTE NUMBERS

            MOV DPL,#00H
              MOV R1,#5        ; No. of bytes in the input number
               MOV R2,#90H
               MOV R3,#91H
              MOV R4,#92H
               CLR C

NXT_BYTE:   MOV DPH,R2
               MOVX A,@DPTR
               MOV R5,A

               MOV DPH,R3
               MOVX A,@DPTR

               ADDC A,R5
               MOV DPH,R4
               MOVX @DPTR,A

               INC DPL
               DJNZ R1,NXT_BYTE
               CLR A
               ADDC A,R1
             MOVX @DPTR, A
      HERE: SJMP HERE
              END
```

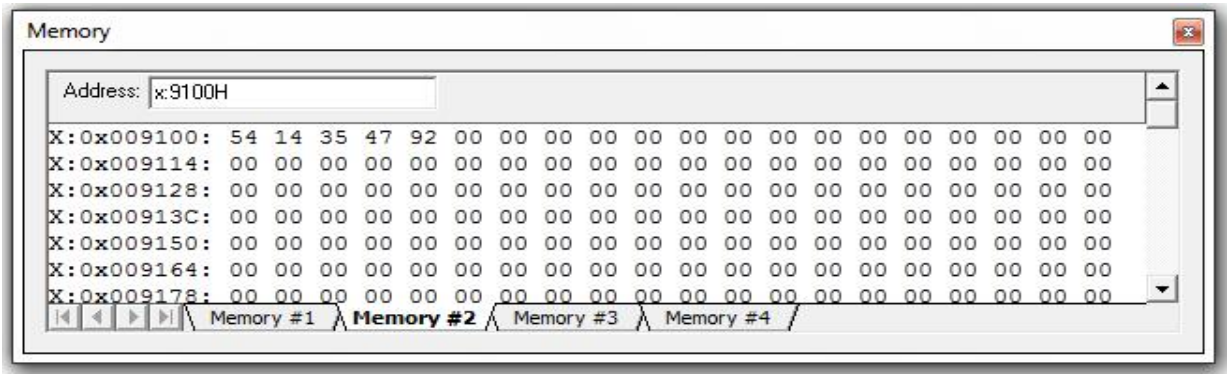
SAMPLE OUTPUT:

Before Execution:

The following five byte number is stored starting at external RAM address 9000H in Little Endian format:

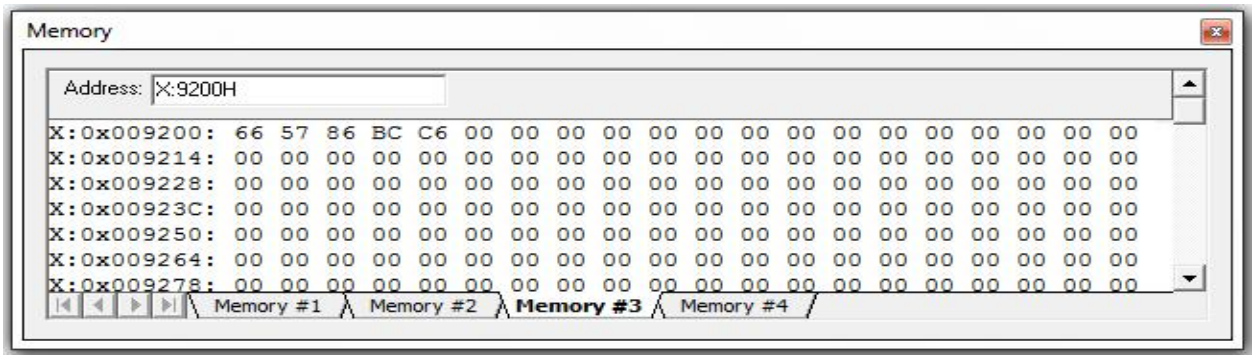


The following five byte number is stored starting at external RAM address 9100H in Little Endian Format:



After Execution:

The sum of two five-byte numbers is stored starting at address 9200H in Little Endian Format.



PROGRAM – 5: Write an ALP to search for the key element in a block of 'N' bytes. If the number is present, show its position in the RAM location, 1050H. Otherwise, show FFH in 1050H location. Assume the key element is stored at RAM address 1000H and the data block starts at the RAM location 1001H.

Description: This program is to search for a specific key element in a block of N bytes of data. Here, we have considered 4 bytes of data in the memory block. We use simple linear search technique to search for the key element. If it is a successful search then, we store the position of the key element at address 1050H. Otherwise, we store FFH at address 1050H to indicate the failure status.

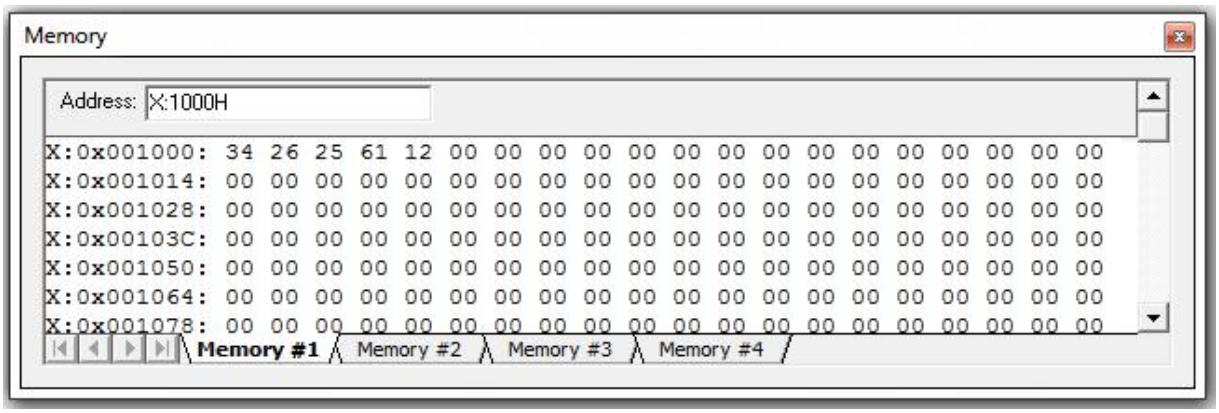
Program:

```
; LINEAR SEARCH

RESULT EQU 1050H
        KEY EQU 1000H
        SIZE EQU 4H

        MOV DPTR,#KEY     ; RAM address of key element
        MOV R1,#00H
        MOVX A,@DPTR
        MOV R0,A

BACK:  INC DPTR
        MOVX A,@DPTR
        CJNE A,00H,NEXT
        SJMP SUCCESS

NEXT:  INC R1
        CJNE R1,#SIZE,BACK
        MOV R1,#0FFH
SUCCESS:MOV DPTR,#RESULT
           MOV A,R1                ; Transfer position of key element to A
            MOVX @DPTR,A
HERE:    SJMP HERE
        END
```

SAMPLE OUTPUT1:

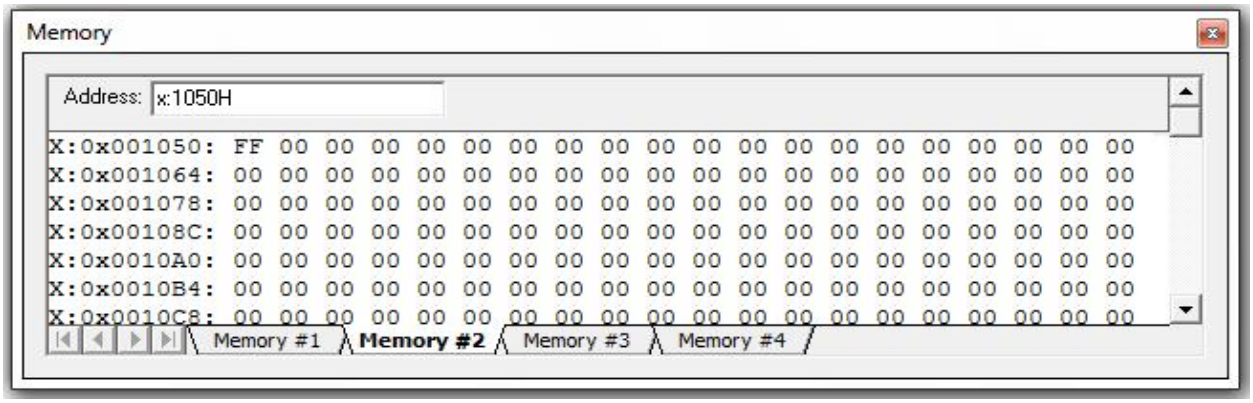
Before Execution:

The key element 34H is stored at external RAM address 1000H and the four array elements are stored starting at external RAM address 1001H:



After Execution:

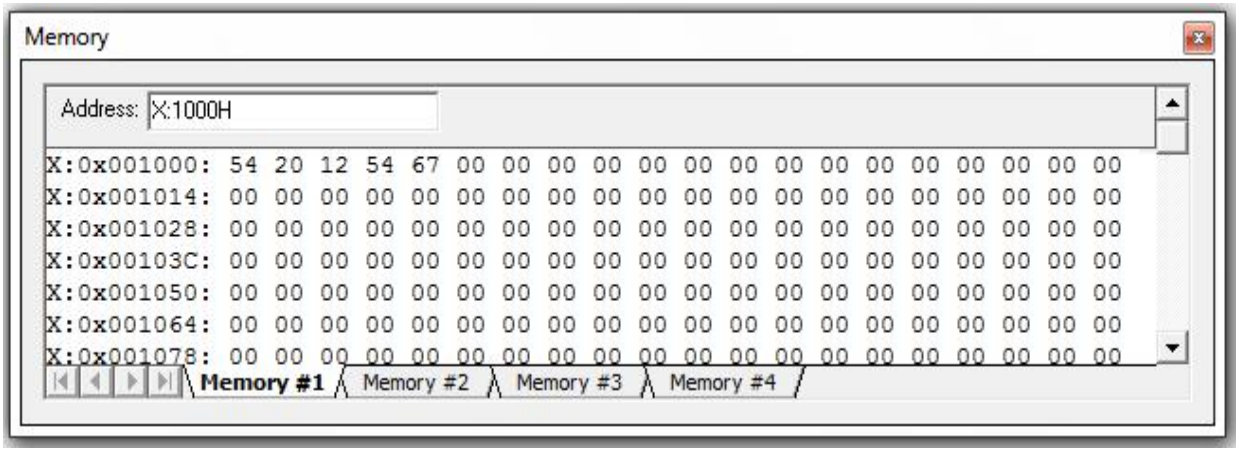
Since the key element 34H is not found in the array, FFH is stored at external RAM address 1050H to indicate failure.



SAMPLE OUTPUT2:

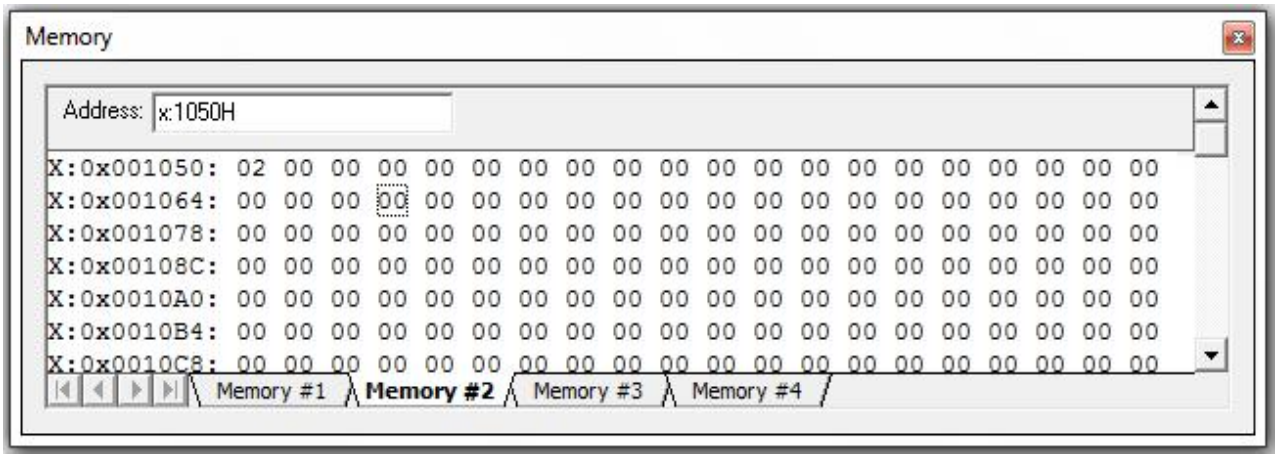
Before Execution:

The key element 54H is stored at external RAM address 1000H and the four array elements are stored starting at external RAM address 1001H:



After Execution:

Since the key element 54H is found in the array at position 02, the external RAM address 1050H contains the position 02 to indicate successful search.



PROGRAM – 6: Write an ALP to convert the binary number stored at RAM location 1000H into BCD and store the result in the next consecutive locations.

Description: This program is to convert the binary number to BCD i.e., we take one hexadecimal number in external memory and convert it into BCD. We are using A (lower byte) and R0 (higher byte) registers to store the result. These register contents are transferred to the external RAM address 1001H and 1002H respectively.

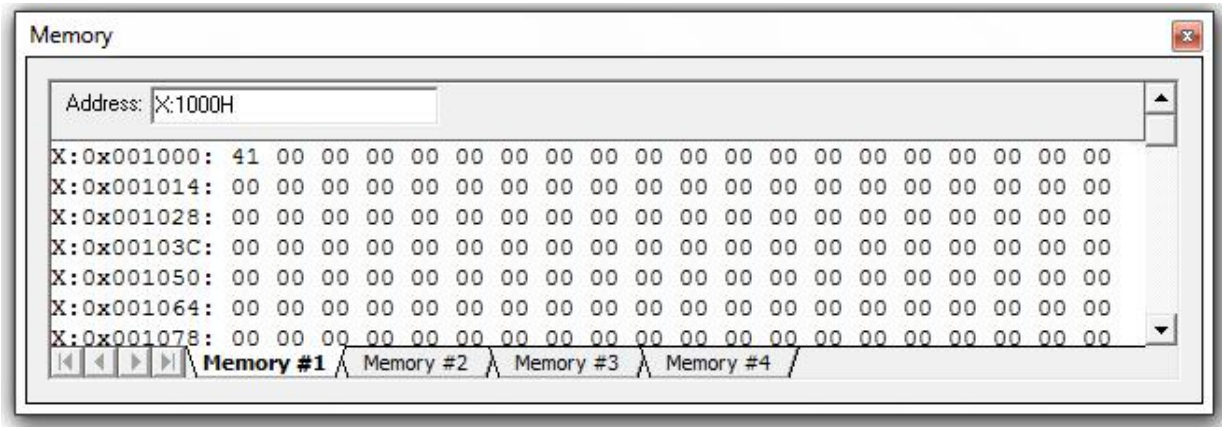
Program:

```
; TO CONVERT BINARY NUMBER INTO BCD

        MOV DPTR, #1000H
        MOVX A,@DPTR     ; fetch the input hexadecimal number from memory
        MOV B,#100
        DIV AB

        INC DPTR
        MOV R0,A

        MOV A,B
        MOV B,#10
        DIV AB

        SWAP A
        ORL A,B

        MOVX @DPTR, A     ;transfer lower byte of the result to memory

        INC DPTR
        MOV A,R0
        MOVX @DPTR,A   ;transfer higher byte of the result to memory

HERE: SJMP HERE
        END
```

SAMPLE OUTPUT:

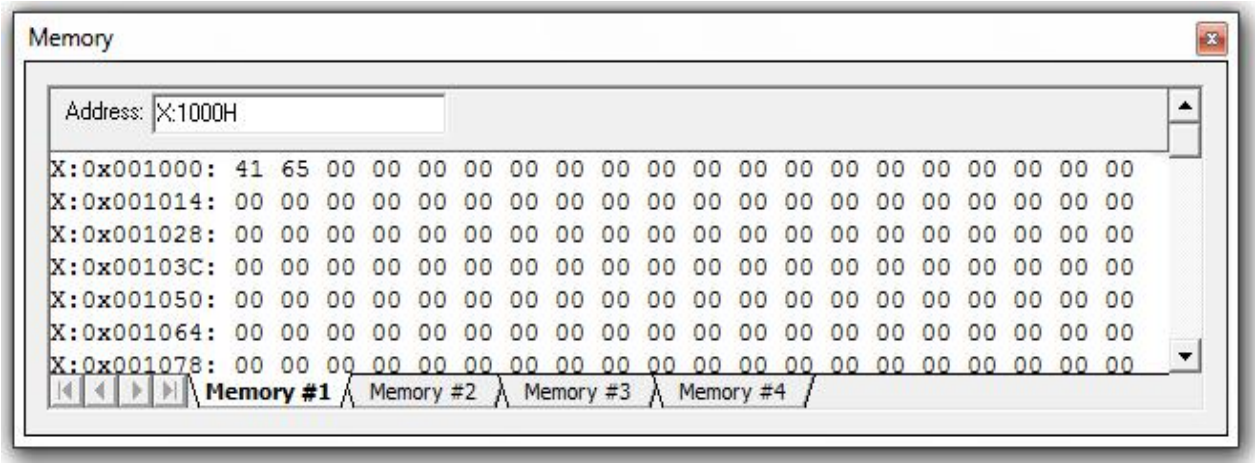
Before Execution:

The binary number 41H is stored at external RAM address 1000H.



After Execution:

The Resultant BCD number 65 is stored at external RAM address 1001H.



PROGRAM – 7: Write an ALP to compute the GCD and LCM of two 8-bit numbers stored at RAM locations 1000H and 1001H and store the result in the next consecutive locations.

Description: This program is to compute the GCD and LCM of the two numbers stored in the external memory. Here, we have used Euclid's algorithm to compute the GCD. LCM is computed by multiplying the two input numbers and dividing it by the GCD. The resultant GCD and LCM are stored in external RAM in the next consecutive memory locations after the input data.

Program:

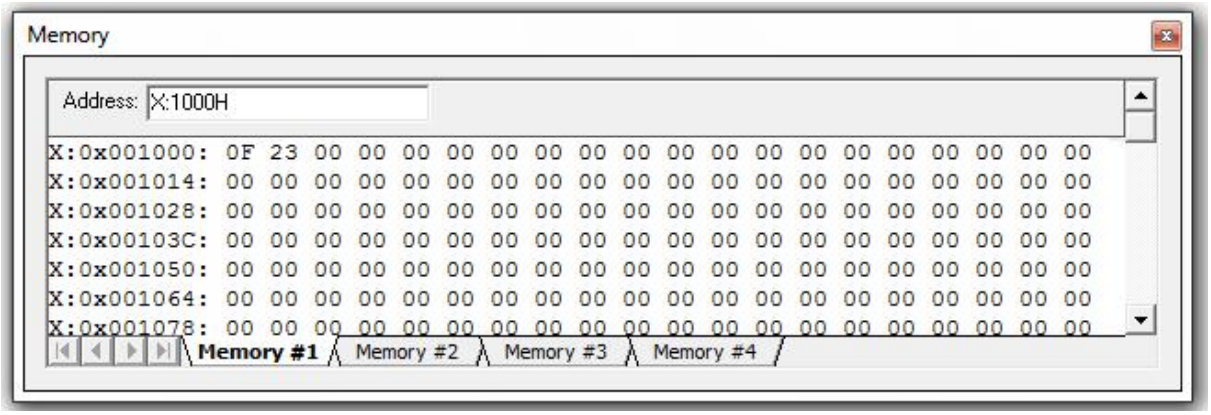
```
;TO FIND GCD AND LCM OF TWO 8BIT NOS

            MOV DPTR,#1000H
              MOVX A,@DPTR    ; fetch the first number from memory
              MOV R0,A

              INC DPTR
              MOVX A,@DPTR     ; fetch the second number from memory
              MOV R1,A

              MOV B,A
              MOV A,R0

AGAIN:        MOV R2,B
              DIV AB

              MOV A,R2
              MOV R3,B
              CJNE R3,#00,AGAIN

             INC DPTR
              MOVX @DPTR,A   ; store the GCD in memory

              MOV A,R0
              MOV B,R2
              DIV AB

              MOV B,R1
              MUL AB

              INC DPTR
              MOVX @DPTR,A    ; store the lower byte of LCM in memory
```

```
        MOV A,0F0H
        INC DPTR
        MOVX @DPTR,A    ; store the higher byte of the LCM in memory

HERE: SJMP HERE
  END
```

SAMPLE OUTPUT:

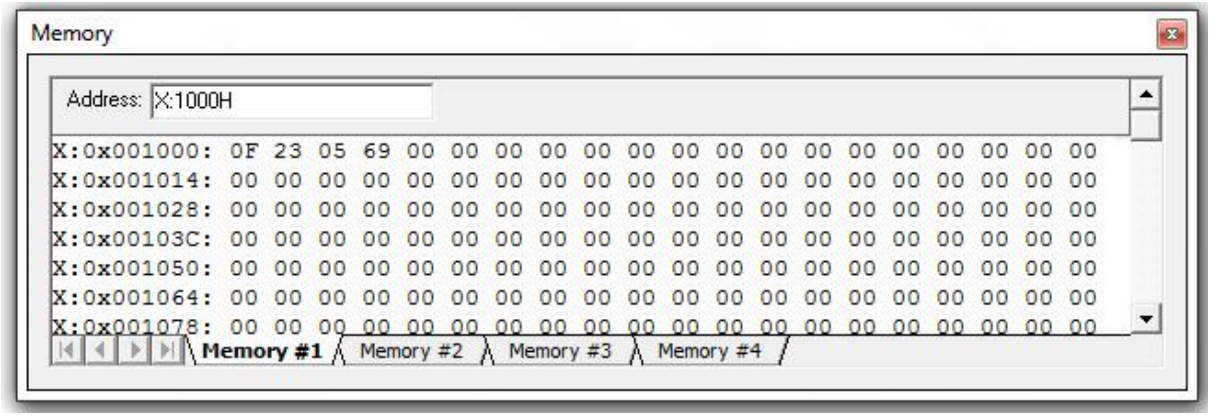
Before Execution:

The two input data bytes 15 (0FH) and 35 (23H) are stored at external RAM address 1000H and 1001H respectively.



After Execution:

The Resultant GCD, 05 and LCM, 105 (69H) are stored at external RAM address 1002H and 1003H respectively.



PROGRAM-8: Write an ALP to simulate BCD up counter.

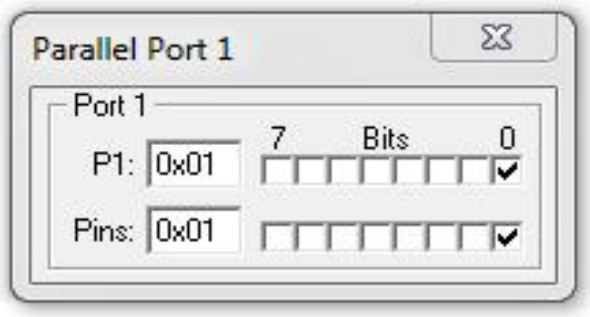
Description: This program counts the BCD number from 00 to 99. The A register is loaded with the initial value 0. This value is sent to port P1. Each time, a value 1 is added to the A register and DA A instruction is used so that the data in A register is in correct BCD form. This BCD number is sent to port P1. We introduce suitable delay between each data so that counter can be simulated. The process is repeated infinitely to count from 00 to 99 BCD.

Program:

```
;SIMULATION OF BCD UP COUNTER

          CLR A
BACK:     MOV P1,A          ;send data to port P1
          ACALL DELAY
          ADD A,#01
          DA A              ;decimal adjust the content of A after addition
          SJMP BACK

DELAY:    MOV R3,#03FH
LOOP3:    MOV R2,#0FFH
LOOP2:    MOV R1,#0FFH
LOOP1:    DJNZ R1,LOOP1
          DJNZ R2,LOOP2
          DJNZ R3,LOOP3

          RET
          END
```

SAMPLE OUTPUT:

The BCD up counter is simulated on PORT P1 which counts from 00-99.



PROGRAM – 9: Write an ALP to arrange the 'N' 8-bit numbers stored starting at RAM address 2000H in ascending order using Bubble Sort technique.

Description: This program is to sort the N 8-bit numbers stored in memory in ascending order using Bubble sort technique. Here, we have considered 4 8-bit numbers in the memory. The resultant numbers after sorting can be seen in the memory.

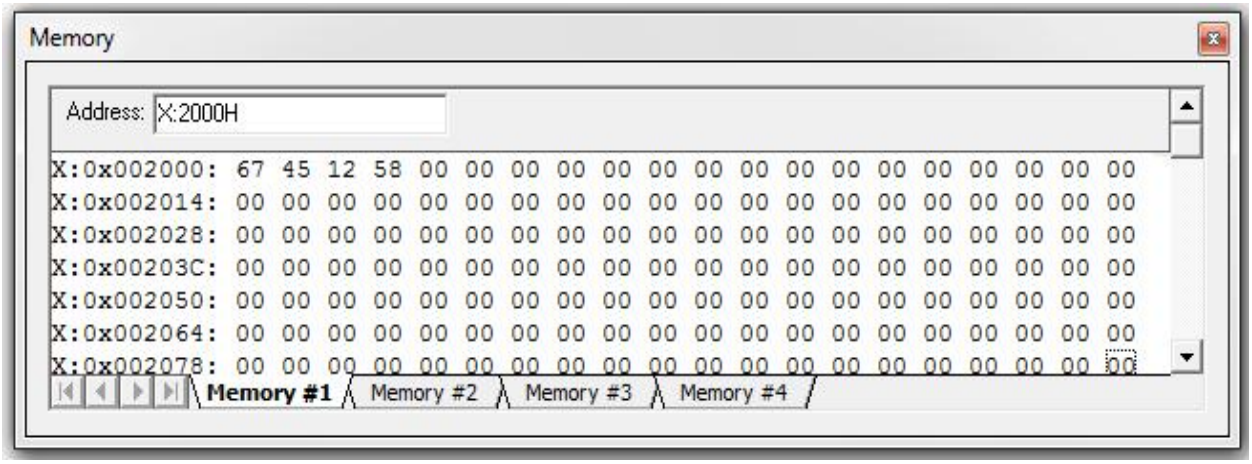
Program:

```
;TO ARRANGE 'N' 8-BIT NUMBERS IN ASCENDING ORDER
                MOV R0,#4
                DEC R0          ; Total number of passes
NXT_PASS:     MOV DPTR,#2000H
                MOV R1,00H      ; Total number of comparisons in each pass

 NXT_CMP:  MOV R2,DPL
                MOVX A,@DPTR
                MOV 0F0H,A

                INC DPTR
                MOVX A,@DPTR
                CJNE A, 0F0H,NOT_EQ
                SJMP NO_SWAP
  NOT_EQ: JNC NO_SWAP

                MOV DPL,R2
                MOVX @DPTR,A
                INC DPTR
                MOV A,0F0H
                MOVX @DPTR,A

 NO_SWAP: DJNZ R1,NXT_CMP
              DJNZ R0,NXT_PASS

HERE:SJMP HERE
                END
```

SAMPLE OUTPUT:

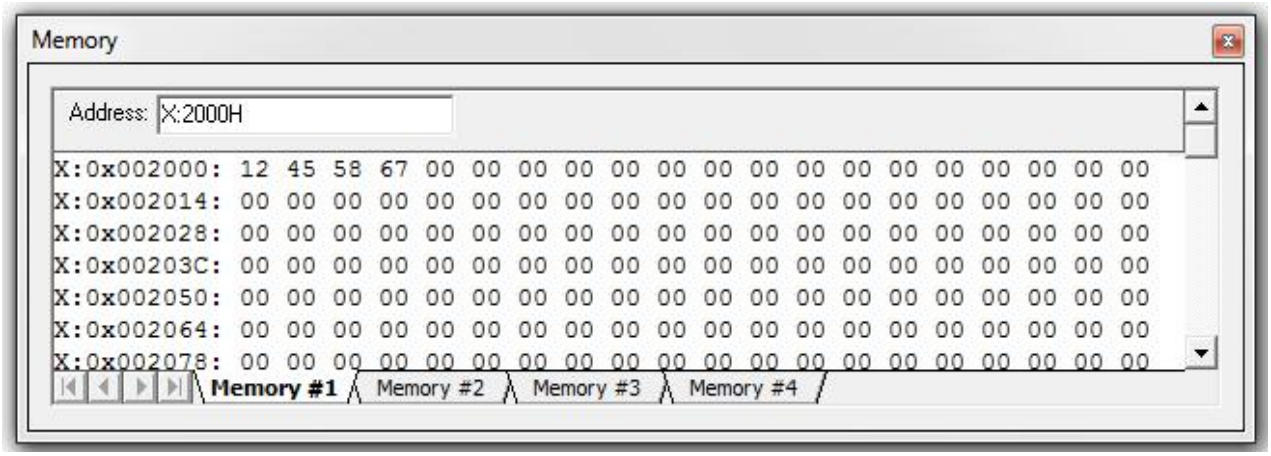
Before Execution:

The following four 8-bit numbers are stored starting at address 2000H:



After Execution:

The four 8-bit numbers after sorting are shown below:



PROGRAM – 10: Write an ALP to multiply an 8-bit number stored at RAM location 1000H with a 16-bit number stored at RAM locations 1001H and 1002H. Store the result in the next consecutive locations.

Description: This program is to multiply an 8-bit number with a 16-bit number. The input numbers are stored in external memory. We first multiply the lower byte of the 16-bit number with 8-bit number and store the 16-bit result in R3 (lower byte) and R4 (higher byte). Then, we multiply the higher byte of the 16-bit number with 8-bit number and store the 16-bit result in R5 (lower byte) and R6 (higher byte). Then, we add the R4 and R5 contents and store the result in A register. The carry generated is added with R6 register content. The result held in R3 and A are stored in memory.

Program:

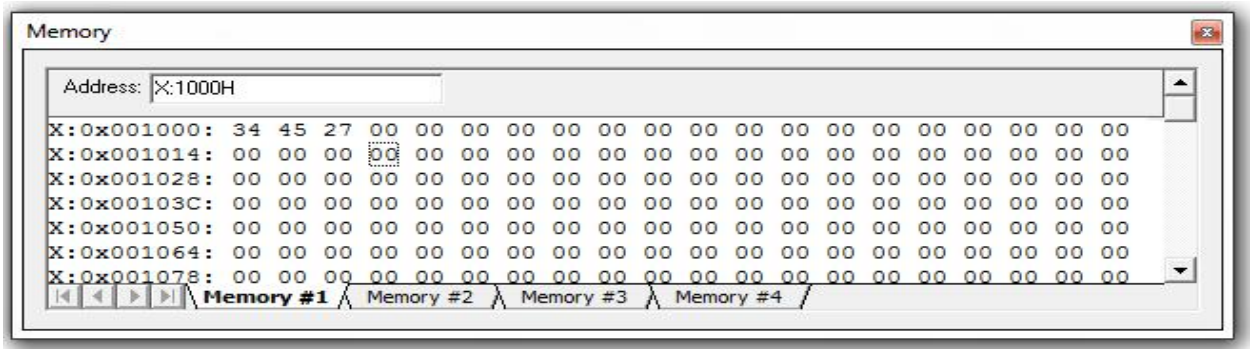
```
;MULTIPLICATION OF 16 BIT NUMBER WITH 8 BIT NUMBER

        MOV DPTR,#1000H
        MOVX A,@DPTR        ;Read the 8-bit data from external memory
        MOV R0,A

        INC DPL
        MOVX A,@DPTR        ;Read the lower byte of the 16-bit data from external memory
      MOV 0F0H,A
        INC DPL
        MOVX A,@DPTR        ;Read the higher byte of the 16-bit data from external memory
        MOV R1,A

        MOV A,R0
        MUL AB              ;Multiply the 8-bit data with the lower byte of 16-bit data
        MOV R3,A

        MOV R4,0F0H

        MOV B,R1
        MOV A,R0
        MUL AB              ;Multiply the 8-bit data with the higher byte of 16-bit data

        MOV R5,A
        MOV R6,B

        INC DPL

        MOV A,R3
```

```
        MOVX @DPTR,A
        MOV A,R4
        CLR C
        INC DPL
        ADD A,R5
        MOVX @DPTR,A
       CLR A
       ADDC A,R6

      INC DPL
       MOVX @DPTR,A
HERE:SJMP HERE
  END
```

SAMPLE OUTPUT:

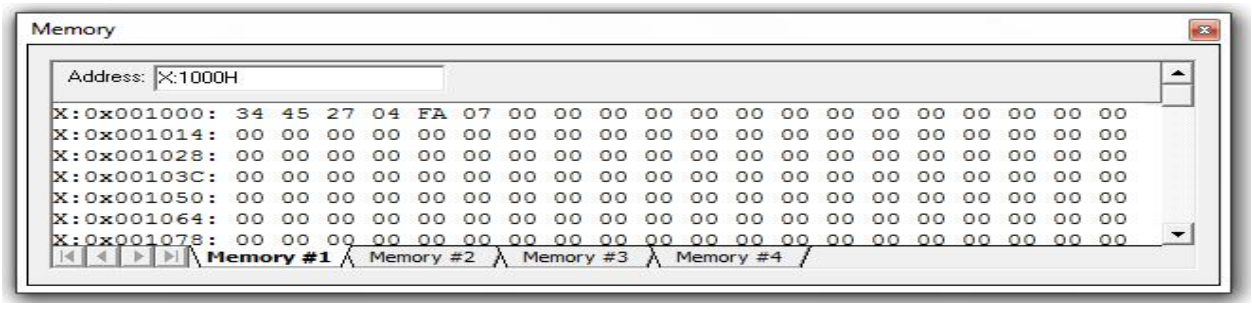
Before Execution:

The 8 bit data 34H is stored at external RAM address 1000H and 16 bit data 2745H is stored in Little Endian format at address 1001H.



After Execution:

The result of multiplication 7FA04H is stored starting at external RAM address 1002H in Little Endian Format.



EXERCISE FOR PART - A

Write an 8051 Assembly language program for the following:

1. Make the lower nibble of R2 the complement of the higher nibble of R3.
2. Exchange the lower nibbles of registers R4 and R5.
3. Double the unsigned number in register R2 and put the result in R3 (MS byte) and R4 (LS byte) using RLC instruction.
4. Random unsigned numbers are placed in registers R0 to R4. Find the largest number and put it in R6.
5. Double the unsigned number in register R2 and put the result in R3 (MS byte) and R4 (LS byte) using MUL instruction.
6. Swap the nibbles of R0 and R1 so that the lower nibble of R0 swaps with the higher nibble of R1 and higher nibble of R0 swaps with the lower nibble of R1.
7. Double the unsigned number in register R2 and put the result in R3 (MS byte) and R4 (LS byte) using ADD instruction.
8. Rotate DPTR one place to the left so that bit15 becomes bit0 and so on.
9. Treat registers R0 and R1 as 16-bit registers, and rotate them one place to the right so that bit0 of R1 becomes bit7 of R0, bit0 of R0 becomes bit7 of R1 and so on.
10. Decrement DPTR from any initialized value to 0025H.
11. Use R4 (LSB) and R5 (MSB) as a single 16-bit counter and decrement the pair until they become equal to 0000H.
12. Set every third byte in internal RAM from address 20H to 7FH to 1.
13. Put the address of each internal RAM location from address 30H to 70H as their content. (Put 30H as data in location 30H, 31H as data in location 31H and so on).
14. Count the number of bytes in external RAM locations 2000H to 3000H that are greater than the random unsigned number in R2 and less than the random unsigned number in R3. Use registers R5 (LSB) and R6 (MSB) to hold the count.

PART - B

INTERFACING MODULES USED:

- **All-in-one I/O interface board**

The Different Interfacing Modules Integrated in this All-in-one I/O Interface Board is.....

1) Logic Controller (8 input switches, 8 output LEDs)

2) Seven Segment Display Module (4 Digits, Implemented using Shift Registers)

3) LCD Interface (2x16 lines)

4) Stepper Motor Interface

5) DAC Interface

6) ADC Interface

7) Temperature Sensor Interface

8) AC Gadget Interface

9) Industrial Sensors Input Interface

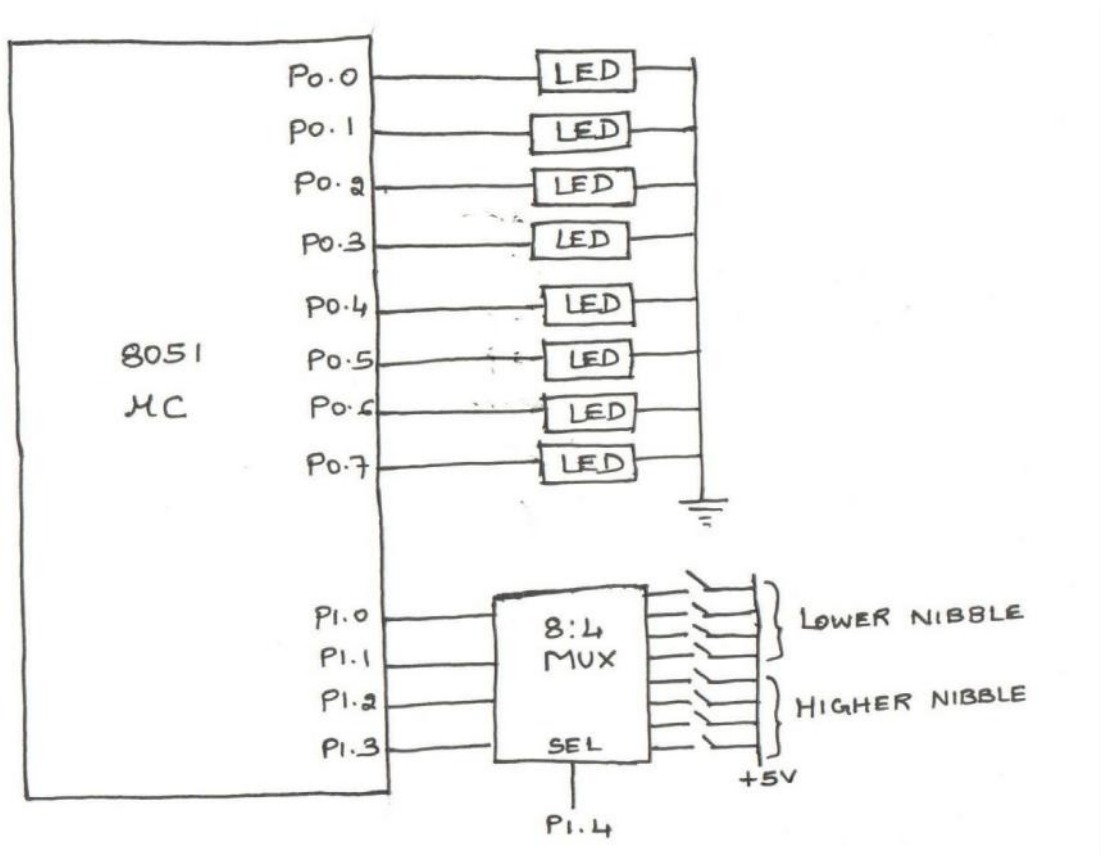
10) Elevator Interface

11) Keyboard Interface

- **8051 Microcontroller Board**

The 8051 microcontroller board is connected to the CPU through a USB to UART bridge. The I/O interface board is connected to this 8051 microcontroller board. Using Keil IDE, the hex file is generated and using Flash Magic tool, the hex file is dumped into ROM of the 8051 Microcontroller so that we can work with the I/O interfacing programs.

LOGIC CONTROLLER INTERFACE



For Logic Controller Interface, port P1 of 8051 is used as an input port and port P0 is used as an output port. Port P0 pins of 8051 are connected to the output LEDs of the logic controller. If FFH is sent on port P0 pins, then all the output LEDs will glow to indicate ON state. If 00H is sent on P0 pins then all the eight LEDs will be in the OFF state.

An 8-bit data cannot be read at a time from the logic controller being used. Hence, the data is read nibble by nibble. The switches used to set the input in the logic controller are connected to 8:4 MUX with control pin, SEL connected to port P1 pin, P1.4.The toggle switches are associated with LEDs to indicate the state of the switches. When the switch is opened, the LED is turned OFF and when the switch is closed, LED is turned ON. The output of MUX is connected to the lower order 4 bits of port P1. When the SEL pin goes low, 8051 reads the lower byte of the input data through port P1. When SEL pin goes high, 8051 reads the higher byte of the input data.

PROGRAM – 1: Write an 8051 C program to design a counter for counting the pulses of an input signal fed through pin P3.4. Display each count on the logic controller interface. (Use Counter 0 in mode2. The Count must be specified by the examiner).

```
#include<reg51.h>
void main()
{
        T1=1;
        TMOD=0x06; //Counter 0 in Mode2
        TH0=0X00;
        TL0=0x00;

        while(1)
       {
            do
           {
               TR0=1;   //start the counter
               P0=TL0;// Send the count to Port P0
           }while(TF0==0);
          TR0=0;
          TF0=0;
      }
}
```

PROGRAM – 2: Write an 8051 C program to read the status of 8 input bits from the Logic Controller Interface and display 'FF' if it is even parity bits otherwise display 00. Also display number of 1's in the input data.

```
#include<reg51.h>
typedef unsigned char tbyte;
typedef unsigned int tword;
sbit SEL=P1^4;
tbyte countOnes(tbyte x)
{
       tbyte i,count=0;
       for(i=0;i<8;i++)
     {
            if(x&(0x01<<i))
                count++;
      }
      return count;
}

tbyte readInput(void)
{
    tbyte temp=0;
    SEL=0;
     temp=P1 & 0x0f; //Read lower nibble of the input data
     SEL=1;
      temp=(P1 & 0x0f)<<4 |temp;  //Read higher nibble of the input data
```

```
    return temp;
}
void delayMs(tword x)
{ tbyte i;
  while(x--)
    for(i=0;i<200;i++);
}

void main(void)
{
      tbyte temp,count;
        while(1)
        {
            temp=readInput();
            count=countOnes(temp);
            if(count%2==0)
                P0=0xff;
              else
                P0=0x00;
              delayMS(1000);
              P0=count;
              delayMS(1000);
        }
}
```

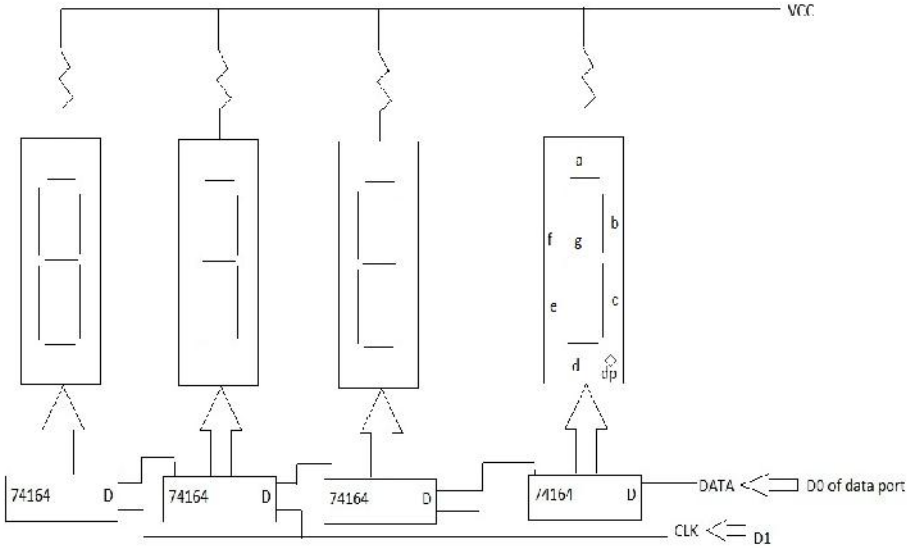
PROGRAM – 3: Write an 8051 C program to compute x * y using Logic Controller Interface.

```
#include<reg51.h>
typedef unsigned char tbyte;
typedef unsigned int tword;
sbit key1=P3^2;
sbit key2=P3^3;
sbit key3=P3^4;
sbit SEL=P1^4;
tbyte readInput(void)
{
        tbyte temp;
        SEL=0;
        temp=P1 & 0x0f;
        SEL=1;
        temp=(P1& 0x0f)<<4|temp;
        return temp;
}
void delayMS(tword x)
{
        tbyte i;
        while(x--)
        for(i=0;i<200;i++);
}
```

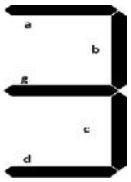
```
int main(void)
{
        tbyte a=0,b=0;
        tword c=0;
        while(1)
        {
                if(!key1)
                 {
                        a=readInput(); //Read the first 8-bit number
                        P0=a;
                        delayMS(200);
                        P0=0x00;
                        delayMS(200);
                 }
                if(!key2)
                 {
                        b=readInput(); //Read the second 8-bit number
                        P0=b;
                        delayMS(200);
                        P0=0x00;
                        delayMS(200);
                 }
                if(!key3)
                 {
                         c=a * b; //Compute the product
```

```
                    P0=c & 0xff;
                    delayMS(500);
                    P0=c>>8;
                    delayMS(1000);
                }
            }
}
```

SEVEN SEGMENT DISPLAY INTERFACE



The above figure shows a 4-digit seven segment display interface. It is of common anode type. i.e., to make a corresponding segment glow (ON), bit value 0 is to be sent and to turn off any particular segment, bit value 1 need to be sent. Below table shows the construction of seven segment code to display 3 on the seven segment display interface.



h	g	f	e	d	c	b	a
1	0	1	1	0	0	0	0

This is B0 in hexadecimal. A Serial In Parallel Out shift register is used to send the 8 bits of data to the Seven segment display. To send the seven segment code B0H, start sending the bits from MSB onwards i.e., D7 first, D6 next and so on with D0 being the last. The data bits are sent through Port P0.0 pin. Clock pulses are required to clock in the data, 8 clock pulses are required to display one byte of data. As the shift registers are cascaded, 8*4=32 clocks are required to clock in 4 bytes of data. To send “1234”, first we have to send seven segment code of ‘1’, then ‘2’,‘3’ and lastly ‘4’. Common clock is applied through Port P0.1 pin.

PROGRAM – 4: Write an 8051 C program to display the messages LIFE and HELP alternately on a 4-digit seven-segment display Interface.

```
#include <reg51.h>

typedef unsigned char tbyte;

sbit DAT=P0^0;

sbit CLK=P0^1;

void writeSeg(tbyte x)

{    tbyte i;

        for(i=0;i<8;i++)

        {

           if(x &(0x80>>i))

                DAT=1;

        else

                DAT=0;

        CLK=0;

        CLK=1;

    }

}

void delayMS(tword x)

{

        tbyte i;

        while(x--)

        for(i=0;i<200;i++);

}
```

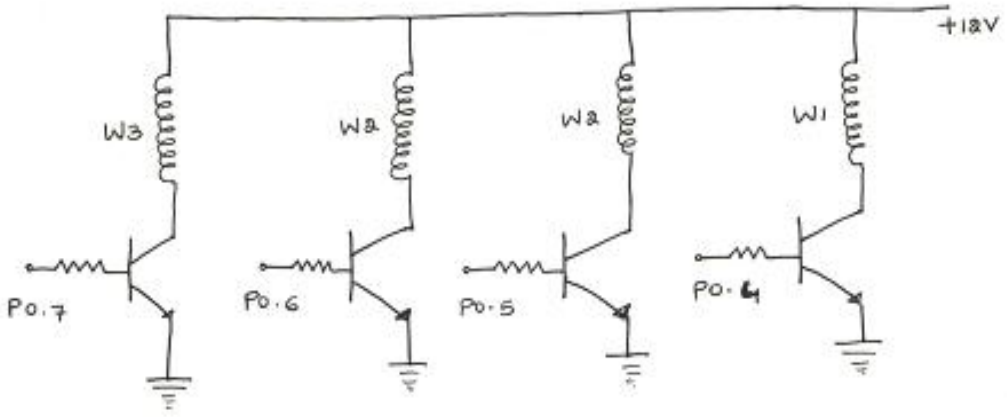
```
int main(void)
{
        tbyte LIFE[4]={0XC7,0xCF,0x8E,0x86};
        tbyte HELP[4]={0x89,0x86,0xC7,0x8C};
        tbyte i;
          while(1)
          {
               for(i=0;i<4;i++)
                    writeSeg(LIFE[i]);
               delayMS(200);
              for(i=0;i<4;i++)
                    writeSeg(HELP[i]);
                delayMS(200);
          }
}
```

PROGRAM – 5: Write an 8051 C program to display the message "dEPt OF CSE" from right-to-left and left-to-right on a 4-digit seven-segment display Interface.

```
#include <reg51.h>
typedef unsigned char tbyte;
typedef unsigned int tword;
sbit DAT=P0^0;
sbit CLK=P0^1;

void writeSeg(tbyte x)
{
        tbyte i;
          for(i=0;i<8;i++)
          {
             if(x &(0x80>>i))
                    DAT=1;
                            else
                   DAT=0;
          CLK=0;
          CLK=1;
}
}
void delayMS(tword x)
{
        tword i;
          while(x--)
```

```
        for(i=0;i<200;i++);
}

void main(void)
{
tbytemsg[]={0xff,0xff,0xff,0xff,0xA1,0x86,0x8c,0x87,0xff,0xc0,0x8e,0xff,0xc6,0x92,0x86,0xff,0xff,0xff,0xff};
char i,j;
while(1)
{
        for(i=0;i<16;i++)   /*right to left */
         {
      for(j=i;j<i+4;j++)
              writeSeg(msg[j]);
      delayMS(300);
       }
      for(i=14;i>=0;i--)    /*Left to right */
       {
      for(j=i;j<i+4;j++)
              writeSeg(msg[j]);
      delayMS(500);
       }
   }
}
```

STEPPER MOTOR INTERFACE



The operating voltage for stepper motor is +12V DC. The stepper motor has four stator windings. The stator windings or coils are connected to the Darlington pair transistors to energize each coil. The Darlington transistor base is connected to the port P0 (P0.4-P0.7). The step angle is 1.8^o i.e., 200 steps per revolution. The port P0 bits P0.7, P0.6, P0.5 and P0.4 are used to energize the four windings.

- Data pattern to be sent through Port P0 to rotate the motor is as follows:

For clockwise					**For anticlockwise**			
P0.7	P0.6	P0.5	P0.4	←windings→	P0.7	P0.6	P0.5	P0.4
1	0	0	0		0	0	0	1
0	1	0	0		0	0	1	0
0	0	1	0		0	1	0	0
0	0	0	1		1	0	0	0

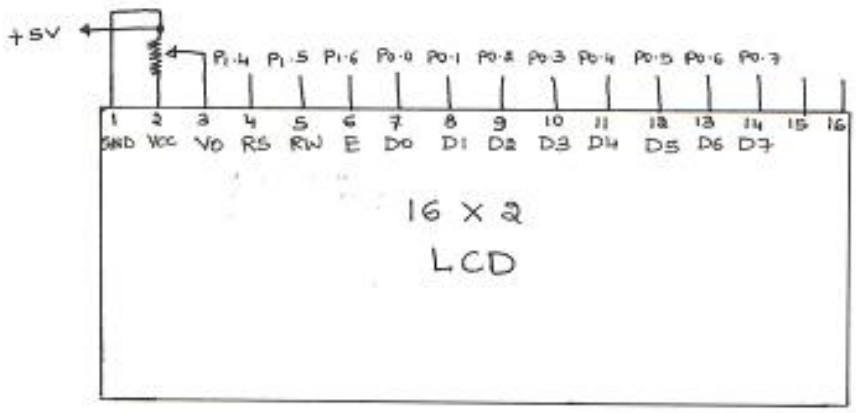
(The motor will move by 1 step for every pattern change)

PROGRAM – 6: Write an 8051 C program to drive a Stepper motor Interface to rotate the motor by N steps in clockwise direction and N steps in anti-clockwise direction. Introduce suitable delay between successive steps.

```
#include <reg51.h>
typedef unsigned char tbyte;
typedef unsigned int tword;

sbit W3=P0^7;
sbit W2=P0^6;
sbit W1=P0^5;
sbit W0=P0^4;

tbyte no_of_steps_clk=100;
tbyte no_of_steps_anticlk=100;
void delayMs(tword x)
{
        tword i;
        while(x--)
        for(i=0;i<200;i++);
}
void main(void)
{
          while(1)
       {
                W3=1;W2=0;W1=0;W0=0; delayMs(100);if(--no_of_steps_clk==0)break;
```

```
            W3=0;W2=1;W1=0;W0=0; delayMs(100);if(--no_of_steps_clk==0)break;
            W3=0;W2=0;W1=1;W0=0; delayMs(100);if(--no_of_steps_clk==0)break;
            W3=0;W2=0;W1=0;W0=1; delayMs(200);if(--no_of_steps_clk==0)break;
    }
   while(1)
  {
            W3=0;W2=0;W1=0;W0=1;delayMs(100);if(--no_of_steps_anticlk==0)break;
              W3=0;W2=0;W1=1;W0=0;delayMs(100);if(--no_of_steps_anticlk==0)break;
              W3=0;W2=1;W1=0;W0=0;delayMs(100);if(--no_of_steps_anticlk==0)break;
            W3=1;W2=0;W1=0;W0=0;delayMs(100);if(--no_of_steps_anticlk==0)break;
    }
}
```

LIQUID CRYSTAL DISPLAY (LCD) INTERFACE



LCD consists of Display Data RAM (DDRAM), CGROM (Character Generator ROM), shift registers, bit/pixel drivers, refreshing logics and LCD controllers. The data to be displayed on LCD is to be written on to the DDRAM using ASCII format. CGROM contains bit/pixel patterns for every character to be displayed (Preprogrammed).Shift registers are used to convert CGROM parallel data to serial data. Drivers are required to drive (ON/OFF) the bits. Refreshing logics are required to hold the display data, as the dots are displayed row by row basis continuously, like in CRT.

Whatever the data you write to LCD is of two types, either it is a command written to the instruction command code register of LCD (Configuration) or ASCII code of character to be displayed on LCD (DDRAM) which is written to the data register of LCD.

- The RS (Register Select) pin is used for selection of either the command register or the data register.
  - If RS = 0, then the instruction command code register is selected, allowing the user to send a command such as clear display etc.
  - If RS= 1, then the data register is selected, allowing the user to send data to be displayed on the LCD.

- R/W (Read/Write) pin allows the user to write information to the LCD or read information from it. R/W = 1 when reading; R/W=0 when writing.
- E (Enable) pin is used by the LCD to latch the information present at the data pins. When data is supplied to data pins, a high-to-low pulse must be applied to this pin in order for

the LCD to latch in the data present at the data pins. This pulse must be minimum of 450ns wide.

- D0-D7: These are 8-bit data pins used to send information to the LCD or read the contents of the LCD's internal registers.

Following two steps are to be followed to program the LCD:

1. Configure LCD by writing commands.
2. Write the actual string data, character by character by issuing DDRAM address command 0x80 for first line, 0xC0 for second line.

LCD Command codes:

Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning of 1st line
C0	Force cursor to beginning of 2nd line
38	2 lines and 5x7 matrix

List of LCD Instructions:

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DD RAM address 0 in address counter
Return Home	0	0	0	0	0	0	0	0	1	-	Sets DD RAM address 0 as address counter. Also returns display being shifted to original position. DD RAM contents remain unchanged.
Entry Mode Set	0	0	0	0	0	0	0	1	1/D	S	Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.
Display On/ Off Control	0	0	0	0	0	0	1	D	C	B	Sets On/Off of entire display (D), cursor On/Off (C), and blink of cursor position character (B).
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Moves cursor and shifts display with-out changing DD RAM contents.
Function Set	0	0	0	0	1	DL	N	F	-	-	Sets interface data length (DL), number of display lines (L), and character font (F).
Set CG RAM Address	0	0	0	1		AGC					Sets CG RAM address. CG RAM data is sent and received after this setting.
Set DD RAM Address	0	0	1			ADD					Sets DD RAM address. DD RAM data is sent and received after this setting.
Read Busy Flag & Address	0	1	BF			AC					Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.
Write Data CG or DD RAM	1	0				Write Data					Writes data into DD or CG RAM.
Read Data CG or DD RAM	1	1				Read Data					Read Data from DD or CG RAM.

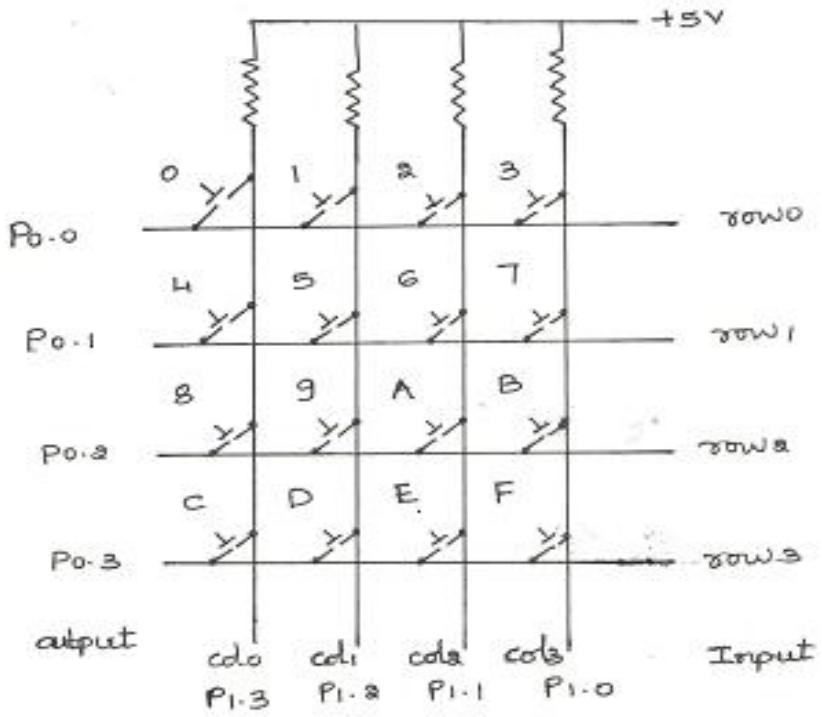
PROGRAM – 7: Write an 8051 C program to display the strings on a 2x16 character LCD Interface.

```
#include<reg51.h>
#include<intrins.h>
typedef unsigned char tbyte;
//name of the LCD pins
sbit RS=P1^4;//0-command 1-data
sbit RW=P1^5;//0-Write 1-read
sbit E=P1^6;//1 to 0.perform writing of command/data
 void delay(tbyte val)
{
        tbyte i;
          for(i=0;i<val;i++)
          {
                _nop_();
                _nop_();
                _nop_();
                _nop_();
                _nop_();
          }
}
 void enpulse(void)
{
       E=1;
      delay(2);
      E=0;
```

```
    delay(2);
}
 void LCD_Command(tbyte command)
{
        RS=0;
          RW=0;
          P0=command;
          enpulse();
          delay(50);
}

void LCD_Data(tbytedatabyte)
{
        RS=1;//data is written
        RW=0;
          P0=databyte;
          enpulse();
          delay(50);
}
void LCD_DispStr(tbyteline_no,char *str)
{
        tbyte i;
        if(line_no==1)
              LCD_Command(0x80);
           else
```

```
            LCD_Command(0xc0);
          for(i=0;str[i]!='\0';i++)
         {
            LCD_Data(str[i]);
            if(i==16)
               break;
         }
}

void LCD_Init(void)
{
     LCD_Command(0x38);//function set-2 line display,byte mode
     LCD_Command(0x0c);//display on
     LCD_Command(0x01); //clear the display
}
main()
{
        tbyte str1[]="WEDNESDAY";
          tbyte str2[]="21/03/2018";
          LCD_Init();
          LCD_DispStr(1,str1);
          LCD_DispStr(2,str2);
           while(1);
    }
```

KEYBOARD INTERFACE



The keyboard interface used is a 4x4 matrix as shown in above figure. Columns are connected to Port P1's lower order 4 pins of 8051 and rows are connected to Port P0's lower order 4 pins of 8051.

- If no key is pressed , we will have the bit status '1111' on input port pins P1.3-P1.0 (col0-col3) , as all the inputs are pulled up by pull up resistors.
- If any key is pressed, say key '0', it will short row0 and col0 lines, so whatever data(0 or 1) available at row0is available at col0. Since already columns are pulled high, it is required to apply logic '0' at row0 to see change in col0when the key is pressed.
- If we apply '0000' on all the rows 0-3 (P0.0 - P0.3 lines) and read corresponding columns 0-3 (P1.3-P1.0), if it is other than '1111', then it means some key is pressed, else no key is pressed. This process can be repeated in a loop indefinitely to check for the key press.
- To identify which key is pressed , after key press is detected
    - Check for a key press in first row by sending '0111'on rows (P0.0-P0.3) and reading the columns. If it is '1111'(P1.3- P1.0), then go to next row, else identify the column number.

- Check for a key press in second row by sending '1011'on rows (P0.0-P0.3) and reading the columns. If it is '1111'(P1.3- P1.0), then go to next row, else identify the column number.
- Check for a key press in third row by sending '1101'on rows (P0.0-P0.3) and reading the columns. If it is '1111'(P1.3-P1.0), then go to next row, else identify the column number.
- Check for a key press in last row by sending '1110'on rows (P0.0-P0.3) and reading the columns, if it is '1111'(P1.3-P1.0), then check for key press again, else identify the column position.

PROGRAM – 8: Write an 8051 C program to scan a 4 x 4 keypad for key closure and display the code of the key pressed on LCD.

```
#include<reg51.h>
#include<intrins.h>
typedef unsigned char tbyte;
typedef unsigned int tword;
sbit r0=P0^0;
sbit r1=P0^1;
sbit r2=P0^2;
sbit r3=P0^3;
sbit c0=P1^3;
sbit c1=P1^2;
sbit c2=P1^1;
sbit c3=P1^0;
sbitrs=P1^4;
sbitrw=P1^5;
```

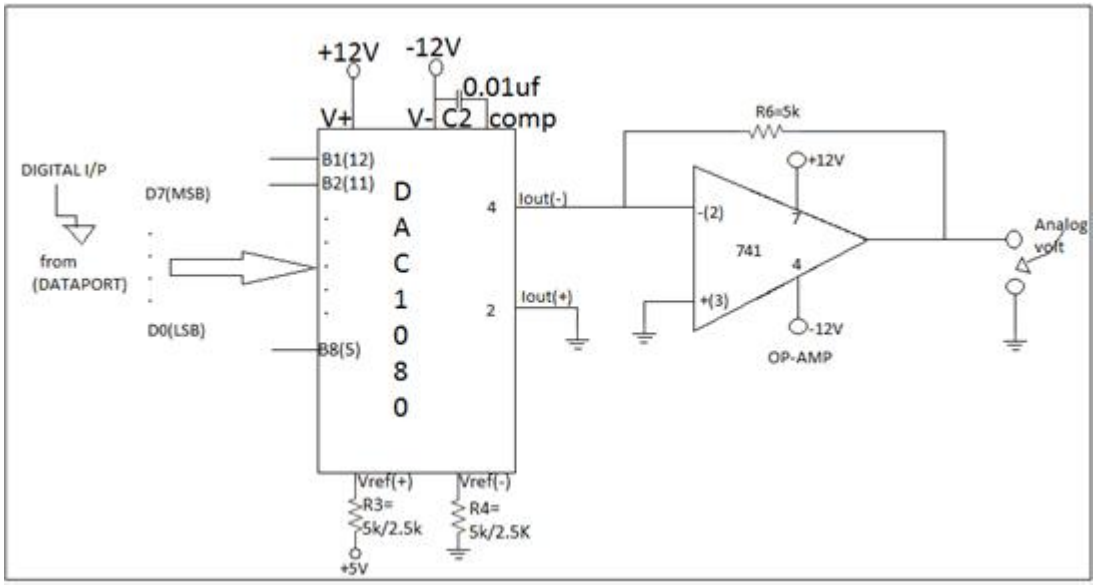
```
sbit e=P1^6;
tbyte key;
tbyte codekeys[4][4]={ {'0','1','2','3'},{'4','5','6','7'},{'8','9','A','B'},{'C','D','E','F'}};
void delay(tbyte val)
{
      tbyte i;
      for(i=0;i<val;i++)
   {
          _nop_();
         _nop_();
        _nop_();
        _nop_();
        _nop_();
     }
}

void lcd_init(void)
{
      lcd_com(0x38);
      lcd_com(0x0c);
      lcd_com(0x01);
}
void enpulse(void)
{
      e=1;
```

```
    delay(2);
     e=0;
     delay(2);
}

void lcd_com(tbyte command )
{
    rs=rw=0;
    P0=command;
    enpulse();
    delay(50);
}
void lcd_data(tbytedatabyte)
{
     rs=1;
    rw=0;
    P0=databyte;
    enpulse();
    delay(50);
}

  void main(void)
{
   tbyte rowp,colp,i;
    r0=r1=r2=r3=0;
```

```
    c0=c1=c2=c3=1;
    while(1)
{
      while(1)
   {
         r0=0,r1=1,r2=1,r3=1;
        rowp=0;
         if(c0==0){colp=0;break;}
         if(c1==0){colp=1;break;}
         if(c2==0){colp=2;break;}
         if(c3==0){colp=3;break;}
         r0=1,r1=0,r2=1,r3=1;
         rowp=1;
         if(c0==0){colp=0;break;}
        if(c1==0){colp=1;break;}
        if(c2==0){colp=2;break;}
        if(c3==0){colp=3;break;}
        r0=1,r1=1,r2=0,r3=1;
        rowp=2;
        if(c0==0){colp=0;break;}
        if(c1==0){colp=1;break;}
         if(c2==0){colp=2;break;}
         if(c3==0){colp=3;break;}
         r0=1,r1=1,r2=1,r3=0;
         rowp=3;
```

```
            if(c0==0){colp=0;break;}
            if(c1==0){colp=1;break;}
           if(c2==0){colp=2;break;}
           if(c3==0){colp=3;break;}
       }
      delay(200);
       key=codekeys[rowp][colp];
        while(c0==0||c1==0||c2==0||c3==0) ;
        delay(20);
        lcd_init();
       lcd_com(0x80);
       lcd_data(key);
    }
}
```

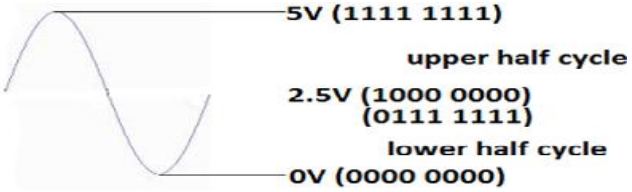
DIGITAL TO ANALOG CONVERTER (DAC) INTERFACE:



The DAC interface consists of DAC chip, OPAMP 741 and voltage converter. A reference voltage of +5V is generated using the voltage regulator and is fed to Vref point of DAC. The output voltage of +5V is obtained from DAC when the digital input is FFH and the output voltage will be 0V when the digital input is 00H. The output of DAC is fed to the operational amplifier to get the final output. The digital values 00H-FFH corresponds to the analog voltage 0-5V. The 8-bit data can be fed into DAC from Port P0 of 8051 microcontroller.

The ***digital formula*** used to get the values for positive cycle of the sine waveform is as follows:

$127 \sin\Theta + 127$



If 30 samplings are done to get the positive cycle of sine waveform then, we can get the values by using the digital formula for every 6º increment (180º/30=6º)i.e.,

127 sin 0º +127 =127

127 sin 6º +127=140 and so on

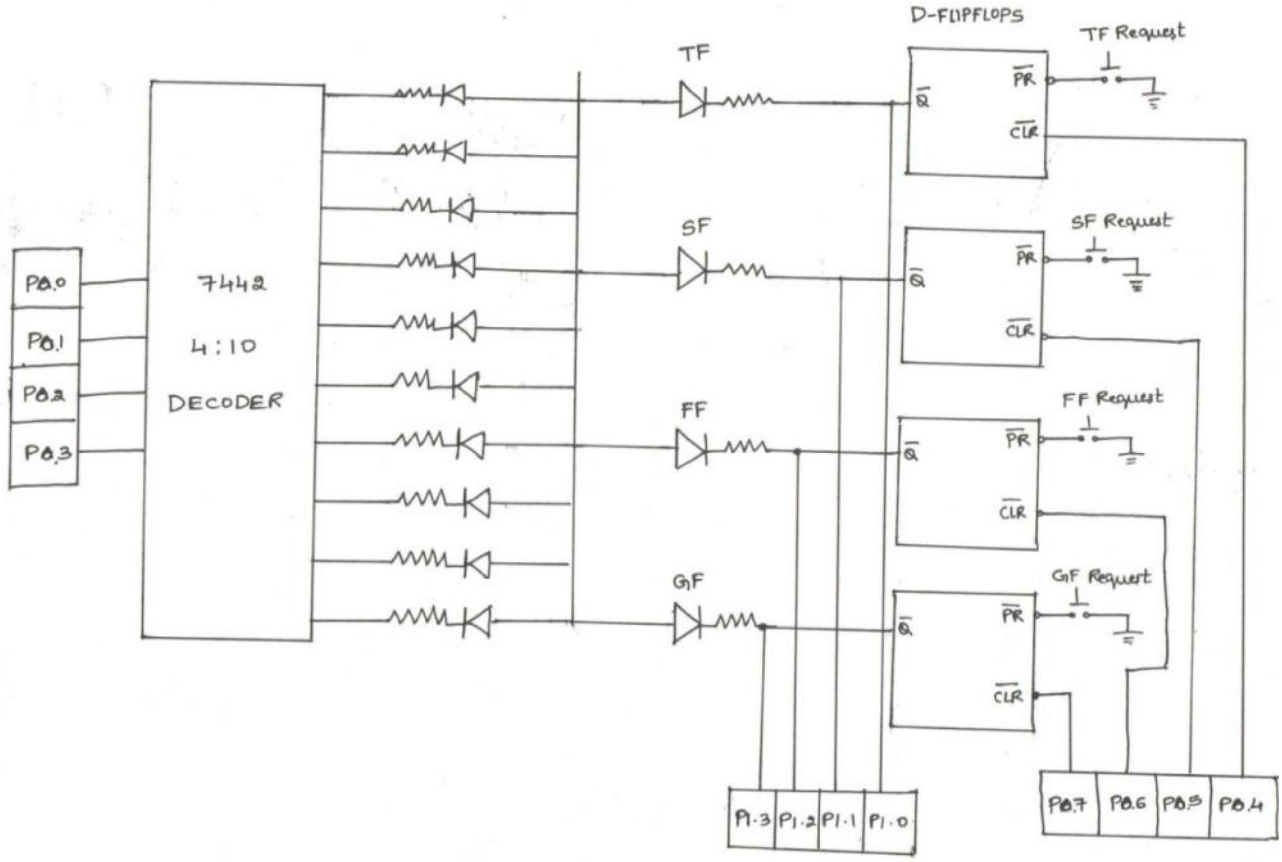
This can be continued till 90º and the values computed can be considered in reverse order for sampling values from 90º -180º.

PROGRAM – 9: Write an 8051 C program to generate Half Rectified Sine wave, Fully Rectified Sine and sine waveform using the DAC Interface. (The output of the DAC is to be displayed on the CRO).

```
#include<reg51.h>
typedef unsigned char tbyte;
sbit key1=P3^2;
sbit key2=P3^3;
sbit key3=P3^4;
void main(void)
{
tbyte sine[]={127, 140, 153, 166, 178, 190, 201, 211, 221, 229, 236, 243, 247, 251, 253, 255, 253, 251, 247, 243, 236, 229, 221, 211, 201, 190, 178, 166, 153, 140, 127};
tbyte i;
while(1)
{
      if(key1==0)  /*Full Rectified*/
      {
            while(1)
          {
               for(i=0;i<31;i++)
                   P0=sine[i];
         }
       }
      if(key2==0)  /*Half Rectified*/
     {
           while(1)
          {
               for(i=0;i<31;i++)
                   P0=sine[i];
              for(i=0;i<31;i++)
```

```
                P0=0x7F;
              }
          }
         if(key3==0)  /*Sine Waveform*/
        {
            while(1)
          {
               for(i=0;i<31;i++)
                  P0=sine[i];
              for(i=0;i<31;i++)
                  P0=~sine[i]+1;
           }
        }
   }
}
```

ELEVATOR INTERFACE



The Elevator interface has 4:10 decoder. The Output port P0 pins, P0.0-P0.3 of 8051 are used as input to the decoder. The output from the decoder is connected to 10 LEDs which indicate the motion of the elevator. The elevator motion is indicated by turning ON/OFF the successive LEDs one at a time in fixed time intervals. The LED's ON state indicates the current position of the elevator. The user can request for service to any of the four floors (Ground floor, first floor, second floor, third floor) by pressing the request keys. Each floor has an LED which gets turned ON when there is a request from the corresponding floor. This request is latched by D Flip-flops. The preset input of each D Flip-flop is connected to each of the request keys. The clear pin of each D Flip-flop is connected to the port P0 pins, P0.7-P0.4. The request status can be read through input Port P1 pins, P1.3-P1.0. To service the request from any of the floors, a 0 is sent to CLR pin of the corresponding floor's D Flip-flop through corresponding Port P0 pin. To indicate that the floor request is serviced, the corresponding floor LED is turned OFF. The elevator waits

at the ground floor till there is any request from any of the floors. If any request comes then the elevator moves to that floor and services that request. If any other request comes while moving up or while coming down then the elevator services it and returns back to the ground floor.

PROGRAM – 10: Write an 8051 C program to drive an elevator interface in the following way:

Initially the elevator should be in the ground floor, with all requests in OFF state. When a request is made from a floor to any other floor, the elevator should move up or move down to that requested floor, service the request and stay in that floor waiting for any new request.

```
#include<reg51.h>
typedef unsigned char tbyte;
typedef unsigned int tword;
 void delayMs(tword x)
 {
        tword i;
        while(x--)
             for(i=0;i<100;i++);
 }
void main(void)
{                    //          1    2          4                    8
       tbyte flr[]={0xff,0x09,0x06,0xff,0x03,0xff,0xff,0xff,0x00};
         tbyte fclr[]={0xff,0xe9,0xd6,0xff,0xb3,0xff,0xff,0xff,0x70};
         tbyte reqflr,curflr=0x08,i,j;
          P0=0x00;
          P0=0xf0;
```

```
while(1)
{           P1=0XFF;
          reqflr=P1 | 0xf0;

           while(reqflr==0xff)
                reqflr=P1 | 0xf0;

            reqflr=~reqflr;

            if(curflr>reqflr)
            {
                  i=flr[reqflr]-flr[curflr];
                    j=flr[curflr] + 1;
                    for(;i>0;i--)
                    {
                          P0=0xf0 | j;
                           j++;
                          delayMs(100);
                    }
            }
            else if(curflr<reqflr)
            {
                  i=flr[curflr]-flr[reqflr];
                    j=flr[curflr] - 1;
                    for(;i>0;i--)
```

```
                    {
                        P0=0xf0 | j;
                         j--;
                         delayMs(100);
                    }
                }

                curflr=reqflr;
                delayMs(100);
               P0=fclr[curflr];
        }
}
```

EXERCISE FOR PART-B

Write an 8051 Embedded C program for the following:

1. Read the status of 8 input bits from the Logic Controller Interface and display the complement of it.
2. Simulate BCD up-down counter using Logic Controller Interface.
3. Simulate Ring counter using Logic Controller Interface.
4. Display “CSE2017” on Seven Segment Display Interface in rolling fashion.
5. Read two numbers through keyboard interface and compute their sum.
6. Generate Square waveform using the DAC Interface.
7. Generate Rectangular waveform using the DAC Interface.
8. Generate Ramp waveform using the DAC Interface.
9. Generate Triangular waveform using DAC Interface.
10. Generate Staircase waveform using the DAC Interface.

VIVA VOCE QUESTION BANK

1. What is a microcontroller?
2. How a microcontroller differs from a microprocessor?
3. 8051 is how many bit microcontroller?
4. What does 8-bit signify in an 8-bit microcontroller?
5. Which is the program memory in 8051?
6. Which is the data memory in 8051?
7. What is the size of internal RAM in 8051?
8. What is the on-chip ROM size of 8051?
9. Which is the ROMless version of 8051?
10. Which are the 16-bit registers in 8051?
11. Name the only registers in 8051 that are used to perform multiplication and division operations.
12. How to access the DPTR register of 8051 individually as two 8-bit registers?
13. Name the register in 8051 that is not having any address.
14. What is the program memory type in 8751 microcontroller?
15. Which register is used to hold the address of external data memory?
16. What is the width of 8051 data bus and address bus?
17. Which register is used as the destination operand in all arithmetic instructions?
18. Which register is used to point to the next instruction in the program memory of 8051?
19. In which memory of 8051, the Stack is implemented?
20. Which register is used to point to the top of the stack in 8051 and what is its size?
21. In which order of memory address, the stack grows in 8051?
22. How many register banks are there in 8051 and how many registers each bank contain?
23. Which register bits need to be updated to select a particular register bank in 8051?
24. Which register bank is selected by default in 8051?
25. How many pin IC is this 8051 microcontroller?

26. How many parallel ports are there in 8051 and how many bits each?
27. Which port of 8051 is dedicated for only I/O operations?
28. Which port of 8051 is used as data bus when external memory is used?
29. Which port of 8051 is used for interrupt input, timer/counter input, and serial I/O?
30. In which memory area, SFRs of 8051are implemented and what is its address range?
31. What is the address range of on-chip RAM in 8051?
32. How many external interrupt and internal interrupts are there in 8051?
33. What is the address range of the bit-addressable memory in internal RAM of 8051?
34. What is the size of the scratch-pad memory of 8051 and what is its address range?
35. Which is the flag register in 8051and what is its size?
36. How many bytes in internal RAM of 8051 are used for register bank?
37. What is the number of registers in each register bank and how they are numbered?
38. How to access the working registers of 8051?
39. How many bytes in internal RAM of 8051 are used as bit-addressable memory?
40. What RAM locations are used for register R0-R7 in register bank 0 of 8051?
41. What RAM locations are used for register R0-R7 in register bank 1 of 8051?
42. What RAM locations are used for register R0-R7 in register bank 2 of 8051?
43. What RAM locations are used for register R0-R7 in register bank 3 of 8051?
44. When $\overline{EA}$ is connected to ground, from which memory the code byte is fetched?
45. What is the On-chip ROM address space of 8051?
46. How many I/O pins are there in 8051?
47. With what value the Stack pointer is initialized to after 8051 is reset?
48. How many interrupt sources 8051 contains?
49. Which bits in the PSW register selects a register bank?
50. Which ports serve as low order and high order address bus for external memory access?
51. How many bytes are allocated for SFR in on-chip data memory?

52. If the bit address in the internal RAM of 8051 is 1EH then what is its byte address?
53. Write the instruction to set the 4^{th} bit at byte address 24H in the internal RAM.
54. Write the set of instructions to store the data 45H to the register R5 in the register bank2.
55. List the instructions that can be used to clear the contents of a register.
56. Which instruction can be used to exchange the nibbles of an 8-bit data?
57. Which instruction can be used to mask certain bits in an 8-bit data?
58. Which instruction can be used to set certain bits in an 8-bit data?
59. Differentiate between "MOVX" and "MOVC" instructions.
60. What is the use of RS and E pins of Liquid Crystal Display?
61. What is the size of external RAM that 8051 can address?
62. Which instruction can be used to invert certain bits in an 8-bit data?
63. What is maximum machine cycle taken to execute an instruction in 8051?
64. What are addressing modes?
65. List the addressing modes supported in 8051.
66. Identify the addressing mode of source and destination operands in the following instructions:

    a) MOV A,#46H b) XRL 37H,A c) MOVX @DPTR,A
67. Which is the only addressing mode allowed in the instructions to perform push and pop stack operations?
68. What type of CPU architecture 8051 contains?
69. When the Auxiliary carry flag is set?
70. When the Overflow flag is set?
71. When the Carry flag is set?
72. When the Parity flag is set?
73. In AT89C51, what does C and 51 signify?
74. Which instructions take maximum machine cycles to execute?
75. What is the length of the following instructions?

    a) MOV 67H,R0
    b) ADD A,R1

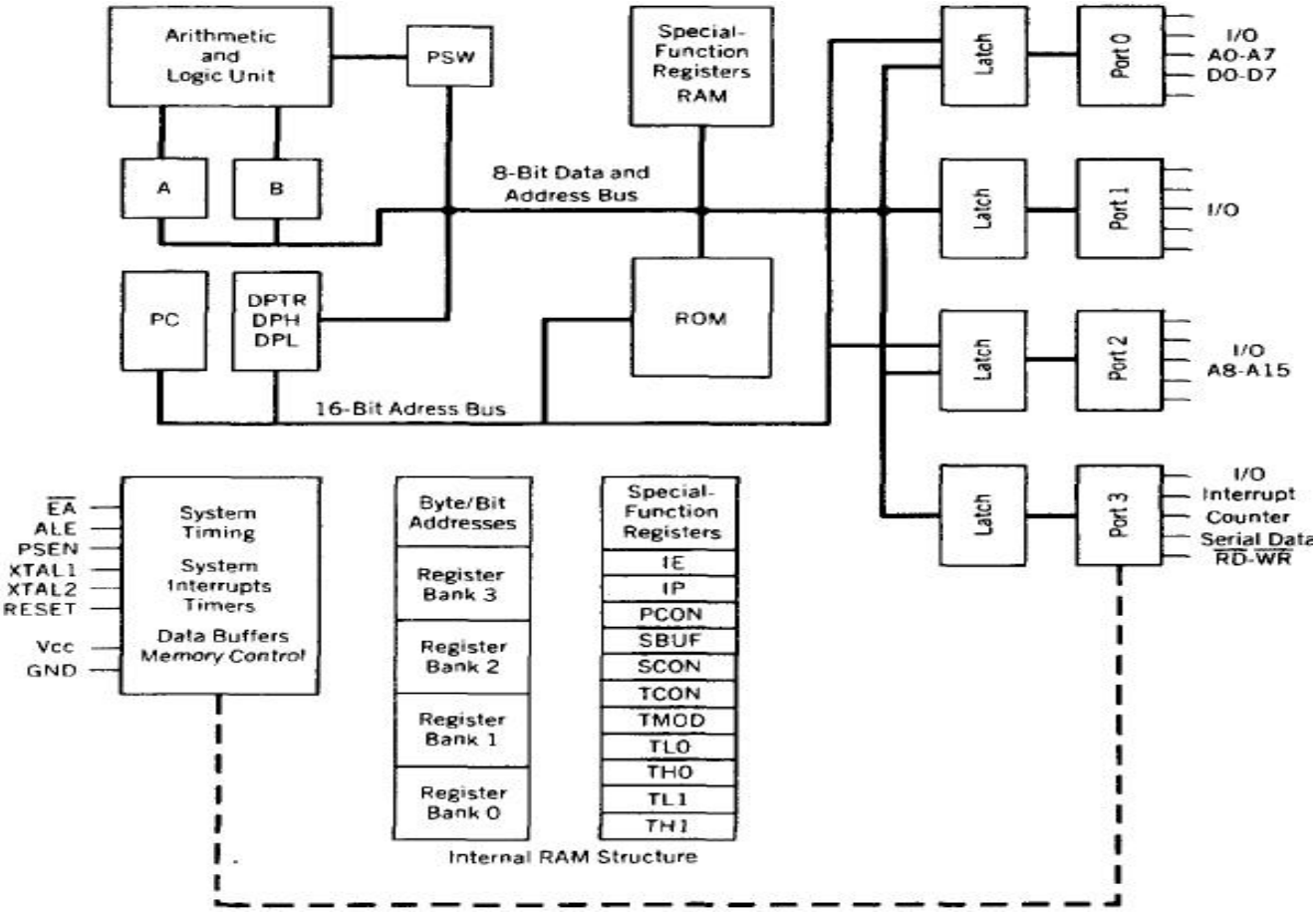
c) ORL 34H,#82H
d) CJNE A,#45H,NEXT
e) MOV DPTR,#9000H

76. What are the differences between RISC and CISC?
77. Which register is mainly used to receive the extended result from multiply and divide operations?
78. Direct addressing mode can be used to access which memory?
79. What is the time taken by the **5 MHz** microcontroller to execute **4 cycle** instruction?
80. The PC in 8051 is associated with which memory?
81. Show how to save the status of P2.7 in RAM bit location 31.
82. To ensure the integrity in which memory, the checksum byte method is used?
83. What is the memory addressing capability of a microprocessor having 20-bit address bus?
84. In 8051, program instructions may require how many machine cycles to execute?
85. How many timers/counters 8051 has and how many bits each?
86. In the instruction JZ Next, which register content is checked for zero?
87. Which register is used to hold the data for serial transmission?
88. Identify the mistakes in the following instructions:
    a) MOV R1,R2 b) MOV A,@R2 c) PUSH R6
89. What is the Command word used to select first line in a 2x16 character LCD?
90. How many address lines are required to access 4Kbytes of memory?
91. Show how you would check whether the P flag is high.
92. What is the function of ALE signal?
93. What are the functions of TxD and RxD?
94. If (A)=F6H and (B)=FEH, then what will be stored in B register after the execution of MUL AB instruction?
95. When the rollover take place in mode1 counter?
96. Who provides the clock pulses to 8051 timers if C/T=1?
97. What is the function of a TMOD register?
98. In which register do we find the timer start bits and timer rollover flags?
99. Which mode of the timer is used to set the baud rate?
100. Which timer of 8051 is used to set the baud rate?

APPENDIX

THE 8051 MICROCONTROLLER

ARCHITECTURAL FEATURES



- 8051 is an 8-bit µc which comes under CISC processors and it uses Harvard architecture.
- 8051 is available as a 40-pin chip which works at +5V.
- It has 8-bit data bus and 16-bit address bus.
- It has 8-bit CPU with registers A (the Accumulator) and B of 8-bit each.
  - A register receives the result of all arithmetic operations. It is also used to hold the data during external memory access.
  - B register is mainly used to receive the extended result from multiply and divide operations.
- It has 8-bit Program Status Word (PSW) register.
  - The result of ALU operation is updated in PSW.

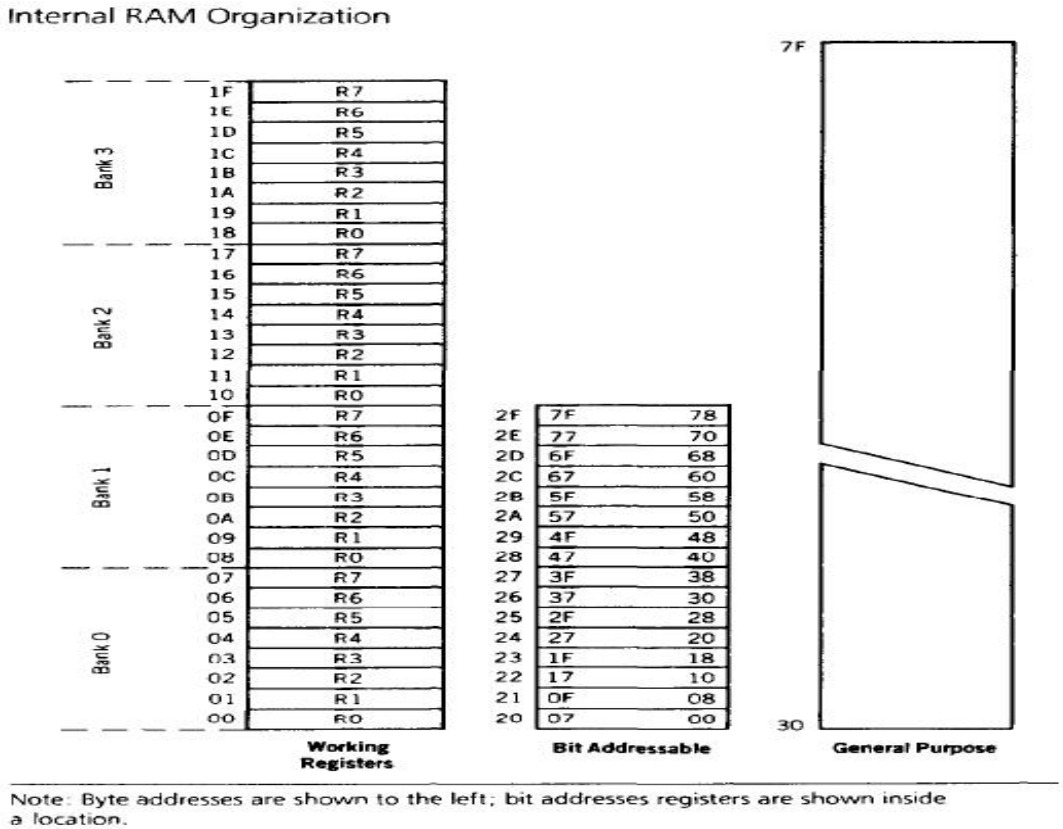
- It keeps track of overflow, carry, negative results etc.
- It also keeps track of which memory bank is currently selected.

- It has 16-bit program counter (PC) register.
  - It is used to point to the next instruction to be fetched from code memory (ROM).
  - Each time an instruction is fetched, PC is incremented so that it points to the next instruction to be fetched for execution.
  - PC is modified when a jump or call instruction is executed.
- It has 16-bit Data pointer (DPTR) register.
  - It is used to access external data memory.
  - It can be accessed individually as two 8-bit registers, DPH and DPL.
  - It is under the control of the program.
- It has 8-bit Stack pointer (SP) register.
  - It is used to point to the top of the stack.
  - It grows in the direction of increasing memory address.
- It has Internal ROM or EPROM of 0 to 4K.
  - It is the program memory which is read-only.
  - 8031 has 0K ROM
  - 8051 has 4K ROM
  - 8751 has 4K UV-EPROM
- It has Internal RAM of 128 bytes which is sectioned as follows:
  - 4 register banks, each containing 8 registers where each register size is 8-bit.
  - 16 bytes, which may be addressed at the bit level.
  - 80 bytes of general-purpose data memory.
- 32 input/output pins arranged as four 8-bit ports: P0 - P3
  - P0 is used as a general purpose I/O port. With presence of external memory it functions as a multiplexed address (lower byte address A0-A7) and data bus (D0-D7).
  - P1 is used only as a general purpose I/O port.
  - P2 is used as a general purpose I/O port. With presence of external memory it functions as an address bus (higher byte address A8-A15).
  - P3 is used as a general purpose I/O port. Alternate functions are
    - interrupt inputs

- serial data I/O
- timer/counter input
- read/write control when external memory is used.

- Special Function Registers (SFRs) are dedicated RAM area (128 bytes) ranging from 80H to FFH. SFRs contain control registers.
  - Control registers: TCON, TMOD, SCON, PCON, IP, and IE.
- It has two 16-bit Timer/Counters: T0 and T1.
- It has one serial port for serial communication.
  - SBUF is a serial port data buffer register which is used to hold the data to be transmitted and receive the data via serial port.
- It has 2 external and 3 internal interrupt sources.
- It has Oscillator and clock circuits to generate timing signals for synchronization of all operations in the microcontroller.

INTERNAL RAM ORGANIZATION



- The 128 bytes internal RAM is organized into three distinct area:

- 32-bytes from address 00H to 1FH that make up 32 working registers organized as four banks of eight registers each.
  - Register banks are numbered 0-3
  - 8 registers are named as R0-R7.
  - Each register can be addressed by name or by its RAM address.
  - Bits RS0 (bit 3) and RS1 (bit 4) in the PSW determine which bank of registers is currently in use at any time program is running.
  - Register banks not selected can be used as general-purpose RAM.
  - Bank 0 is selected upon reset.
- A bit-addressable area of 16 bytes occupies RAM byte addresses 20H to 2FH, forming a total of 128 addressable bits.
  - An addressable bit may be specified by its bit address of 00H to 7FH, or 8 bits may form any byte address from 20H to 2FH.
- A general-purpose RAM area (Scratch pad memory) above the bit area, from 30H to 7FH, are addressable as bytes.

FLAGS AND THE PROGRAM STATUS WORD (PSW)

- Flags are 1-bit registers provided to store the results of certain program instructions.
- There are instructions in 8051 which can check the flags status and make decisions based on them.
- The 8051 has
  - 4 math flags which are updated based on result of math operations.
    - Math flags include Carry (CY), Auxiliary Carry (AC), Overflow (OV), and Parity (P).
  - 3 general-purpose user flags which may be used by the programmer to record some event in the program.
    - General-purpose User flags are named F0 (in PSW register), GF0 and GF1 (in PCON register)
- The direct RAM address of PSW register is D0H.
- PSW register is bit addressable as PSW.0 to PSW.7

The PSW is shown in below figure:

Bit	Symbol	Function
7	CY	Carry flag; CY=1 if carry is generated out of MSB or borrow is generated into MSB. It is used in arithmetic, JUMP, ROTATE, and Boolean instructions
6	AC	Auxiliary Carry flag; AC=1 if carry is generated from lower nibble to higher nibble or borrow is generated from higher nibble to lower nibble. It is used for BCD arithmetic.
5	F0	User Flag 0
4	RS1	Register bank Select bit 1
3	RS0	Register bank Select bit 0
RS1 and RS0 bits identify which of the four general-purpose register banks is currently in use by the program. RS1 RS0 0 0 Select register bank 0 0 1 Select register bank 1 1 0 Select register bank 2 1 1 Select register bank 3		
2	OV	Overflow flag; OV=1 if the result of operation exceeds the representable range (- 128 to +127). It is used in arithmetic instructions
1	-	Undefined; It is reserved for future use
0	P	Parity flag; It shows the parity of register A. P=1 if register A contains odd number of 1's (odd parity).

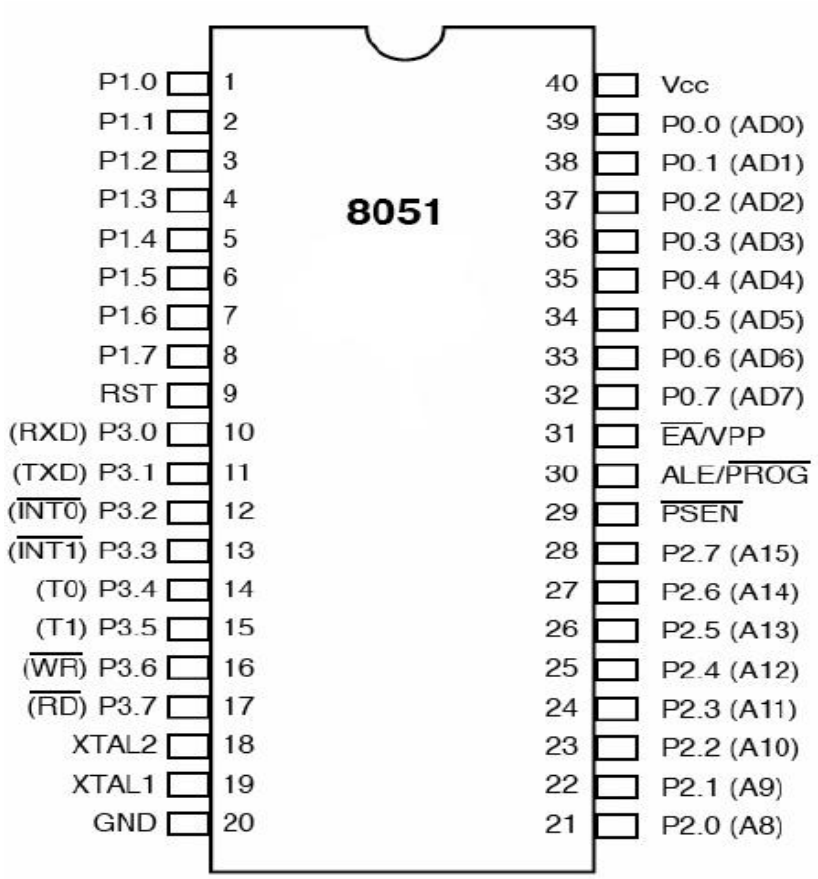
SPECIAL FUNCTION REGISTERS (SFRs)

- SFRs may be addressed much like 128 byte internal RAM, using addresses from 80H to FFH.
- Some SFRs are also bit-addressable. This feature allows the programmer to change only what needs to be altered, leaving the remaining bits in that SFR unchanged.

- Not all of the addresses from 80H to FFH are used for SFRs, and attempting to use an address that is not defined results in unpredictable results.
- SFRs are named in certain opcodes by their functional names, and are referenced by other opcodes by their addresses.
- Note that PC is not part of SFR and has no internal RAM address.
- The SFR names and equivalent internal RAM addresses are given in the following table:

NAME	FUNCTION	INTERNAL RAM ADDRESS
A	Accumulator	0E0H
B	Arithmetic	0F0H
DPH	Addressing external memory	83H
DPL	Addressing external memory	82H
IE	Interrupt enable control	0A8H
IP	Interrupt priority	0B8H
P0	I/O port Latch	80H
P1	I/O port Latch	90H
P2	I/O port Latch	0A0H
P3	I/O port Latch	0B0H
PCON	Power control	87H
PSW	Program Status Word	0D0H
SCON	Serial port control	98H
SBUF	Serial port data buffer	99H
SP	Stack Pointer	81H
TMOD	Timer/Counter mode control	89H
TCON	Timer/Counter control	88H
TL0	Timer 0 low byte	8AH
TH0	Timer 0 high byte	8CH
TL1	Timer 1 low byte	8BH
TH1	Timer 1 high byte	8DH

PIN- OUTS OF 8051 MICROCONTROLLER



- Out of 40 pins, 32 are I/O pins which are available as 4 parallel ports of 8-bits each.
  - **Port 0 (32-39):**
    - It is an 8-bit bi-directional I/O port.
    - It is associated with a latch whose address is 80H.
    - It is bit/byte addressable.
    - During external memory access (RAM/ROM), it functions as multiplexed data and lower-order address bus AD0-AD7.
  - **Port 1 (1-8):**
    - It is an 8-bit bi-directional I/O port.
    - It is associated with a latch whose address is 90H.
    - It is bit/byte addressable.
    - It functions as simply an I/O port and it does not have any alternate functions.
  - **Port 2 (21-28):**
    - It is an 8-bit bi-directional I/O port.
    - It is associated with a latch whose address is A0H.

- It is bit/byte addressable.
- During external memory access (RAM/ROM), it functions as higher-order address bus A8-A15.

- **Port 3 (10-17):**
  - It is an 8-bit bi-directional I/O port.
  - It is associated with a latch whose address is B0H.
  - It is bit/byte addressable.
  - Each pin of this port has alternate functions:
    - ✓ P3.0 (RxD): It is an input signal to serial port data buffer register SBUF, through which microcontroller receives data of serial communication network.
    - ✓ P3.1 (TxD): It is an output signal of serial port data buffer register SBUF, through which microcontroller transmits data of serial communication network.
    - ✓ P3.2 ($\overline{INT0}$) and P3.3 ($\overline{INT1}$): These are external interrupt input signals through which microcontroller can be interrupted by peripheral.
    - ✓ P3.4 (T0) and P3.5 (T1): These are input signals to internal timer-0 and timer-1 circuits respectively.
    - ✓ P3.6 ($\overline{WR}$): It is an active low write output control signal. During external RAM access, it is generated by microcontroller (WR = 0) to perform write operation to external RAM.
    - ✓ P3.7 ($\overline{RD}$): It is an active low read output control signal. During external RAM access, it is generated by microcontroller (RD = 0) to perform read operation from external RAM.
- **RST (9):**
  - It is an active high input signal used to reset microcontroller.
  - To reset 8051 microcontroller, RST is made high for at least two machine cycles.
  - Example: After reset, PC=0000H, SP = 07H, all internal RAM locations are cleared to zero.
- **XTAL2 (18) and XTAL1 (19):**
  - These two input lines for on-chip oscillator and clock generator circuit. A crystal resonator is connected between these two pins.

- **VCC (40):**

  It is connected to + 5V power supply.

- **GND (20):**

  It is connected to ground reference.

- $\overline{PSEN}$ **(29) – Program Store Enable:**
  - It is an active low output signal used to enable external program memory (ROM).
  - It is connected to OE of external ROM.
  - When PSEN = 0, ROM becomes enabled and microcontroller reads the contents of external ROM locations.
- **ALE (30) – Address Latch Enable:**
  - It is an active high output signal used to demultiplex AD0-AD7 of port0.
  - When ALE goes high, external address latch becomes enabled and whatever is there in input to latch will be available at the output of latch. When ALE goes low, it is used as data bus.
- $\overline{EA}$ **(31) – External Access:**
  - It is an active low input signal to the microcontroller.
  - For 8051, this pin is connected to Vcc so that the microcontroller can access both internal and external program memory (ROM). Internal ROM is selected for address 0000H-0FFFH. Beyond this address (1000H-FFFFH) external ROM is selected.
  - For 8031, this pin is connected to GND so that the microcontroller can access only external ROM.

ADDRESSING MODES

The different methods of specifying the location of the operand in an instruction is referred to as addressing modes.

The different addressing modes supported in 8051 are:

- Immediate addressing mode
- Register addressing mode
- Direct addressing mode

- Register-indirect addressing mode
- Indexed addressing mode

Immediate addressing mode

- In this addressing mode, the operand value is specified as part of the instruction.
- The operand value is preceded by '#' symbol to indicate immediate addressing mode in the instruction.
- Only the source operand can be specified using this addressing mode.
- The destination operand can be either a register or a memory location.
- Examples:

```
MOV A,#34H
ANL A,#01000010B
ADD A,#20
MOV R2,#78H
MOV 26H,#10H
MOV @R0,#45H
```

Register addressing mode

- In this addressing mode, the operand value is specified in a register and the register name is specified in the instruction.
- The registers that can be specified in this mode are A, R0-R7, DPTR. Other registers can be specified using their direct RAM address.
- Either source operand or destination operand can be specified using this addressing mode. Both the source and destination operands can be in this mode if one of the operand is A register and other operand is register R0-R7
- Examples:

```
MOV A,#34H
ANL A,R0
ADD A,@R0
INC R0
MOV 26H,R3
```

Direct addressing mode (used to access internal data memory only)

- In this addressing mode, the address of the operand is specified as part of the instruction.
- The valid direct address that can be used is in the range 00H-7FH. The SFRs implemented in the address range 80H-FFH can be referred using direct addressing mode.
- Either source or destination or both the operands can be specified using this addressing mode. The operand specified using this mode cannot be altered during execution of the program.
- Examples:

```
MOV A,34H
ANL 20H,R0
ADD A,40H
MOV R2,01H
PUSH 0E0H
MOV 26H,35H
```

Register-Indirect addressing mode

- In this addressing mode, the address of the operand is specified in a register and the register name is used in the instruction.
- The register name is preceded by '@' symbol to indicate register-indirect addressing mode in the instruction.
- The valid registers that can be used to hold the address are R0, R1, and DPTR.
- Either source or destination operand can be specified using this addressing mode.
- The operand specified using this mode can be altered during execution of the program.
- Examples:

```
MOV A,@R0
ADD A,@R0
MOV @R1,01H
MOVX A,@DPTR
```

Indexed addressing mode

- In this addressing mode, the effective address of the operand is calculated by adding the offset held in A register with the base address held in DPTR or PC register.
- The expression A+DPTR or A+PC preceded by '@' symbol is used to indicate indexed addressing mode in the instruction.
- Only the source operand can be specified using this addressing mode.
- This addressing mode can also be used to access the look-up table entries stored in the program memory.
- The operand specified using this mode can be altered during execution of the program.
- Example:

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

INSTRUCTION SET AT GLANCE

SL. NO.	INSTRUCTION		DESCRIPTION	LENGTH (BYTES)	CYCLE(S)
	MNEMONIC	OPERAND(S)			
ARITHMETIC OPERATIONS					
1.	ADD	A,Rn	Add register to Accumulator	1	1
2.	ADD	A,Direct	Add direct byte to Accumulator	2	1
3.	ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
4.	ADD	A,#Data	Add immediate data to Accumulator	2	1
5.	ADDC	A,Rn	Add register to Accumulator with Carry flag	1	1
6.	ADDC	A,Direct	Add direct byte to Accumulator with Carry flag	2	1
7.	ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry flag	1	1
8.	ADDC	A,#Data	Add immediate data to Accumulator with Carry flag	2	1
9.	SUBB	A,Rn	Subtract register from Accumulator with Borrow	1	1
10.	SUBB	A,Direct	Subtract direct byte from Accumulator with Borrow	2	1
11.	SUBB	A,@Ri	Subtract indirect RAM from Accumulator with Borrow	1	1
12.	SUBB	A,#Data	Subtract immediate data from Accumulator with Borrow	2	1
13.	INC	A	Increment Accumulator by one	1	1
14.	INC	Rn	Increment register by one	1	1
15.	INC	Direct	Increment direct byte by one	2	1
16.	INC	@Ri	Increment indirect RAM by one	1	1
17.	INC	DPTR	Increment Data pointer by one	1	2
18.	DEC	A	Decrement Accumulator by one	1	1
19.	DEC	Rn	Decrement register by one	1	1
20.	DEC	Direct	Decrement direct byte by one	2	1

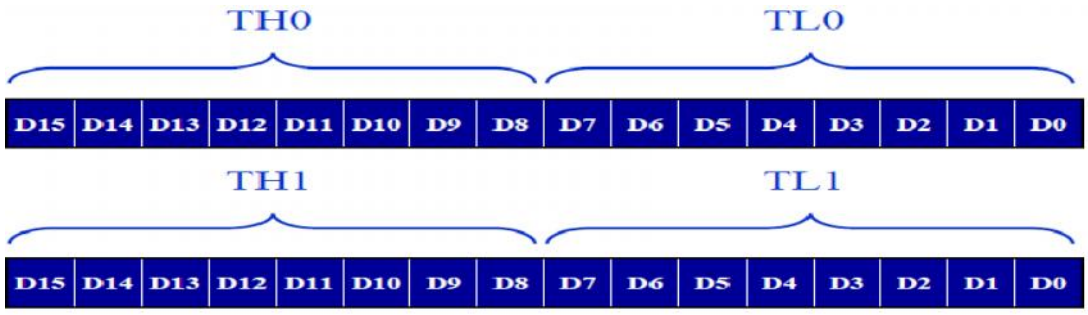
21.	DEC	@Ri	Decrement indirect RAM by one	1	1
22.	MUL	AB	Multiply Accumulator and B. (A)=lower byte, (B)=higher byte	1	4
23.	DIV	AB	Divide Accumulator by B. (A)=Quotient, (B)=Remainder	1	4
24.	DA	A	Decimal Adjust Accumulator	1	1
DATA TRANSFER OPERATIONS					
25.	MOV	A,Rn	Move register to Accumulator	1	1
26.	MOV	A,Direct	Move direct byte to Accumulator	2	1
27.	MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
28.	MOV	A,#Data	Move immediate data to Accumulator	2	1
29.	MOV	Rn,A	Move Accumulator to register	1	1
30.	MOV	Rn,Direct	Move direct byte to register	2	2
31.	MOV	Rn,#Data	Move immediate data to register	2	1
32.	MOV	Direct,A	Move Accumulator to direct byte	2	1
33.	MOV	Direct,Rn	Move register to direct byte	2	2
34.	MOV	Direct,Direct	Move direct byte to direct byte	3	2
35.	MOV	Direct,@Ri	Move indirect RAM to direct byte	2	2
36.	MOV	Direct,#Data	Move immediate data to direct byte	3	2
37.	MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
38.	MOV	@Ri,Direct	Move direct byte to indirect RAM	2	2
39.	MOV	@Ri,#Data	Move immediate data to indirect RAM	2	1
40.	MOV	DPTR,#Data16	Load Data pointer with a 16-bit constant	3	2
41.	MOVC	A,@A+DPTR	Move code byte relative to DPTR to Accumulator	1	2
42.	MOVC	A,@A+PC	Move code byte relative to PC to Accumulator	1	2
43.	MOVX	A,@Ri	Move external RAM (8-bit address) to Accumulator	1	2
44.	MOVX	A,@DPTR	Move external RAM (16-bit address) to Accumulator	1	2
45.	MOVX	@Ri,A	Move Accumulator to external RAM (8-bit address)	1	2
46.	MOVX	@DPTR,A	Move Accumulator to external RAM (16-bit address)	1	2
47.	PUSH	Direct	Push direct byte onto stack	2	2
48.	POP	Direct	Pop direct byte from stack	2	2
49.	XCH	A,Rn	Exchange register with Accumulator	1	1
50.	XCH	A,Direct	Exchange direct byte with Accumulator	2	1
51.	XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	1
52.	XCHD	A,@Ri	Exchange low-order Digit indirect RAM with Accumulator	1	1
LOGICAL OPERATIONS					
53.	ANL	A,Rn	AND register to Accumulator	1	1
54.	ANL	A,Direct	AND direct byte to Accumulator	2	1
55.	ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
56.	ANL	A,#Data	AND immediate data to Accumulator	2	1
57.	ANL	Direct,A	AND Accumulator to direct byte	2	1
58.	ANL	Direct,#Data	AND immediate data to direct byte	3	2
59.	ORL	A,Rn	OR register to Accumulator	1	1

60.	ORL	A,Direct	OR direct byte to Accumulator	2	1
61.	ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
62.	ORL	A,#Data	OR immediate data to Accumulator	2	1
63.	ORL	Direct,A	OR Accumulator to direct byte	2	1
64.	ORL	Direct,#Data	OR immediate data to direct byte	3	2
65.	XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
66.	XRL	A,Direct	Exclusive -OR direct byte to Accumulator	2	1
67.	XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	1
68.	XRL	A,#Data	Exclusive-OR immediate data to Accumulator	2	1
69.	XRL	Direct,A	Exclusive-OR Accumulator to direct byte	2	1
70.	XRL	Direct,#Data	Exclusive-OR immediate data to direct byte	3	2
71.	CLR	A	Clear Accumulator	1	1
72.	CPL	A	Complement Accumulator	1	1
73.	RL	A	Rotate Accumulator Left	1	1
74.	RLC	A	Rotate Accumulator Left through Carry flag	1	1
75.	RR	A	Rotate Accumulator Right	1	1
76.	RRC	A	Rotate Accumulator Right through Carry flag	1	1
77.	SWAP	A	Swap nibbles within the Accumulator	1	1
BOOLEAN VARIABLE MANIPULATION					
78.	CLR	C	Clear Carry flag	1	1
79.	CLR	Bit	Clear direct bit	2	1
80.	SETB	C	Set Carry flag	1	1
81.	SETB	Bit	Set direct bit	2	1
82.	CPL	C	Complement Carry flag	1	1
83.	CPL	Bit	Complement direct bit	2	1
84.	ANL	C,Bit	AND direct bit to Carry flag	2	2
85.	ANL	C,/Bit	AND complement of direct bit to Carry flag	2	2
86.	ORL	C,Bit	OR direct bit to Carry flag	2	2
87.	ORL	C,/Bit	OR complement of direct bit to Carry flag	2	2
88.	MOV	C,Bit	Move direct bit to Carry flag	2	1
89.	MOV	Bit,C	Move Carry flag to direct bit	2	2
PROGRAM AND MACHINE CONTROL					
90.	ACALL	Addr11	Absolute Subroutine Call	2	2
91.	LCALL	Addr16	Long Subroutine Call	3	2
92.	RET		Return from Subroutine	1	2
93.	RETI		Return from Interrupt	1	2
94.	AJMP	Addr11	Absolute Jump	2	2
95.	LJMP	Addr16	Long Jump	3	2
96	SJMP	Rel	Short Jump (relative addr)	2	2
97.	JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
98.	JZ	Rel	Jump if Accumulator is zero	2	2
99.	JNZ	Rel	Jump if Accumulator is Not zero	2	2
100.	JC	Rel	Jump if Carry flag is set	2	2
101.	JNC	Rel	Jump if Carry flag is Not set	2	2
102.	JB	Bit,Rel	Jump if direct bit is set	3	2

103.	JNB	Bit,Rel	Jump if direct bit id Not set	3	2
104.	JBC	Bit,Rel	Jump if direct bit is set and Clear bit	3	2
105.	CJNE	A,Direct,Rel	Compare direct to A and Jump if Not Equal	3	2
106.	CJNE	A,#Data,Rel	Compare immediate to A and jump if Not Equal	3	2
107.	CJNE	Rn,#Data,Rel	Compare immediate to register and Jump if Not Equal	3	2
108.	CJNE	@Ri,#Data,Rel	Compare immediate to indirect and Jump if Not Equal	3	2
109.	DJNZ	Rn,Rel	Decrement register and Jump if Not Zero	2	2
110.	DJNZ	Direct,Rel	Decrement direct and Jump if Not Zero	3	2
111.	NOP		No Operation	1	1

Timer/Counter Programming

- The 8051 has two 16-bit timers: Timer0 and Timer1.
- These timers can be used either as timers to generate a time delay or as counters to count events happening outside the microcontroller.
- Timer 0 can be accessed as two 8-bit registers, TL0 (Timer0 Low byte) and TH0 (Timer0 High byte) whose RAM addresses are 8AH and 8CH respectively.
- Timer 1 can be accessed as two 8-bit registers, TL1 (Timer1 Low byte) and TH1 (Timer1 High byte) whose RAM addresses are 8BH and 8DH respectively.
- Below figure shows Timer0 and Timer1 registers:



TMOD (Timer Mode) register (RAM address:89H)

- Both Timer0 and Timer1 use TMOD register to set the various timer operation modes.
- TMOD is an 8-bit register in which the lower 4 bits are set aside for Timer0 and the upper 4 bits for Timer 1.

- M0 and M1 bits in both the nibbles are used to set the timer mode.

M1	M0	Mode	Operating Mode
0	0	0	13-bit timer mode
0	1	1	16-bit timer mode
1	0	2	8-bit auto reload
1	1	3	Split timer mode

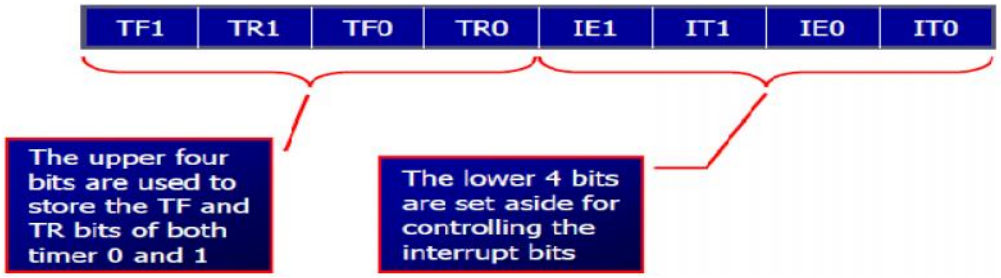
- C/T is used to decide whether the timer is used as a delay generator or an event counter. If C/T = 0, then it is used as timer for delay generation (input from internal system clock). The frequency for the timer is always $1/12^{th}$ the frequency of the crystal attached to 8051. If C/T = 1, then it is used as event counter to count each clock pulse coming from external pin (Pin P3.4 for timer0 and P3.5 for timer1).
- If GATE = 0, then
  - instructions TR0 = 1 and TR0 = 0 starts and stops the timer0,
  - instructions TR1 = 1 and TR1 = 0 starts and stops the timer1.

  (TR0 and TR1 are bits of TCON register)

  If GATE = 1, then the timer is started and stopped by an external source (Pin P3.2 for timer0 and P3.3 for timer1).

TCON register (RAM address: 88H)

- While TMOD controls the timer modes, another register called TCON controls the timer/counter operations.



- Timer is started by setting TRx bit, which is called Timer Run control bit and it is stopped by clearing TRx bit.
- Whenever a timer counts to its maximum value, it sets Timer overflow Flag, TFx.

Bit	Symbol	Function
TCON.7	TF1	Timer1 overflow Flag

TCON.6	TR1	Timer1 Run control bit
TCON.5	TF0	Timer 0 overflow Flag
TCON.4	TR0	Timer 0 Rum control bit

Mode2 Programming:

The following are the characteristics and operations of mode2:

1. It is an 8-bit timer. Therefore, it allows only values of 00H to FFH to be loaded into the timer's register TH.
2. After TH is loaded with the 8-bit value, the 8051 gives a copy of it to TL. Then, the timer must be started, which is done by the instruction TR0 = 1 for Timer0 and TR1=1 for Timer1.
3. After the timer is started, it starts to count up by incrementing the TL register. It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00H, it sets high the TF0 for Timer0 and TF1 for Timer1.
4. When TL register rolls from FFH to 0 and TF=1, TL is reloaded automatically with the original value kept in TH register. To repeat the process, we must simply clear TF and let it go without any need by the programmer to reload the original value. This makes mode2 an auto-reload.

Steps to program in mode2:

To generate a time delay using the timer's mode2, the following steps are taken:

1. Load the TMOD register with value indicating which timer (Timer0 or Timer1) is to be used, and select timer mode as mode2.
2. Load the TH register with the initial count value.
3. Start the timer.
4. Keep monitoring the timer flag (TF) with the statement while(TF==0); to see whether it is raised. Come out of the loop when TF goes high.
5. Clear the TF flag.
6. Go back to step4, since mode2 is auto-reload.

References

1. Kenneth Ayala, "The 8051 Microcontroller", 3rd edition, Cengage Learning 2005.

2. Muhammad Ali Mazidi, Janice Gillespie Mazidi, Rollin D. McKinlay, "The 8051 Microcontroller and Embedded Systems – using assembly and C", 2nd edition - PHI, 2006 / Pearson, 2006.