ATME COLLEGE OF ENGINEERING

13th KM Stone, Bannur Road, Mysore - 570 028

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

4th Semester Lab Manual

Microcontroller Lab

(BEC456A)

Purpose of the Laboratory Course

- An understanding of the principles as well as the practical limitations of electronics commonly used experimental science and engineering.
- This lab provides students with an environment where they can take apart the electronic components and put them together again in a functional mode.
- The laboratory is equipped with modern test equipment including both analog and digital oscilloscopes, Digital Multi-meters, Signal Generators and Power Supplies. The hands-on approach learning is used as a laboratory teaching method.
- Students will gain a working knowledge in design, integrating and testing of electronics systems.
- To allow the students to perform experiments that demonstrate the theory of Analog and Digital Electronics that will be discussed in the lecture course.
- To introduce the electronic components.
- To familiarize the proper laboratory procedure.
- To enable students, implement their creative ideas within and beyond the scope of the syllabus with the guidance of the Faculty/ Lab In-charge.

DO's

- Study the theory related to the experiment before coming to Lab to carry out the Lab exercises.
- Enter your name and ID number in the register provided for this purpose. Enter the IN-TIME, OUT-TIME and station number along with your signature.
- Submit the Record of previously conducted experiment.
- All intermediate results, designs and observations to be recorded in the Observation book during Laboratory hours.
- Note down the specifications of the devices and equipment's used.
- Avoid unnecessary talking while conducting experiments.
- Handle the equipment, devices and components carefully.
- Before energizing the wired circuit gets it checked by the faculty/staff in-charge.
- If you need to make changes to the circuit, disconnect the power supply and the signal input.
- Note down the readings and observations immediately in the observation book.
- After the completion of the experiment switch-off the power supply.
- Do necessary calculations and draw the graph after completing the experiment.
- Show the result to the faculty/staff in-charge and get it checked for correctness or repetition of the experiment.
- Handover any components taken to the Lab instructor/ Lab assistant before leaving the lab.

DONT's

- Do not come late to the lab.
- Do not exceed the voltage/current ratings of the devices.
- Do not energize the equipment and circuits without getting it checked by the staff in charge.
- Do not spoil the connecting wires/ CRO probes.
- Avoid loose connection and short circuits.
- Do not use mobile phones in the laboratory.
- Any intentional damage to components and equipments will result in serious punishment.

RECORD – KEEPING

- Each student will have to compulsorily maintain a standard laboratory record in which all calculations, measurements, circuit diagram, and procedures should be written individually.
- Your record book and observation book will be checked each day for adequate progress and neatness during the course.
- All the pre-lab assignments must be completed before coming inside the lab.

MICROCONTROLLERS LAB

Sub Code: BEC456A **CIE Marks:** 50
Hrs / Week: 0:0:2 **Exam Hours:** 02
Credits: 01 **Exam Marks:** 50

CYCLE I: ASSEMBLY LANGUAGE PROGRAMMING

Data Transfer Programs:

1. Write an ALP to move a block of n bytes of date from source (20h) to destination (406) using Internal RAM
2. Write an ALP to move a block of n bytes of data from source (2000h) to destination (2050b) using External RAM
3. Write an ALP To exchange the source block starting with address 20h, (Internal RAM) containing N (05) bytes of data with destination block starting with address 40h (Internal RAM)
4. Write an ALP in exchange the source block starting with address 10h (Internal memory), contatung in (06) bytes of data with destination block starting at location 00h (External memory)

Arithmetic & Logical Operation Programs:

5. Write an ALP to add the byte in the RAM at 34h and 35h, store the result in the register R5 (LSB) and R6 (MSB), using Indirect Addressing Mode.
6. Write an ALP to subtract the bytes in Internal RAM 34h & 35h store the result in register R5 (LSB) & R6 (MSB).
7. Write an ALP to multiply two 8-bit numbers stored at 30h and 31h and store 16 bit result in 37h and 33h of Internal RAM
8. Write an ALP to perform division operation on 8-bit number by 8-bit number.
9. Write an ALP to separate positive and negative in a given array.
10. Write an ALP to separate even or add elements in a given array.
11. Write ALP to arrange the numbers in Descending order & ascending order.
12. Write an ALP to find Largest & Smallest number from a given array starting from 20h & store if in internal memory location 40h.
13. Write an ALP for Decimal UP – Counter.
14. Write an ALP for Decimal DOWN – Counter.
15. Write an ALP for Hexadecimal UP – Counter.
16. Write an ALP for Hexadecimal Down – Counter.

CYCLE II. C & Hardware Interfacing Programming

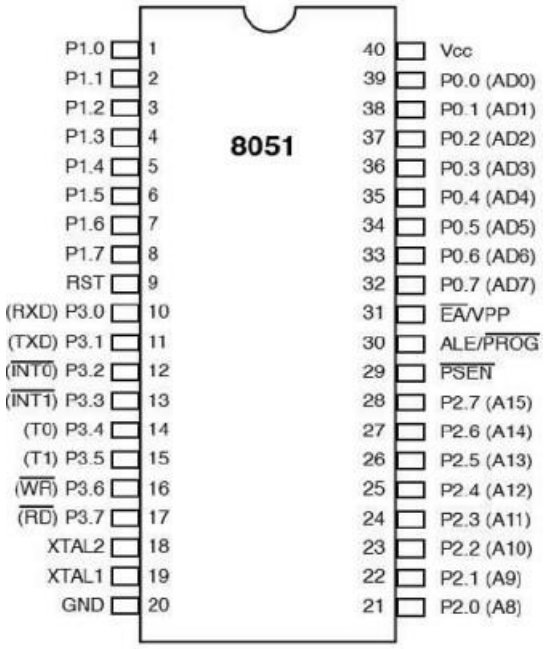
17. Write an 8051 C program to find the suns of first 10 Integer Numbers
18. Write an 8051 C program to find Factorial of a green number.
19. Write an 8051 C program to find the Square of a number (1 to 10) using Look Up Table
20. Write an 8051 C program to count the number of Ones and Zeroes in two consecutive memory locations
21. Write an 8051 C Program to rotate stepper motor in Clock & Ann Clockwise direction.
22. Write an 8051 C Program to Generate Sine & Square waveform DAC interface

Introduction to 8051 Microcontroller

8051 is one of the first and most popular microcontrollers also known as MCS-51. Intel introduced it in the year 1981. Initially, it came out as an N-type metal-oxide-semiconductor (NMOS) based microcontroller, but later versions were based on complementary metal-oxide-semiconductor (CMOS) technology. These microcontrollers were named 80C51, where C in the name tells that it is based on CMOS technology. It is an 8-bit microcontroller which means the data bus is 8-bit. Therefore, it can process 8 bits at a time. It is used in a wide variety of embedded systems like robotics, remote controls, the automotive industry, telecom applications, power tools, etc.

Features of 8051 MC

- 4K bytes internal ROM
- 128 bytes internal RAM
- Four 8-bit I/O ports (P0 - P3).
- Two 16-bit timers/counters
- One serial interface
- only 1 On chip oscillator (external crystal)
- 6 interrupt sources (2 external , 3 internal, Reset)
- 64K external code (program) memory(only read)PSEN
- 64K external data memory(can be read and write) by RD,WR
- Code memory is selectable by EA (internal or external)



The 8051 Microcontroller is a 40-pin Plastic Dual Inline Package (PDIP). The functions of the pins of this Microcontroller are as follows:

• Ports of 8051 Microcontroller –

Port 0 –

Port 0 or P0 is a General Purpose I/O Port. Consequently, it consists of 8 pins starting from pin 32 to pin 39. However, this port can also be utilized as a multiplexed Address and Data bus (from AD0 to AD7).

Port 1 –

Port 1 or P1, is also an 8-bit port starting from pin 1 to pin 8. Although similar to the P0, P1 is also a General Purpose I/O Port, however, unlike the other three ports, P1 does not serve any dual purpose. Hence the sole purpose of P1 is for interfacing.

Port 2 –

The pins from 21 to 28 belong to Port 2, or P2. Now when there is no presence of an external memory, the P2 acts as a General Purpose I/O Port. However, in the presence of external memory, P2 acts as an Address Bus, starting from A8 to A15.

Port 3 –

Though Port 3 or P3 usually acts as a normal I/O Port, it can provide some other functions as well. The pin numbers are from 10 to 17. The other functions are below.

• Pin10 – RXD

• Pin11 – TXD

• Pin12 – INT0 complement

• Pin13 – INT1

• Pin14 – T0

• Pin15 – T1

• Pin16 – WR

• Pin17 – RD complement

Additionally, the other pins are as follows –

• Pin20 – GND (Ground)

• Pin40 – VCC (Supply)

• Pin9 – RST (Reset)

• Pin18 – XTAL1 (Oscillator)

• Pin19 – XTAL2 (Oscillator)

• Pin29 – PSEN (Program Store Enable)

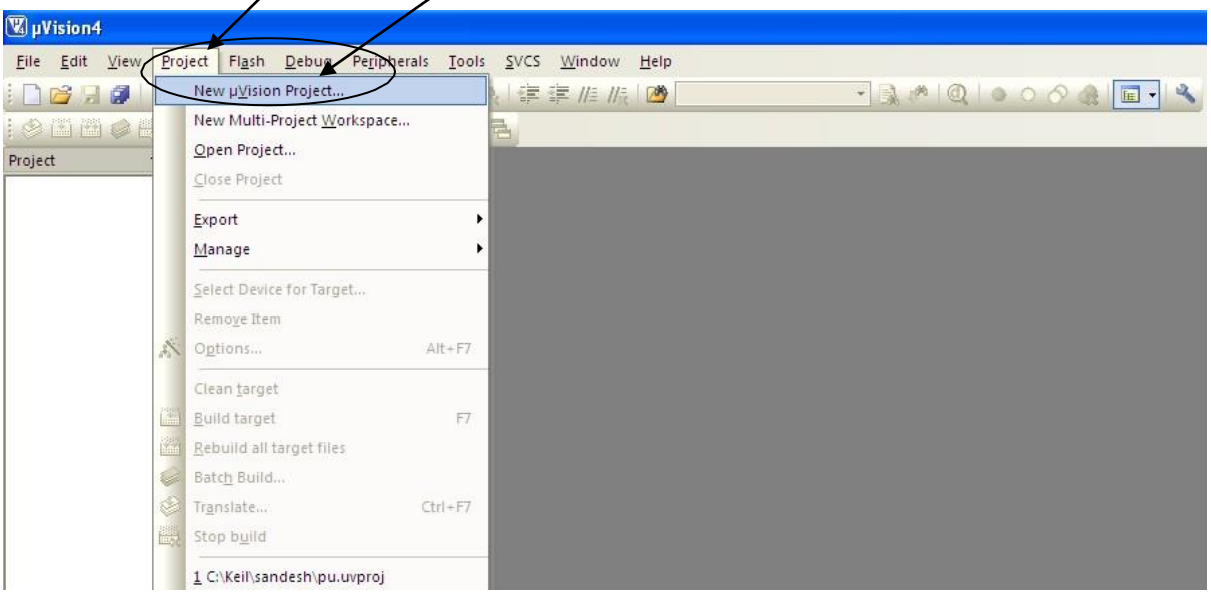
• Pin30 – ALE (Address Latch Enable)

• Pin31 – EA (External Access)

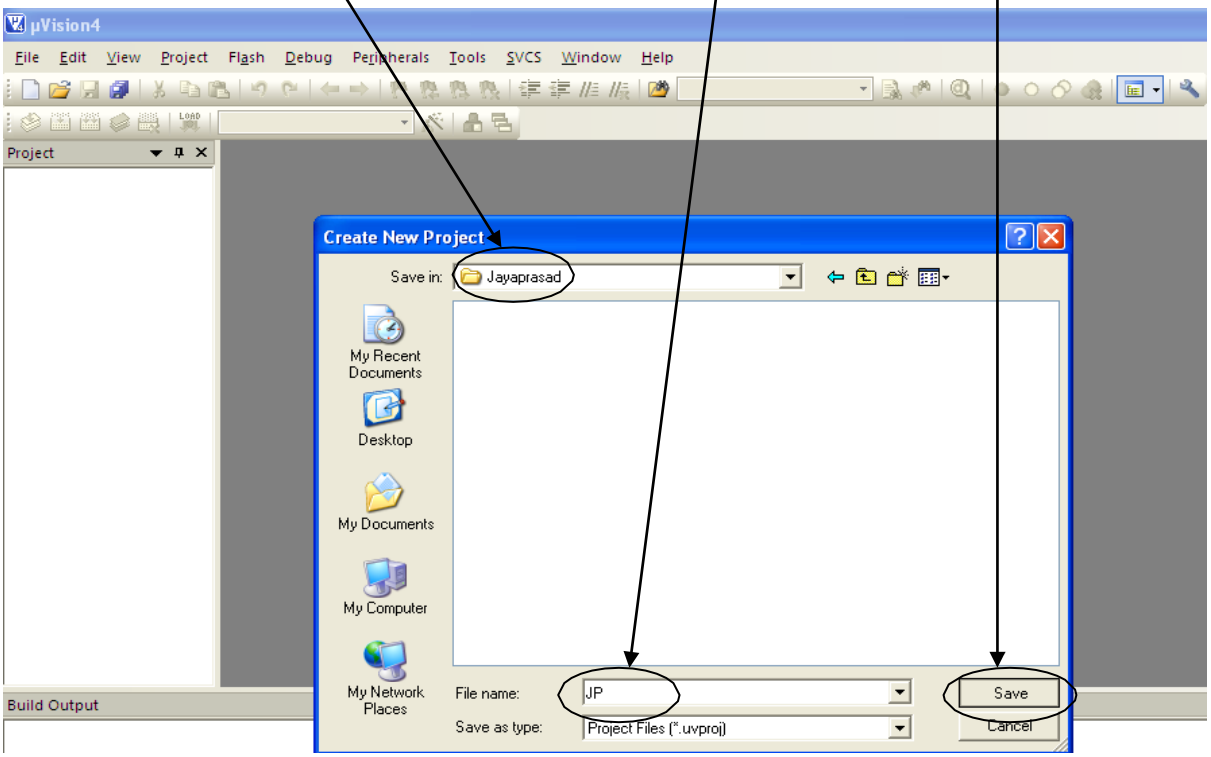
STEPS FOR EXECUTING THE SOFTWARE PROGRAM:

STEP 1: Select the Kiel µVision software.

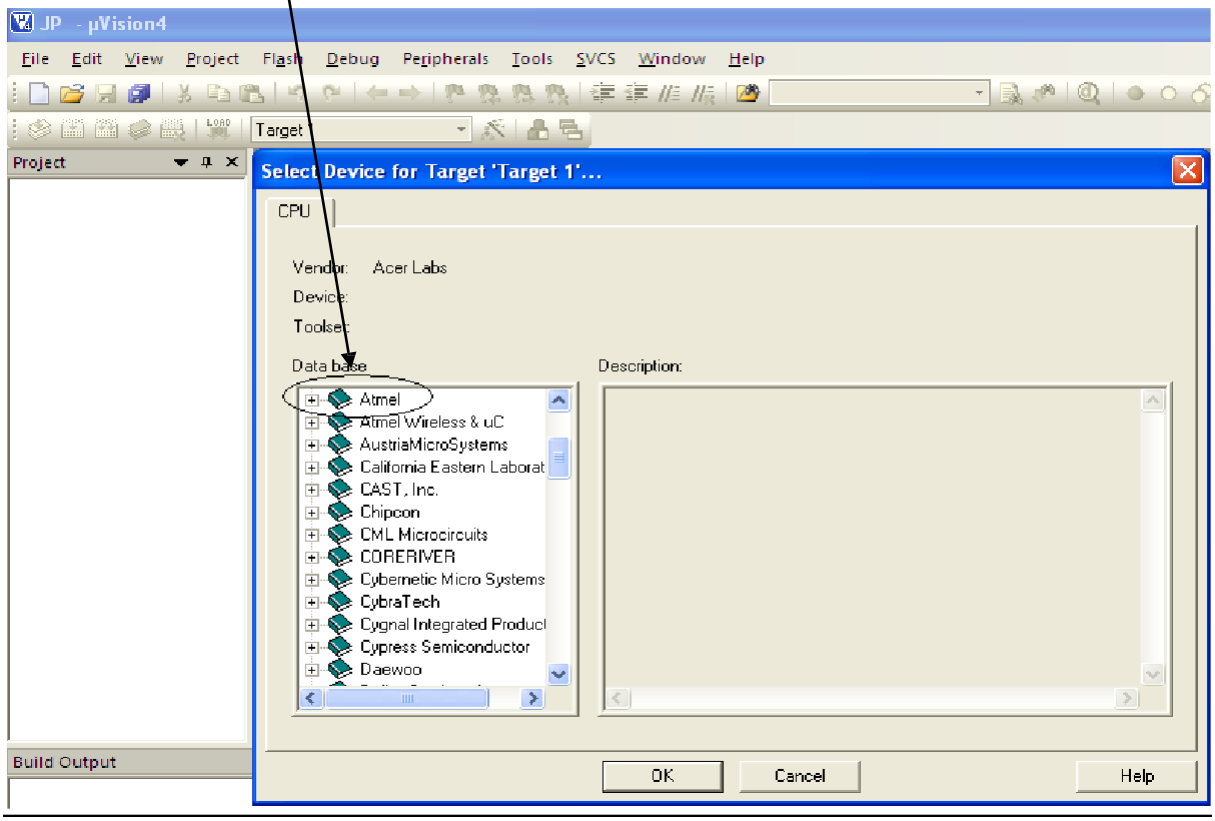
STEP 2: Select "Project" -> "New µVision Project".



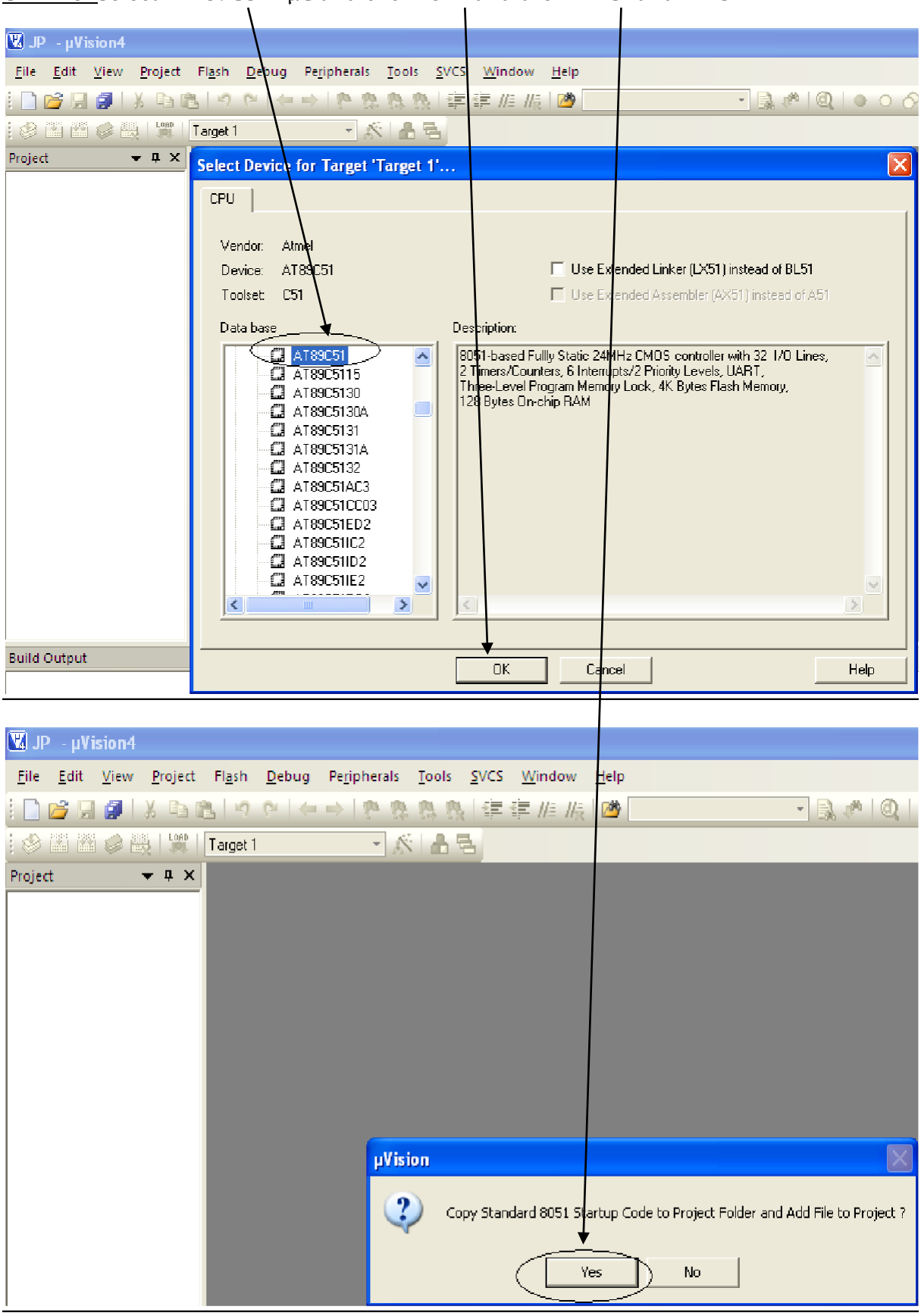
STEP 3: Create new project by entering your "File name" and then "Save" your file.



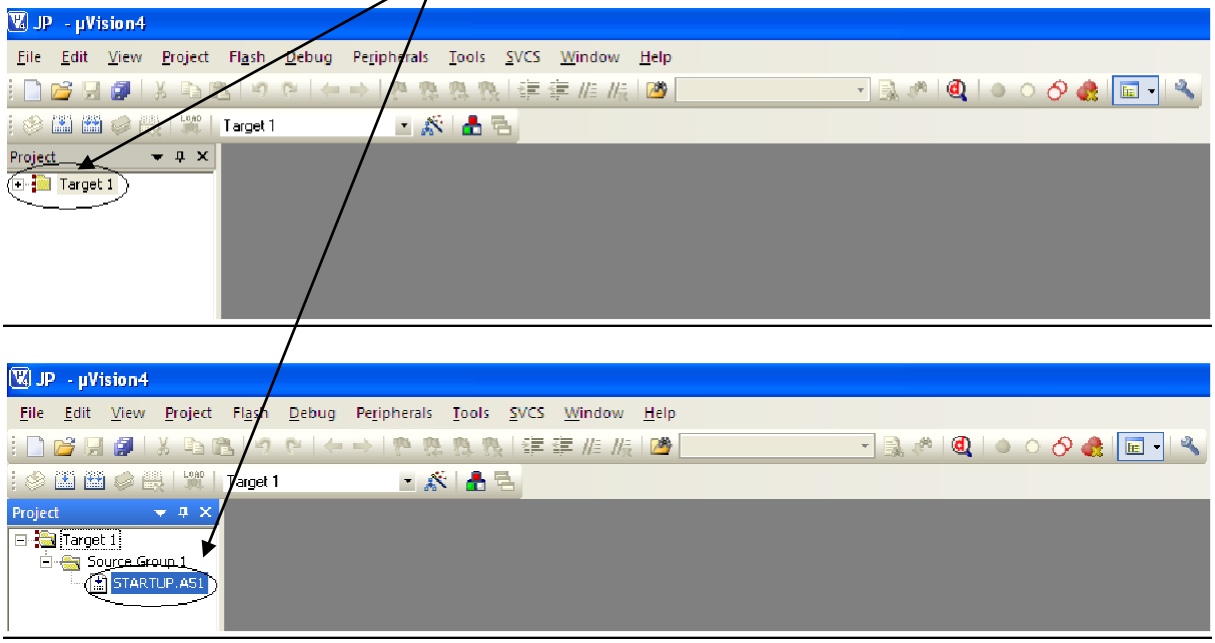
STEP 4: Choose "Atmel" microcontroller from the database.



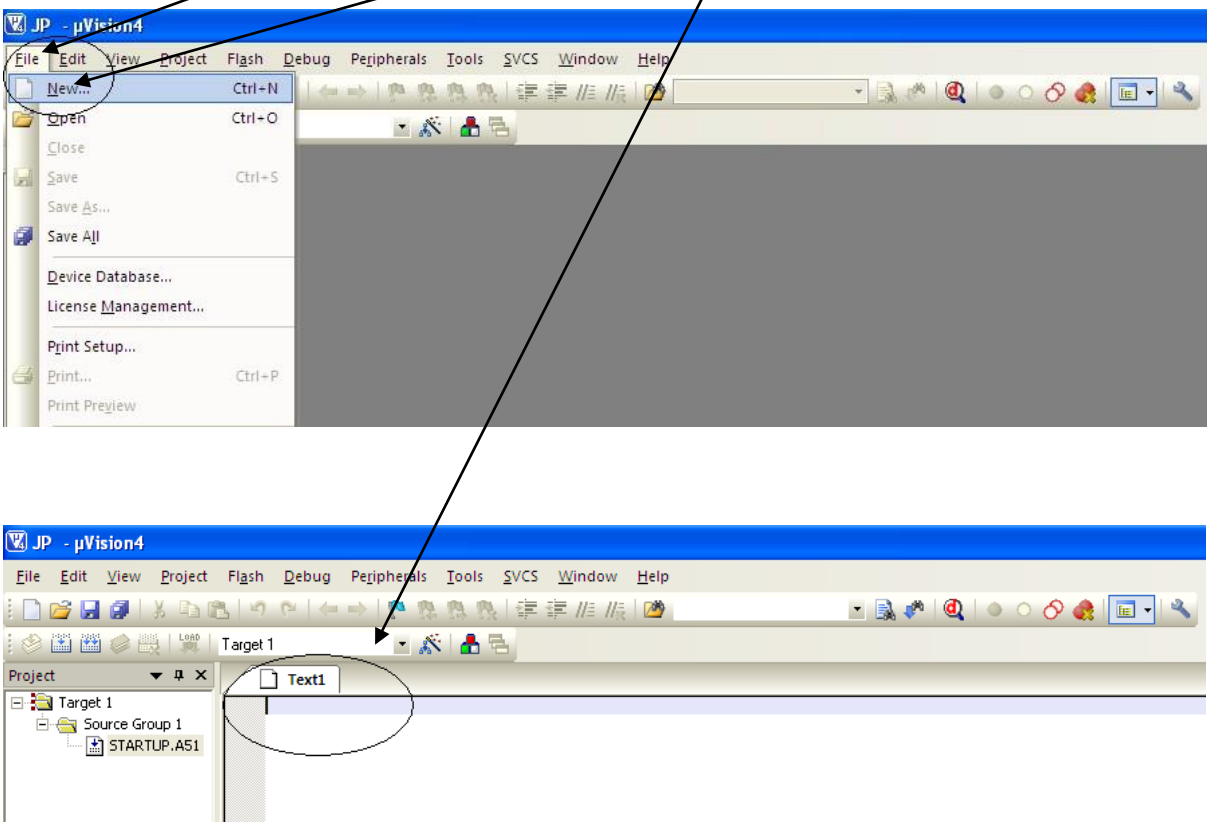
STEP 5: Select "AT89C51" µC and click "OK" and then "YES" and "YES".



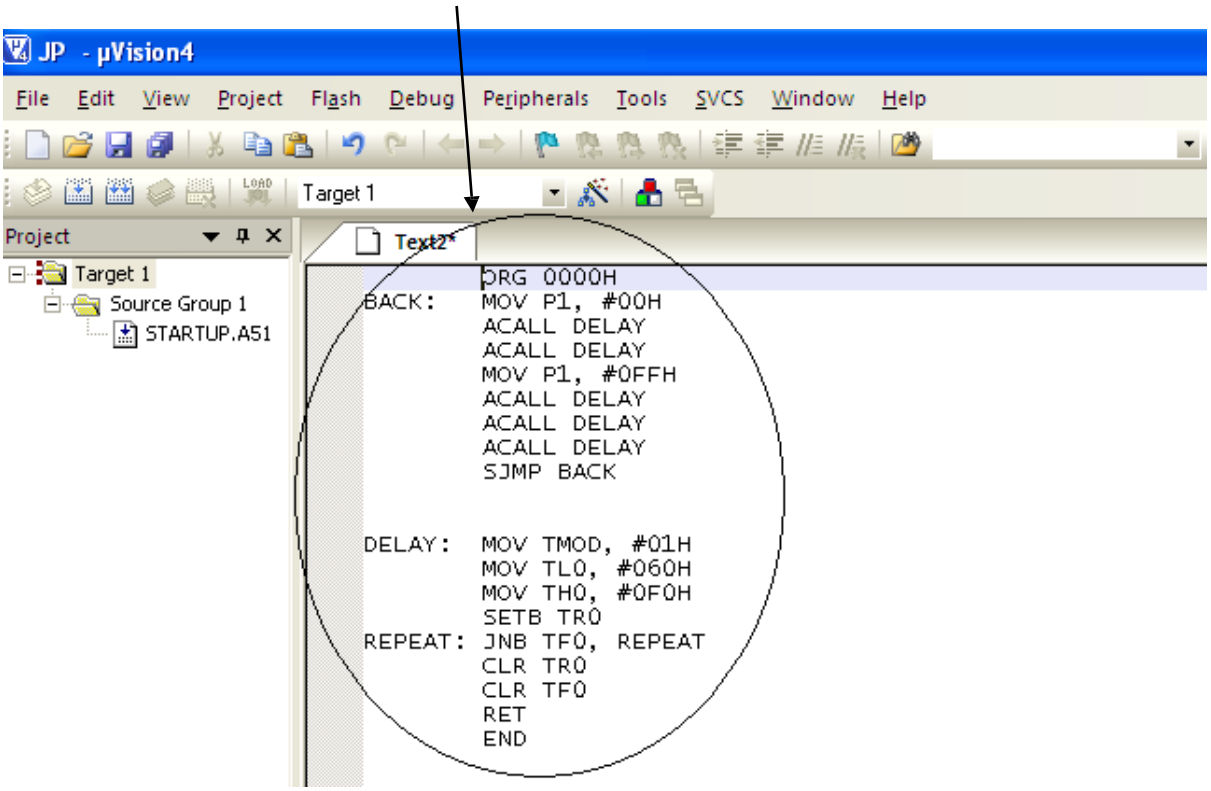
STEP 6: Make sure that "STARTUP.A51" file is added to the target.



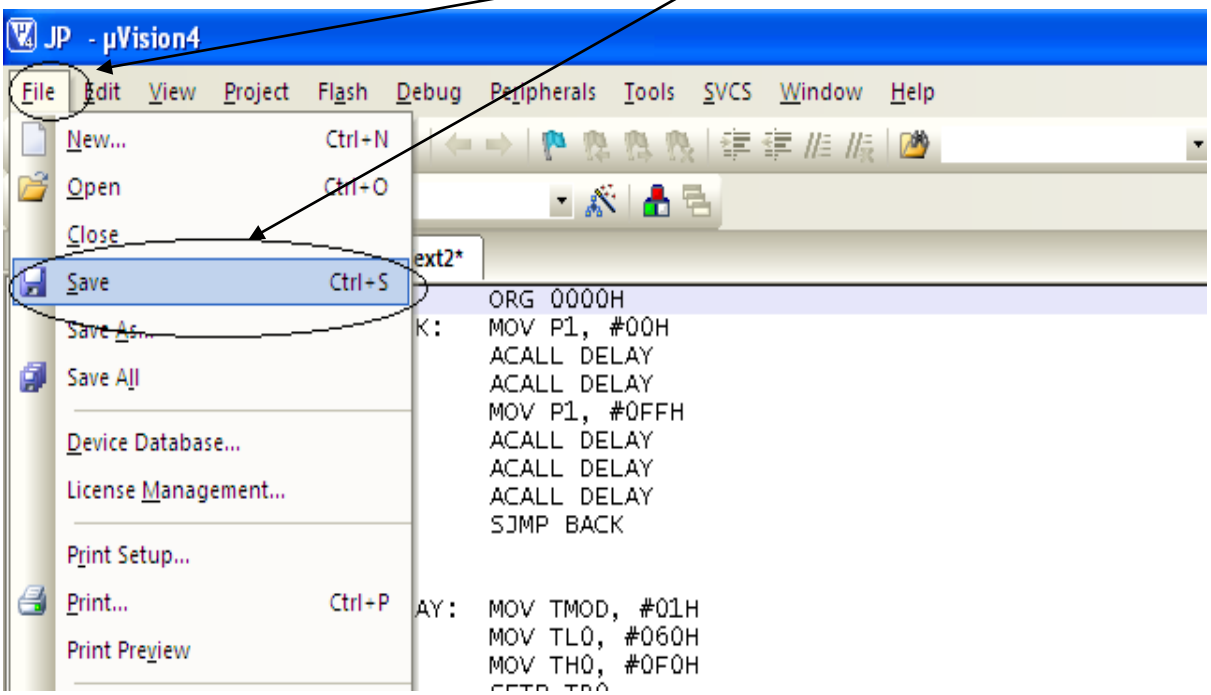
STEP 7: Go to "File" and select "New" for text (program) editing window.



STEP 8: Type your program in the editing window.

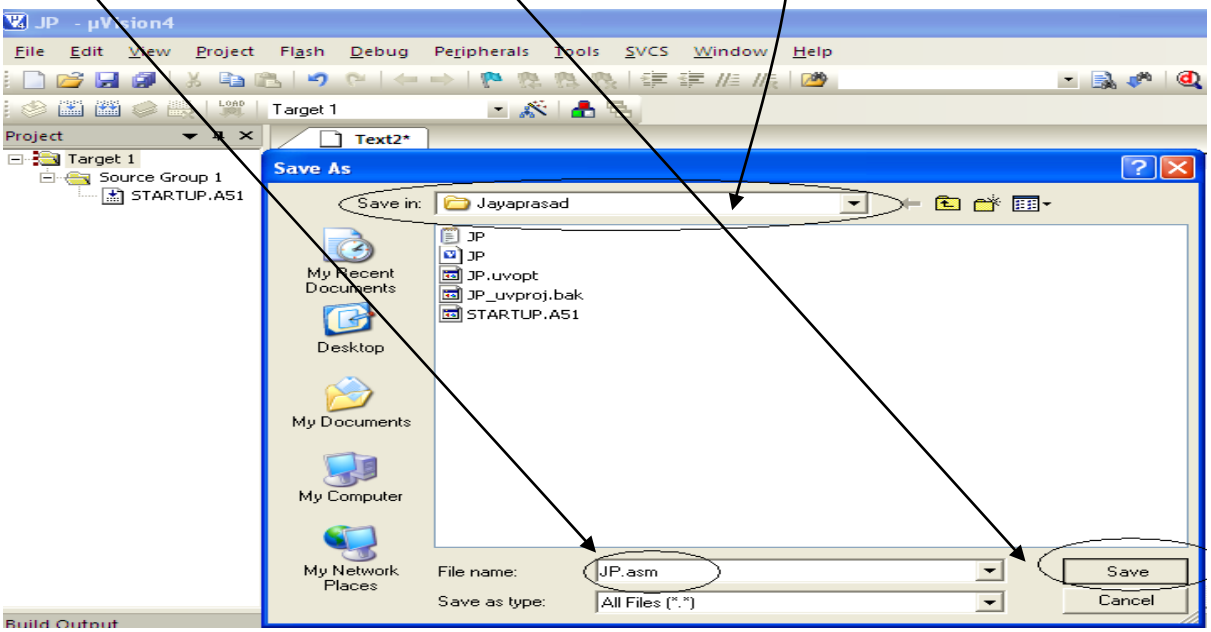


STEP 9: Save your program by going to "File" -> "Save" option.



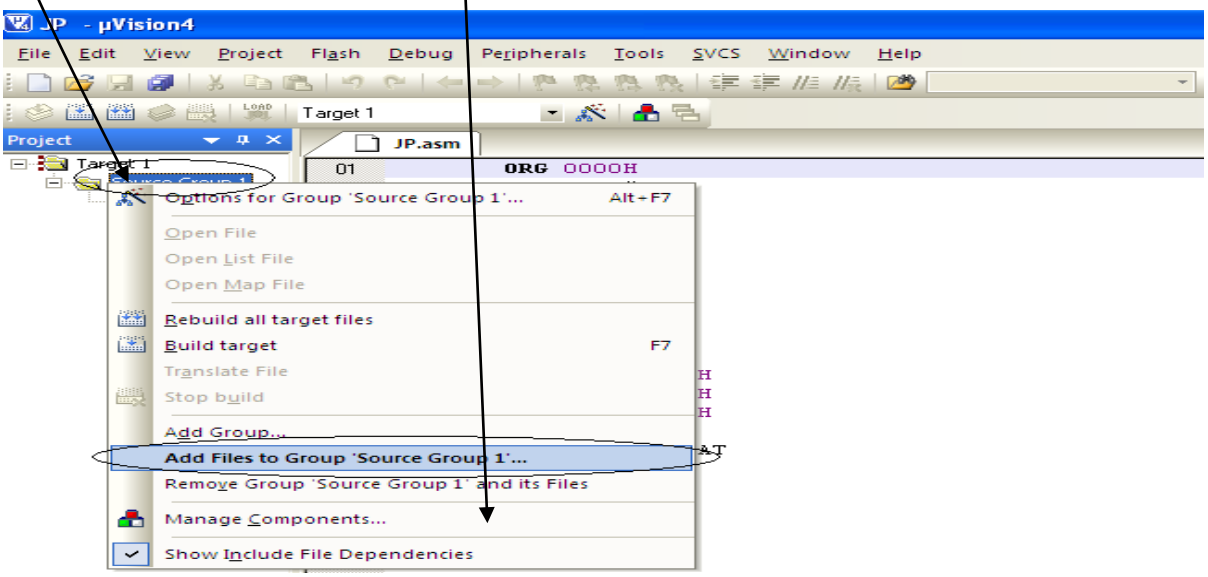
STEP 10:

- "Save in" your project folder.
- Give file name with ".asm" extension.
- And then click on "Save" option.



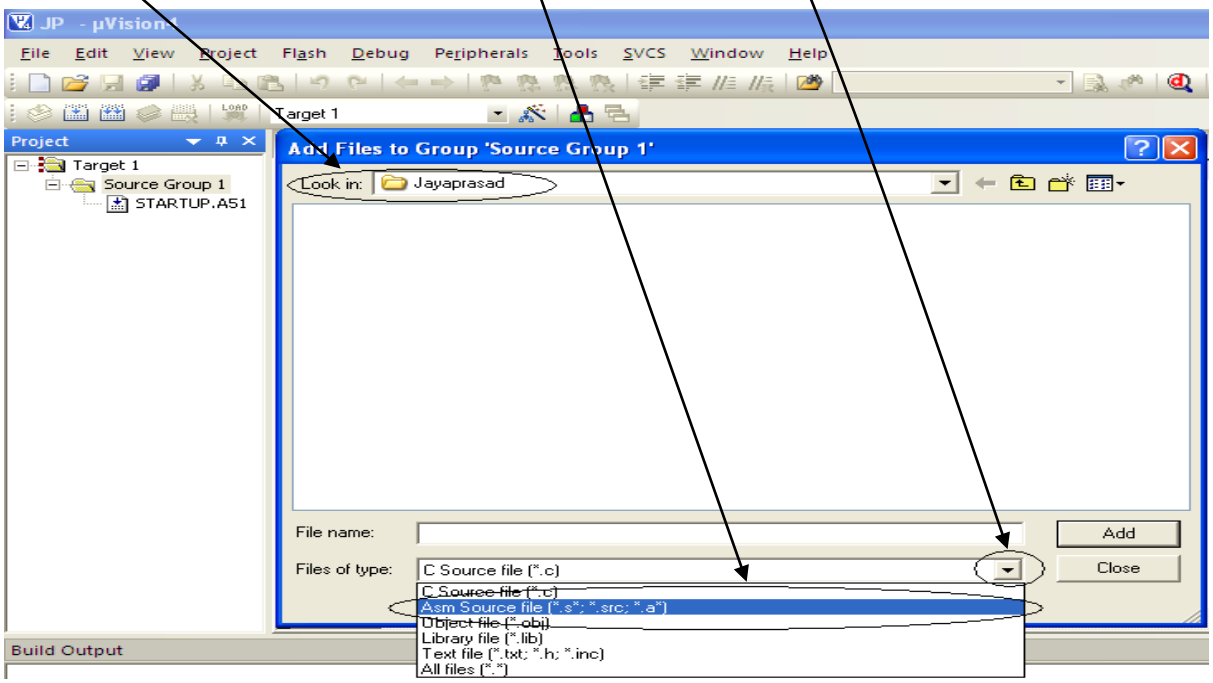
STEP 11:

- Right click on "Source Group1"
- Select "Add Files to „Group Source Group 1"".



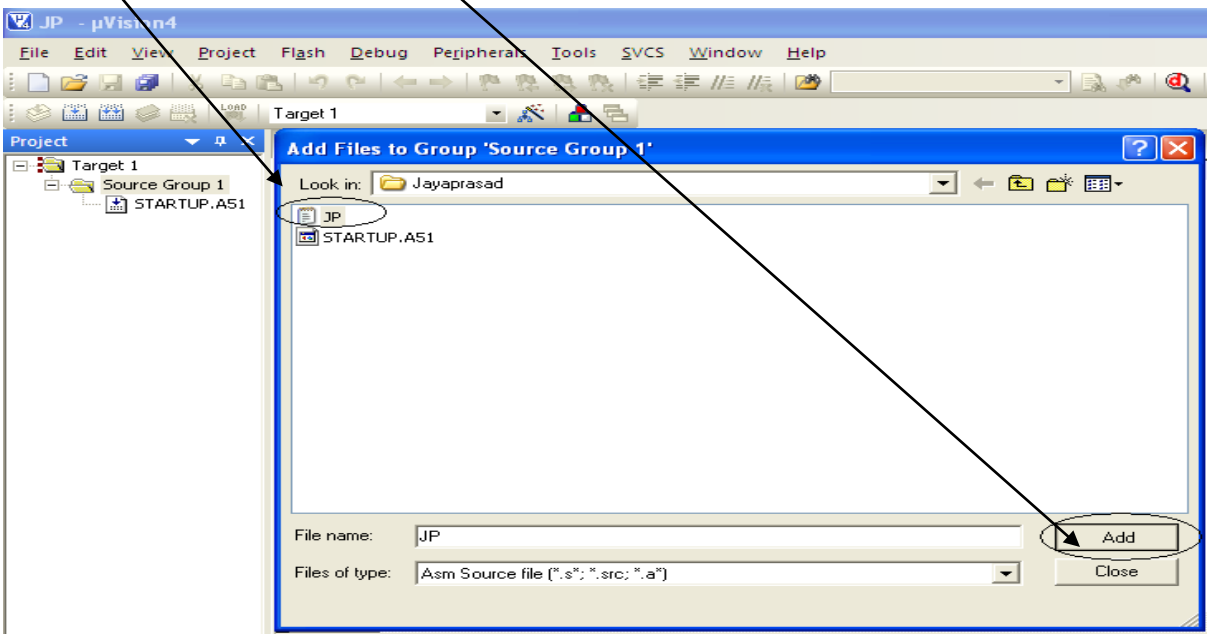
STEP 12:

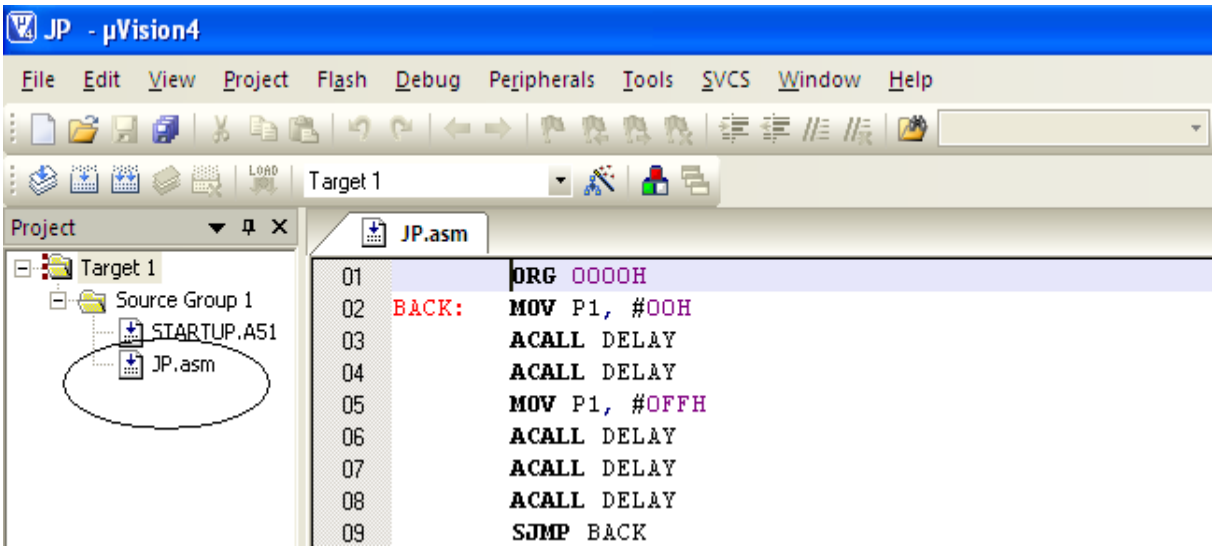
- Select to your Project folder
- Select "Files of type" as "Asm source file" if your program is written in assembly level language or else select "C file" if your program is in C language.



STEP 13:

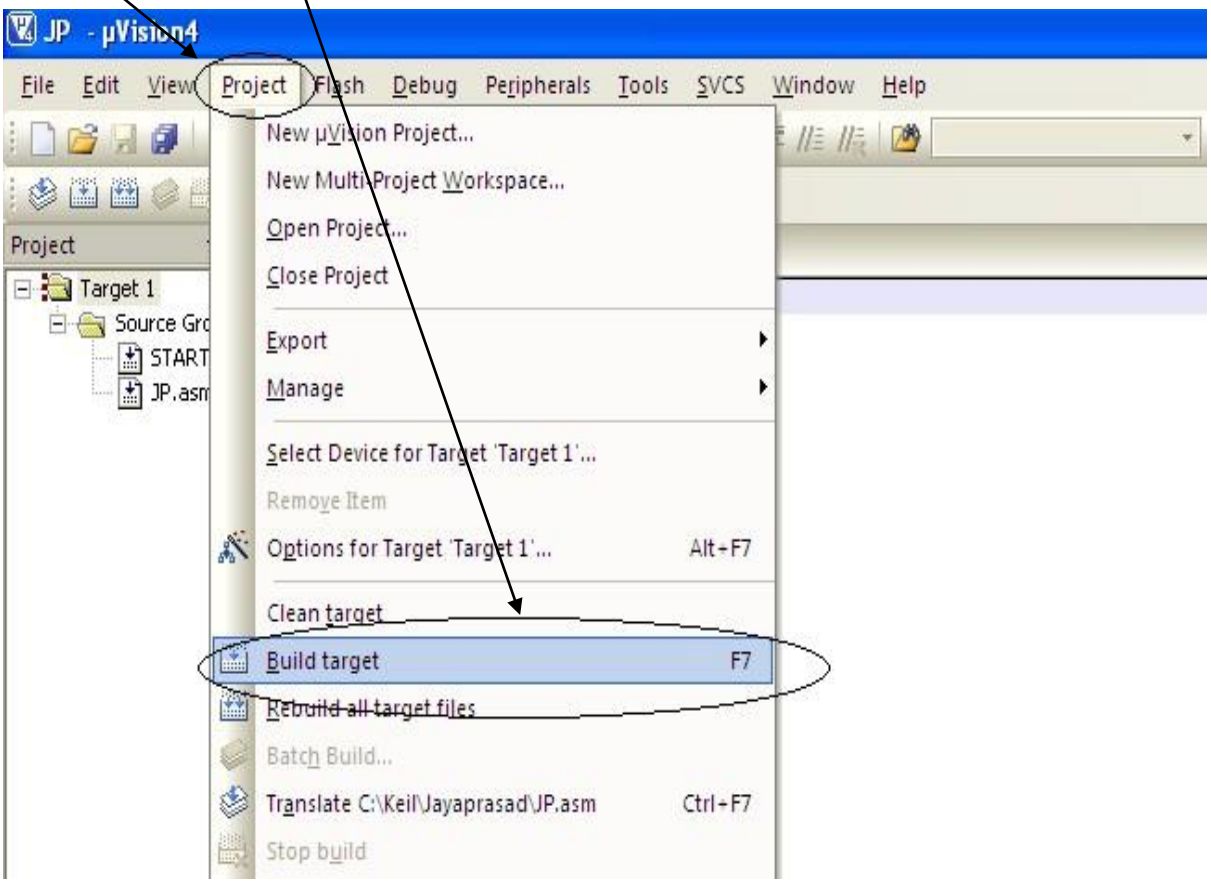
- Select your program file.
- And then click on "Add" to add the file to your source group.
- Notice that your file is added to the Source group



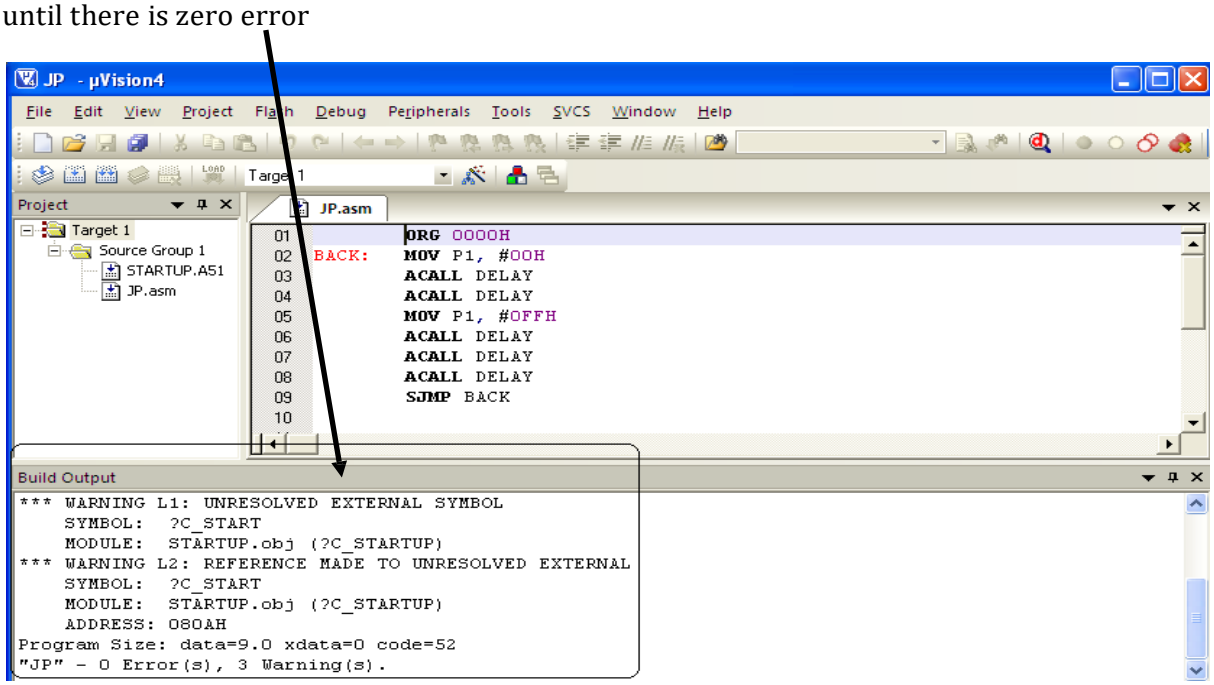


STEP 14: Build the target

- Go to "Project".
- Select "Build Target" or press"F7" key.

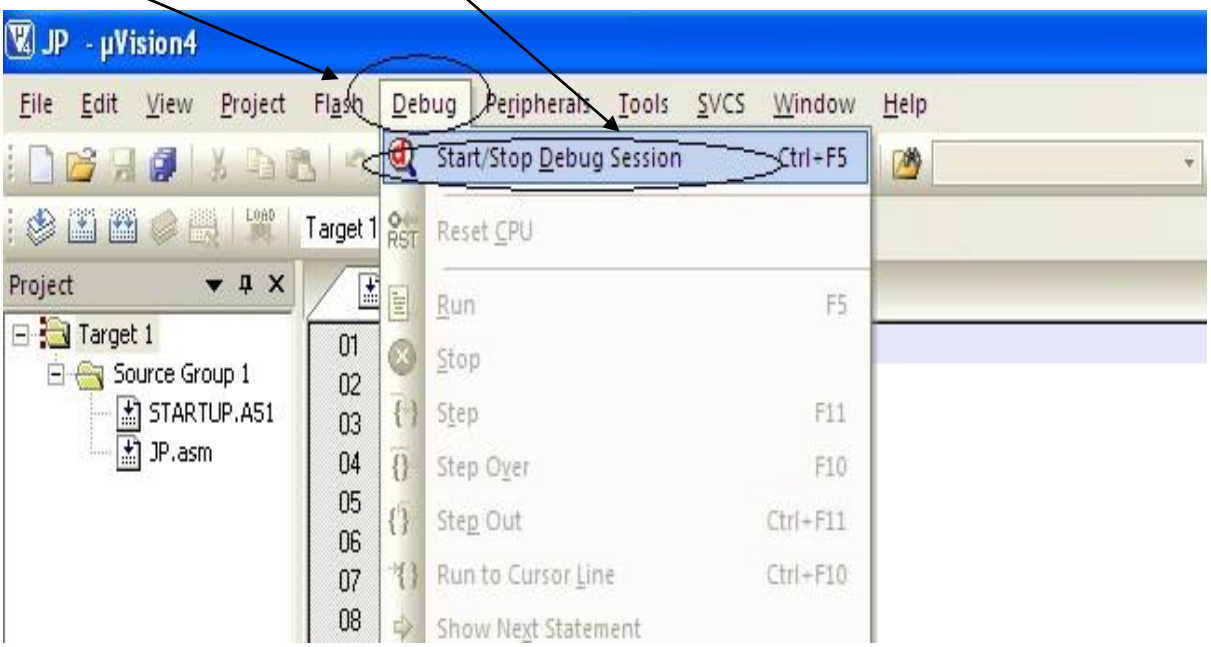


IMPORTANT: After building the target check for the error(s). If there is any error(s) go back to your program, correct the error(s). The output window shows the line where error is found. After correcting the error go back to Step 14 and repeat the processes until there is zero error

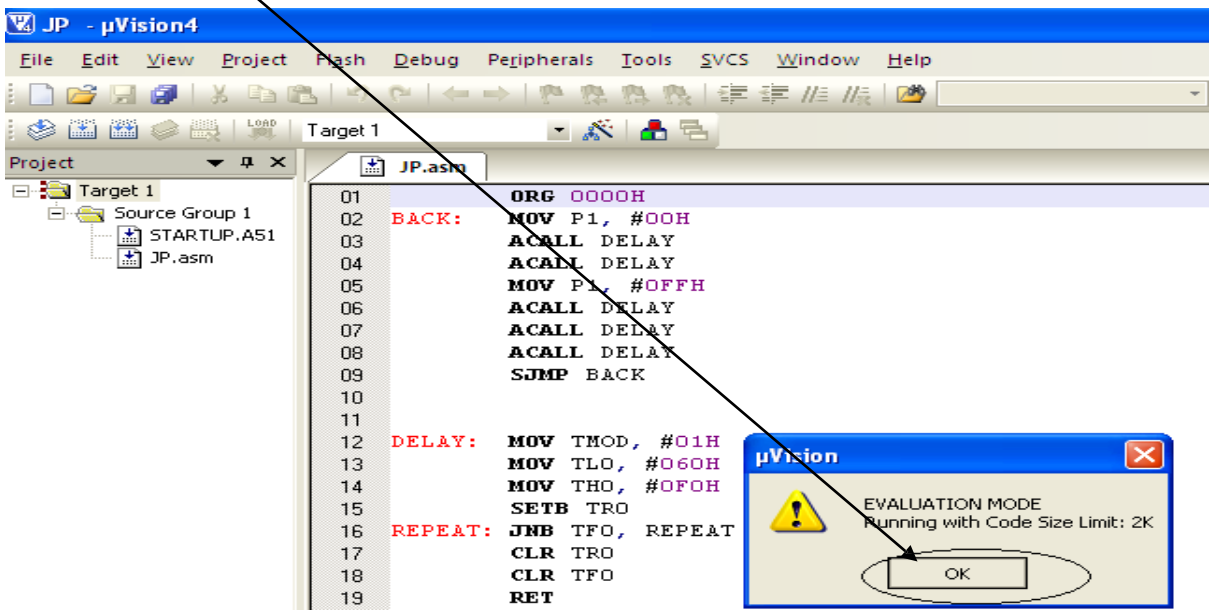


STEP 15: Debugging

- Go to "Debug".
- Select "Start/ Stop Debug Session" or press"Ctrl+F5" key.



- Select "OK".

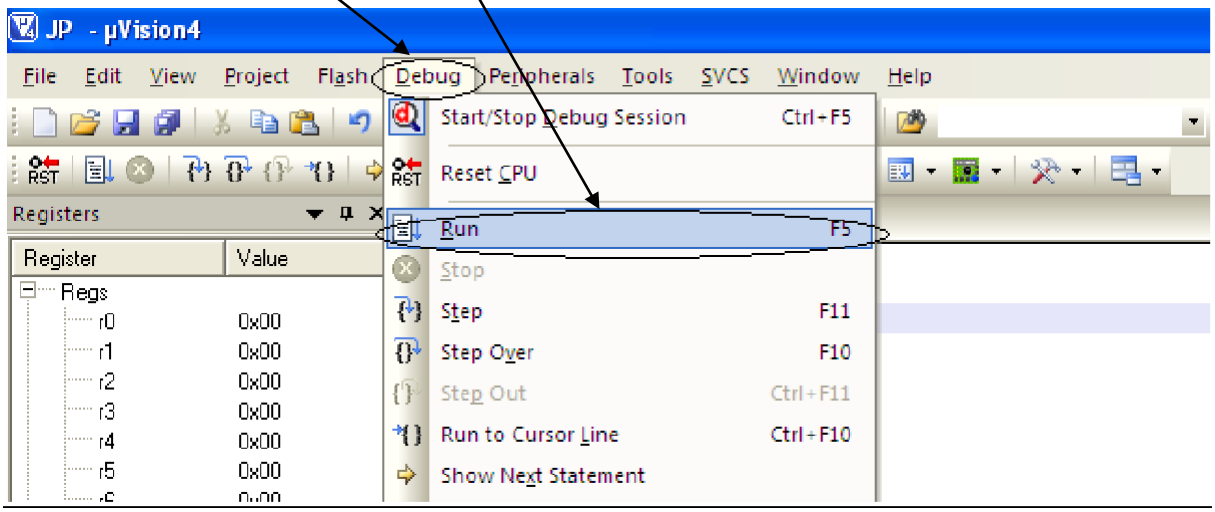


STEP 16: Selecting Output Window

- Choose appropriate Output window (Memory/serial/logic analyzer) according to your program output.
- Type in the input parameters (memory address/ port address/ timer) according to your program.
-

STEP 17: Execution

- Go to "Debug", Select "Run" or press" F5" key for one time execution.
- For single step execution Press11.



Program No.: 1

Write an ALP to move a block of 5 bytes of data from source (20h) to destination (40h) using Internal RAM.

Aim: Program to transfer the block of data from source memory to destination memory.

Brie Explanation: To transfer 5 bytes of data from internal RAM starting from 20h to external memory location starting from 40h

```
        ORG 0000
        MOV R2, #05H        ; initialize the count
        MOV R0, #20H        ; initialize the source memory location
        MOV R1, #40H        ; initialize the destination memory location
BACK:   MOV A,@R0           ; get the data from source memory location address
        MOV @R1, A          ; copy the accumulator content to destination memory
        INC R0              ; increment to next source and destination memory
        INC R1
        DJNZ R2, BACK       ; decrement count. If count! =0 go to label "BACK"
        SJMP $
        END
```

Result:

Before execution

Address	Data	Address	Data
0x20	0x12	0x40	00
0x21	0x24	0x41	00
0x22	0x56	0x42	00
0x23	0xFF	0x43	00
0x24	0xEE	0x44	00

After execution

Address	Data	Address	Data
0x20	0x12	0x40	0x12
0x21	0x24	0x41	0x24
0x22	0x56	0x42	0x56
0x23	0xFF	0x43	0xFF
0x24	0xEE	0x44	0xEE

Program No.: 2

Write an ALP to move a block of N (05) bytes of data from source (2000h) to destination (2050h) using External RAM.

Aim: Program to move the data between two external memory locations.

Brie Explanation : To transfer 8 bytes of data from external memory location starting from 2000h to External memory location starting from 2050h

```
        ORG 0000H
        MOV R0, #05H        ; initialize the count
        MOV R1, #00H        ; initialize the memory block1 location lower byte
        MOV R2, #50H        ; initialize the memory block 2 location lower byte
        MOV R3, #20H        ; initialize the memory block 1&memory block 2 location
                            ; higher byte
BACK:   MOV DPH, R3         ; get the memory block 1 location address to DPTR
        MOV DPL, R1
        MOVX A, @DPTR       ; get the data from memory block 1 to Accumulator
        MOV DPL, R2         ; point the memory block 2 location address to DPTR
        MOVX @DPTR, A       ; copy the accumulator content to memory block 2
        INC R1              ; increment to next source and destination memory location
        INC R2
        DJNZ R0, BACK       ; decrement count. If count! =0 go to label "BACK"
        SJMP $
        END
```

Result:

Before execution

Address	Data	Address	Data
0x2000	0x12	0x2050	00
0x2001	0x24	0x2051	00
0x2002	0x56	0x2052	00
0x2003	0xFF	0x2053	00
0x2004	0xEE	0x2054	00

After execution

Address	Data	Address	Data
0x2000	0x12	0x2050	0x12
0x2001	0x24	0x2051	0x24
0x2002	0x56	0x2052	0x56
0x2003	0xFF	0x2053	0xFF
0x2004	0xEE	0x2054	0xEE

Program No.: 3

Write an ALP To exchange the source block starting with address 20h, (Internal RAM) containing N (05) bytes of data with destination block starting with address 40h (Internal RAM).

Aim: Program to exchange the source block **(Internal RAM)** with destination block **(Internal RAM)**

Brie Explanation: To exchange 5 bytes of data between internal memories location starting from and internal memory location starting from 40h

```
        ORG 0000H
        MOV R2, #05H        ; initialize the count
        MOV R0, #20H        ; initialize the memory block 1 location
        MOV R1, #40H        ; initialize the memory block 2 location
BACK:   MOV A, @R0          ; get the data from memory block 1 to Accumulator
        XCH A, @R1          ; exchange the data between the memory block
        MOV @R0, A          ; copy the accumulator content to memory2
        INC R0              ; increment to next source and destination memory block
        INC R1
        DJNZ R2, BACK       ; decrement count. If count! =0 go to label "BACK"
        SJMP $
        END
```

Result:

Before execution

Address	Data	Address	Data
0x20	0x12	0x40	0x0A
0x21	0x24	0x41	0x0B
0x22	0x56	0x42	0x0C
0x23	0xFF	0x43	0x01
0x24	0xEE	0x44	0x02

After execution

Address	Data	Address	Data
0x20	0x0A	0x40	0x12
0x21	0x0B	0x41	0x24
0x22	0x0C	0x42	0x56
0x23	0x01	0x43	0xFF
0x24	0x02	0x44	0xEE

Program No.: 4

Write an ALP to exchange the source block starting with address 10h (Internal memory), containing N (06) bytes of data with destination block starting at location 2000h (External memory).

Aim: Program to exchange the source block **(Internal memory)** with destination block **(External Memory).**

```
        ORG 0000H
        MOV R2, #06H            ; initialize the count
        MOV R0, #10H            ; initialize the internal memory location
        MOV DPTR, #2000H        ; initialize the external memory location
BACK:   MOVX A,@DPTR            ; get the data from external memory to accumulator
        MOV B, @R0              ; get the data from internal memory to accumulator
        XCH A, B                ; exchange the external and internal memory content
        MOVX @DPTR, A           ; store the data of the internal memory from accumulator to
                                ; External memory
        MOV @ R0, B             ; store the data of the external memory from B register to
                                ; Internal memory
        INC R0                  ; increment the internal memory location
        INC DPTR                ; increment the external memory location
        DJNZ R2, BACK           ; decrement count. If count! =0 go to label "BACK"
        SJMP $
        END
```

Result:

Before execution

Address	Data	Address	Data
0x20	0x12	0x2000	0x0A
0x21	0x24	0x2001	0x0B
0x22	0x56	0x2002	0x0C
0x23	0xFF	0x2003	0x01
0x24	0xEE	0x2004	0x02

After execution

Address	Data	Address	Data
0x20	0x0A	0x2000	0x12
0x21	0x0B	0x2001	0x24
0x22	0x0C	0x2002	0x56
0x23	0x01	0x2003	0xFF
0x24	0x02	0x2004	0xEE

Program No.: 5

Write an 8051 ALP to add the byte in the RAM at 34h and 35h, store the result in the register R5 (LSB) and R6 (MSB), using Indirect Addressing Mode.

Aim: Program to add the Number

```
ORG 0000H            ; Start of the program
MOV R0, #34H         ; Load R0 with the address of 34h in RAM
MOV  A, @R0          ; Move the byte at address 34h to accumulator
INC R0               ; Increment R0 to point to the next byte (35h)
MOV R6, #00H         ; Clear the R6 to store the generated carry after the addition
ADD A, @R0           ; add the contents bytes
JNC DOWN             ; if the carry is not generated store the LSB of the sum.
INC R6               ; if carry is generated store MSB of the sum in 35h
DOWN: MOV R5, A      ; Store the LSB of the result in R5 (LSB)
SJMP $               ; End of the program (infinite loop)
END                  ; End of the program code
```

Result:

Before execution

Address/ REG	Data
0x34	0x12
0x35	0x28
R5	0x00
R6	0x00

After execution

Address/ REG	Data
0x34	0x12
0x35	0x28
R5	0x3A
R6	0x00

Program No.:6

Write an 8051 ALP to subtract the bytes in Internal RAM 34h & 35h store the result in register R5 (LSB) & R6 (MSB).

Aim: Program to subtract the Number

```
        ORG 0000H          ; Start of the program
        MOV R0, #34H       ; Load R0 with the address of 34h in RAM
        MOV  A, @R0        ; Move the byte at address 34h to accumulator
        INC R0             ; Increment R0 to point to the next byte (35h)
        MOV R6, #00H       ; Clear the R6 to store the generated carry after the addition
        SUBB A, @R0        ; Perform the subtraction operation
        JNC DOWN           ;if the borrow is not taken store the LSB of the Difference
        DEC R6             ; if borrow is taken store MSB of the difference in 35h
DOWN:   MOV R5, A          ; Store the LSB of the difference in R5 (LSB)
        SJMP $             ; End of the program (infinite loop)
        END                ; End of the program code
```

Result:

Before execution

Address/ REG	Data
0x34	0x32
0x35	0x42
R5	0x00
R6	0x00

After execution

Address/ REG	Data
0x34	0x32
0x35	0x42
R5	0xF0
R6	0xFF

Program No.:7

Write an 8051 ALP to multiply two 8-bit numbers stored at 30h and 31h and Store16- bit result in 32h and 33h of Internal RAM.

Aim: Program to perform Multiplication operation.

```
ORG 0H          ; Start of the program
MOV A, 30H      ; Move the content of memory location 30h to accumulator (1st number)
MOV B, 31H      ; Move the content of memory location 31h to register B (2nd number)
MUL AB          ; Multiply the contents of accumulator and register B
MOV R2, A       ; Store the LSB of the result in register R2
MOV R3, B       ; Store the MSB of the result in register R3
MOV 32H, R2     ; Store the LSB of the result in memory location 32h
MOV 33H, R3     ; Store the MSB of the result in memory location 33h
SJMP $          ; End of the program (infinite loop)
END             ; End of the program code
```

Result:

Before execution

Address/ REG	Data
0x30	0xFF
0x31	0xFF
R2	0x00
R3	0x00

After execution

Address/ REG	Data
0x30	0xFF
0x31	0xFF
R2	0x01
R3	0xFE

Program No.:8

Write an 8051 ALP to perform division operation on 8-bit number by 8-bit number.

Aim: Program to perform division operation

```
ORG 0H          ; Start of the program
MOV A, 30H      ; Move the dividend from memory location 30h to accumulator
MOV B, 31H      ; Move the divisor from memory location 31h to register B
DIV AB          ; Divide accumulator (dividend) by register B (divisor)
MOV 32H, A      ; Store the quotient in memory location 32h
MOV 33H, B      ; Store the remainder in memory location 33h
SJMP $          ; End of the program (infinite loop)
END             ; End of the program code
```

Result:

Before execution		After execution	
Address/ REG	**Data**	**Address/ REG**	**Data**
0x30	0xF8	0x30	0xF8
0x31	0x07	0x31	0x07
A	0x00	A	0x23
B	0x00	B	0x03

Program No.:9

Write an 8051 ALP to separate positive and negative in a given array

Aim: Program to separate positive and negative numbers in a given array

```
        ORG 0H                  ; Start of the Program
        MOV R0, #20H            ; Initialize R0 to Hold the Count of Positive Numbers
        MOV R1, #10H            ; Initialize R1 to Hold the Count of Negative Numbers
        MOV R2, #05H            ; Initialize R2 to Hold the Size of the Array
        CLR C                   ; Clear the Carry Flag Status
        MOV DPTR, #9000H        ; Initialize Dptr to Hold the Starting Address Of the array
BACK:   MOVX A,@DPTR            ; Read the Data from the Memory to Accumulator
        MOV B,A
        RLC A                   ; Check the Sign Bit (Most Significant Bit) Of the Given Data
        JC NEG                  ; If the Given Number Is Negative Update the Count
        MOV @R0, B
        INC R0                  ; If the Given Number Is Positive Update the Positive No.
                                ; Count Size.
        SJMP NEXT               ; Repeat the Steps till the Count Reaches To 0
NEG:    MOV @R1,B       ; If The Given Number Is Negative Update The Negative No. Count
        INC R1                  ; Read the Next Data.
        NEXT:INC DPTR           ;Read The Next Data.
        DJNZ R2, BACK           ; Repeat the Steps till The Array Size Reaches To 0
        SJMP $                  ; End Of the Program (Infinite Loop)
        END
```

Result:

Before execution

Address	Data	Address	Data	Address	Data
0x9000	0x48	0x20	0x00	0x10	0x00
0x9001	0x97	0x21	0x00	0x11	0x00
0x9002	0x68	0x22	0x00	0x12	0x00
0x9003	0x82	0x23	0x00	0x13	0x00
0x9004	0x91	0x24	0x00	0x14	0x00

After execution

Address	Data	Address	Data	Address	Data
0x9000	0x48	0x20	0x48	0x10	0x97
0x9001	0x97	0x21	0x68	0x11	0x82
0x9002	0x68	0x22	0x00	0x12	0x91
0x9003	0x82	0x23	0x00	0x13	0x00
0x9004	0x91	0x24	0x00	0x14	0x00

Program No.:10

Write an 8051 ALP to separate even or odd elements in a given array

Aim: Program to separate even or odd numbers in a given array

```
        ORG 0H                  ; Start of the Program
        MOV R0, #20H            ; Initialize R0 to Hold the Count of Even Numbers
        MOV R1, #10H            ; Initialize R1 to Hold the Count of Odd Numbers
        MOV R2, #07H            ; Initialize R2 to Hold the Size of the Array
        CLR C                   ; Clear the Carry Flag Status
        MOV DPTR, #9000H        ; Initialize Dptr to Hold the Starting Address Of the array
BACK:   MOVX A,@DPTR            ; Read the Data from the Memory to Accumulator
        MOV B,A
        RLC A                   ; Check the Least Significant Bit Of the Given Data
        JC ODD                  ; If the Given Number Is odd Update the Count
        MOV @R0, B
        INC R0                  ; If the Given Number Is even Update the Count Size.
        SJMP NEXT               ; Repeat the Steps till the Count Reaches To 0
ODD:    MOV @R1,B               ; If The Given Number Is Negative Update The Negative No. Count
        INC R1                  ; Read the Next Data.
        NEXT:INC DPTR           ;Read The Next Data.
        DJNZ R2, BACK           ; Repeat the Steps till the Array Size Reaches To 0
        SJMP $                  ; End Of the Program (Infinite Loop)
        END
```

Result:

Before execution

Address	Data	Address	Data	Address	Data
0x9000	0x48	0x20	0x00	0x10	0x00
0x9001	0x97	0x21	0x00	0x11	0x00
0x9002	0x68	0x22	0x00	0x12	0x00
0x9003	0x82	0x23	0x00	0x13	0x00
0x9004	0x91	0x24	0x00	0x14	0x00

After execution

Address	Data	Address	Data	Address	Data
0x9000	0x48	0x20	0x48	0x10	0x97
0x9001	0x97	0x21	0x68	0x11	0x91
0x9002	0x68	0x22	0x82	0x12	0x00
0x9003	0x82	0x23	0x00	0x13	0x00
0x9004	0x91	0x24	0x00	0x14	0x00

Program No.: 11a.

Aim: Program to arrange the data in given array in ascending order

Program: The array of data which has to be arranged in the ascending order starts from 5100h external memory location. The array contains 5 data‟s. Rearrange the data in the ascending order

```
        ORG 0000H
        MOV R1, #04H          ; initialize the step count (outer loop)
L1:     MOV A, R1             ; move the count to accumulator
        MOV R2, A             ; move accumulator content to R2 (comparison) (inner loop)
        MOV DPTR, #5100H      ; Initialize the external memory location
L2:     MOVX A,@DPTR          ; get the data from memory to accumulator
        MOV B, A              ; move the accumulator content to B register
        INC DPTR              ; increment the external memory location.
        MOVX A, @DPTR         ; get the data from memory to accumulator
        CJNE A, B, L3         ; compare accumulator content and B register content, if not
                                equal Jump to label „L3‟
        SJMP L5               ; short jump to label L5
L3:     JC L4                 ; If A & B are not equal, then check CY = 1(A<B)
                              ; and if so jump to label „L4‟
        SJMP L5               ; short jump to label L5
L4:     XCH A, B              ; Exchange A & B
        MOVX @DPTR, A         ; move accumulator content to external memory
        DEC DPL               ; decrement the lower byte of external memory
        XCH A, B              ; Exchange A & B
        MOVX @DPTR, A         ; move accumulator content to external memory
        INC DPTR              ; increment the external memory location
L5:     DJNZ R2, L2           ; decrement comparison count, if count! = 0 then jump to
                              ; label L2‟.
        DJNZ R1, L1           ; decrement step count, if count! = 0 then jump to  label „L1‟
        SJMP $
        END
```

Result:

Before execution:

Address	Data
0x5100	0x1F
0x5101	0xD4
0x5102	0x56
0x5103	0xFF
0x5104	0x01

After execution:

Address	Data
0x5100	0x01
0x5101	0x1F
0x5102	0x56
0x5103	0xD4
0x5104	0xFF

Program No.: 11b.

Aim: Program to arrange the data in given array in descending order

Program: The array of data which has to be arranged in the descending order starts from 5100h external memory location. The array contains 5 data‟s. Rearrange the data in the ascending order

```
        ORG 0000H
        MOV R1, #04H          ; initialize the step count
L1:     MOV A, R1             ; move the count to accumulator
        MOV R2, A             ; move accumulator content to R2 (comparison)
        MOV DPTR, #5100H      ; Initialize external memory location
L2:     MOVX A,@DPTR          ; get the data from memory to accumulator
        MOV B, A              ; move the accumulator content to B register.
        INC DPTR              ; increment the external memory location.
        MOVX A, @DPTR         ; get the data from memory to accumulator
        CJNE A, B, L3         ;compare accumulator content and B register content, if not
                                equal Jump to label „L3‟
        SJMP L5               ; short jump to label L5
L3:     JNC L4                ; If A & B are not equal, then check CY = 1(A<B)
                              ; If CY! = 1(A>B) jump to label „L4‟
        SJMP L5               ; short jump to label L5
L4:     XCH A, B              ; If CY! = 1, Exchange A & B
        MOVX @DPTR, A         ; move the data from accumulator to external memory
        DEC DPL               ; decrement the lower byte of external memory
        XCH A, B              ; Exchange A & B
        MOVX @DPTR, A         ; move accumulator content to external memory
        INC DPTR              ; increment the external memory location
L5:     DJNZ R2, L2           ; decrement comparison count, if count! =0 then jump to
                              ; label‟ L2‟.
        DJNZ R1, L1           ; decrement step count, if count! =0 then jump to  label „L1‟
        SJMP $
        END
```

Result:

Before execution:

Address	Data
0x5100	0x1F
0x5101	0xD4
0x5102	0x56
0x5103	0xFF
0x5104	0x01

After execution:

Address	Data
0x5100	0xFF
0x5101	0xD4
0x5102	0x56
0x5103	0X1F
0x5104	0x01

Program No.: 12a

Aim: Program to find the largest number in a given array

Program: To find the largest number in a given array of size 5 starting from 5100h external memory location. The largest number has to be stored in 8100h external memory location.

```
        ORG 0000H
        MOV R1, #04H          ; initialize the count
        MOV DPTR, #5100H      ; initialize the external memory location
        MOVX A,@DPTR          ; get the data from memory to accumulator
BACK:   MOV B, A              ; move the content from accumulator to B register
        INC DPTR              ; increment the external memory location
        MOVX A, @DPTR         ; get the data from memory to accumulator
        CJNE A, B, NEXT       ; compare accumulator content and B register content, if not
                                equal Jump to label „NEXT"
        DJNZ R1, BACK         ; if A & B are equal, then decrement count, if count! =0
                                Jump to label „BACK"
        SJMP LAST             ; If count=0, then short jump to label" LAST"
NEXT:   JNC L2                ; If A & B are not equal, then check CY=1(A<B)
                              ;  If CY! =1(A>B) jump to label „L2"
        XCH A, B              ; If CY=1, Exchange A & B
L2:     DJNZ R1, BACK         ; Decrement count, if count! =0, jump to label," BACK"
LAST:   MOV DPTR, #8100H      ; Initialize new memory location for storing largest data
        MOVX @DPTR, A         ; move the largest data from accumulator to new memory
        SJMP $
        END
```

Result:

Before execution

Address	Data	Address	Data
0x5100	0x12	0x8100	0x00
0x5101	0x24		
0x5102	0x56		
0x5103	0xFF		
0x5104	0xEE		

After execution

Address	Data	Address	Data
0x5100	0x12	0x8100	0xFF
0x5101	0x24		
0x5102	0x56		
0x5103	0xFF		
0x5104	0xEE		

Program No.: 12b

Aim: Program to find the smallest number in a given array

Program: To find the smallest number in a given array of size 5 starting from 5100h external memory location. The largest number has to be stored in 8100h external memory location.

```
        ORG 0000H
        MOV R1, #04H          ; initialize the count
        MOV DPTR, #5100H      ; initialize the external memory location
        MOVX A,@DPTR          ; get the data from memory to accumulator
BACK:   MOV B, A              ; move the content from accumulator to B register
        INC DPTR              ; increment the external memory location
        MOVX A,@DPTR          ; get the data from memory to accumulator
        CJNE A, B, NEXT       ; compare accumulator content and B register content, if not
                                equal Jump to label „NEXT"
        DJNZ R1, BACK         ; if A & B are equal, then decrement count, if count! =0
                                Jump to label „BACK"
        SJMP LAST             ; If count=0, then short jump to label" LAST"
NEXT:   JC L2                 ; If A & B are not equal, then check for CY= 1(A<B)
                              ; and if so jump to label „L2"
        XCH A, B              ; else if CY! =1, exchange A & B
L2:     DJNZ R1, BACK         ; Decrement count, if count! = 0, jump to label," BACK"
LAST:   MOV DPTR, #8100H      ; Initialize new memory location for storing smallest data
        MOVX @DPTR, A         ; move the smallest data from accumulator to new memory
        SJMP $
        END
```

Result:

Before execution

Address	Data	Address	Data
0x5100	0x12	0x8100	0x00
0x5101	0x24		
0x5102	0x56		
0x5103	0xFF		
0x5104	0xEE		

After execution

Address	Data	Address	Data
0x5100	0x12	0x8100	0x12
0x5101	0x24		
0x5102	0x56		
0x5103	0xFF		
0x5104	0xEE		

Program No.: 13

Write an ALP for Decimal UP-Counter.

Aim: Program to verify the working operations of Decimal UP-Counter.

```
        ORG 000H
            MOV A,#00H          ; get the first BCD value to accumulator
        BACK:MOV  P0,A     ; display the count in P0
         ACALL DELAY            ; call the delay
        ADD  A,#01H             increment the count
         DA A                   ; decimal adjust the count
        SJMP BACK
DELAY: MOV R1,#0FFH
        L1: MOV R2,#0FFH
        L2: MOV R3,#0FFH
     L3:DJNZ R3,L3
        DJNZ  R2,L2
        DJNZ  R1,L1
        RET
        END
```

Result:

After execution of the program on port0 the decimal up count from 00 to 99 is displayed continuously.

Program No.: 14

Write an ALP for Decimal Down Counter.

Aim: Program to verify the working operations of Decimal down Counter.

```
        ORG 000H
        MOV A,#99H      ; get the first BCD value to accumulator
BACK:   MOV P0,A        ; display the count in P0
        ACALL DELAY     ; call the delay
        SUBB  A ,#99H   ; get the next BCD down count value
        DA A            ; Display the result in Decimal form
        SJMP BACK
DELAY:  MOV R1,#0FFH
L1:     MOV R2,#0FFH
L2:     MOV R3,#0FFH
L3:     DJNZ R3,L3
        DJNZ  R2,L2
        DJNZ  R1,L1
        RET
        END
```

Result:

After execution of the program on port0 the decimal down count from 99 to 00 is displayed continuously.

Program No.: 15

Write an ALP for Hexadecimal UP Counter.

Aim: Program to verify the working operations of Hexadecimal UP Counter

```
        ORG 000H
        MOV A,#00H          ; get the first BCD value to accumulator
BACK:   MOV P0,A            ; display the count in P0
        ACALL DELAY         ; call the delay
        ADD  A,#01H         ;increment the count
        SJMP BACK
DELAY:  MOV R1,#0FFH
L1:     MOV R2,#0FFH
L2:     MOV R3,#0FFH
L3:     DJNZ R3,L3
        DJNZ  R2,L2
        DJNZ  R1,L1
        RET
        END
```

Result:

After execution of the program on port0 the hexadecimal up count from 00 to 99 is displayed continuously.

Program No.: 16

Write an ALP for Hexadecimal Down Counter.

Aim: Program to verify the working operations of Hexadecimal Down Counter

```
        ORG 000H
        MOV A,#0FFH         ; get the first BCD value to accumulator
BACK:MOV P0,A       ; display the count in P0
        ACALL DELAY ;   call   the   delay
        SUBB A,#01H             ;decrement the count
        SJMP BACK
DELAY: MOV R1,#0FFH
        L1: MOV R2,#0FFH
        L2: MOV R3,#0FFH
    L3:DJNZ R3,L3
        DJNZ R2,L2
        DJNZ R1,L1
        RET
        END
```

Result:

After execution of the program on port0 the hexadecimal down count from 99 to 00 is displayed continuously.

II. C Programming

Program No.: 17

Write an 8051 C program to find the sum of first 10 Integer Numbers.

AIM: To find the sum of first 10 Integer Numbers.

```
#include<reg51.h>
void main ( )
{
unsigned char sum = 0, i;
for (i=0; i<=10; i++)
{
sum +=i;

}
ACC=sum;
P0=sum;
}
```

Result:

After execution of the program on port0 the sum of first 10 Integer Numbers 37h is displayed.

Program No.: 18

Write an 8051 C program to find the Factorial of a given number.

AIM: To find the sum of first 10 Integer Numbers.

```
#include<reg51.h>
#include<stdio.h>

void main ( )

{

unsigned int  i;

unsigned char num = 7; // Number to find the factorial

unsigned long factorial = 1;

//calculate the factorial of the number

for (i=1; i<=num; i++)

{

factorial = factorial*i;

}

//output the result

P1= factorial;

P2= (factorial & 0xff00)  >>8;

}
```

Result:

After execution of the program on port1 & port2 the Factorial of a given number is displayed.

Program No.: 19

Write an 8051 C program to find the Square of a number (1 to 10) using Look-Up Table.

AIM: To find the Square of a number (1 to 10).

```
#include<reg51.h>
void main ( )
{
unsigned char LUT[]={1,4,9,16,25,36,49,64,81,100};
unsigned char num, square;

for (num =1; num<11; num ++)
{
square = LUT[num-1];
P0 = square;
}
while(1);
}
```

Result:

After execution of the program the square of number from 0 to 10 will be displayed on Port 0 continuously.

Program No.: 20

Write an 8051 C program to count the number of Ones and Zeros in two consecutive memory locations.

AIM: To count the number of Ones and Zeros in two consecutive memory locations.

```
#include<reg51.h>
void main ( )
{
unsigned char array[]={0x57,0xfc};
unsigned char i,ones,zeros;

for (i=0;i<8;i++)
{
array[0]>>=1;
if(CY==1) ones++;
else zeros++;
}
for(i=0;i<8;i++)
{
array[1]>>=1;
if(CY==1) ones++;
else zeros++;
}

P0 = zeros;
P1 = ones;
while(1);
}
```

Result:

After execution of the program on port 0 & port1 the number of Zeros and ones will be stored respectively.

II. Hardware Interfacing Programming

STEPPER CONTROL INTERFACE TO 8051

OBJECTIVE: this program demonstrates a stepper motor control using an h-bridge controller. direction and speed are selected with hyper terminal.

Hardware: Microcontroller 89s51

Crystal Freq: 11.0592MHz

I/O Port configuration: Port 1 output Stepper motor

THEORY: A Stepper Motor or a step motor is a brushless, synchronous motor which divides a full rotation into a number of steps. Unlike a brushless DC motor which rotates continuously when a fixed DC voltage is applied to it, a step motor rotates in discrete step angles as shown in the Figure 1.

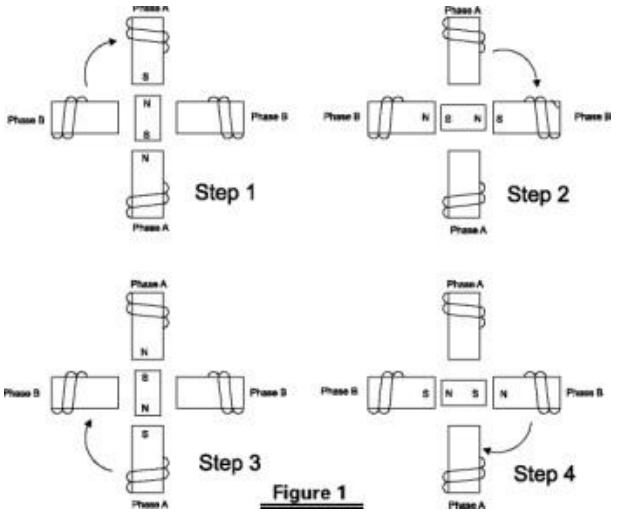


Figure 1

The Stepper Motors therefore are manufactured with steps per revolution of 12, 24, 72, 144, 180, and 200, resulting in stepping angles of 30, 15, 5, 2.5, 2, and 1.8 degrees per step. The stepper motor can be controlled with or without feedback. Stepper motors work on the principle of electromagnetism. There is a soft iron or magnetic rotor shaft surrounded by the electromagnetic stators. The rotor and stator have poles which may be teethed or not depending upon the type of stepper. When the stators are energized the rotor moves to align itself along with the stator (in case of a permanent magnet type stepper) or moves to have a minimum gap with the stator (in case of a variable reluctance stepper). This way the stators are energized in a sequence to rotate the stepper motor.

Program No.: 21

Write an 8051 C program to rotate stepper motor.

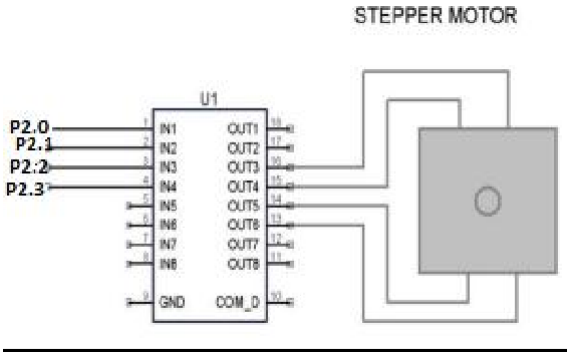


Figure 2: Stepper Motor Interfacing circuit

21a. To rotate a stepper motor in clockwise directions

Program No.: 21a

Aim: Program to interface Stepper motor

Program: To rotate Stepper Motor Clockwise

```
#include <at89c51xd2.h>

Void delay (void);

Void main (void)
{
while(1)
{
P2=0x07; // output 0x07 to port P2
delay(); // generate delay

P2=0x0b; // output 0x0b to port P2
delay(); // generate delay

P2=0x0d; // output 0x0d to port P2
delay(); // generate delay

P2=0x0e; // output 0x0e to port P2
delay(); // generate delay
}
}

void delay(void)
{
int i;
for (i=0;i<=30000;i++);
}
```

Result:

Interfaced Stepper motor and rotated Stepper Motor in Clockwise direction.

Program No.:21b

Aim: Program to interface Stepper motor

Program: To rotate Stepper Motor Anti-Clockwise

```
#include <at89c51xd2.h>

void delay(void);

void main(void)
{
while(1)
{
P2=0x0e; // output 0x0e to port P2
delay(); // generate delay

P2=0x0d; // output 0x0d to port P2
delay(); // generate delay

P2=0x0b; // output 0x0b to port P2
delay(); // generate delay

P2=0x07; // output 0x07 to port P2
delay(); // generate delay
}
}

void delay(void)
{
int i;
for(i=0;i<=30000;i++);
}
```

Result:

Interfaced Stepper motor and rotated Stepper Motor in anti-clockwise direction.

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 is as shown in the below Figure 1. The digital values 00H-FFH corresponds to the analog voltage 0-5V. The 8-bit data can be fed into DAC from Port P1 of 8051 microcontroller.

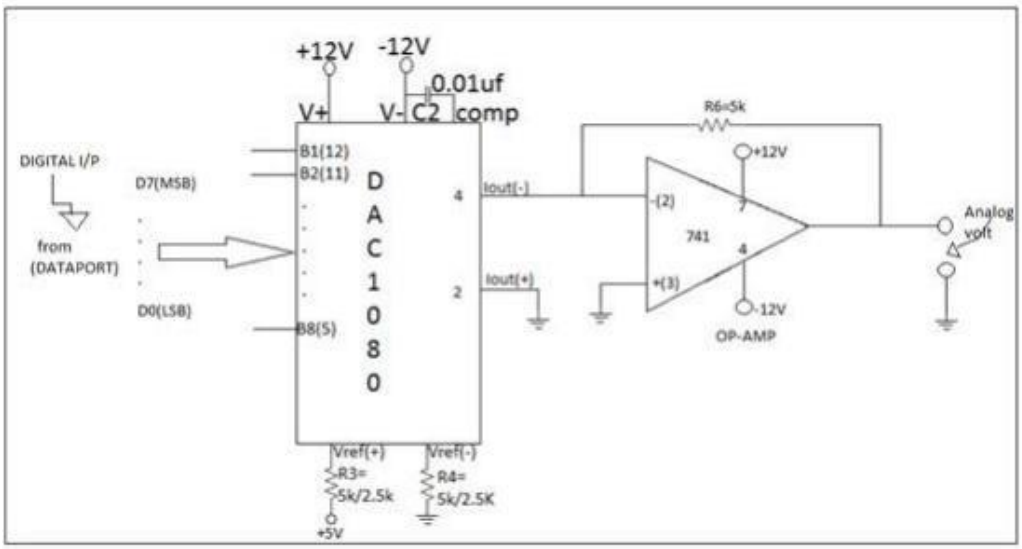


Figure 1: Digital to Analog Converter (DAC) interface

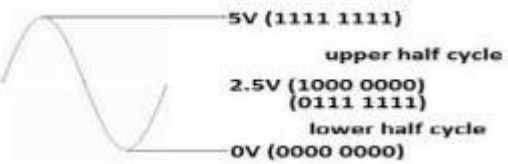


Figure 2: Sine waveform generation.

The digital formula used to get the values for positive cycle of the sine waveform is as follows: 127 sinΘ + 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 No.: 22

Write an 8051 C program to Generate Square & Sine waveforms using DAC interface.

OBJECTIVE: This program implements the basic wave form generation using DAC. Output is displayed on a CRO.

Hardware: Microcontroller 89s51

Crystal Freq 11.0592MHz

I/O Port configuration Port output DAC

22a.Square wave generation

SQUARE WAVE OF 50% DUTY CYCLE

Duty cycle =50% =TON / (TON+TOFF)

Here TON =TOFF

ALGORITHM:

Step1: move FF (analog voltage 10v)to DAC input

Step2: move 00 (analog voltage 0v) to DAC input and go to step1

```
#include<at89c51xd2.h>
void delay(void);
void main()
{
while(1)
{
P1=0x00;
delay();
P1=0x0FF;
delay();
}
}

void delay(void)
{
unsigned int i;
for(i=0;i<10000;i++);
}
```

Outcome: By using DAC interfacing the square waveform has been generated on the CRO.

22b. Sine wave generation

```
#include<at89c51xd2.h>
#include<math.h>
unsigned char arr[62];
float x;
unsigned int i=0;
```

```
void main()
{
P1=0xFF;
for (x = 0; x < (2 * 3.1415); x += 0.1)
{
arr[i]=127+127 * sin(x);
i++;
}
P1=0X00;
while(1)
{
for (i=0;i<62;i++)
{
P1=arr[i];
}
}
}
```

22b.Sine_wave

Alternative code for generation of sine waveform

```
#include<at89c51xd2.h>
 sfr dacdata=0x90;
 void main(void)
{
unsigned char sine_value[12]={128, 192,238,255,238,192,128,64,17,0,17,64};
unsigned int x;
while(1)
{
for(x=0;x<12;x++)
dacdata=sine_value[x];
}
}
```

Outcome: By using DAC interfacing the sine waveform has been generated on the CRO..