VTU SYLLABUS

MICROCONTROLLERS LAB

CYCLE I: PROGRAMMING

1. Data Transfer - Block move, Exchange, Sorting, Finding largest element in an array
2. Arithmetic Instructions - Addition/subtraction, multiplication and division, square, Cube (16 bits Arithmetic operations – bit addressable)
3. Counters
4. Boolean & Logical Instructions (Bit manipulations)
5. Conditional CALL & RETURN
6. Code conversion: BCD – ASCII; ASCII – Decimal; Decimal – ASCII; HEX - Decimal and Decimal – HEX
7. Programs to generate delay, Programs using serial port and on-Chip timer /counter

CYCLE II. INTERFACING

CYCLE II.A

Write programs to interface 8051 chip to Interfacing modules to develop single chip solutions.

1. Interface a simple toggle switch to 8051 and write an ALP to generate an interrupt which switches on an LED (i) continuously as long as switch is on and (ii) only once for a small time when the switch is turned on.
2. Write a C program to (i) transmit and (ii) to receive a set of characters serially by interfacing 8051 to at terminal.

CYCLE II.B

3. Write programs to generate waveforms using ADC interface.
4. Write programs to interface an LCD display and to display a message on it.
5. Write programs to interface a Stepper Motor to 8051 to rotate the motor.
6. Write programs to interface ADC-0804 and convert an analog input connected to it.

Program No.: 1A

Objective: To write an ALP to transfer the block of data from source memory to destination memory

Algorithm

1. Start.
2. Set the counter value which is equal to number of data to be transferred.
3. Initialize source and destination memory locations.
4. Fetch the first data from source memory location to Accumulator.
5. Transfer the fetched data to destination memory location with the help of data pointer.
6. Decrement the counter value by 1 and increment the data pointer to fetch next data.
7. Repeat steps from 3 to 6 till counter value becomes zero.
8. End.

Program: To transfer 8 bytes of data from external memory location starting from 8100h to external memory location starting from 8200h

```
        ORG 0000H
        MOV R0, #08H        ; initialize the count
        MOV R1, #81H        ; initialize the source memory location higher byte
        MOV R2, #82H        ; initialize the destination memory location higher byte
        MOV R3, #00H        ; initialize the destination & source location lower byte
BACK:   MOV DPH, R1         ; get the source memory location address to DPTR
        MOV DPL, R3
        MOVX A, @DPTR       ; get the data from source memory to Accumulator
        MOV DPH, R2         ; get the destination memory location address to DPTR
        MOVX @DPTR, A       ; copy the accumulator content to destination memory
        INC R3              ; increment to next source and destination memory
        DJNZ R0, BACK       ; decrement count. If count! =0 go to label "BACK"
        SJMP $
        END
```

Outcome:

Before execution

Address	Data
0x8100	0x12
0x8101	0x24
0x8102	0x56
0x8103	0XFF
0x8104	0xEE
0x8105	0xAB
0x8106	0x10
0x8107	0x03

Address	Data
0x8200	0x00
0x8201	0x00
0x8202	0x00
0x8203	0x00
0x8204	0x00
0x8205	0x00
0x8206	0x00
0x8207	0x00

After execution

Address	Data
0x8100	0x12
0x8101	0x24
0x8102	0x56
0x8103	0xFF
0x8104	0xEE
0x8105	0xAB
0x8106	0x10
0x8107	0x03

Address	Data
0x8200	0x12
0x8201	0x24
0x8202	0x56
0x8203	0xFF
0x8204	0xEE
0x8205	0xAB
0x8206	0x10
0x8207	0x03

Program No.: 1B

Objective: To write an ALP to exchange the data between two external memory locations

Algorithm

1. Start.
2. Set the counter value which is equal to number of data to be exchanged.
3. Initialize two blocks of memory locations.
4. Fetch the first data from one memory location and save it in the intermediate register.
5. Fetch the first data from other memory location to accumulator
6. Exchange the date between accumulator and register
7. Transfer the data to corresponding memory location with the help of data pointer.
8. Decrement the counter value by 1 and increment the data pointer to fetch next data
9. Repeat steps from 4 to 8 till counter value becomes zero.
10. End

Program: To exchange 8 bytes of data between external memories location starting from 8100h and external memory location starting from 8200h

```
                ORG 0000H

                MOV R0, #08H        ; initialize the count

                MOV R1, #81H        ; initialize the memory1 location higher byte

                MOV R2, #82H        ; initialize the memory2 location higher byte

                MOV R3, #00H        ; initialize the memory1&memory2 location lower byte

BACK:           MOV DPH, R1         ;get the memory1 location address to DPTR

                MOV DPL, R3

                MOVX A, @DPTR       ; get the data from memory1 to Accumulator

                MOV B, A            ; copy the accumulator content to B register

                MOV DPH, R2         ; get the memory2 location address to DPTR

                MOVX A,@DPTR        ; get the data from memory2 to Accumulator

                XCH A, B            ; exchange the accumulator and B register content

                MOVX @DPTR, A       ; copy the accumulator content to memory2

                MOV A, B            ; get the B register content to accumulator

                MOV DPH, R1         ; get the memory1 location address to DPTR

                MOVX @DPTR, A       ; copy the accumulator content to memory1

                INC R3              ; increment to next source and destination memory

                DJNZ R0, BACK       ; decrement count. If count! =0 go to label “BACK”

                SJMP $

                END
```

Outcome:

Before execution

Address	Data
0x8100	0x12
0x8101	0x24
0x8102	0x56
0x8103	0xFF
0x8104	0xEE
0x8105	0xAB
0x8106	0x10
0x8107	0x03

Address	Data
0x8200	0x32
0x8201	0xFF
0x8202	0xAD
0x8203	0xDA
0x8204	0x88
0x8205	0x99
0x8206	0x56
0x8207	0x55

After execution

Address	Data
0x8100	0x32
0x8101	0xFF
0x8102	0xAD
0x8103	0xDA
0x8104	0x88
0x8105	0x99
0x8106	0x56
0x8107	0x55

Address	Data
0x8200	0x12
0x8201	0x24
0x8202	0x56
0x8203	0xFF
0x8204	0xEE
0x8205	0xAB
0x8206	0x10
0x8207	0x03

Program No.: 1C

Objective: To write an ALP to find the largest number in a given array

Algorithm

1. Start.
2. Set the counter value which is equal to number of data minus one.
3. Initialize memory location to provide the input and to view the output.
4. Fetch the first two data from memory location and compare them.
5. Check whether two numbers are equal, if they are equal then no need to compare continue checking with the next data. If they are not equal then compare the two numbers.
6. If the first data is greater than second data then exchange the data between accumulator and register so that largest number lies in accumulator.
7. Increment the data pointer to fetch next data to be compared with the previously stored largest number in accumulator.
8. Repeat steps from 5 to 7 till counter becomes zero
9. After all comparison the largest number will be present in accumulator, transfer the number to initialized memory location to view the result.
10. End

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
                                     Location
           SJMP $
           END
```

Outcome:

Before execution

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

Address	Data
0x8100	0x00

After execution

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

Address	Data
0x8100	0xFF LARGEST

Program No.: 1D

Objective: To write an ALP to find the smallest number in a given array

Algorithm

1. Start.
2. Set the counter value which is equal to number of data minus one.
3. Initialize memory location to provide the input and to view the output.
4. Fetch the first two data from memory location and compare them.
5. Check whether two numbers are equal, if they are equal then no need to compare continue checking with the next data. If they are not equal then compare the two numbers.
6. If the first data is smaller than second data then exchange the data between accumulator and register so that smallest number lies in accumulator.
7. Increment the data pointer to fetch next data to be compared with the previously stored smallest number in accumulator.
8. Repeat steps from 5 to 7 till counter becomes zero
9. After all comparison the largest number will be present in accumulator, transfer the number to initialized memory location to view the result.
10. End

Program: To find the smallest number in a given array of size 5 starting from 5100h external memory location. The smallest 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
                               Location
        SJMP $
        END
```

Outcome:

Before execution

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

Address	Data
0x8100	0x00

After execution

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

Address	Data
0x8100	0x12

Program No.: 1E

Objective: To write an ALP to arrange the data in given array in ascending order

Algorithm

1. Start.
2. Set the counter1 value for outer loop which is equal to number of data minus one.
3. Set the counter2 value for inner loop which is equal to number of data minus one.
4. Initialize memory location to provide the number of data to be arranged.
5. Point data pointer to initial memory location.
6. Fetch the data from memory location and compare it with next number.
7. If the first data is greater than second data then exchange the data between accumulator and register so that largest number lies in accumulator.
8. Decrement counter 2 by 1 and increment data pointer by 1 to fetch the next data.
9. Repeat steps from 6 to 8 till counter 2 becomes zero.
10. Decrement counter1 by one, load the counter2 to initial value
11. Repeat step from 5 to 10 till counter1 becomes zero.
12. The numbers will be arranged in ascending order in the same memory location.
13. End

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
```

Outcome:

Before execution:

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

After execution:

Address	Data
0x5100	0x01 SMALLEST
0x5101	0x1F
0x5102	0x56
0x5103	0xD4
0x5104	0XFF LARGEST

Program No.: 1F

Objective: To write an ALP to arrange the data in given array in descending order

Algorithm

1. Start.
2. Set the counter1 value for outer loop which is equal to number of data minus one.
3. Set the counter2 value for inner loop which is equal to number of data minus one.
4. Initialize memory location to provide the number of data to be arranged.
5. Point data pointer to initial memory location.
6. Fetch the data from memory location and compare it with next number.
7. If the first data is smaller than second data then exchange the data between accumulator and register so that smallest number lies in accumulator.
8. Decrement counter 2 by 1 and increment data pointer by 1 to fetch the next data.
9. Repeat steps from 6 to 8 till counter 2 become zero.
10. Decrement counter1 by one, load the counter2 to initial value
11. Repeat step from 5 to 10 till counter1 becomes zero.
12. The numbers will be arranged in ascending order in the same memory location.
13. End

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
```

Outcome:

Before execution:

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

After execution:

Address	Data
0x5100	0XFF LARGEST
0x5101	0xD4
0x5102	0x56
0x5103	0X1F
0x5104	0x01 SMALLEST

Program No.: 2A

Objective: To write an ALP to add two 16 bit numbers

Algorithm

1. Start.
2. Initialize 2 memory location to provide 2 data to be added.
3. Initialize a memory location to view the output
4. Fetch the lower byte of first data and add it with lower byte of second data.
5. Transfer the result to the output memory location.
6. Fetch the higher byte of first data and add it with higher byte of second data with the carry generated in the previous addition.
7. Transfer the result to the output memory location.
8. Clear the accumulator, add its content with carry generated.
9. Transfer the final carry generated to the output memory location
10. End

Program: To add two 16 bit numbers, first 16 bit number placed in 8100h and 8101h external memory locations and second 16 bit number placed in 8200h and 8201h external memory locations. The result has to be stored in 8300h, 8301h and 8302h external memory locations.

```
ORG 0000H
MOV DPTR,#8101H       ; initialize the external memory location
MOVX A,@DPTR          ; get the 1st LSB data from memory to accumulator
MOV B,A               ; move the content from accumulator to B register
MOV DPTR,#8201H       ; initialize new memory location
MOVX A,@DPTR          ; get the 2nd LSB data from memory to accumulator
ADD A,B               ; add the content of A and B
MOV DPTR,#8302H       ; initialize new memory location
MOVX @DPTR,A          ; move the accumulator content to memory
MOV DPTR,#8100H       ; initialize new memory location
MOVX A,@DPTR          ; get the 1st MSB data from memory to accumulator
MOV B,A               ; move the content from accumulator to B register
MOV DPTR,#8200H       ; initialize new memory location
MOVX A,@DPTR          ; get the 2nd MSB data from memory to accumulator
ADDC A,B              ; add the content of A and B with carry
MOV DPTR,#8301H       ; initialize new memory location
MOVX @DPTR,A          ; move the accumulator content to memory
```

```
MOV A,#00H          ; move the value „00‟ to accumulator
ADDC A,#00H         ; add accumulator data with carry
DEC DPL             ; decrement lower byte of memory
MOVX @DPTR,A        ; move the accumulator content to memory
SJMP $
END
```

Outcome:

Before execution

Address	Data
0x8100	0xFF
0x8101	0xFF

Address	Data
0x8200	0xFF
0x8201	0xFF

Address	Data
0x8300	0x00
0x8301	0x00
0x8301	0x00

After execution

Address	Data
0x8100	0XFF ADDEND
0x8101	0XFF ADDEND

Address	Data
0x8200	0XFF AUGEND
0x8201	0XFF AUGEND

Address	Data
0x8300	0x01 SUM
0x8301	0xFF SUM
0x8301	0xFE SUM

Program No.: 2B

Objective: To write an ALP to subtract one 16- bit number from another

Algorithm

1. Start.
2. Initialize 2 memory locations to provide 2 data to be subtracted.
3. Initialize a memory location to view the output.
4. Fetch the lower byte of second data and subtract it from lower byte of first data with borrow.
5. Transfer the result to the output memory location.
6. Fetch the higher byte of second data and subtract it from higher byte of first data with borrow.
7. Transfer the result to the output memory location.
8. Clear the accumulator, subtract its content from borrow.
9. Transfer the final borrow generated to the output memory location.
10. End

Program: To subtract one 16-bit number from another. Minuend is placed in 8100h and 8101h external memory locations and Subtrahend is placed in 8200h and 8201h external memory locations. The difference has to be stored in 8300h, 8301h and 8302h external memory locations. The 8300h memory location should indicate the sign of the result.

```
ORG 0000H

MOV DPTR, #8101H        ; initialize the external memory location

MOVX A,@DPTR            ; get the 1st LSB data from memory to accumulator

MOV B, A                ; move the content from accumulator to B register

MOV DPTR,#8201H         ; initialize new memory location

MOVX A,@DPTR            ; get the 2nd LSB data from memory to accumulator

SUBB A, B               ; Subtract the content of B from Accumulator with
                        borrow

MOV DPTR, #8302H        ; initialize new memory location

MOVX @DPTR, A           ; move the accumulator content to memory

MOV DPTR, #8100H        ; initialize new memory location

MOVX A,@DPTR            ; get the 1st MSB data from memory to accumulator

MOV B, A                ; move the content from accumulator to B register

MOV DPTR, #8200H        ; initialize new memory location

MOVX A,@DPTR            ; get the 2nd MSB data from memory to accumulator

SUBB A, B               ; Subtract the content of B from Accumulator with
                        borrow

MOV DPTR, #8301H        ; initialize new memory location
```

```
MOVX @DPTR, A            ; move the accumulator content to memory
MOV A, #00H              ; move the value „00‟ to accumulator
SUBB A, #00H             ; subtract „00‟ from A with borrow
DEC DPL                  ; decrement lower byte of memory location
MOVX @DPTR, A            ; move the accumulator content to memory
SJMP $
END
```

Outcome:

CASE 1: Negative result

Before execution

Address	Data
0x8100	0x23
0x8101	0x12

Address	Data
0x8200	0x12
0x8201	0x45

Address	Data
0x8300	0x00
0x8301	0x00
0x8302	0x00

After execution

Address	Data
0x8100	0x02
0x8101	0x01

Address	Data
0x8200	0x12
0x8201	0x45

Address	Data
0x8300	0xFF
0x8301	0xEF
0x8302	0x33

CASE 2: Positive result

Before execution

Address	Data
0x8100	0x12
0x8101	0x45

Address	Data
0x8200	0x23
0x8201	0x12

Address	Data
0x8300	0x00
0x8302	0x00
0x8302	0x00

After execution

Address	Data
0x8100	0x12
0x8101	0x45

Address	Data
0x8200	0x23
0x8201	0x12

Address	Data
0x8300	0x00
0x8301	0x10
0x8302	0xCD

Program No.: 2C

Objective: To write an ALP to multiply an 8-bit number with a 16 -bit number

Algorithm

1. Start.
2. Initialize 2 memory location to provide 8 bit multiplier and 16 bit multiplicand.
3. Initialize a memory location to view the output.
4. Fetch the lower byte of multiplicand and multiply it with multiplier.
5. Transfer the lower byte of result to the output memory location.
6. Save the higher byte of result in register.
7. Fetch the higher byte of multiplicand and multiply it with multiplier.
8. Add with carry the lower byte of result obtained with previously stored intermediate result.
9. Transfer the result to the output memory location.
10. Add the higher byte of result obtained with carry and transfer to the output memory location.
11. End

Program: To multiply an 8-bit number placed in external memory location 8100h and the 16 bit number is placed in external memory locations 8200h and 8201h.The product will be stored in external memory locations 8300h, 8301h and 8302h.

```
ORG 0000H

MOV DPTR, #8100H          ; initialize the external memory location

MOVX A,@DPTR              ; get the data from memory to accumulator

MOV B, A                  ; move the content from accumulator to B register

MOV R0, A                 ; get the multiplier to R0 register

MOV DPTR, #8201H          ; get the lower byte of multiplicand to accumulator

MOVX A,@DPTR

MUL AB                    ; multiply - lower byte of Multiplicand * Multiplier

MOV DPTR, #8302H          ; store the lower byte result in result+2 memory

MOVX @DPTR, A

MOV R1, B                 ; move the upper byte result in R1

MOV DPTR, #8200H          ; get the upper byte of multiplicand to accumulator

MOVX A, @DPTR

MOV B, R0                 ; get the multiplier to B register

MUL AB                    ; multiply - upper byte multiplicand* Multiplier

ADDC A, R1                ; Add lower byte result with R1 (upper byte result of
                          lower multiplicand multiplication)

MOV DPTR, #8301H          ; store the result in result memory+1 location

MOVX @DPTR, A
```

```
MOV A, B                    ; get the upper byte result of upper multiplicand
ADDC A, #00H                ; add the carry to upper multiplicand result
DEC DPL
MOVX @DPTR, A               ; store the result in result memory location
SJMP $
END
```

Outcome:

Before execution

Address	Data
0x8100	0xFF

Address	Data
0x8200	0xFF
0x8201	0xFF

Address	Data
0x8300	0x00
0x8301	0x00
0x8302	0x00

After execution

Address	Data
0x8100	0xFF

Address	Data
0x8200	0xFF
0x8201	0xFF

Address	Data
0x8300	0xFE
0x8301	0xFF
0x8302	0x01

Program No.: 2D

Objective: To write an ALP to find square of a given number

Algorithm

1. Start.
2. Initialize 2 memory location, one to provide input and one to view the output.
3. Fetch the data from memory location and multiply the number with itself.
4. Transfer the result to the output memory location.

End

Program: To find square of given number, input is placed in external memory location 8100h, and square is placed in the external memory 8101h and 8102h.

```
ORG 0000H
 MOV DPTR, #8100H        ; get the source address
 MOVX A, @DPTR           ; get the input data to accumulator
 MOV B, A                ; move the input data to B register
 MUL AB                  ; get the square of the number
 INC DPTR                ; get the result+1 address to store the square result
 INC DPTR
 MOVX @DPTR, A           ; save the lower byte of the result
 DEC DPL                 ; get the result memory location
 MOV A, B                ; get the upper byte of the result to the Accumulator
 MOVX @DPTR, A           ; store the upper byte of the result to memory location
 SJMP $
 END
```

Outcome:

Before execution

Address	Data
0x8100	0xFF

Address	Data
0x8101	0X00
0x8102	0X00

After execution

Address	Data
0x8100	0XFF Given Number

Address	Data
0x8101	0XFE SQUARE
0x8102	0X01 SQUARE

Program No.: 2E

Objective: To write an ALP to find cube of a given number

Algorithm

1. Start.
2. Initialize memory location to provide input and to view output.
3. Fetch the data and multiply the number with itself to find square of a number.
4. The lower and higher byte of result is again multiplied with the number to find a cube of a number.
5. Transfer the result obtained to the output memory location
6. End

Program: To find cube of given number, the given number is placed in external memory location 8100h, and the cube is placed in the external memory 8200h, 8201h and 8202h

```
ORG 0000H

MOV DPTR, #8100H          ; get the source address

MOVX A,@DPTR              ; get the input data to accumulator

MOV B, A                  ; move the input data to B register

MOV R0, A                 ; copy the input data to the register R0

MUL AB                    ; get the square of the input number

MOV R1, B                 ; copy the upper byte of the square result in the R1 register

MOV B, R0                 ; get the input data to register B

MUL AB                    ; get the lower byte of the cube result

MOV DPTR, #8202H          ; get the result+2 memory location

MOVX @DPTR, A             ; store the lower byte of cube output in result+2 memory

MOV R2, B                 ; store the upper byte partial result in R2

MOV B, R1                 ; get the previous partial result to register B

MOV A, R0                 ; get the input to accumulator

MUL AB                    ; get the second upper byte partial result

ADDC A, R2                ; add the input data to the partial result with the previous carry

DEC DPL                   ; get the result+1 memory location

MOVX @DPTR, A             ; store the 2nd byte of cube output in result+1 memory

MOV A, B                  ; get the upper byte of the multiplied output to accumulator

ADDC A, #00H              ; add with the previous carry
```

```
DEC DPL                 ; get the result memory location

MOVX @DPTR, A           store the 3rd byte of cube output in result memory

SJMP $

END
```

Outcome:

Before execution

Address	Data
0x8100	0xFF

Address	Data
0x8200	0X00
0x8201	0X00
0x8202	0X00

After execution

Address	Data
0x8100	0XFF Given number

Address	Data
0x8200	0XFD CUBE
0x8201	0X02 CUBE

Program No.: 2F

Objective: To write an ALP to perform 8 bit / 8bit division

Algorithm

1. Start.
2. Initialize memory location to provide dividend and divisor.
3. Initialize memory location to view the remainder and quotient.
4. Fetch the inputs, divide the dividend by the divisor.
5. Transfer the quotient and remainder obtained to the output memory location.
6. End

Program: To perform 8 bit / 8bit division. Dividend is placed in external memory location 8200h, and divisor is placed in the external memory location 8100h, the result will be placed in the memory locations 8300h (quotient) and 8301h (remainder).

```
ORG 0000H

MOV DPTR, #8100H          ; get the divisor data address

MOVX A, @DPTR             ; get the divisor to accumulator

MOV B, A                  ; save the divisor in the register B

MOV DPTR, #8200H          ; get the dividend data address

MOVX A, @DPTR             ; get the dividend to accumulator

DIV AB                    ; divide A/B

MOV DPTR, #8300H          ; get the quotient memory address to DPTR

MOVX @DPTR, A             ; store the quotient in 8300h memory location

MOV A, B                  ; get the remainder to accumulator

INC DPTR                  ; get the next address to store the remainder

MOVX @DPTR, A             ; store the remainder in 8301h memory location

SJMP $

END
```

Outcome:

Before execution

Address	Data
0x8100	0x13 Divisor

Address	Data
0x8200	0x45 Dividend

Address	Data
0x8300	0X00
0x8301	0X00

After execution

Address	Data
0x8100	0x13

Address	Data
0x8200	0x45

Address	Data
0x8300	0X03 Quotient
0x8301	0X0C Remainder

Program No.: 3A

Objective: To write an ALP to display BCD up count

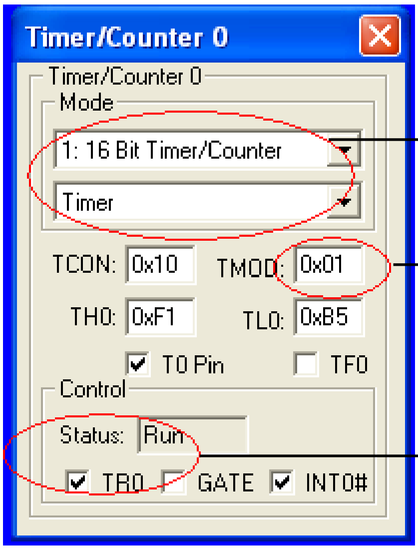
Algorithm

1. Start.
2. Initialize timer 0 in mode 1 configuration to generate delay.
3. Initialize port1 to view result.
4. Initial accumulator with value 00
5. Load the value from accumulator to port1.
6. Call delay subroutine.
7. Increment the accumulator.
8. Repeat step from 5 to 7 till accumulator value reaches 99h
9. Repeat step from 4 to 8 continuously.
10. End

Program: To display BCD up count (00 to 99) continuously in Port1. The delay between two counts should be 1 second. Configure TMOD register in Timer0 Mode1 configuration.

```
          ORG 0000H

          MOV A, #00H            ; get the first BCD value to accumulator

L1:       MOV P1, A              ; display the count in P1

          ADD A, #01H            ; get the next count to be displayed

          DA A                   ; decimal adjust the count

          LCALL DELAY            ; call the delay of 1sec

          SJMP L1                ; repeat forever


DELAY:    MOV TMOD, #01H         ; configure timer0 in mode1

          MOV R0, #0EH           ; get the count for repetition of timer register count (14 d)

BACK:     MOV TL0, #00H          ; set the initial count for "0.071 second x 14 = 1 second"

          MOV TH0, #00H

          SETB TR0               ; start the timer

REPEAT:   JNB TF0, REPEAT        ; wait until timer overflows

          CLR TR0                ; halt the timer

          CLR TF0                ; clear the timer0 overflow interrupt

          DJNZ R0, BACK          ; if repetition count != 0, go to label back

          RET                    ; return to the main program

          END
```

Sample view:



Timer/Counter 0

Mode

1: 16 Bit Timer/Counter

Timer

TCON: 0x10 TMOD: 0x01

TH0: 0xF1 TL0: 0xB5

☑ T0 Pin ☐ TF0

Control

Status: Run

☑ TR0 ☐ GATE ☑ INT0#

Timer 0 working in mode1 in Timer mode

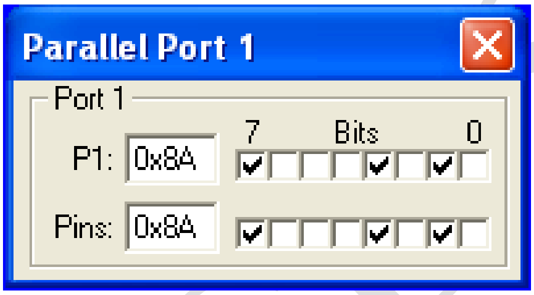
TMOD register is configured to work as:

- Timer 0 in Timer mode
- To work in mode 1 (16 bit timer)

TR0 bit controls the running of the timer

TR0=1; Timer0 will be in running

state TR0=0;Timer0 will be in halt



Port 1

7 Bits 0

P1: 0x8A ☑ ☐ ☐ ☐ ☑ ☐ ☑ ☐

Pins: 0x8A ☑ ☐ ☐ ☐ ☑ ☐ ☑ ☐

Outcome:

Observed the BCD up count operation on Port1.

Program No: 3B

Objective: To write an ALP to display BCD down count

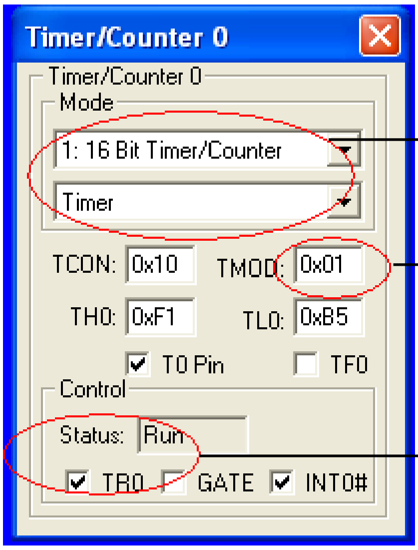
Algorithm

1. Start.
2. Initialize timer 0 in mode 1 configuration.
3. Initialize port1 to view result.
4. Initial accumulator with value 99
5. Load the value from accumulator to port1.
6. Call delay subroutine.
7. Add accumulator content with 99 to decrement the value by1..
8. Repeat step from 5 to 7 till accumulator value reaches 99h
9. Repeat step from 4 to 8 continuously.
10. End

Program: To display BCD down count (99 to 00) continuously in Port1. The delay between two counts should be 1 second. Configure TMOD register in Timer0 Mode1 configuration.

```
        ORG 0000H
        MOV A, #99H        ; get the first BCD value to accumulator
L1:     MOV P1, A          ; display the count in P1
        ADD A, #99H        ; get the next BCD down count value
        DA A               ; decimal adjust the count
        LCALL DELAY        ; call the delay of 1sec
        SJMP L1            ; repeat forever

DELAY:  MOV TMOD, #01H     ; configure timer0 in mode1
        MOV R0, #0EH       ; get the count for repetition of timer register count (14 d)
BACK:   MOV TL0, #00H      ; set the initial count for "0.071 second x 14 = 1 second"
        MOV TH0, #00H
        SETB TR0           ; start the timer
REPEAT: JNB TF0, REPEAT    ; wait until timer overflows
        CLR TR0            ; halt the timer
        CLR TF0            ; clear the timer0 overflow interrupt
        DJNZ R0, BACK      ; if repetition count != 0, go to label back
        RET                ; return to the main program
        END
```

Sample view:



1: 16 Bit Timer/Counter
Timer

TCON: 0x10 TMOD: 0x01
TH0: 0xF1 TL0: 0xB5
☑ T0 Pin ☐ TF0

Status: Run
☑ TR0 ☐ GATE ☑ INT0#

Timer 0 working in mode1 in Timer mode

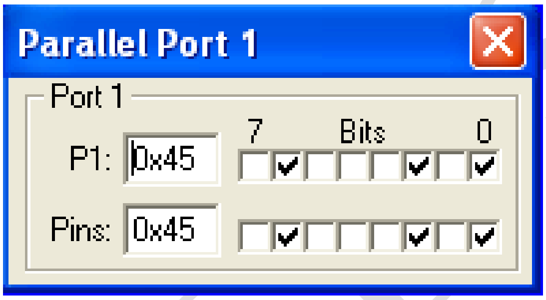
TMOD register is configured to work as:

- Timer 0 in Timer mode
- To work in mode 1 (16 bit timer)

TR0 bit controls the running of the timer

TR0=1; Timer0 will be in running

state TR0=0;Timer0 will be in halt



P1: 0x45 ☐ ☑ ☐ ☐ ☐ ☑ ☐ ☑
Pins: 0x45 ☐ ☑ ☐ ☐ ☐ ☑ ☐ ☑

Outcome:

Observed the BCD down count operation on Port1

Program No.: 4A

Objective: To write an ALP to find whether the given number is odd or even

Algorithm

1. Start.
2. Initialize memory location to provide input.
3. Initialize register to indicate whether the number is odd or even.
4. Fetch the data from memory location.
5. Rotate right the content of data with carry in order to check its LSB.
6. If carry is generated, means if LSB is one then the number is odd.
7. Indicate the number is odd by moving FF to the register.
8. If carry is not generated, means if LSB is zero then the number is even.
9. Indicate the number is even by moving 11 to the register.
10. End

Program: To check whether the given number placed in external memory location 8100h is odd or even, If the given number is odd store FF h in R1 register else if even store 11h in R1 register.

```
        ORG 0000H

        MOV DPTR, #8100H      ; get the input data from source memory location

        MOVX A,@DPTR

        RRC A                 ; get the 0th bit of input data to carry flag

        JC ODD                ; if 0th bit=1, input number is odd

        MOV R1, #11H          ; store "11" in R1 to indicate even number

        SJMP LAST

ODD:    MOV R1, #0FFH         ; store "FF" in R1 to indicate odd number

LAST:   SJMP $

        END
```

Outcome:

CASE 1: Odd number

Before execution

Address	Data
0x8100	0xFF

Register	Data
R1	0X00

After execution

Address	Data
0x8100	0xFF 1111 1111b

Register	Data
R1	0XFF

CASE 2: Even number

Before execution

Address	Data
0x8100	0xFE

Register	Data
R1	0X00

After execution

Address	Data
0x8100	0xFE 1111 1110b

Register	Data
R1	0X11

Program No.: 4B

Objective: To write an ALP to find whether the given number is Positive or Negative

Algorithm

1. Start.
2. Initialize memory location to provide input.
3. Initialize register to indicate whether the number is odd or even.
4. Fetch the data from memory location.
5. Rotate left the content of data with carry inorder to check its MSB.
6. If carry is generated, means if MSB is one then the number is Negative.
7. Indicate the number is Negative by moving FF to the register.
8. If carry is not generated, means if MSB is zero then the number is Positive.
9. Indicate the number is positive by moving 11 to the register.
10. End

Program: To check whether the given number placed in external memory location 8100h is Positive or Negative. If the given number is Negative store FF h in R1 register else if Positive store 11h in R1 register.

```
            ORG 0000H

            MOV DPTR, #8100H   ; get the input data from source memory location

            MOVX A,@DPTR

            RLC A              ; get the 7th bit of input data to carry flag

            JC NEGATIVE        ; if 7th bit=1, input number is negative

            MOV R1, #11H       ; store “11” in R1 to indicate positive number

            SJMP LAST

NEGATIVE:   MOV R1, #0FFH      ; store “FF” in R1 to indicate negative number

LAST:       SJMP $

            END
```

Outcome:

CASE 1: Negative number

Before execution

Address	Data
0x8100	0xFF

Register	Data
R1	0X00

After execution

Address	Data
0x8100	0xFF 1111 1111b

Register	Data
R1	0XFF

CASE 2: Positive number

Before execution

Address	Data
0x8100	0x77

Register	Data
R1	0X00

After execution

Address	Data
0x8100	0x77 0111 0111b

Register	Data
R1	0X11

Program No.: 4C

Objective: To write an ALP to find number of logical ones and zeroes in the given number

Algorithm

1. Start.
2. Initialize memory location to provide input.
3. Set the counter value which is equal to number of bits in the data.
4. Initialize two registers to store number of one‟s and zero’s value.
5. Fetch the data from memory location.
6. Rotate right the content of data with carry in order to check the bit value.
7. If carry is generated, then increment the register which contains number of one’s value.
8. If carry is not generated, then increment the register which contains number of zeros value
9. End

Program: To find the number of logical zeroes and ones in the given number placed in the external memory location 8100h. The number of logical ones is indicated in the R2 register and the number of logical zeroes is indicated in the register R3.

```
          ORG 0000H
          MOV DPTR, #8100H       ; get the input data from source memory location
          MOVX A,@DPTR
          MOV R1, #08H           ; keep the count in R1 to check 8 bits of input data
          MOV R2, #00H           ; counter for logical ones
          MOV R3, #00H           ; counter for logical zeroes
NEXTBIT:  RRC A                  ; get the LSB bit to carry flag
          JC ONES                ; if bit is one jump to label ONES
          INC R3                 ; if no carry increment zero counter
          SJMP LAST
ONES:     INC R2                 ; if no carry increment ones counter
LAST:     DJNZ R1, NEXTBIT       ; if all the 8 bits are not checked, go back to label NEXTBIT
          SJMP $
          END
```

Outcome:

Before execution

Address	Data
0x8100	0x72

Address	Data
R2	0X00
R3	0X00

After execution

Address	Data
0x8100	0x72 Given number 0111 0010b

Address	Data
R2	0X04 Number of Logical ones
R3	0X04 Number of Logical zeros

Program No.: 5

Objective: To write an ALP using Call and return instructions

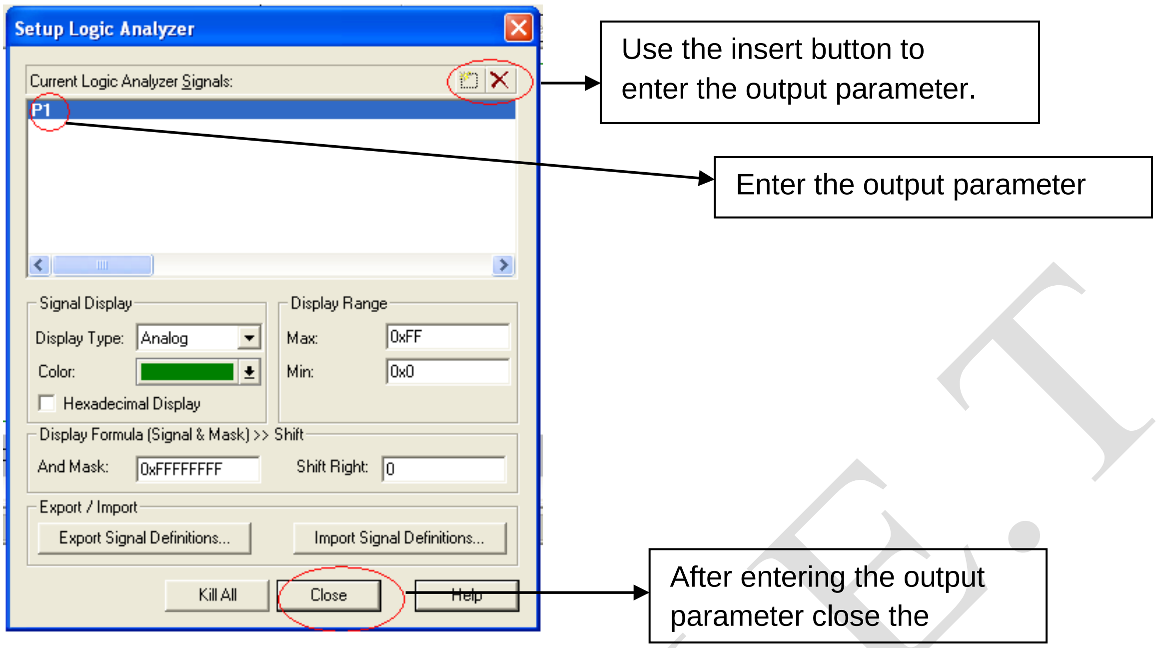
Algorithm

1. Start.
2. Initialize timer 0 in mode 1 configuration to generate delay.
3. Initialize port1.
4. Initial accumulator with value 00
5. Load the value from accumulator to port1.
6. Call delay subroutine.
7. Compliment the content of accumulator.
8. Repeat step from 4 to 8 continuously
9. Use logical analyzer to view the square wave output
10. End

Program: To generate the square wave in P1 with the 50% duty cycle and the time delay of 10 ms using timer. Assume the crystal frequency of 11.0592 MHz Configure the timer in Timer0 mode1.

```
            ORG 0000H

            MOV A, #00H         ; initialize P1

BACK:       MOV P1, A           ; generate square wave signal

            CPL A

            LCALL DELAY         ; call 10ms delay

            SJMP BACK           ; repeat forever


DELAY:      MOV TMOD, #01H      ; configure the timer0 in mode1

            MOV TL0, #000H      ; set the initial value in timer register for 5ms

            MOV TH0, #0dcH

            SETB TR0            ; start the timer

REPEAT:     JNB TF0, REPEAT     ; wait until timer overflows

            CLR TR0             ; halt the timer

            CLR TF0             ; clear the timer0 overflow interrupt

            RET                 ; ret to the main program

            END
```

Sample view:



Setup Logic Analyzer

Current Logic Analyzer Signals:

P1

Signal Display

Display Type: Analog

Color:

☐ Hexadecimal Display

Display Range

Max: 0xFF

Min: 0x0

Display Formula (Signal & Mask) >> Shift

And Mask: 0xFFFFFFFF

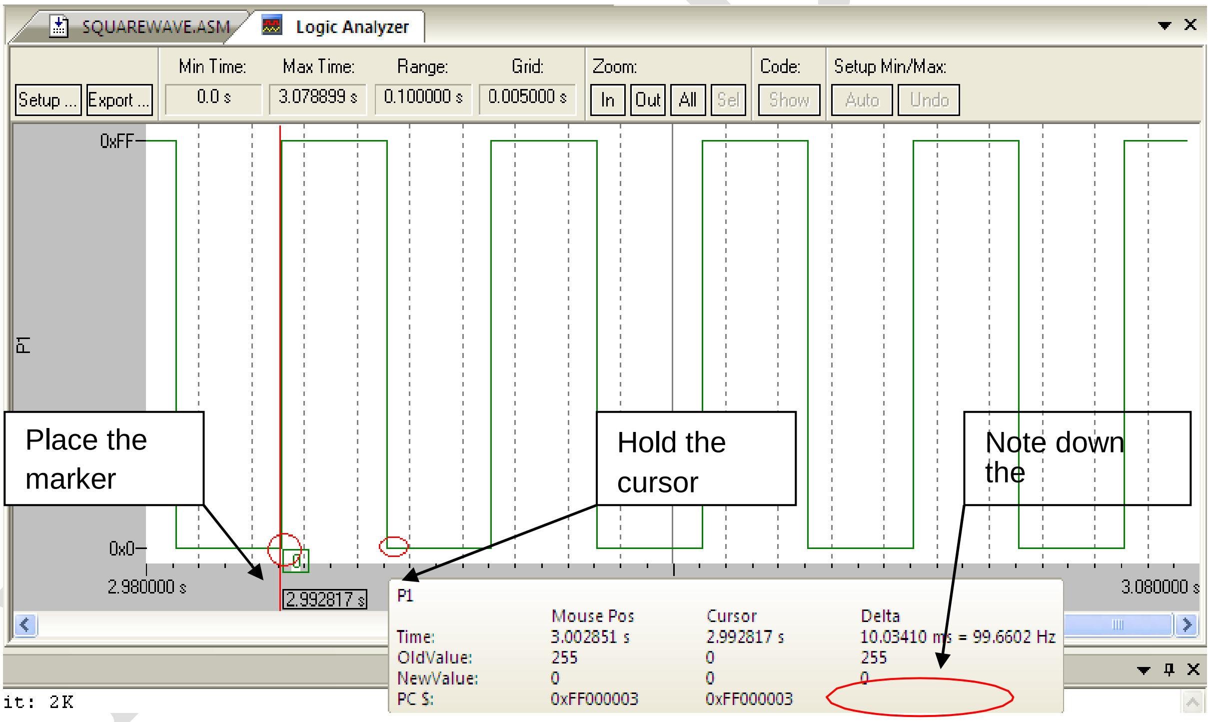
Shift Right: 0

Export / Import

Use the insert button to enter the output parameter.

Enter the output parameter

After entering the output parameter close the



Min Time: 0.0 s

Max Time: 3.078899 s

Range: 0.100000 s

Grid: 0.005000 s

2.980000 s

2.992817 s

3.080000 s

P1

	Mouse Pos	Cursor	Delta
Time:	3.002851 s	2.992817 s	10.03410 ms = 99.6602 Hz
OldValue:	255	0	255
NewValue:	0	0	0
PC $:	0xFF000003	0xFF000003	

it: 2K

Place the marker

Hold the cursor

Note down the

Outcome:

Observed the 50% duty cycle square wave on CRO generated on P1 and measured the time delay of 10ms.

Program no: 6A

Objective: To write an ALP to convert BCD number to its equivalent ASCII number.

Algorithm

1. Start.
2. Initialize memory location to provide input.
3. Initialize memory location to view output.
4. Fetch the data, obtain its higher and lower nibble.
5. Add 30 separately to higher and lower nibble to obtain its ascii value
6. Transfer the output the initialized output memory location.
7. End

Program: To convert unpacked BCD number (00-99) placed in internal memory location 20h to its equivalent ASCII number (30-39). The result as to be stored in internal memory location 40h and 41h.

```
ORG 0000H

MOV R0, #20H      ; get the source memory address in R0

MOV R1, #40H      ; get the destination memory address in R1

MOV A,@R0         ; get the input data to accumulator

ANL A, #0F0H      ; mask off the lower nibble

SWAP A            ; exchange upper and lower nibble

ORL A, #30H       ; convert upper nibble to ASCII

MOV @R1, A        ; send the ASCII data to destination memory

MOV A,@R0         ; get the input data to accumulator

ANL A, #0FH       ; mask off the upper nibble

ORL A, #30H       ; convert lower nibble to ASCII

INC R1            ; increment the destination memory location

MOV @R1, A ; send the ASCII data to destination memory

SJMP $

END
```

Outcome:

Before execution

Address	Data
0x0020	0x76

Address	Data
0x0040	0x00
0x0041	0x00

After execution

Address	Data
0x0020	0x76 Packed BCD

Address	Data
0x0040	0x37 ASCII
0x0041	0x36 ASCII

Program No.: 6B

Objective: To write an ALP to convert hexadecimal number to decimal number

Algorithm

1. Start.
2. Initialize memory location to provide input.
3. Initialize memory location to view output.
4. Fetch the data, and divide the number by 10 in decimal.
5. Store the remainder in register.
6. Divide the quotient obtained by 10 in decimal.
7. Add the remainder obtained with the previously stored remainder.
8. Transfer the result to the initialized output memory location.
9. Transfer the quotient obtained to the initialized output memory location.
10. End

Program: To convert the hexadecimal number placed in the external memory location 8100h to decimal number and store the result in the external memory location 8200h and 8201h.

```
ORG 0000H

MOV DPTR, #8100H        ; get the input data (hex number) memory location

MOVX A,@DPTR            ; get the input data to accumulator

MOV B, #0AH             ; get the divisor to B register

DIV AB                  ; divide input data by 10d

MOV R1, B               ; store the remainder in register in R1

MOV B, #0AH             ; get the divisor to B register

DIV AB                  ; divide the quotient of previous division by 10d

MOV R0, A               ; move the quotient to R0 register

MOV A, B                ; get the remainder to accumulator

SWAP A                  ; interchange upper and lower nibble

ORL A, R1               ; concatenate units and tens place

MOV DPTR, #8201H        ; get the result+1 memory location

MOVX @DPTR, A           ; store the tens and units (accumulator) place result

DEC DPL                 ; get the result+0 memory address

MOV A, R0               ; get the hundreds place value of the output to accumulator

MOVX @DPTR, A           ; store the result.
```

```
SJMP $

END
```

Outcome:

Before execution

Address	Data
0x8100	0xFF

Address	Data
0x8200	0X00
0x8201	0X00

After execution

Address	Data
0x8100	0xFF Hexa Decimal

Address	Data
0x8200	0X02 DECIMAL
0x8201	0X55 DECIMAL

Program No.: 6C

Objective: To write an ALP to convert decimal number to hexadecimal number.

Algorithm

1. Start.
2. Initialize memory location to provide input.
3. Initialize memory location to view output.
4. Fetch the data, and save its lower nibble in register.
5. Obtain the upper nibble of data and multiply with 0A.
6. Add the resultr obtained with the lower nibble of data.
7. Transfer the result to the initialized output memory location.
8. End

Program: To convert the decimal number placed in the external memory location 8100h to hexadecimal number and store the result in the external memory location 8101h

```
ORG 0000H

MOV DPTR, #8100H        ; get the input data (decimal number) memory location

MOVX A,@DPTR            ; get the input data (decimal number) to accumulator

MOV B, A                ; get the data to register B

ANL A, #0FH             ; mask off the upper nibble of the input data

MOV R1, A               ; save the accumulator data in register R1

MOV A, B                ; get the input data to accumulator

ANL A, #0F0H            ; mask off the lower nibble

SWAP A                  ; interchange the upper and lower nibble

MOV B, #0AH             ; get the multiplier to register B

MUL AB                  ; multiply upper nibble of input data with 0Ah

ADD A, R1               ; add multiplied data with input data‟s lower nibble value

INC DPTR                ; get the result memory location address to DPTR

MOVX @DPTR, A           ; store the hex decimal value in the result memory location

SJMP $

END
```

Outcome:

Before execution

Address	Data
0x8100	0x99

Address	Data
0x8101	0X00

After execution

Address	Data
0x8100	0x99 Decimal

Address	Data
0x8101	0X63 Hexa-decimal

Program No.: 7

Objective: To write an ALP to generate square wave with the on time delay of 6 ms and off time delay of 4 ms

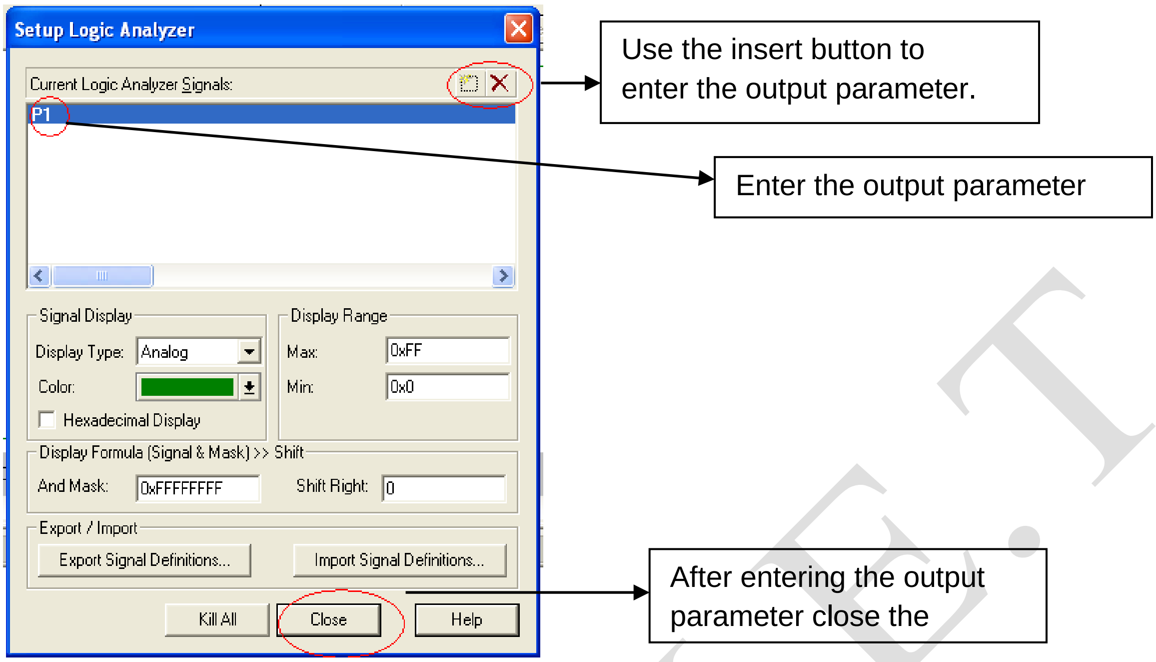
Algorithm

1. Start.
2. Initialize timer 0 in mode 1 configuration to generate delay.
3. Initialize port1.
4. Load port 1 with 00h.
5. Call delay subroutine of 1msec twice to obtain 2ms OFF time.
6. Load port 1 with FFh.
7. Call delay subroutine of 1msec four times to obtain 4ms ON time.
8. Repeat step from 4 to 7 continuously
9. Use logical analyzer to view the square wave output
10. End

Program: To generate the square wave with the on time delay of 6ms and off time delay of 4 m sec. Configure the timer in Timer0 mode1. Assume the crystal frequency of 11.0592 M Hz.

```
                ORG 0000H

BACK:           MOV P1, #00H          ; generate OFF time through P1

                LCALL DELAY           ; Call 1ms delay subroutine twice to get 2ms

                LCALL DELAY

                MOV P1, #0FFH         ; generate ON time through P1

                LCALL DELAY           ; Call 1ms delay subroutine four times to get 4ms

                LCALL DELAY

                LCALL DELAY

                LCALL DELAY

                SJMP BACK             ; repeat the processes forever

DELAY:          MOV TMOD, #01H        ; configure the timer0 in mode1

                MOV TL0, #0cdH        ; set the initial value in timer register for 2ms

                MOV TH0, #0F8H

                SETB TR0              ; start the timer

REPEAT:         JNB TF0, REPEAT       ; wait until timer overflows

                CLR TR0               ; halt the timer

                CLR TF0               ; clear the timer0 overflow interrupt

                RET                   ; ret to the main program

                END
```

Sample view:



Current Logic Analyzer Signals: P1

Display Type: Analog

Color: 🟩

☐ Hexadecimal Display

Max: 0xFF

Min: 0x0

And Mask: 0xFFFFFFFF

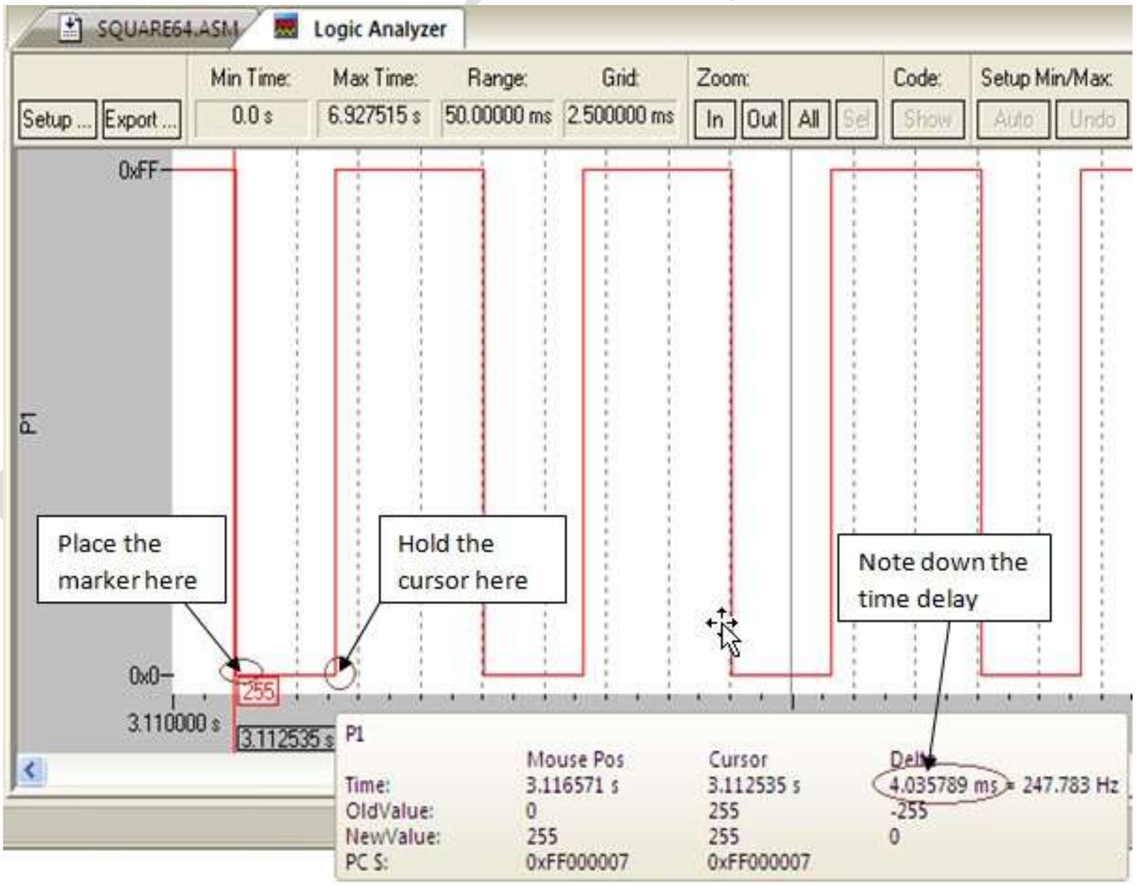
Shift Right: 0

Use the insert button to enter the output parameter.

Enter the output parameter

After entering the output parameter close the

OFF Time measure:



Min Time: 0.0 s

Max Time: 6.927515 s

Range: 50.00000 ms

Grid: 2.500000 ms

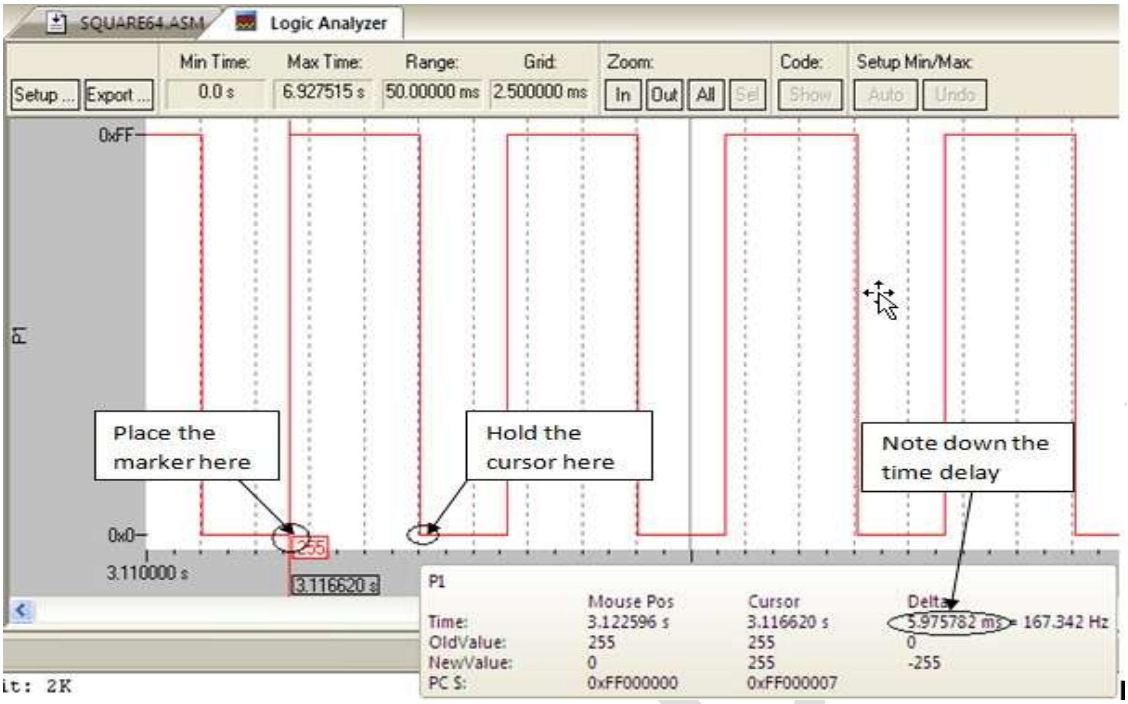
P1	Mouse Pos	Cursor	Delta
Time:	3.116571 s	3.112535 s	4.035789 ms = 247.783 Hz
OldValue:	0	255	-255
NewValue:	255	255	0
PC $:	0xFF000007	0xFF000007	

Place the marker here

Hold the cursor here

Note down the time delay

ON Time measure:



Min Time: 0.0 s
Max Time: 6.927515 s
Range: 50.00000 ms
Grid: 2.500000 ms

3.110000 s
3.116620 s

Place the marker here

Hold the cursor here

Note down the time delay

P1

	Mouse Pos	Cursor	Delta
Time:	3.122596 s	3.116620 s	5.975782 ms = 167.342 Hz
OldValue:	255	255	0
NewValue:	0	255	-255
PC $:	0xFF000000	0xFF000007	

Outcome:

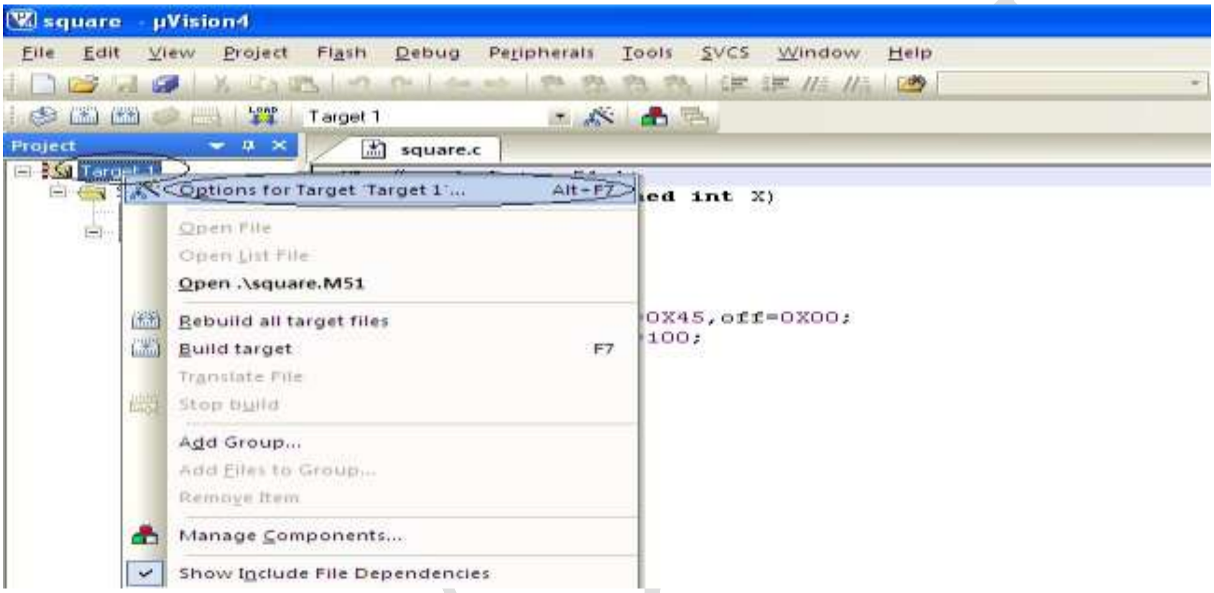
Observed the waveform with 6 msec ON time and 4 msec OFF time on CRO as generated on parallel port 1

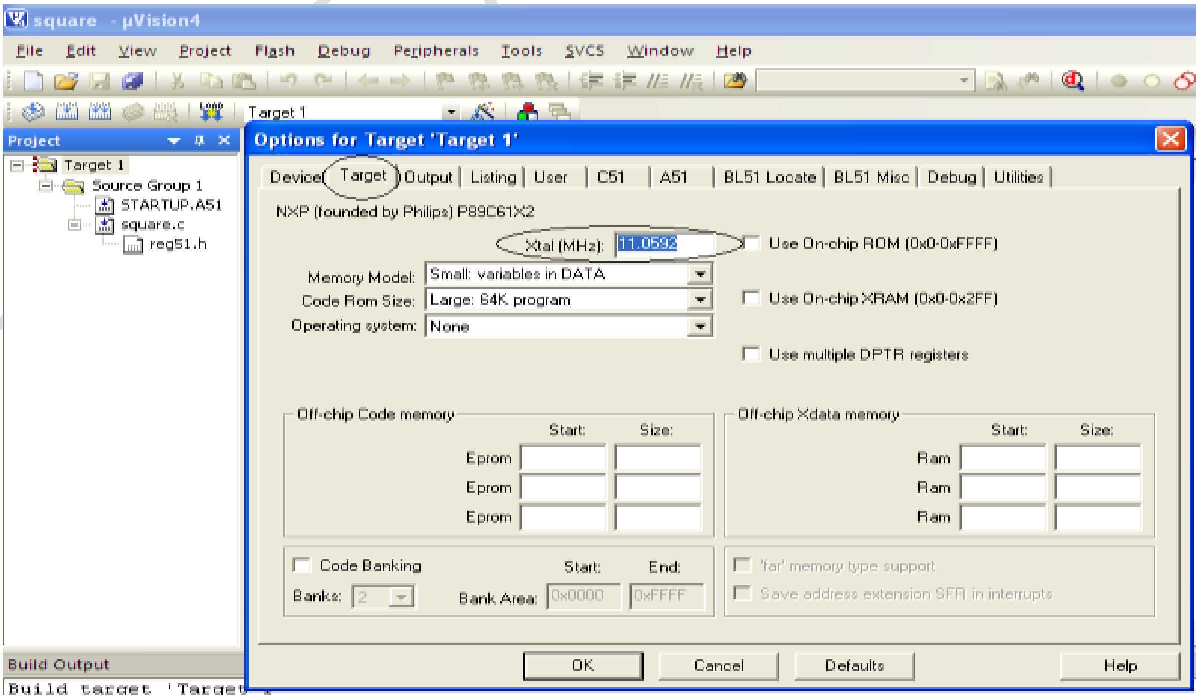
Hardware Programs

Steps FOR EXECUTING THE hardware PROGRAM:

STEP 1: Target setup.

- Right click on “Target 1” and select “Option for Target, Target 1”.
- Choose “Target” and change XTAL frequency as 11.0592.
- Choose “Device” and then choose “ATMEL- AT89C51”
- Choose “Output” and tick “Create Hex file” and then click “OK”.
- Choose “Debug” and then choose “Keil monitor -51 Driver” .





NXP (founded by Philips) P89C61X2

Xtal (MHz): 11.0592

☐ Use On-chip ROM (0x0-0xFFFF)

Memory Model: Small: variables in DATA

Code Rom Size: Large: 64K program

Operating system: None

☐ Use On-chip XRAM (0x0-0x2FF)

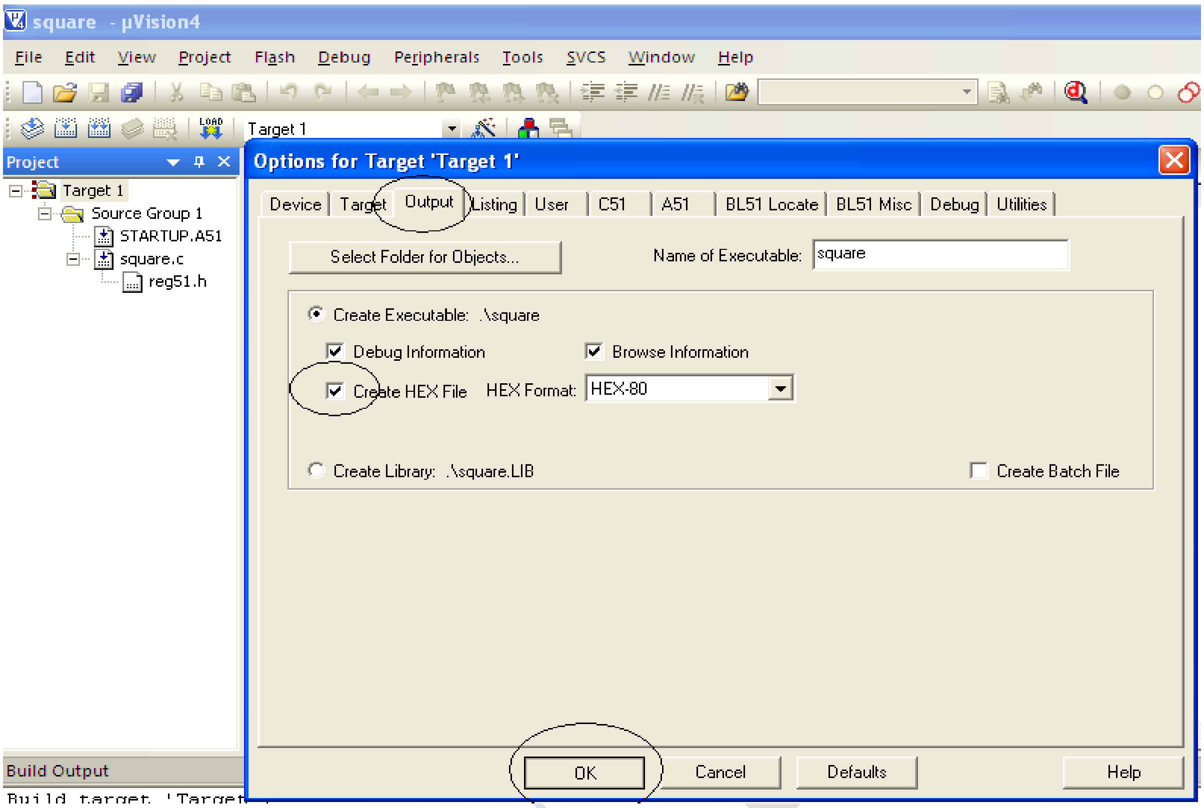
☐ Use multiple DPTR registers

☐ Code Banking Start: End:

Banks: 2 Bank Area: 0x0000 0xFFFF

☐ 'far' memory type support

☐ Save address extension SFR in interrupts



Options for Target 'Target 1'

Name of Executable: square

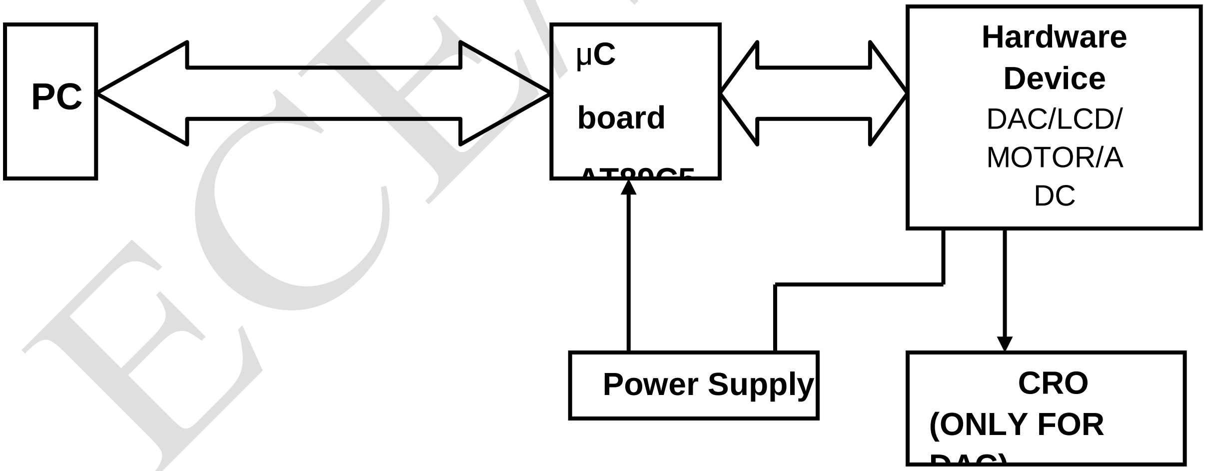
● Create Executable: .\square

☑ Debug Information ☑ Browse Information

☑ Create HEX File HEX Format: HEX-80

○ Create Library: .\square.LIB ☐ Create Batch File

STEP 2: Make all the hardware connection required.



Experiment 1: Toggle Switch Interface

Objective: Interface a simple toggle switch to 8051 and write an ALP to generate an interrupt which switches on an LED

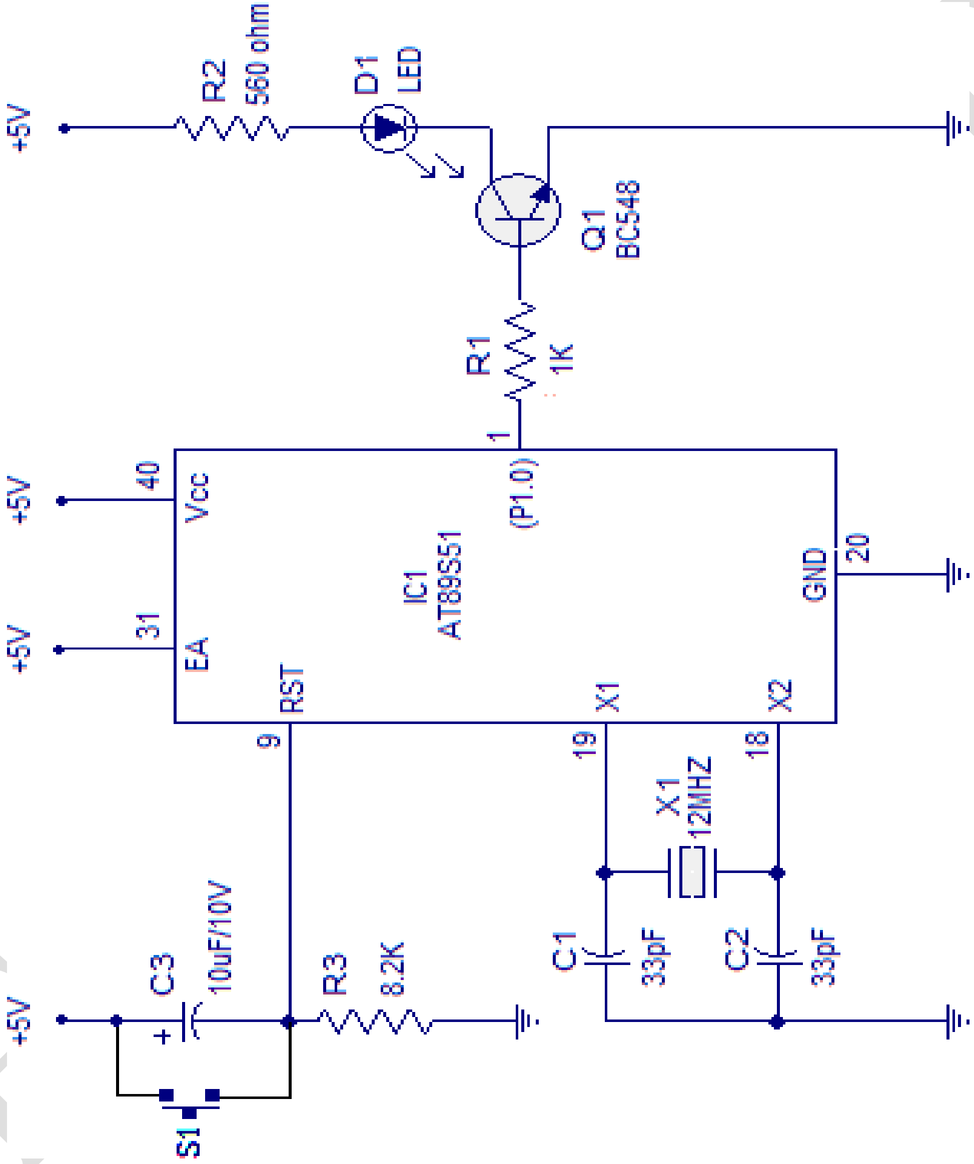


Fig. 1 Toggle switch interfacing to 8051 Microcontroller

Algorithm

Step 1: Initializing LEDs and Push button switches

Step 2: Checking the status of the switch1 (ON or OFF)

Step 3: Checking the status of the switch2 (ON or OFF)

Step 4: Turning LED ON based on switch condition

Program

```
            MOV P0,#83H         // Initializing push button LED
READSW:     MOV A,P0            // Moving the port value to Accumulator.
            RRC A               // Checking the switch 1 is ON or not
            JC NXT              // If switch 1 is OFF then to check if switch 2
            CLR P0.7            // Turn ON LED because Switch 1 is ON
            SJMP READSW         // Read switch status again.
NXT:        RRC A               // Checking the value of Port 0
            JC READSW           // check status of switch 1 again
            SETB P0.7           // Turning OFF LED because Switch 2 is ON
            SJMP READSW         // Jumping to READSW to read status of switch
            END
```

Outcome:

Toggle switch is successfully interfaced with microcontroller by observing the status of LED.

Experiment 2: Serial Communication

Objective: To write an ALP to send your name serially using UART at the baud rate of 9600

Algorithm:

Step1: Initialize Data pointer register

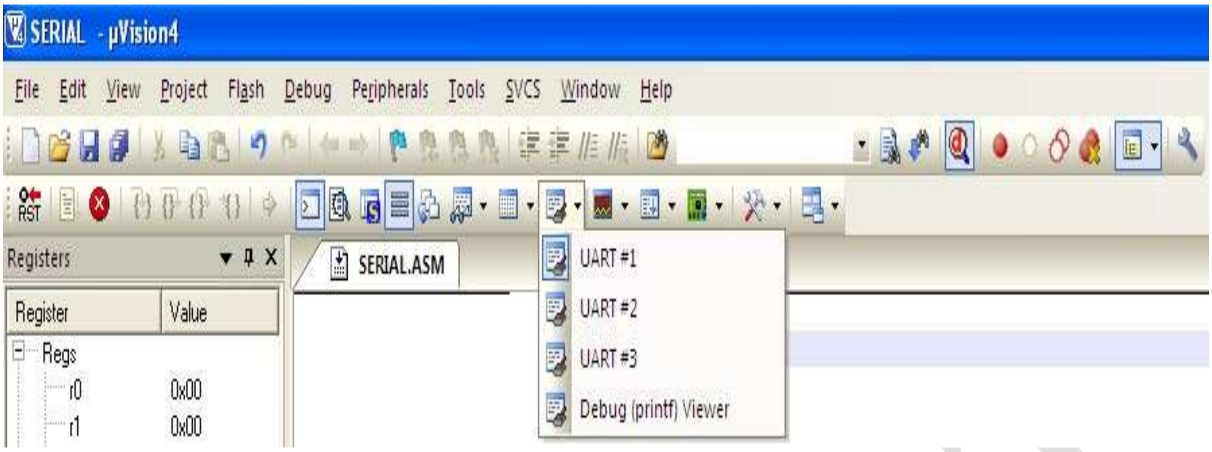
Step 2: Initialize Timer mode register and activate timer 1

Step 3 : Initialize serial communication register to send the given data through serial buffer

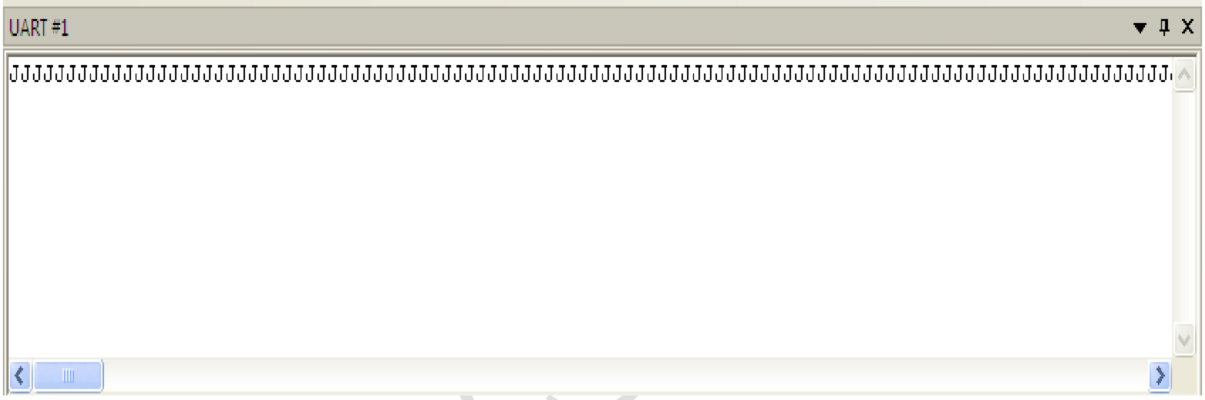
Step 4 : Observe the data in the specified memory register

Program: To send the letter „J‟ serially using the UART at the baud rate of 9600. Configure SCON register in mode 1. Assume the crystal frequency of 11.0592MHz.

```
        ORG 0000H
BACK:   MOV TMOD, #20H        ; configure the timer1 in mode2
        MOV TH1, #-3          ; count for the baud rate of 9600
        MOV SCON, #50H        ; configure SCON to mode1
        SETB TR1              ; start the timer
        MOV SBUF, #'J'        ; send the letter „J‟ through SBUF register
HERE:   JNB TI, HERE          ; wait until „J‟ character is sent (8bits are transferred)
        CLR TI                ; clear serial interrupt for next character to be sent
        SJMP BACK             ; repeat the processes
        SJMP $
        END
```



OUTPUT:



```
JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
```

Outcome:

Transmitted the letter „J“ serially using the UART at the baud rate of 9600

Experiment 3:

DAC Interface

Objective: To interface DAC to 8051 Microcontroller and to display different waveforms Square, Triangular and Staircase waveforms on CRO

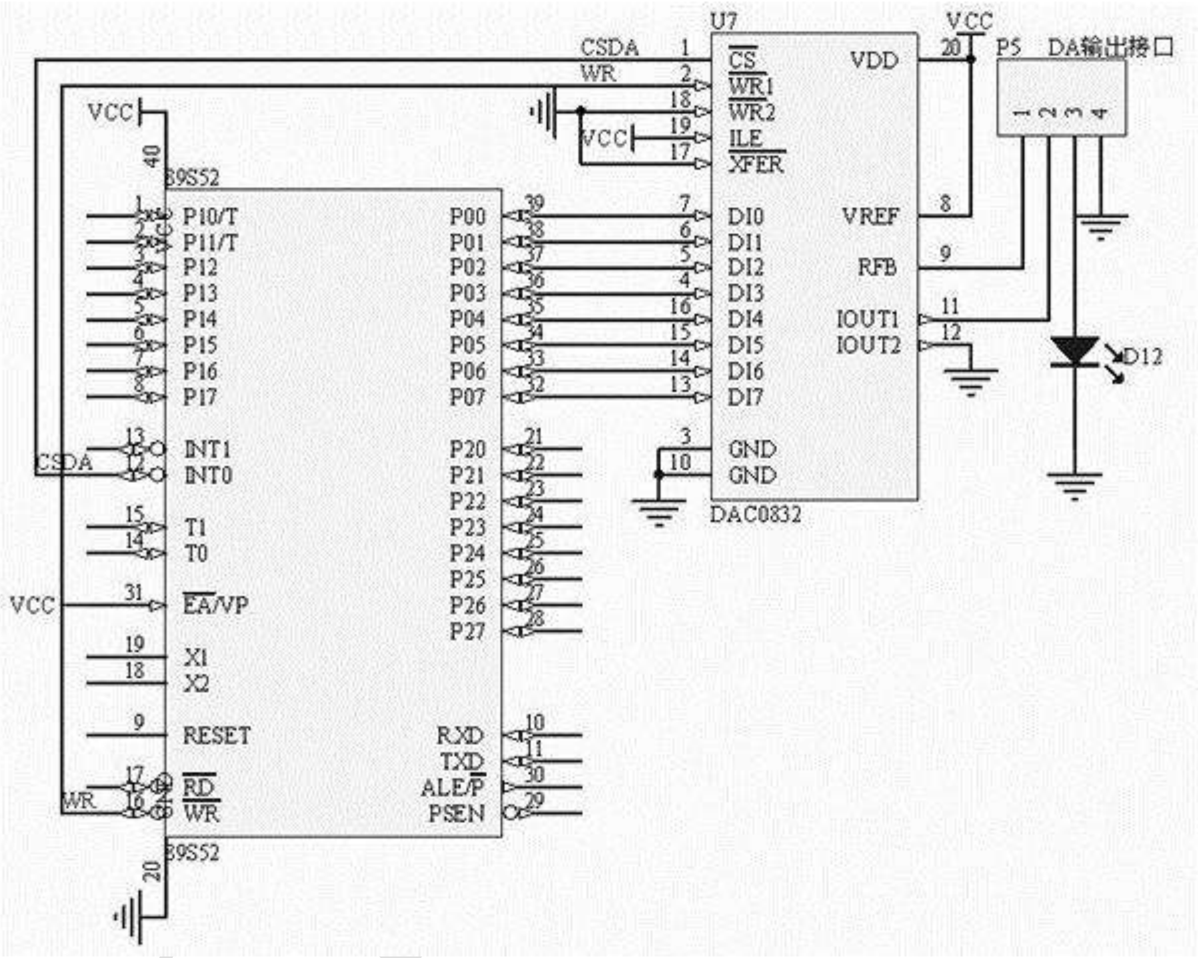


Fig. 3 Dual DAC 0832 interfacing to 8051 Microcontroller

Algorithm

Step 1: Initializing DAC 0832 and Micro-controller.

Step 2: Checking for the data available in the microcontroller ports.

Step 3: Sending the analog data to CRO after conversion.

Step 4: Waiting for the next sample from the microcontroller.

Program:

```
ORG 0000H
                MOV P0,#00H
REPEAT:         CALL SQUARWAVE     ; Generate Square Wave
                CALL TRIWAVE       ; Generate Triangular Wave
                CALL STAIRWAVE     ; Generate Staircase Wave
                JMP REPEAT
SQUARWAVE:      MOV P1,#0FFH
                CALL DELAY2SEC
                MOV P1,#00H
                CALL DELAY2SEC
                RET
TRIWAVE:        MOV R7,#00H
TRIWAVE1:       MOV P1,R7
                INC R7
                CJNE R7,#0FFH,TRIWAVE1
                MOV R7,#0FFH
TRIWAVE2:       MOV P1,R7
                DJNZ R7,TRIWAVE2
                RET
STAIRWAVE:      MOV P1,#00H
                CALL DELAY2SEC
                MOV P1,#20H
                CALL DELAY2SEC
                MOV P1,#40H
                CALL DELAY2SEC
                MOV P1,#80H
```

```
                    CALL DELAY2SEC
                    RET
DELAY1SEC:          MOV R0,#10
DEL2:               MOV R1,#250
DEL1:               MOV R2,#250
                    DJNZ R2,$
                    DJNZ R1,DEL1
                    DJNZ R0,DEL2
                    RET

DELAY2SEC:          MOV R0,#20
DEL22:              MOV R1,#250
DEL21:              MOV R2,#250
                    DJNZ R2,$
                    DJNZ R1,DEL21
                    DJNZ R0,DEL22
                    RET
END
```

Outcome: Observed Square, Triangular and Staircase waveforms on CRO

Experiment 4: Stepper Motor Interface

Objective: To interface stepper motor to 8051 Microcontroller and to make rotations in clockwise and anticlockwise directions

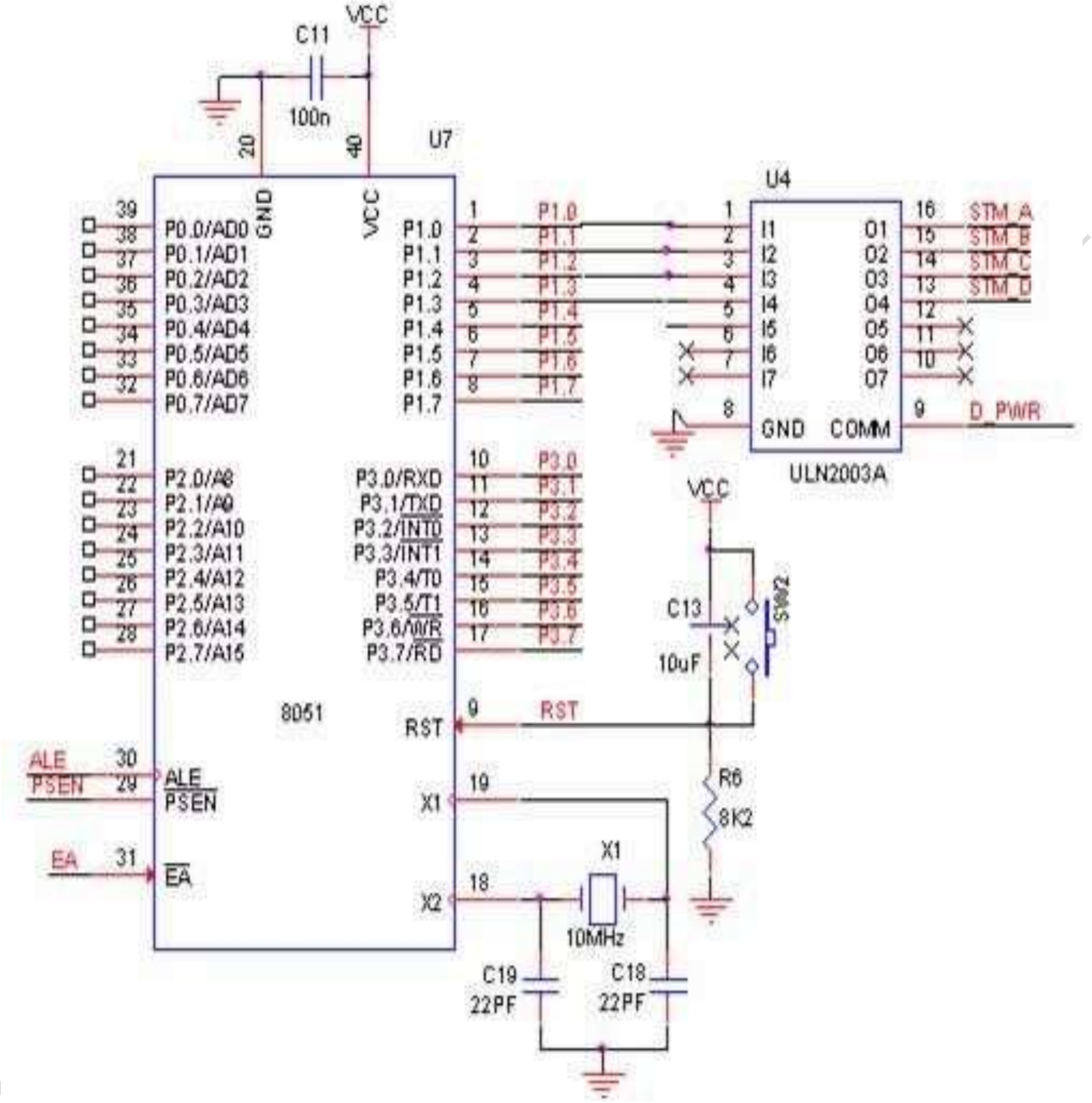


Fig. 4 Stepper motor interfacing with 8051 Microcontroller

Algorithm

1. Initialize the port pins used for the motor as outputs.
2. Write a common delay program.
3. Trigger each bit of Port 1 (P1.0-1.3) continuously to observe the rotations.

Program No.: 4 a

Program: To rotate Stepper Motor Clockwise

```
A1 EQU P1.0
A2 EQU P1.1
A3 EQU P1.2
A4 EQU P1.3
        ORG 00H
        MOV TMOD,#00000001B

MAIN: CLR A1
        ACALL DELAY
        SETB A1

        CLR A2
        ACALL DELAY
        SETB A2

        CLR A3
        ACALL DELAY
        SETB A3

        CLR A4
        ACALL DELAY
        SETB A4
        SJMP MAIN

DELAY:MOV R6,#1D
BACK: MOV TH0,#00000000B
        MOV  TL0,#00000000B
        SETB TR0
HERE2: JNB TF0, HERE2
        CLR TR0
        CLR TF0
        DJNZ R6, BACK
        RET
END
```

Program No.: 4b

Program: To rotate Stepper Motor Anti-Clockwise

```
A1 EQU P1.0
A2 EQU P1.1
A3 EQU P1.2
A4 EQU P1.3
        ORG 00H
        MOV TMOD,#00000001B
MAIN: CLR A1
        ACALL  DELAY
        SETB A4

        CLR A2
        ACALL DELAY
        SETB A3

        CLR A3
        ACALL DELAY
        SETB A3

        CLR A4
        ACALL DELAY
        SETB A1
        SJMP MAIN
DELAY:MOV R6,#1D
BACK: MOV TH0,#00000000B
        MOV  TL0,#00000000B
        SETB TR0
HERE2: JNB TF0,HERE2
        CLR TR0
        CLR TF0
        DJNZ R6,BACK
        RET
        END
```

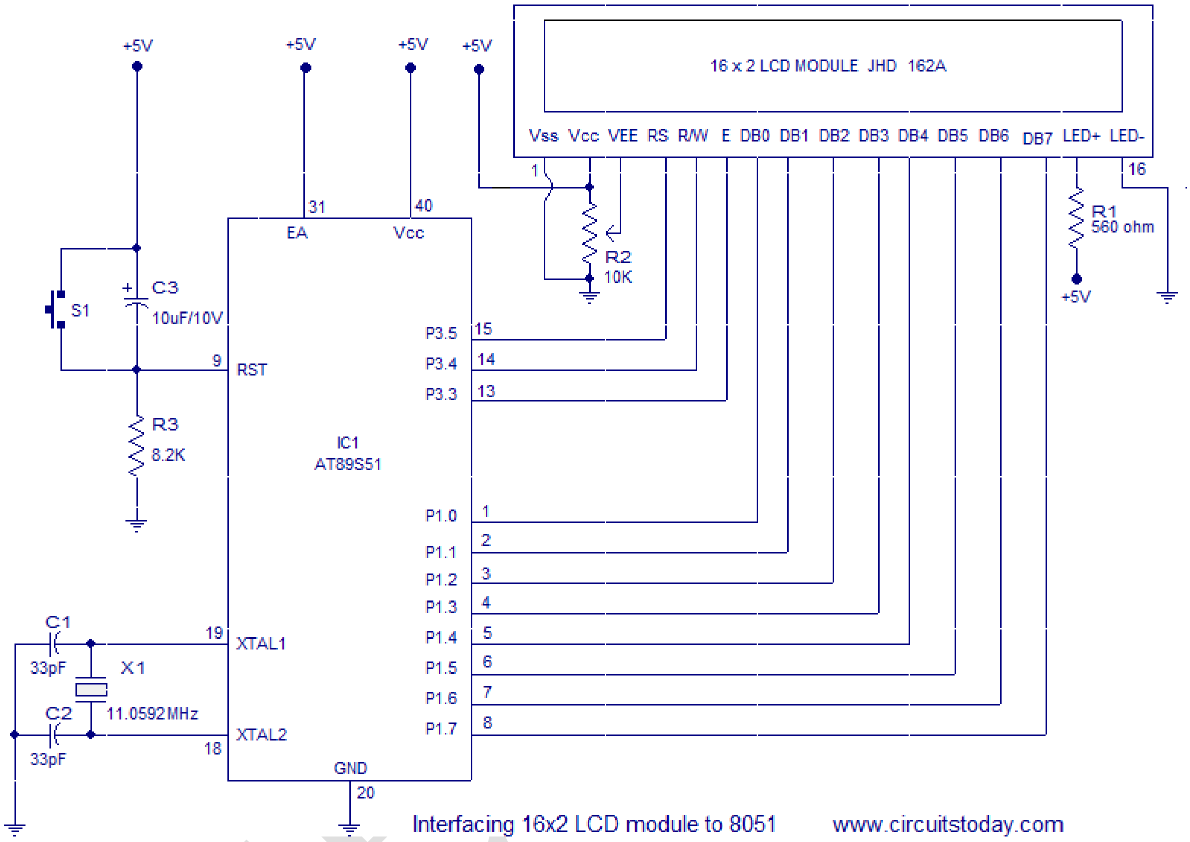
Outcome:

Interfaced Stepper motor and rotated Stepper Motor in both clockwise and anti-clockwise directions.

Experiment 5:

LCD Interface

Objective: To write an ALP to interface 16 X 2 LCD to 8051 Microcontroller to display message



Interfacing 16x2 LCD module to 8051 www.circuitstoday.com

Fig. 5 Interfacing 16 X 2 LCD module to 8051 Microcontroller

Algorithm

LCD initialization

The steps that has to be done for initializing the LCD display is given below and these steps are common for almost all applications.

Step 1: Send 38H to the 8 bit data line for initialization
Step 2: Send 0FH for making LCD ON, cursor ON and cursor blinking ON.
Step 3: Send 06H for incrementing cursor position.
Step 4: Send 01H for clearing the display and return the cursor.

Sending data to the LCD

The steps for sending data to the LCD module are given below. I have already said that the LCD module has pins namely RS, R/W and E. It is the logic state of these pins that make the module to determine whether a given data input is a command or data to be displayed.

Step 1: Make R/W low.
Step 2: Make RS=0 if data byte is a command and makes RS=1 if the data byte is a data to be displayed.
Step 3: Place data byte on the data register.
Step 4: Pulse E from high to low.
Step 5: Repeat above steps for sending another data.

Program

```
A1 EQU P1.0
A2 EQU P1.1
A3 EQU P1.2
A4 EQU P1.3
        ORG 00H
        MOV A,#38H          // Use 2 lines and 5x7 matrix
        ACALL CMND
        MOV A,#0FH          // LCD ON, cursor ON, cursor blinking ON
        ACALL CMND
        MOV A,#01H          //Clear screen
        ACALL CMND
        MOV A,#06H          //Increment cursor
        ACALL CMND
        MOV A,#82H          //Cursor line one , position 2
        ACALL CMND
        MOV A,#3CH          //Activate second line
        ACALL CMND
        MOV A,#49D
        ACALL DISP
        MOV A,#54D
        ACALL DISP
        MOV A,#88D
```

```
ACALL DISP
MOV A,#50D
ACALL DISP
MOV A,#32D
ACALL DISP
MOV A,#76D
ACALL DISP
MOV A,#67D
ACALL DISP
MOV A,#68D
ACALL DISP

MOV A,#0C1H                //Jump to second line, position 1
ACALL CMND

MOV A,#67D
ACALL DISP
MOV A,#73D
ACALL DISP
MOV A,#82D
ACALL DISP
MOV A,#67D
ACALL DISP
MOV A,#85D
ACALL DISP
MOV A,#73D
ACALL DISP
MOV A,#84D
ACALL DISP
MOV A,#83D
ACALL DISP
```

```
        MOV A,#84D

        ACALL DISP

        MOV A,#79D

        ACALL DISP

        MOV A,#68D

        ACALL DISP

        MOV A,#65D

        ACALL DISP

        MOV A,#89D

        ACALL DISP


HERE: SJMP HERE


CMND: MOV P1,A

        CLR P3.5

        CLR P3.4

        SETB P3.3

        CLR P3.3

        ACALL DELY

        RET


DISP:MOV P1,A

        SETB P3.5

        CLR P3.4

        SETB P3.3

        CLR P3.3

        ACALL DELY

        RET


DELY: CLR P3.3

        CLR P3.5
```

```
        SETB P3.4

        MOV P1,#0FFh

        SETB P3.3

        MOV A,P1

        JB ACC.7,DELY


        CLR P3.3

        CLR P3.4

        RET

        END
```

Outcome:

Interfaced 16 X 2 LCD to 8051 µC and observed the given message on the display.

Experiment 6:

ADC 804 Interfaces

Objective: To write an ALP to interface ADC-0804 and convert an analog input connected to it.

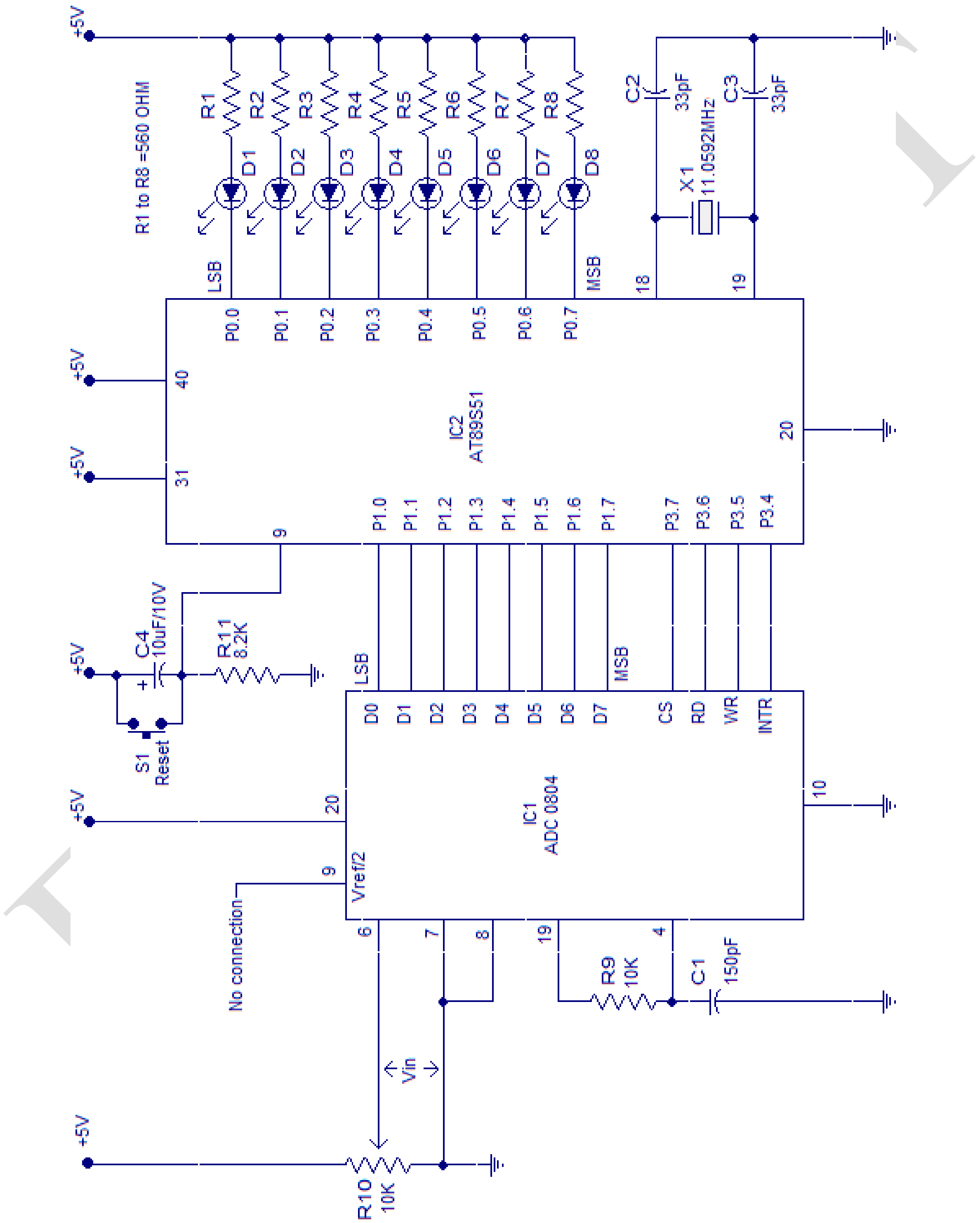


Fig. 6 ADC 804 Interface to 8051 Microcontroller

Algorithm

Step 1: Initiate the circuit with ADC to convert a given analog input ,

Step 2: The circuit accepts the corresponding digital data and displays it on the LED array connected at P0.

Program

```
        MOV P1, #11111111B      // initiates P1 as the input port
MAIN:   CLR P3.7                // makes CS=0
        SETB P3.6               // makes RD high
        CLR P3.5                 // makes WR low
        SETB P3.5                // low to high pulse to WR for starting conversion
WAIT:   JB P3.4, WAIT            // polls until INTR=0
        CLR P3.7                 // ensures CS=0
        CLR P3.6                 // high to low pulse to RD for reading the data from ADC
        MOV A,P1                // moves the digital data to accumulator
        CPL A                   // complements the digital data (*see the notes)
        MOV P0,A                // outputs the data to P0 for the LEDs
        SJMP MAIN               // jumps back to the MAIN program
        END
```

Outcome:

Interfaced 8 bit ADC to 8051 µC and observed the voltage rating.