JSS MAHAVIDYAPEETHA
JSS SCIENCE AND TECHNOLOGY UNIVERSITY
SRI JAYACHAMARAJENDRA COLLEGE OF ENGINEERING

- Constituent College of JSS Science and Technology University
- Approved by A.I.C.T.E
- Governed by the Grant-in-Aid Rules of Government of Karnataka
- Identified as lead institution for World Bank Assistance under TEQIP Scheme

Department of Electronics and communication

Microcontroller LAB (EC420)

Jan-Apr 2020

Staff in Charge

Dr. Shankariah

Prof. Renuka B S

LAB EXPERIMENTS:

Software programs: To be implemented on 8051 microcontroller
1. Problems related with data transfer and exchange.
2. Problems related with arithmetic and logical operations.
3. Problems related with programming timers in all modes with and without interrupts.
4. Problems related with programming serial communication with and without interrupts.
5. Program related with handling external interrupts.

Hardware programs: To be implemented on 8051 and ARM CORTEX-M3 (using Embedded C)
1. Interface LCD.
2. Interfacing of matrix keypad.
3. Interfacing of ADC and DAC.
4. Interfacing of multi digit 7 segment displays.
5. Interfacing of stepper motor and D C motor.

MICROCONTROLLER LAB

Set of experiments

SET I

1. Write an 8051 assembly level program to add 10 bytes of data.
2. Write an 8051 assembly level program to transfer 10 bytes of data from external RAM location starting with 2000h to internal RAM starting from 30h.
3. Write an 8051 assembly level program to transfer 10 bytes of data from location starting at 30h to location 35h.
4. Write an 8051 assembly level program to transfer 10 bytes of data from location starting at 35h to location 30h.
5. Write an 8051 assembly level program to exchange 10 bytes of data from location starting at 30h with data from location starting from 1000h.
6. Write an 8051 assembly level program to transfer 10 bytes of data starting from location 8000h to location 9000h within the external memory.

SET II

1. Write an 8051 assembly level program to add 'N' bytes of data taking into account the possible carry output.
2. Write an 8051 assembly level program to add 'N' bytes of BCD numbers taking into account the possible carry output.
3. Write an 8051 assembly level program to find the average of 'N' bytes of data.
4. Write an 8051 assembly level program to subtract two BCD numbers.
5. Write an 8051 assembly level program to add two multi-byte numbers.

SET III

1. Write an 8051 assembly level program to count the number of even numbers and number of odd numbers in an array of 'N' bytes of data.
2. Write an 8051 assembly level program to count the number of +ve numbers and number of -ve numbers in an array of 'N' bytes of data.
3. Check whether the given byte of data is present in an array of 'N' bytes of data. If present send 00 in Port 0 else send FF in Port 0.
4. Read the data from Port 1. If P1.1 is at logic 0, find the largest number in an array of 'N' bytes of data and store in location 40h. If P1.0 is at logic 1, find the smallest number in the array and store in the location 40h.

SET IV

1. Write an 8051 assembly level program to arrange an array of 'N' bytes of data in ascending order.
2. Write an 8051 assembly level program to arrange an array of 'N' bytes of data in descending order.
3. Write an 8051 assembly level program to find whether the given number is prime or not. If prime send FF to Port 0 else send 00 to Port 0.
4. Write an 8051 assembly level program to find the factorial of a given number (using recursive procedure).
5. Write an 8051 assembly level program for BCD up counter. Show each count in Port 0 with appropriate delay.

6. Write an 8051 assembly level program for BCD down counter. Show each count in Port 0 with appropriate delay.

SET V

1. Write an 8051 assembly level program to check whether the given byte of data is palindrome. If 'yes' send 00 to Port 0 else send FF to Port 0.
2. Write an 8051 assembly level program to check whether the lower nibble is greater than higher nibble of A. If 'yes' send 00 to Port 0 else send FF to Port 0.
3. Write an 8051 assembly level program to convert 2 digit BCD to ASCII numbers and store them in location 30h(LSB) and 31h(MSB).
4. Write an 8051 assembly level program to find the square of a number using look up table technique.
5. Write an 8051 assembly level program to find the square root of a number.

SET VI

1. Write an 8051 assembly level program to find LCM and HCF of two numbers.
2. Write an 8051 assembly level program to check whether the given number is 2 out of 5 code. If 'yes' send 00 to Port 0 else send FF to Port 0.
3. Write an 8051 assembly level program to generate Fibonacci series.

SET VII

1. Write an 8051 assembly level program to generate square wave on P1.5 with 50% duty cycle. Program timers in mode 0, mode1 and mode2 to generate the delay.
2. Write an 8051 assembly level program to generate square wave with ON period of 20ms and OFF period of 40ms. Use timers in mode 0, mode 1 and mode 2 to generate the delay.
3. Repeat the problems with interrupt for timers.
4. Repeat the above problems by writing programs in embedded C.

SET VIII

1. A switch is connected to P2.5. write an 8051 assembly level program to read the status of switch and if switch is closed send serially 'HELLO', else send 'WELCOME' at baud rate 9600.
2. Write an 8051 assembly level program to transfer a message SJCE serially by programming serial communication in interrupt mode with baud rate 9600.
3. Repeat the problem 1 & 2 using embedded C program.

SET IX

1. Interface 4 digit multiplexed 7 segments LED and writes a program to display the message SJCE.
2. Interface 4x4 hex keyboards and write a program to read the key closure and display hex code for key pressed on 7 segment display**.**

SET X

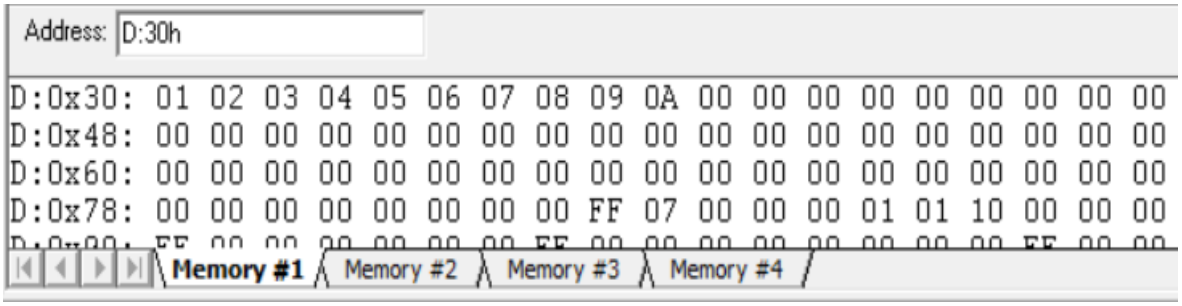
1. Interface LCD module and write program to display a message.
2. Interface ADC a write a program to sample the signal and convert it into digital
3. Interface DAC and write program to generate various waveforms.

SET 1 PROGRAMS

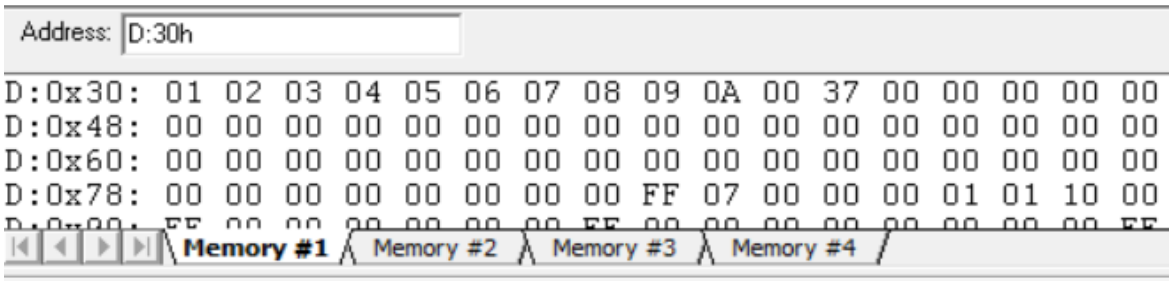
1. WRITE AN 8051 ALP TO ADD 10 BYTES OF DATA

```
;Write an 8051 ALP to add 10 bytes of data.
        org 000h
        mov r0, #30h
        mov a, #00h
        mov r2, #0ah
        mov r3, #00h
loop:   add a, @r0
        jnc next
        inc r3
next:   inc r0
        djnz r2, loop
        inc r0
        mov @r0, a
        inc r0
        mov a, r3
        mov @r0, a
        end
```

Before Execution:



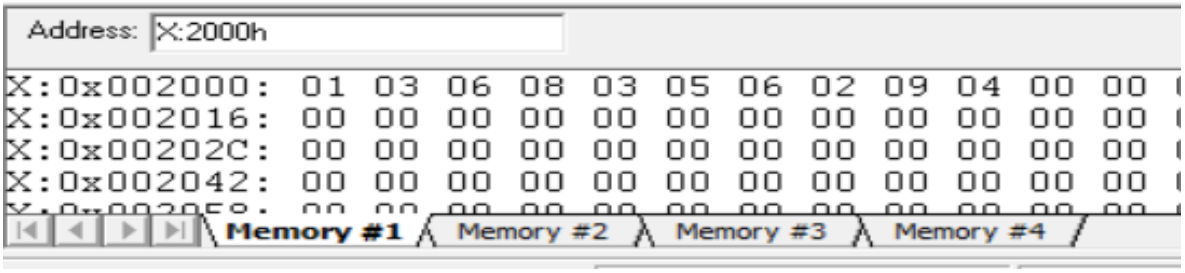
After Execution:



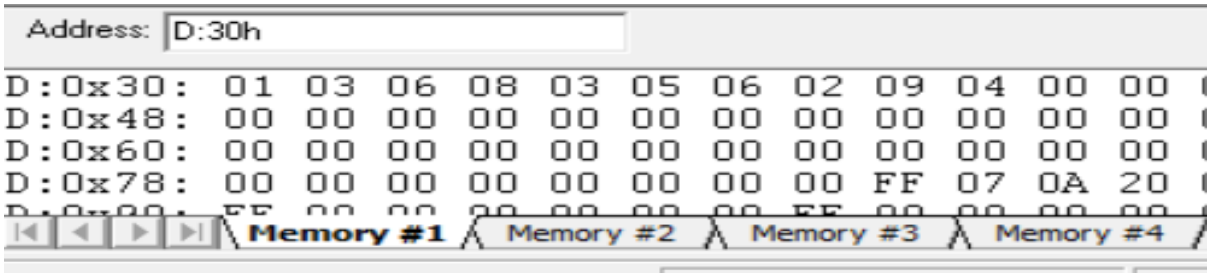
2 Write an 8051 assembly level program to transfer 10 bytes of data from external RAM location starting with 2000h to internal RAM starting from 30h

```
;ALP to tranfer 10 bytes of data from external RAM
;location starting from 2000h to internal RAM location
;starting from 30h

org 000h
    mov dptr,#2000h
    mov r1,#30h
    mov r0,#0Ah
    mov a,#00h
rpt:    movx a,@dptr
    mov @r1,a
    inc r1
    inc dptr
    djnz r0,rpt
    end
```

Before Execution:



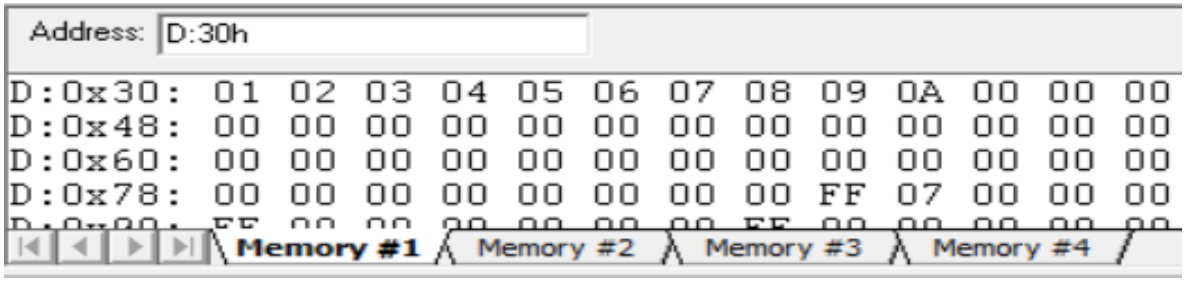
After Execution:



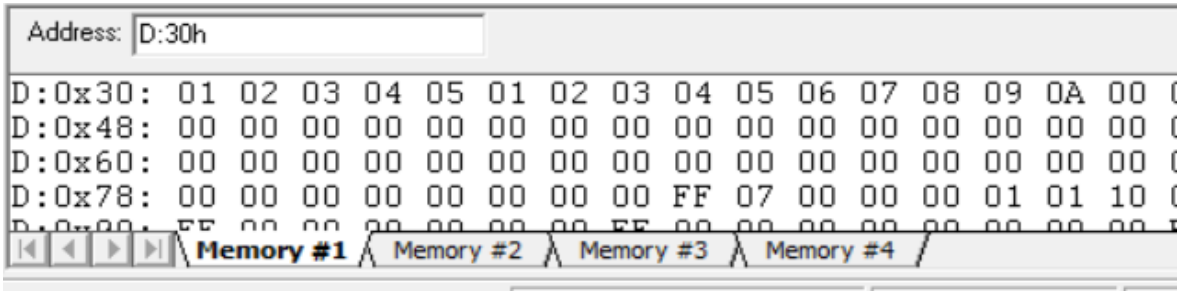
3.Write an 8051 assembly level program to transfer 10 bytes of data from location starting at 30h to location 35h.

```
01  ; Write an 8051 ALP to transfer 10 bytes of data
02  ; from location starting at 30h to location 35h.
03          org 000h
04          mov r0, #30h
05          mov r1, #35h
06          mov r2, #10h
07          mov a, r2
08          add a, r0
09          dec a
10          mov r0, a
11          mov a, r2
12          add a, r1
13          dec a
14          mov r1, a
15  repeat: mov a, @r0
16          mov @r1, a
17          dec r0
18          dec r1
19          djnz r2, repeat
20          end
```

Before Execution :



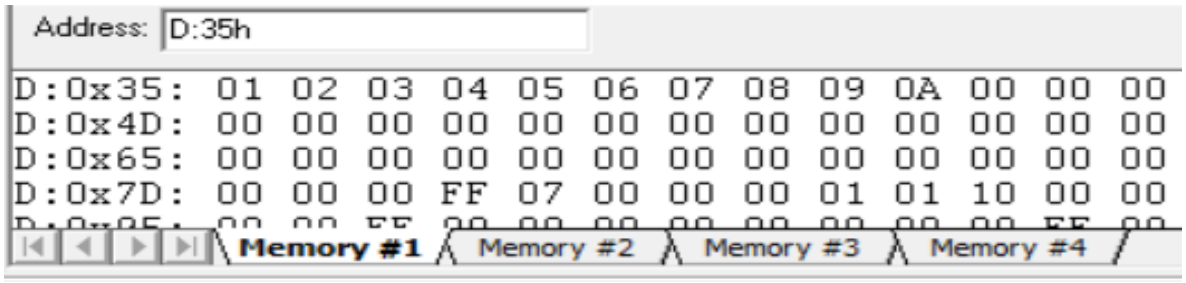
After Execution:



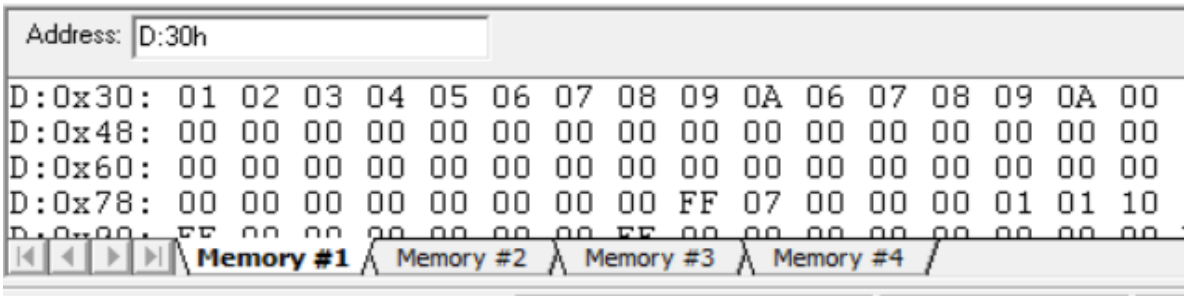
4.Write an 8051 assembly level program to transfer 10 bytes of data from location starting at 35h to location 30h.

```
; Write an 8051 ALP to transfer 10 bytes of data from
; location starting at 35h to location 30h.

        org 000h
        mov r0, #35h
        mov r1, #30h
        mov r2, #0ah
repeat: mov a, @r0
        mov @r1, a
        inc r0
        inc r1
        djnz r2, repeat
        end
```

Before Execution:



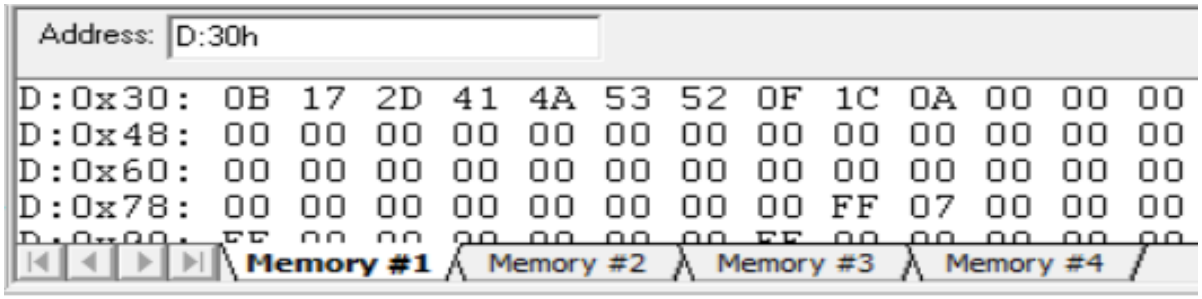
After execution:



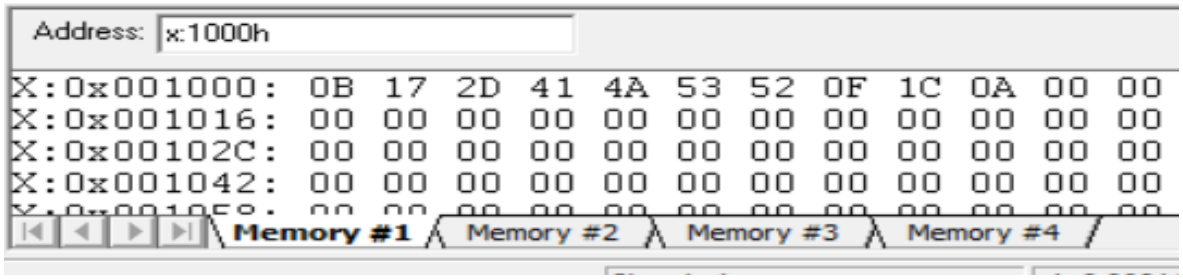
3. **Write an 8051 assembly level program to exchange 10 bytes of data from location starting at 30h with data from location starting from 1000h.**

```
; Write an 8051 assembly level program to exchange 10 bytes
; of data from location starting
; at 30h with data from location starting from 1000h.

        org 000h
        mov r0, #30h
        mov dptr, #1000h
        mov r2, #0ah
repeat: movx a, @dptr
        xch a, @r0
        movx @dptr, a
        inc r0
        inc dptr
        djnz r2, repeat
        end
```

Before Execution
1000h:0BH,17H,2DH,41,4AH,53,52H,0FH,1CH,0AH
30H: 01H,02H,3H,4H,5H,6H,7H,8H,9H,0AH



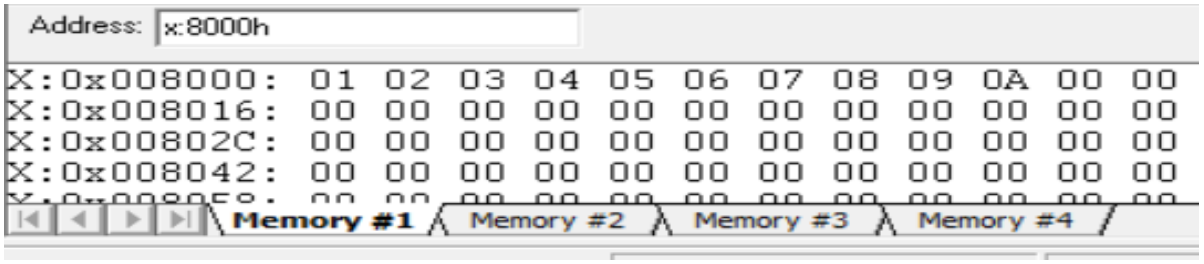
After Execution:
30H: 0BH, 17H, 2DH, 41, 4AH, 53, 52H, 0FH, 1CH, 0AH
1000H:01H,02H,3H,4H,5H,6H,7H,8H,9H,0AH



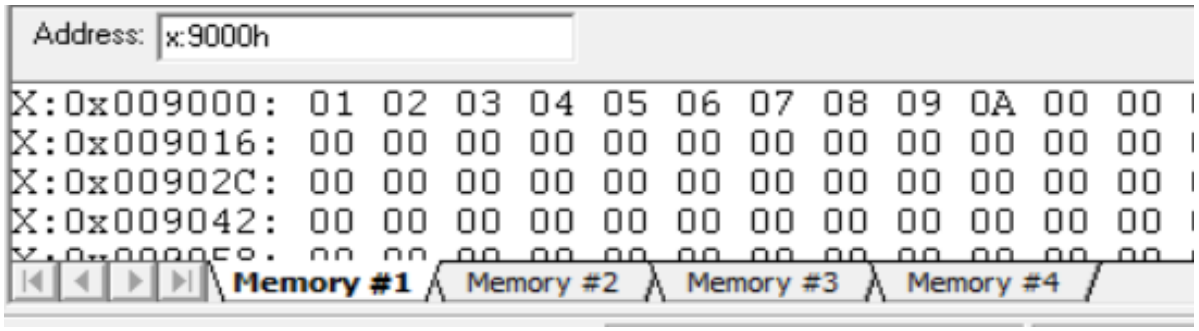
4. Write an 8051 assembly level program to transfer 10 bytes of data starting from location 8000h to location 9000h within the external memory.

```
;  Write an 8051 assembly level program to transfer
; 10 bytes of data starting from location
; 8000h to location 9000h within the external memory.

org 000h
     mov dptr,#8000h
     mov r0,#00h
     mov r1,#90h
     mov r2,#0Ah
rpt:     movx a,@dptr
     push dpl
     push dph
     mov r3,a
     mov a,r1
     mov dph,a
     mov a,r0
     mov dpl,a
     mov a,r3
     movx @dptr,a
     inc dptr
     mov a,dph
     mov r1,a
     mov a,dpl
     mov r0,a
     pop dpl
     pop dph
     inc dptr
     djnz r2,rpt
     end
```

Before Execution:



After Execution:

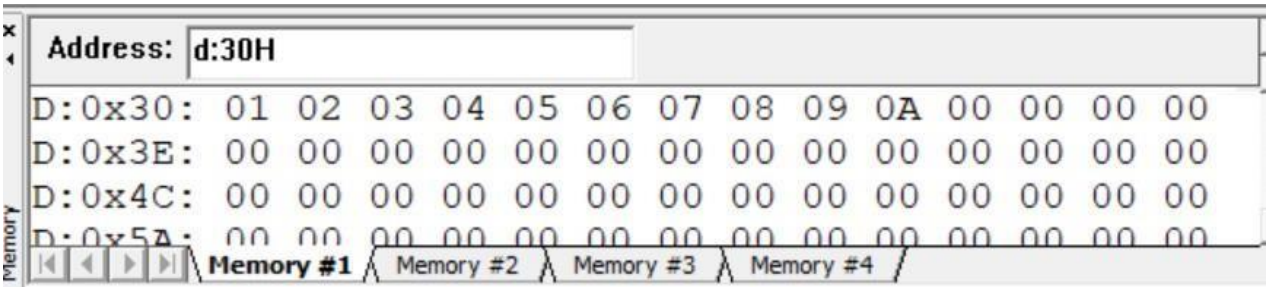


SET 2 PROGRAMS

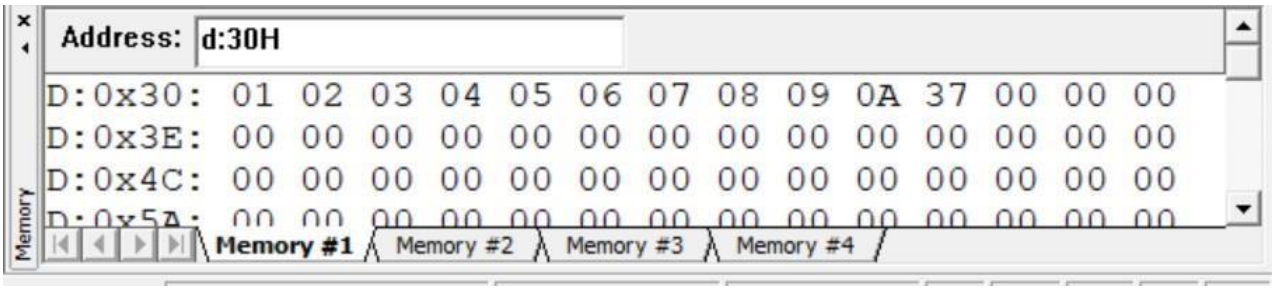
1. Write an alp to add N bytes of data taking into account the possible carry output.

```
01      org 000h
02      mov r0,#30h
03      mov r1,#0Ah
04      mov r2,#00h
05      clr a
06
07  rpt: add a,@r0
08      inc r0
09      jnc nocar
10      inc r2
11  nocar:  djnz r1,rpt
12      mov @r0, A
13      inc r0
14      mov a,r2
15      mov @r0,a    ;Resulting Carry stored after Sum
16      end
```

Before Execution:



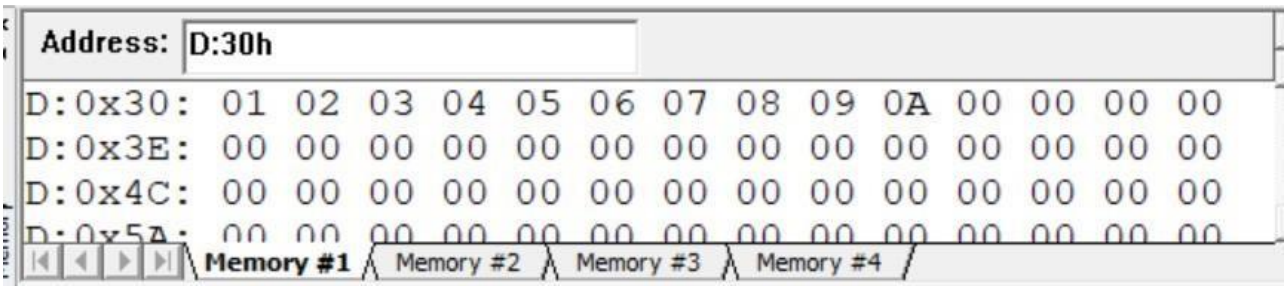
After Execution:



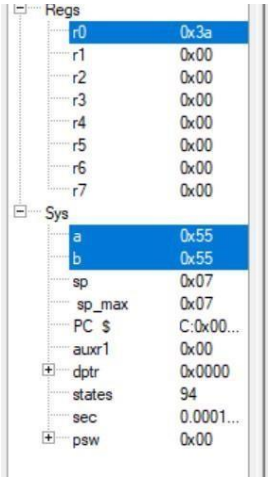
2. Write an alp to add N bytes of BCD numbers talking into account the possible carry output.

```
01              org 000h
02              mov r0,#30h
03              mov r2,#0ah
04              mov r1,#00h
05              mov a,#00h
06  repeat:     add a,@r0
07              da a
08              mov b,a
09              jnc next
10              mov a,r1
11              add a,#01h
12              da a
13              mov r1,a
14  next:       mov a,b
15              inc r0
16              djnz r2,repeat
17              end
18
```

Before Execution:



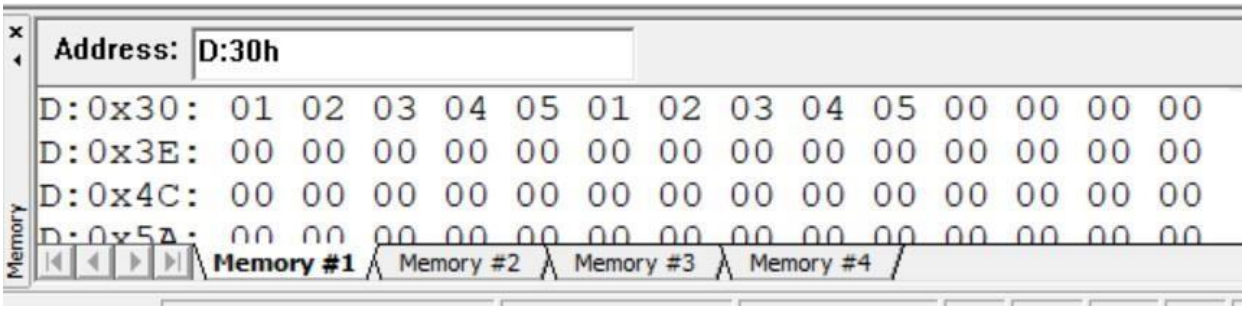
After Execution:



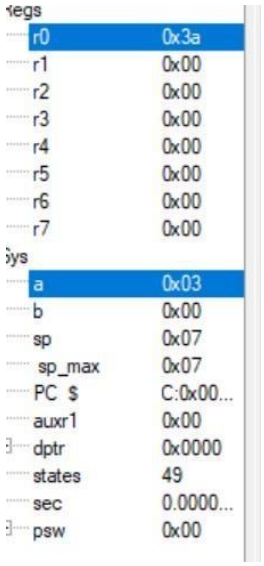
3. **Write an alp to find the average of N bytes of data.**

```
01              org 000h
02              mov r0,#30h
03              mov r1,#0ah
04              mov b,r1
05              mov a,#00h
06    repeat:   add a,@r0
07              inc r0
08              djnz r1,repeat
09              div ab
10              end
```

Before Execution:



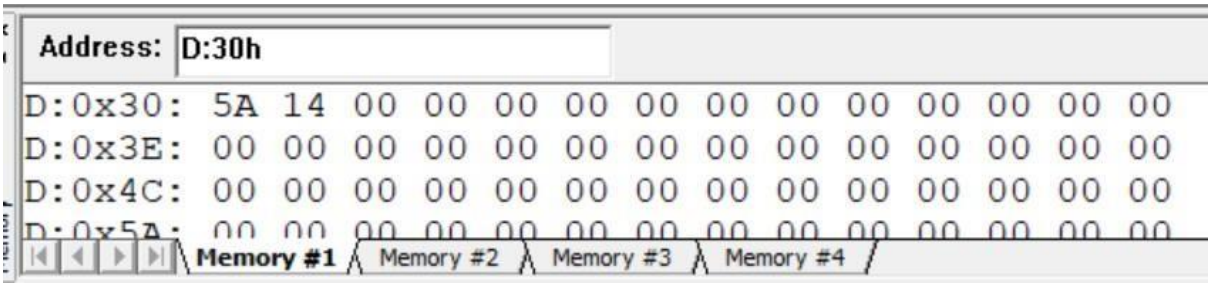
After Execution



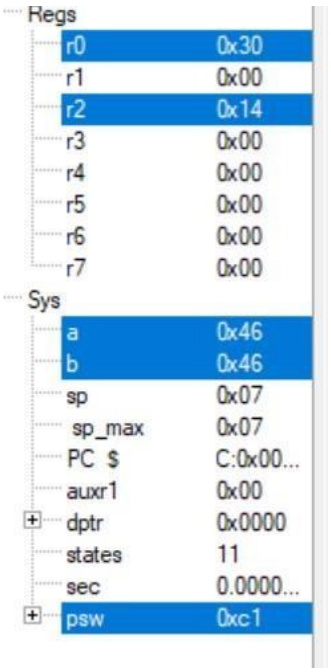
4. Write an alp to subtract two BCD numbers.

```
01  org 0000h
02  mov r0,#30h
03  inc r0
04  mov a,@r0
05  mov r2,a
06  mov a,#99h
07  subb a,r2
08  add a,#01h
09  dec r0
10  add a,@r0
11  da a
12  mov b,a
13  end
```

Before Execution:



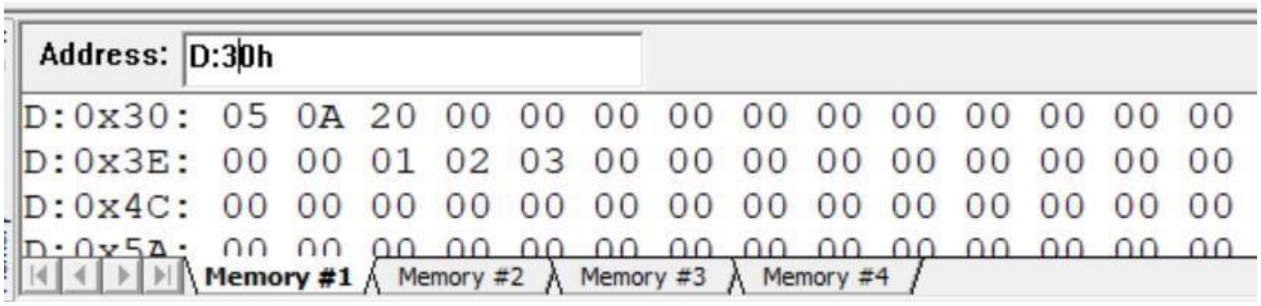
After Execution:



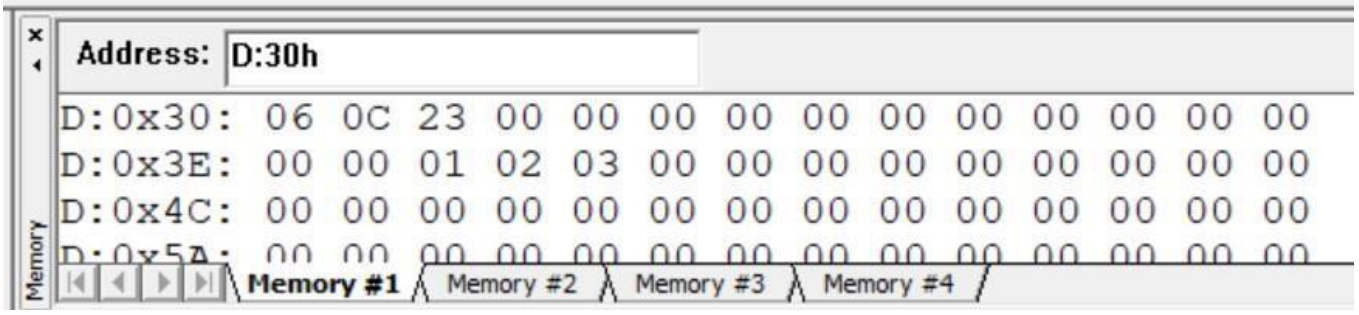
5. Write an alp to add 2 multibyte numbers. Numbers starts from location with address 30h and 40h. Store the results starting from location 30h.

```
01          org 000h
02          mov r0,#30h
03          mov r1,#40h
04          mov r2,#03h
05          clr c
06  repeat: mov a, @r0
07          add a,@r1
08          mov @r0,a
09          inc r0
10          inc r1
11          djnz r2,repeat
12          mov a,#00h
13          addc a,#00h
14          mov @r0,a
15          end
16
```

Before Execution:



After Execution:

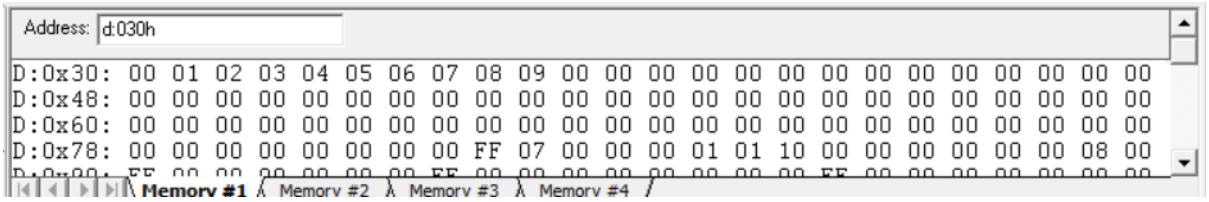


SET 3 PROGRAMS

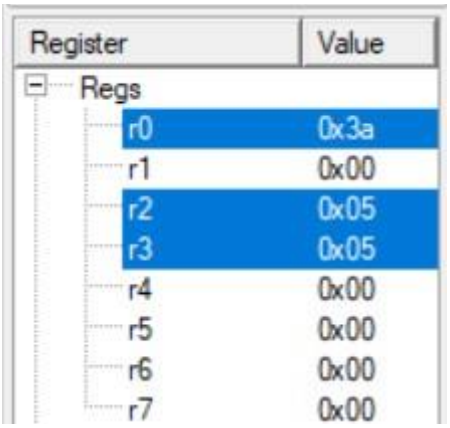
1.Write an 8051 assembly level program to count the number of even Numbers and number of odd numbers in an array of ‘N’ bytes of data

```
01          org 000h
02          mov r0,#30h
03          mov r1,#0ah
04          mov r2,#00h
05          mov r3,#00h
06    loop: mov a,@r0
07          jb acc.0,odd
08          inc r2
09          sjmp next
10    odd:  inc r3
11    next: inc r0
12          djnz r1,loop
13          end
14
15
```

Before execution



After Execution

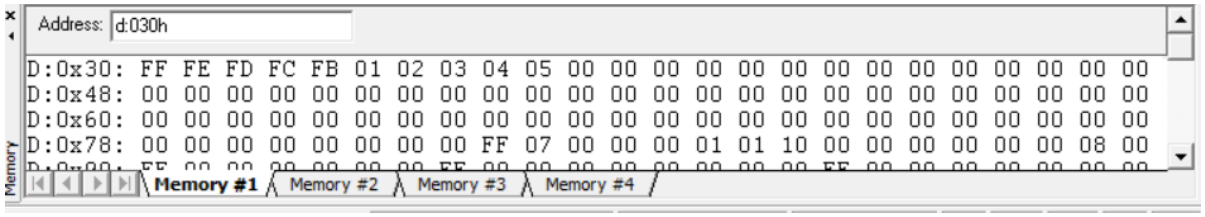


2.Write an 8051 assembly level program to count the number of +ve numbers and number of –ve numbers in an array of ‘N’ bytes of data.

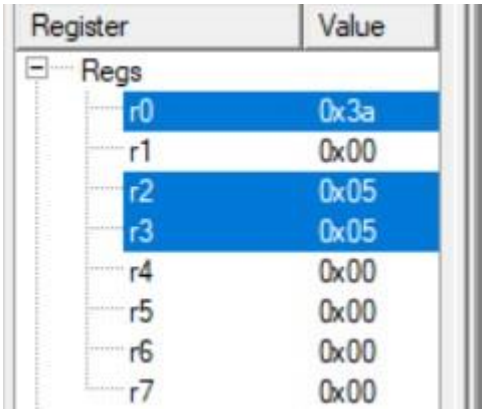
MSB bit of data represents sign of the number. If MSB is 1, then the number is negative.
If MSB is 0, then the number is positive

```
01              org 000h
02              mov r0,#30h
03              mov r1,#0ah
04              mov r2,#00h
05              mov r3,#00h
06    loop:     mov a,@r0
07              jb acc.7,neg
08              inc r2
09              sjmp next
10    neg:      inc r3
11    next:     inc r0
12              djnz r1,loop
13              end
14
```

Before execution:



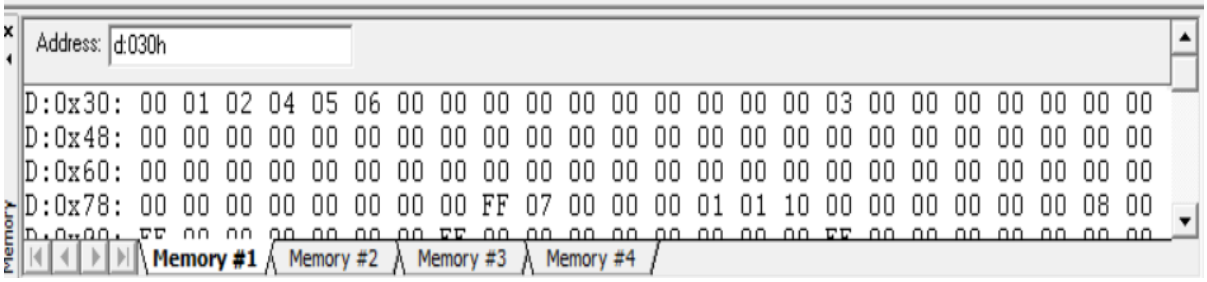
After execution



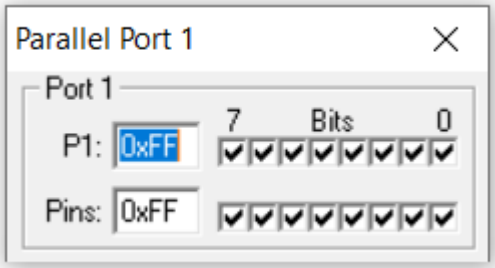
3. Check whether the given byte of data is present in an array of ‘N’ bytes of data. If present send 00 in Port 0 else send FF in Port 0

```
01          org 000h
02          mov r0,#30h
03          mov r1,#0ah
04          mov 40h,#03h
05  repeat:mov a,@r0
06          cjne a,40h,no
07          mov p0,#00h
08          sjmp last
09  no:     inc r0
10          djnz r1,repeat
11          mov p1,#0ffh
12  last:
13          end
```

Before execution:



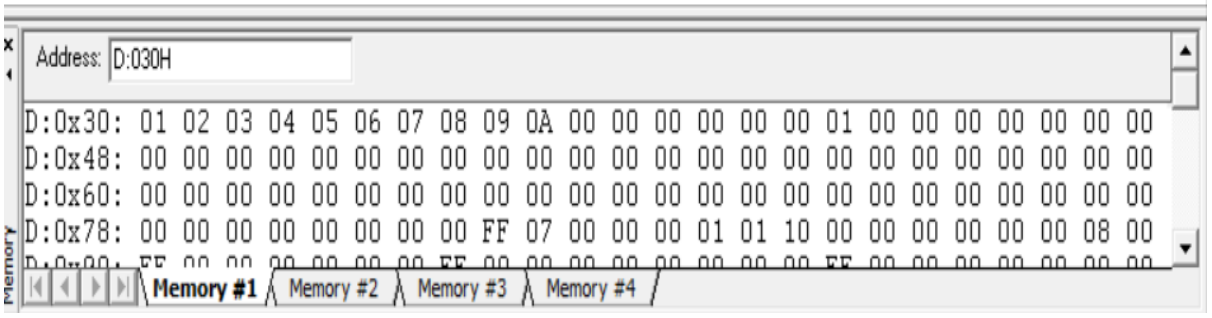
After Execution:



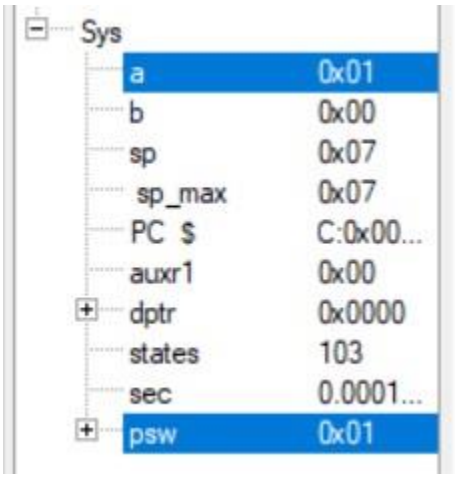
4.Read the data from Port1. IfP1.1 is at logic0, find the largest number in an array of 'N' bytes of data and store in location 40h. If P1.0 is at logic1, find the smallest number in the array and store in the location 40h

```
01                      ORG  000H
02                      MOV  R0,#30H
03                      MOV  R1,#0AH
04                      MOV  A,#00H
05                      MOV  P1,#0FFH
06                      JB   P1.0,SMALLEST
07                      CLR  C
08    REPEAT1:          SUBB A,@R0
09                      JNC  NOEXCH
10                      MOV  A,@R0
11    NOEXCH:           INC  R0
12                      DJNZ R1,REPEAT1
13                      MOV  40H,A
14                      SJMP LAST
15    SMALLEST:         MOV  R0,#30H
16                      MOV  A,@R0
17                      DEC  R1
18    REPEAT2:          INC  R0
19                      MOV  B,@R0
20                      CJNE A,B,NEXT
21    NEXT:             JC   NOEXCH2
22                      MOV  A,@R0
23    NOEXCH2:          CLR  C
24                      DJNZ R1,REPEAT2
25                      MOV  40H,A
26    LAST:             MOV  B,#00H
27                      END
```

Before execution:



After execution:

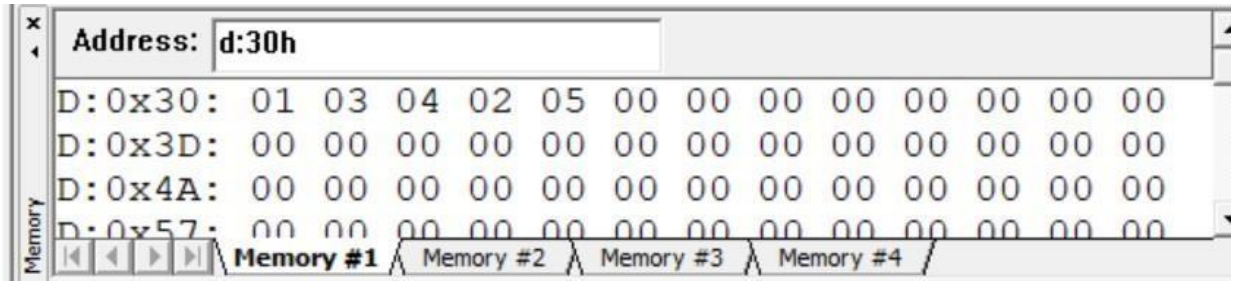


SET 4 PROGRAMS

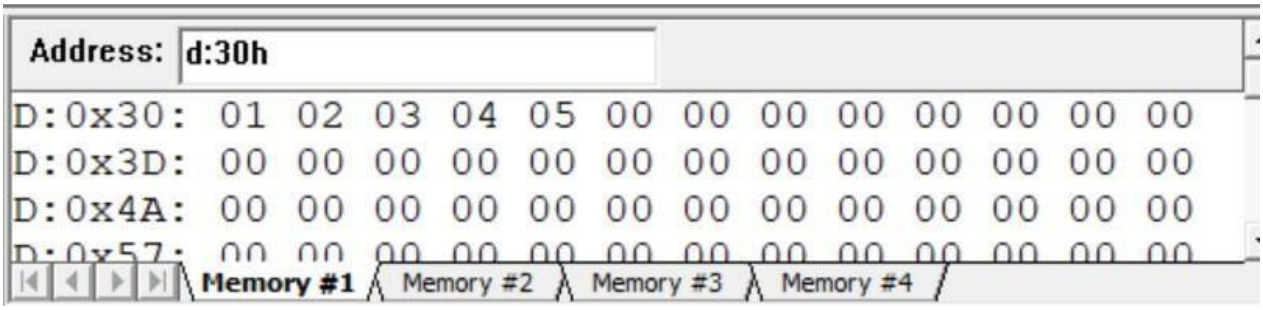
1. Write an 8051 assembly level program to arrange an array of 'N' bytes of data in ascending order.

```
01        org 000h        ;ascending
02        mov r2,#05h     ;Counter
03  rpt1: mov r1,#05h
04        dec r1
05        mov r0,#30h
06  rpt2: mov a,@r0
07        inc r0
08        mov b,@r0
09       cjne a,b,nex1
10  nex1: jc next
11       xch a,b
12       mov @r0,b
13       dec r0
14       mov @r0,a
15       inc r0
16  next:djnz r1,rpt2
17       djnz r2,rpt1
18       end
```

Before Execution:



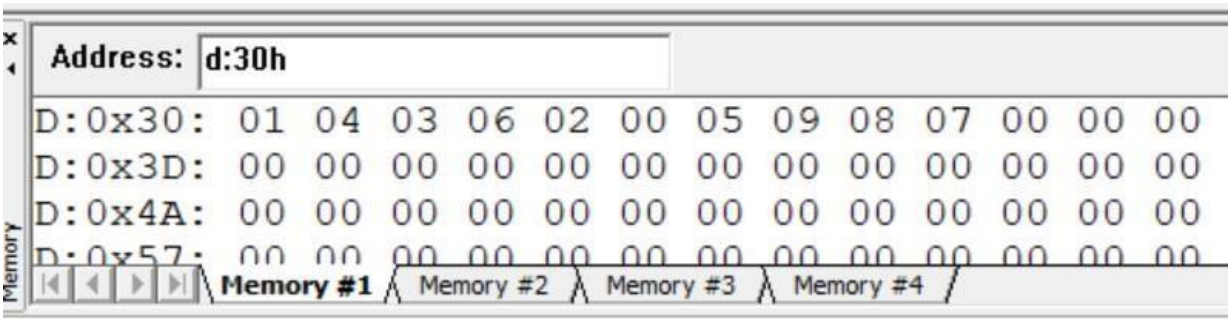
After Execution:



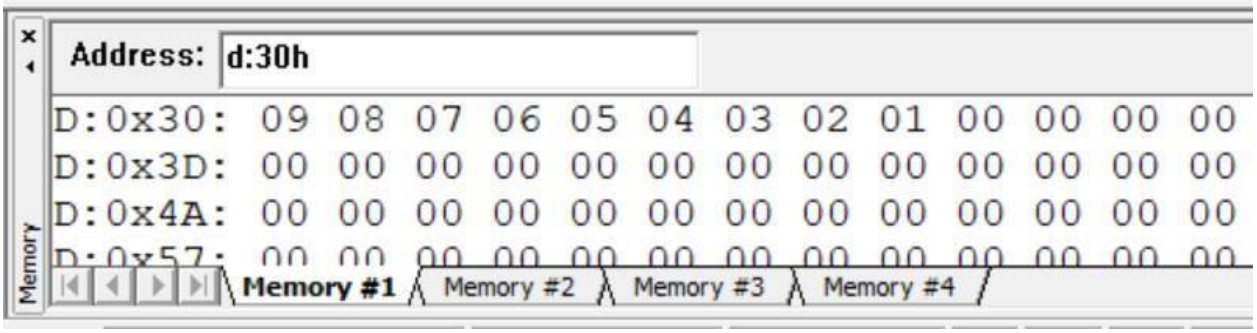
2. Write an 8051 assembly level program to arrange an array of 'N' bytes of data in descending order.

```
01              org 000h     ;descending
02              mov r2,#0Ah    ;Counter
03  again1:     mov r1,#0Ah
04              dec r1
05              mov r0,#30h
06  again2:     mov a,@r0
07              inc r0
08              mov b,@r0
09              cjne a,b,nex1
10  nex1:       jnc next
11              xch a,b
12              mov @r0,b
13              dec r0
14              mov @r0,a
15              inc r0
16  next:       djnz r1,again2
17              djnz r2,again1
18              end
19
```

Before Execution:



After Execution:



3. Write an 8051 assembly level program to find whether the given number is prime or not. If prime send FF to Port 0 else send 00 to Port 0.

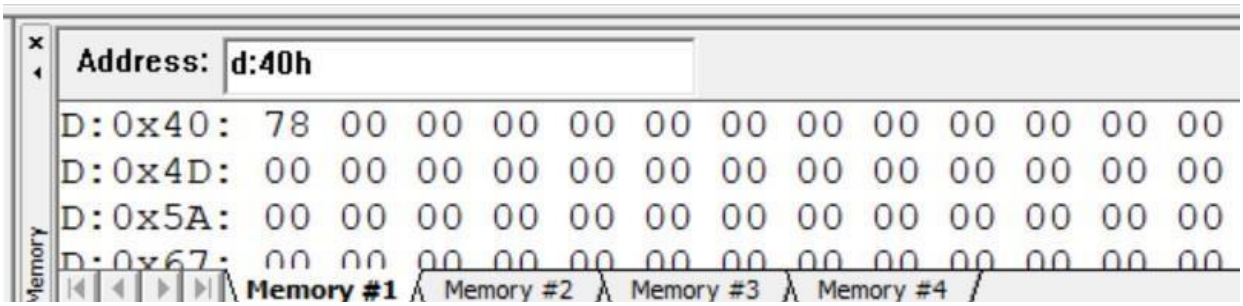
```
01              org 000h          ;prime number
02              mov r1,#02h
03              mov r2,#13       ;Enter decimal 'N' data to be tested
04              cjne r2, #02h, next     ;Check number is less than 2
05      next:   jc prime
06              mov b,#02h
07              mov a,r2
08              div ab
09              mov r0,a
10              inc r0
11      rpt:    mov b,r1
12              mov a,r2
13              div ab
14              xch a,b
15              jz compo         ;Check for divisibility from 2 to 'N/2'
16              inc r1
17              mov a,r1
18              cjne a,00h,rpt
19      prime:  mov p0,#0FFh
20              sjmp done
21      compo:  mov p0,#00h
22      done:
```

After Execution:

4. Write an 8051 assembly level program to find the factorial of a given number (using recursive procedure).

```
01          org 000h    ;factorial
02          mov a,#05h
03          mov r0,a
04          Acall factorial
05          mov 40h,a
06          sjmp last1
07  factorial:dec r0
08          cjne r0,#01h,product
09          sjmp last
10  product: mov b,r0
11          mul ab
12          Acall factorial
13  last:   RET
14  last1:
15          END
16
```

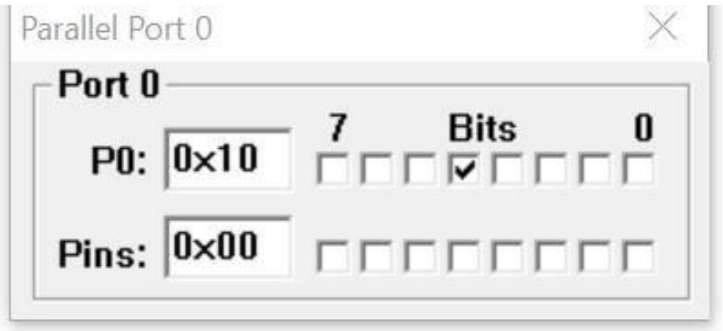
After Execution:



5. Write an 8051 assembly level program for BCD up counter. Show each count in Port 0 with appropriate delay.

```
01          org 000h        ;BCD u
02  again:  mov a,#00h
03  upc:    mov p0,a
04          acall delay
05          add a,#01
06          da a
07          cjne a,#00h,upc
08          sjmp again
09  delay:  mov r1,#0FFh
10  del1:   mov r2,#0FFh
11  del2:   mov r3,#0FFh
12  del3:   djnz r3,del3
13          djnz r2,del2
14          djnz r1,del1
15          ret
16          end
```

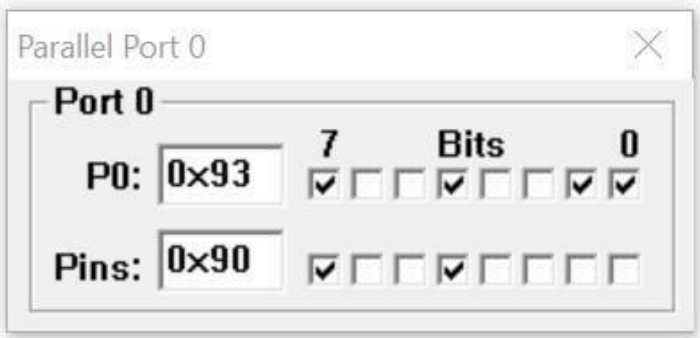
After Execution:



6. Write an 8051 assembly level program for BCD down counter. Show each count in Port 0 with appropriate delay.

```
01          org 000h     ; BCD down counter
02  again:  mov a,#99h
03  upc:    mov p0,a
04          acall delay
05          add a,#99h
06          da a
07          cjne a,#00h,upc
08          sjmp again
09  delay:  mov r1,#0FFh
10  del1:   mov r2,#0FFh
11  del2:   mov r3,#0FFh
12  del3:   djnz r3,del3
13          djnz r2,del2
14          djnz r1,del1
15          ret
16          end
```

After Execution:

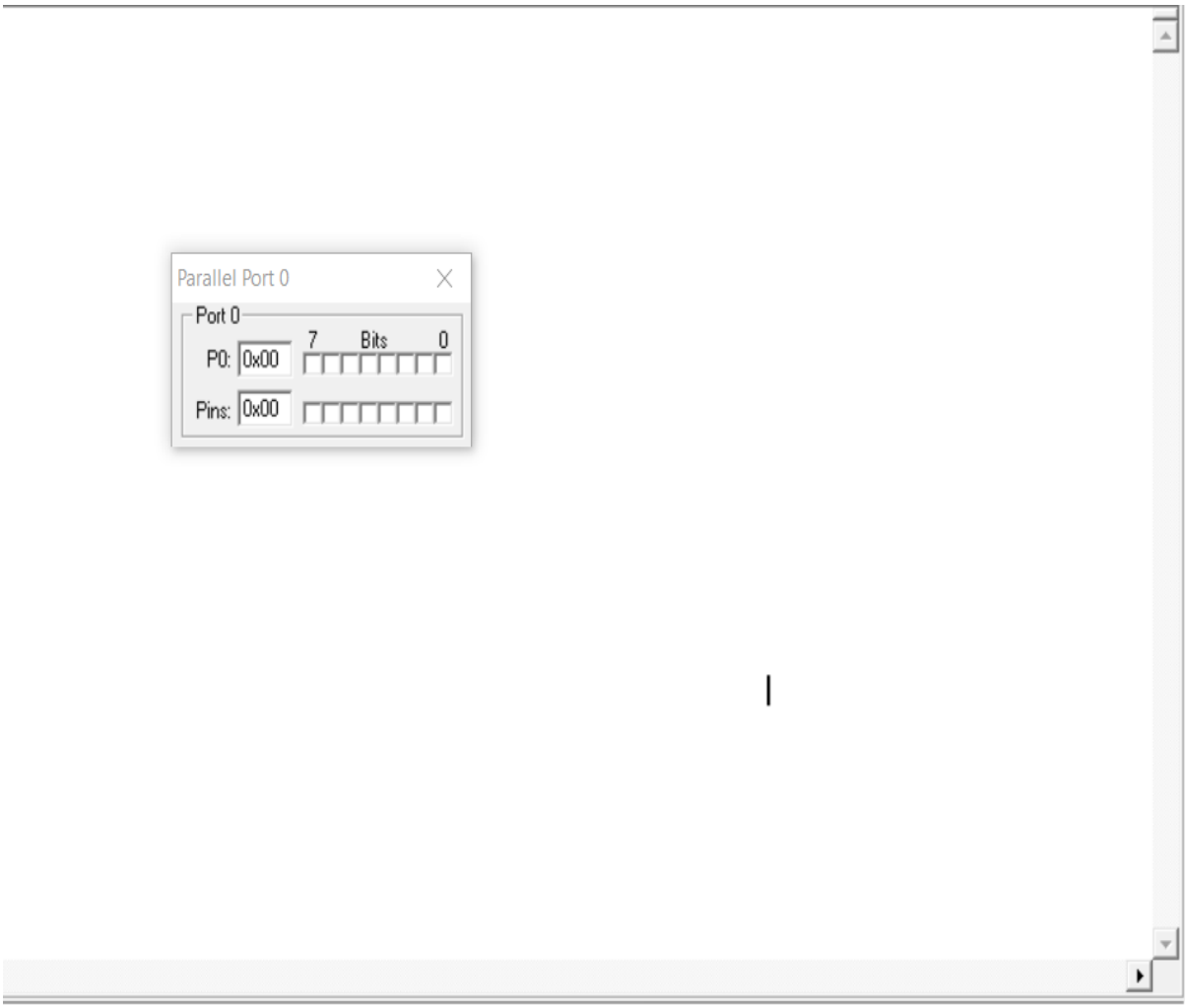


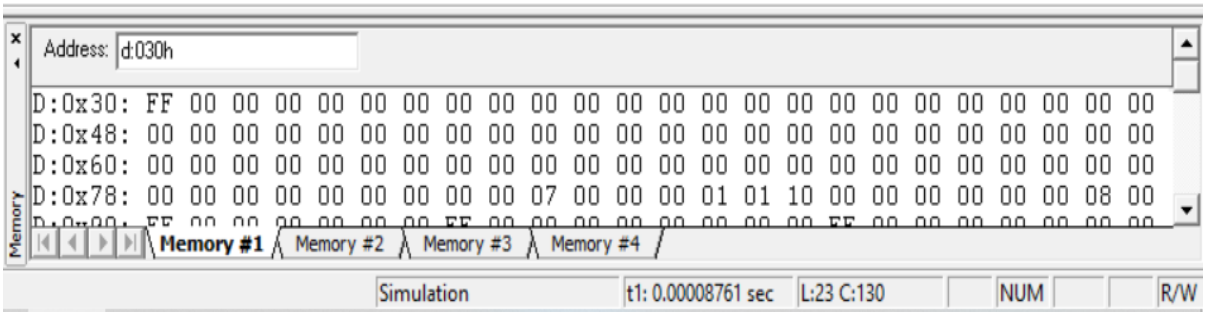
SET 5 PROGRAMS

1. 1.Write an 8051 assembly level program to check whether the given byte of data is palindrome. If 'yes' send 00 to Port 0 else send FF to Port 0

```
01          ORG 000H
02          MOV R0, #30H        ; SOURCE OF DATA
03          MOV R1, #08H        ; NO OF TIMES ROTATION TO BE DONE
04          MOV B, #00H     ; REGISTER TO HOLD ROTATED DATA
05          MOV A, @R0          ; GET THE DATA BYTE
06    repeat:RLC A                    ; BRING D7 BIT TO CY POSITION
07          MOV R2, A             ; SAVE DATA
08          MOV A, B
09          RRC A                   ; GET CY INTO B REGISTER
10          MOV B, A
11          MOV A, R2              ; GET THE DATA FOR ONE MORE ROTATION
12          DJNZ R1, repeat        ; CHECK WHETHER ALL 8 BITS ARE ROTATED
13          MOV A, B
14          ANL A, #0FH           ; MASK D7 TO D0
15          MOV B, A
16          MOV A,@R0
17          ANL A, #0FH
18          CJNE A, B, NO         ; NOT EQUAL IT IS NOT PALINDROME
19          MOV A, #00H
20          MOV P0, A
21          SJMP LAST
22       NO:MOV A, #0FFH
23          MOV P0, A
24     LAST:MOV A,#00H
25          END
26
```

Before and after execution

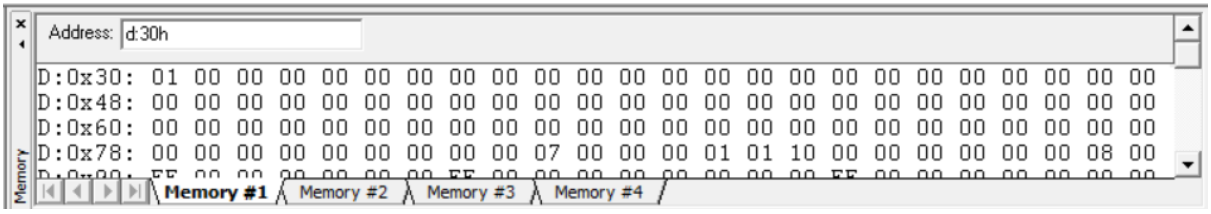

Parallel Port 0
Port 0
P0: 0x00
7 Bits 0
Pins: 0x00


Address: d:030h
D:0x30: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 00 07 00 00 00 01 01 10 00 00 00 00 00 00 08 00
Memory
Memory #1
Memory #2
Memory #3
Memory #4
Simulation
t1: 0.00008761 sec
L:23 C:130
NUM
R/W

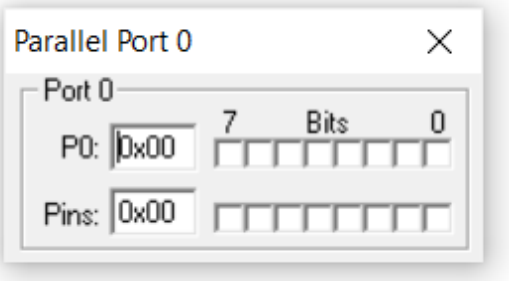
2. Write an 8051 assembly level program to check whether the lower nibble is greater than upper nibble of A. If 'yes' send 00 to Port 0 else send FF to Port 0.

```
01                ORG 000H
02                MOV R0, #30H          ; SOURCE DATA
03                MOV A, @R0            ; GET DATA
04                ANL A, #0FH              ; MASK   UPPER NIBBLE (0000D3D2D1D0)
05                MOV B, A
06                MOV A, @R0
07                SWAP A                   ; SWAP NIBBLES   (D3D2D1D0D7D6D5D4)
08                ANL A, #0FH           ; MASK UPPER NIBBLE (0000D7D6D5D4)
09                CLR C
10                SUBB A, B                ; IS LOWER NIBBLE > UPPER NIBBLE
11                JC  YES               ; IF YES SEND FF TO P0, ELSE 00 TO P0
12                MOV A, #0FFH
13                MOV P0, A
14                SJMP LAST
15  YES:          MOV A, #00H
16                MOV P0, A
17  LAST:
18                END
19
```

Before execution



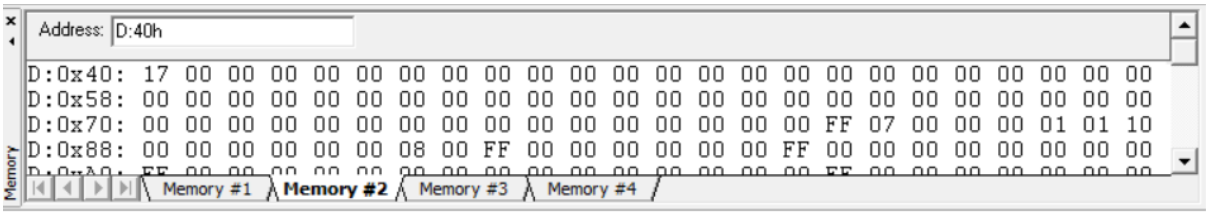
After execution



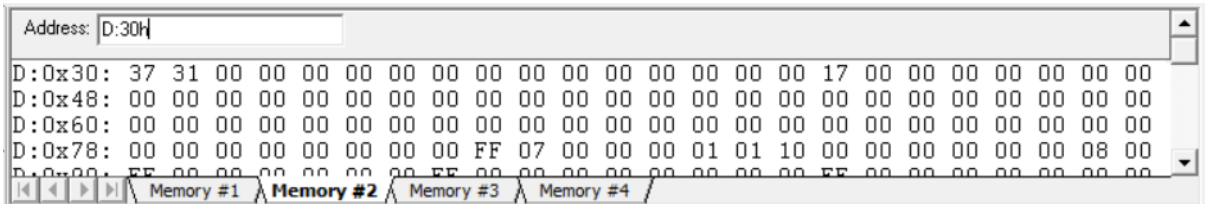
3. Write an 8051 assembly level program to convert 2 digit BCD to ASCII numbers and store them in location 30h(LSB) and 31h(MSB)

```
01          ORG 000H
02          MOV R0, #40H
03          MOV A, @R0          ; GET THE DATA INTO ACC
04          MOV B, A
05          ANL A, #0FH          ; MASK UPPER NIBBLE
06          ADD A, #30H          ; CONVERT TO ASCII
07          MOV 30H, A           ; SAVE LSB
08          MOV A, B               ; GRT THE DATA
09          ANL A, #0F0H         ; MASK LOWER NIBBLE
10          SWAP A                   ; BRING TO LOWER NIBBLE POSITION
11          ADD A, #30H          ; CONVERT IT TO ASCII
12          MOV 31H, A           ; SAVE THE MSB
13          END
14
```

Before execution



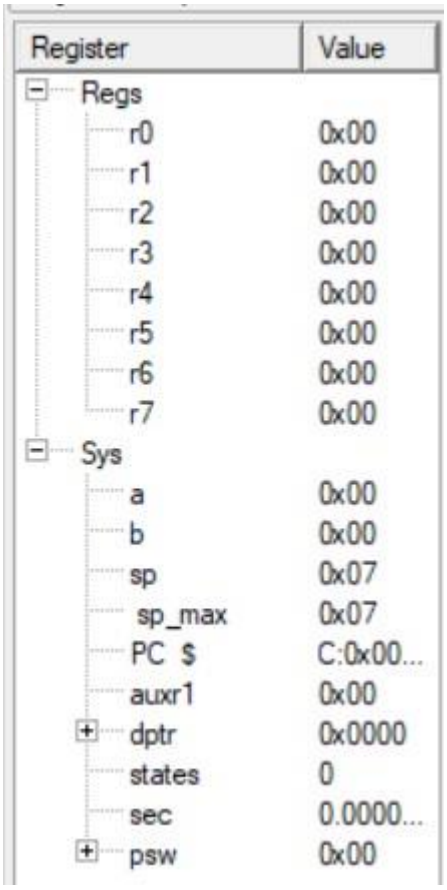
After execution



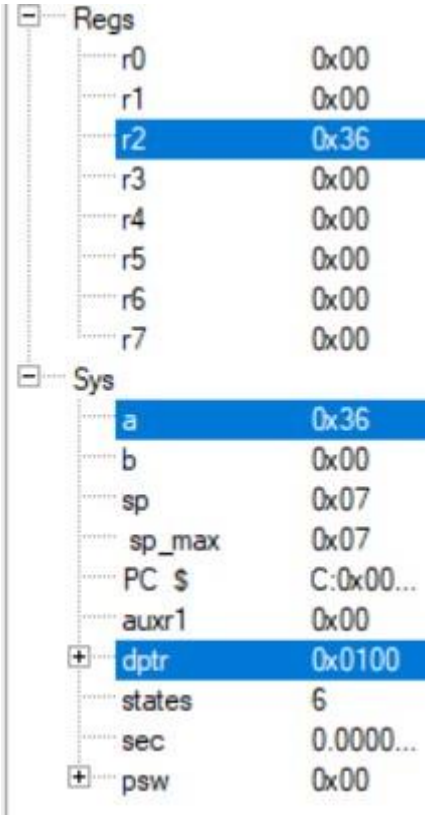
4. Write an 8051 assembly level program to find the square of a number using look up table technique

```
01  ORG 000H
02  MOV DPTR,#100H
03  MOV A,#06
04  MOVC A,@A+DPTR
05  MOV R2,A
06
07
08  ORG 100H
09  SQUARE:DB 00H,01H,04H,09H,16H,25H,36H,49H
10  END
```

Before execution



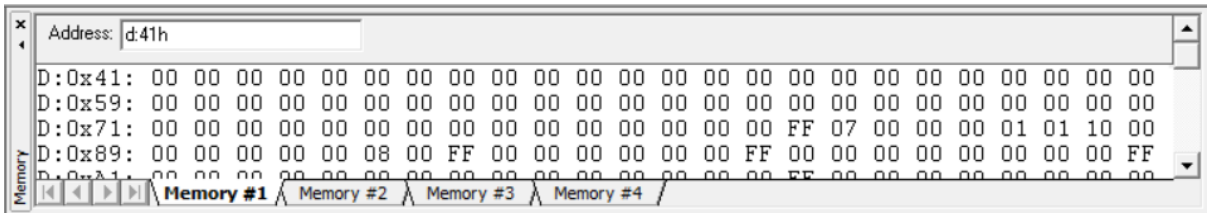
After execution



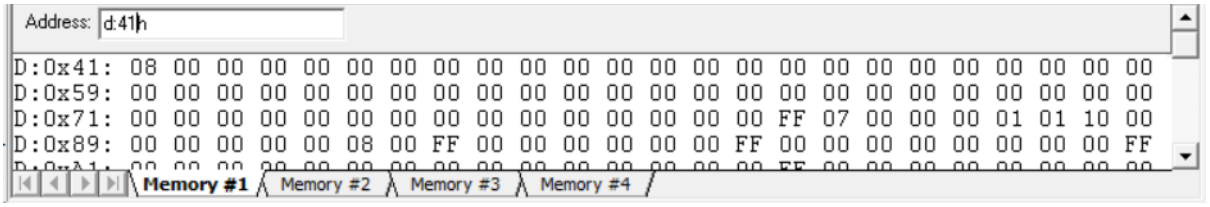
4.Write an 8051 assembly level program to find the square root of a number

```
01  org 000h
02      mov r1,#64   ;Number to be sqaure-rooted
03      mov r0,#01
04  again:  mov b,r0
05      mov a,r0
06      mul ab
07      cjne a,01h,next
08      sjmp jump1
09  next:    jnc jump2
10      inc r0
11      sjmp again
12  jump2:  dec r0
13  jump1:  mov r0,00h
14       mov 41h,r0
15      end
```

Before execution



After execution

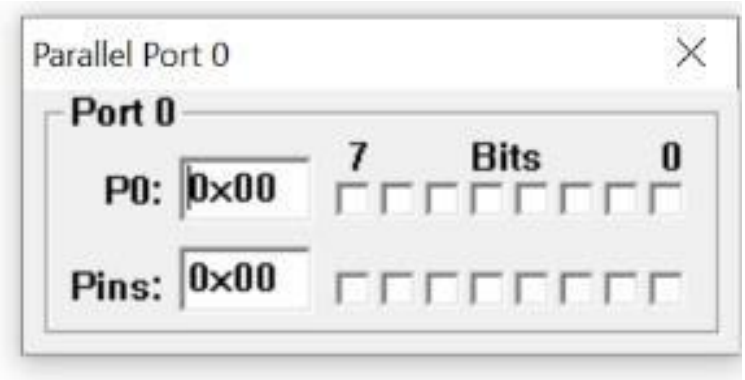


SET 6 PROGRAMS

1.Write a assembly level program (8051) to check if the number is a 2 out of 5 code or not.

```
01      org 000h
02      mov r0,#00h
03      mov r1,#05h
04      mov r2,#03h ;Byte to be tested for 2-out-of-5 code
05      mov a,r2
06      anl a,#0E0h
07      jnz done
08      mov a,r2
09      clr c
10  repeat: rrc a
11      jnc next
12      inc r0
13  next:   djnz r1,repeat
14      cjne r0,#02h,done
15      mov p0,#00h
16      sjmp ext
17  done:   mov p0,#0FFh
18  ext:
19      end
```

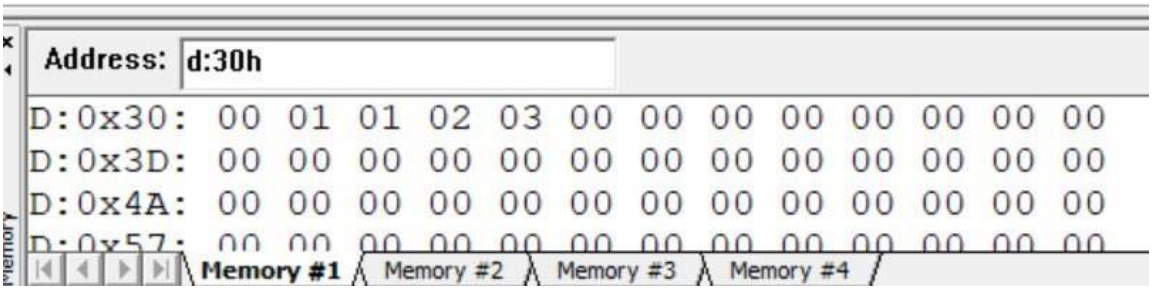
After Execution:



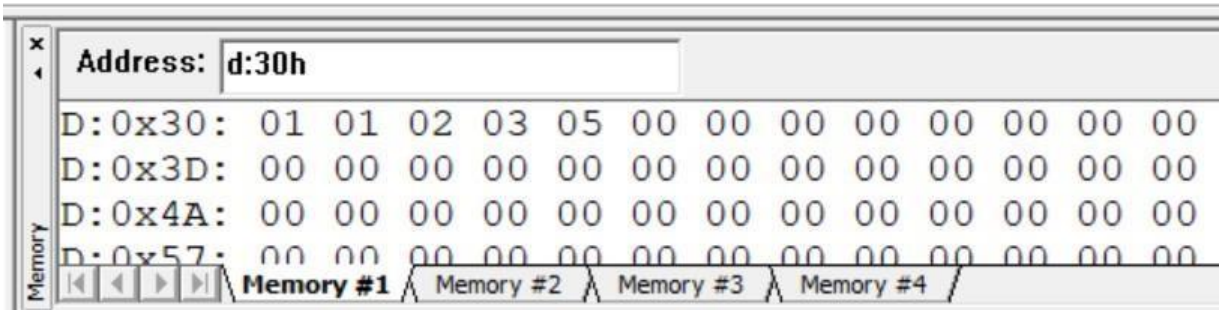
2.Write a assembly level program to find the Fibonacci series.

```
01  ; 3. Write an 8051 ALP to find the fib0nacci series.
02
03      org 000h
04      mov r3,#05h ;Set 'N' number of terms
05      mov a,#00h
06      mov r1,#01h
07      mov r2,#00h
08      mov r0,#30h
09  repeat: add a,r1
10      mov @r0,a
11      inc r0
12      mov r1,02h
13      mov r2,a  ; result will be in r2
14      djnz r3,repeat
15      end
```

Before Execution:



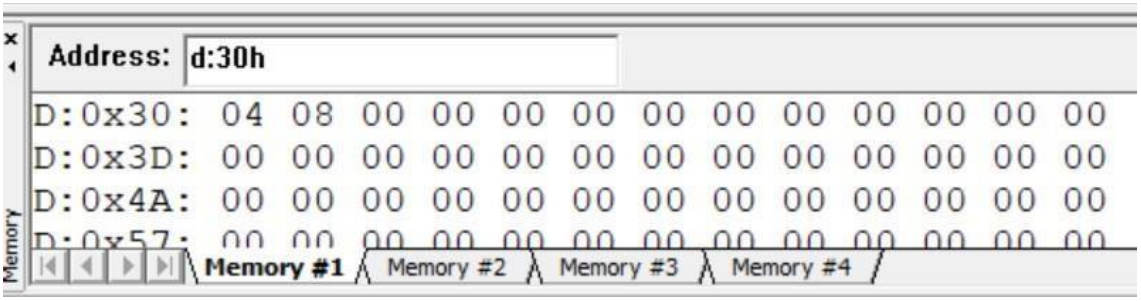
After Execution:



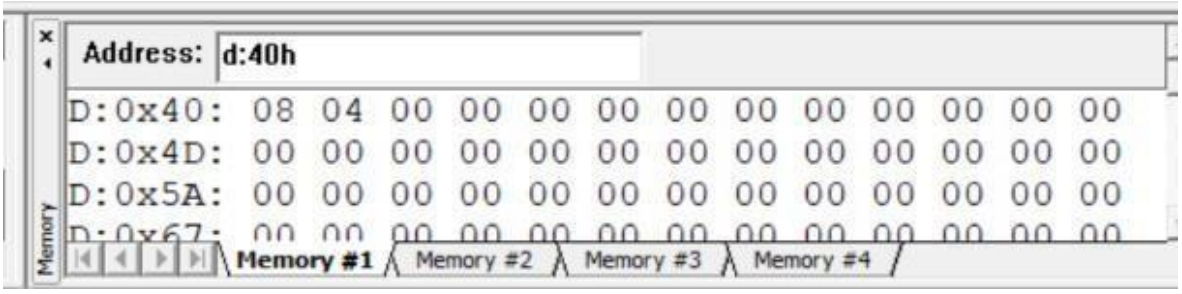
3. Write an assembly level program to find LCM and HCF of 2 numbers.

```
; Write an 8051 ALP to find LCM and HCF of given two numbers.
        org 000h
        mov a, 30h
        mov b, 31h
        mul ab
        mov r2, a
        mov r3, 30h
        mov r4, 31h
loop:   clr c
        mov a, r3
        cjne a, 04h , skip
        sjmp next
skip:   jc no
        clr c
        mov a, r3
        subb a, r4
        mov r3, a
        sjmp yes
no:     clr c
        mov a, r4
        subb a, r3
        mov r4, a
yes:    mov a, r3
        clr c
        cjne a, 04h, loop
next:   mov a, r2
        mov b, r3
        div ab
        mov 40h, a
        mov 41h, r3
        end
```

Before Execution:



After Execution:



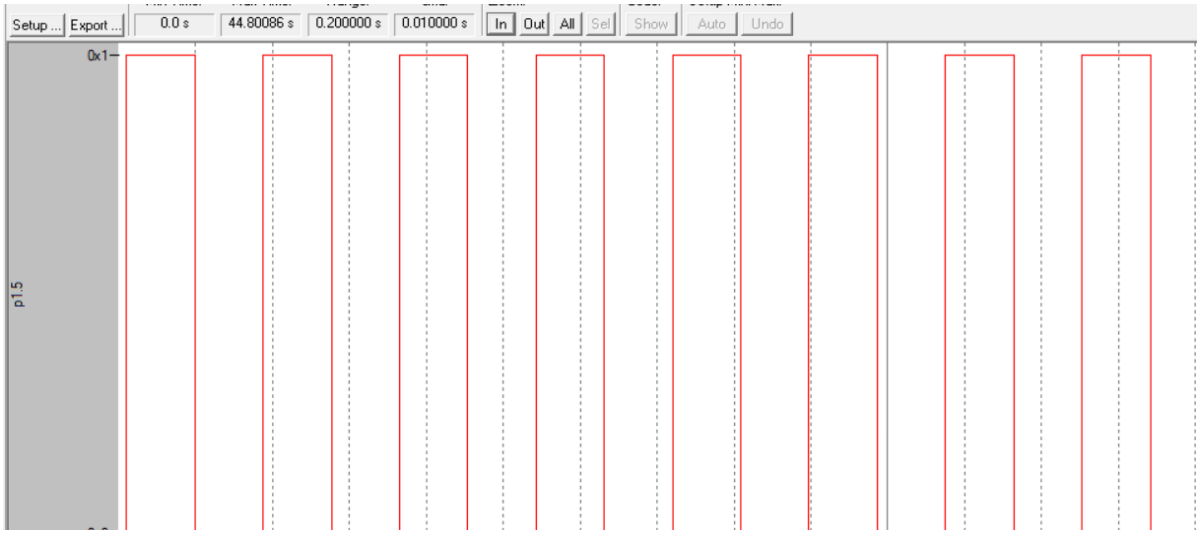
SET 7 PROGRAMS

1. WAP to generate a square wave on P1.5 with 50% Duty Cycle. Program timers in mode0, mode1 and mode2 to generate delay.

Mode 0

```
        ORG 00H
        MOV TMOD,#00H
HERE:   MOV TL0,#04EH
        MOV TH0,#54H
        CPL P1.5
        ACALL DELAY
        SJMP HERE
DELAY:  SETB TR1
WAIT:   JNB TF1,WAIT
        CLR TF1
        CLR TR1
        RET
        END
```

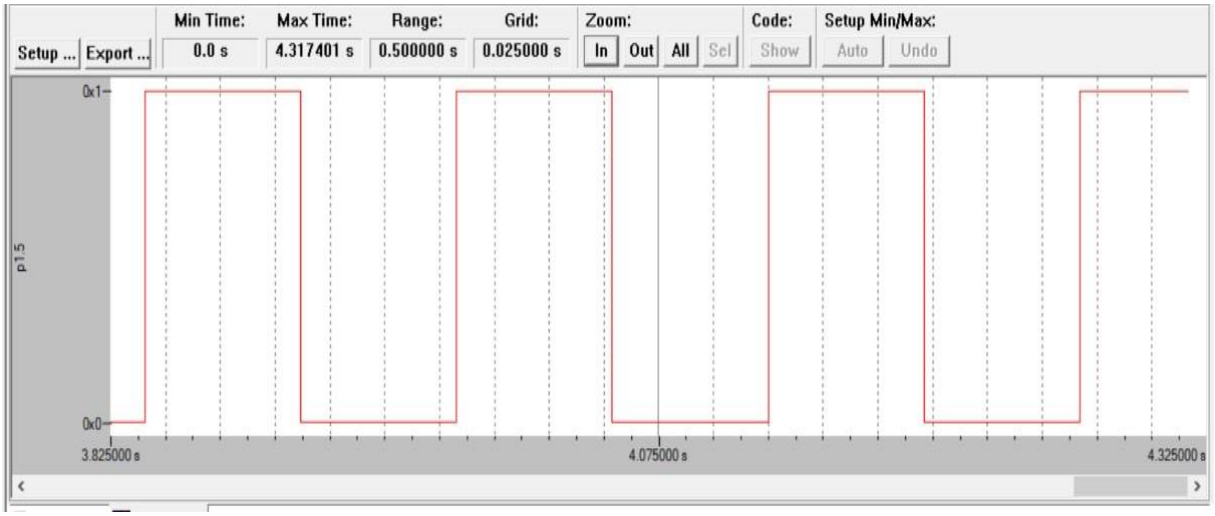
After execution



Mode 1

```
01  org 000h
02      mov tmod,#01h
03  repeat: mov tl0,#00h
04      mov th0,#00h
05      setb tr0
06  wait:   jnb tf0,wait
07      clr tr0
08      clr tf0
09      cpl p1.5
10      sjmp repeat
11      end
```

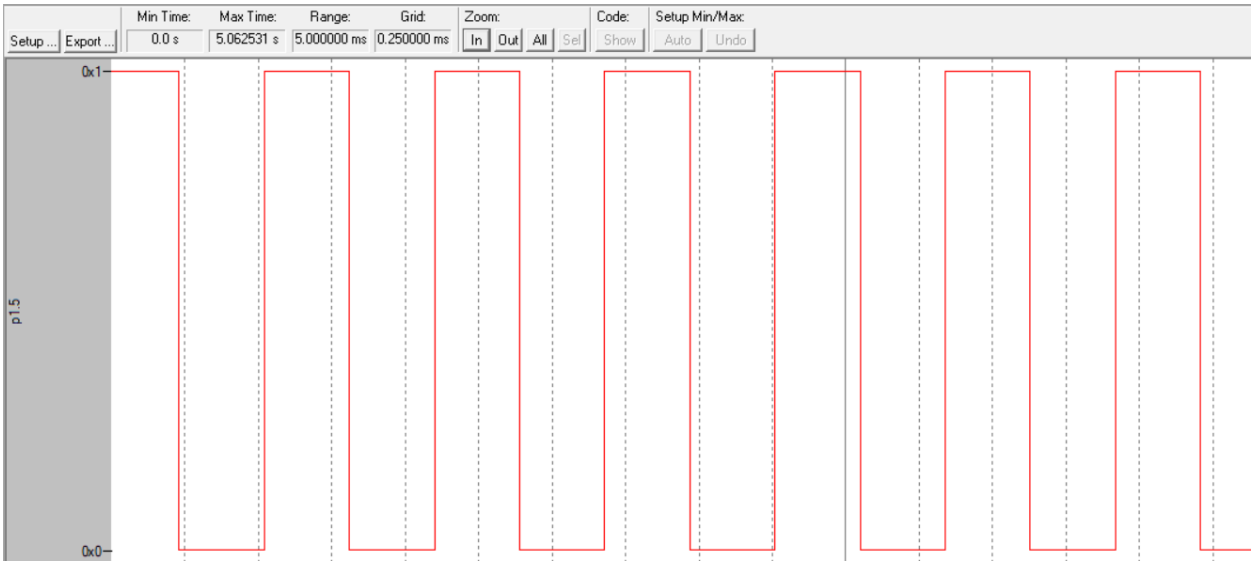
After execution



Mode 2

```
01   ORG 00H
02   MOV TMOD,#20H
03   HERE:MOV TL0,#0A4H
04   MOV TH0,#0A4H
05   CPL P1.5
06   ACALL DELAY
07   SJMP HERE
08   DELAY:SETB TR1
09   WAIT:JNB TF1,WAIT
10   CLR TF1
11   CLR TR1
12   RET
13   END
```

After execution

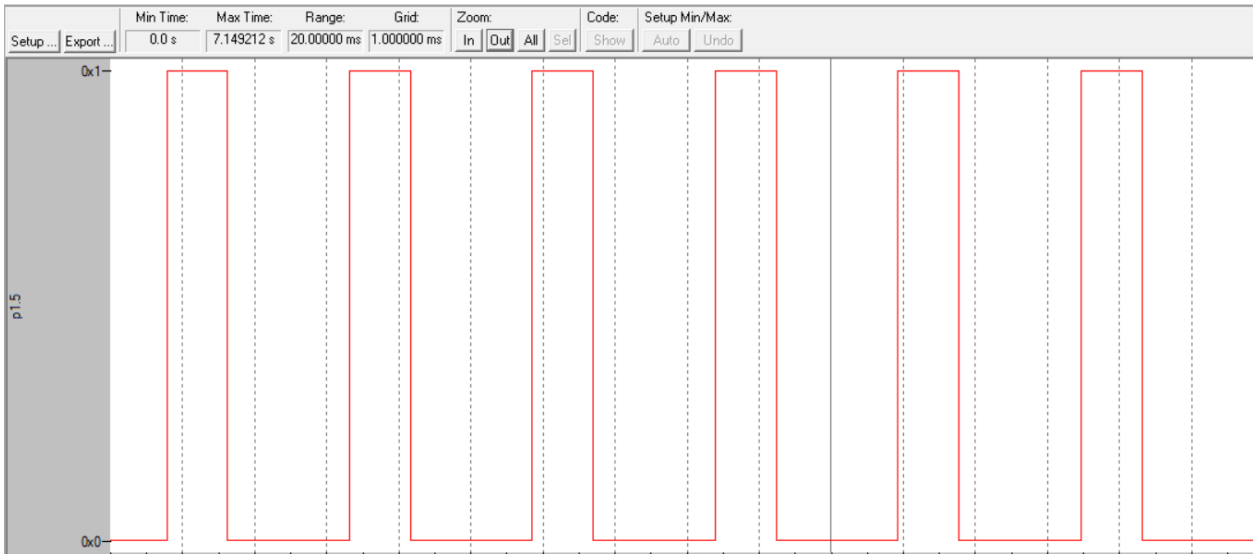


2. WAP to generate a rectangular wave with T(on)=20ms and T(off)=40ms. Use Timer0 in mode0, mode1 and mode2 to generate delay.

Mode 0

```
org 000h
          mov TMOD,#20h        ;timer1 in mode2
          mov TH1,#00h             ;load count value
          mov TL1,#00h
repeat: mov r0,#3d            ;count to get 20ms
         setb p1.5                        ;toggle p2
loop:    setb TR1                       ;start timer
wait:     jnb TF1,wait                ;wait for timer overflow
             clr TR1              ;stop timer
             clr TF1
             djnz r0, loop                          ;clear timer flag
repeat1: mov r1,#6d            ;count to get 40ms
         clr p1.5                       ;toggle p2
loop1:    setb TR1                     ;start timer
wait1:     jnb TF1,wait1               ;wait for timer overflow
             clr TR1              ;stop timer
             clr TF1                          ;clear timer flag
             djnz r1, loop1               ;repeat loop for 2times
             sjmp repeat
             end
```

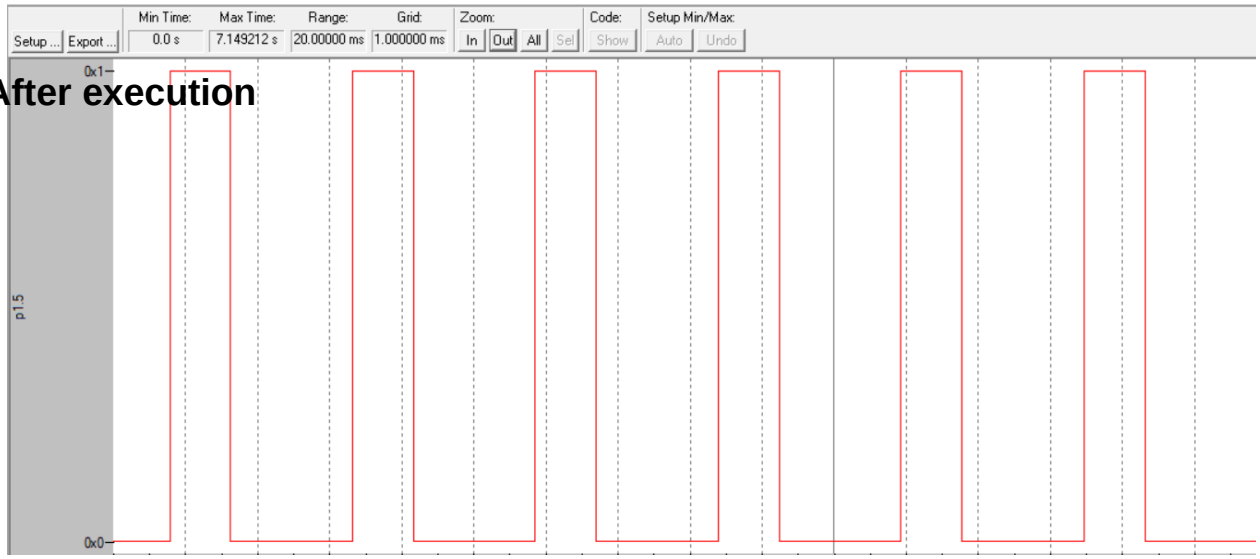
After Execution:



Mode 1

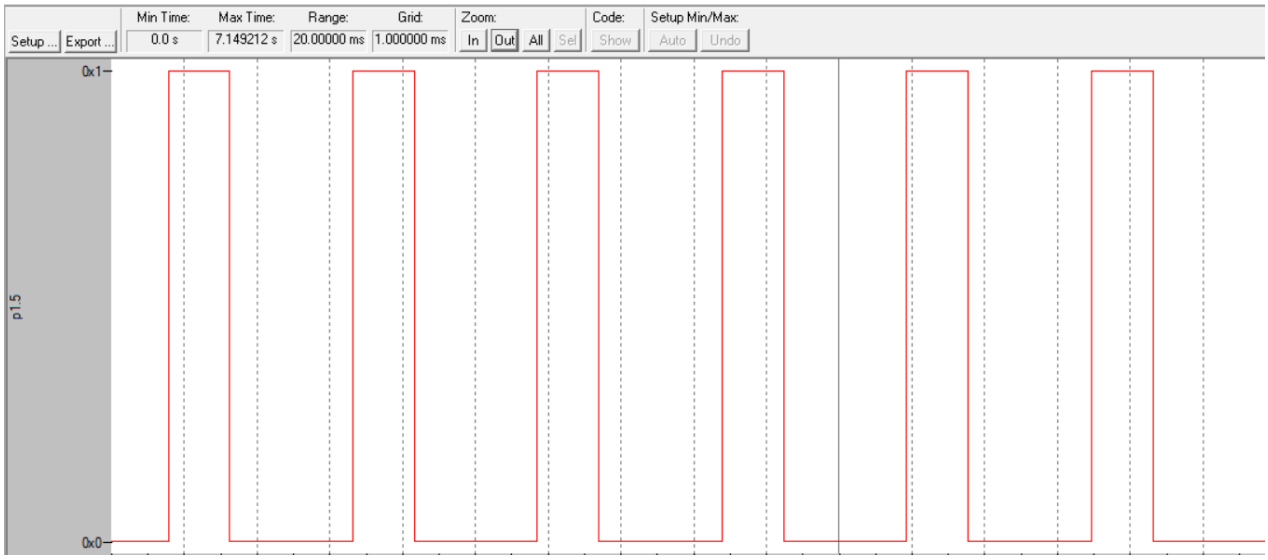
```
01  ORG 000H
02  MOV TMOD,#01H
03  rpt:MOV TL0,#0FEH
04  MOV TH0,#0B7H
05  SETB P1.5
06  ACALL DELAY
07  MOV TL0,#0FEH
08  MOV TH0,#6FH
09  CLR P1.5
10  ACALL DELAY
11  SJMP RPT
12  DELAY:SETB TR0
13  WAIT:JNB TF0,WAIT
14  CLR TF0
15  CLR TR0
16  RET
17  END
```

After execution



Mode 2

```
org 000h
          mov TMOD,#20h          ;timer1 in mode2
          mov TH1,#00h               ;load count value
          mov TL1,#00h
repeat: mov r0,#74d              ;count to get 20ms
         setb p1.5                          ;toggle p2
loop:    setb TR1                         ;start timer
wait:     jnb TF1,wait                   ;wait for timer overflow
             clr TR1                 ;stop timer
             clr TF1
             djnz r0, loop                              ;clear timer flag
repeat1: mov r1,#148d            ;count to get 40ms
         clr p1.5                          ;toggle p2
loop1:    setb TR1                        ;start timer
wait1:     jnb TF1,wait1                  ;wait for timer overflow
             clr TR1                 ;stop timer
             clr TF1                            ;clear timer flag
             djnz r1, loop1                  ;repeat loop for 2times
             sjmp repeat
             end
```

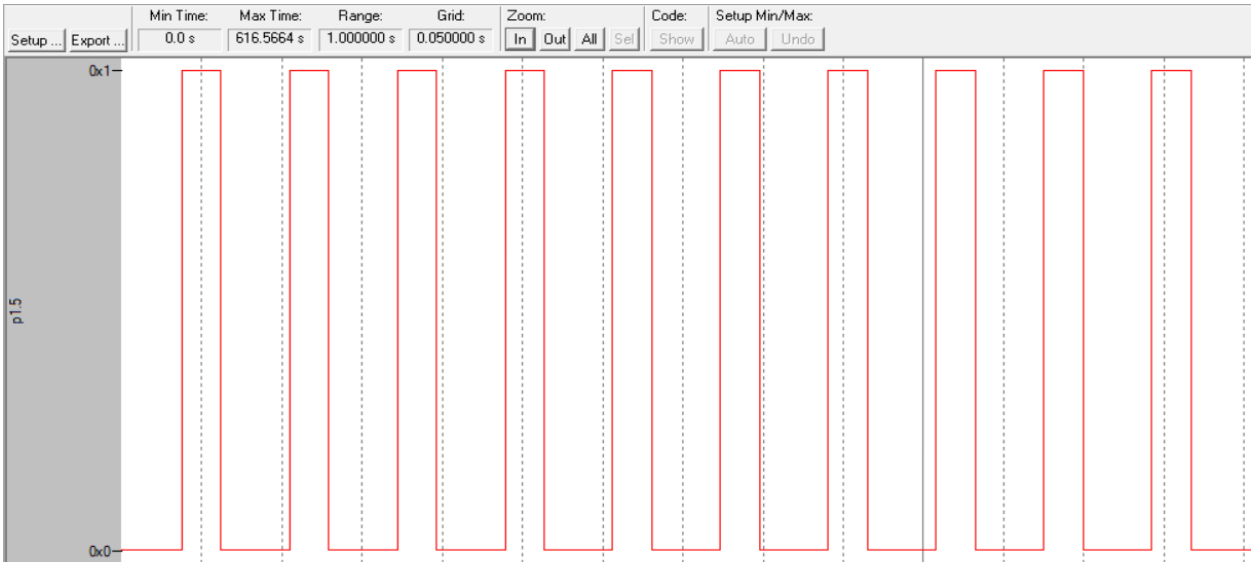


1. Repeat above problems to generate square wave with interrupts for timers

Mode 0

```
01  org 000h
02      ljmp main
03  org 000bh
04      ljmp timer
05
06  org 100h
07  main:   mov tmod,#00h
08      mov ie,#82h ;Enable timer0 interrupt
09      mov th0,#50h
10      mov tl0,#00h
11      setb tr0
12  wait:   sjmp wait
13
14  org 200h
15  timer:  clr tr0
16      jb p1.5,next1
17      cjne a,#06h,next2
18      setb p1.5
19      clr a
20      sjmp done
21  next2:  inc a
22      sjmp done
23  next1:  cjne a,#03h,next3
24      clr p1.5
25      clr a
26      sjmp done
27  next3:  inc a
28  done:   mov th0,#50h
29      mov tl0,#00h
```

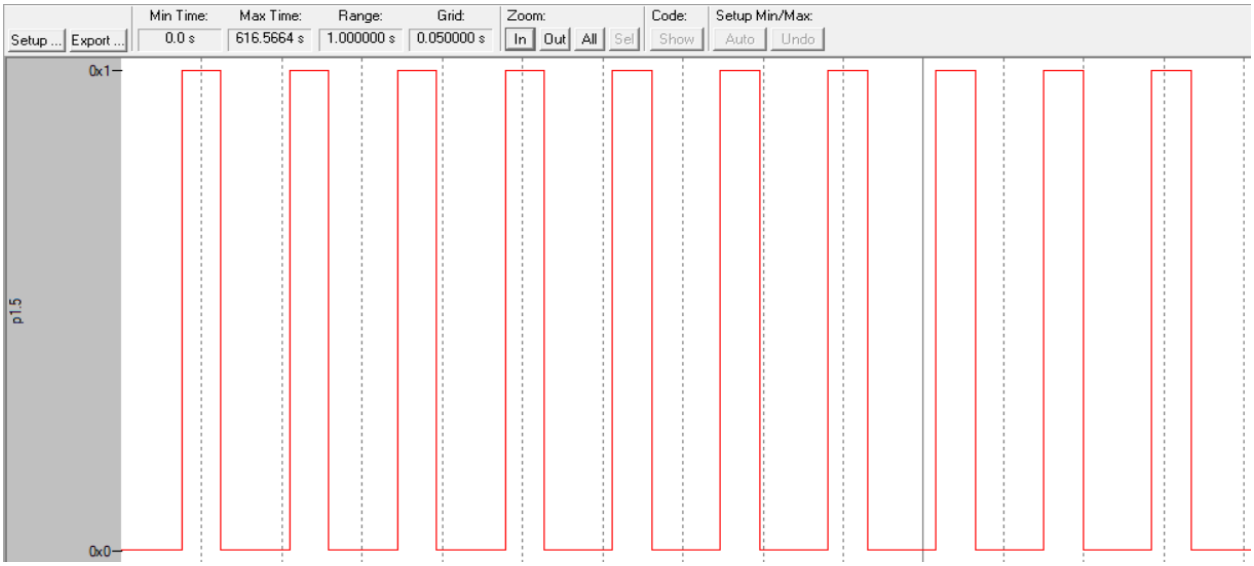
After execution



Mode 1

```
01  org 000h
02      ljmp main
03  org 000bh
04      ljmp timer
05
06  org 100h
07  main:   mov tmod,#01h
08      mov ie,#82h ;Enable timer0 interrupt
09      mov th0,#50h
10      mov tl0,#00h
11      setb tr0
12  wait:   sjmp wait
13
14  org 200h
15  timer:  clr tr0
16      jb p1.5,next1
17      cjne a,#06h,next2
18      setb p1.5
19      clr a
20      sjmp done
21  next2:  inc a
22      sjmp done
23  next1:  cjne a,#03h,next3
24      clr p1.5
25      clr a
26      sjmp done
27  next3:  inc a
28  done:   mov th0,#50h
29      mov tl0,#00h
```

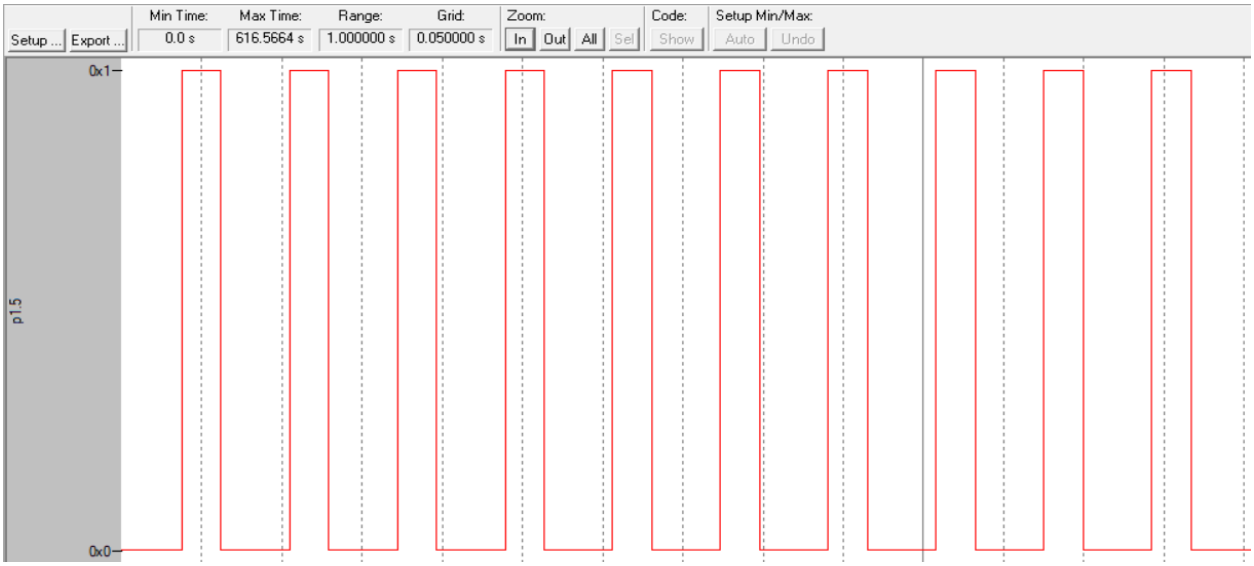
After execution



Mode 2

```
01  org 000h
02      ljmp main
03  org 000bh
04      ljmp timer
05
06  org 100h
07  main:   mov tmod,#02h
08      mov ie,#82h ;Enable timer0 interrupt
09      mov th0,#00h
10      mov tl0,#00h
11      setb tr0
12  wait:   sjmp wait
13
14  org 200h
15  timer:  jb p1.5,next1
16      cjne a,#90h,next2
17      setb p1.5
18      clr a
19      sjmp done
20  next2:  inc a
21      sjmp done
22  next1:  cjne a,#48h,next3
23      clr p1.5
24      clr a
25      sjmp done
26  next3:  inc a
27  done:   reti
28
29      end
```

After execution

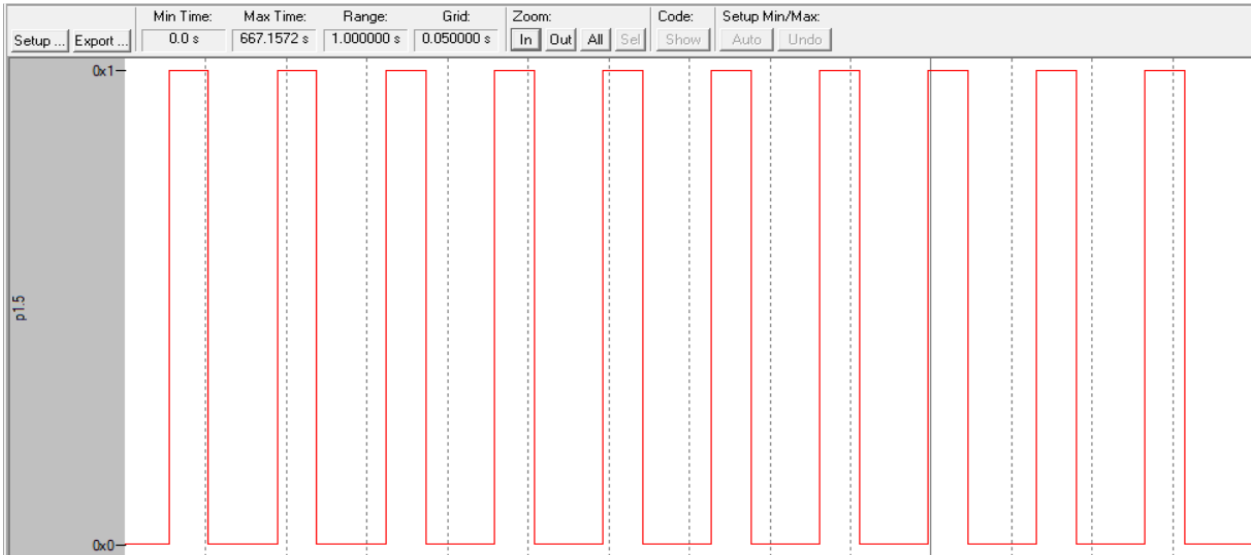


2. Repeat above problems of status check and interrupt methods to generate square wave using embedded c programs.

Mode 0 using interrupt

```
01 #include<reg51.h>
02 sbit mybit = P1^5;
03 int a=0;
04
05 void timer(void) interrupt 1
06 {    TR0 = 0;
07      if(!mybit){ if(a==6){mybit=1;a=0;}
08                  else{a++;}   }
09      else{    if(a==3){mybit=0;a=0;}
10          else{a++;}   }
11      TH0=0x50;
12      TL0=0x00;
13      TR0=1;
14 }
15
16 void main(void)
17 {    TMOD = 0x00;
18      IE = 0x82;
19      TH0 = 0x50;
20      TL0 = 0x00;
21      TR0=1;
22      while(1);
23 }
```

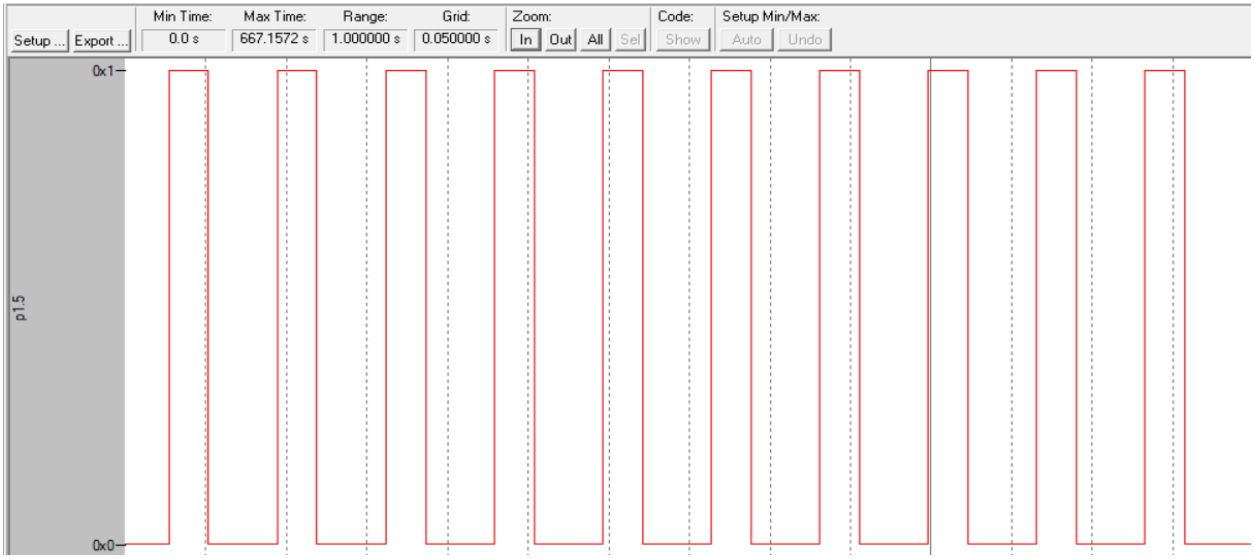
After execution



Mode 1 using interrupt

```
01 #include<reg51.h>
02 sbit mybit = P1^5;
03 int a=0;
04
05 void timer(void) interrupt 1
06 {    TR0 = 0;
07      if(mybit)
08      {    mybit=0;
09           TH0=0x6F;
10           TL0=0xFE;}
11      else{    mybit=1;
12           TH0=0xB7;
13           TL0=0xFF;}
14      TR0=1;
15 }
16
17 void main(void)
18 {    TMOD = 0x01;
19      IE = 0x82;
20      TH0 = 0xB7;
21      TL0 = 0xFF;
22      TR0=1;
23      while(1);
24 }
```

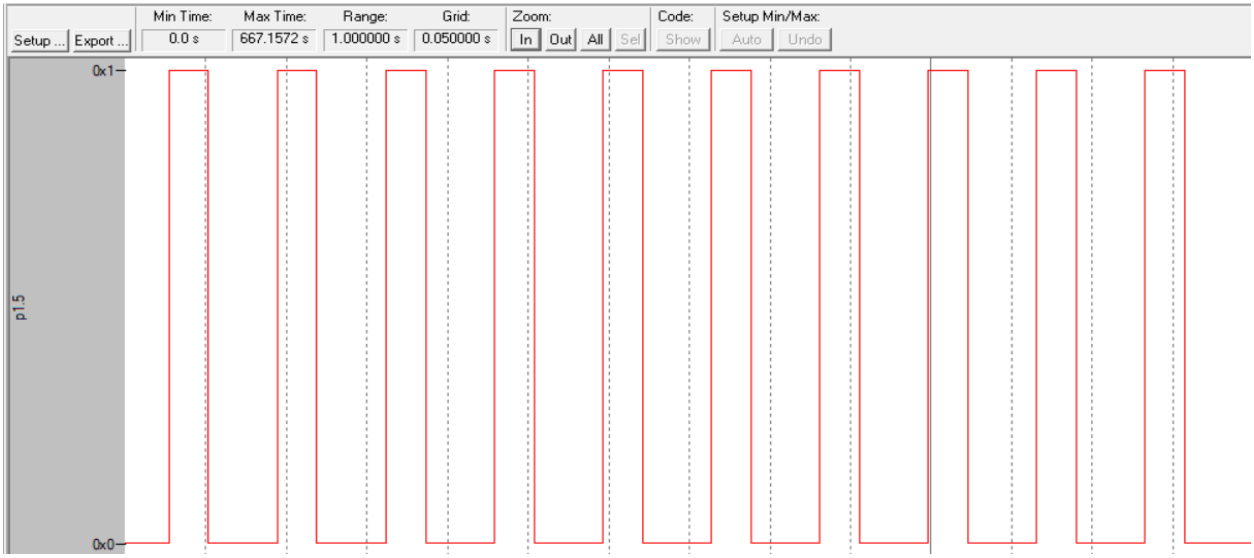
After execution



Mode 2 using interrupt

```
#include<reg51.h>
sbit mybit = P1^5;
int a=0;

void timer(void) interrupt 1
{    TR0 = 0;
     if(!mybit){ if(a==0x90){mybit=1;a=0;}
               else{a++;}   }
     else{    if(a==0x48){mybit=0;a=0;}
         else{a++;}   }
     TR0=1;
}

void main(void)
{    TMOD = 0x02;
     IE = 0x82;
     TH0 = 0x00;
     TL0 = 0x00;
     TR0=1;
     while(1);
}
```

After execution



Mode 0 using status check

```
#include<reg51.h>
sbit mybit = P1^5;

void timer(char);

 main(void)
{    TMOD = 0x00;
     while(1)
     {    mybit=1;
          timer(1);
          mybit=0;
          timer(2);
     }
}

void timer(char n)
{    unsigned char i;
     for(i=0; i<3*n; i++)
     {    TL0 = 0x00;
          TH0 = 0x50;
          TR0 = 1;
          while(!TF0);
          TR0 = 0;
          TF0 = 0;
     }
}
```

After execution

Mode 1 using status check

```
01 #include<reg51.h>
02 sbit mybit = P1^5;
03
04 void timer(char);
05
06 main(void)
07 {    TMOD = 0x01;
08      while(1)
09      {    mybit=1;
10           timer(1);
11           mybit=0;
12           timer(2);
13      }
14 }
15
16 void timer(char n)
17 {    unsigned char i;
18      for(i=0; i<n; i++)
19      {    TL0 = 0x00;
20           TH0 = 0x50;
21           TR0 = 1;
22           while(!TF0);
23           TR0 = 0;
24           TF0 = 0;
25      }
26 }
```

After execution

Mode 2 using status check

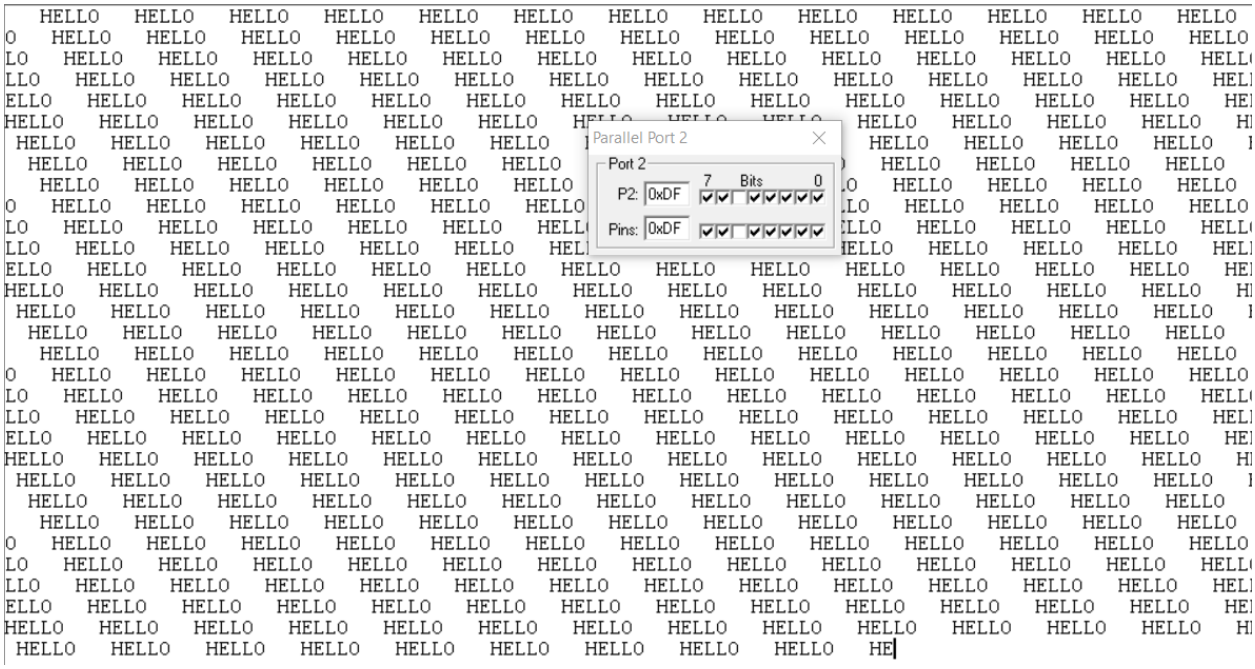
```
01 #include<reg51.h>
02 sbit mybit = P1^5;
03
04 void timer(char);
05
06  main(void)
07 {    TMOD = 0x02;
08      while(1)
09      {    mybit=1;
10           timer(1);
11           mybit=0;
12           timer(2);
13      }
14 }
15
16 void timer(char n)
17 {    unsigned char i;
18      for(i=0; i< 73*n; i++)
19      {    TL0 = 0x03;
20           TR0 = 1;
21           while(!TF0);
22           TR0 = 0;
23           TF0 = 0;
24      }
25 }
```

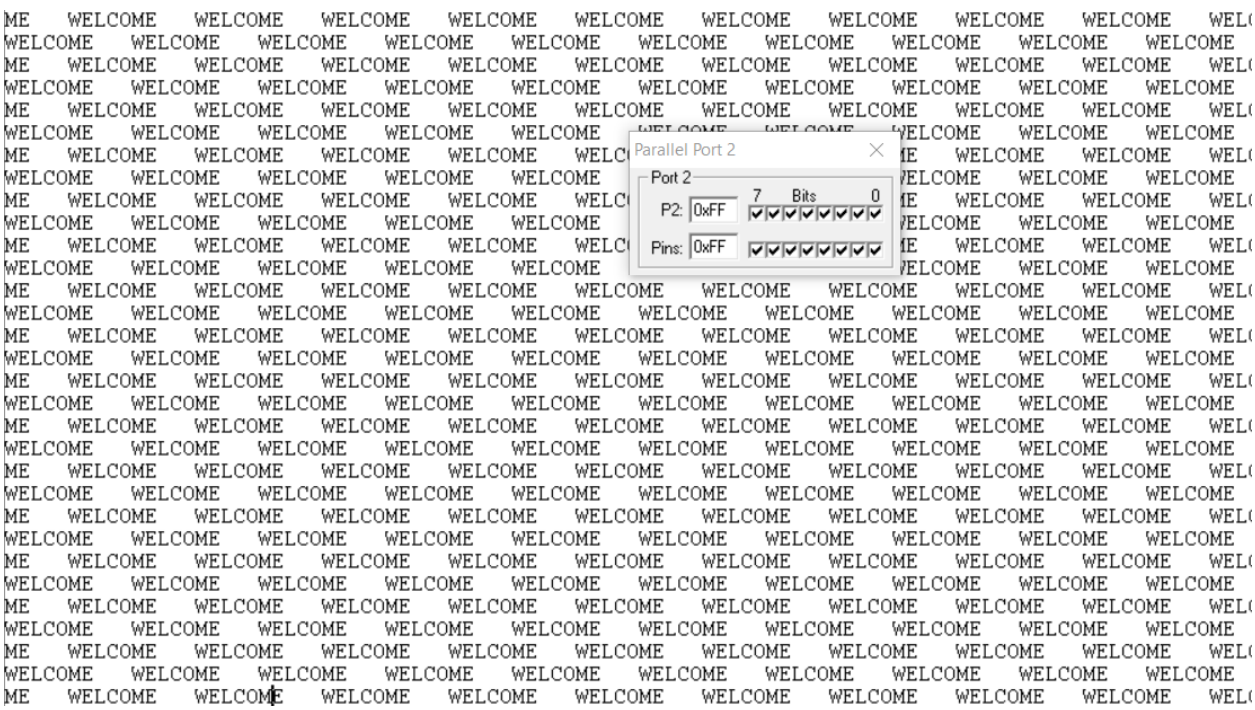
After execution

SET 8 PROGRAMS

1. A switch is connected to P2.5. write an 8051 assembly level program to read the status of switch and if switch is closed send serially "HELL0', else send WELCOME' at baud rate 9600.

```
01          org 000h
02  main:   mov TMOD,#20h
03          mov TH1,#-3
04          mov TL1,#-3
05          mov SCON,#50h
06          setb TR1
07  S1:     jnb p2.5,next
08          mov dptr,#message2
09  FN:     clr A
10          movc a,@a+dptr
11          jz S1
12          acall send
13          inc dptr
14          sjmp FN
15          next: mov dptr, #message1
16  LN:     clr a
17          movc a, @a+dptr
18          jz S1
19          acall send
20          inc dptr
21          sjmp LN
22  send:   mov sbuf, a
23  here:   jnb TI,here
24          clr TI
25          ret
26  message1: db "HELLO    ", 0
27  message2: db "WELCOME    ",0
28  END
```


Parallel Port 2
Port 2
P2: 0xDF
7 Bits 0
Pins: 0xDF


Parallel Port 2
Port 2
P2: 0xFF
7 Bits 0
Pins: 0xFF

2.Write an 8051 assembly level program to transfer a message SJCE serially by programming serial1 communication in interrupt mode with baud rate 9600.

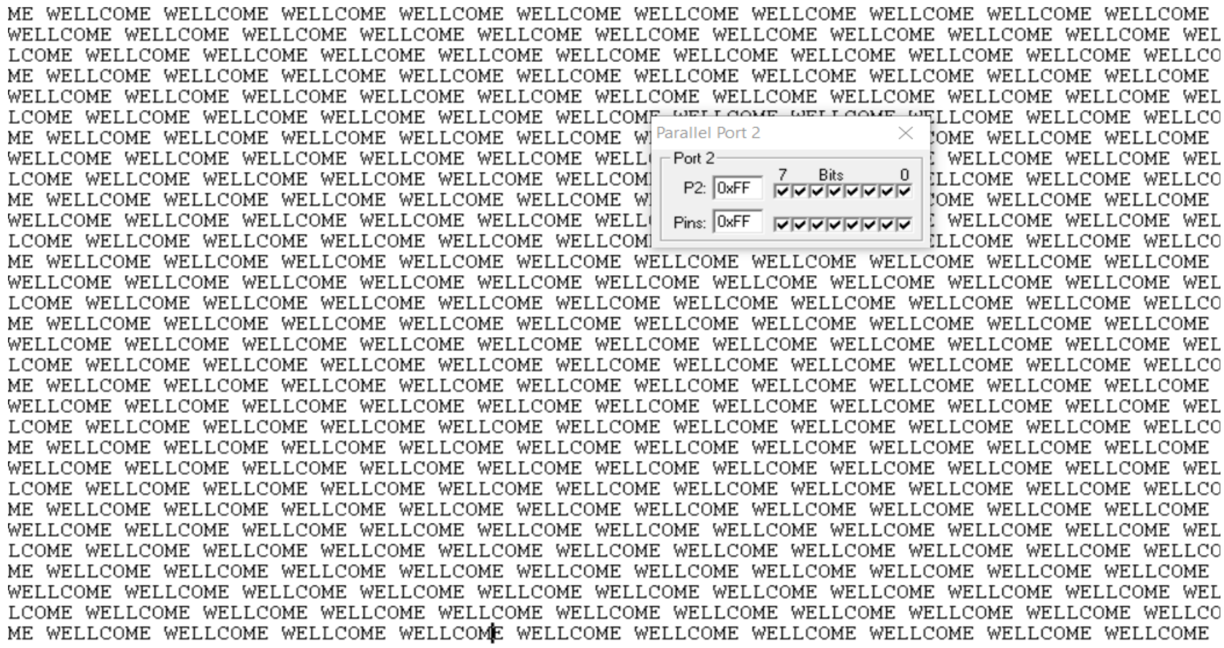
```
01  org 000h
02  main:  mov TMOD,#20h
03  mov TH1,#-3
04  mov TL1,#-3
05  mov SCON,#50h
06  setb TR1
07  mov dptr,#message1
08  FN: clr A
09  movc a,@a+dptr
10  acall send
11  inc dptr
12  sjmp FN
13  s1: sjmp s1
14  send: mov sbuf, a
15  here: jnb TI,here
16  clr TI
17  ret
18  message1: db "SJCE", 0
19  END
```

```
"SJCE
```

Repeat the problem 1 & 2 using embedded C program.

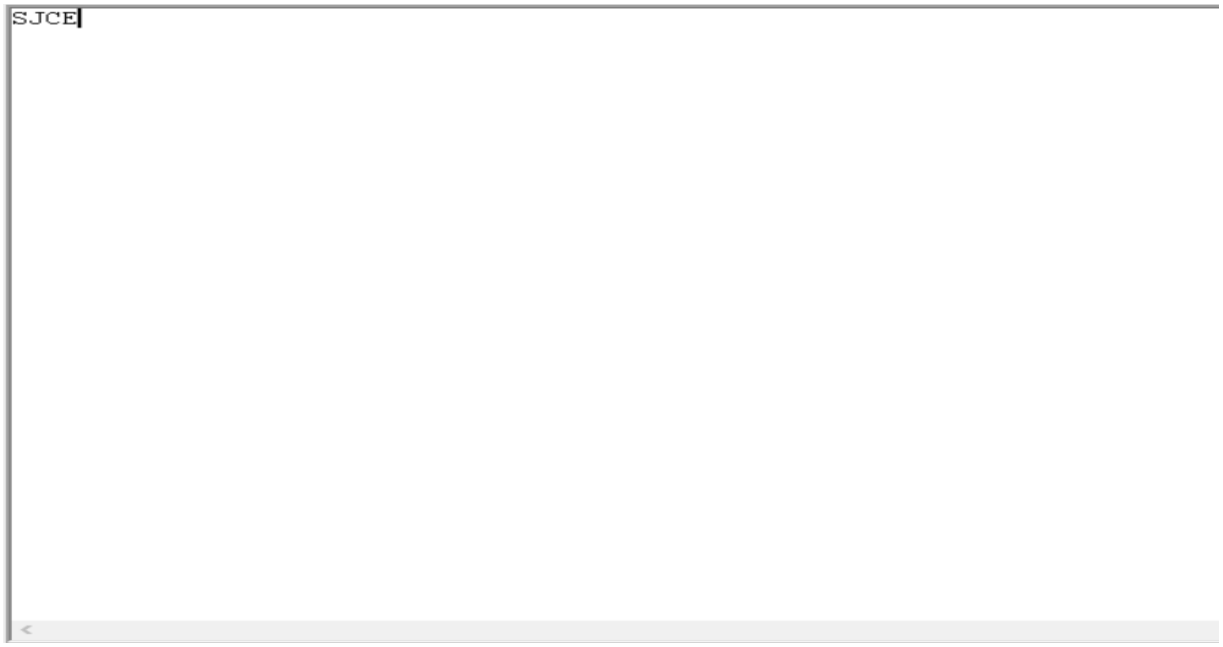
Problem 1

```
01 #include <reg51.h>
02 sbit MYBIT=P2^5;
03 void main (void)
04 {
05 unsigned char z;
06 unsigned char Mess2[]="HELLO ";
07
08 unsigned char Mess1[]="WELLCOME ";
09 TMOD=0x20;
10 TH1= -3 ; // 9600 for normal speed
11 SCON=0x50;
12 TR1=1;
13 while(1)
14 if (MYBIT==1)
15 {
16 for (z=0;z<9;z++)
17 {
18 SBUF=Mess1[z];
19 while (TI==0);
20 TI=0;
21 }
22 }
23 else
24 {
25 for (z=0;z<6;z++)
26 {
27 SBUF=Mess2[z] ;
28 while (TI==0) ;
29
```



Problem 2

```c
#include <reg51.h>
void main (void)
{
unsigned char z;
unsigned char Mess1[]="SJCE";
TMOD=0x20;
TH1= -3 ; // 9600 baud rate
SCON=0x50;
TR1=1;
{
for (z=0;z<4;z++)
{
SBUF=Mess1 [z];
while (TI==0);
TI=0;
}
}
}
```

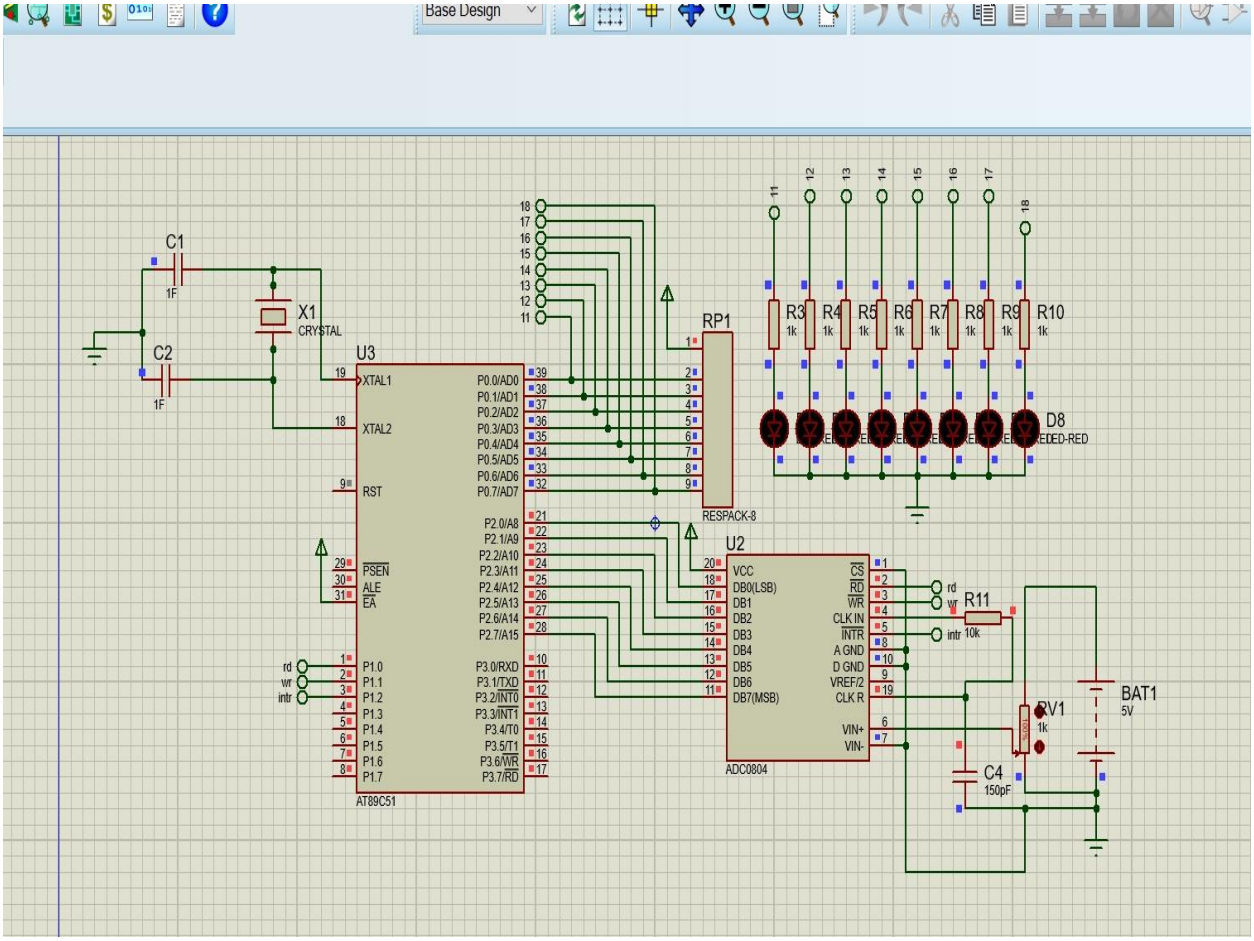

SJCE

HARDWARE EXPERIMENT

Question Number - 1

Interface a Multichannel ADC to 8051 and develop a program to read the analog data, Convert into digital value and display the digitized value on port 0 connected to leds

Circuit Diagram :



Program

```
#include <REGX52.H>

#define input P2                // Input port to read the values of ADC
#define output P0               // Output port, connected to LED's.

sbit wr= P1^1;                  // Write pin. It is used to start the conversion.
sbit rd= P1^0;                  // Read pin. It is used to extract the data from internal register to the output pins of ADC.
sbit intr= P1^2;                // Interrupt pin. This is used to indicate the end of conversion. It goes low when conversion is

void delay(unsigned int msec )                  // The delay function provides delay in msec.
{
    int i,j ;
    for(i=0;i<msec;i++)
  for(j=0;j<1275; j++);
}

void adc()                      // Function to read the values from ADC and display on the LED's.
{
    rd=1;
    wr=0;
    delay(1);
    wr=1;
    while(intr==1);
```

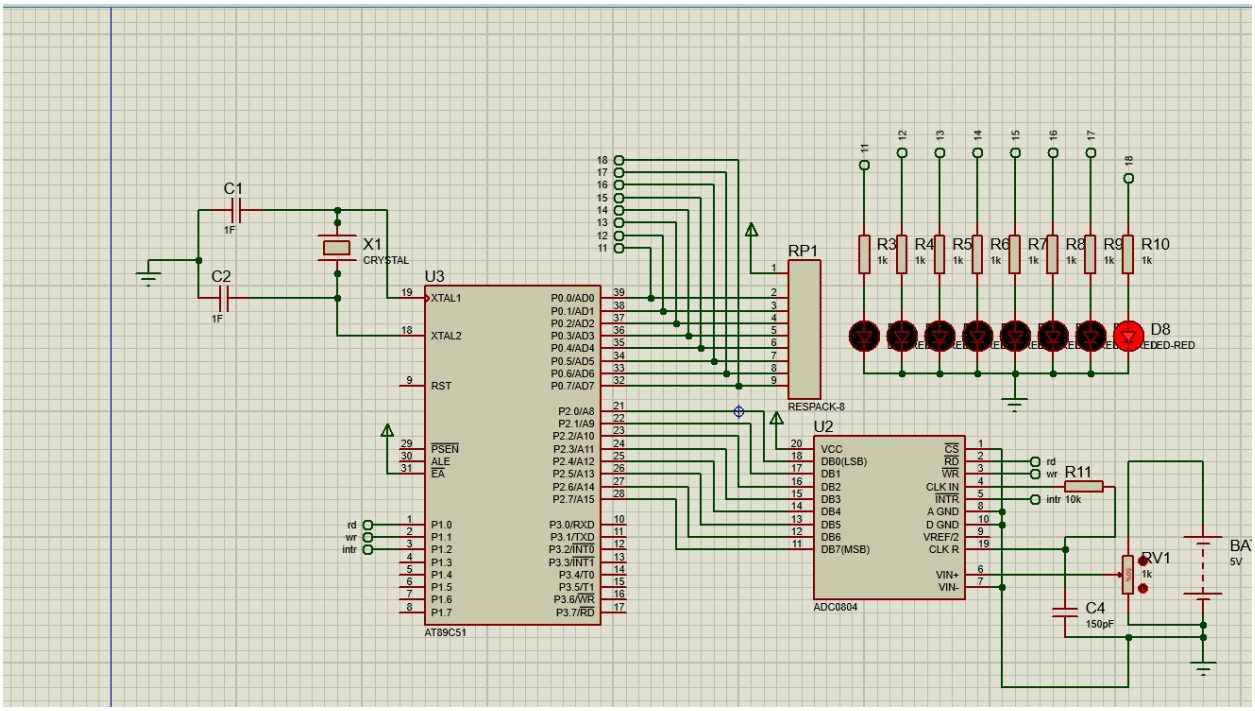
adcrt

g Flash Peripherals Tools SVCS Window Help

t 1

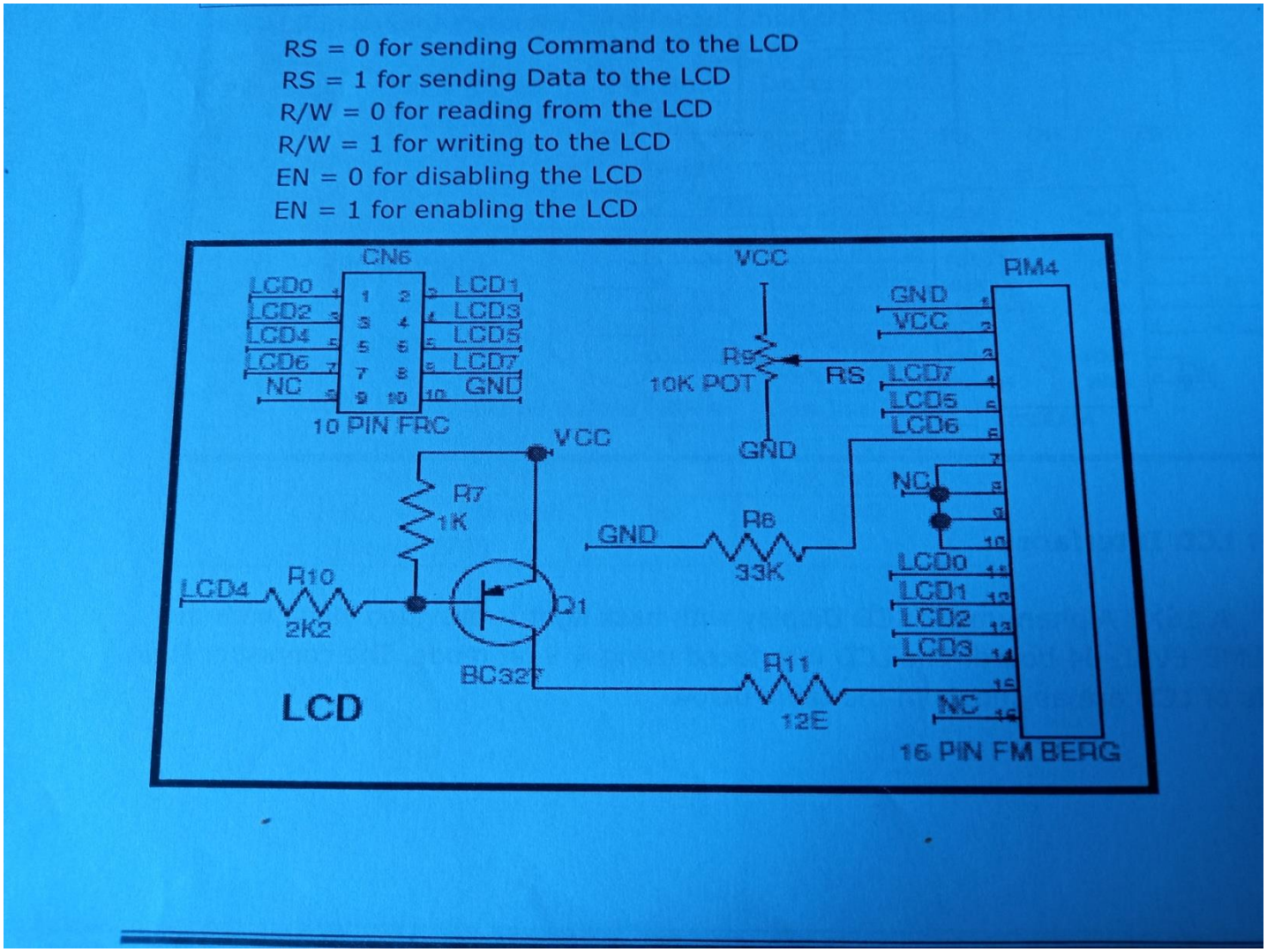
```
15  }
16
17  void adc()                      // Function to read the values from ADC and display on the LED's.
18  {
19      rd=1;
20      wr=0;
21      delay(1);
22      wr=1;
23      while(intr==1);
24      rd=0;
25      output=input;
26      delay(1);
27      intr=1;
28  }
29
30  void main()
31  {
32    input=0xff;                   // Declare port 0 as input port.
33      while(1)
34      {
35     adc();
36    }
37  }
```

Result :



For the analog voltage of 2.5 volts output is 10000000

LCD interface



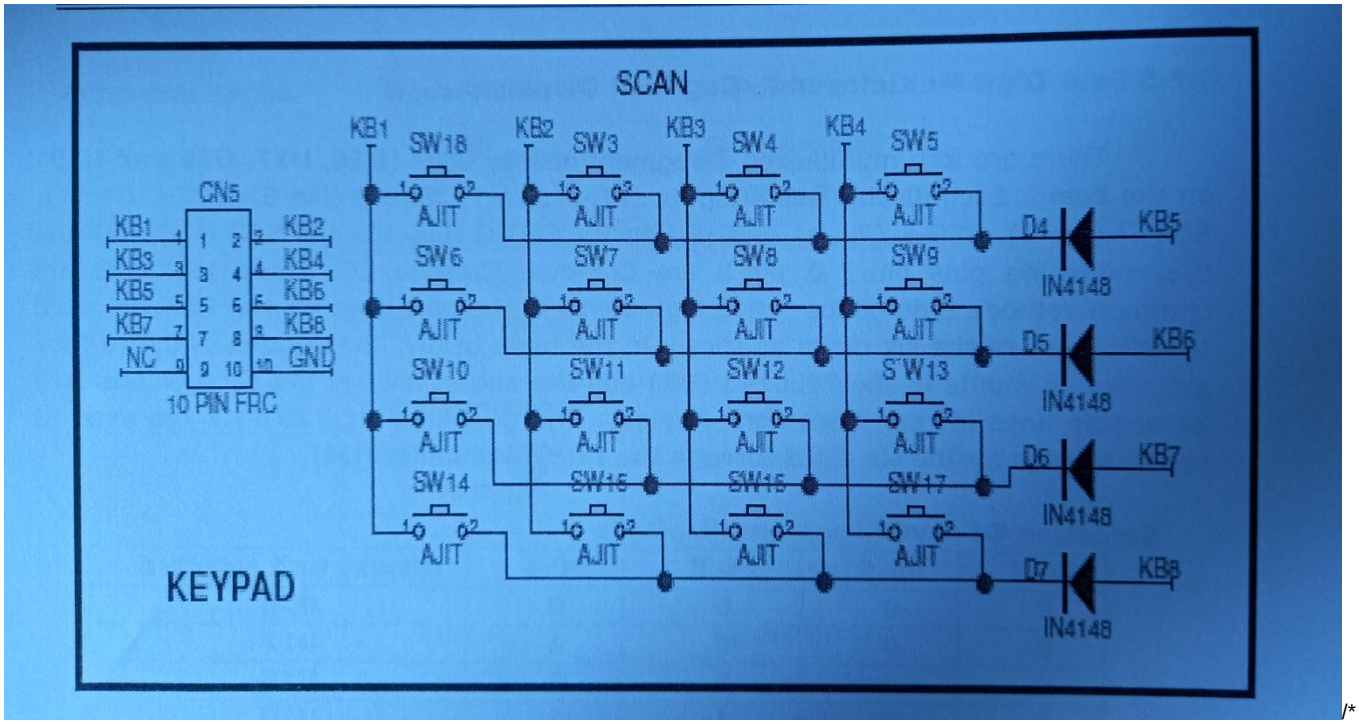
LCD routine:

```
/* LCD related functions. */
#include "at89c51ed2.h"
#include <intrins.h>            //For _nop_();

// LCD FUNCTION PROTOTYPE

extern void lcd_comm(void);
extern void wr_cn(void);
extern void wr_dn(void);
extern void lcd_data_cmnd(char, char);
void delay(int);

extern unsigned char temp1;
extern unsigned char temp2;
unsigned char var;
```

```
void lcd_init(void)
{
   temp1 = 0x30;    // D5(P3.3)=1,D4(P3.2)=1
            wr_cn();
            delay(500);

   temp1 = 0x30;    // D5(P3.3)=1,D4(P3.2)=1
            wr_cn();
            delay(500);

            temp1 = 0x30;       // D5(P3.3)=1,D4(P3.2)=1
            wr_cn();
            delay(500);

            temp1 = 0x20;       // D5(P3.3)=1
            wr_cn();
            delay(500);

   temp1 = 0x28;    // Set
            lcd_comm();
            delay(500);

            temp1 = 0x0f;  //display on, cursor on, cursor blinking
            lcd_comm();
            delay(500);

            temp1 = 0x06;  //shift cursor right with auto increment
            lcd_comm ();
            delay (500);

            temp1 = 0x80;  //clear display with cursor on first position
            lcd_comm ();
            delay (500);

            temp1 = 0x01;
            lcd_comm ();
            delay(500);
}

// Function to pass commands to LCD
void lcd_comm(void)
{
            var = temp1;
            temp1 = temp1 & 0xf0;
            temp1 = temp1 >> 4;

            wr_cn ();

            temp1 = var & 0x0f;
            wr_cn ();

            delay (60);
}

// Function to pass data to LCD
Void lcd_data (void)
{
            var = temp2;
            temp2 = temp2 & 0xf0;  // convert the byte into nibble
            temp2 = temp2 >> 4;
            wr_dn();

            temp2 = var & 0x0f;
//          temp2 = temp2 << 2;
            wr_dn();

            delay(60);
}

// Function to write to command reg of LCD
void wr_cn(void)
{
            temp1 = temp1 & 0x7f;       //          RS(P3^7)=0
            temp1 = temp1 & 0xDF;
            temp1 = temp1 | 0x40;       //          EN(P3^6)=1, TXD(P3^1)=1, RXD(P3^0)=1

            P2 = temp1;

            _nop_();
```

```
        _nop_();
        _nop_();
        _nop_();
        _nop_();


        temp1 = temp1 & 0xbf;       //      EN(P3^6)=0,
        P2 = temp1;

}

// Function to write to data reg of LCD
void wr_dn(void)
{
        temp2 = temp2 | 0xc0;       //      RS(P3^7)=1,EN=1,TXD=1,RXD=1
        temp2 = temp2 & 0xDF;
        P2 = temp2;

        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();

        temp2 = temp2 & 0xbf;       //      EN = 0
        P2 = temp2;
}

// Function to clear the LCD display
/*void clear_lcd()
{
         temp1 = 0x01;
         lcd_comm();
         delay(500);
}
          */
void delay(int count)
{
        int i;

        for(i=0;i<count;i++);
}
 /*
void lcd_data_cmnd(char cmnd, char l_data)
{
        char check;

        check =cmnd;
        var = l_data;

        l_data = l_data & 0xf0;
        temp1 = l_data >> 2;
        if(check == 1)
                wr_cn();
        else if(check == 2)
                {
                temp2 = temp1;
                wr_dn();
                }
        l_data = var & 0x0f;
        temp1 = l_data << 2;
        if(check == 1)
                wr_cn();
        else if(check == 2)
                        temp2 = temp1;
                        wr_dn();

        delay(60);
}
```

KEYBOARD INTERFACE:



/*

Program to demonstrate keyboard operation
Takes a key from key board and displays it on LCD screen*/

```
#include "at89c51xd2.h"

// This project includes the following files:
// 1. kbdisp.c   the source program      to keypad
// 2. LCD_routine.c  to display the key read

void scan(void);
void get_key(void);
void display(void);
void delay_ms(int i);
void uart_init(void);

/******* LCD FUNCTION PROTOTYPE*******/
void lcd_init(void);
void lcd_comm(void);
void lcd_data(void);
void clear_lcd(void);
void delay(int);

unsigned char temp1=0x00;
unsigned char temp2;

idata unsigned char  row,col,key;

unsigned char scan_code[16]={0xEE,  0xDE,     0xBE,      0x7E,
                                                    0xED,     0xDD,      0xBD,      0x7D,
                                                    0xEB,     0xDB,      0xBB,      0x7B,
                                                0xE7,  0xD7,      0xB7,      0x77 };

unsigned char LED_CODE[16]= {0x3f,0x66,0x7f,0x39,
                                                    0x06,0x6d,0x6f,0x5e,
```

```
                                                0x5b,0x7d,0x77,0x79,
                                                0x4f,0x07,0x7c,0x71};

unsigned char LCD_CODE[16] = {'0','4','8','C',
                                                        '1','5','9','D',
                                                        '2','6','A','E',
                                                        '3','7','B','F'         };

idata unsigned char temp,temp4,temp3,res1,flag,result=0x3F;
 /* in this key pad program keypad lines are not kept high .we will make them high
          low throgh writing data to port.*/
void main ()
{
        lcd_init();

        while(1)
        {
                get_key();
                display();
                P3 = 0xFF;
        }
}  //end of main()



void get_key(void)                              // get_key() function calls scan() function
   {                                                                    // on sensing a key from scan() function it
    int i;                      // will compare the received scan code with
    display();                  // scan code lookup table and returns ASCII code
    flag = 0x00;                // rows are read from Port P0 is scan() function
    while(flag == 0x00)         // this function is in an eternal loop
                // wiil return to main() only after getting a key
        {
                for(row=0;row<4;row++)          // This for loop makes the one of the ROW low at
                {                                                               // one time .Then scan function is called
                   if( row == 0)
                        temp3=0xFE;
                   else if(row == 1)
                        temp3=0xFD;
                   else if(row == 2)
                        temp3=0xFB;
                   else if(row == 3)
                        temp3=0xF7;
                P0 = temp3;                                     // each time temp3 value is put into this
                        scan();
                        delay_ms(10);                           // on sensing a key scan() function will make flag = 0xff
                        if(flag == 0xff)
                        break;
                }                                                               // end of for

                if(flag == 0xff)
                break;
        }  // end of while

                P3 = 0x00;                                                      //          Enable U21
                for(i=0;i<16;i++)                       // in this for for loop scan code received  which is in res1
                {                                                               // variable is compared with
                if(scan_code[i] == res1)        // the lookup table for array scan code[] and when a match is
                                                                                // found will return the correspoding led code for
the key pressed
                                        {
                                        temp1 = 0x87;
                                        lcd_comm();
                                        temp2 = LCD_CODE[i];
                                        lcd_data();
                                        result =  LED_CODE[i];
                                        break;
                                        }

                                }
}                                                                               // end of get_key();

void scan(void)                                                 // will return the scan code for the key pressed
   {                                            // Both row lines and column lines  are connected to
                unsigned char t;                        //Port 0 only.Columns are connected to P0.0-P0.3
                temp4 = P0;                                             // P0.4-P0.7 are connected to rows
                temp4 = temp4 & 0xF0;           //read port0 ,mask with 0xF0h

        if(temp4 != 0xF0)               // Means a key is sensed
        {
                delay_ms(30);
```

```
                delay_ms(30);           // give some delay for debouncing

        temp4 = P0;             // read the port again
                temp4 = temp4 & 0xF0;

            if(temp4 != 0xF0)       // debounce
            {
                flag = 0xff;        // set the flag denoting a key is received
                res1 = temp4;
                t = temp3 & 0x0F;
                res1 = res1 | t;                // and OR it with column value
            }                                                                   // to get the scan code of the key pressed
            else
                {
                flag = 0x00;
                }

            }
}

          // end of scan()

      void display(void)
     {
    P1 = result;
     }

     void delay_ms(int i)
    {
   int j;
   for(j=0;j<i;j++);
     }
```

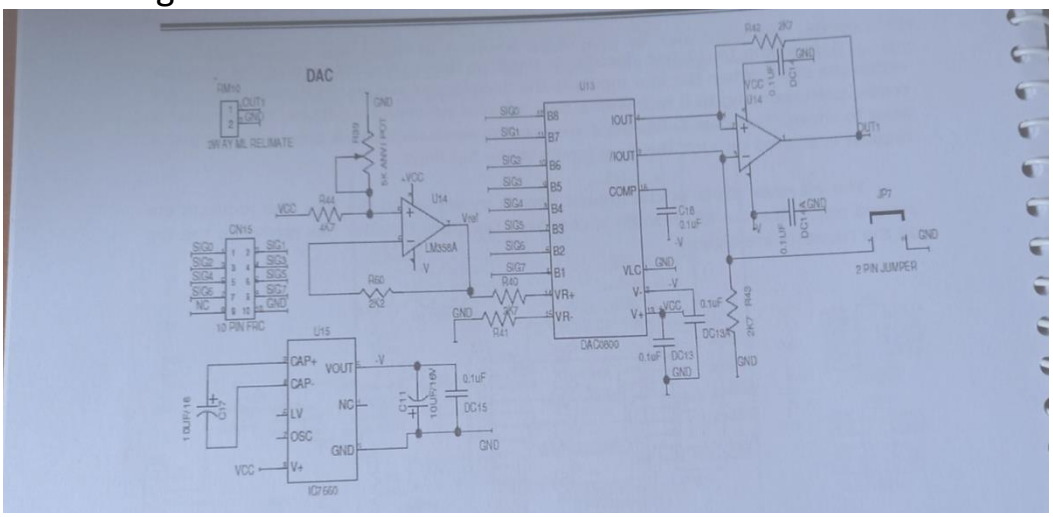
DAC INTERFACE:

1. Interfacing DAC with 8051 and developing an algorithm to generate the following outputs

   I) Square wave with 50% duty cycle
   II) Square wave with 75% duty cycle
   III) Triangle wave.
   IV) Ramp(+ve &-ve)
   V) Sine wave
   VI) Staircase
   VII) Staircase triangle

Solution:

Interfacing Circuit:

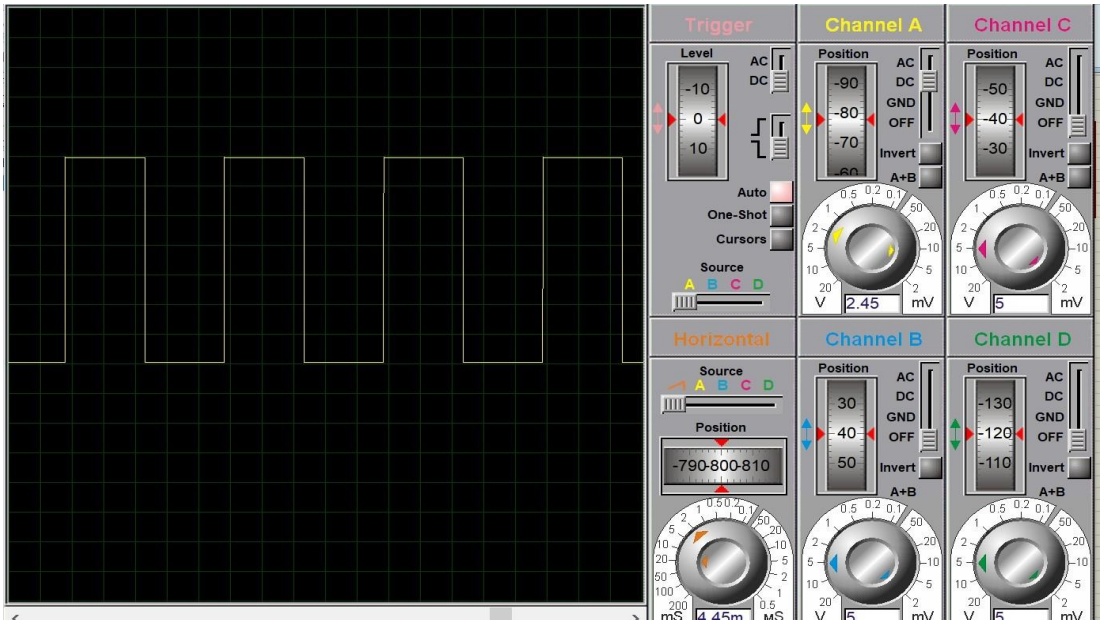


I) Generation of square wave with 50%duty cycle.

- Solution:

```
01  //GENERATION OF SQUARE WAVE WITH 50% DUTY CYCLE AND DEALY OF 1ms//
02          ORG 0000h
03          MOV TMOD,#10H
04  repeat: MOV A,#00H
05          MOV P2,A
06          ACALL DELAY
07          MOV A,#0FFH
08          MOV P2,A
09          ACALL DELAY
10          SJMP REPEAT
11  delay:  mov TH1,#0FEH
12          MOV TL1,#33H
13          SETB TR1
14  WAIT:   JNB TF1,WAIT
15          CLR TR1
16          CLR TF1
17          ret
18          END
```

RESULT:

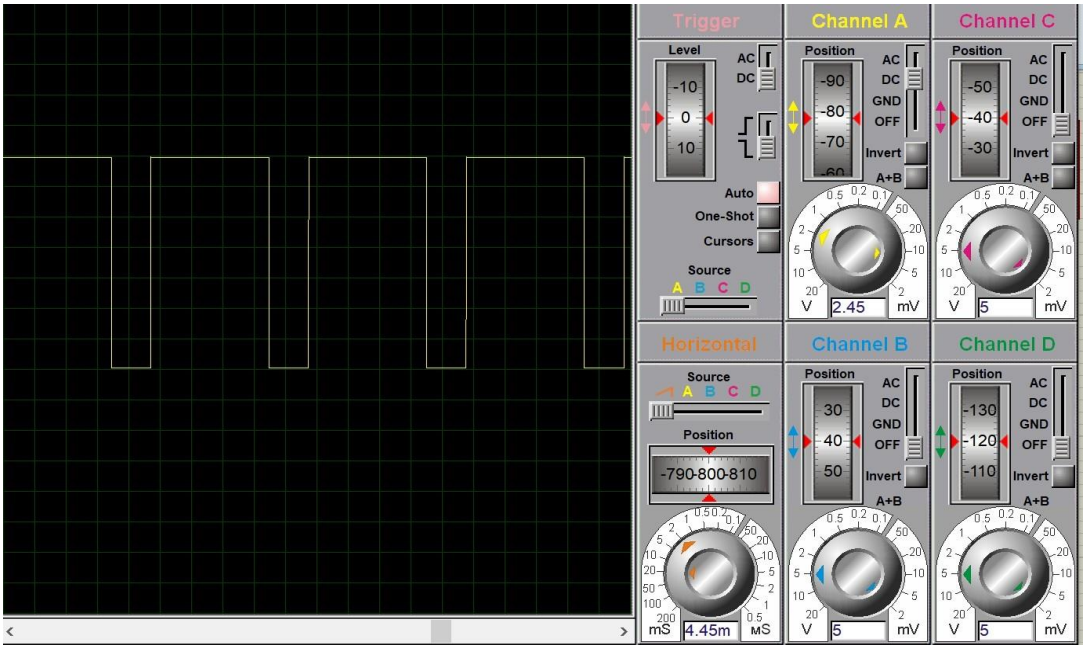


2.Generation of square wave with 75%duty cycle.

- Solution:

```
01  //GENERATION OF SQUARE WAVE WITH 75% DUTY CYCLE //
02            ORG 000h
03            mov P2,#00H
04  repeat:   Acall squarwave
05            sjmp repeat
06  squarwave:mov P2,#0FFH
07            Acall delay
08            mov P2,#00H
09            Acall delay1
10            ret
11  delay:    mov r0,#30
12  up2:      mov r1,#250
13  Here:     djnz r2,Here
14            djnz r0,up2
15            ret
16  delay1:   mov r0,#10
17  up1:      mov r1,#250
18  Here1:    djnz r2,Here1
19            djnz r0,up1
20            ret
21            END
22
```

RESULT:

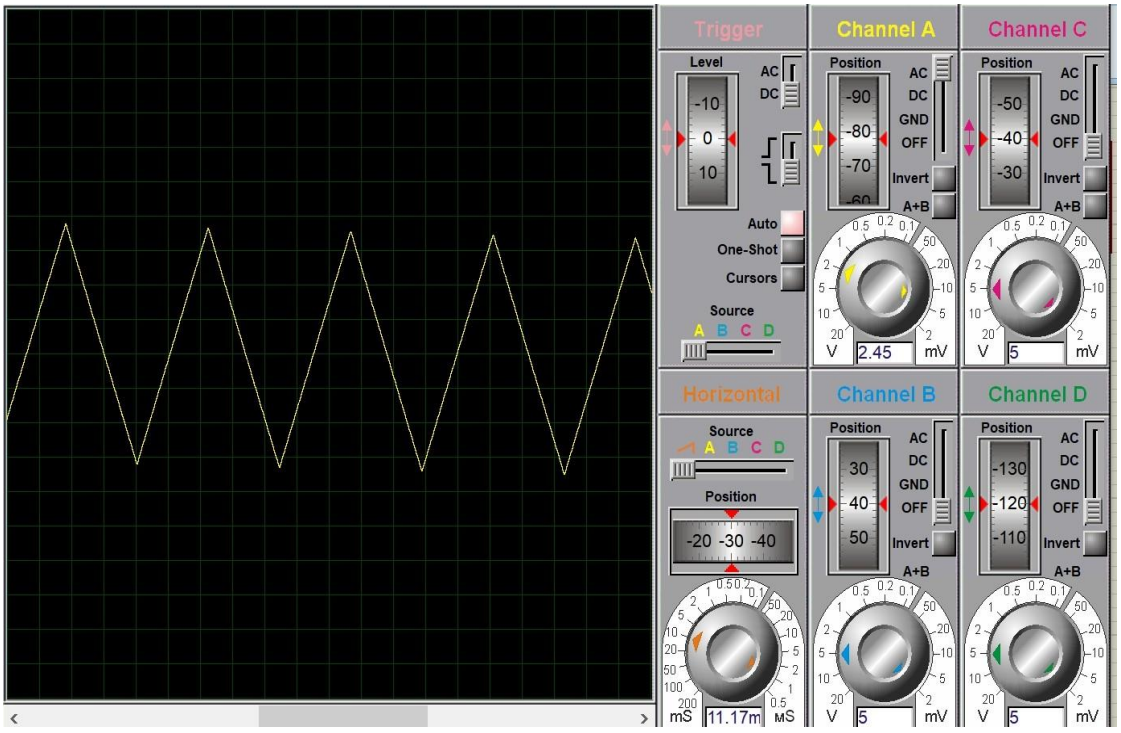


Generation of Triangular wave.

➢ ALGORITHM

```
// GENERATION OF TRAIANGULAR WAVE OF 20mSEC delay //
        ORG 0000h
        MOV TMOD,#10H
repeat: MOV A,#00H
RISE:   MOV P2,A
        ACALL DELAY
        INC A
        CJNE A,80H,RISE
FALL:   DEC A
        MOV P2,A
        ACALL DELAY
        CJNE A,#00H,FALL
        SJMP REPEAT
delay:  mov TH1,#0FFH
        MOV TL1,#0B7H
        SETB TR1
WAIT:   JNB TF1,WAIT
        CLR TR1
        CLR TF1
        ret
        END
```

RESULT:

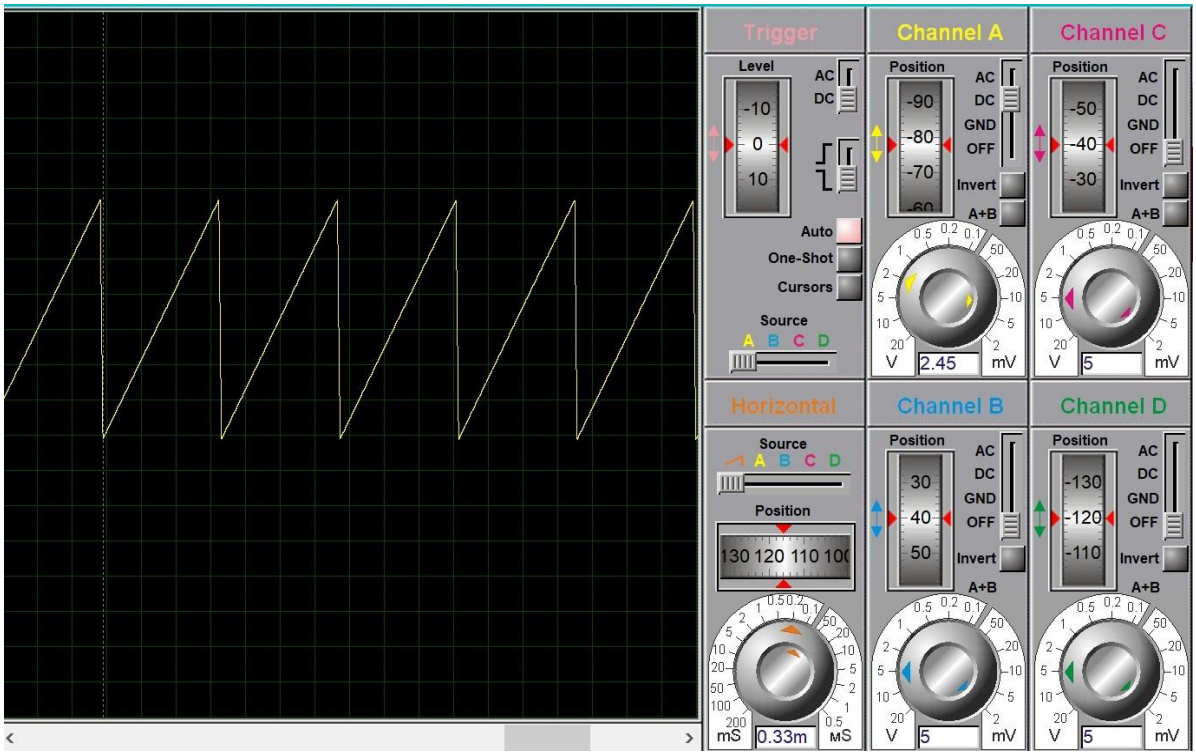


II) A. Generation of positive Ramp with signal.

- ALGORITHM

```
// GENERATION OF POSITIVE RAMP SIGNAL //
        ORG 000H
BACK: MOV A,#00H
 L1:  MOV P2,A
      INC A
      CJNE A,#0FFH,L1
      MOV P2,A
      SJMP BACK
      END
```

RESULT:

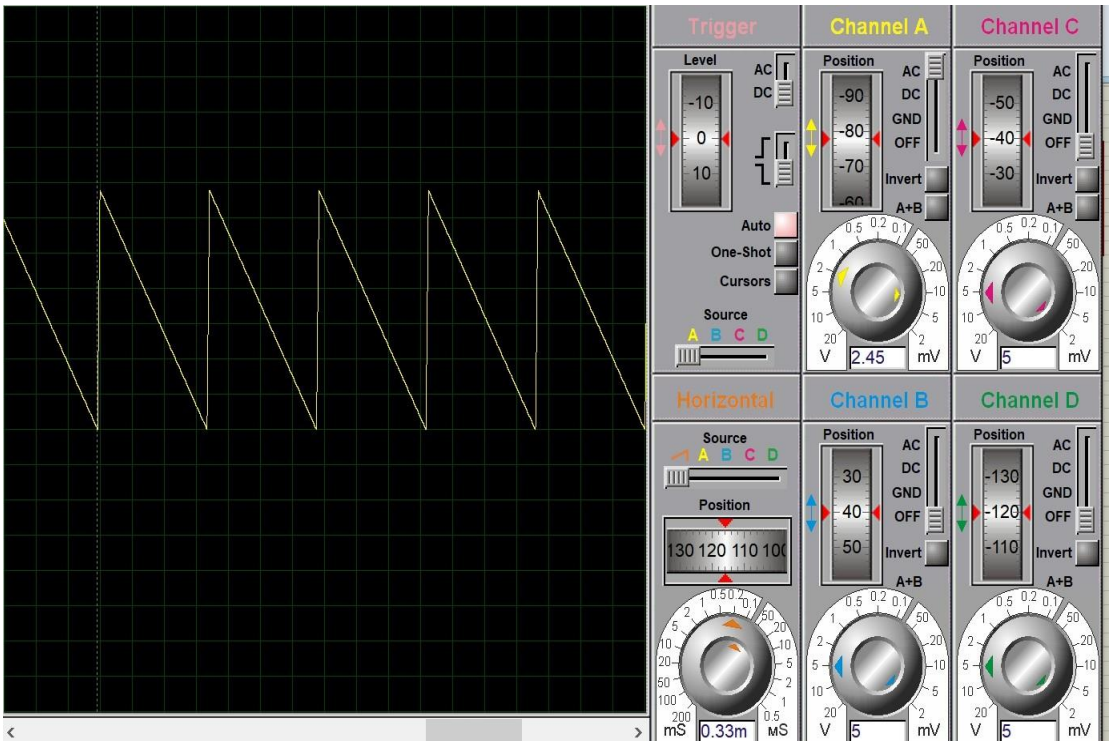


B. Generation of negative Ramp with signal.

- ALGORITHM

```
01 // GENERATION OF NEGATIVE RAMP SIGNAL //
02       ORG 000H
03 BACK: MOV A,#0ffH
04  L1:  MOV P2,A
05       DEC A
06       CJNE A,#00H,L1
07       MOV P2,A
08       SJMP BACK
09       END
10
11
```

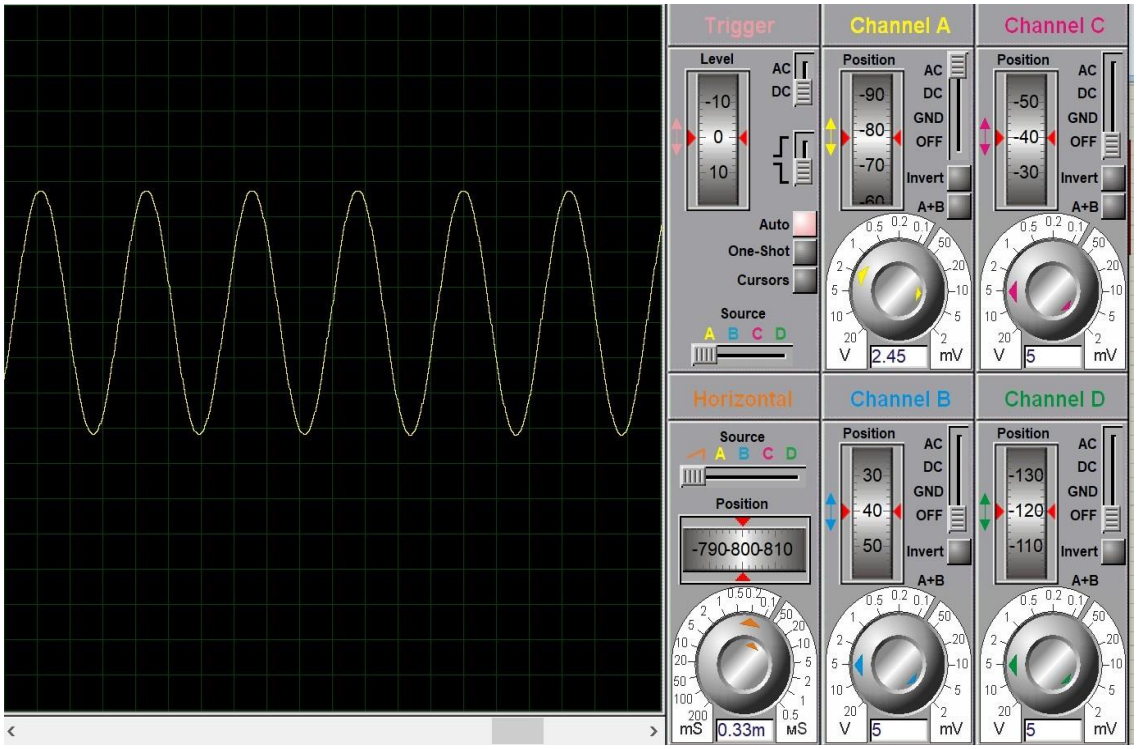
RESULT:



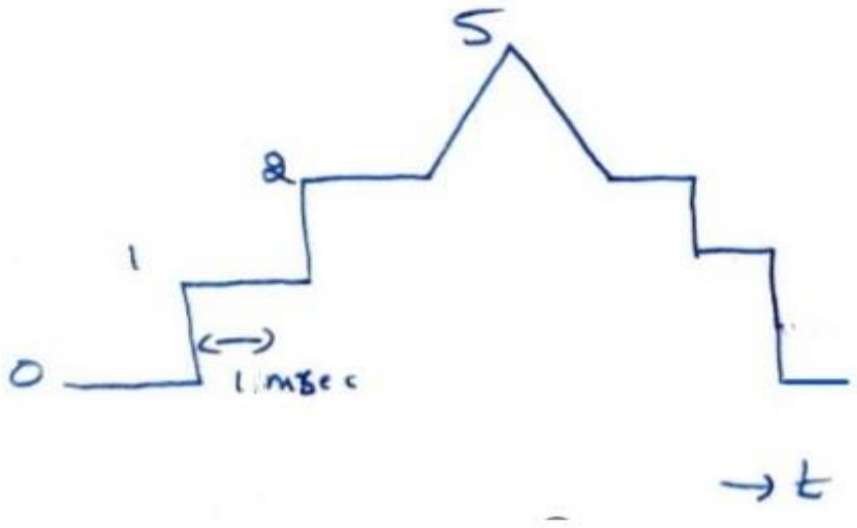
- GENERATION OF SINE WAVE
- ALGORITHM

```
//GENERATION OF SINEWAVE //
        ORG 000h
AGAIN:  mov dptr,#TABLE
        MOV R2,#78H
BACK:   CLR A
        MOVC A,@A+DPTR
        MOV P2,A
        INC DPTR
        DJNZ R2,BACK
        SJMP AGAIN
        ORG 300h
TABLE:  DB 128,135,141,148,155,161,168,174,180,186,192,198,203,209,214,219,223,227,232,
        DB 235,239,242,245,247,250,252,253,254,255,255,255,255,255,254,253,252,250,247,
        DB 245,242,239,235,232,227,223,219,214,209,203,198,192,186,180,174,168,161,155
        DB 148,141,135,128,121,115,108,101,95,88,82,76,70,64,58,53,47,42,37,33,29,24,
        DB 21,17,14,11,9,6,4,3,2,1,0,0,0,1,2,3,4,6,9,11,14,17,21,24,29,33,37,42,47,53,
        DB 58,64,70,76,82,88,95,101,108,115,121,128
        END
```

RESULT



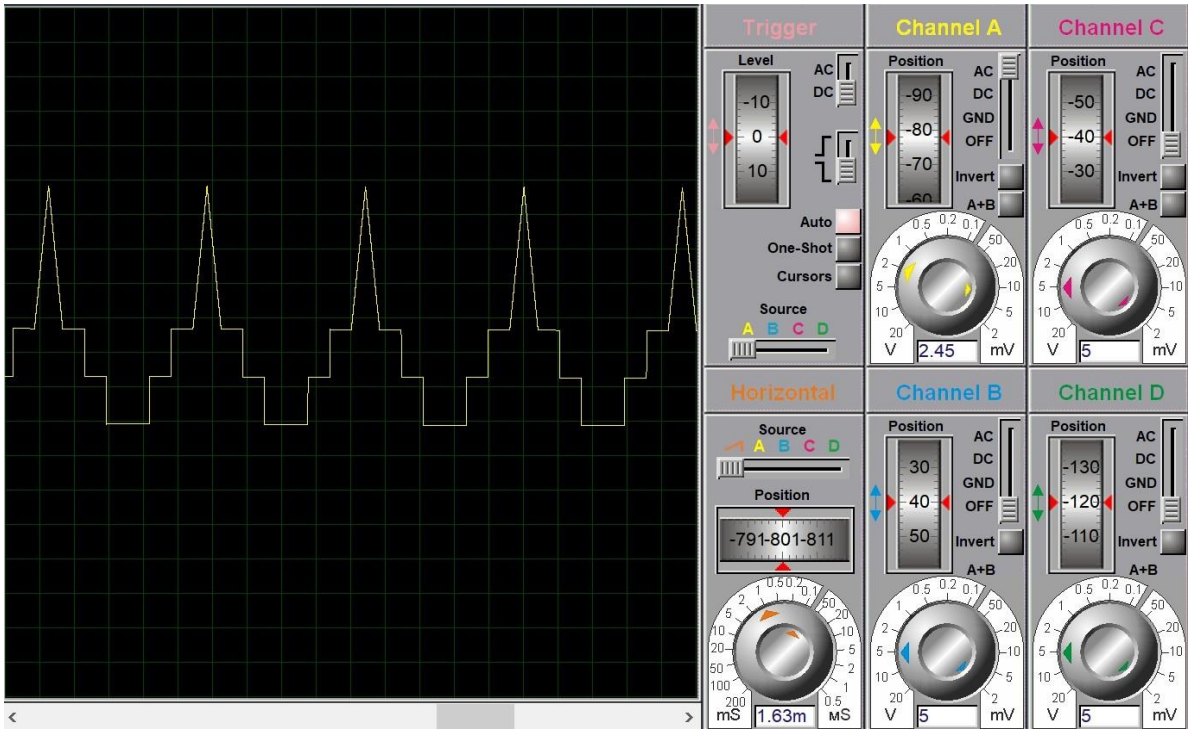
III) Generation of the following waveform.



- ALGORITHM

```
// GENERATION OF STAIRCASETRIANGLE //
        ORG 0000h
        mov p2,#00H
repeat: Acall stair_case_wave
        sjmp repeat
stair_case_wave:mov A,#00H
        mov p2,A
        Acall delay
Back:   ADD A,#33h
        mov p2,A
        Acall delay
        CJNE A,#66h,Back
START:  mov a,#66h
ON:     mov p2,a
        inc a
        cjne a,#0ffh,ON
OFF:    mov p2,a
        dec a
        cjne a,#066h,OFF
        mov A,#66H
        mov p2,A
        Acall delay
Back1:  SUBB A,#33h
        mov p2,A
        acall delay
        jnz Back1
        sjmp stair_case_wave
        ==========
delay:  mov r0,#4
up1:    mov r1,#115
here:   djnz r1,here
        djnz r0,up1
        ret
        END
```

RESULT

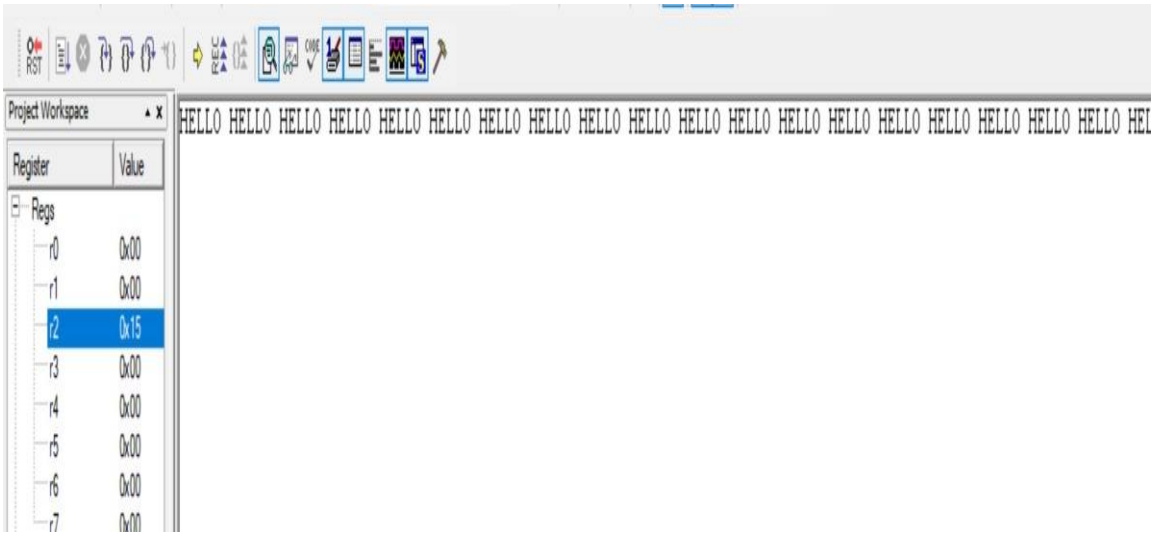


Assignment question:

1. Write an 8051 ALP to send message HELLO serially once in every 2 sec. Program timer and serial communication in interrupt mode

```
01  org 000h
02  mov TMOD,#21h       ;timer 1 in mode2 to set baudrate
03  mov TH1,#0FDH       ;set the baudrate=4800
04  mov SCON,#50h
05  setb TR1
06  mov r0,#28h
07  mov dptr,#msg       ;pointer to data
08  rpt: clr a
09  movc A,@A+DPTR      ;get the data into A
10  cjne A,#"$",send
11  ACALL DELAY         ;if data is not end of the message ,  go to send
12  sjmp rpt            ;if all characters are sent REPEAT
13  send:  mov SBUF,A   ;move the data to SBUF
14  wait:  jnb TI,wait  ;if TI=0 wait
15         clr TI       ;clear TI
16         inc DPTR     ;go to transfer the next character
17         sjmp rpt
18  delay: mov tl0,#0Bh
19          mov th0,#3Ch
20          setb TR0
21  WAIT1:  JNB TF0,WAIT1
22          Clr tr0
23          clr tf0
24          mov r0,#28h
25          djnz r0,delay
26          RET
27  msg: db "HELLO$"
28  last:
29              end
```

RESULT:



END