Sl. No	Program	Comments

Experiment no. 1: Data Transfer Programming

1. **Program to Exchange the contents of two Registers**

```
MOV R0,#30H
MOV R1,#40H
MOV A,R0
XCH A,R1
MOV R0,A
SJMP $
```

2. **Program To Transfer A Block Of Data Bytes Storing From Memory Location 30h To Another Block Starting From Memory Location 40h**

```
MOV R0,#30H
MOV R1,#40H
MOV R2,#COUNT
LOOP: MOV A,@R0
MOV@R1,A
INC R0
INC R1
DJNZ R2,LOOP
SJMP $
```

3. **Program to interchange the block of data bytes present in internal memory**

```
MOV R0,#30H
MOV R1,#40H
MOV R2,#COUNT
LOOP:MOV A,@R0
XCH A,@R1
MOV @R0,A
INC R0
INC R1
DJNZ R2,LOOP
LOOP1:SJMP LOOP1
```

4. **Program to transfer a blocks of data to another block considering overlapping blocks in internal memory or (Program to copy blocks of data in N memory locations to N+ith memory locations with overlapping(Right to Left & Vice Versa) in internal memory)**

```
Mov r0,#30h
Mov r1,#35h
Mov r2,#06h
Mov a,r0
Cjne a,01,next              ;compare A & r1, if not equal jump to next
Sjmp stop
Next : jc bottom_trf      ; if carry first address is lesser than overlap address
                          ;so overlap is from Right to Left else Left to Right overlap

   Back : mov a,@r0
          mov @r1,a
```

```
                    inc r0
                    inc r1
                    djnz r2,back   ;decrement r2 ,if not zero jump to back
      stop : sjmp stop
bottom_trf : mov a,r0          ; to Calculate the address from where the
                                ;element should be started copying
                    add a,r2
                    dec a            ; out of range so dec by 1 to get in range
                    mov r0,a     ; reinitialize the R0
                    mov a,r1      ; to Calculate the address to where the
                                ;element should be copied

                    add a,r2
                    dec a
                    mov r1,a
 back1 : mov a,@r0
                    mov @r1,a
                    dec r0
                    dec r1
                    djnz r2,back1
sjmp $
```

Output:
Before execution:
30: 1 2 3 4 5 6
35: 0 0 0 0 0 0
After execution: 30: 1 2 3 4 5 1 2 3 4 5 6

5. Program to interchange the contents of two external memory locations

```
MOV DPTR,#9000H
MOVX A,@DPTR
MOV R0,A
INC DPTR
MOVX A,@DPTR
XCH A, R0
MOVX @DPTR,A
DEC 82H
MOV A,R0
MOVX @DPTR,A
LOOP:SJMP LOOP ( or SJMP $)
```

6. Program to transfer contents from external memory locations to internal memory locations

```
mov dptr,#9000h
Mov r0,#30h
mov r2,#06h
back: movx a,@dptr
mov @r0,a
inc r0
inc dptr
djnz r2,back
```

```
end
```

7. Program to transfer a blocks of data to another block considering overlapping blocks in external memory Or (Program to copy blocks of data in N memory locations to N+ith memory locations with overlapping(Right to Left & Vice Versa) in External memory)

```
exad equ 9000h
exad1 equ 9004h
ad1 equ 30h
cnt equ 06
Mov r0,#ad1
mov r2,#cnt
mov dptr,#exad
back: movx a,@dptr
mov @r0,a
inc r0
inc dptr
djnz r2,back
mov r0,#ad1
mov r2,#cnt //sjmp nx1
mov dptr,#exad1
l1:mov a,@r0
movx @dptr,a
inc r0
inc dptr
djnz r2,l1
end
```

8. Program to copy blocks of data in N memory locations to N+ith memory locations with overlapping(Right to Left & Vice Versa) in Internal & External memory

```
exad equ 9000h
inad1 equ 30h
inad2 equ 32h
cnt equ 06
Mov r0,#inad1
Mov r1,#inad2
mov r2,#cnt
mov dptr,#exad
back: movx a,@dptr
mov @r0,a
inc r0
inc dptr
djnz r2,back
mov r0,#inad1
mov r2,#cnt //sjmp nx1
Mov a,r0
Cjne a,01,next        //compare A & r1, if not equal jump to next
Sjmp stop               //short jump to address stop
Next : jc bottom_trf  //jump if there is a carry to given address
   Back1 :mov a,r1
          anl a,#0fh
```

```
            mov dpl,a
           mov a,@r0
            mov @r1,a
            movx @dptr,a
            inc r0
            inc r1
            inc dptr
            djnz r2,back1   //dec r2 by1and jmp if not zero
   stop : sjmp stop
bottom_trf : mov a,r0
            add a,r2
            dec a
            mov r0,a
            mov a,r1
            add a,r2
            dec a
            mov r1,a
            anl a,#0Fh
            mov dpl,a
 back2 : mov a,@r0
            mov @r1,a
            movx @dptr,a
            dec 82h
            dec r0
            dec r1
            djnz r2,back2
            end
```

Output: Before execution: x:9000h:0 0 0 0 1 2 3 4 5 6
D:30h:0 0 0 0 1 2 3 4 5 6
After execution: x:9000h: 1 2 3 4 5 6 3 4 5 6
D: 30h : 1 2 3 4 5 6 3 4 5 6

9. Program to exchange (interchange) the blocks of data bytes present in External memory location starting from 9000h with 9010h memory locations

```
exad equ 9000h
exad1 equ 9010h
inad equ 30h
inad1 equ 40h
cnt equ 06h
mov dptr,#exad
MOV R0,#inad
MOV R2,#cnt
back: movx a,@dptr
mov @r0,a
inc r0
inc dptr
djnz r2,back
MOV dptr,#exad1
MOV R1,#inad1
```

```
mov r2,#cnt
back1:movx a,@dptr
mov @r1,a
inc r1
inc dptr
djnz r2,back1
mov r2,#cnt
mov r1,#inad1
MOV dptr,#exad
loop:MOV A,@R1
movx @dptr,a
inc r1
inc dptr
djnz r2,loop
mov r2,#cnt
mov r0,#inad
MOV dptr,#exad1
loop1:MOV A,@R0
movx @dptr,a
inc r0
inc dptr
djnz r2,loop1
end

OR
exad equ 9000h
exad1 equ 9010h
inad equ 30h
cnt equ 06h
mov dptr,#exad
MOV R0,#inad
MOV R2,#cnt
back: movx a,@dptr
mov @r0,a
inc r0
inc dptr
djnz r2,back
MOV dptr,#exad1
MOV R0,#inad
mov r2,#cnt
back1:movx a,@dptr
xch a,@r0
movx @dptr,a
inc r0
inc dptr
djnz r2,back1
mov r2,#cnt
mov r0,#inad
MOV dptr,#exad
loop:MOV A,@R0
```

```
movx @dptr,A
inc r0
inc dptr
djnz r2,loop
end
```

10. Program to find the largest number in an array

```
MOV R0,#40H
MOV R1, #(N-1)
MOV A,@R0
LOOP:INC R0
MOV 50H,@R0
CJNE A, 50H, NEXT
SJMP NEXT1
NEXT: JNC NEXT1
MOV A, @R0
NEXT1:DJNZ R1,LOOP
MOV 50H,A
LOOP1: SJMP LOOP1
```

11. Program to find the largest number in an array @ External Memory

```
MOV DPTR,#9000H
MOV R2, #6
MOV R0,#40H
MOV R1, #(5-1)
L1:MOVX A,@DPTR
MOV @R0,A
INC R0
INC DPTR
DJNZ R2,L1
MOV R0,#40H
MOV A,@R0
LOOP:INC R0
MOV 50H,@R0
CJNE A, 50H, NEXT
SJMP NEXT1
NEXT: JNC NEXT1
MOV A, @R0
NEXT1:DJNZ R1,LOOP
MOV 50H,A
MOV DPTR,#9010H
MOVX @DPTR,A
LOOP1: SJMP LOOP1
```

(Other way: Optimized Code)

```
MOV DPTR,#9000H
MOV R1, #(5-1)
MOVX A,@DPTR
LOOP:MOV 50H,A
INC DPTR
MOVX A,@DPTR
```

```
    CJNE A,50H, L1
    SJMP NEXT1
    L1:JNC NEXT1
    MOV A,50H
    SJMP NEXT1
    NEXT1:DJNZ R1,LOOP
    MOV DPTR,#9010H
    MOVX @DPTR,A
    LOOP1: SJMP LOOP1
```

12. Program to find the Smallest number in an array @ Internal Memory

```
        MOV R0,#44H
        MOV R1, #(N-1)
        MOV A,@R0
LOOP:INC R0
        MOV 50H,@R0
        CJNE A, 50H, NEXT
        SJMP NEXT1
NEXT: JC NEXT1
        MOV A, @R0
NEXT1:DJNZ R1,LOOP
        MOV 50H,A
LOOP1: SJMP LOOP1
```

13. Program to find the Smallest number in an array located @ External Memory

```
        MOV DPTR,#9000H
        MOV R1, #(5-1)
        MOVX A,@DPTR
        LOOP:MOV 50H,A
        INC DPTR
        MOVX A,@DPTR
        CJNE A,50H, L1
        SJMP NEXT1
        L1:JC NEXT1
        MOV A,50H
        SJMP NEXT1
        NEXT1:DJNZ R1,LOOP
        MOV DPTR,#9010H
        MOVX @DPTR,A
        LOOP1: SJMP LOOP1
```

14. Program to arrange the given set of numbers in ascending (Increasing Order-JC) order

```
        MOV R2, #(N-1)
LOOP1:  MOV R0,#40H
        MOV R3,#(N-1)
LOOP:   MOV A,@R0
        INC R0
        MOV 50H,@R0
```

```
        CJNE A, 50H, NEXT
        SJMP NEXT1
NEXT:   JC NEXT1
        MOV @R0, A
        DEC R0
        MOV @R0,50H
        INC R0
NEXT1:  DJNZ R3, LOOP
        DJNZ R2, LOOP1
LOOP2:  SJMP LOOP2
```

15. Program to arrange the given set of numbers in ascending (Increasing) order in the external memory

```
        MOV R2,#04          ; Outer Loop
LOOP1:  MOV DPTR,#9000H
        MOV R3,#04h         ; Inner Loop
LOOP:   MOVX A,@DPTR
        MOV 50H,A
        INC DPTR
        MOVX A,@DPTR
        CJNE A, 50H, NEXT
        SJMP NEXT1
NEXT:   JNC NEXT1
        DEC 82H
        MOVX @DPTR, A
        INC DPTR
        MOV A,50H
        MOVX @DPTR,A
NEXT1:  DJNZ R3, LOOP
        DJNZ R2, LOOP1
LOOP2:  SJMP LOOP2
```

16. Program to arrange the given set of numbers in descending order (JNC)

```
        MOV R2, #(N-1)
LOOP1:  MOV R0,#40H
        MOV R3,#(N-1)
LOOP:   MOV A,@R0
        INC R0
        MOV 50H,@R0
        CJNE A, 50H, NEXT
        SJMP NEXT1
NEXT:   JNC NEXT1
        MOV @R0, A
        DEC R0
        MOV @R0,50H
        INC R0
NEXT1:  DJNZ R3, LOOP
        DJNZ R2, LOOP1
LOOP2:  SJMP LOOP2
```

17. Program to arrange the given set of numbers in descending order in External Memory

```
        MOV R2, #(N-1)
LOOP1:  MOV DPTR,#9000H
        MOV R3, #(N-1)
LOOP:   MOVX A,@DPTR
        MOV 50H,A
        INC DPTR
        MOVX A,@DPTR
        CJNE A, 50H, NEXT
        SJMP NEXT1
NEXT:   JNC NEXT1
        DEC 82H
        MOVX @DPTR, A
        INC DPTR
        MOV A,50H
        MOVX @DPTR,A
NEXT1:  DJNZ R3, LOOP
        DJNZ R2, LOOP1
LOOP2:  SJMP LOOP2
```

Experiment no. 2: Arithmetic Instruction programming

18. Program to add two 8bit number present in internal memory location and store the result in next consecutive memory location

```
        MOV R1,#00H
        MOV R0,#30H
        MOV A,@R0
        INC R0

        ADD A,@R0
        JNC NEXT
        INC R1
   NEXT: INC R0
        MOV @R0,A
        INC RO
        MOV A,R1
        MOV @RO,A
        SJMP $
```

19. Program to Subtract two 8bit number present in internal memory location and store the result in next consecutive memory location

```
        CLR C
        MOV R1,#00H
        MOV R0,#30H
```

```
        MOV A,@R0
        INC R0
        SUBB A,@R0
        JNC NEXT
        INC R1
NEXT: INC R0
        MOV @R0,A
        INC RO
        MOV A,R1
        MOV @RO,A
        SJMP $
```

20. Program to add two 16 bit numbers

```
Clr c                    //clear the carry flag
mov r0,#30h
mov r1,#40h
mov a,@r0
add a,@r1
mov 50h,a
Inc r0
Inc r1
mov a,@r0
addc a,@r1    //add contents of A and address in r1 with carry
mov 51h,a
end
```

Output:

Before execution:

30: 1234

40:7453

After execution:

50: 87 86

21. Program to subtract two 16 bit numbers

```
Clr c
mov r0,#30h
mov r1,#40h
mov a,@r0
subb a,@r1
mov 50h,a
Inc r0
Inc r1
mov a,@r0
subb a,@r1
mov 51h,a
end
```

Output:
Before execution:30: 8234
40:7453
After execution:50: D0 0C

22. Program to perform 8bit binary multiplication

```
        MOV R0,#30H
        MOV A,@R0
        INC R0
        MOV 0F0H,@R0
        MUL AB
        INC R0
        MOV@R0,A
        INC R0
        MOV @R0,0F0H
LOOP:SJMP LOOP
```

23. Program to perform 8bit binary DIVISION

```
MOV R0,#30H
MOV A,@R0
INC R0
MOV 0F0H,@R0
DIV AB
INC R0
MOV@R0,A
INC R0
MOV @R0,0F0H
LOOP:SJMP LOOP
```

24. Program to find square of a given 8 bit number

```
MOV R,#30H
MOV A,@R1
MOV 0F0H, A
MUL AB
MOV 50H, A
MOV 51H, 0F0H
SJMP $
```

25. Cube of a Number

```
MOV R1,#30H
MOV A,@R1
MOV 0F0H,A
MUL AB
MOV R3,0F0H
MOV 0F0H, @R1
MUL AB
MOV 50H,A
MOV R5,0F0H
MOV 0F0H,@R1
MOV A,R3
```

```
MUL AB
ADD A,R5
MOV 51H, A
MOV 52H, 0F0H
SJMP $

OUTPUT:
        D: x30H: 25
        50h: 2556 01
```

26. Program to Perform 16 bit multiplication

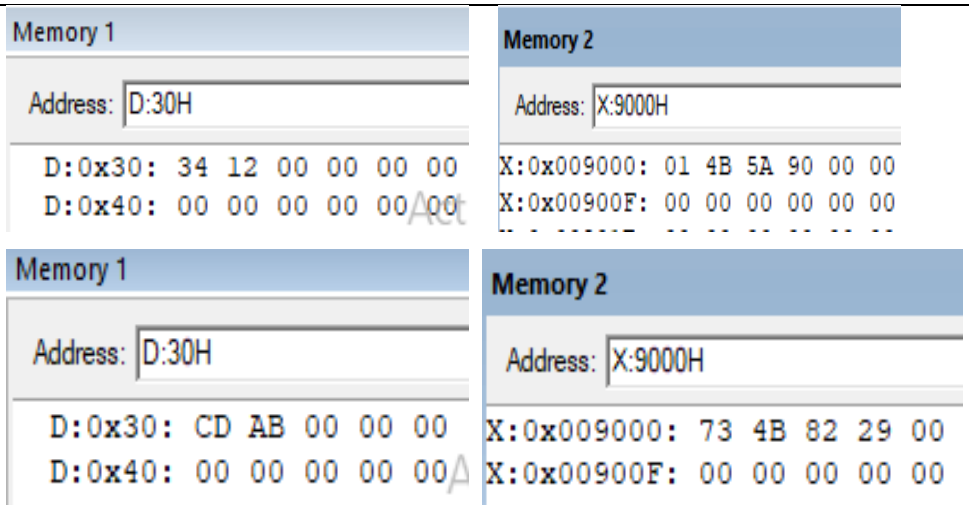
```
MOV DPTR,#9000H
MOV R4,#00
MOV R0,#30H
MOV R1,#40H
MOV A,@R0
MOV 0F0H,@R1
MUL AB
MOVX @DPTR, A
MOV R2,0F0H
INC R1
MOV A,@RO
MOV 0F0H,@R1
MUL AB
ADD A,R2
MOV R2,A
MOV A,0F0H
ADDC A,#00H
MOV R3,A
INC RO
DEC R1
MOV A,@R0
MOV 0F0H,@R1
MUL AB
ADD A,R2
INC DPTR
MOVX @DPTR,A
```

```
MOV A,0F0H
ADDC A,R3
MOV R3,A
JNC NEXT
INC R4
NEXT:   INC R1
    MOV A,@R0
MOV 0F0H,@R1
    MUL AB
    ADD A,R3
    INC DPTR
    MOVX @DPTR,A
    MOV A,0F0H
    ADDC A,R4
    INC DPTR
    MOVX @DPTR,A
    LOOP:SJMP LOOP
```

27. Program to Perform Square of a 16 bit Number

```
    MOV DPTR,#9003H
    MOV R1,#00
    MOV R0,#30H
    MOV A,@R0
    MOV R4,A
    INC R0
    MOV A,@R0
    MOV R5,A
    DEC R0
    MOV A,@R0
    MOV B,R4
    MUL AB
    MOVX @DPTR,A
    MOV R2,0F0H
    MOV A,@R0
```

```
MOV B,R5
MUL AB
ADD A,R2
MOV R2,A
MOV A,0F0H
ADDC A,#00H
MOV R3,A
INC R0
MOV A,@R0
MOV B,R4
MUL AB
ADD A,R2
DEC DPL
MOVX @DPTR,A
MOV A,0F0H
ADDC A,R3
MOV R3,A
JNC NEXT1
INC R1
NEXT1:MOV A,@R0
MOV 0F0H,R5
MUL AB
ADD A,R3
DEC DPL
MOVX @DPTR,A
MOV A,0F0H
ADDC A,R1
DEC DPL
MOVX @DPTR,A
LOOP:SJMP LOOP
```



Experiment no. 3: Boolean and Logical Instructions programming

28. Program to check whether given data in memory is even or odd if even pass FFH – R1 register , else 00H

```
        mov r0,#30h
        mov a,@r0
        jnb acc.0,even
        mov r1,#00h
        even:mov r1,#0ffh
        end
```

29. Program to check whether given bytes of data in internal memory locations is even or odd , separate even and odd numbers in different internal memory locations

```
    cnt1 equ 30h
    mov r0,#cnt1
    mov r1,#40h
    cnt equ 08h
    mov r2,#cnt
    l1:mov a,@r0
    jb acc.0,od
    inc r0
    djnz r2,l1
    sjmp arng
    od:mov a,@r1
    xch a,@r0
    mov @r1,a
    inc r1
    inc r0
    djnz r2,l1
    arng:MOV R2, #(cnt-1) ; Descending order pgm to arrange even elements
LOOP1:  MOV R0,#cnt1
        MOV R3,#(cnt-1)
LOOP2:  MOV A,@R0
        INC R0
        MOV 50H,@R0
        CJNE A, 50H, NEXT
        SJMP NEXT1
```

```
NEXT:   JNC NEXT1
        MOV @R0, A
        DEC R0
        MOV @R0,50H
        INC R0
NEXT1:  DJNZ R3, LOOP2
        DJNZ R2, LOOP1
        end
```

30. Program to check whether given bytes of data in External memory locations is even or odd , separate even and odd numbers in different external memory locations

```
MOV DPTR,#9000H
AD equ 30h
mov r0,#AD
mov r1,#40h
cnt equ 08h
mov r2,#cnt
l1:MOVX A,@DPTR
jb acc.0,od
inc dptr
djnz r2,l1
SJMP TRNSODD
od:xch a,@r0
MOVX @DPTR,A
INC DPTR
INC R0
DJNZ R2,l1
TRNSODD:MOV A,R0
anl a,#0fh
MOV R3,A
MOV R0,#AD
MOV DPTR,#9010H
L2:MOV A,@R0
MOVX @DPTR,A
INC DPTR
INC R0
DJNZ R3,L2

arng:MOV R2,#(cnt-1)            ; Outer Loop
LOOP1:  MOV DPTR,#9000H
        MOV R3,#(cnt-1)         ; Inner Loop
LOOP:   MOVX A,@DPTR
        MOV 50H,A
        INC DPTR
        MOVX A,@DPTR
        CJNE A, 50H, NEXT
        SJMP NEXT1
NEXT:   JC NEXT1
        DEC 82H
```

```
            MOVX @DPTR, A
            INC DPTR
            MOV A,50H
            MOVX @DPTR,A
NEXT1:      DJNZ R3, LOOP
            DJNZ R2, LOOP1
LOOP2:      SJMP LOOP2
```

31. Program to check whether the given number is nibble wise palindrome or not

```
        MOV R0,#40H
        MOV A,@R0
        SWAP A
        CJNE A, 40H, NOT PAL
        INC RO
        MOV @R0, #00
LOOP:SJMP LOOP
NOTPAL: INC RO
        MOV @R0,#0FFH
LOOP1: SJMP LOOP1
```

32. Program to logically AND, OR, XOR two 8 bit numbers in the internal memory locations and store the result in successive memory locations

```
MOV R0,#30H
MOV A,@R0
MOV R2,A
INC R0
MOV A,@R0
MOV R3,A
MOV A,R2
ANL A,R3        //logically AND the contents
INC R0
MOV @R0,A
MOV A,R2
ORL A,R3        //logically OR the contents
INC R0
MOV @R0,A
MOV A,R2
XRL A,R3        //logically XOR the contents
INC R0
MOV @R0,A
SJMP $
```

33. Write an ALP to compare two eight bit numbers NUM1 and NUM2 stored in external memory locations 8000h and 8001h respectively. Reflect your result as: If NUM1<NUM2, SET LSB of data RAM location 2FH (bit address 78H). If NUM1>NUM2, SET MSB of location 2FH (bit address 7FH). If NUM1 = NUM2, then set both LSB & MSB of bit addressable memory location 2FH.

```
    mov  dptr,#8001h
    movx a,@dptr
    mov  r0,a
    dec  dpl
    movx a,@dptr
```

```
clr c
subb a,r0     ;( Compare instrun with a and r0 is not there , So Subb is used ,
              ;Compare instruction allowed are cjne a, direct ,
               ; cjne a,#data, cjne Rn,#data, cjne @Ri,#data )
jz  equal
jnc  greater
setb 78h
sjmp  end1
greater: setb  7Fh
sjmp  end1
equal: setb  78h
setb  7fh
end1:sjmp end1
```

34. Write an assembly language program to count number of ones and zeros in a eight bit number

```
mov r0,#30h
mov r1,#00h  // to count number of 0s
mov  r2,#00h  // to count number of 1s
mov  r7,#08h  // counter for 8-bits
mov  a,@r0    //Bring data to count number of 1s and 0s
again: rlc  a
jc  next
inc  r1
sjmp here
next: inc  r2
here: djnz  r7,again
end
```

Experiment no. 4: Code Conversion Programs:

35. Decimal to Hexadecimal Conversion (Last 8 Bit Value in Decimal is 99 so given data should be less than or equal to 99)

```
;ORG 0000H
;SJMP30h
;ORG 30h
MOV DPTR,#40H      //2-digit decimal number to be converted
                   ; Is given in data memory 40h
MOVX A, @DPTR
ANL A, #0F0H             //obtain upper decimal digit
SWAP A                   //bring to the units place
MOV B,#0AH               //MULTIPLY tens digit with#0Ato
get tens in hex
MUL AB
MOV r1,a           //temporarily store the converted tens value
MOVX A,@DPTR    //get the decimal number again
ANL A,#0FH         //obtain the units digit
ADD A,R1           //add to the converted tens value
INC DPTR           //increment data address
MOVX @DPTR,A   //converted hexadecimal number in next location
END
```

OUTPUT:

RESULT: before execution- X:0040H = 45 (Decimal/BCD)
After Execution: X:0041h = 2D (hex value)

36. Hexadecimal to Decimal

```
MOV R0,#30H
MOV A,@RO
MOV 0F0H, #0AH
DIV AB
MOV R1,0F0H
MOV 0F0H, #0AH
DIV AB
MOV 40H,A
MOV A,0F0H
SWAP A
ORL A,R1
MOV 41H, A
END
```

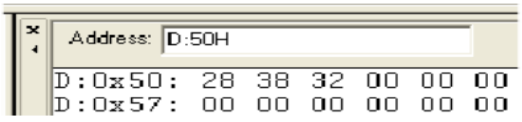
OUTPUT: D 30H: FF
D40H:02 55

37. BCD TO ASCII

```
ORG 0000H
SJMP30h
ORG 30h
MOV R1,#50H
MOV A,@R1        //get BCD data byte from RAM location 50h
MOV R2,A         //Store in R2
ANL A,#0FH       //Get the lower nibble
ORL A,#30H       //Add/or with 30h i.e., 0-9 converted to 30-39h
INC R1
MOV @R1,A        //Store the lower digit's ASCII code
MOV A,R2         //Get back the number
 SWAP A          //Swap nibbles in A
ANL A,#0F H      //Get the upper BCD digit
ORL A,#30H       //Convert to ASCII
INC R1
MOV @R1,A        //Store the upper digit's ASCII code
here: sjmp here
END
```

OUTPUT:

RESULT: The BCD code 28 at D:0050h is converted to 2 ASCII codes-38h 32

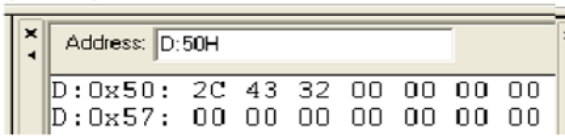


38. HEXADECIMAL TO ASCII

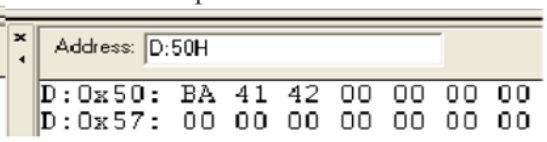
```
ORG 0000H
SJMP30h
ORG 30h
MOV R1,#50H
MOV A,@R1          //get hexadecimal data byte from
RAM location 50h
MOV R2,A           //Store in R2
ANL A,#0FH         //Get the lower nibble
ACALL ASCII        //Convert to ASCII
INC R1
MOV @R1,A          //Store the lower digit's ASCII code
MOV A,R2 //Get back the number
SWAP A //Swap nibbles in A
ANL A,#0FH //Get the upper BCD digit
ACALL ASCII
INC R1
MOV @R1,A //Store the upper digit's ASCII code
here: sjmp here
ASCII:MOV R4,A //Store a
CLR C
SUBB A,#0AH //Check if digit >=0A
MOV A,R4
JC SKIP
ADD A,#07H //Add 07 if >09
SKIP:ADD A,#30H //Else add only 30h for 0-9
RET
END
```

OUTPUT:

RESULT: The BCD code 2C at D:0050h is converted to 2 ASCII codes-43h(for 0B) & 32h (for 02)



Another Example-BA



39. ASCII TO HEXADECIMAL

```
ORG 0000H
SJMP30h
ORG 30h
MOV R1,#50H
MOV A,@R1                  //get ascii byte from RAM location 50h
CLR C
SUBB A,#41H
                           //MOV A,@R1
JC SKIP
SUBB A,#07H
```

```
SKIP:SUBB A,#30H
INC R1
MOV @R1,A                    //Store the hex code
here: sjmp here
END
```

OUTPUT:

RESULT: The ASCII code 45 at D:0050h is converted to hexadecimal -0E at 51h

Note: For this program the input data should be only in the range **30h-39h** & **41h to 46h**.

40. DECIMAL TO ASCII

```
ORG 0000H
SJMP 30h
ORG  30h
MOV R1,#50H
MOV A,@R1       //get BCD data byte from RAM location 50h
MOV R2,A        //Store in R2
ANL A,#0FH      //Get the lower nibble
ORL A,#30H      //Add/or with 30h i.e., 0-9 converted to 30-39h
INC R1
MOV @R1,A       //Store the lower digit's ASCII code
MOV A,R2        //Get back the number
 SWAP A         //Swap nibbles in A
ANL A,#0FH      //Get the upper BCD digit
ORL A,#30H      //Convert to ASCII
INC R1
MOV @R1,A       //Store the upper digit's ASCII code
here: sjmp here
END
```

OUTPUT: D50H: 34 34 33

41. ASCII TO DEC

```
MOV R0,#30H
MOV A,@R0
SUBB A,#30H
INC R0
MOV @R0,A
END
```

OUTPUT: D50H : 34 04

Experiment no. 5: Counter programming

42. Write an assembly language program to implement (display) an eight bit UP, binary (hex) counter on watch window(ONCE/ CONTINUOUSLY)

```
up:mov a,#00
back: acall delay              ; (back: MOV 30h,a for counting in Internal memory )
inc a
jnz back
sjmp up              ;( terminate the program here THIS PGM  WILL EXECUTE
                              ; only once)
delay: mov r1,#055h
decr1: mov r2,#066h
decr: mov r3,#0FFh
djnz r3,$
djnz r2,decr
djnz r1,decr1
ret
end
```

RESULT: Accumulator A is incremented in binary from
00, 01, 02…09,0A, 0B,…,0F,10,11,…FF
Note: To run this program, after selecting DEBUG session in the main menu use **View-> Watch & call Stack window**, in the Watches select watch 1(or 2) and press F2 and enter a (for accumulator A)

43. Write an assembly language program to implement (display) an eight bit DOWN binary (hex) DOWN on watch window(ONCE/ CONTINUOUSLY)

```
down: mov a,#0FFH
back: acall delay
DEC a
jnz back
mov a,#00H
acall delay
sjmp down     ;( terminate the program here THIS PGM  WILL EXECUTE
          ; only once)

delay: mov r1,#055h
decr1: mov r2,#066h
decr: mov r3,#0FFh
djnz r3,$
djnz r2,decr
djnz r1,decr1
ret
end
```

RESULT: Accumulator A is incremented in binary from
FF,FE,FD.....00.. FF,FE,FD.....00 (CONTINUOUSLY)

44. Write an assembly language program to implement (display) an eight bit decimal UP counter on watch window(ONCE/ CONTINUOUSLY)

(To understand this pgm remove delay pgm and comment acall delay and press F11 , you will understand step wise)

```
up: mov a,#00h
back: acall delay
add a,#01h
da a                                    ;decimal adjust accumulator
jnz back
sjmp up;( terminate the program here THIS PGM  WILL EXECUTE
                                        ; only once)
delay: mov r1,#055h
decr1: mov r2,#066h
decr: mov r3, #0ffh
djnz r3,$
djnz r2, decr
djnz r1, decr1
ret
end
```

RESULT: Accumulator A is incremented in BCD from 00,01,02,03.....99...00,01,02,03....99 (Continuously)

45. Write an assembly language program to implement (display) an eight bit Decimal Down counter on watch window(ONCE/ CONTINUOUSLY)

```
down: mov a,#99h
back: acall delay
add a,#99h
da a                                    ;decimal adjust accumulator
jnz back
mov a,#00h
acall delay
sjmp down;( terminate the program here THIS PGM  WILL EXECUTE
                ; only once)
delay: mov r1,#0aah
decr1: mov r2,#0bbh
decr: mov r3, #0ffh
djnz r3,$
djnz r2, decr
djnz r1, decr1
ret
end
```

RESULT: Accumulator A is incremented in BCD from 99,98,.....00...99,98...00 (Continuously)

46. Program to implement 8bit Binary(hex) up counter at external memory location (Once/Continuously)

```
mov dptr,#9000h
up:mov a,#00h
back: movx @dptr,a
```

```
acall delay
inc a
jnz back
sjmp up                 ;( terminate the program here THIS PGM  WILL EXECUTE
                                ; only once)

delay: mov r1,#55h
loop1: mov r2,#066h
loop2: mov r3,#0ffh
loop3: djnz r3,loop3
djnz r2,loop2
djnz r1,loop1
ret
end
```

Output: x:9000h=00,01,02....ff.....00,01,02.....ff (Continuously)

47. Program to implement 8bit Binary(hex) Down counter at external memory location (Once/Continuously)

```
mov dptr,#9000h
down:mov a,#0ffh
back: movx @dptr,a
acall delay
dec a
jnz back
sjmp down               ;( terminate the program here THIS PGM  WILL EXECUTE
                                ; only once)

delay: mov r1,#99h
loop1: mov r2,#0aah
loop2: mov r3,#0ffh
loop3: djnz r3,loop3
djnz r2,loop2
djnz r1,loop1
ret
end
```

Output: x:9000h=ff,fe,fd,.....00..ff,fe,fd.....00 (Continuously)

48. Program for Decimal up counter at external memory location (Once/Continuously)

```
mov dptr,#9000h
up:mov a,#00h
back: movx @dptr,a
acall delay
add a,#01h
da a
```

```
jnz back
sjmp up;( terminate the program here THIS PGM  WILL EXECUTE
                    ; only once)
delay: mov r1,#55h
loop1: mov r2,#66h
loop2: mov r3,#0ffh
loop3: djnz r3,loop3
djnz r2,loop2
djnz r1,loop1
ret
end
```

Output: x : 9000h=00,01,02.....99...00,01,02,....99 (Continuously)

49. Program for Decimal down counter at external memory location (Once/Continuously)

```
mov dptr,#9000h
down:mov a,#99h
back: movx @dptr,a
acall delay
add a,#99h
da a
jnz back
movx @dptr,a
sjmp down;( terminate the program here THIS PGM  WILL EXECUTE
                    ; only once)
delay: mov r1,#55h
loop1: mov r2,#66h
loop2: mov r3,#0ffh
loop3: djnz r3,loop3
djnz r2,loop2
djnz r1,loop1
ret
end
```

Output: x: 9000 h= 99,98,97……00...99,98,97....00 (continuously)

50. Design MOD – N UP Counter CONTINUOUSLY

```
val equ 20h
up:MOV A,#00H          ;load the accumulator with 00H
back: LCALL DELAY      ;call delay
ADD A,#01H             ; Add accumulator by 1
DA A                   ; Decimal Adjust After addition to get decimal value
CJNE A,#val,back       ;compare A with MODN Value if not equal jump to up
SJMP up          ;( terminate the program here This Pgm Will Execute only
once)

DELAY: MOV R0,#099H    ; load r0 by FFH
```

```
BACK1: MOV R1,#0aaH    ; load r1 by FFH
BACK2: MOV R2,#0FFH    ; load r2 by FFH
HERE: DJNZ R2,HERE     ; decrement r2 if not equal to zero jump here
DJNZ R1, BACK2         ;decrement r1 if not equal to zero jump Back
DJNZ R0,BACK1          ;decrement r0 if not equal to zero jump Back1
RET
END
```

RESULT: Accumulator A is incremented 00, 01.... (N-1) (Continuously)

51. Design MOD - N DOWN Counter CONTINUOUSLY

```
INIT EQU 19H
DOWN:MOV A,#INIT       ;load the accumulator with MODN-1 Value for down
                       cntng ;dnt use bracket ( Just give values, Ex: MOD 20 : 20-1
                       =19H)
BACK: LCALL DELAY      ;call delay
ADD A,#99H             ; decrement accumulator by 1
DA A
CJNE A,#00H,back    ;compare accumulator with 00H if not equal jump up
LCALL DELAY
MOV A,#00H
SJMP down;( terminate the program here THIS PGM  WILL EXECUTE
                       ; only once)
DELAY: MOV R0,#099H    ; load r0 by FFH
BACK1: MOV R1,#0aaH    ; load r1 by FFH
BACK2: MOV R2,#0FFH    ; load r2 by FFH
HERE: DJNZ R2,HERE     ; decrement r2 if not equal to zero jump here
DJNZ R1, BACK2         ; decrement r1 if not equal to zero jump Back
DJNZ R0,BACK1          ;decrement r0 if not equal to zero jump Back1
RET
END
```

52. Design MOD - N UP Counter CONTINUOUSLY at external memory location

```
MOV DPTR,#9000H
up:MOV A, #00H
back:MOVX @DPTR, A
ACALL DELAY            ;call delay
ADD A,#01H
DA A
CJNE A,#(N)H,back   ;compare accumulator with 00H if not equal jump up
SJMP up          ;( terminate the program here THIS PGM WILL EXECUTE
              ; only once)
DELAY: MOV R0,#099H    ; load r0 by FFH
BACK1: MOV R1,#0aaH    ; load r1 by FFH
BACK2: MOV R2,#0FFH    ; load r2 by FFH
HERE: DJNZ R2,HERE     ; decrement r2 if not equal to zero jump here
DJNZ R1, BACK2         ;decrement r1 if not equal to zero jump Back
DJNZ R0,BACK1    ;decrement r0 if not equal to zero jump Back1
```

```
RET
END
```

RESULT: Accumulator A is incremented 00, 01.... (N-1) (Continuously)

53. Design MOD - N Down Counter CONTINUOUSLY at external memory location

```
MOV DPTR,#9000H
down:MOV A,#(N-1)H
back:MOVX @DPTR,A
ACALL DELAY              ;call delay
ADD A,#99H
DA A
CJNE A,#00H,back   ;compare accumulator with 00H if not equal jump up
MOVX @DPTR,A
ACALL DELAY
sjmp down             ;( terminate the program here THIS PGM WILL EXECUTE
        ; only once
DELAY: MOV R0,#99H     ; load r0 by FFH
BACK1: MOV R1,#0AAH  ; load r1 by FFH
BACK2: MOV R2,#0FFH   ; load r2 by FFH
HERE: DJNZ R2,HERE     ; decrement r2 if not equal to zero jump here
DJNZ R1, BACK2           ;decrement r1 if not equal to zero jump Back
DJNZ R0,BACK1            ;decrement r0 if not equal to zero jump Back1
RET
END
```

RESULT:)X:9000:n-1.....00....n-1.....00(Continuously)

54. ALP TO MOD-N UP&DOWN COUNTER on WATCH WINDOW CONTINUOUSLY

```
up:MOV A,#00H            ;load the accumulator with 00H
back:LCALL DELAY       ;call delay
ADD A,#01H    ; increment accumulator by 1
DA A
CJNE A,#(N)H,back      ;compare accumulator with 40h if not equal jump
;up
down:ADD A,#99H
DA A
LCALL DELAY
CJNE A,#00H, down
SJMP up;( terminate the program here THIS PGM WILL EXECUTE
        ; only once
DELAY:MOV R0,#099H     ; load r0 by FFH
BACK1:MOV R1,#0aaH     ; load r1 by FFH
BACK2:MOV R2,#0FFH     ; load r2 by FFH
HERE:DJNZ R2,HERE      ; decrement r2 if not equal to zero jump here
DJNZ R1, BACK2    ;decrement r1 if not equal to zero jump Back
DJNZ R0,BACK1     ;decrement r0 if not equal to zero jump Back1
```

```
RET                 ;return to main
END
```

RESULT: A: 0 .....N-1 .....0 CONTINUOUSLY

55. ALP TO MOD-N UP &DOWN COUNTER AT EXTERNAL MEMORY LOCATION CONTINUOUSLY

```
MOV DPTR,#9000H
up:MOV A,#00H           ;load the accumulator with 00H
back:MOVX @DPTR,A
LCALL DELAY     ;call delay
ADD A,#01H    ; increment accumulator by 1
DA A
CJNE A,#(N)H,back     ;compare accumulator with 40h if not equal jump ;up
down:ADD A,#99H
DA A
MOVX @DPTR,A
LCALL DELAY
CJNE A,#00H, down
SJMP up;( terminate the program here THIS PGM WILL EXECUTE
       ; only once)
DELAY:MOV R0,#099H    ; load r0 by FFH
BACK1:MOV R1,#0aaH    ; load r1 by FFH
BACK2:MOV R2,#0FFH    ; load r2 by FFH
HERE:DJNZ R2,HERE     ; decrement r2 if not equal to zero jump here
DJNZ R1, BACK2    ;decrement r1 if not equal to zero jump Back
DJNZ R0,BACK1     ;decrement r0 if not equal to zero jump Back1
RET                 ;return to main
END
```

Experiment no. 6: Delay Generation using Timer/Counter Programming

56. Generate a delay of 15us .. f=24MHz

```
MOV TMOD,#01                ;Timer 0, mode 1(16-bit mode)
HERE: MOV TL0,#0F2H         ;TL0=F2H, the low byte
MOV TH0,#0FFH               ;TH0=FFH, the high byte
CPL P1.5                    ;toggle P1.5
ACALL DELAY
SJMP HERE
DELAY:SETB TR0              ;start the timer 0
AGAIN: JNB TF0,AGAIN        ;monitor timer flag 0
                            ;until it rolls over
CLR TR0                      ;stop timer 0
CLR TF0             ;clear timer 0 flag
RET
END
```

57. Generate a delay of 20ms

```
MOV TMOD,#01 ;Timer 0, 16-bitmode
HERE: MOV TL0,#3EH ;TL0=3Eh, the low byte
MOV TH0,#0B8H ;TH0=B8H, the high byte
CPL P2.3 ;SET high timer 0
SETB TR0 ;Start the timer 0
AGAIN: JNB TF0,AGAIN ;Monitor timer flag 0
CLR TR0 ;Stop the timer 0
CLR TF0 ;Clear TF0 for next round
;CLR P2.3
SJMP HERE
END

;f=11.06MHz
;CLR P2.3 ;Clear P2.3
MOV TMOD,#01 ;Timer 0, 16-bitmode
HERE: MOV TL0,#3EH ;TL0=3Eh, the low byte
MOV TH0,#0B8H ;TH0=B8H, the high byte
CPL P2.3 ;SET high timer 0
SETB TR0 ;Start the timer 0
AGAIN: JNB TF0,AGAIN ;Monitor timer flag 0
CLR TR0 ;Stop the timer 0
CLR TF0 ;Clear TF0 for next round
;CLR P2.3
SJMP HERE
   END
```

58. Generate a delay of , 5 ms f=1.4 MHz

```
MOV TMOD,#10;Timer 1, mod 1 (16-bitmode)
AGAIN: MOV TL1,#34H ;TL1=34H, low byte of timer
MOV TH1,#76H ;TH1=76H, high byte timer
CPL P1.5
SETB TR1 ;start the timer 1
BACK: JNB TF1,BACK ;till timer rolls over
CLR TR1 ;stop the timer 1
CLR TF1 ;clear timer flag 1
SJMP AGAIN ;is not auto-reload
END

;f=11.06 MHz, 5 ms
CLR P2.3 ;Clear P2.3
MOV TMOD,#01 ;Timer 0, 16-bitmode
HERE: MOV TL0,#0 ;TL0=0, the low byte
MOV TH0,#0EEH ;TH0=EE, the high byte
CPL P2.3 ;SET high P2.3
SETB TR0 ;Start timer 0
AGAIN: JNB TF0,AGAIN ;Monitor timer flag 0
CLR TR0 ;Stop the timer 0
CLR TF0 ;Clear timer 0 flag
SJMP HERE
END
```

59. Generate a Delay of 1 Second

```
mov tmod,#01
loop:mov r0,14h
here:mov tl0,#6Bh
mov th0,#4Bh
acall delay
djnz r0,here
cpl P1.5
sjmp loop
delay:setb tr0
again:jnb tf0,again
clr tr0
clr tf0
ret
end
```

Experiment no. 7: Serial Communication programming

60. Conduct an experiment to configure 8051 microcontroller to transmit characters "ENTER YOUR NAME" to a PC using the serial port and display on the serial window.

Note: To use result of this program, after selecting DEBUG session in the main menu use View-> serial window #1. On running & halting the program, the data is seen in the serial window.

```
mov tmod,#20h        //setting Timer-1 in mode-2
mov scon,#70h
mov th1,#-3
setb tr1
again: mov r0,#03h
mov dptr,#8000h
nextchar: movx a,@dptr
acall transfer
inc dptr
djnz r0,nextchar
sjmp again
transfer: mov sbuf,a
wait:   jnb ti,wait
clr ti
ret
end
```

61. Conduct an experiment to configure 8051 microcontroller to transmit characters "YES" to a PC using the serial port and display on the serial window.

```
MOV TMOD,#20H ;timer 1,mode 2(auto reload)
MOV TH1,#-3 ;9600 baud rate
MOV SCON,#50H ;8-bit, 1 stop, REN enabled
SETB TR1 ;start timer 1
```

```
AGAIN: MOV A,#'Y' ;transfer “Y”
ACALL TRANS
MOV A,#'E' ;transfer “E”
ACALL TRANS
MOV A,#'S' ;transfer “S”
ACALL TRANS
MOV A,#' '
ACALL TRANS
SJMP AGAIN ;keep doing it
;serial data transfer subroutine
TRANS: MOV SBUF,A ;load SBUF
HERE: JNB TI,HERE ;wait for the last bit
CLR TI ;get ready for next byte
RET
END
```

Part B: Programming in C to interface with 8051

1. Alpha Numeric LCD panel to display the names

```
//To display message on lcd display
#include <reg51.h>
#define ldata P1                                        // define ldata to P1
sbit rs = P0^4;                                         // single bits
sbit rw = P0^5;
sbit en = P0^6;
sbit back_lite =P0^7;

void MSDelay(unsigned int);                             // Prototyping
void lcdcmd(unsigned char );
void lcddata(unsigned char );
void lcdready();
void main()
        {
                unsigned char lcd_command[]={0x38,0x0e,0x01,0x06,0x83};     //array of
command send to lcd intialising  for display on first line
                unsigned char lcd_message[]=" i2c logic     ";              // string to be
display on lcd   first line

                unsigned char lcd_command1[]={0x38,0x0e,0x06,0xc0,0xc2};    // array of
command send to lcd intialising  for display on second line
                unsigned char lcd_message1[]=" i2c-200-URD3   ";            // string to be
display on lcd second line
                unsigned char c,d;

                back_lite =0;                                       // backlite

                for(c=0;c<5;c++)
                {
                lcdcmd(lcd_command[c]);                 //function send the first
command
                }

                for(d=0;d<15;d++)
                {
                lcddata(lcd_message[d]);            //send data to line from lcd first
massage
                MSDelay(30);
                }

back_lite =1;

                for(c=0;c<5;c++)
                {
```

```
                lcdcmd(lcd_command1[c]);          //function send the command
for second line
        }

        for(d=0;d<15;d++)
        {
                lcddata(lcd_message1[d]);          //send data to line from lcd
second massage
                MSDelay(30);
        }
        while(1)
        {
        }
}


void lcdcmd(unsigned char value)                  //function to send command
{
        ldata=value;                              //put the value on the pins
        rs=0;                                     //rs =0 for commands
        rw=0;                                     // rw =0 to write on lcd
        en=1;                                     // send pulse on enable
        MSDelay(1);
        en=0;
        return;
 }


void lcddata(unsigned char value)                 // function to send data
{
        ldata = value;                            // put the value on the pin
        rs=1;                                     //rs =1 for data
        rw=0;                                     // rw =0 to write on lcd
        en=1;                                     // send pulse on enable
        MSDelay(1);
        en=0;
        return;
 }


void MSDelay(unsigned int itime)                  //delay function
{
        unsigned int i,j;
        for(i=0;i<itime;i++)
        for(j=0;j<1275;j++);
}
```

2. External ADC and Temperature control

```
#include<reg51.h>
#define sevn_seg_databus P1                    //use name seven_seg_databus to P1
#define adc_data P2                             //use name adc_data to P2

sbit digit0=P0^2;                             //disply digit for first seven_segment digit
sbit digit1=P0^3;                              // disply digit for second seven_segment digit

        void delay_ms(unsigned int );  //delay Prototype
        void data_disp();                       //adc display prototype


void main(void)                                 // main program start
{

        while(1)                                //contineuos loop
        {

                data_disp();                    //call data display

        }
}

void data_disp()
{
        unsigned char temp;
   unsigned bcd_code[]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,
                                        0x6f,0x77,0x7f,0x39,0x5e,0x79,0x71};
//array to display on sevensegment

        temp = adc_data;                        //take adc data in temp for first digit
        temp &= 0xf0;                           //mask with F0 to get first digit
        temp = temp >> 4;                       // shift it 4 time to right
        digit0=0;                               // on first digit
        sevn_seg_databus = bcd_code[temp];    //assign data to sevensegemnt to display
first digit
        delay_ms(1);                            // call delay
        digit0=1;                               // off first digit

        temp = adc_data;                        //take adc data in temp for second
digit
        temp &= 0x0f;                                   //mask with 0F for
second digit

        digit1=0;                                       // on secod digit
        sevn_seg_databus = bcd_code[temp];      //assign data to sevensegment to
display second digit
        delay_ms(1);                            // call delay
        digit1=1;                                       // digit second off
```

```
}

void delay_ms(unsigned char itime)                    // delay function
{
        unsigned int i,j;
        for(i=0;i<itime;i++)
        for(j=0;j<1275;j++);
}
```

3. DAC to generate different wave forms(sine, square,trangular, ramp etc.) with variable frequuency and amplitude using variable pot.

```
#include<reg51.h>
#define wave_sel P2                                   //define P2 to wave_sel
#define dac_databus P1                                //define P1 to dac_databus
#include<math.h>                        // include math library

void delay_ms(unsigned int );                         // prototype
void sine_wave(void);
void square_wave(void);
void ramp_wave(void);
void trangular_wave(void);

void main(void)                                       // start program
{
        unsigned char temp=0;
        P3=0;

                        while(1)
                {

                temp = wave_sel;                              // selection of wave from the
keys
                P3 = temp ;

                temp &= 0xF0;                         // masking with F0 to
                P3 = temp ;
                temp= temp >>4;                       // shift right 4 time
                P3 = temp ;
                temp= ~temp ;                         // invert temp
                P3 = temp ;
                temp &= 0x0f;                         // mask with 0F

                P3 = temp ;

                switch(temp)                          //switch case for selecting of wave
form
                {
```

```
                        case(0):                        // 0 for sine wave
                        {


                                sine_wave();
                                break;
                        }

                        case(1):                         // 1 for RAMP
                        {
                                ramp_wave();
                                break;
                        }

                        case(2):                        // 2 for triangular
                        {
                                trangular_wave();
                                break;
                        }

                        case(4):                         // 4 for square wave
                        {
                                square_wave();
                                break;
                        }
                        case(8):
                        {
                                sine_wave();
                                break;
                        }

                }
                }

}


void ramp_wave(void)                                    // function for the rampwave
{
                unsigned char dac_data;
                while(1)                        //infinet loop
                {
                        for(dac_data=0;dac_data<255;dac_data++)     //increament data
from 0 to fffor ramp
                        {
                                dac_databus=dac_data;                           // assign
data
                        }
                }
 }
```

```
void sine_wave()                                    // sine wave function
{
        unsigned char i,j;
        unsigned char sample_point[ ]={0x80,0xc0,0xee,0xff,0xee,0xc0,0x80,0x40,0x11,
                                                    0x00,0x11,0x40,0x80} ;    //
array of data tobe sent to the DAC to ganarate sine wave
        while(1)
        {
                for(i=0;i<=11;i++)                          // select data from array
                {
                        dac_databus = sample_point[i];      // assign data from array to
DAC_data

                        for(j=0;j<=50;j++);                         // delay
                }
        }
}


void square_wave()                                  //square wave function
{
        while(1)
        {
                dac_databus = 0x00;                 // assign 00 to 0 volt
                delay_ms(100);                      //delay
                dac_databus = 0xff;                 // assign ff to 5 volt
                delay_ms(100);                      // delay
        }

}


void trangular_wave(void)                           // triangular wave function
{
        unsigned char dac_data;

        while(1)
        {
                for(dac_data=0;dac_data<255;dac_data++)      // incrament data from 00
to  ff
                {
                        dac_databus=dac_data;
                }

                for(dac_data=255;dac_data>0;dac_data--)      // decreament data from ff
to 00
                {
                        dac_databus=dac_data;
```

```
                }
        }



}

        void delay_ms(unsigned int itime)                       // delay function
{
        unsigned int i;
        for(i=0;i<itime;i++);

}
```

4. Stepper motor speed and direction control

```
#include<reg51.h>
#define stm_databus P0                          // define P0 to stm databus
sbit REV = P0^4;                                        // single bits
sbit FRW = P0^5;
sbit DCR = P0^6;
sbit INR = P0^7;

void port_initi(void);                          // prototyping
void stm_forward(unsigned int);
void stm_reverse(unsigned int);
void delay_ms(unsigned int value);

 void main(void)                                 // main function
{
        port_initi();                           // port initialisition
        FRW = 0;                                // intial motor rotate in forward
direction
        while(1)
        {
                if(FRW==0)                      //check switch to press
                {
                        stm_forward(500);       // function call for forwrd rotation
                }

                if(REV==0)
                {
                        stm_reverse(500);       // function call for reverse rotation
                }
         }
}
```

```
void stm_forward(unsigned int count )                    // forward rotation
{
        while(1)
        {
                stm_databus = 0x06;                 // pulses given to the stepper motor
           delay_ms(count);                                 //delay
           stm_databus = 0x0A;
           delay_ms(count);
           stm_databus = 0x09;
           delay_ms(count);
           stm_databus = 0x05;
           delay_ms(count-5);

                stm_databus =0XF5;
                if(REV == 0)
                {
                   stm_reverse(count);          // for revese direction pulses sequence will
reverse
                }

                if(DCR == 0)                  // decreament speed check key press
                {
                        count += 1;                 // increase counter which increse delay
beteween two pulses
                        if(count > 450)
                        {
                          count += 10;
                        }
                        if(count > 1000)
                        {
                          count += 100;
                        }
                        if(count > 5000)
                        {
                          count += 500;
                        }

                }
                if(INR==0)                           //increament speed check key press
                {
                        count -= 1;                  //decrease counter which decrease
delay between two pulses
                        if(count > 450)
                        {
                          count -= 10;
                        }
                        if(count > 1000)
                        {
                          count -= 100;
```

```
                }
                if(count > 5000)
                {
                  count -= 500;
                }
                if(count < 160)
                {
                  count = 160;
                }

        }


    }

}

void stm_reverse(unsigned int count)                    // reverse rotation
{

    while(1)
    {
        stm_databus= 0x05;
       delay_ms(count);
       stm_databus = 0x09;
       delay_ms(count);
       stm_databus = 0x0a;
       delay_ms(count);
       stm_databus = 0x06;
       delay_ms(count-5);

        stm_databus =0XF5;

        if(FRW ==0)
        {
            stm_forward(count);
        }

        if(DCR ==0)
        {
            count += 1;
            if(count>450)
            {
              count += 10;
            }
            if(count > 1000)
            {
              count += 100;
            }
            if(count > 5000)
```

```
                {
                  count += 500;
                }
        }
        if(INR ==0)
        {
        count -= 1;
                if(count >450)
                {
                  count -= 10;
                }
                if(count > 1000)
                {
                  count -= 100;
                }
                if(count > 5000)
                {
                  count -= 500;
                }
                if(count < 160)
                {
                  count = 160;
                }

        }
    }
}


void delay_ms(unsigned int value)                    //delay function
{
     unsigned int y;
    for(y=0;y<value;y++);
}
void port_initi(void)
{
    P0=0xff;
    P1=0xff;
    P2=0xff;
    P3=0xff;
}
```

5. Interface DC motor with 8051 microcontroller

```
#include<reg51.h>
sbit mtr_1 =P2^0;          //single bit declaration
sbit mtr_2 =P2^1;
```

```
sbit pwm_control =P2^3;
sbit key2_REV        = P2^4;
sbit key1_FRW        = P2^5;
sbit speed_dec       = P2^6;
sbit speed_inc       = P2^7;


void main(void)          // main function
{
        unsigned int count,value=500;
        while(1)
        {
                if(key1_FRW == 0)            // check key is pressed for forward direction
                {
                        mtr_1 =0;            // directional bits for forward
                        mtr_2 =1;

                 }

                 if(key2_REV == 0)           // check key is pressed for reverse direction
                {
                        mtr_1 =1;            // directional bits for revers direction
                        mtr_2 =0;

                 }

                 if(speed_dec ==0 )          // check key is pressed for decreament the
speed
                 {
                        value -= 1;           // value decreament to change delay in pwm
wave
                        if(value <= 15)
                        {
                                value = 16;
                        }
                 }

                 if(speed_inc ==0 )          // check key is press for increament the speed
                 {
                        value += 1;               // value increament to change delay in
pwm wave
                        if(value >= 1000)
                        {
                                value = 999;
                        }
                 }


                pwm_control=1;               // ganaration of pwm wave to increase or
decrease the speed
```

```
                for(count=0;count<= value ;count++);

                pwm_control=0;
                for(count=0;count<= 500;count++);


        }
        while(1);
}
```

Stepper Motor

```
#include<reg51.h>
#define stm_databus P0              // define P0 to stm databus
sbit REV = P0^4;                  // single bits
sbit FRW = P0^5;
sbit DCR = P0^6;
sbit INR = P0^7;

void port_initi(void);              // prototyping
void stm_forward(unsigned int);
void stm_reverse(unsigned int);
void delay_ms(unsigned int value);

void main(void)                     // main function
{
        port_initi();               // port initialisition
        FRW = 0;                    // intial motor rotate in forward direction
        while(1)
        {
                if(FRW==0)                //check switch to press
                {
                        stm_forward(500);          // function call for forwrd rotation
                }

                if(REV==0)
                {
                        stm_reverse(500);          // function call for reverse rotation
                }
```

```
        }
}

void stm_forward(unsigned int count )     // forward rotation
{
        while(1)
        {
           stm_databus = 0x06;              // pulses given to the stepper motor
           delay_ms(count);                 //delay
           stm_databus = 0x0A;
           delay_ms(count);
           stm_databus = 0x09;
           delay_ms(count);
           stm_databus = 0x05;
           delay_ms(count-5);

                stm_databus =0XF5;
                if(REV == 0)
                {
                    stm_reverse(count);             // for revese direction pulses sequence will
reverse
                }


        }

}

void stm_reverse(unsigned int count)              // reverse rotation
{

        while(1)
        {
                stm_databus= 0x05;
           delay_ms(count);
           stm_databus = 0x09;
           delay_ms(count);
           stm_databus = 0x0a;
           delay_ms(count);
           stm_databus = 0x06;
           delay_ms(count-5);

                stm_databus =0XF5;

                if(FRW ==0)
                {
                        stm_forward(count);
                }

        }
```

```
}


void delay_ms(unsigned int value)                    //delay function
{
          unsigned int y;
         for(y=0;y<value;y++);
}
void port_initi(void)
{
        P0=0xff;
        P1=0xff;
        P2=0xff;
        P3=0xff;
}
```

4. Write a C Program to Read the Keypad and send the result to the First Serial Port

```
//P1.0 - P1.3 Connected to Rows
//P2.0 - P2.3 Connected to Columns
// Configure the Serial Port for 9600 Baud, 8 Bit , and 1 stop bit.


#include <reg52.H>

#define         COL P2                  //Define port for easier Reading
#define         ROW P1

void msdelay(unsigned int value);
void sertx      (unsigned char);
//void sevenseg         (unsigned char );

unsigned char keypad[4][4]=          {'0','1','2','3',
                                    '4','5','6','7',
                                    '8','9','A','B',
                                    'C','D','E','F'};
void main()
{
        unsigned char colloc,rowloc;
TMOD =0x20;         //Timer 1,MOde 2
TH1=-3;             //9600 Baud
SCON=0x50; //8bit, 1 stop bit
TR1=1;              //Start Timer 1

//KEYBOARD ROUTINE. This sends the ASCII
//Code for Pressed Key to the Serial Port

COL   =0XFF;                        //make p2 an input port
while(1)
```

```
{
        do
                {
                        ROW  =0x00;          //ground all rows at once
                        colloc  =COL;                //read the coloumn
                        colloc &=0x0F;               //mask used bit
                        }while(colloc !=0x0F);       //check until all key realeased
         do
         {
                do
                {
                        msdelay(20);         //call delay
                        colloc  =COL;                //see if key is pressed
                        colloc &=0x0F;               //mask unused bits
                        }while(colloc ==0x0F);  //Keep checking for key pressed

                        msdelay(20);         //call delay
                        colloc  =COL;                //see if key is pressed
                        colloc &=0x0F;               //mask unused bits
                        }while(colloc ==0x0F);//check for key pressed

                        while(1)
                        {
                        ROW  =0xfe;          //ground row 0
                        colloc  =COL;                //read coloumns
                        colloc &=0x0F;               //mask unused bits
                        if(colloc !=0x0F)    //coloumn detected
                        {
                                rowloc=0;    //save row
                                break;       //exit while loop
                        }

                        ROW  =0xfd;          //ground row 1
                        colloc  =COL;                //read coloumn
                        colloc &=0x0F;               //mask unused bits
                        if(colloc !=0x0F)    //coloumn detected
                        {
                                rowloc=1;    //save row
                                break;       //exit while loop
                        }

                        ROW  =0xfb;          //ground row 2
                        colloc  =COL;                //read coloumn
                        colloc &=0x0F;               //mask unused bits
                        if(colloc !=0x0F)    //coloumn detected
                        {
                                rowloc=2;    //save row
                                break;       //exit while loop
                        }
```

```
                    ROW  =0XF7;                //ground row 2
                    colloc  =COL;              //read coloumn
                    colloc &=0x0F;             //mask unused bits
                    rowloc=3;             //save row
                    break;                //exit while loop
                    }

//CHECK COLOUMN AND SEND DATA TO SERIAL PORT

if(colloc ==0x0E)
sertx(keypad[rowloc][0]);

else if(colloc ==0x0D)
sertx(keypad[rowloc][1]);

else if(colloc ==0x0B)
sertx(keypad[rowloc][2]);

else

sertx(keypad[rowloc][3]);

}
}


void sertx(unsigned char x)
{
        SBUF=x;                         //Place Value in Buffer
        while(TI==0);               //Wait Untill Transmitted
        TI=0;                       //Clear Flag
}

void msdelay(unsigned int value)
{
        unsigned int x,y;
        for (x=0;x<1275;x++)
        for (y=0;y<value;y++);
}
```

Example 2

In the next example, you can Toggle the LEDs ON and OFF (Blinking LEDs) that are connected to PORT1 of the 8051 Microcontroller.

```
ORG 00H                 ; Assembly Starts from 0000H.
        ; Main Program
        START:      MOV P1, #0XFF           ; Move 11111111 to PORT1.
                    CALL WAIT       ; Call WAIT
                    MOV A, P1    ; Move P1 value to ACC
```

```
                CPL A                   ; Complement ACC
                MOV P1, A    ; Move ACC value to P1
                CALL WAIT           ; Call WAIT
                SJMP START          ; Jump to START
        WAIT:           MOV R2, #10         ; Load Register R2 with 10 (0x0A)
        WAIT1:  MOV R3, #200                ; Load Register R3 with 10 (0xC8)
        WAIT2:  MOV R4, #200                ; Load Register R4 with 10 (0xC8)
                DJNZ R4, $    ; Decrement R4 till it is 0. Stay there if not 0.
                DJNZ R3, WAIT2    ; Decrement R3 till it is 0. Jump to WAIT2 if
not 0.
                DJNZ R2, WAIT1    ; Decrement R2 till it is 0. Jump to WAIT1 if
not 0.
                RET             ; Retu rn to Main Program
        END                             ; End Assembly
```

Displaying Hindi Alphabets on LCD Using 8051

```
#include<reg51.h>
        #define lcd P1
        sbit rs=P3^0;
        sbit rw=P3^1;
        sbit e=P3^2;

        void delay (int);
        void cmd (char);
        void display (char);
        void custom (void);
        void init (void);
        unsigned char hindi_char[]={0x1D,0x15,0x1D,0x05,0x1F,0x15,0x09,0x01,
                                    0x1F,0x08,0x08,0x08,0x08,0x08,0x08,0x08,
                                    0x1F,0x04,0x04,0x1C,0x10,0x08,0x04,0x02,
                                    0x1F,0x02,0x02,0x0E,0x12,0x12,0x12,0x00,
                                    0x00,0x0A,0x15,0x11,0x11,0x0A,0x04,0x00};


        void delay (int d)
        {
                unsigned char i;
                for(;d>0;d--)
                {
                        for(i=250;i>0;i--);
                        for(i=248;i>0;i--);
                }
        }
        void cmd (char c)
        {
                lcd=c;
```

```
        rs=0;
        rw=0;
        e=1;
        delay(5);
        e=0;
}
void display (char c)
{
        lcd=c;
        rs=1;
        rw=0;
        e=1;
        delay(5);
        e=0;
}
void custom (void)
{
        int k;
        cmd(0x40);
        for(k=0;k<40;k++)
        display(hindi_char[k]);
        cmd(0x80);
}
void init (void)
{
        cmd(0x38);
        cmd(0x0c);
        cmd(0x01);
        cmd(0x80);
}
void main()
{
        init();
        custom();
        cmd(0x84);
        display('I');
  display(' ');
        display(4);
        display(' ');
        display(0);
        display(1);
        display(2);
        display(3);
while(1);
}
```