QMP 7.1 D/F

Channabasaveshwara Institute of Technology

(An ISO 9001:2008 Certified Institution)

NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

Department of Electronics & Communication Engineering

MICROPROCESSOR LAB

Sub Code: 10ECL68

B.E - VI Semester

Lab Manual 2016-17

Name : _____

USN : _____

Batch : _____ Section : _____

Channabasaveshwara Institute of Technology

(An ISO 9001:2008 Certified Institution)

NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

Department of Electronics & Communication Engineering

MICROPROCESSOR LAB

Version 1.0

February 2017

Prepared by:

Ms. Vidhyarani K R
Mr. Mohamed Shafiulla S
Mrs. Lakshmidevi P
Mrs. Ankitha B M

Reviewed by:

Mr. Nagaraja P
Associate professor

Approved by:

Dr. Suresh D S
Professor & Head,
Dept. of ECE

SYLLABUS

MICROPROCESSOR LAB

Subject Code: **10ECL68** IA Marks : 25
No. of Practical Hrs/Week: 03 Exam Hours: 03
Total no. of Practical Hrs.: 42 Exam Marks : 50

I) **Programs involving**

1) Data transfer instructions like:
   - i] Byte and word data transfer in different addressing modes.
   - ii] Block move (with and without overlap)
   - iii] Block interchange

2) Arithmetic & logical operations like:
   - i] Addition and Subtraction of multi precision nos.
   - ii] Multiplication and Division of signed and unsigned Hexadecimal nos.
   - iii] ASCII adjustment instructions
   - iv] Code conversions
   - v] Arithmetic programs to find square cube, LCM, GCD, factorial

3) Bit manipulation instructions like checking:
   - i] Whether given data is positive or negative
   - ii] Whether given data is odd or even
   - iii] Logical 1's and 0's in a given data
   - iv] 2 out 5 code
   - v] Bit wise and nibble wise palindrome

4) Branch/Loop instructions like:
   - i] Arrays: addition/subtraction of N nos.
   - ii] Finding largest and smallest nos.
   - iii] Ascending and descending order
   - iv] Near and Far Conditional and Unconditional jumps, Calls and Returns

5) Programs on String manipulation like string transfer, string reversing, searching for a string, etc.

6) Programs involving Software interrupts

7) Programs to use DOS interrupt INT 21h Function calls for

8) Reading a Character from keyboard, Buffered Keyboard input, Display of String on console

II) **Experiments on interfacing 8086 with the following interfacing modules through DIO card**

a) Matrix keyboard interfacing
b) Seven segment display interface
c) Logical controller interface
d) Stepper motor interface

III) Other Interfacing Programs

a) Interfacing a printer to an X86 microcomputer
b) PC to PC Communication

MICROPROCESSOR LAB Course Objectives and Outcomes

OBJECTIVES

The objectives of this course are;

- To make the student understand and have hands on-expertise of assembly language programming.
- To design a hardware application by interfacing 8255 with 8086 microprocessor.

OUTCOMES

At the end of the Lab Course Student is able to:

- Proficiently use DOS assemblers like MASM, TASM
- Use the knowledge of the 8086 instruction set and utilize it in programming
- Perform Logical, Arithmetic, and Rotate/Shift operations on Data
- Understand and implement delay generation using 8086 instructions
- Understand different interfacing concepts and use of PPI (Add-on cards)
- Implement programming module of Keyboard, Stepper motor, Seven Segment Display and PC to PC Communication to work with x86 processor.

Instructions to the student

1. Come prepared to the lab with relevant theory about the experiment you are conducting.

2. Each experiment will be evaluated for 25 marks. More weightage will be given for preparation and understanding.

3. Handle the desktop system and interfacing boards with care

4. Do not delete or change the system settings and files.

5. For any missing items, penalty will be imposed on the respective batch.

6. Maintain professional attitude and discipline during lab sessions.

CONTENTS

PART B
Interfacing programs

Cycle - 2

Additional Programs

INTRODUCTION

MASM: [Macro Assembler]

The Microsoft Macroassembler (MASM) is a program that can be used to assemble source files into object modules. The assembler converts the contents of the source input file for example: **PROG.ASM** file into two output files called **PROG.OBJ** and **PROG.LST**. The file **PROG.OBJ** contains the object code module. The **PROG.LST** file provides additional information useful for debugging the application program.

Object module **PROG.OBJ** can be linked to other object modules with **LINK** program. It produces a run module in file **PROG.EXE** and a map file called **PROG.MAP** as outputs. Map file **PROG.MAP** is supplied as support for the debugging operation by providing additional information such as where the program will be located, when loaded into the microcomputers memory.

MASM COMMANDS:

1. -Go to Start<Run<command or cmd< then Press enterkey
2. -Type **cd..** (Enter)
3. -Type **cd..** (Enter)

C :/> cd foldername

C:/foldername>edit filename.asm

After this command is executed in the command prompt, an editor window will open. Program should be typed in this window and saved. The program structure is given below.

Structure of Program:

```
.model small/tiny/medium/large

.Stack <some number>

.data
        ; Initialization of Data which is used in program.
        ; Variable declaration goes here.
.code
        ; Initialization of data segment,
        ; Program logic goes here.

End
```

To run the program in MASM 5.0 following steps have to be executed:

C:/foldername>masm filename.asm *(Press enter key thrice or type ;)*

After this command is executed, if there are no syntax errors, the assembler will generate an object module.

C:/foldername>link filename.obj *(Press enter key thrice or type ;)*

The generated object files should be linked together. This is done by executing the above link command which will generate an .EXE file.

C:/foldername>debug filename.exe

After generating .EXE file by the assembler, it's time to check the output. For this the above command is executed. The execution of the program can be done in different ways as shown below:

__ g ; complete execution of program in single step.

__ t ; Stepwise execution.

__d ds: starting address or Ending address **;** To see data in memory locations

__p **;** Used to execute Interrupt or procedure during stepwise execution of program

__ q **;** To quit the execution.

PART A

1. a) Program for data transfer using different addressing modes

```
.model small
.data
        Num dw 4321h
.code
        Mov ax,@data        ; Initialize the data segment
        Mov ds, ax

        Mov ax, 1234h       ; immediate addressing
        Mov bx, ax          ; register addressing
        Mov ax, num         ; direct addressing
        Mov si, 1000h
        Mov al, [si]        ; indirect addressing
        Mov bl, [si+100h]   ; relative addressing
        Mov bx, 1000h
        Mov ax, [si+bx]     ; base index addressing
        Mov cx, [si+bx+100h]; relative base index addressing
        Int 3               ; Terminate the program
End
```

1. b) Program to move data from source to destination using indirect addressing mode *(Block Move without overlap)*

```
.model small
.data
        d1 db 0ah,0bh,0ch,0dh,0eh
        d2 db 10 dup(0)
.code
        Mov ax,@data        ; Initialize the data segment
        Mov ds, ax

        Lea si, d1          ; Load offset address of d1 in si
        Lea di, d2          ; Load offset address of d2 in di
        Mov cx, 05          ; load cx with count

Up:     Mov al, [si]        ; Move the 1st element of d1 to al
        Mov [di], al        ; Move to d2
        Inc si              ; Increment si
        Inc di              ; Increment di
        Dec cx              ; Decrement the counter
        Jnz Up              ; Repeat till cx becomes zero

        Int 3               ; Terminate the program
       Align 16             ; DS starts from page boundary

End
```

NOTE: 1) The function of **Mov si, offset src** is same as **Lea si, src**.
Therefore in the above program **Lea si, src** and **Lea di, dst** can be used.

2) When we use **loop** instruction the counter value should be in CX. This instruction Decrements CX, checks for CX = 0, if so, it loops backs to the label specified with this instruction.

1. c) Program to move a block of data from source to destination
(With overlap in either direction)

```
.model small
.stack  100
.data
      len       equ  0ah
      src       equ  0024h
      dst       equ  002ah
      nums      db   01h,02h,03h,04h,05h,06h,07h,08h,09h,0ah
.code
 Start: Mov ax, @data        ; Initialization of data and extra data segment.
        Mov ds, ax
        Mov es, ax
        Mov si, 00           ; move 00 to si
        Mov cx, len          ; copy the value of len to cx reg.
   up:  Mov dl, nums[si]     ;loading the data starting
        Mov src [si], dl     ; from source address 'src'
        Inc si               ; Increment si by 1
        loop up
        Mov si, src          ; initialization of source
        Mov di, dst          ; & destination blocks along
        Mov cx, len          ; with their length
        cmp si, di           ; compare si with di inorder to Decide whether
        jc btmtrf            ; src addr is > Dst addr
        cld                  ; clear DF for incrementing si and di by 1.
  trf:  rep Movsb
        jmp ovr              ; unconditional jump to ovr label
btmtrf: add si, len          ;bottom transfer
        Dec si               ; decrement si by 1
        add di,len           ; add di with len and store the result in di.
        Dec di               ; decrement di by 1
        Std                  ; set direction flag
        jmp trf              ; unconditional jump to trf label.
```

```
ovr:  Int 3          ; Terminate the program
      Align 16
      End start
```

Result:

1. d) Program to interchange two blocks of data

```
.model small
.stack 20
.data
        src db 31h,32h,33h,34h,35h,36h,37h,38h
        dst db 41h,42h,43h,44h,45h,46h,47h,48h
        Count dw 0008h
.code
Start:  Mov ax,@data        ; Initialization of data and extra segment.
        Mov ds, ax
        Mov es, ax

        Mov cx, count       ; Initialize cx reg with count
        Mov bx, 0000h       ; store bx with 00

repeat:
        Mov al, src[bx]     ; move the contents at location o to al
        xchg al, dst[bx]    ; Excahnge the contents of al with dst
        Mov src [bx], al    ; copy the contents of al to src
        Inc bx              ; Increment bx by 1
        loop repeat         ; go to the label repeat if cx != 0

        Int 3               ; Terminate the program
        Align 16

End Start
```

Result:

2. a) Program to add two multi-precision numbers

```
.model small
.data
    num1 db 3Dh,62h,48h,0A3h     ;lower byte first (num1-A348623Dh)
    num2 db 8Ch,0B2h,76h,0FDh    ;lower byte first (num2-FD76B28Ch)
    len equ ($-num2)
    res db len+1 dup(?)
.code
start: Mov ax,@data
    Mov ds, ax          ; initializes DS
    Mov cx, len         ; sets counter for 4-bit byte addition
    Lea si, num1        ; points SI to 1st Multibyte no num1
    Lea di, num2        ; points DI to 2nd Multibyte no num2
    Lea bx, res         ; points BX to the result memory location
    clc                 ; clears carry flag CF
back : Mov al,[si]
    Adc al, [di]        ;adds two Multibyte no.s byte by byte with carry
    Mov [bx], al
    Inc si
    Inc di
    Inc bx
    loop back           ; go to label back if cx != 0
    Jnc zero            ; checks for CF = 0
    Inc cl
Zero: Mov [bx], cl      ; stores carry
    Int 3               ; Terminate the program
    Align 16
 End start
```

Result:

Note: For subtraction of Multi-precision numbers, replace the instruction ADC with SBB.

2. b) Program to Multiply unsigned 16-bit numbers

```
.model small
.data
        N1 dw 0ffffh
        N2 dw 0ffffh
        res dw 5 dup(0)
.code
Start: Mov ax,@data          ; initializes DS
       Mov ds, ax

        Mov ax, N1           ; copy ax with first number
       Mov cx, N2            ; copy cx with second number
       Mul cx                ; multiply ax with cx

       Mov res, ax           ; store the result in ax and dx
       Mov res+2, dx
        Int 3                ; Terminate the program
       Align 16

    End start
```

Result:

2. c) Program to multiply signed 16-bit numbers

```
.model small
.stack 100
.data
        num1 dw -1h
        num2 dw 0032h
        res dw 2 dup(0)
.code
Start: Mov ax,@data
       Mov ds, ax
       Mov ax, num1
       Mov dx, 0000h        ; initialize dx with zeros
       Mul num2
       Mov res, ax          ; take the higher order word of product from dx
       Mov res+2, dx          and lower order word from ax

       Int 3                ; Terminate the program
       Align 16
End start
```

Result:

2. d) Program to Divide 32-bit unsigned number by 16-bit number

```
.model small
.data
 dvd dd 15752510h
 dvr dw 0ffffh
 qut dw ?
 rem dw ?
.code
 Start: Mov ax,@data            ; Initialization of data segment.
        Mov ds, ax
        Mov si, offset dvd      ; copy the offset of dividend to si.
        Mov ax, word ptr [si]   ; copy the dividend to ax and dx
        Mov dx, word ptr [si+2]
        Mov cx, dvr             ; copy the divisor to cx
        Div cx
        Mov qut, ax             ; copy the ax to quoitent
        Mov rem, dx             ; copy the dx to reminder
        Int 3                   ; Terminate the program
        Align 16
End start
```

Result:

3. a) Program to illustrate use of AAA instruction (ASCII addition)

```
.model small
.data
 Read macro            ; macro to read ASCII value
        Mov ah, 01h
         Int 21h
Endm
 Write macro X
        Mov dl, X      ; macro to display ASCII value
        Mov ah, 02h
        Int 21h
 Endm
.code
Start: Mov ax,@data
        Mov ds, ax              ; Initialization of data segment
        read                    ; read 1st no.
        Mov bl, al              ; copy al to bl
        Write '+'
        read                    ; read 2nd no.
        Mov cl, al
        Write '='
        Mov al, cl
         add al,bl              ;result in hex
         mov ah, 0
        aaa                     ;converts to unpacked BCD
        add ax,3030h
        push ax
        write ah                ;displays higher nibble
        pop ax
        write al                ;displays higher nibble
        Int 3                   ; Terminate the program
    End start
```

3. b) Program to illustrate use of AAS instruction (ASCII subtraction)

```
.model small
.data
read macro                ; macro for read ascii value from key board
        Mov ah, 01h
        Int 21h
Endm
        write macro X     ; macro to display ascii value on screen
        Mov dl, X
        Mov ah, 02h
        Int 21h
Endm
.code
start:  Mov ax,@data
        Mov ds, ax        ; initialize ds
        read              ; read first number
        Mov cl, al
        Write '-'
        read              ; read second number
        Mov bl, al
        Write '='
        Cmp cl, bl
        Jnc sb
        Write '-'
        Mov al, cl
        Xchg al, bl
        Jmp sb1
 Sb:    Mov al, cl
Sb1:    Sub al, bl
        aas
        or al, 30h
        write al
        Int 3h
 End start
```

3. c) Program to illustrate use of AAM instruction (ASCII Multiplication)

```
.model small
.data
Read macro                ; macro for read ascii value from key board
Mov ah, 01h
Int 21h
Endm
Write macro x
Mov dl, x                 ; macro to display ascii value on screen
Mov ah, 02h
Int 21h
Endm
.code
start:  Mov ax,@data
        Mov ds,ax         ; initialize ds
        read              ; read first number
        and al,0fh
        Mov bl, al        ; store the read number in bl register
        Write '*'
        read              ; read second number
        and al,0fh
        mov cl, al
        Write '='
        Mov al, cl
        Mul bl            ; multiple first and second numbers.
        aam               ; unpacked bcd result
        or ax,3030h       ; result in ascii
        push ax
        Write ah
        Pop ax
        Write al
        Int 3h            ; terminate the program
End start
```

4. a) Program to convert binary number to BCD number

```
.model small
.data
        num db 0FFh
        res db 03 dup(0)
.code
        Mov ax,@data
        Mov ds, ax              ; Initialize data segment

        Mov al, num             ; Move the binary number to al
        Mov ah, 00h             ; clear ah
        Mov bl, 64h
        Div bl                  ; Divide ax by 64h

        Lea si, res
        Mov [si], al            ; Move the quotient to [si]
        Mov al, ah              ; Move the remainder to al
        Mov ah, 0h              ; cLear ah
        Mov bl, 0ah
        Div bl                  ; Divide ax by bl

        Mov [si+1], al          ; Move the quotient to [si+1]
        Mov [si+2], ah          ; Move the remainder to [si+2]
        Int 3                   ; Terminate the program
        End
```

Result:

Input: 0FFh
Output: 02 05 05 stored in locations [si, si+1, si+2]

Input: 063h
Output: 09 09 stored in locations [si+1, si+2]

4. b) Program to convert BCD number to binary number

```
.model small
.stack 20
.data
        bcd dw 1234h
        bin dw 0
.code
Start: Mov ax,@data                ; initializes DS
        Mov ds, ax
        Mov es, ax
        Mov bx, 0001h
        call bcd2bin
         Mov bx, 000ah
        call bcd2bin
        Mov bx, 0064h
        call bcd2bin
        Mov bx,03e8h
        call bcd2bin
        Int 3                      ; Terminate the program

        bcd2bin proc near
        Mov ax, bcd
        and ax,000fh
        Mul bx
        add bin,ax
        Mov cl, 04
        ror bcd,cl
        ret
        bcd2bin Endp
End start
```

Result:

5. a) Program to find square and cube of a 16-bit number

```
.model small
.data
        num dw 0ffeeh
        res dw 10 dup()
.code
start: Mov ax, @data
       Mov ds, ax                ; initialize ds
       Mov si, offset num
       Lea di, res               ; poInt di to res location
       Mov ax,[si]               ; get the number
       Mul ax                    ; square of a given no.
       Mov [di], ax
       Mov [di+2], dx            ; store the square in memory

       Mov ax, [si]              ; to find cube of a given no.
       Mov cx, [di]
       Mul cx
       Mov [di+4], ax
       Mov bx, dx

       Mov ax, [si]
       Mov cx, [di+2]
       Mul cx
       add ax, bx
       adc dx,0000
       Mov [di+6], ax
       Mov [di+8], dx
       Int 3                     ; Terminate the program
End start
```

Result:

5. b) Program to find LCM of two 8-bit numbers

```
.model small
.data
    nums dw 0010,0048
    lcm dw 2 dup(?)
.code
start: Mov ax,@data
    Mov ds, ax
    Mov ax, nums
    Mov cx, nums+2
    Mov dx, 00h
back: push ax
    push dx
    Div cx          ; Divide one no. by another
    cmp dx,00h      ; compare the remainder with zero
    je lcm1
    pop dx
    pop ax
    add ax,nums     ; if the remainder is not zero, take the next Multiple of
                      the DividEnd and again try to Divide it by the Divisor
                      till the remainder becomes zero
    Jnc skip
    Inc dx
skip: jmp back
lcm1:pop lcm+2      ; when the remainder is zero,
    pop lcm         ; the DividEnd value is the lcm
    Int 3           ; Terminate the program
End start
```

Result:

5. c) Program to find GCD of two 8-bit numbers

```
.model small
.data
        Num dw 1bh, 09h
        Gcd dw ?
.code
        Mov ax,@data
        Mov ds, ax          ; Initiliaze data segment
        Mov ax, num         ; Move the 1st number to ax
        Mov bx, num+2       ; and 2nd number to bx
Again: cmp ax, bx           ; check whether both numbers are same or not.
        Je exit             ; if equal the number is gcd
        Jb down             ; if first number is < second go to label down
Divaxbx: Mov dx, 0          ; else divid the larger number with smaller number.
         Div bx
         Cmp dx, 0          ; check the remainder is 0 to stop the division process.
         Je exit
         Mov ax, dx
         Jmp again
Down: xchg ax, bx
        Jmp Divaxbx
Exit:   Mov gcd, bx
        Mov gcd+2, ax
        Int 3

End
```

Result:

5. d) Program to find the factorial of a given number

```
.model small
.stack    100h
.data
          n1      dw 3
          nfact   dw ?
.code
start:    Mov     ax, @data
          Mov     ds, ax
          Mov     ax, 01
          Mov     dx, 00
          Mov     cx, n1
; CALL THE PROCEDURE FACTN
          call    factn
          Mov     nfact, ax
          Mov     nfact+2, dx
          Int     3
factn     proc
          cmp     cx,00
          je      exit
          cmp     cx, 01
          je      exit
          push    cx
          Dec     cx
          call    factn
          pop     cx
          Mul     cx
exit:     ret
factn     Endp
          End     start
```

Result:

6. a) Program to check whether given data is positive or negative

```
.model small
.data
        nums    dw     2345h
        msg1 db 0ah,0dh,"the data is positive $"
        msg2 db 0ah,0dh,"the data is negative $"
.code
start:  Mov ax,@data
        Mov ds, ax

        Mov ax, nums                ; get the number in ax
        rol ax,1                    ; rotate left to check the MSB
        jc neg                      ; if CF = 1, number is negative.
                                    ; else positive
        Mov dx, offset msg1
        Mov ah, 09h
        Int 21h
        Jmp exit

neg:    Mov dx,offset msg2
        Mov ah, 09h
        Int 21h

exit:   Int 3                       ; Terminate the program
End start
```

Result:

6. b) Program to check whether given number is Odd or Even

```
.model small
.data
        nums    dw      123fh
        msg1 db 0ah,0dh," given number is even $"
        msg2 db 0ah,0dh," given number is odd $"

.code
start:  Mov ax,@data
        Mov ds, ax

        Mov ax, nums            ; get the number in ax
        ror ax,1                ; rotate right to check theLSB
        jc odd                  ; check if CF = 1, then the number is odd
                                ; else even
        Mov dx, offset msg1     ; Code to display a message
        Mov ah, 09h
        Int 21h
        jmp exit

odd:    Mov dx, offset msg2     ; code to display a message
        Mov ah, 09h
        Int 21h

exit:   Int 3                   ; Terminate the program
End start
```

Result:

6. c) Program to count logical 1's and 0's in a given data

```
.model small
.data
        nums    dw      00fah
        len     equ     16
        zero    db      01 dup(0)
        one     db      01 dup(0)
.code
start:  Mov ax,@data
        Mov ds, ax
        Mov bx, 00
        Mov cx, len
        Mov ax, nums

rpt:    rol ax,1
        jc one
        Inc zero
        jmp ovr

 one: Inc one
 ovr: loop rpt

        Int 3                   ; Terminate the program
        Align 16
        End start
```

Result:

6. d) Program to check whether the given number is 2 out of 5 code or not

```
.model small
.data
        num db 18h
        cnt1 equ 03h
        cnt2 equ 05h
        res db 4 dup(?)
.code
start: Mov ax, @data
        Mov ds,ax
        Mov al,num
        Mov cx,cnt1
again: rol al,01h
        jc no
        loop again
        Mov cx,cnt2
        Mov bl, 00h
back:  rol al,01h
        Jnc jpnxt
        Inc bl

jpnxt: loop back
        cmp bl,02h
        Jnz no
        Mov res,'Y'
        Mov res+1,'E'
        Mov res+2,'S'
        jmp ovr
no:    Mov res,'N'
       Mov res+1,'O'
ovr:   Int 3                    ; Terminate the program
    End start
```

Result:

7. a) Program to check whether the given 8-bit data is bit wise palindrome or not

```
.model small
.stack  100
.data
        pali db 0a6h
        msg1 db 0ah,0dh," palindrome $"
        msg2 db 0ah,0dh," not a palindrome $"
.code
start : Mov ax,@data
      Mov ds,ax
      Mov al,pali
      Mov bl,al
      and al,81h
      Jnp no
      Mov al,bl
      and al,42h
      Jnp no
      Mov al,bl
      and al,24h
      Jnp no
      Mov al,bl
      and al,18h
      Jnp no
        Mov dx,offset msg1
        Mov ah,09h
        Int 21h
        jmp exit
 no:  Mov dx, offset msg2
      Mov ah, 09h
      Int 21h
exit: Int 3                    ; Terminate the program
End start
```

Result:

7. b) Program to check whether the given 8-bit data is nibble wise palindrome or not

```
.model small
.data
        num db 84h
        msg1 db 0ah,0dh,"it is a palindrome$"
        msg2 db 0ah,0dh,"it is not a palindrome$"
.code
        Mov ax,@data            ; Initialize data segment
        Mov ds,ax

        Mov al, num             ; Move number to ax
        Mov cl, 04              ; Move 04 to cl
        Mov bl, al              ; Move ax to bx
        Clc                     ; Clear carry
  up:   ror bl,01               ; Rotate right bl once, through carry
        Dec cl                  ; Decrement cl
        Jnz up                  ; Repeat the loop if cl!=0,
        cmp al,bl               ;if cl=0, compare bh with bl
        jz pali                 ;If bh=bl, jump to label pali
        Mov ah, 09h             ;Display _it as a not palindrome.
        Lea dx, msg2
        Int 21h
        jmp End1                ;Jump to label End1
 pali:  Mov ah,09h              ;Display _it is a palindrome.
        Lea dx, msg1
        Int 21h
 End1:  Int 3                   ; Terminate the program
  End start
```

Result:

NOTE : The data should be string of hex numbers and data should not be terminated with ‘h’.

8. a) ALP to add 'n' 16 bit numbers stored in consecutive memory locations

```
.model small
.data
        nums dw 1234h,2345h,0abcdh,0deffh
        len equ($-nums)/2
        rst dw 2 dup(?)
.code
start: Mov ax,@data
        Mov ds, ax

        Mov cx, len
        Mov ax, 00
        Mov si, 00
        Mov dx, 00

 rpt: add ax,nums[si]
        Jnc skip
        Inc dx

skip: Inc si
        Inc si
        loop rpt
        Mov rst, ax
        Mov rst+2, dx

        Int 3                   ; Terminate the program
End start
```

Result:

8. b) Program to find smallest/largest number in a given array

```
.model small
.data
        nums dw 2222h,5555h,3333h,0aaaah
        len equ ($-nums)/2
        res dw ?
.code
start:  Mov ax,@data
        Mov ds, ax

        Mov si, offset nums
        Mov cx, (len-1)
        Mov ax, [si]

back :  Inc si
        Inc si
        cmp ax,[si]
        Jnc skip
        Mov ax,[si]

skip :  loop back
        Mov res, ax

        Int 3           ; Terminate the program
        Align 16
End start
```

Result:

8. c) Program to sort given numbers in ascending /descending order

```
.model small
.data
        arr1    db 5h, 89h, 3h, 56h, 1h
        len1    equ $-arr1
        arr2    db 29h, 12h, 45h, 89h, 34h
        len2    equ $-arr2
.code
start:  Mov     ax, @data
        Mov     ds, ax

    ; Ascending Sort
        Mov     ch, len1-1          ;no of itterations
agn1:   Mov     cl, ch              ;no of comparisions
        Mov     si, offset arr1
rept1:  Mov     al, [si]
        Inc     si
        cmp     al, [si]
        jbe     next1
        xchg    al, [si]
        Mov     [si-1], al
next1:  Dec     cl
        Jnz     rept1
        Dec     ch
        Jnz     agn1

    ; Descending Sort
        Mov     ch, len2-1          ; no of itterations
agn2:   Mov     cl, ch              ; no of comparisions
        Mov     si, offset arr2
rept2:  Mov     al, [si]
        Inc     si
        cmp     al, [si]
        jae     next2
```

```
        xchg    al, [si]
        Mov     [si-1], al
next2:  Dec     cl
        Jnz     rept2
        Dec     ch
        Jnz     agn2

        Int     3
        End     start
```

Result:

9**.** a) Program to move a string from source to destination

```
.model small
.stack 50
.data
        src db 'believe in yourself'
        n equ ($-src)
        space db 5 dup( )
        dst db n dup(0)
.code
start: Mov ax,@data
        Mov ds, ax
        Mov es, ax

        Mov cx,n
        Lea si,src
        Lea di,dst
        rep Movsb

        Int 3; Terminate the program
        Align 16
End start
```

Result:

9. b) Program to reverse a given string

```
.model small
.data
        str1 db 'sahyadri'
        len equ ($-str1)
        space db 5 dup( )
        str2 db len dup( )
.code
start: Mov ax,@data
        Mov ds,ax

        Lea si, str1
        add si, len-1
        Lea di, str2
        Mov cx, len

 back: Mov al,[si]
        Mov [di], al
        Dec si
        Inc di
        loop back

        Int 3                ; Terminate the program
        Align 16
 End start
```

Result:

9. c) Program to search a character in a given string

```
.model small
.data
        str db 'tumkur'
        len equ ($-str)
        char db 'm'
         msg1 db 0ah, 0dh," character found $"
         msg2 db 0ah, 0dh," character not found $"
.code
start: Mov ax,@data
        Mov ds, ax
        Mov es, ax

        Mov cx, len
        Lea di, str
        Mov al, char
        cld
        repne scasb

        je found
         Mov dx, offset msg2
         Mov ah, 09h
         Int 21h
         jmp exit
found: Mov dx,offset msg1
         Mov ah, 09h
         Int 21h
   exit:  Int 3                    ; Terminate the program
   End start
```

Result:

10. a) Program to display a character/ string on console

```
.model small
.data
    disp db 'Dos interrupt function 09h – to display a string.$'
.code
start: Mov ax,@data
    Mov ds, ax
    Mov dx, offset disp
    Mov ah, 09h
    Int 21h
    Int 3; Terminate the program
    End start
```

Result:

10. b) Program to read a character from the keyboard

```
.model small
.data
    msg db 'enter a key from keyboard:$'
.code
start: Mov ax,@data
    Mov ds, ax
    Mov dx, offset msg
    Mov ah, 09h
    Int 21h
    Mov ah, 01h
    Int 21h
    Int 3              ; Terminate the program
  End start
```

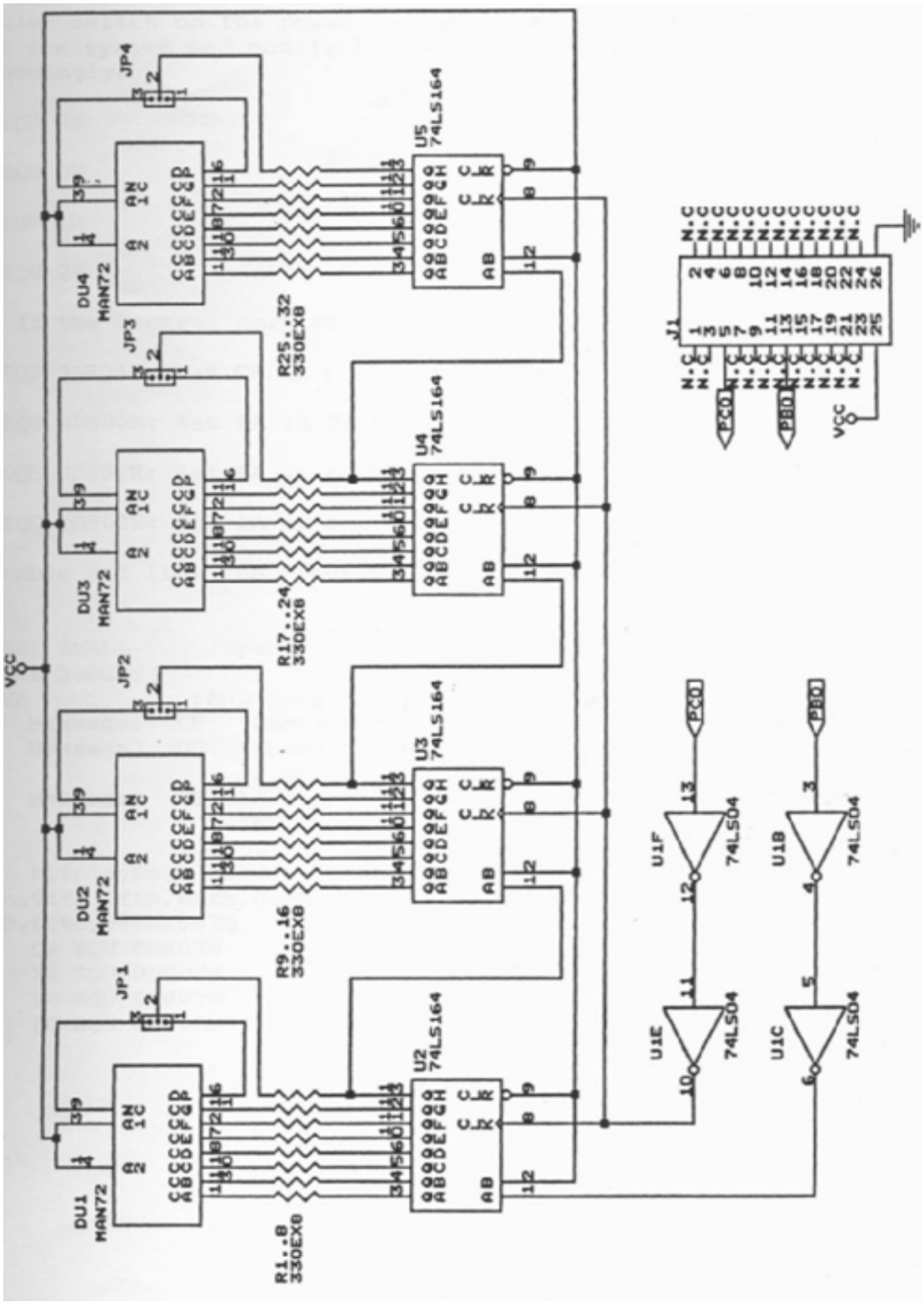
Result:

10. c) Program for buffered keyboard input

```
.model small
.data
        buff db 10 dup(20h)
.code
start: Mov ax,@data
        Mov ds, ax

        Mov dx, offset buff
        Mov ah, 0ah
        Int 21h

        Int 3                   ; Terminate the program
  End start
```

Result:

Circuit diagram of interfacing device for 4 Seven Segment Displays



PART B

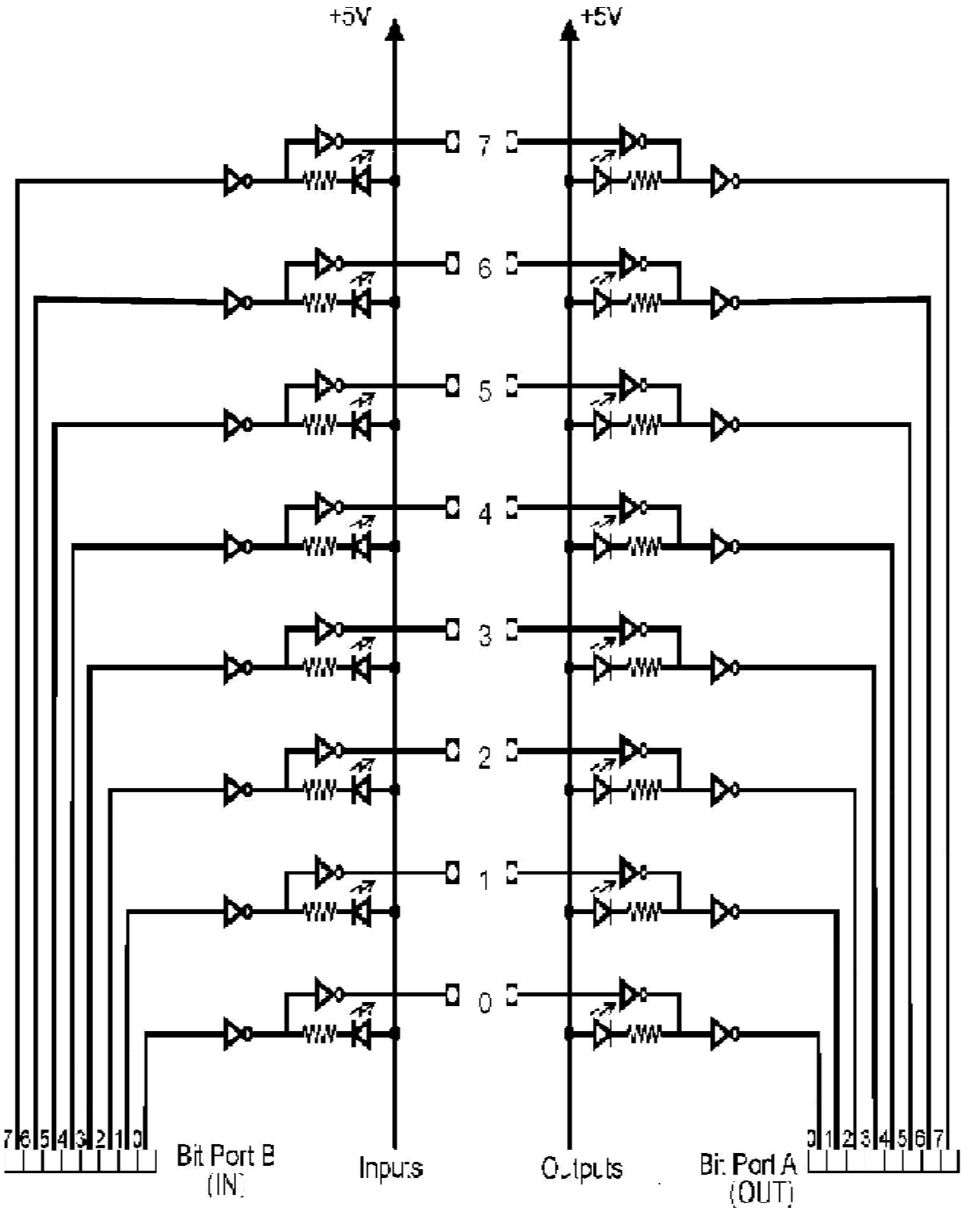
1. Program to interface 8086 and 7-Segment display to display 'FIRE' AND 'HELP'.

```
.model small
.stack 100
.data
        pa      equ   0d800h
        pb      equ   0d801h
        pc      equ   0d802h
        ctrl    equ   0d803h
        str1    db    8eh, 0f9h, 88h, 86h
        str2    db    89h, 86h, 0c7h, 8ch
.code
start:  Mov     ax, @data
        Mov     ds, ax

        Mov     al, 80h
        Mov     dx, ctrl
        out     dx, al

again:  Mov     bx, offset str1
        call    display
        call    delay
        Mov     bx, offset str2
        call    display
        call    delay

        Mov     ah, 06h
        Mov     dl, 0ffh
        Int     21h
        cmp     al, 'q'
        Jne     again
        Int     3

display proc
        Mov     si, 03h
up1:    Mov     cl, 08h
        Mov     ah, [bx+si]
up:     Mov     dx, pb
        rol     ah, 1
        Mov     al, ah
        out     dx, al
        call    clock
        Dec     cl
        Jnz     up
        Dec     si
        cmp     si, -1
```

```
        Jne     up1
        Ret

Displa  9i;y    Endp

clock   proc
        Mov     dx, pc
        Mov     al, 01h
        out     dx, al
        Mov     al, 0
        out     dx, al
        Mov     dx, pb
        ret
clock   Endp

delay   proc
        push    cx
        push    bx
        Mov     cx, 0ffffh
d2:     Mov     bx, 8fffh
d1:     Dec     bx
        Jnz     d1
        loop    d2
        pop     bx
        pop     cx
        ret
delay   Endp

        End     start
```

Result :

Circuit diagram :

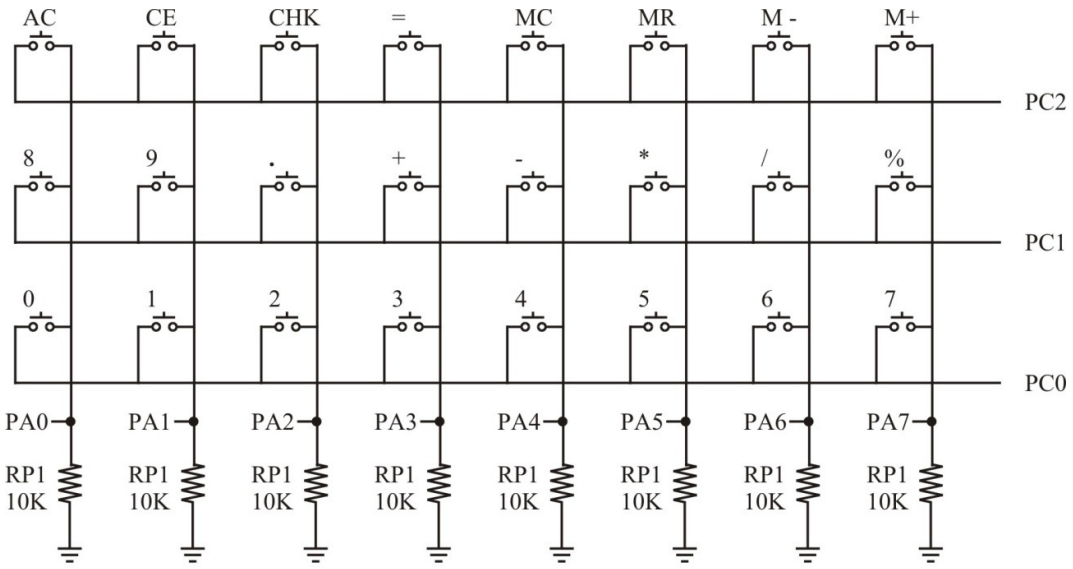


Schematic of a Logic Controller

2. Program to interface 8086 and a logic controller to check for odd or even parity

```
.model small
.data
        pa      equ 0d800h
        pb      equ 0d801h
        pc      equ 0d802h
        ctrl    equ 0d803h
.code
start:  Mov     ax, @data   ; Initialization of
        Mov     ds, ax      ; data segment

        Mov     dx, ctrl     ; configure 82C55A to mode 0
        Mov     al, 82h      ; Port A as O/P & Port B as I/P
        out     dx, al       ; by s Ending the control word to control register.

        Mov     dx, pb      ; Read the data from port B
        in      al, dx      ; of 82C55A

        Mov     bl, 00h     ; Set BL to Zero.
        Mov     cx, 08      ; Load CX with 08.
up:     rcl     al,1        ; rotate left AL by one bit along with carry.
        Jnc     down        ; jump to label "down" if CF = 1.
        Inc     bl          ; If CF = 0, Increment the contents of BL by 1.
down:   loop    up          ; Dec CL and CL != 0, jump to label up.

        test    bl,01h
        Jnz     oddp        ; If ZF = 1, jump to label oddp
        Mov     al,0ffh     ; If ZF = 0, load AL with 0FFh
        jmp     next        ; An unconditional jump to label next.
oddp:   Mov     al,00h      ; Load AL with 00h
next:   Mov     dx,pa       ; Display 00/FF on port A as input
        out     dx,al       ; contains Odd/Even number of ones.
        call    delay       ; Call delay procedure
        Mov     al, bl      ; Load AL with contents of BL
        Mov     dx, pa      ; SEnd it to port A
        out     dx, al
        Int     3
delay   proc                ; delay procedure
        Mov     ax,0ffffh   ; Just waste the time
up2:    Mov     dx,4fffh    ; by executing instructions
up1:    Dec     dx          ; which won't affect the logic
        Jnz     up1         ; of the program.
        Dec     ax
        Jnz     up2
        ret
delay   Endp
        End     start
```

Circuit diagram of interfacing devices for 8x3 Matrix keypad:



8 x 3 Matrix keypad

Label on the keytop	Hex code
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
.	0A
+	0B

Label on the keytop	Hex code
-	0C
X	0D
/	0E
%	0F
AC	10
CE	11
CHK	12
=	13
MC	14
MR	15
M	16
M+	17

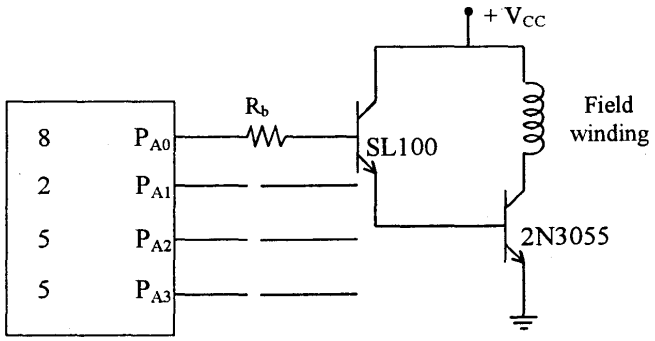
Table: HEX value equivalent to Key Press

3. Program to interface 8086 and 8x3 keypad, to scan for key closure and to store the code of the key pressed in a memory location and display on screen and also to display row and column numbers of the key pressed.

```
.model small
.stack      100
.data
    pa equ 0d800h
    pb equ 0d801h
    pc equ 0d802h
    ctrl equ 0d803h
    ASCIICODE db "0123456789.+-*/%ack=MRmn"
    str db 13,10,"press any key on the matrix keyboard$"
    str1 db 13,10,"Press y to repeat and any key to exit $"

    msg db 13, 10,"the code of the key pressed is :"
    key db ?
    msg1 db 13,10,"the row is "
    row db ?
    msg2 db 13,10,"the column is "
    col db ?,13,10,'$'

.code
disp        macro x
            Mov  dx, offset x
            Mov  ah, 09
            Int    21h
     Endm

start:  Mov    ax,@data
        Mov    ds,ax

        Mov    al,90h
        Mov    dx,ctrl
        out    dx,al

again1: disp   str
        Mov    si,0h

again:  call   scan
        Mov    al,bh   ; Row number
        add    al,31h
        Mov    row,al

        Mov    al,ah   ; Column number
        add    al,31h
        Mov    col,al
        cmp    si,00
        je     again
```

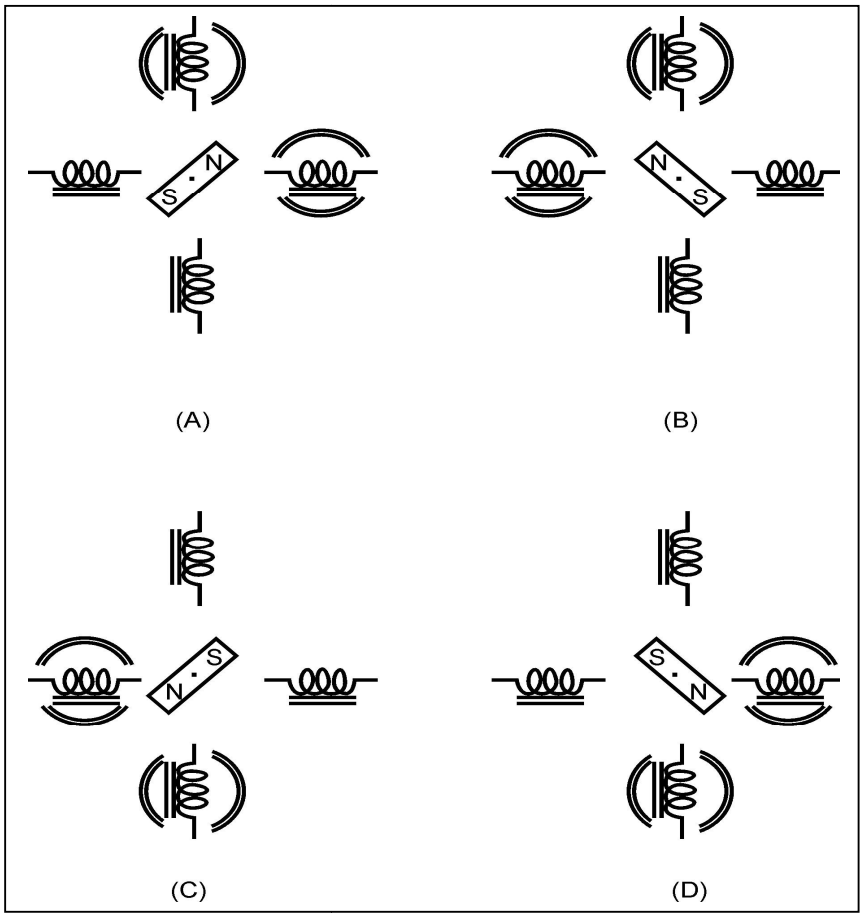
```
        Mov   cl,03
        rol   bh,cl
        Mov   cl,bh
        Mov   al,ah
        Lea   bx,ASCIICODE
        add   bl,cl
        xlat
        Mov   key,al

        disp  msg
        disp  str1
        Mov   ah, 01
        Int   21h
        cmp   al,'y'
        je    again1

        Int   3

scan    proc
        Mov   cx,03
        Mov   bh,0
        Mov   al,80h

nxtrow: rol   al,1
        Mov   bl,al
        Mov   dx,pc
        out   dx,al
        Mov   dx,pa
        in    al,dx
        cmp   al,0
        Jne   keyid

        Mov   al,bl
        Inc   bh
        loop  nxtrow
        ret

keyid: Mov si,1
        Mov cx,8
        Mov ah,0
agn: ror al,1
        jc skip
        Inc ah
        loop agn

 skip: ret
       scan Endp
     End start
```

Circuit diagram of interfacing device for stepper motor



The power circuit for one winding of the stepper motor is as shown in figure above. It is connected to the port A (P_{A0}) of 82C55A. Similar circuits are connected to the remaining lower bits of port A (P_{A1}, P_{A2}, P_{A3}). One winding is energized at a time. The coils are turned ON/OFF one at a time successively.

The stepper motor showing full-step operation is shown below.
(A) 45-degrees. (B) 135-degrees (C) 225-degrees (D) 315-degrees.



4. Program to interface 8086 and a stepper motor to rotate in both directions

```
.model small
.data
        pa      equ 0d800h
        pb      equ 0d801h
        pc      equ 0d802h
        ctrl    equ 0d803h
        nstep   db 2
.code
start:  Mov     ax, @data
        Mov     ds, ax

        Mov     al, 80h
        Mov     dx, ctrl
        out     dx, al

        Mov     bh, nstep
again:  Mov     al, 88h
        Mov     bl, 04

again1: call    step
        ror     al, 1
        Dec     bl
        Jnz     again1

        Dec     bh
        Jnz     again
        Int     3

step    proc
        Mov     dx, pa
        out     dx, al

        Mov     cx, 0ffffh
d2:     Mov     di, 8fffh
d1:     Dec     di
        Jnz     d1
        loop    d2
        ret
step    Endp

        End     start
```

Note: To rotate stepper motor in anti clockwise direction use **rol** instruction.

5. Program for PC to PC communication

Program for PC-1(Tx)

```
.model small
.data
        pa      equ 0d800h
        ctrl    equ 0d803h

.code
start:  Mov     ax, @data
        Mov     ds, ax

        Mov     al, 80h
        Mov     dx, ctrl
        out     dx, al

        Mov     al, 11h   ; data to be transmitted
        Mov     dx, pa
        out     dx, al

        int 3
        end
```

Program for PC-2(Rx)

```
.model small
.data
        pa      equ 0d800h
        ctrl    equ 0d803h

.code
start:  Mov     ax, @data
        Mov     ds, ax

        Mov     al, 90h
        Mov     dx, ctrl
        out     dx, al

        Mov     dx, pa
        in      al, dx

        int 3
        end
```

6. Program for LPT. An ALP to interface “Printer”

```
.model small
.data
        String db 63h,69h,74h,20h,20h; Ascii vaue for CIT
.code
        Mov ax,@data
        Mov ds,ax
        Mov cx, 09h
Back1:  mov bl, 05h
        Mov si, 00h
Back:   mov al, string[si]
        Mov dl, al
        Mov ah, 05h
        Int 21h
        Inc si
        Dec bl
        Jnz back
        Loop back1
        Mov ah, 4ch
        Int 21h
end
```

Additional Programs

1. Program to Generate a SINE wave using DAC

```
.model small
.data
        pa      equ 0c400h
        pb      equ 0c401h
        pc      equ 0c402h
        ctrl    equ 0c403h
        table   db 128,132,137,141,146,150,154,159,163,167,171,176,180,184,188
                db 192,196,199,203,206,210,213,217,220,223,226,229,231,234,236
                db 239,241,243,245,247,248,250,251,252,253,254,255
                db 255,254,253,252,251,250,248,247,245,243,241,239,236,234,231
                db 229,226,223,220,217,213,210,206,203,199,196,192,188,184,180
                db 176,171,167,163,159,154,150,146,141,137,132,128
                db 123,119,114,110,105,101,97,93,88,84,80,76,72,68,64,60,56,52,49
                db 45,42,39,36,33,30,27,24,22,19,17,15,11,9,7,6,5,4,3,2,1,0
                db 0,1,2,3,4,5,6,7,9,11,15,17,19,22,24,27,30,33,36,39,42,45,49,52,56
                db 60,64,68,72,76,80,84,88,93,97,101,105,110,114,119,123
.code
start:  mov     ax,@data
        mov     ds,ax

        mov     al,80h                  ; All the ports are out put ports
        mov     dx,ctrl
        out     dx,al

again:  mov     bx,05h
up:     mov     cx,164                  ; Load 164 values
        mov     si,00h
        mov     dx,pa

again1: mov     al,table[si]            ; Load each value from Look-up-table to
al
        out     dx,al
        inc     si
        loop    again1

        dec     bx
        cmp     bx,00
        jne     up

        mov     ah,06h                  ; direct console input or output
        mov     dl,0ffh                 ; Read the character from the keyboard
        int     21h
        jz      again
        int     3
        end     start
```

2. Program to generate a half rectified SINE wave using DAC

```
 .model small
.data
        pa      equ 0c400h
        pb      equ 0c401h
        pc      equ 0c402h
        ctrl    equ 0c403h
        table   db 128,132,137,141,146,150,154,159,163,167,171,176,180,184,188
                db 192,196,199,203,206,210,213,217,220,223,226,229,231,234,236
                db 239,241,243,245,247,248,250,251,252,253,254,255,254,253,252
                db 251,250,248,247,245,243,241,239,236,234,231,229,226,223,220
                db 217,213,210,206,203,199,196,192,188,184,180,176,171,167,163
                db 159,154,150,146,141,137,132,128  ; Look_up_table
.code
start:  mov     ax,@data
        mov     ds,ax

        mov     al,80h          ; All the ports are out put ports
        mov     dx,ctrl
        out     dx,al

again3: mov     bx,05h
up:     mov     cx,83           ; Load 83 values
        mov     si,00

again4: mov     dx,pa
        mov     al,table[si]    ; Load each value from Look-up-table to al
        out     dx,al
        inc     si
        loop    again4

        mov     cx,83
        mov     al,128
next:   out     dx,al
        loop    next
        dec     bx
        cmp     bx,00h
        jnz     up

        mov     ah,06h          ; direct console input or output
        mov     dl,0ffh         ; Read the character from the keyboard
        int     21h
        jz      again3
        int     3               ; Terminate the program
end     start
```

3. Program to generate a fully rectified SINE wave using DAC

```
.model small
.data
        pa      equ 0c400h
        pb      equ 0c401h
        pc      equ 0c402h
        ctrl    equ 0c403h
        table   db 128,132,137,141,146,150,154,159,163,167,171,176,180,184,188
                db 192,196,199,203,206,210,213,217,220,223,217,220,223,226,229
                db 231,234,236,239,241,243,245,247,248,250,251,252,253,254,255
                db 254,253,252,251,250,248,247,245,243,241,239,236,234,231,229
                db 226,223,220,217,213,210,206,203,199,196,192,188,184,180,176
                db 171,167,163,159,154,180,146,141,137,132,128
        count   dw 83
.code
start:  mov     ax,@data
        mov     ds,ax

        mov     al,80h          ; All the ports are out put ports
        mov     dx,ctrl
        out     dx,al

agn :   mov     bx,05
back1:  mov     cx,count        ; Load 83 values
        mov     si,00h

back:   mov     al,table[si]    ; Load each value from Look-up-table to al
        mov     dx,pa
        out     dx,al
        inc     si
        loop    back

        dec     bx
        cmp     bx,00
        jnz     back1

        mov     ah,06h          ; direct console input or output
        mov     dl,0ffh         ; Read the character from the keyboard
        int     21h
        jz      agn
        int     3
end     start
```

4. Program to drive an elevator interface in the following way:

i. Initially the elevator should be in the ground floor, with all requests in OFF state.

ii. When a request is made from a floor, the elevator should move to that floor, wait there for a couple of seconds (approximately), and then come down to ground floor and stop. If some requests occur during going up or coming down they should be ignored.

```
.model small
.data
        pa equ 0c800h           ;define port addresses
        pb equ 0c801h
        pc equ 0c802h
        ctrl equ 0c803h         ;define control word address
.code
        mov ax, @data
        mov ds, ax              ;initialize data segment

        mov al, 82h             ;initialize port A as output and port B as input port
        mov dx, ctrl
        out dx, al

        mov bl, 0               ; Initially display lift in ground floor

                                ; PRESS ANY KEY TO EXIT
start:  call delay
        mov ah, 06h
        mov dl, 0ffh
        int 21h
        jz proceed              ;if none of the key is pressed then jump to location
proceed
        int 3                   ;else terminate program execution

; PLACE LIFT IN GROUND FLOOR
proceed: call delay
        mov al, bl              ;take floor number to AL
        or al, 0f0h             ;set upper nibble of the number
        mov dx, pa
        out dx, al
        cmp bl, 0               ;check whether the lift is in ground floor or not
        jnz down                ;if not in then jump to location down to move lift to ground
floor
        jmp fchk                ;else jump to location fchk to check the request from any
floor
down:   dec bl
        jmp proceed

;CHECK REQUEST FROM ANY FLOOR
fchk:   call chk                ;call procedure chk to check is there request from any
floor
```

```
        shr al, 01          ;shift right the request by 1 position
        jnc gfr             ;if carry is not set then request will be from ground
                            ; floor and jump to location gfr
        shr al, 01          ;else shirt right the request by 1 more position
        jnc ffr             ;if carry is not set then request will be from 1st floor and
                            ;jump to location ffr
        shr al, 01          ;else shirt right the request by 1 more position
        jnc sfr             ;if carry is not set then request will be from 2nd floor
                            ;and jump tp location sfr
        shr al, 1           ;else shirt right the request by 1 more position
        jnc tfr             ;if carry is not set then request will be from 2nd floor
                            ;and jump tp location sfr
        jmp start           ;else jump to start

gfr:    call delay
        mov al, 0e0h        ;data to disable ground floor request
        mov dx, pa          ;load port A address to DX reg.
        out dx, al          ;send data to port A
        jmp start           ;to repeat the process jump to location start

ffr:    call delay
        mov bl, 3
        call floor
        mov al, 0d3h
        mov dx, pa
        out dx, al
        jmp start

sfr:    call delay
        mov bl, 6
        call floor
        mov al, 0b6h
        mov dx, pa
        out dx, al
        jmp start

tfr:    call delay
        mov bl, 9
        call floor
        mov al, 79h
        mov dx, pa
        out dx, al
        jmp start

chk     proc
        mov dx, pb
        in al,dx            ;read data from port b
        or al,0f0h          ;set upper nibble of the data
        cmp al,0ffh         ;check is there any request or not
        jz chk              ;if no request then jump to location chk
```

```
        ret                 ;else return to main program
chk     endp                ;end of procedure

floor   proc

mov cl, 0

floor1: inc cl
        mov al, cl
        or al, 0f0h
        mov dx, pa
        out dx, al
        call delay
        cmp cl, bl
        jnz floor1
        ret
floor   endp

delay   proc
delay     proc
          push    cx
          push    bx

          mov     cx, 0ffffh
d2:       mov     bx, 8fffh
d1:       dec     bx
          jnz     d1
          loop    d2

          pop     bx
          pop     cx
          ret
delay     endp
```

5. Assume any suitable message of 12 characters length and display it in the rolling fashion on a 7-segment display Interface for a suitable period of time. Ensure a flashing rate that makes it easy to read the message.

```
.model small
.stack 100
.data
        pa      equ 0d800h
        pb      equ 0d801h
        pc      equ 0d802h
        ctrl    equ 0d803h
        str1    db 0c0h,0f9h,0a4h,0b0h,99h,92h,83h,0f8h,80h,98h,0c0h,0f9h
.code
start: Mov      dx, @data
        Mov     ds, dx

        Mov     al, 80h
        Mov     dx, ctrl
        out     dx, al

again:          Mov bx, offset str1
        call    display
        call    delay
        Mov     ah, 06h
        Mov     dl, 0ffh
        Int     21h
        cmp     al, 'q'
        Jnz     again
        Int     3

display         proc
        Mov     si, 0bh

up1:  call      delay
        Mov     cl, 08h
        Mov     ah, [bx+si]

up:     Mov     dx, pb
        rol     ah, 1
        Mov     al, ah
        out     dx, al
        call    clock
        Dec     cl
        Jnz     up

        Dec     si
        cmp     si, -1
        Jne     up1
        ret
display         Endp
```

```
clock   proc
        Mov     dx, pc
        Mov     al, 01h
        out     dx, al
        Mov     al, 0
        out     dx, al
        Mov     dx, pb
        ret
clock           Endp

delay           proc
        push    cx
        push    bx

        Mov     cx, 0ffffh
d2:     Mov     bx, 8fffh
d1:     Dec     bx
        Jnz     d1
        loop    d2

        pop     bx
        pop     cx
        ret
delay           Endp

        End     start
```

6. Program to read the status of two 8-bit inputs (x & y) from the logical controller Interface and display x*y.

```
.model small
.data
pa  equ 0d800h
pb  equ 0d801h
pc  equ 0d802h
ctrl equ 0d803h
X   db ?
Y   db ?
Z   dw ?
str1 db 13,10,"read X$"
str2 db 13,10,"read Y$"
str3 db 13,10,“display result $”,13,10
.code
start:Mov   ax,@data
Mov ds,ax

Mov al,82h
Mov dx,ctrl
out dx,al

up: Mov ah,09h
Mov dx,offset str1
Int  21h

Mov ah,01
Int  21h

Mov dx,pb
in al,dx
Mov x,al

Mov ah,09h
Mov dx,offset str2
Int 21h

Mov ah,01
Int 21h

Mov dx,pb
in al,dx
Mov y,al

Mov ah,x
Mul ah
Mov z,ax

Mov ah,09h
```

```
Mov dx,offset str3
Int 21h

Mov dx,pa
Mov ax,z
Mov al,ah
out dx,al

call delay
Mov dx,pa
Mov ax,z
out dx,al

Mov ah,01
Int 21h
cmp al,"y"
je up
Int  3

delay proc
Mov ax,0FFFFH
up2: Mov  bx, 0FFFFH
up1: Dec   bx
Jnz          up1
Dec ax
Jnz          up2
ret
delay Endp
End          start
```

7. Program to interace logic controller as BCD up-down counter function

```
.model small
.data
        pa      equ 0d800h
        pb      equ 0d801h
        pc      equ 0d802h
        ctrl    equ 0d803h
.code
start: Mov      ax,@data
        Mov     ds,ax
        Mov     al,82h
        Mov     dx,ctrl
        out     dx,al

        Mov     cl,00h
up:     call    delay
        Mov     ah,06h
        Mov     dl,0ffh
        Int     21h
        cmp     al,'q'
        je      exit

        Mov     dx,pb
        in      al,dx
        cmp     al,00
        je      downc

        Mov     al,cl
        Mov     dx,pa
        out     dx,al
        add     al,01h
        daa
        Mov     cl,al
        cmp     cl,99h
        jbe     up
        Mov     cl,00
        jmp     up

; down counter

downc:      Mov al,cl
        Mov     dx,pa
        out     dx,al
        cmp     cl,00h
        je      down
        Mov     al,cl
        sub     al,01
        das
        Mov     cl,al
```

```
        jmp     down1

down:   Mov cl,99h
down1:  jmp  up
exit:  Int      3

delay   proc
        Mov ax,0ffffh
up2:  Mov     bx,4fffh
up1:  Dec     bx
        Jnz     up1
        Dec     ax
        Jnz     up2
        ret
delay   Endp

        End     start
```

Viva Questions

1. What is a Microprocessor?

2. What is the difference between 8086 and 8088?

3. What are the functional units in 8086? .

4. What are the flags in 8086?

5. What is the Maximum clock frequency in 8086?

6. What are the various segments registers in 8086?

7. Logic calculations are done in which type of registers?

8. How 8086 is faster than 8085?

9. What does EU do?

10. Which Segment is used to store Interrupt and subroutine return address register?

11. What does microprocessor speed depend on?

12. What is the size of data bus and address bus in 8086?

13. What is the maximun memory addressing capability of 8086?

14. What is flag?

15. Which Flags can be set or reset by the programmer and also used to control the operation of the processor?

16. In how many modes 8086 can be opertaed and how?

17. What is the difference between min mode and max mode of 8086?

18. Which bus controller used in maximum mode of 8086?

19. What is stack?

20. Which Stack is used in 8086?

21. What is the position of the Stack PoInter after the PUSH instruction

22. What is the position of the Stack PoInter after the POP instruction?

23. What is an Interrupt?

24. What are the various Interrupts in 8086?

25. What is meant by Maskable Interrupts?

26. What is Non-Maskable Interrupts?

27. Which Interrupts are generally used for critical events?

28. Give example for Non-Maskable Interrupts?

29. Give examples for Maskable Interrupts?

30. What are SIM and RIM instructions?

31. What is macro?

32. What is the difference between Macro and Procedure?

33. What is meant by LATCH?

34. What is a compiler?

35. What is the disadvantage of microprocessor?

Instruction Set:

Instructions	Operands	Description
MOV	**REG, memory** **memory, REG** **REG, REG** **memory, immediate** **REG, immediate** **SREG, memory** **memory, SREG** **REG, SREG** **SREG, REG**	**Copy operand2 to operand1.** The MOV instruction cannot: • Set the value of the CS and IP registers. • Copy value of one segment register to another segment register (should copy to general register first). • Copy immediate value to segment register (should copy to general register first). **Algorithm: operand1 = operand2** Ex: **Mov AX,BX ;Copy contents of BX to AX** **Mov si,00h ;load Si with 00h**
MUL	**REG** **Memory**	**Unsigned Multiply.** Multiply the contents of REG/Memory with contents of AL register. **Algorithm:** When operand is a **byte**: AX = AL * operand. When operand is a **word**: (DX: AX) = AX * operand.
CMP	**REG, memory** **memory, REG** **REG, REG** **memory, immediate** **REG, immediate**	**Compare.** **Algorithm: operand1 - operand2** Result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.
JMP	**Label**	**Unconditional Jump.** Transfers control to another part of the program. 4-byte address may be entered in this form: 1234h: 5678h, first value is a segment second value is an offset. **Algorithm:** always jump
JA	**Label**	**Jump If Above.** Short Jump if first operand is Above second operand (as set by CMP instruction). Unsigned. **Algorithm:** if (CF = 0) and (ZF = 0) then jump
JAE	**Label**	**Jump If Above Or Equal** Short Jump if first operand is Above or Equal to second operand (as set by CMP instruction). Unsigned. **Algorithm:** if CF = 0 then jump
JB	**Label**	**Jump If Below.** Short Jump if first operand is Below second operand (as set by

		CMP instruction). Unsigned. **Algorithm:** if CF = 1 then jump
JBE	**Label**	**Jump If Below Or Equal** Short Jump if first operand is Below second operand (as set by CMP instruction). Unsigned. **Algorithm**: if CF = 1 then jump
JC	**Label**	**Jump If Carry** Short Jump if Carry flag is set to 1. **Algorithm:** if CF = 1 then jump
JE	**Label**	**Jump If Equal.** Short Jump if first operand is Equal to second operand (as set by CMP instruction). Signed/Unsigned. **Algorithm:** if ZF = 1 then jump
JG	**Label**	**Jump If Greater** Short Jump if first operand is Greater then second operand (as set by CMP instruction). Signed. **Algorithm:** if (ZF = 0) and (SF = OF) then jump
JGE	**Label**	**Jump If Greater Or Equal.** Short Jump if first operand is Greater or Equal to second operand (as set by CMP instruction). Signed. **Algorithm:** if SF = OF then jump
JL	**Label**	**Jump If Less than.** Short Jump if first operand is Less then second operand (as set by CMP instruction). Signed. **Algorithm:** if SF <> OF then jump
JLE	**Label**	**Jump If Less Or Equal.** Short Jump if first operand is Less or Equal to second operand (as set by CMP instruction). Signed. **Algorithm:**

		if SF <> OF or ZF = 1 then jump
JNZ	**Label**	**Jump If Non Zero.** Short Jump if Not Zero (not equal). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions. **Algorithm:** if ZF = 0 then jump
JZ	**Label**	**Jump If Zero.** Short Jump if Zero (equal). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions. **Algorithm:** if ZF = 1 then jump
LEA	**REG, memory**	**Load Effective Address.** **Algorithm:** • REG = address of memory (offset)
LOOP	**Label**	**Decrease CX, jump to label if CX not zero.** **Algorithm:** • CX = CX - 1 • if CX <> 0 then ○ jump else ○ no jump, continue
ADD	**REG, memory** **memory, REG** **REG, REG** **memory, immediate** **REG, immediate**	**Add.** **Algorithm:** operand1 = operand1 + operand2
AND	**REG, memory** **memory, REG** **REG, REG** **memory, immediate** **REG, immediate**	Logical AND between all bits of two operands. Result is stored in operand1. These rules apply: **1 AND 1 = 1; 1 AND 0 = 0** **0 AND 1 = 0; 0 AND 0 = 0**

OR	**REG, memory** **memory, REG** **REG, REG** **memory, immediate** **REG, immediate**	Logical OR between all bits of two operands. Result is stored in first operand. These rules apply: **1 OR 1 = 1; 1 OR 0 = 1** **0 OR 1 = 1; 0 OR 0 = 0**
SUB	**REG, memory** **memory, REG** **REG, REG** **memory, immediate** **REG, immediate**	**Subtract.** **Algorithm:** operand1 = operand1 - operand2

DAA	**No Operands**	**Decimal adjust After Addition.**
		Corrects the result of addition of two packed BCD values.
		
		Algorithm:
		if low nibble of AL > 9 or AF = 1 then:
		• AL = AL + 6
		• AF = 1
		
		if AL > 9Fh or CF = 1 then:
		• AL = AL + 60h
		
		• CF = 1

DAS	**No Operands**	**Decimal adjust After Subtraction**.
		Corrects the result of subtraction of two packed BCD values.
		
		Algorithm:
		if low nibble of AL > 9 or AF = 1 then:
		• AL = AL - 6
		• AF = 1
		
		if AL > 9Fh or CF = 1 then:
		• AL = AL - 60h
		
		• CF = 1

INC	**REG** **memory**	**Increment.** **Algorithm:** operand = operand + 1
DEC	**REG** **Memory**	**Decrement.** **Algorithm:** operand = operand – 1
DIV	**REG** **Memory**	**Unsigned Divide.** **Algorithm:** when operand is a **byte**: AL = AX / operand AH = remainder (modulus) when operand is a **word**: AX = (DX AX) / operand DX = remainder (modulus)

SHL	**memory, immediate** **REG, immediate** **memory, CL** **REG, CL**	**Shift Left.** Shift operand1 Left. The number of shifts is set by operand2. **Algorithm:** • Shift all bits left, the bit that goes off is set to CF. • Zero bit is inserted to the right-most position.
SHR	**memory, immediate** **REG, immediate** **memory, CL** **REG, CL**	**Shift Right.** Shift operand1 Right. The number of shifts is set by operand2. **Algorithm:** • Shift all bits right, the bit that goes off is set to CF. • Zero bit is inserted to the left-most position.
ROL	**memory, immediate** **REG, immediate** **memory, CL** **REG, CL**	**Rotate Left.** Rotate operand1 left. The number of rotates is set by operand2. **Algorithm:** Shift all bits left, the bit that goes off is set to CF and the same bit is inserted to the right-most position.
ROR	**memory, immediate** **REG, immediate** **memory, CL** **REG, CL**	**Rotate Right.** Rotate operand1 right. The number of rotates is set by operand2. **Algorithm:** Shift all bits right, the bit that goes off is set to CF and the same bit is inserted to the left-most position.
CALL	**procedure name** **label**	Transfers control to procedure, return address is (IP) pushed to stack.
RET	**No operands** **Or even immediate date**	Return from near procedure. **Algorithm:** • Pop from stack: o IP if immediate operand is present: SP = SP + operand
IN	**AL, im.byte** **AL, DX** **AX, im.byte** **AX, DX**	Input from port Into **AL** or **AX**. Second operand is a port number. If required to access port number over 255 - **DX** register should be used.
OUT	**AL, im.byte** **AL, DX** **AX, im.byte** **AX, DX**	Output from **AL** or **AX** to port. First operand is a port number. If required to access port number over 255 - **DX** register should be used.

POP	**REG** **SREG** **memory**	Get 16 bit value from the stack. Algorithm: Operand = SS : [SP](top of stack) SP = Sp + 2.
PUSH	**REG** **SREG** **memory**	Store 16 bit value in the stack. **Algorithm:** • SP = SP - 2 • SS:[SP] (top of the stack) = operand
XOR	**REG, memory** **memory, REG** **REG, REG** **memory, immediate** **REG, immediate**	Logical XOR (Exclusive OR) between all bits of two operands. Result is stored in first operand. These rules apply: **1 XOR 1 = 0; 1 XOR 0 = 1** **0 XOR 1 = 1; 0 XOR 0 = 0**
XCHG	**REG, memory** **memory, REG** **REG, REG**	Exchange values of two operands. **Algorithm:** operand1 < - > operand2
XLAT	**No Operands**	Translate byte from table. Copy value of memory byte at DS:[BX + unsigned AL] to AL register. **Algorithm:** AL = DS:[BX + unsigned AL]

AAA	**No Operands**	**ASCII Adjust after Addition.**
		Corrects result in AH and AL after addition when working with BCD values.
		
		Algorithm:
		
		if low nibble of AL > 9 or AF = 1 then:
		• AL = AL + 6
		• AH = AH + 1
		• AF = 1
		• CF = 1
		
		else
		• AF = 0
		• CF = 0
		
		in both cases:
		cLear the high nibble of AL.
		
		Example:
		MOV AX, 15 ; AH = 00, AL = 0Fh
		AAA ; AH = 01, AL = 05

		ASCII Adjust after Subtraction. Corrects result in AH and AL after subtraction when working with BCD values.

AAS	**No Operands**	**Algorithm:**
		
		if low nibble of AL > 9 or AF = 1 then:
		• AL = AL - 6
		• AH = AH - 1
		• AF = 1
		• CF = 1
		
		else
		• AF = 0
		• CF = 0
		
		in both cases:
		cLear the high nibble of AL.
		
		Example:
		MOV AX, 02FFh ; AH = 02, AL = 0FFh
		AAS ; AH = 01, AL = 09

AAM	**No Operands**	**ASCII Adjust after Multiplication.** Corrects the result of Multiplication of two BCD values. **Algorithm:** • AH = AL / 10 • AL = remainder Example: MOV AL, 15 ; AL = 0Fh AAM ; AH = 01, AL = 05

DOS Interrupt INT 2lH Function Calls

01H	**READ THE KEYBOARD**
Entry	AH = 01H
Exit	AL = ASCII character
Notes	If Al = 00H, the function call must be invoked again to read an extEnded ASCII character. This function call automatically echoes whatever is typed to the video screen.
02H	**WRITE TO STANDARD OUTPUT DEVICE**
Entry	AH = 02H DL = ASCII character to be displayed
Notes	This function displays character on video display
06H	**DIRECT CONSOLE READ/WRITE**
Entry	AH = 06H DL = 0FFH or DL = ASCII character
Exit	AL = ASCII character
Notes	If DL = 0FFH on entry, then this function reads the console. If DL = ASCII character, then this function displays the ASCII character on the console (CON) video screen.
09H	**DISPLAY A CHARACTER STRING**
Entry	AH = 09H DS:DX = Address of the character string
Exit	AL = ASCII character
Notes	The character string must End with an ASCII $ (24H). The character string can be of any length and may contain control characters such as carriage return (0DH) and line feed (0AH).
2CH	**READ SYSTEM TIME**
Entry	AH = 2CH
Exit	CH = Hours (0-23) CL = Minutes DH = Seconds DL = Hundredth of seconds
Notes	All times are returned in binary form, and hundredths of seconds may not be available.
3CH	**CREATE A NEW FILE**
Entry	AH = 3CH CX = Attribute word DS:DX = Address of ASCII-Z string file name
Exit	AX = Error code if carry set AX = File handle if carry cLeared
Notes	The attribute word can contain any of the following(adding

	together) : 01H = Read-only access, 02H = Hidden file or directory, 04H = System file, 0BH = Volume label, 10H = Subdirectory, and 20H = Archive bit. In most cases, a file is created with 0000H.
41H	**DELETE A FILE**
Entry	AH = 41H DS:DX = Address of ASCII-Z string file name
Exit	AX = Error code if carry set
4CH	**TERMINATE A PROCESS**
Entry	AH = 4CH AL = Error code
Exit	Returns control to DOS
Notes	This function codes are AL = 00H to load and execute a program, AL = 01H to load a program but not execute it, AL = 03H to load a program overlay, and AL = 05H to enter the EXEC state.

BIOS Interrupt INT 10H Function Calls	
02H	**SELECT CURSOR POSITION**
Entry	AH = 02H BH = Page number (usually 0) DH = Row number (beginning with 0) DL = Column number (beginning with 0)
Exit	Changes cursor to new position
03H	**READ CURSOR POSITION**
Entry	AH = 03H BH = Page number
Exit	CH = starting line (cursor size) CL = Ending line (cursor size) DH = current row DL = current column

QUESTION BANK

1. a) Write an ALP to Move data from source to destination without overlap.
   b) Write an ALP to scan a 8x3 keypad for key closure and to store the code of the key pressed in a memory location and display on screen. Also display row and column numbers of the key pressed.
2. a) Write an ALP to convert BCD number to BINARY number.
   b) Write an ALP Interface a stepper motor to rotate in clockwise/anti clockwise direction.
3. a) Write an ALP to Move string from source to destination.
   b) Write an ALP to Interface seven-segment display that displays 'FIRE' and 'HELP'.
4. a) Write an ALP to add/subtract two Multi-precision numbers.
   b) Assume any suitable message of 12 characters length and display it in the rolling fashion on a 7-segment display interface for a suitable period of time..
5. a) Write an ALP to find LCM of two 8-bit number.
   b) Program to read the status of two 8-bit inputs (x & y) from the logical controller interface and display x*y..
6. a) Write an ALP to reverse a given string.
   b) Program to interface logic controller as BCD up-down counter.
7. a) Write an ALP to Divide 32-bit unsigned number by a 16-bit number.
   b) Write an ALP to Interface a logic controller to check the parity (even/odd) and to display number of 1's in the given data.
8. a) Write an ALP to illustrate the use of AAA/AAS instruction.
   b) Write an ALP Interface a stepper motor to rotate in clockwise/anti clockwise direction.
9. a) Write an ALP to check whether given data is positive or negative.
   b) Write an ALP to Interface a logic controller to check the parity (even/odd) and to display number of 1's in the given data.
10. a) Write an ALP to Multiply two 16 bit numbers (signed/unsigned).
    b) Write an ALP to Interface seven-segment display that displays 'FIRE' and 'HELP'.
11. a) Write an ALP to find HCF of two 8-bit numbers.
    b) Write an ALP to Interface a logic controller to check the parity (even/odd)

and to display number of 1's in the given data.

12. a) Write an ALP to Move a block of data from source to destination with overlap in either direction.
    b) Write an ALP to Interface a stepper motor to rotate in clockwise /anticlock-wise direction.
13. a) Write an ALP to find factorial of a given number by using recursive method.
    b) Write an ALP to scan a 8x3 keypad for key closure and to store the code of the key pressed in a memory location and display on screen. Also display row and column numbers of the key pressed.
14. a) Write an ALP to find square and cube of a 16-bit number
    b) Write an ALP Interface a stepper motor to rotate in clockwise/anti clock-wise direction.
15. a) Write an ALP to check whether given number is odd or even.
    b) Assume any suitable message of 12 characters length and display it in the rolling fashion on a 7-segment display interface for a suitable period of time
16. a) Write an ALP to count logical 1's and 0's in a given data.
    b) Write an ALP to Interface seven-segment display that displays 'FIRE' and 'HELP'.
17. a) Write an ALP to find smallest/largest number in a given array.
    b) Write an ALP to Interface a logic controller to check the parity (even/odd) and to display number of 1's in the given data.
18. a) Write an ALP to check whether the given number is 2 out of 5 code or not.
    b) Write an ALP Interface a stepper motor to rotate in clockwise/anticlock-wise direction.
19. a) Write an ALP to Interchange two blocks of data stored in memory location.
    b) Program to interface logic controller as BCD up-down counter
20. a) Write an ALP to illustrate use of AAM instruction .
    b) Write an ALP to Interface a logic controller to check the parity (even/odd) and to display number of 1's in the given data.
21. a) Write an ALP to check whether the given 8-bit data is bit wise palindrome or not.
    b) Write an ALP to Interface seven-segment display that displays 'FIRE' and 'HELP'.

22. a) Write an ALP to sort given numbers in ascending/descending order.
    b) Write an ALP Interface a stepper motor to rotate in clockwise/anti-clockwise direction.
23. a) Write an ALP to search a character in a given string.
    b) Program to read the status of two 8-bit inputs (x & y) from the logical controller interface and display x*y.
24. a) Write an ALP to display a character/ string on console.
    b) Write an ALP to Interface a logic controller to check the parity (even/odd) and to display number of 1's in the given data.
25. a) i) Write an ALP to read buffered keyboard input.
    ii) Write an ALP to read a character from the keyboard.
    b) Write an ALP Interface a stepper motor to rotate in clockwise/anti clock-wise direction.
26. a) Write an ALP to check whether the given 8-bit data is nibble wise palindrome or not.
    b) Write an ALP to Interface a logic controller to check the parity (even/odd) and to display number of 1's in the given data.

REFERENCES:

1. **Microcomputer systems-The 8086 / 8088 Family**
Y.C. Liu and G. A. Gibson, 2E PHI -2003

2. **The Intel Microprocessor, Architecture, Programming and Interfacing**
Barry B. Brey, 6e, Pearson Education / PHI, 2003