MICROPROCESSOR 8086

REVA INSTITUTE OF TECHNOLOGY & MANAGEMENT, Bangalore

REVA

KNOWLEDGE IS POWER

IV Semester B.E. (CSE/ISE)

Department of Computer Science & Engineering and Information Science & Engineering

Prepared by:

Prof. Venkatesh Prasad & Prof. G. C. Satish, Dept of CSE
L. Krishnananda, Asst. Professor, Dept of ISE

06CSL - 48

LAB MANUAL

Kattigenahalli, Yelahanka, Bangalore-560064
Tel: 080-65687563/64 Fax:080-28478534 www.revainstitution.org

Books to be Referred:

1. Microprocessors and Interfacing – 2nd Edition, Douglas V Hall
2. IBM PC Assembly language programming – Peter Abel
3. Microprocessor X86 Programming – K R Venugopal
4. Advanced Microprocessor & IBM PC Assembly language programming – Uday Kumar K

Theory:

- A **Microprocessor** is a programmable, digital logic device fabricated on a single VLSI chip which can perform a set of arithmetic and logic operations as per the "instructions" given by the user.

- Any microprocessor has minimum three basic functional blocks: Arithmetic Logic Unit (ALU), Timing & Control unit, Register array

- The user writes his/her programs using English-like words (called 'mnemonics') and is known as "assembly language program" (ALP).

- A software called "Assembler" converts the user ALP into **HEX/binary form** (called machine language) which is fed to the processor. The processor internally decodes this binary code and performs the operation**.**

8086 Internal Block diagram

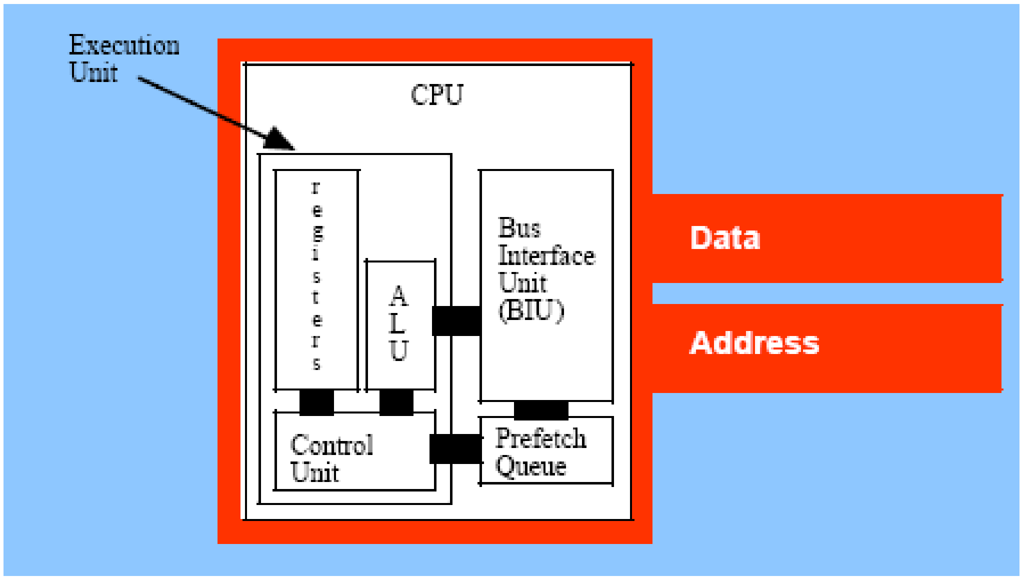
8086 is a 16-bit processor having 16-bit data bus and 20-bit address bus. The block diagram of 8086 is as shown. This can be subdivided into two parts; the Bus Interface Unit (BIU) and Execution Unit (EU).

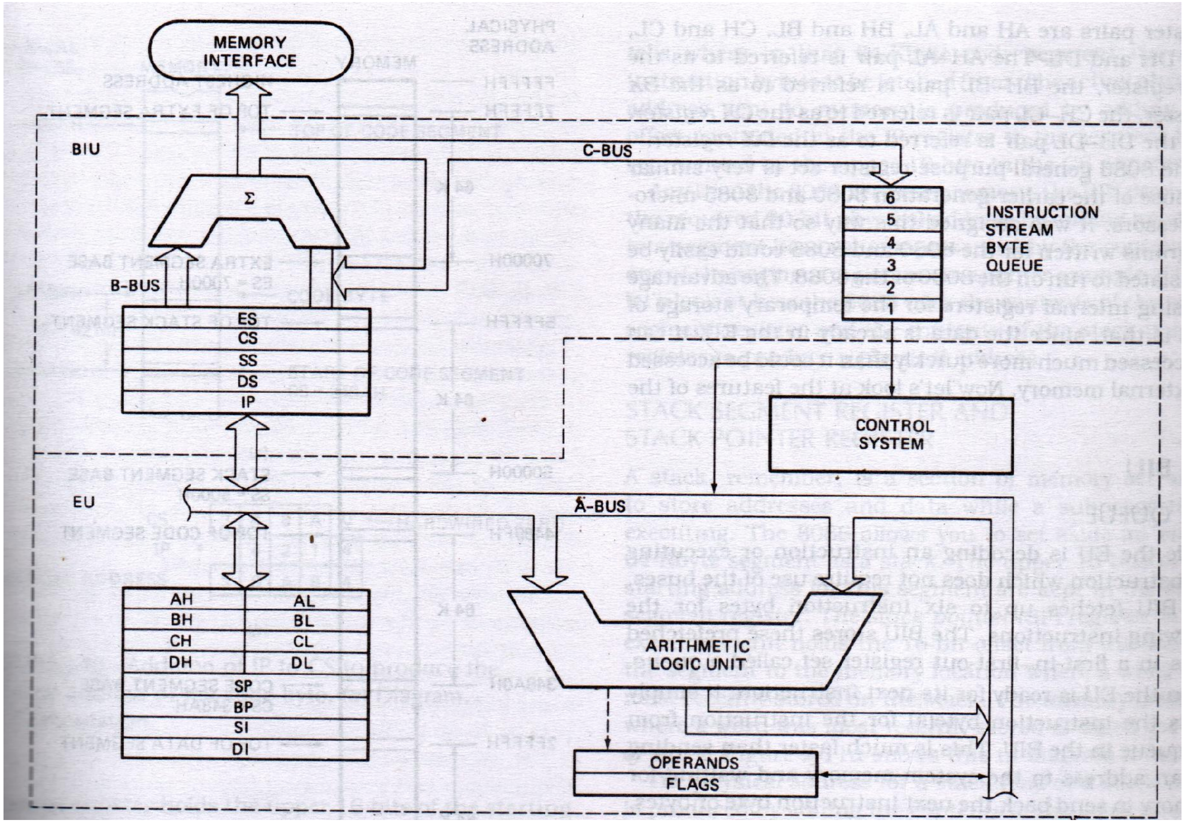
BUS INTERFACE UNIT:

The BIU consists of **segment registers**, an adder to generate 20 bit address and instruction **prefetch queue**. It is responsible for all the external bus operations like opcode fetch, mem read, mem write, I/O read/write etc,. Once this address is sent OUT of BIU, the instruction and data bytes are fetched from memory and they fill a 6-byte First In First Out (FIFO) queue.

EXECUTION UNIT:

The execution unit consists of: General purpose (scratch pad) registers AX, BX, CX and DX; Pointer registers SP (Stack Pointer) and BP (Base Pointer); index registers source index (SI) & destination index (DI) registers; the Flag register, the ALU to perform operations and a control unit with associated internal bus. The 16-bit scratch pad registers can be split into two 8-bit registers. AX ⇒ AL, AH ; BX ⇒ BL, BH; CX ⇒ CL, CH; DX ⇒ DL, DH.



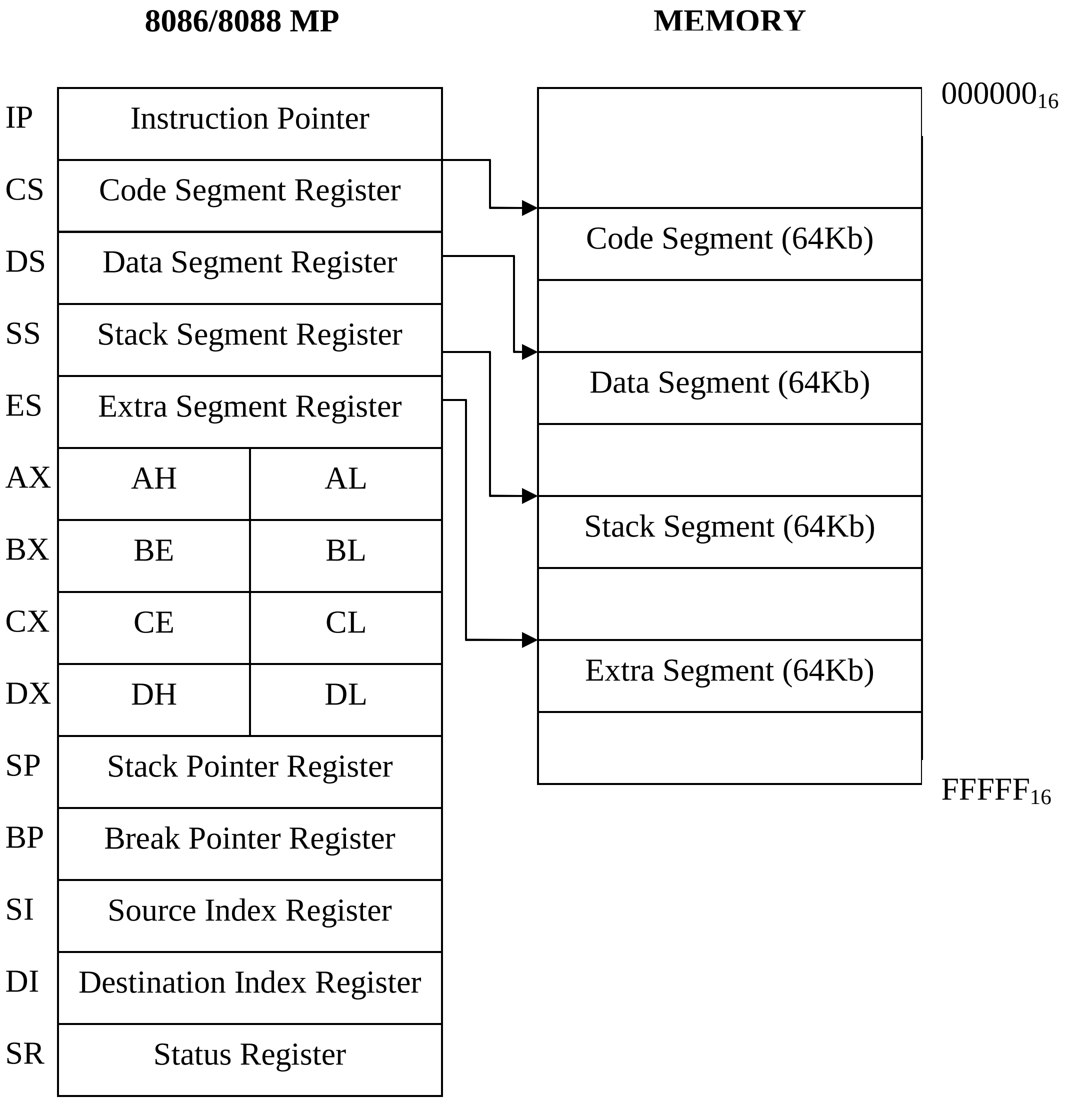


Note: All registers are of size 16-bits.

Different registers and their operations are listed below:

Register **Uses/Operations**

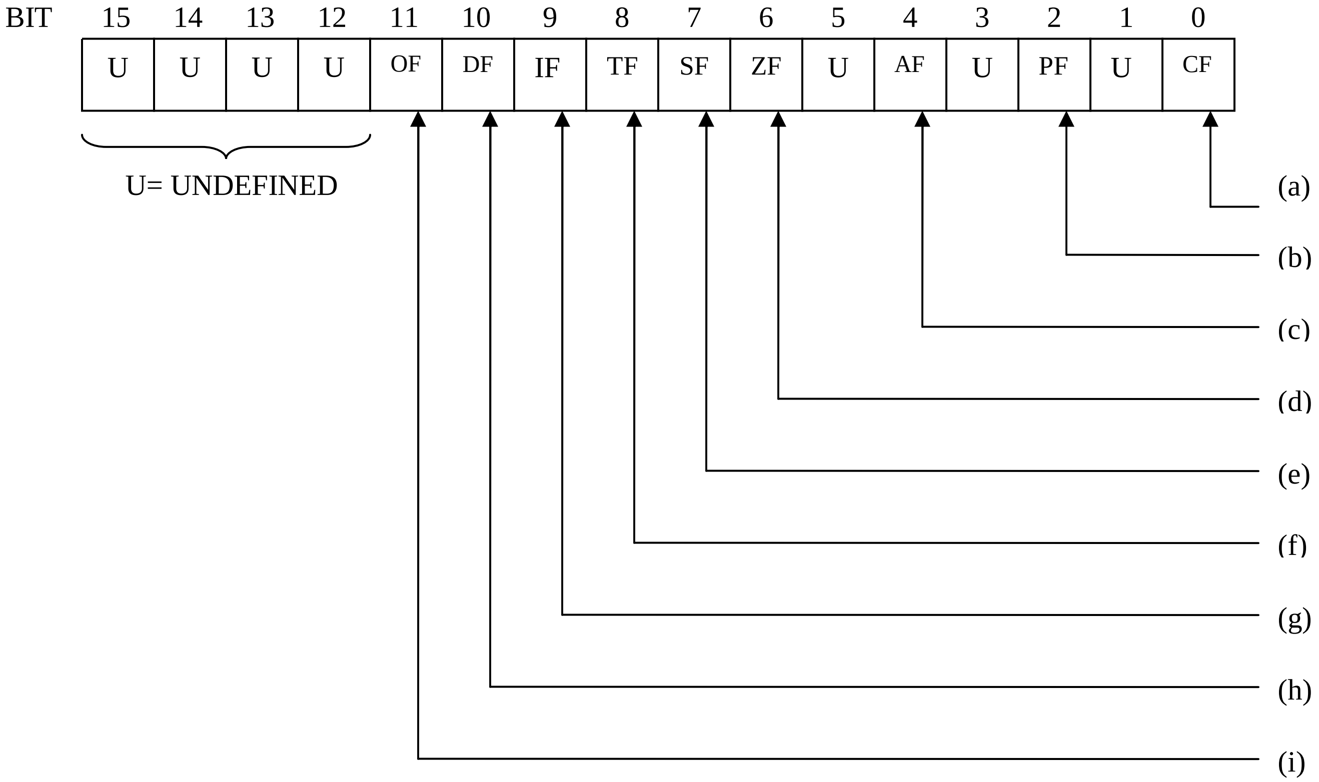
AX	As accumulator in Word multiply & Word divide operations, Word I/O operations
AL	As accumulator in Byte Multiply, Byte Divide, Byte I/O, translate, Decimal Arithmetic
AH	Byte Multiply, Byte Divide
BX	As Base register to hold the address of memory
CX	String Operations, as counter in Loops
CL	As counter in Variable Shift and Rotate operations
DX	Word Multiply, word Divide, Indirect I/O



The Execution of Instructions in 8086:

The microprocessor sends OUT a 20-bit physical address to the memory and fetches the first instruction of a program from the memory. Subsequent addresses are sent OUT and the queue is filled up to 6 bytes. The instructions are decoded and further data (if necessary) are fetched from memory. After the execution of the instruction, the results may go back to memory or to the output peripheral devices as the case may be.

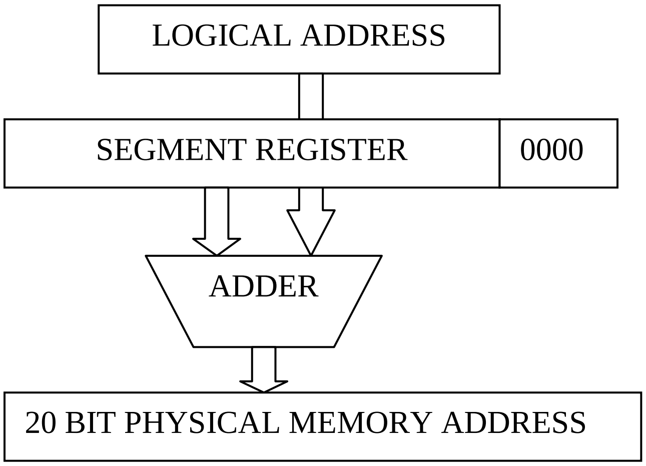
8086 Flag Register format



BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

(a) : CARRY FLAG – SET BY CARRY OUT OF MSB
(b) : PARITY FLAG – SET IF RESULT HAS EVEN PARITY
(c) : AUXILIARY CARRY FLAG FOR BCD
(d) : ZERO FLAG – SET IF RESULT = 0
(e) : SIGN FLAG = MSB OF RESULT
(f) : SINGLE STEP TRAP FLAG
(g) : INTERRUPT ENABLE FLAG
(h) : STRING DIRECTION FLAG
(i) : OVERFLOW FLAG

Generation of 20-bit Physical Address:



Programming Models:

Depending on the size of the memory the user program occupies, different types of assembly language models are defined.

TINY	⇒ All data and code in one segment
SMALL	⇒ one data segment and one code segment
MEDIUM	⇒ one data segment and two or more code segments
COMPACT	⇒ one code segment and two or more data segments
LARGE	⇒ any number of data and code segments

To designate a model, we use ".MODEL" directive.

Assembly Language Development Tools:

1. EDITOR:

- It's a system software (program) which allows users to create a file containing assembly instructions and statements. Ex: Wordstar, DOS Editor, Norton Editor
- Using the editor, you can also edit/delete/modify already existing files.
- While saving, you must give the file extension as ".asm".
- Follow the AL syntax while typing the programs
- Editor stores the ASCII codes for the letters and numbers keyed in.
- Any statement beginning with semicolon is treated as comment.

When you typed all your program, you have to save the file on the disk. This file is called "source" file, having a '.asm' extension. The next step is to convert this source file into a machine executable '.obj' file.

2. ASSEMBLER:

- An “assembler” is a system software (program) used to translate the assembly language mnemonics for instructions to the corresponding binary codes.
- An assembler makes two ‘passes’ thro’ your source code. On the first pass, it determines the displacement of named data items, the offset of labels etc., and puts this information in a symbol table. On the second pass, the assembler produces the binary code for each instruction and inserts the offsets, etc., that is calculated during the first pass. The assembler checks for the correct syntax in the assembly instructions and provides appropriate warning and error messages. You have to open your file again using the editor to correct the errors and reassemble it using assembler. Unless all the errors are corrected, the program cannot be executed in the next step.
- The assembler generates two files from the source file; the first file, called the object file having an extension “.obj” which contains the binary codes for instructions and information about the addresses of the instructions. The second file is called “list file” with an extension “’.lst”. This file contains the assembly language statements, the binary codes for each instruction, and the offset for each inst. It also indicates any syntax errors or typing errors in the source program.

Note: The assembler generates only offsets (i.e., effective addresses); not absolute physical addresses.

3. LINKER:

- It’s a program used to join several object files into one large object file. For large programs, usually several modules are written and each module is tested and debugged. When all the modules work, their object modules can be linked together to form a complete functioning program.
- The LINK program must be run on “.obj” file.
- The linker produces a link file which contains the binary codes for all the combined modules. The linker also produces a link map file which contains the address information about the linked files.
- The linker assigns only relative addresses starting from zero, so that this can be put anywhere in physical primary memory later (by another program called ‘locator’ or ‘loader’). Therefore, this file is called relocatable. The linker produces link files with “.exe” extension.
- Object modules of useful programs (like square root, factorial etc) can be kept in a “library”, and linked to other programs when needed.

4. LOADER:

- It’s a program used to assign absolute physical addresses to the segments in the “.exe” file, in the memory. IBM PC DOS environment comes with EXE2BIN loader program. The “.exe” file is converted into “.bin” file.

- The physical addresses are assigned at run time by the loader. So, assembler does not know about the segment starting addresses at the time program being assembled.

5. DEBUGGER:

- If your program requires no external hardware, you can use a program called debugger to load and run the ".exe" file.
- A debugger is a program which allows you to load your object code program into system memory, execute the program and troubleshoot or debug it. The debugger also allows you to look at the contents of registers and memory locations after you run your program.
- The debugger allows you to change the contents of registers & memory locations and rerun the program. Also, if facilitates to set up "breakpoints" in your program, single step feature, and other easy-to-use features.
- If you are using a prototype SDK 86 board, the debugger is usually called "monitor program".

We would be using the development tool MASM 5.0 or higher version from Microsoft Inc. MASM stands for Microsoft Macro Assembler. Another assembler TASM (Turbo Assembler) from Borland Inc., is also available.

8255 Programmable Peripheral Interface:

8255 is a programmable peripheral IC which can be used to interface computer (CPU) to various types of external peripherals such as: ADC, DAC, Motor, LEDs, 7-segment displays, Keyboard, Switches etc. It has 3 ports A, B and C and a Control word register. User can program the operation of ports by writing appropriate 8-bit "control word" into the control word register.

Control Word format

Bits →	D7	D6	D5	D4	D3	D2	D1	D0
	1 for I/O	PA mode: 00 – mode 0, 01 – mode1, 10/11 – mode 2		PA direction 0 – output 1 – input	PCU direction 0 – output 1 – input	PB mode 0 – mode 0 1 – mode 1	PB direction 0 – output 1 – input	PCL direction 0 – output 1 – input

How to Write and Execute your ALP using MASM?

Steps to be followed:

1. Type EDIT at the command prompt (C:\>\MASM\). A window will be opened with all the options like File, Edit etc., In the workspace, type your program according to the assembly language syntax and save the file with a ".asm" extension. (say test.asm)

2. Exit the Editor using File menu or pressing ALT + F + X.

3. At the prompt, type the command MASM followed by filename.asm (say, test.asm). Press Enter key 2 or 3 times. The assembler checks the syntax of your program and creates ".obj" file, if there are no errors. Otherwise, it indicates the error with line numbers. You have to correct the errors by opening your file with EDIT command and changing your instructions. Come back to DOS prompt and again assemble your program using MASM command. This has to continue until MASM displays "0 Severe Errors". There may still be "Warning Errors". Try to correct them also.

4. Once you get the ".obj" file from step 3, you have to create the ".exe" file. At the prompt, type the command LINK followed by "filename.obj" (say, test.obj) and press Enter key. (Note that you have to give the extension now as ".obj" and not as ".asm"). If there are no linker errors, linker will create ".exe" file of your program. Now, your program is ready to run.

5. There are two ways to run your program.

a) If your program accepts user inputs thro' keyboard and displays the result on the screen, then you can type the name of the file at the prompt and press Enter key. Appropriate messages will be displayed.

b) If your program works with memory data and if you really want to know the contents of registers, flags, memory locations assigned, opcodes etc., then type CV test (file name) at the prompt. Another window will be opened with your program, machine codes, register contents etc., Now, you also get a prompt > sign within CV window. Here you can use "d" command to display memory contents, "E" command to enter data into memory and "g" command to execute your program. Also, you can single step thro' your program using the menu options. In many ways, CV (Code View) is like Turbo C environment.

Once you are familiar with the architecture and basics of assembly language tools, you can start typing and executing your program.

Instructions for Laboratory Exercises:

1. The programs with comments are listed for your reference. Write the programs in observation book.
2. Create your own subdirectory in the computer. Edit (type) the programs with program number and place them in your subdirectory. Have a copy of MASM.EXE, CV.EXE and LINK.EXE files in your subdirectory. You can write comments for your instructions using Semicolon (;) symbol.
3. Execute the programs as per the steps discussed earlier and note the results in your observation book.
4. Make changes to the original program according to the questions given at the END of each program and observe the outputs.
5. For part A programs, input-output is through computer keyboard and monitor or through memory.
6. For part B programs, you need an external interface board. Connect the board to the computer using the FRC available. Some boards may require external power supply also.
7. Consult the Lab In-charge/Instructor before executing part B experiments.
8. The assembler is not case sensitive. However, we have used the following notation: uppercase letters to indicate register names, mnemonics and assembler directives; lowercase letters to indicate variable names, labels, segment names, and models.

Title 1a. Search a Key element in a list of N 16-bit numbers using binary search algorithm

```
.model small             ; memory model

.stack                   ; stack segment

.data                    ; data segment area. Define all variables and messages here

  arr DW 1111H, 2112H, 3113H, 4114H, 0a115H
  len DW ($-ARR)/2
  key EQU 2113H

  msg1 DB 10,13,  "KEY IS FOUND AT  "
  res  DB    "    POSITION ", 13,10, "$"
  msg2 DB  10,13, 'KEY NOT FOUND! $'

.code                            ; code segment. Put all instructions in this segment.

          MOV AX, @data          ; data segment initialization
          MOV DS, AX

        MOV BX, 00               ; pointing to first element
        MOV DX, len              ; pointing to last element
        MOV CX, key

 again: CMP BX, DX                ; compare first and last element indexes
          JA fail                ; conditional jump instruction

          MOV AX, BX             ; calculating the mid of the array
          ADD AX, DX
          SHR AX, 1
          MOV SI, AX
          ADD SI, SI

          CMP CX, arr [SI]       ; compare key with mid element
          JAE big

          DEC AX                  ; search elements below mid
          MOV DX, AX              ; high=mid-1
          JMP again              ; unconditional jump to repeat the above instructions

  big:    JE success

            INC AX                ; search elements above mid
            MOV BX, AX             ; low=mid+1
           JMP again
```

```
success: ADD AL, 01              ; element found. Get the position
            ADD AL, 30h           ; convert to ASCII
            MOV res, AL
            LEA DX, msg1        ; display the position
            JMP disp

   fail:     LEA DX, msg2        ; element not found

  disp:     MOV AH, 09H         ; DOS software interrupt to display the message
             INT 21H

         MOV AH, 4CH            ; DOS software interrupt
         INT 21H                  ; to terminate the program

   END
```

title 1b Read status of 8 input bits from the logic controller interface & display FF if it is even parity bits otherwise display 00. Also display number of 1's in the input data.

```
.model small

.stack

.data

       pa EQU 0d400h                  ; Addressing 8255 ports A, B and C
       pb EQU 0d401h
       pc EQU 0d402h
       cr EQU 0d403h                  ; Addressing 8255 Control Register

       cw EQU 82h                     ; Control Word for 8255. Make PA as output and PB as input

       msg DB 10,13,  "Number of 1's  =  $"

.code                                             ; main program
                  ; the first two instructions mandatory for all programs.

       MOV AX, @data             ; initialize data segment register
       MOV DS, AX

       MOV DX, cr                 ; Initialization of 8255
       MOV AL, cw
       OUT DX, AL

       MOV DX, pb                 ; Reading Logic Controller switch status thro' port B of 8255
       IN AL, DX
       OR AL, AL                    ; To affect the Parity Flag. The value in AL is not changed

       MOV BL, AL                   ; switch status in BL
       JPO oddp

          MOV DX, pa              ; Parity Even
          MOV AL, 0ffh            ; FF sent to Logic Controller
          OUT DX, AL

          JMP count

oddp:     MOV DX, pa
          MOV AL, 00h
          OUT DX, AL

          JMP count

count:   MOV CL, 08               ; Maximum number of switches =8
         MOV BH, 00               ; To count Number of 1s (BH) in the input
         MOV CH, 00
 back:    SHR BL, 1               ; check how many switches are closed by checking BL
         JNC skip                 ;  repeat 8 times.
         INC BH
 skip:   LOOP back               ; LOOP instruction decrements CX reg and goes to label if CX ≠ 0.

          MOV DX, OFFSET msg         ; Display the message  using DOS interrupt
          MOV AH, 09h
          INT 21h

          ADD BH, 30H             ; convert the number in BH to ASCII
          MOV DL, BH
          MOV AH, 02h             ; display the number using DOS interrupt
          INT 21h

          MOV AH, 4ch
          INT 21h
 END
```

Exercise questions:

1. **Modify prob 1a for a set of N 8-bit numbers.**
2. **Modify prob 1a to accept the 'key' value from memory.**
3. **Modify prob 1b to display messages 'even parity' and 'odd parity' on the screen**
4. **Name different search algorithms.**
5. **Write the block diagram of 8255 PPI and explain.**
6. **Write the control word format of 8255 and explain.**

%%%

Title   2a. Write ALP macros
; (1) To read a character from the keyboard in module 1 (file 1)
; (2) To display a character in module 2 (file 2)
; (3) Use the above two modules to read a string of characters terminated by the
; carriage return and print the string on the display in the next line

```
INCLUDE readch.mac              ; include the file readch.mac
INCLUDE dispch.mac              ; include the file dispch.mac
.model small
.stack                                          ; optional declaration
.data
  arr DB  40 DUP (?)                            ; declaring an array to store 40 bytes
  msg1 DB   10,13, "Enter the String :  $"
  msg2 DB 13,10, "The Entered String is :  $"

.code                                   ; main program
                                        ; label for the first instruction is optional.
   start: MOV AX, @data
          MOV DS, AX

         LEA DX, msg1                   ; display a string on screen using DOS Interrupts
         MOV AH, 09h
         INT 21h

          MOV SI, 0                     ; array index to store the character read from keyboard
back:    read arr [SI]                  ; Macro invoked to read a character
          INC SI                         ; and stored in array

         CMP AL, 13                      ; If carriage return goto display
         JNZ back

          LEA DX, msg2
          MOV AH, 09h
          INT 21h

          MOV SI, 0
again:   disp  arr[SI]                  ; Macro invoked to display a character on the screen
           INC SI
          CMP AL, 13                    ; until the carriage return
          JNZ again
          MOV AH, 4CH
          INT 21H
END start                       ; if label is given at the beginning, END must be followed by label


; following codes are written separately having filename     dispch.mac  and  readch.mac
```

```
 disp MACRO  var                        ; macro definition to display a character on the screen

    MOV DL, var
    MOV AH, 02h                         ; DOS Software interrupt  to display a character on screen
    INT 21H
  ENDM
```

```
read MACRO c                 ; macro definition to read a character from keyboard

    MOV AH, 01h              ; DOS Software interrupt  to read a character from keyboard
    INT 21h
    MOV c, AL                ; ASCII value of character stored in variable c
ENDM
```

```
title       2b Perform the BCD up/down (00-99-00) counter and   ring counter operations
            ; using Logic Controller

.model small

.stack

.data

  pa EQU 0d400h
  pb EQU 0d401h
  pc EQU 0d402h
  cr EQU 0d403h

  cw EQU 82h

.code                                   ; main program

          MOV AX, @data
          MOV DS, AX

       MOV DX, cr
       MOV AL, cw                       ; set 8255 port B  as input and port A as output
       OUT DX, AL

       MOV BL, 00h                      ; BL holds count for BCD up and down counter
       MOV BH, 01h                      ; BH used for Ring Counter

 again:  MOV DX, pb                     ; read switch position from Logic controller using port B
          IN AL, DX

          CMP AL, 0ff h                  ; set ff input on the interface board using switches for UP
counter
```

```
           JE up
           CMP AL, 0fe h              ; set fe input for BCD down counter
           JE down

          CMP AL, 0fc h               ; set fc input for Ring Counter
           JE ring

       MOV AH, 4c h                   ; terminate program if any other switch input is given
       INT 21h
  up:  MOV AL,BL                      ; BCD UP counter
        CALL disp                     ; Transfer control to a procedure named as disp
        ADD AL, 1                       ;incrementing count
        DAA                              ; change result to decimal after addition
        MOV BL, AL
       JMP again                        ; go back and check switch inputs

 down: MOV AL,BL                      ; BCD DOWN counter
         CALL disp
         SUB AL, 1                     ;decrementing count
         DAS                          ; change result to decimal after subtraction
         MOV BL, AL
         JMP again

 ring:    MOV AL, BH                  ; Ring counter operation
          CALL disp
          ROR BH, 1                    ;shifting bit to the right
          JMP again

 disp  PROC  NEAR                     ; procedure to display result on 8255 port
       MOV DX, pa
       OUT DX, AL
       CALL delay                     ; call another procedure named as delay
       RET                            ; return to the calling program
disp  ENDP

delay   PROC   NEAR                     ; Delay procedure to wait for few sec
          PUSH CX                       ; save original contents of AX and CX registers on stack
          PUSH AX
          MOV CX, 2000 h                 ; count for outer loop in CX
back1:   MOV AX, 0ffff h                 ; count for inner loop in AX
back2:   DEC AX
           JNZ back2
           LOOP back1
           POP AX                     ; retrieve original contents of AX and CX before returning
           POP CX
           RET                        ; return back to called program
delay ENDP                            ;End of delay procedure
END
```

Exercise questions:

1. **Modify prob 2a to accept a string ending with $ sign.**
2. **Modify prob 2a with only CR or LF values and observe the output.**
3. **Modify prob 2b to have only two options: UP/Down or Ring counter**
4. **Modify prob 2b delay procedure for different delays by varying count value.**
5. **Modify prob 2b for HEX up/down counter and shift left ring counter.**

%%%

Title 3a Sort a given set of N 8-bit numbers in ascending order and ; descending order using bubble sort algorithm

```
.model small

.stack

.data

 list DB  33h, 54h, 0a2h, 17h, 76h             ; declare and initialize an array of bytes
 n   DW $-list                                 ; length of the array

 order EQU 0                           ; order = 0 for ascending (assumed)
                                        ; order = 1 for descending

 msg DB   'THE SORTED ARRAY IS::  $'

.code                                          ; main program

            MOV AX, @data
            MOV DS, AX

            MOV BX, n                           ; length of the array (n) in BX reg
            DEC BX                              ; n-1 value in BX

nextpass:  MOV CX, BX                          ; n-1 value in CX
           MOV SI, 00H                         ; SI used for indexing into the array

nextcomp:    MOV AL, list[SI]          ; take an element from the array in AL register
             INC SI
             CMP AL, list[SI]          ; comparing elements

            IF order EQ 0              ; conditional assembly
            JBE next                   ; ascending order. Check CY and Z flags.
            ELSE
            JAE next                  ; descending order
            ENDIF
            XCHG AL, list [SI]          ; exchange elements if required
            MOV list [SI-1], AL

    next:  LOOP nextcomp                  ; inner loop
           DEC BX
           JNZ nextpass                 ;outer loop
                                        ; sorting is over

          LEA DX, msg                   ; display the message
          MOV AH, 09h
          INT 21H
                                  ; Below instructions are to display the elements on screen
          MOV BX, n
          MOV SI, 00                    ; SI as pointer to the array element

 again: MOV AL, LIST[SI]                ; take the element from the array into AL

          CALL  unpack                  ; use procedure to unpack the digits of the nubmer

          MOV AH, 02h                     ; keep space between elements
          MOV DL,'   '
          INT 21H

          INC SI
          DEC BX
          JNZ again                     ; repeat for all elements in the array

          MOV AH, 4Ch
          INT 21H

unpack    PROC   NEAR                           ; procedure to unpack the digits
        MOV CH, AL
        AND AL, 0F0h                    ; mask higher nibble (digit) of the number
        MOV AH, AL
        MOV CL, 4
        SHR AH, CL                      ; interchange (swap) the digits
        CALL asciidisp                  ; call procedure to convert to ascii and display the numbers
        MOV AL, CH
        AND AL, 0Fh                     ; mask lower nibble of the number
        MOV AH, AL
        CALL disp                       ; call procedure to convert to ascii and display the numbers
        RET
unpack   ENDP


asciidisp   PROC    NEAR                ; procedure to convert to ascii and display the numbers
          CMP AH, 0Ah                   ; if digit is 0-9, ADD 30 to convert to ASCII
          JB skip                       ; if digit is A-F, ADD 37 to convert to ASCII
          ADD AH, 7
skip:     ADD AH, 30h
          MOV DL, AH
          MOV AH, 02
          INT 21 h
          RET
asciidisp   ENDP

          MOV AH, 4Ch                    ; terminate
          INT 21h

END
```

Title 3b) read the status of two 8-bits inputs (x & y) from the logic controller
; interface and display x * y.

```
.model small
.stack

.data
  pa EQU 0d400h                          ; 8255 port addresses
  pb EQU 0d401h
  pc EQU 0d402h
  cr EQU 0d403h

  cw EQU 82h

      msg1 DB  10,13, "enter number x from the interface and press Enter :$"
      msg2 DB 10, 13,  "enter number y from the interface and press Enter :$"
      msg3 DB 10, 13, "Product is displayed on the interface in binary form: press any key to exit$"

.code                                          ; main program
                                               ;start label is optional
 start: MOV AX, @data
        MOV DS, AX

       MOV DX, cr                              ;initialize 8255 ports
       MOV AL, cw
       OUT DX, AL

       MOV DX, OFFSET msg1                     ;display message
       MOV AH, 09h
       INT 21h

       MOV AH, 01h                             ; press any key to continue
       INT 21h

       MOV DX, pb                       ; read first number (switch status) through port B of 8255
```

```
        IN AL, DX
        MOV BL, AL                          ; first number copied to BL reg
        MOV DX, OFFSET msg2                 ; display next message
        MOV AH, 09h
        INT 21h
        MOV AH, 01h                         ; press a key to continue
        INT 21h

        MOV DX, pb                          ;read second number
        IN AL, DX
                                            ; both data read. Now multiply them
        MUL BL                              ; 16-bit product in AX reg
        MOV DX, pa
        OUT DX, AL                          ; send lower digit of the product
        MOV BL, AH

        MOV DX, OFFSET msg3                 ;display message
        MOV AH, 09h
        INT 21h

        MOV AH, 01H                         ;press any key
        INT 21h
        MOV AL, BL                          ; send the higher digit
        MOV DX, pa
        OUT DX, AL
        MOV AH, 4Ch                         ;terminate
        INT 21h

delay   PROC   NEAR                         ; delay procedure

          PUSH CX
          PUSH AX
          MOV CX, 2000h
back1:  MOV AX, 0ffffh
back2:  DEC AX
         JNZ back2
         LOOP back1
         POP AX
         POP CX
         RET
delay     ENDP
END start
```

Exercise questions:

1. **Modify prob 3a for a set of N 16-bit numbers.**
2. **Modify prob 3b to obtain the product in decimal and display it.**
3. **Name different sorting algorithms.**

%%

title   7a) Read your name from the keyboard and display it in a specified location
;   on the screen in front of the message "what is your name". Clear the screen
;   before display.

```
.model small

.stack

readstr   MACRO  loc                    ; macro to read a character
       MOV AH, 01H
       INT 21H
       MOV loc, AL
ENDM
clrscr   MACRO                          ; macro to blank the screen
       MOV AL, 2                         ; clear the screen using BIOS interrupt
       MOV AH, 0
       INT 10H
ENDM

 .data
       msg0 DB 10, ' ENTER THE NAME:$'
       msg1 DB      ' WHAT IS YOUR NAME? $'
       msg2 DB  10, '$'                          ;insert line feed

       len DW ($-msg1)
       arr DB 40 DUP(?)

display MACRO str                                ; macro definition to display a string on screen
         LEA DX, str
         MOV AH,9
         INT 21H
ENDM

.code                                            ; main program

start:   MOV AX, @data
          MOV DS, AX
          MOV SI, 00                             ; SI is array pointer

       display msg0                              ;invoke macro to display message


back:   readstr arr[SI]                 ;READ NAME FROM THE KEYBOARD
          INC SI
          CMP AL, 13                    ; and store in array
          JNZ back
          MOV arr[SI],'$'                        ; END of string character inserted
```

```
        clrscr                          ; invoke macro to clear screen

                                        ;Position The Cursor  on the screen
        MOV BH, 0                       ; using BIOS interrupt
        MOV DH, 13                      ; row coordinate
        MOV DL, 28                      ;column coordinate
        MOV AH, 2
        INT 10H

        display msg1                            ;invoke macro to display message

        MOV SI, 0                               ;index to the array
        LEA DX, arr[SI]                 ; read name from the array and display
        MOV AH, 09H
        INT 21H

         display msg2                           ;invoke macro to display message

        MOV AH, 4CH
        INT 21H

END start
```

____-

```
   title    7B) Drive a stepper motor interface to rotate the motor in clockwise direction
                ;  by N steps

.model small

.stack

.data
  pa EQU  0d400h                        ;Addressing 8255 ports
  pb EQU  0d401h
  pc EQU  0d402h
  cr EQU  0d403h                        ;Addressing 8255 Control Register

  cw EQU 80h                            ;Control Word for 8255 for making all ports as output
  n  EQU     50                         ; no of rotations. N=50 is one rotation
  PHASE_A   EQU     88H                 ; pattern to energize the windings of motor

.code

start:  MOV DX, cr
        MOV AL, cw
        OUT DX, AL
        MOV CX, n
```

```
again:   MOV BL, 4
          MOV AL, PHASE_A              ; load pattern into AL
up:      MOV DX, pa
         OUT DX, AL                    ; energize winding of the motor
         CALL delay

          ROR AL, 1                    ; clockwise rotation
          DEC BL
          JNZ up
          LOOP again

         MOV AH, 4CH
         INT 21H

delay    PROC      NEAR                ; delay procedure
            MOV SI, 1000h
back2:     MOV DI, 0FFFH
back1:     DEC DI
            JNZ back1
            DEC SI
             JNZ back2
             RET
delay    ENDP

END start
```

Exercise questions:

1. **Modify prob 7a to display the name character-by-character.**
2. **Write a code to clear the screen.**
3. **Modify prob 7b to rotate motor (i) 2 rotations (ii) 5 rotations**
4. **Modify prob 7b delay counts and observe the speed of the motor.**
5. **Write a note on DOS interrupts.**

%%%

title 8a) Compute Factorial of Positive Integer 'N' Using Recursive Procedure

```
.model small

.stack

.data
       num DW  5                          ; number whose factorial is needed
       res   DW  ?                        ; to store the result

       msg  DB  10,13, "THE FACTORIAL OF "
       msg1 DB         "    IS: $"
       msg2 DB 10,13,  'factorial of 0 is 1 $'

display   MACRO   str                   ; macro definition to display a string on screen
         LEA DX, str
         MOV AH, 9
         INT 21H
ENDM
.code                                           ; main program

       MOV AX, @data
       MOV DS, AX
       MOV CX, num
       ADD CX, 3030h
       MOV msg1, CL                     ; store the ASCII value of number in memory

       CMP num, 0                       ; if number is 0, factorial is 1
       JE last                          ; else compute the factorial

       MOV AX, 01H
       CALL fact                        ; transfer control to procedure named fact
       MOV res, AX                      ; result copied to memory

       display msg                      ; invoke macro to display the message


       MOV AX, res                      ; use a procedure to unpack the digits of result
       CALL unpack
       JMP stop

last:   display msg2                    ; invoke macro to display message

stop:   MOV AH,4CH
        INT 21H

fact    PROC   NEAR                     ; procedure to find factorial
        MUL num
        DEC NUM
        JZ  over
        CALL fact                       ; recursively call the same procedure
over:     RET                            ; result in AX register
 fact   ENDP

unpack   PROC   NEAR                    ; procedure to unpack the digits

       MOV BX, AX
       AND AH, 0F0H                     ; mask leftmost digit (MSD)
       MOV AL, AH
       MOV CL, 4
```

```
        SHR AL, CL
        CALL asciidisp                   ; use another procedure to convert to ASCII and display
        MOV AX, BX
        AND AH, 0FH                      ; mask next digit
        MOV AL, AH
        CALL disp
        MOV AX, BX
        AND AL, 0F0H                     ; mask next digit
        SHR AL, CL
        CALL disp
        MOV AX, BX
        AND AL, 0FH                      ; maks rightmost digit (LSD)
        CALL disp
        RET

unpack   ENDP

asciidisp    PROC    NEAR                ; procedure to obtain ascii value
         CMP AL, 0AH                     ; and to display the number on the screen
         JB skip
         ADD AL, 7
skip:    ADD AL, 30H
         MOV DL, AL
         MOV AH, 02
         INT 21H
         RET
asciidisp    ENDP

END
```

---

title 8B) Drive a stepper motor interface to rotate the motor in anti-clockwise ; direction by N steps

```
.model small

.stack

.data
  pa EQU   0d400h                        ;Addressing 8255 ports
  pb EQU   0d401h
  pc EQU   0d402h
  cr EQU   0d403h                        ;Addressing 8255 Control Register

  cw EQU  80h                  ;Control Word for 8255 for making all ports as output
  n  EQU     50                ; no of rotations. N=50 is one rotation
 PHASE_A EQU     88H                     ; pattern to energize the windings of motor
```

```
.code                                ; main program
start:  MOV DX, cr
        MOV AL, cw
        OUT DX, AL
        MOV CX, n

again:   MOV BL, 4
         MOV AL, PHASE_A             ; load pattern into AL
up:      MOV DX, pa
         OUT DX, AL                  ; energize winding of the motor
         CALL delay                  ; wait before sending pulse to next winding
        ROL AL, 1
        DEC BL
        JNZ up
        LOOP again                   ; repeat for all n steps
        MOV AH, 4CH
        INT 21H

delay   PROC  NEAR
           MOV SI, 2000h
back2:     MOV DI, 0FFFH
back1:     DEC DI
           JNZ back1
           DEC SI
           JNZ back2
           RET
delay    ENDP
END   start
```

Exercise questions:

1. **Modify prob 8a to get the factorial without using a recursive procedure.**
2. **Modify prob 8a to display the factorial of 0 and 1 without computing and for other numbers (2 - 8) it should compute.**
3. **Modify prob 8a to check for the input >8 and display an error condition.**
4. **Is it possible to rotate the motor in prob 8b without using ROL instruction? If yes, write the complete code.**
5. **Write a note on BIOS interrupts.**

%%%

```
title   9a) COMPUTE nCr USING RECURSION PROCEDURE. ASSUME THAT 'n' AND 'r' ARE
        ;   NON NEGATIVE INTEGER NUMBERS.

.model small
.stack
.data
```

```
        n DW  5                         ; value of n
        r DW  3                         ; value of r
        ncr DW 1                        ; to store the result

        msg DB 10,13, "The nCr is: $"
        msg1 DB 10,13, " error! n value cannot be zero! $"

display   MACRO   str                   ; macro definition to display a string on screen
           LEA DX, str
           MOV AH, 9
           INT 21H
ENDM
.code
                                        ; main program
start:  MOV AX, @data
        MOV DS, AX

         CMP n, 0                       ; if n=0, error condition
         JZ  error

         MOV BX, n                      ; BX has value of n
          INC BX
          MOV CX, r                     ; CX has value of r
          CALL ncp                      ; transfer control to procedure

         display msg                    ; invoke macro to display the message

         MOV AX, ncr                    ; copy result into AX reg
         CALL unpack                    ; use procedure to unpack the digits
         JMP  stop

error:   display msg1                           ; invoke macro to display message

stop:    MOV AH,4CH
          INT 21H

ncp     PROC     NEAR                           ; procedure to find ncr value
        CMP CX, 00H
        JE over                                 ; if r=0, ncr value is 1
        PUSH CX
        DEC CX
        CALL NCP
        MOV AX,BX
        POP CX
        SUB AX,CX
        MUL NCR
        DIV CX
        MOV NCR,AX
```

```
over:   RET
ncp  ENDP

unpack   PROC   NEAR                            ; procedure to unpack the digits
        MOV BX, AX
        AND AH, 0F0H                  ; mask leftmost digit (MSD)
        MOV AL, AH
        MOV CL, 4
        SHR AL, CL
        CALL disp                     ; use another procedure to convert to ASCII and display
        MOV AX, BX
        AND AH, 0FH                   ; mask next digit
        MOV AL, AH
        CALL disp
        MOV AX, BX
        AND AL, 0F0H                  ; mask next digit
        SHR AL, CL
        CALL disp
        MOV AX, BX
        AND AL, 0FH                   ; maks rightmost digit (LSD)
        CALL disp
        RET
unpack    ENDP

disp    PROC    NEAR                  ; procedure to obtain ascii value
          CMP AL, 0AH                 ; and to display the number on the screen
          JB skip
          ADD AL, 7
skip:   ADD AL, 30H
          MOV DL, AL
          MOV AH, 02
          INT 21H
          RET
disp    ENDP

 END  start
```

---

title 9B) Drive a stepper motor interface to rotate the motor N steps clockwise
; and N steps in anti-clockwise direction

```
.model small

.stack

.data
  pa EQU  0d400h                      ;Addressing 8255 ports
  pb EQU  0d401h
  pc EQU  0d402h
  cr EQU  0d403h                          ;Addressing 8255 Control Register
  cw EQU  80h                   ;Control Word for 8255 for making all ports as output
  n1  EQU  50
  n2  EQU  75                   ; no of rotations. N = 50 is one rotation

 PHASE_A EQU    88H             ; pattern to energize the windings of motor
 PHASE_D EQU    11H

.code
start:  MOV DX, cr
        MOV AL, cw
        OUT DX, AL

          MOV CX, n1                          ;for clockwise rotation of motor
clockw:  MOV BL, 4
           MOV AL, PHASE_A
up1:      MOV DX, pa
          OUT DX, AL
           CALL delay

           ROR AL, 1
           DEC BL
           JNZ up1
           LOOP clockw

            MOV CX, n2                        ; for anti-clockwise rotation of motor
anticlk:    MOV BL, 4
            MOV AL, PHASE_D
up2:       MOV DX, Pa
            OUT DX, AL
            CALL delay
             ROL AL, 1
            DEC BL
            JNZ up2
            LOOP anticlk
            MOV AH, 4CH
            INT 21H
delay    PROC   NEAR
            MOV SI, 1000h
back2:     MOV DI, 0FFFH
back1:     DEC DI
            JNZ back1
             DEC SI
             JNZ back2
             RET
delay      ENDP
END start
```

Exercise questions:

1. **Modify prob 9a with another logic (mathematically) for finding ncr.**
2. **Modify prob 9a to check r > n and if yes, print an error condition.**
3. **Modify prob 9b to rotate the motor either clockwise or anti-clockwise depending on the key pressed from keyboard.**
4. **What are the uses of Stepper motor?**

%%%

```
title     10a) Find whether a given Sub- string is present or not in a  main string of
             ; characters.

.model small

.stack

disp_msg   MACRO  str        ; macro to display string on screen
        LEA DX, str        ; using DOS interrupts
        MOV AH, 09h
        INT 21H
ENDM

read  MACRO  str            ; macro to read a string from keyboard
        LEA DX, str
        MOV AH, 0AH
        INT 21H
ENDM

.data
        msg1 DB 10, 13,  'ENTER THE MAIN STRING:$'
        msg2 DB 10, 13,  'ENTER THE SUB STRING:$'
        msg3 DB 10, 13, 10,'      Congrats!! THE SUB STRING IS FOUND: *** ', 10, '$'
        msg4 DB 10, 13, 10, '      Sorry!!THE SUB STRING IS NOT FOUND:!!! ', 10, '$'

          z DB 50H                          ; array to store main string
             DB 0H
             DB 50H DUP (?)
          y DB 50H                          ; array to store substring
             DB 0H
             DB 50H DUP (?)
.code

start:  MOV AX, @data
         MOV DS, AX
         disp_msg msg1                     ; invoke macro to display message
         read z                           ; invoke macro to read a main string
         disp_msg msg2
         read y                            ; invoke macro to read SUB string
```

```
           MOV CL, z+1              ; length of main string in CL reg
          LEA SI, z+2               ; point to the main string
back2:  PUSH SI
          LEA DI, y+2                 ; point to the substring
           MOV CH, y+1                ; length of SUB-string in CH reg
           MOV BH, 00H
back1:  MOV AL, [SI]
          CMP AL, [DI]                 ; compare characters of both strings
          JNE nextword                 ; if not equal, go for next word of string
           INC SI
          INC DI
          INC BH
          CMP BH, y+1                 ; repeat till all char in substring is compared
          JE found                      ; if all characters are equal, display msg1
          DEC CH
          JNZ back1

nextword:
           POP SI
           INC SI
           DEC CL
           CMP CL, 00H
           JNE back2                         ; after all comparsions, if not found display msg4
           disp_msg  msg4                  ; invoke macro
           JMP stop
found:   disp_msg  msg3                 ; invoke macro to display msg3

stop:      MOV AH,4CH
            INT 21H
END start
```

____

title   10b) Scan a 8 X 3 keypad for key closure and to store the code of the key
; pressed in memory location or display on the screen. Display row and column
; numbers of the key pressed.

```
.model small

.stack

clrscr MACRO                          ; macro definition to clear screen
       MOV AL, 2                      ; using BIOS interrupt function
       MOV AH, 0
       INT 10h
ENDM
```

```
mdisp   MACRO   str                          ; macro definition to display message
         LEA DX, str
         MOV AH,9
         INT 21H
ENDM
cdisp   MACRO                    ; macro definition to display character/number
        ADD AL, 30H
        MOV DL, AL
        MOV AH, 02h
        INT 21h
ENDM
OUT_pc     MACRO                         ; macro definition for output to keypad
        MOV DX, PC
        OUT DX, AL
ENDM
in_pa     MACRO                     ; macro definition to read from keypad
        MOV DX, pa
        IN AL,DX
ENDM

.data
    msg1  DB  'DEMONSTRATION PROGRAM FOR KEYBOARD INTERFACE' ,13,10,'$'
    msg2  DB  'press a key on keypad interface to know row and column number...', 10, 13, '$'
    msg3  DB  'This program is running...',13,10,'Press any key on computer to EXIT.',13,10,'$'
    msg4  DB  ' Key Pressed is : ','$'
    msg5  DB  13,'Row no:  ','$'
    msg6  DB  '  Column no: ','$'

    keys     DB  '0 1 2 3 4 5 6 7 8 9 . + - X / % ACCECK= MCMRM-M+','$'
    Show     DB  '01','$'

    pa EQU 0D400h
    pb EQU 0D401h
    pc EQU 0D402h
    cr EQU 0D403h
.code

start:  MOV AX, @data
        MOV DS, AX

         clrscr                          ; invoke macro to clear screen
         mdisp msg1                       ; invoke macro to display messages
        mdisp msg2
        mdisp msg3

        MOV AX, 90h                       ;Initialize Port A - Input, CU & CL - Output
        MOV DX, cr
        OUT DX, AX                       ;Write to Control Register of 8255
```

```
getkey:
            MOV BH, 1h                 ;Scan Lines
            MOV BL, 0h                 ;Initialize a counter. It contains the no of the Key

scanlines:

            MOV AL, BH
            OUT_pc                   ;invoke macro to send Line Number to Port CL
             in_pa                   ;invoke macro to read from Port A

            MOV CH, AL                 ; CH Has the value indicating the key pressed
            MOV AL, 0H

check:                                ; Initialize the counter
                                       ; Now Repeatedly check which key was selected.
        MOV CL, CH
        AND CL, 01h                   ; mask all bits except lsb
        CMP CL, 01h
        JZ display                    ; If that bit is set, key is pressed
        INC BL                       ; else check next bit by shifting the value of CH
        SHR CH, 01h
        INC AL
        CMP AL, 08h                   ; If alll bits are not compared,
        JNZ check                    ; go back for next scan line

        SHL BH, 01h                   ;Move to next scan line
        CMP BH, 10h
        JNZ scanlines                 ;Repeat the SCAN Lines Loop (4 times)
        JMP loopout

display:                              ; Display the selected key
         PUSH AX
         mdisp   msg5                ; invoke macro
         MOV AL, BH

          cdisp
          mdisp msg6
          POP AX
          cdisp
          mdisp msg4

         MOV AX, 0h
         MOV AL, BL
         MOV BL, 02h
         MUL BL
         MOV BX, AX
         MOV DI, OFFSET Show
         MOV AL, Keys[BX]
         MOV Show [0h], AL
         MOV AL, Keys [BX + 1h]
         MOV Show [1h],AL
                                             ;Display the character pressed.
         mdisp show
         CALL delay
loopout:

          MOV AH, 01h
          INT 16h                          ;press any key to exit
          JNZ next
          JMP getkey
next:    MOV AH,4ch                         ;Exit the program safely.
          INT 21h

delay    PROC    NEAR                     ; delay procedure
          MOV CX, 0FFFFh
back2:   MOV AX, 0FFh                       ; outer loop
back1:   DEC  AX                            ; inner loop
          JNZ back1
            LOOP back2
            RET
delay     ENDP

END start                                 ;this is the END of your program.
```

Exercise questions:

1. **Modify prob 10a to print the length of the main string and substring.**
2. **Modify prob 10b to display row number starting from 0.**

%%

title 11a) Generate first ‘n’ Fibonacci numbers

```
.model small

.stack

.data
       arr   DW  0,1,50 DUP (?)   ; array to store Fib numbers
;       arrdec DW  0,1, 50 DUP (?) ; array to store Decimal fib numbers
       count DW 15             ; how many numbers to generate

.code

start:  MOV AX, @data
         MOV DS, AX
```

```
          MOV SI, 0
          MOV CX, count
                              ; generating HEX fib numbers

back:    MOV AX, arr[SI]          ; take the first number from memory
          ADD AX, arr[SI+2]        ; ADD it to the second number
          MOV arr[SI+4], AX        ; store the sum in next location
          ADD SI, 2               ; increment pointer
          LOOP back               ; repeat until count is over
                              ; Hex result stored in memory


  ; for generating Decimal Fib numbers use the following code and
         use memory location arrdec.

  ;      LEA DI, arrdec
  ;      MOV CX, count
  ;      XOR AX, AX
;back:  MOV AL, BYTE PTR [DI]
  ;      ADD AL, BYTE PTR [DI+2]
  ;      DAA
  ;      MOV BYTE PTR [DI+4], AL
  ;      MOV AL, BYTE PTR [DI+1]
  ;     ADC AL, BYTE PTR [DI+3]
  ;      DAA
  ;      MOV BYTE PTR [DI+5], AL
  ;      ADD DI, 2
  ;      LOOP back

        MOV AH, 4Ch
        INT 21h
 END start
```

____--

TITLE 11b) Scan a 8 X 3 keypad for key closure and to simulate
; ADD and SUBRACT operations as in a calculator.

```
.model small

.stack

clrscr MACRO
        MOV AL, 3
        MOV AH, 0
        INT 10h
ENDM
```

```
mdisp    MACRO  str
          LEA DX, str
          MOV AH, 9
          INT 21H
ENDM
cdisp      MACRO
        MOV DL, AL
        MOV AH, 02h
        INT 21h
ENDM
.data

  pa  EQU  0d400h
  pb  EQU  0d401h
  pc  EQU  0d402h
  cr  EQU  0d403h

  msg1 DB 10,13, 'Input value X and press a key on computer keyboard:$'
  msg2 DB 10,13, 'Input value Y and press a key on computer keyboard:$'
  msg3 DB 10,13, 'Input operator: +/- $'
  msg4 DB 10,13, 'Result = $'

  disp1 DB  '0123456789.+-*/%c$'
  inp DB 4

.code

        MOV AX, @data
        MOV DS, AX

        MOV DX, cr
         MOV AL, 90h                    ;initialize 8255 port A as input
         OUT DX, AL                     ; and other ports as output

         MOV DI, OFFSET inp

         mdisp msg1
         CALL delay
         CALL delay
         CALL keypress                  ; use a procedure to get first number from keypad interface
        PUSH AX
        MOV AH, 07h                     ;press any key to continue
        INT 21h
        POP AX

         mdisp msg2
        INC DI
        CALL delay
        CALL delay
        CALL keypress           ; use a procedure to get second number from keypad interface

        PUSH AX
        MOV AH, 07h
        INT 21h
        POP AX

         mdisp msg3
        INC DI

        CALL delay
        CALL delay
        CALL keypress           ; use a procedure to get operator - or + from keypad interface

        PUSH AX
        MOV AH,  07h
        INT 21h
        POP AX

        mdisp msg4

        MOV AL, [DI-2]
        SUB AL, 30h
        MOV BL, [DI-1]
        SUB BL, 30h
        MOV DL, [DI]
        CMP DL, '+'                     ; check operator. IF +, do addition, if - do subtraction
        JNZ subt
        ADD AL, BL                      ; addition. result in decimal
        DAA
        JMP exit
 subt:  SUB AL, BL
          DAS                           ; subtraction. result in decimal

exit:   MOV DL, AL                      ; display the result on screen by converting to ASCII
        AND AL, 0f0h
        MOV CL, 04h
        SHR AL, CL
        ADD AL, 30h
        PUSH DX
        cdisp                           ; invoke macro
        POP DX
        MOV AL, DL
        AND AL, 0fh
        ADD AL, 30h
         cdisp                          ; invoke macro
        MOV AH, 4ch
        INT 21h

keypress   PROC  NEAR                ; procedure to scan keypad to read numbers and operator

repeat:     MOV DX, pc
            MOV AL, 01h              ; select a row of keypad
            OUT DX, AL
            MOV DX, pa                        ; read column of that row
            IN AL, DX
            CMP AL, 00
            JZ next                           ; if no key pressed, check in next row
            JMP f_c
next:       MOV DX, pc
            MOV AL, 02h
            OUT DX, AL
            MOV DX, pa
            IN AL, DX
            CMP AL, 00
            JNZ s_c
            JMP repeat
f_c:        CALL delay
            MOV SI, OFFSET disp1
next1:      SHR AL, 1
            JC nextc                          ; if key is pressed, display it
            INC SI
            JMP next1
s_c:        CALL delay
            MOV SI, OFFSET disp1
            ADD SI, 08h
next2:      SHR AL, 1
            JC nextc
            INC SI
            JMP next2
nextc:      MOV DL, [SI]
            MOV AH, 2h
            INT 21h
            MOV [DI], DL
            RET
keypress    ENDP

delay   PROC  NEAR                                  ;delay procedure
          PUSH AX
          PUSH CX
          MOV CX, 80h
back2:    MOV AX, 1000h
back1:    DEC AX
```

```
          JNZ back1
           LOOP back2
           POP CX
           POP AX
            RET
delay    ENDP

END start
```

Exercise questions:

1. **Modify prob 11a to display the Fib numbers on the screen.**
2. **Modify prob 11b to include Multiplication operation also.**

%%

title 15a (i) Program to Create a file using DOS interrupts.

```
.model small

.stack

Disp  MACRO  str
       LEA DX, str
       MOV AH, 09h
       INT 21h
ENDM


.data
       filen DB 'd:\mpa_09\test.txt $'
       msg1 DB 'Creation successful $'
       msg2 DB 'Creation Fails $'

.code

      MOV AX, @data
      MOV DS, AX

      MOV AH, 3ch                       ; DOS function to create a file
      MOV CX, 00h                        ; file attributes in CX reg
      LEA DX, filen
      INT 21h

      JC error
      disp msg1                          ; invoke macro to display messages
      JMP stop

error: disp msg2
```

```
stop:     MOV AH,4ch
           INT 21h

 END start
```

Title   15a (ii) program to delete a File

```
.model small

.stack

disp  MACRO  str         ; macro definition to display a string
       LEA DX, str
       MOV AH, 09h
       INT 21h
ENDM

.data
       filen DB 'd:\mpa_09\test.txt'
       msg1 DB 10,13, '  file deleted successfully . $'
       msg2 DB 10, 13, ' !!!! file not found !!!$'

.code

       MOV AX, @data
       MOV DS, AX

       LEA DX, filen           ; DOS function to delete a file
       MOV CX, 20h
       MOV AH, 41h
       INT 21h

         JC fail
          disp msg1
         JMP next

fail:   disp msg2

next:   MOV AH, 4CH
         INT 21H
 END
```

title   15b) Drive an Elevator Interface

; Initially, Elevator is in Ground floor, with all requests in OFF state. When a
; request is made, the Elevator moves to that floor and stays there until further
; requests.

```
.model small
.stack
.data
          pa  EQU  0d400h
          pb  EQU  0d401h
          pc  EQU  0d402h
          cr  EQU  0d403h
        fcode DB  00h, 03h, 06h, 09h                        ; floor numbers
        fclear DB  0e0h, 0d3h, 0b6h, 79h                    ; code to clear the request LED

.code
        MOV AX, @data
        MOV DS, AX
        MOV DX, cr
        MOV AL, 82H                       ; port A as output, port B as input
        OUT DX, AL

           XOR AX, AX
back1:  MOV AL, AH
          OR AL, 0F0H
          MOV DX, pa
          OUT DX, AL
          MOV DX, pb                      ; point to port B
back2:  MOV CH, AH                        ; initially AH =0
                                          ; initially, elevator in grd floor
          MOV AH, 01h
          INT 16H
          JNZ  stop

         MOV AH, CH
         IN AL, DX                         ; read floor request from port B
         AND AL, 0FH
         CMP AL, 0FH
         JZ back2

         MOV SI, 0
findf:   ROR AL, 1                          ; find the floor number
         JNC found
         INC SI
         JMP findf
found:  MOV AL, fcode[SI]                   ; move the elevator to
          CMP AL, AH                       ; the requested floor after
          JA up                             ; comparing request with present elevator position
          JB down

clear:  MOV AL, fclear[SI]                ; after reaching the floor, clear the request
         MOV DX, pa
         OUT DX, AL
```

```
          JMP back1
up:      CALL delay              ; to move elevator upwards
        INC AH
        XCHG AL, AH
        OR AL, 0F0H
        MOV DX, pa
        OUT DX, AL
        AND AL, 0FH
        XCHG AH, AL
        CMP AL, AH
        JNZ up
        JMP clear
down:   CALL delay              ; to move elevator downwards
          DEC AH
          XCHG AL, AH
         OR AL, 0F0H
         MOV DX, pa
         OUT DX, AL
         AND AL, 0FH
         XCHG AH, AL
         CMP AL, AH
         JNZ down
         JMP clear
stop:  MOV AH, 4CH
          INT 21

delay     PROC     NEAR                 ; delay procedure

           PUSH CX
           PUSH AX
           MOV CX, 04fffh
back3:  MOV AX, 02fffh
back4:  DEC AX
           JNZ back4
           LOOP back3
           POP AX
           POP CX
           RET
delay ENDP

END
```

Exercise questions:

1. **Modify prob 15a to display the present working directory**
2. **Modify prob 15b to move the Elevator to Ground floor after all the requests are serviced.**

```
title    12a) Read the current time from system and display it in a standard format on
         ;   the screen.

.model small

.stack

.data
       msg1 DB 10,13, " @@@ Reading system Time :::$"

       msg2 DB 10, 13, ' The system time is >> $'

clrscr   MACRO                            ; macro definition to clear screen
       MOV AL, 2
       MOV AH, 0
       INT 10H
ENDM
dispm   MACRO  str
       LEA DX, str
       MOV AH, 9H
       INT 21H
ENDM
set_cursor   MACRO                          ; macro definition to fix the cursor position on screen
        MOV BL, 0
        MOV AL, 3                          ; using BIOS function
        MOV DH, 15
        MOV DL, 20
        MOV AH, 2
        INT 10H
ENDM
.code

        MOV AX, @data
        MOV DS, AX
        clrscr

        dispm msg1                          ;invoke macros
        set_cursor
        dispm msg2

        MOV AH, 2Ch                                ; DOS function to read system time
        INT 21h

         MOV AL, CH                                ;Hours in CH register
         AAM                                       ; unpack the digits
         MOV BX, AX
         CALL display                      ; use a procedure to convert to ASCII and display on screen
         MOV DL,':'                      ; the format is hh:mm:ss
        MOV AH, 02h
        INT 21h

         MOV AL, CL                    ; minutes in CL register
        AAM
        MOV BX, AX
        CALL display
        MOV DL, ':'
        MOV AH, 02h
        INT 21h

         MOV AL, DH                    ; seconds in DH register
        AAM
        MOV BX,AX
        CALL display
        MOV AH,4ch
        INT 21h

display    PROC    NEAR               ;convert to ASCII and display
        MOV DL, BH
        ADD DL, 30h
        MOV AH, 02h
        INT 21h

        MOV DL, BL
        ADD DL, 30h
        MOV AH, 02h
        INT 21h
        RET
display  ENDP
END
```

---

title   12b) Generate a sine wave using the dac interface (the output of the dac is to
;   be displayed on a CRO).

```
.model small
.data
      porta   EQU  0d400h
      portb   EQU  0d401h
      portc   EQU  0d402h
      cwr     EQU   0d403h

      sines DB   00,11,22,33,43,53,63,72,81,89,97,104,109,115,119,122,125,126,127
                                                  ; array to store values of sin θ

      msg DB 10, 13, ' Observe Sine wave on CRO; Press any key to exit', 10, 13, '$'
.code
```

```
        MOV AX, @data
        MOV DS, AX

        MOV DX, cwr                     ; make all ports as output
        MOV AL, 80h                     ; only port A of 8255 is used
        OUT DX, AL

        LEA DX, msg                      ; display message on the screen
        MOV AH, 9H
        INT 21H

        MOV DX, porta                   ; DX has address of port A of 8255

full_wave:
        MOV SI, OFFSET sines            ; use SI as pointer to array
        MOV CX, 13h                     ; number of values in the array
                                  ; the entire sinewave (1 cycle) is divided into 4 quadrants
first_quart:                      ; peak value is 5V
        MOV AL, 7FH
        MOV BL, BYTE PTR [SI]             ; take sine value from array
        ADD AL, BL
        OUT DX, AL                        ; and send it to port (CRO)
        INC SI
        LOOP first_quart
        MOV  CX, 12h
        DEC SI

second_quart:
        MOV AL, 7FH
        MOV BL, byte ptr [SI]
        ADD AL, BL
        OUT DX, AL
        DEC SI
        LOOP second_quart

        MOV SI, offset sines
        MOV CX, 13h

third_quart:
        MOV AL, 7fh
        MOV BL, byte ptr [SI]
        SUB AL, BL
        OUT DX, AL
        INC SI
        LOOP third_quart

        DEC SI
        MOV CX, 12h
```

```
fourth_quart:
        MOV AL, 7Fh
        MOV BL, BYTE PTR [SI]
        SUB AL, BL
        OUT DX, AL
        DEC SI
        LOOP  fourth_quart

        MOV AH, 1                                   ; stop if any key is pressed
        INT 16H
        JNZ stop
        JMP   full_wave                              ; otherwise, continuously generate sine wave

stop:   MOV AH,4ch
        INT 21h

END
```

Exercise questions:

1. **Modify prob 12a to unpack the digits without using AAM instruction.**
2. **Modify prob 12a to read the current time and implement a real-time clock.**
3. **Modify prob 12b to generate waveforms with 2.5V and 4V peak value.**

%%

title 13a) To simulate a decimal UP counter to display 00-99

```
.model small
.stack

.data
        msg DB    'The decimal Counter is running##', 10,10,13, '$'

clrscr    MACRO
           MOV AL, 2
           MOV AH, 0
           INT 10H
ENDM

dispm    MACRO   str
            LEA DX,  str
            MOV AH, 9H
            INT 21H
ENDM

.code
         MOV AX, @data
        MOV DS, AX
        clrscr                      ; invoke macro to clear screen
        XOR AX, AX
        dispm  msg                  ; invoke macro to display message

         CALL delay
         MOV AL, 30H                  ; AL contains first (higher) digit
again:  MOV DL, AL                    ; display higher digit
         MOV AH, 02h
         INT 21h

         MOV SI, AX                   ; save value of AL
         MOV BL, 30h                   ;BL contains second digit
back:   MOV DL, BL
         MOV AH, 2                    ; display second (lower) digit
         INT 21h
         INC BL                      ; increment second digit
         CALL delay

         MOV AH, 03h                 ; get current cursor position
         INT 10h
         MOV DL, 1                  ; set cursor to next column
         MOV AH, 2
         INT 10h

        CMP BL, 39h                  ; inner loop
        JLE back                     ; display all second digit (0-9)

        MOV DL, 0                    ; set cursor position to previous column
        MOV AH, 2
        INT 10h

        MOV AX, SI
        INC AL                       ;increment 1st digit
        CMP AL, 39h
        JLE again                   ;loop 1st digit(0-9)
                                    ; outer loop for higher digit
        MOV AH, 4Ch
        INT 21h
delay    PROC     NEAR                          ; delay procedure
         PUSH CX
         PUSH AX
         MOV CX, 1000H
back2:  MOV AX, 04FFFh
back1:  DEC AX
         JNZ  back1
         LOOP  back2
         POP AX
```

```
          POP CX
        RET
delay     ENDP

END
```

```
title     13b) Generate a half rectified sine wave form using the DAC interface    (the
              ; output of the DAC is to be displayed on a CRO).
.model
.data
       sines DB 00,22,44,66,87,108,127,146,164,180,195,209,221,231,240,246,251,254,255
       msg DB 10,13, 10,  'Observe Half Rectified wave on CRO. Press any key to exit $'

       porta  EQU  0d400h
       portb  EQU  0d401h
       portc  EQU  0d402h
       ctrl    EQU  0d403h

.stack
.code
       MOV AX, @data
       MOV DS, AX
       LEA DX, msg
       MOV AH, 9
       INT 21H
       MOV AL, 80h                            ; make all ports as output
       MOV DX, ctrl
       OUT DX, AL
       CALL delay

half_wave:
       MOV DX, porta
       MOV CX, 13h
       MOV SI, OFFSET sines                   ; use SI as pointer to array
                                              ; half-rectified wave will have two quadrants output
                                              ; and next two quadrants zero voltage
first_quart:
       MOV AL, BYTE PTR [SI]
       OUT DX, AL
       CALL delay
       INC SI
       LOOP first_quart
       DEC SI
       MOV CX, 12H

second_quart:
       MOV AL, BYTE PTR [SI]
```

```
        OUT DX,  AL
        CALL delay
        DEC SI
        LOOP  second_quart

        MOV CX, 25H
no_wave:
        MOV AL, 00h
        OUT DX, AL
        CALL delay
        LOOP no_wave

        MOV AH, 1                           ;check if any is pressed. IF yes, stop
        INT 16H                             ; else start again
        JNZ stop
        JMP half_wave

stop:    MOV AH, 4CH
         INT 21H

delay PROC   NEAR
         PUSH CX
         MOV CX, 2FFFH
back:   NOP
         LOOP BACK
         POP CX
         RET
  delay  ENDP

END
```

Exercise questions:

1. **Modify prob 13a to generate Decimal DOWN counter.**
2. **Modify prob 13a to generate HEX UP counter.**
3. **Modify prob 13b to generate waveforms with 5V peak value and 2.5 V during no_wave duration.**

%%%

title 14a) Read a pair of input co-ordinates in BCD and move cursor to specified
; position on screen.

```
.model small

.stack

.data
```

```
        xmsg DB 13,10,'ENTER VALUE OF X CO-ORDINATES in BCD:$'

        x    DB ?                          ; to store X coordinate value

        ymsg DB 13,10,'ENTER VALUE OF Y CO-ORDINATES in BCD:$'
        y    DB ?                          ; to store Y coordinate value

        msg  DB  'the cursor is moved here.$'

clrscr    MACRO
        MOV AH, 0                          ;macro TO CLEAR THE SCREEN
        MOV AL, 3
        INT 10h
ENDM
dispm    MACRO  str                        ; macro to display string
          MOV DX, OFFSET str
          MOV AH,9H
          INT 21H
ENDM
.code

        MOV AX, @data
        MOV DS, AX
                                                   ; TO READ BCD CO-ORDINATES
        dispm  xmsg
        CALL read_bcd                              ; using a procedure
        MOV x, BH                                  ; X coordinate value stored
        dispm  ymsg
        CALL read_bcd
        MOV y, BH                                 ; Y coordinate value stored

        clrscr                                    ; invoke macro to clear screen
                                                  ; to set cursor position
        MOV DH, x                                 ; using BIOS function
        MOV DL, y
        MOV BH, 0
        MOV AH, 2H
        INT 10H
        dispm msg

        MOV DL,'*'                                 ; at cursor position, * is shown
        MOV AH, 02h
        INT 21h

        MOV AH, 1H                                 ; press any key to exit
        INT 21H
        MOV AH, 4Ch
        INT 21h
```

```
read_bcd   PROC  NEAR                      ; procedure to read number from keyboard
                                           ; and convert that ASCII to packed BCD
       MOV AH, 01h                         ; read first digit
       INT 21h
       MOV BH, AL

       MOV AH, 01h                         ; read SECOND DIGIT
       INT 21h
       MOV BL, AL
       MOV AX, BX
       SUB AX, 3030H                      ; get unpacked BCD numbers
       AAD                                ; get packed BCD numbers
       MOV BH, AL                         ; copy packed value into BH
       RET
read_bcd    ENDP

END
```

---

title   14b) Generate a fully rectified sine wave form using the DAC interface
; (the output of the DAC is to be displayed on a CRO).

```
.model small
.data
       porta   EQU  0d400h
       portb   EQU  0d401h
       portc   EQU  0d402h
       cwr     EQU  0d403h

       sines DB   00,11,22,33,43,53,63,72,81,89,97,104,109,115,119,122,125,126,127
              ; array to store values of sin

       msg DB 10, 13, ' Observe Full Rectifier Sine wave on CRO; Press any key to exit', 10, 13, '$'

.code
       MOV AX, @data
       MOV DS, AX

       MOV DX, cwr                     ; make all ports as output
       MOV AL, 80h                     ; only port A of 8255 is used
       OUT DX, AL

       LEA DX, msg                     ; display message on the screen
       MOV AH, 9H
       INT 21H

       MOV DX, porta                          ;access port A using DX register
```

```
fullrec_wave:
        MOV SI, OFFSET sines
        MOV CX, 13h
                ; the full rectified sinewave will have first two quadrants repeated continuously.
first_quart:
        MOV AL, 7FH
        MOV BL, BYTE PTR [si]          ; take sine value from array
        ADD AL, BL
        OUT DX, AL                     ; and send it to port (CRO)
        INC SI
        LOOP   first_quart
        MOV CX, 12H
        DEC SI

second_quart:
        MOV AL, 7FH
        MOV BL, BYTE PTR [SI]
        ADD AL, BL
        OUT DX, AL
        DEC SI
        LOOP second_quart

        MOV AH, 1                              ; if any key is pressed, stop.
        INT 16H
        JNZ stop
        JMP fullrec_wave

stop:   MOV AH,4CH
         INT 21H
END
```

Exercise questions:

1. **Modify prob 14a to display your name at the position after reading the coordinate points.**
2. **Modify prob 14b to generate waveforms with output waveforms 0 to 2.5V and 0 to 5V range.**

%%%

```
title    4A) Read an alphanumeric character and display its equivalent
         ;   ASCII code(in HEX) at the center of the screen.

.model small
.stack

.data
        msg1 DB 10,  'ENTER A KEY FROM KEYBOARD',10,13,'$'
        msg2 DB       'The ASCII value is: $'
```

```
clrscr   MACRO
       MOV AL, 3                      ;Video mode = 3
       MOV AH, 0                      ;To clear the screen
       INT 10H
ENDM
dispm   MACRO  str
       MOV DX, OFFSET str
       MOV AH, 9H
       INT 21H
ENDM

.code
       MOV AX, @data
       MOV DS, AX                      ;Initialize DS

       clrscr                         ; invoke macro
       dispm msg1                      ; invoke macro to DISPLAY MSG

       MOV AH, 1                      ;Read a char from KB with echo
       INT 21H

       MOV BL, AL                     ;Store it in BL

                                      ;set cursor position using BIOS function
       MOV BH, 0                      ;page 0
       MOV DH, 12                     ;row=12 central row
       MOV DL, 40                     ;col=40 central col
       MOV AH, 2
       INT 10H
       dispm msg2
       MOV AL, BL                    ;unpack the digits of the character
       AND AL, 0F0H                  ;select the higher order nible
       MOV CL, 4                     ;Shift count
       SHR AL, CL                    ;Shift right by 4
       CALL DISP                     ;display it
       MOV AL, BL
       AND AL, 0FH                   ;select the lower order nibble
       CALL disp                      ;display it
       MOV AH, 4CH                   ;safe exit to dos
       INT 21H

disp:   CMP AL, 0AH                   ;convert an alphanumeric character to
        JB SKIP                       ;equivalent ASCII value
        ADD AL, 7
SKIP:  ADD AL, 30H
        MOV DL, AL
        MOV AH, 02                    ; call dos function 02h to print a character
        INT 21H
        RET
END
```

Exercise questions:

1. **Modify prob 4a to display the ASCII value at any position on the screen**
2. **Modify prob 4b to display messages PASS and FAIL alternately on a 7-segment display**

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

title 5A) Reverse a given string and check whether it is a palindrome or not.

```
.model small
.stack
.data
          buf DB 60                              ; array to store original string
               DB ?
               DB 60 DUP(?)
         revbuf DB 60 DUP (?)                   ; array to store reverse string

       msg DB ' ENTER THE STRING: $'

       msg1 DB 13,10,10,"     ENTERED STRING IS A PALINDROME  $"
       msg2 DB 13,10,10,"     ENTERED STRING IS NOT A PALINDROME !!! $"

dispm    MACRO    str                           ; macro definition to display message
          LEA DX, str
          MOV AH, 9H
          INT 21H
ENDM
clrscr   MACRO                                  ; macro definition to clear screen
         MOV AL, 2
         MOV AH, 0
         INT 10H
ENDM
.code
       MOV AX, @data
       MOV DS, AX
       MOV ES, AX
        clrscr
       dispm msg                                ; invoke macro

        LEA DX, buf                              ; read a string from keyboard
        MOV AH, 0AH
        INT 21H

      LEA SI, buf+1
      LEA DI, revbuf
      MOV CH, 0
      MOV CL, buf+1
      ADD SI, CX

back:  MOV AL, BYTE PTR [SI]                 ; reverse string  and store in memory
        MOV BYTE PTR [DI], AL
        INC DI
        DEC SI
        LOOP back

         CLD                                 ; auto increment pointers
         LEA SI, buf+2                       ; compare original and reversed strings
         LEA DI, revbuf                      ; using CMPSB instruction
         MOV CL, SIZE buf+2                 ; get size of string in CL reg
         repe CMPSB
         JNZ  noteq

         dispm msg1                          ; invoke macro to display appropriate message
         JMP stop
noteq: dispm msg2

stop:   MOV AH, 4CH
         INT 21H

       END
```

Exercise questions:

1. **Modify prob 5a to display original as well as reversed string on the screen.**
2. **Modify prob 5a to check for palindrome without reversing the original string.**
3. **Modify prob 5b to scroll the message in one direction only for a specified number of times.**

%%

```
title      6a) read two strings from keyboard and store them in locations.
          ; check whether they are equal or not and display appropriate messages.
           ; also display the length of the strings
.model small

.stack
.data
str1   DB 150                      ; reserve memory to store string 1
       DB ?                        ; string length stored here
       DB 150 dup(?)
str2   DB 150                      ; reserve memory array to store string 2
       DB ?
       DB 150 dup(?)

msg1  DB  10,10,13, '      Strings are Equal.  $'             ; Messages
```

```
msg2  DB  10,10,13, '       Strings Not Equal !!!!! $'
msg3  DB  10,13, ' Enter string1 (upto 9 characters): $'
msg4  DB  10,13, ' Enter string2 (upto 9 characters): $'
msg5  DB  10,13, '   Length of string1 =  $'
msg6  DB  10,13, '   Length of string2 =  $'

clrscr   MACRO                                  ; macro definition to clear screen
         MOV AL, 2
         MOV AH,0
         INT 10H
ENDM

dispm    MACRO    str                           ; macro definition to display string on screen
          LEA DX, str
          MOV AH, 09h
          INT 21h
ENDM

.code
       MOV AX,@data
       MOV DS, AX
       MOV ES, AX                       ; Extra segment required for CMPSB instruction
         clrscr
       dispm msg3                        ; invoke macro to display message

       MOV DX, OFFSET str1              ; read string1 from keyboard
       MOV AH,0ah                       ; using DOS interrupt
       INT 21h

       dispm msg4

       MOV DX, OFFSET str2                        ; read  string2 from keyboard
       MOV AH, 0AH
       INT 21h
                                                    ;To display the string1 length
        dispm msg5                                  ; invoke macro
        MOV DL,  str1[1]
        ADD DL, 30H
        MOV AH, 2
        INT 21H
        dispm msg6                                  ;To display the string2 length
        MOV DL, str2[1]
        ADD DL, 30H
        MOV AH, 2
        INT 21H

        MOV AL, str1[1]                           ;Compare string lengths
        CMP AL, str2[1]
        JNE noteq                              ;If lengths are not equal, display 'not equal'

          MOV CH, 00h                          ; If string lengths are equal, then compare two strings
          MOV CL, str1[1]                       ; get size of string 2 in CL reg
          CLD
          LEA SI, str1+2
          LEA DI, str2+2
          repe  CMPSB                          ;Compare the strings usign CMPSB instruction
          JNZ noteq                             ;If they are not equal display 'not equal'

          dispm msg1                            ;If strings are equal, display 'Equal'

          JMP stop
noteq:  dispm msg2                               ;Display Not Equal

stop:   MOV AH,4CH
          INT 21h

END
```

Exercise questions:

1. **Modify prob 6a to check for string equality without using CMPSB instruction.**
2. **Modify prob 6a to accept strings with more than 9 characters and display their lengths appropriately in Hex or Decimal.**