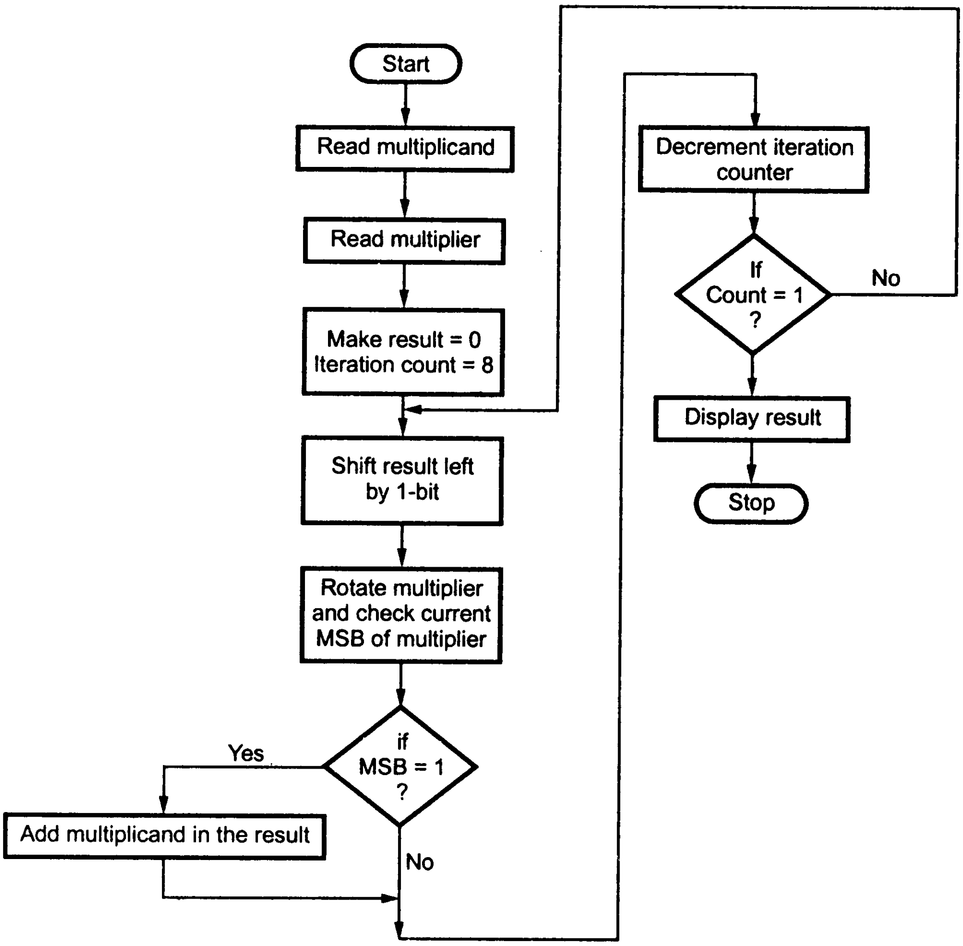
Flowchart :



Multiplication of HEX numbers

```
PROMPT MACRO MESSAGE                ; Define macro with MESSAGE as a
                                    ; parameter
        PUSH     AX                 ; save register
        MOV      AH, 09H            ; display message
        LEA      DX, MESSAGE
        INT      21H
        POP      AX                 ; restore register
        ENDM

.MODEL SMALL                 ; select small model
.STACK 100
.DATA                        ; start data segment
```

```
    MUL_ER DB ?             ; define NUMBER
    MUL_AND DB ?            ; define NUMBER
    MES1 DB 10,13, 'Enter 2-digit hex number as a multiplicand:$'
    MES2 DB 10,13, 'Enter 2-digit hex number as a multiplier :$'
    MES3 DB 10,13, 'The result of multiplication is :$'
.CODE                       ; start code segment
START:   MOV AX, @DATA      ; [Initialize
         MOV DS, AX         ;  data segment]

         PROMPT   MES1
         CALL     READ_HEX2
         MOV      MUL_AND,BL

         PROMPT   MES2
         CALL     READ_HEX2
         MOV      MUL_ER,BL

         MOV      DH,00
         MOV      DL,MUL_AND
         MOV      CX,0008
         MOV      AX,0000
REP1:    SHL      AX,1
         ROL      BL,1
         JNC      SKIP
         ADD      AX,DX
SKIP:    LOOP     REP1
         PROMPT   MES3
         CALL     D_HEX
         MOV      AH,02H
         MOV      DL,'H'
         INT      21H
         MOV      AH, 4CH       ; [Exit to
         INT      21H           ;  DOS]

READ_HEX2 PROC NEAR

         MOV      CL, 04        ;  load shift count
         MOV      SI, 02        ;  load iteration count
         MOV      BL, 0         ;  clear result
BACK :   MOV      AH, 01        ; [Read a key
```

```
        INT     21H         ;  with echo]
        CALL    CONV        ;  convert to binary
        SHL     BL, CL      ; [pack two
        ADD     BL, AL      ;  binary digits
        DEC     SI          ;  as 8-bit
        JNZ     BACK        ;  number]
        RET
        ENDP

    ; The procedure to convert contents of AL into
    ; hexadecimal equivalent
CONV PROC NEAR
        CMP AL,'9'
        JBE SUBTRA30       ; if number is between 0 through 9
                             CMP AL,'a'
        JB  SUBTRA37       ; if letter is uppercase
        SUB AL, 57H        ; subtract 57H if letter is lowercase
        JMP LAST
SUBTRA30:SUB AL, 30H       ; convert number
        JMP LAST
SUBTRA37:SUB AL, 37H       ; convert uppercase letter
LAST:   RET
CONV    ENDP

D_HEX PROC NEAR
        MOV CL, 04H        ; Load rotate count
        MOV CH, 04H        ; Load digit count
BAC1:   ROL AX, CL         ; rotate digits
        PUSH AX            ; save contents of AX
        AND AL, 0FH        ; [Convert
        CMP AL,9           ;  number
        JBE Add30          ; to
        ADD AL, 37H        ; its
        JMP DISP1          ; ASCII
Add30:
        ADD AL,30H         ; equivalent]
DISP1:  MOV AH,02H
        MOV DL,AL          ; [Display the
        INT 21H            ;  number]
        POP AX             ; restore contents of AX
```

```
        DEC CH              ; decrement digit count
        JNZ BAC1            ; if not zero repeat
        RET
        ENDP
        END
```

Multiplication of BCD numbers

```
PROMPT MACRO MESSAGE ;Define macro with MESSAGE as a parameter
        PUSH    AX
        MOV     AH, 09H
        LEA     DX, MESSAGE
        INT     21H
        POP     AX
        ENDM


.MODEL SMALL                ; select small model
.STACK 100



.DATA                       ; start data segment
        MUL_ER DB ?         ; define NUMBER
        MUL_AND DB ?        ; define NUMBER
        MES1 DB 10,13,       'Enter 2-digit BCD number (<256) as a
                              multiplicand : $'
        MES2 DB 10,13,       'Enter 2-digit BCD number (<256) as a
                              multiplier : $'
        MES3 DB 10,13,       'The result of multiplication is : $'
.CODE                       ; start code segment
START:  MOV AX, @DATA       ; [Initialize
        MOV DS, AX          ;  data segment]

        PROMPT  MES1
        CALL    BTH
        MOV     MUL_AND,AL

        PROMPT  MES2
        CALL    BTH
        MOV     MUL_ER,AL
        MOV     BL,AL
```

```
        MOV     DH,00
        MOV     DL,MUL_AND
        MOV     CX,0008
        MOV     AX,0000
REP1:   SHL     AX,1
        ROL     BL,1
        JNC     SKIP1
        ADD     AX,DX
SKIP1:  LOOP    REP1
        PROMPT  MES3
        CALL    D_BCD

        MOV     AH, 4CH         ; [Exit to
        INT     21H             ;  DOS]

BTH PROC NEAR
        MOV     CX, 10          ; load 10 decimal in CX
        MOV     BX, 0           ; clear result
BACK2:  MOV     AH,01H          ;[Read key
        INT     21H             ; with echo]
        CMP     AL,'0'
        JB      SKIP            ; jump if below '0'
        CMP     AL,'9'
        JA      SKIP            ; jump if above '9'
        SUB     AL, 30H         ; convert to BCD
        PUSH    AX              ; save digit
        MOV     AX, BX          ; multiply previous result by 10
        MUL     CX
        MOV     BX, AX          ; get the result in BX
        POP     AX              ; retrieve digit
        MOV     AH, 00H
        ADD     BX, AX          ; Add digit value to result
        JMP     BACK2           ; Repeat
SKIP :  MOV     AX,BX           ; save the result in AX
        RET
        ENDP
```

```
D_BCD PROC NEAR

        MOV CX, 0           ; Clear digit counter
        MOV BX, 10          ; Load 10 decimal in BX
BACK1:  MOV DX, 0           ; Clear DX
        DIV BX              ; divide DX : AX by 10
        PUSH DX             ; Save remainder
        INC CX              ; Counter remainder
        OR  AX, AX          ; test if quotient equal to zero
        JNZ BACK1           ; if not zero divide again
        MOV AH, 02H         ; load function number
DISP:   POP DX              ; get remainder
        ADD DL, 30H         ; Convert to ASCII
        INT 21H             ; display digit
        LOOP DISP
        RET
        ENDP

        END
```

Program 21 : Divide 4 digit BCD number by 2 digit BCD number.

```
PROMPT MACRO MESSAGE ;Define macro with MESSAGE as a parameter
        PUSH    AX
        MOV     AH, 09H
        LEA     DX, MESSAGE
        INT     21H
        POP     AX
        ENDM

.MODEL SMALL                    ; select small model
.STACK 100

.DATA                           ; start data segment
    DIVISOR  DB  ?              ; define NUMBER
    DIVIDEND DW  ?              ; define NUMBER
    MES1     DB  10,13,'Enter 4-digit BCD number as dividend:$'
    MES2     DB  0,13, 'Enter 2-digit BCD number as a divisor:$'
    MES3     DB  10,13, 'The Quotient of Division is  : $'
    MES4     DB  10,13, 'The Remainder of Division is : $'
```

```
.CODE                                   ; start code segment
START:  MOV     AX, @DATA               ; [Initialize
        MOV     DS, AX                  ;  data segment]

        PROMPT  MES1
        CALL    ATB

        PROMPT  MES2
        CALL    BTH
        MOV     DIVISOR,AL

        MOV     AX, DIVIDEND
        DIV     DIVISOR

        MOV     BX,AX
        PROMPT  MES3
        MOV     AH,00
CALL D_BCD

        PROMPT  MES3

        MOV     AH,00
        MOV     AL,BH
CALL D_BCD
        MOV     AH, 4CH                 ; [Exit to
        INT     21H                     ;  DOS]
BTH PROC NEAR
        MOV     CX, 10                  ; load 10 decimal in CX
        MOV     BX, 0                   ; clear result
BACK2:  MOV     AH,01H                  ;[Read key
        INT     21H                     ; with echo]
        CMP     AL,'0'
        JB      SKIP1                   ; jump if below '0'
        CMP     AL,'9'
        JA      SKIP1                   ; jump if above '9'
        SUB     AL, 30H                 ; convert to BCD
        PUSH    AX                      ; save digit
        MOV     AX, BX                  ; multiply previous result by 10
```

```
        MUL CX
        MOV BX, AX          ; get the result in BX
        POP AX              ; retrieve digit
        MOV AH, 00H
        ADD BX, AX          ; Add digit value to result
        JMP BACK2           ; Repeat
SKIP1:  MOV AX,BX           ; save the result in AX
        RET
        ENDP
```

```
D_BCD PROC NEAR
        PUSH BX
        MOV CX, 0           ; Clear digit counter
        MOV BX, 10          ; Load 10 decimal in BX
BACK1:  MOV DX, 0           ; Clear DX
        DIV BX              ; divide DX : AX by 10
        PUSH DX             ; Save remainder
        INC CX              ; Counter remainder
        OR  AX, AX          ; test if quotient equal to zero
        JNZ BACK1           ; if not zero divide again
        MOV AH, 02H         ; load function number
DISP:   POP DX              ; get remainder
        ADD DL, 30H         ; Convert to ASCII
        INT 21H             ; display digit
        LOOP DISP
        POP BX
        RET
        ENDP
```

```
ATB PROC NEAR

        PUSH CX             ; Save registers
        PUSH BX
        PUSH AX

        MOV CX, 10          ; load 10 decimal in CX
        MOV BX, 0           ; clear result
BACK:   MOV AH,01H          ;[Read key
        INT 21H             ; with echo]
        CMP AL,'0'
```

```
        JB   SKIP          ; jump if below '0'
        CMP  AL,'9'
        JA   SKIP          ; jump if above '9'
        SUB  AL, 30H       ; convert to BCD
        PUSH AX            ; save digit
        MOV  AX, BX        ; multiply previous result by 10
        MUL  CX
        MOV  BX, AX        ; get the result in BX
        POP  AX            ; retrieve digit
        MOV  AH, 00H
        ADD  BX, AX        ; Add digit value to result
        JMP  BACK          ; Repeat
SKIP:   MOV  DIVIDEND,BX   ; save the result in NUMBER

        POP  AX            ; Restore registers
        POP  BX
        POP  CX
        RET
        ENDP

        END
```

Program 22 : To perform conversion of temperature from °F to °C.

```
PROMPT MACRO MESSAGE ;Define macro with MESSAGE as a parameter
        PUSH    AX
        MOV     AH, 09H
        LEA     DX, MESSAGE
        INT     21H
        POP     AX
        ENDM

.MODEL SMALL            ; select small model
.STACK 100

.DATA                   ; start data segment
   NUMBER DW ?          ; define NUMBER
   MES1 DB 10,13, 'Enter Temperature in Degree FARENHEIT : $'
   MES2 DB 10,13, 'The Temperature in Degree Celsius is : $'
```

```
.CODE                             ; start code segment
START:  MOV     AX, @DATA         ; [Initialize
        MOV     DS, AX            ;  data segment]

        PROMPT  MES1

        CALL    ATB               ; Get the Temperature in F

        MOV     AX,NUMBER
        SUB     AX,20H            ; Subtract 32
        MOV     NUMBER,AX


        MOV     BX,05
        MOV     CX,09
        MUL     BX                ; Multiply by 5
        DIV     CX                ; Divide by 9

        MOV     NUMBER, DX        ; Save remainder
        PROMPT  MES2
        CALL    D_BCD             ; Display result in decimal
        CMP     NUMBER,0          ; If remainder is zero exit
        JZ      LAST
        MOV     DL, '.'           ; Display decimal point
        MOV     AH,02H
        INT     21H

        MOV     AX,0064H          ; Multiply remainder by 100
        MUL     NUMBER            ; Divide result by 9
        DIV     CX
        CALL    D_BCD             ; display fractions

LAST:   MOV     AH, 4CH           ; [Exit to
        INT     21H               ;  DOS]
```

```
D_BCD PROC NEAR
        PUSH CX
        MOV  CX, 0         ; Clear digit counter
        MOV  BX, 10        ; Load 10 decimal in BX
BACK1:  MOV  DX, 0         ; Clear DX
        DIV  BX            ; divide DX : AX by 10
        PUSH DX            ; Save remainder
        INC  CX            ; Counter remainder
        OR   AX, AX        ; test if quotient equal to zero
        JNZ  BACK1         ; if not zero divide again
        MOV  AH, 02H       ; load function number
DISP:   POP  DX            ; get remainder
        ADD  DL, 30H       ; Convert to ASCII
        INT  21H           ; display digit
        LOOP DISP
        POP  CX
        RET
        ENDP
```

```
ATB PROC NEAR

        PUSH CX            ; Save registers
        PUSH BX
        PUSH AX

        MOV  CX, 10        ; load 10 decimal in CX
        MOV  BX, 0         ; clear result
BACK:   MOV  AH,01H        ;[Read key
        INT  21H           ; with echo]
        CMP  AL,'0'
        JB   SKIP          ; jump if below '0'
        CMP  AL,'9'
        JA   SKIP          ; jump if above '9'
        SUB  AL, 30H       ; convert to BCD
        PUSH AX            ; save digit
        MOV  AX, BX        ; multiply previous result by 10
        MUL  CX
        MOV  BX, AX        ; get the result in BX
        POP  AX            ; retrieve digit
        MOV  AH, 00H
```

```
        ADD BX, AX          ; Add digit value to result
        JMP BACK            ; Repeat
SKIP:   MOV NUMBER,BX       ; save the result in NUMBER

        POP AX              ; Restore registers
        POP BX
        POP CX
        RET
        ENDP
    END
```

Program 23 : String operations

Program Statement : Write 8086 ALP for the following operations on the string entered by the user.

a. Calculate length of the string.

b. Reverse the string.

c. Check whether the string is palindrome or not.

Make your program user friendly by providing MENU like :

a. Enter the string.

b. Calculate length of string.

c. Reverse string.

d. Check palindrome.

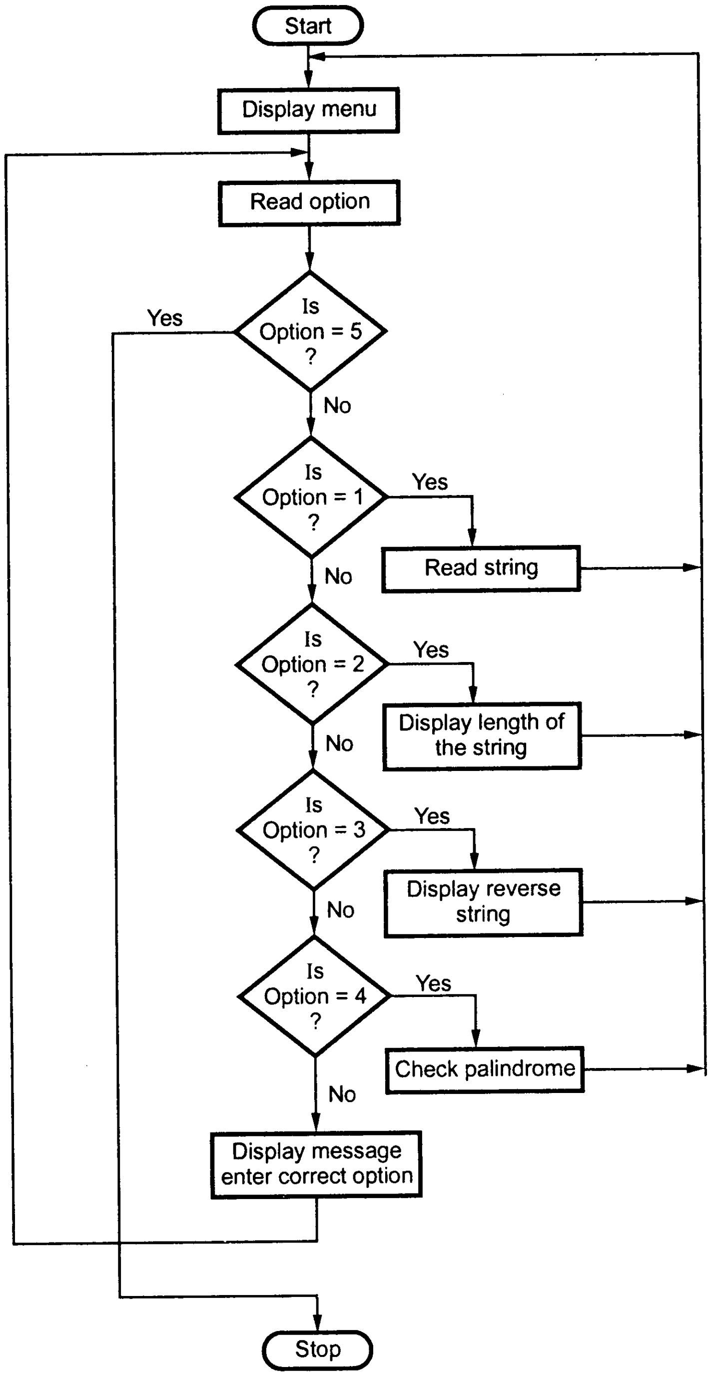
e. Exit.

Here we use PROMPT macro to display the message on the screen, accept choice from the user and call proper procedure to perform desired task. To enter a string we use function 0AH of INT21. This function accepts a string and stores it in the buffer along with its length. Let us see the algorithm and flow chart.

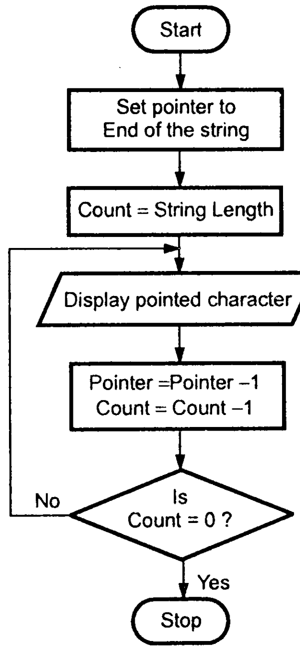
Algorithm :

1. Display Menu
   a. Enter the string.
   b. Calculate length of the string.
   c. Reverse the string.
   d. Check whether the string is palindrome or not.
   e. Exit.
   Enter the option : -
2. Read the option
   If option is
   a. Read the string.
   b. Read the string length and display it.
   c. Initialize pointer at the end of the string and display the string from end to start.
   d. i) Initialize two pointers one at start and other at the end.
      ii) Compare two bytes; if not equal stop and display string is not palindrome.
      iii) Increment start pointer and decrement end pointer.
      iv) Repeat step ii) and iii) until two pointers overlap i.e. until start pointer reach the half the string.
   e. Exit to DOS.
3. Stop.

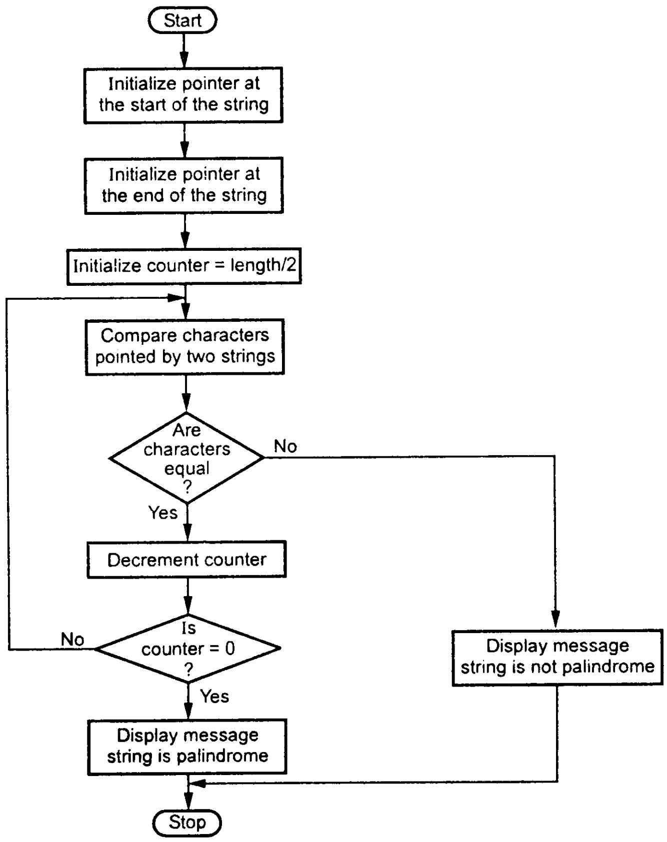
Flowchart :


Start
Display menu
Read option
Is Option = 5 ?
Yes
No
Is Option = 1 ?
Yes
Read string
No
Is Option = 2 ?
Yes
Display length of the string
No
Is Option = 3 ?
Yes
Display reverse string
No
Is Option = 4 ?
Yes
Check palindrome
No
Display message enter correct option
Stop

Flowchart : String Reverse



Flowchart : String Palindrome



```
PROMPT MACRO MESSAGE ;Define macro with MESSAGE as a parameter
        PUSH AX                 ; save AX register
        MOV  AH, 09H            ; display message
        LEA  DX, MESSAGE
        INT  21H
        POP  AX                 ; restore AX register
        ENDM

.MODEL SMALL                    ; select small model
.STACK 100


.DATA                           ; start data segment
    MES1      DB 10, 13, '1. ENTER THE STRING $'
    MES2      DB 10, 13, '2. CALCULATE THE LENGTH OF STRING $'
    MES3      DB 10, 13, '3. REVERSE THE STRING $'
    MES4      DB 10, 13, '4. PALINDROME $'
    MES5      DB 10, 13, '5. EXIT $'
    MES6      DB 10, 13, 'ENTER THE CHOICE : $'
    MES7      DB 10, 13, 'ENTER CORRECT CHOICE : $'
    MES8      DB 10, 13, '$'
    MES9      DB 10, 13, 'FAILED : STRING IS MISSING - PLEASE
                           ENTER THE STRING$'
    MES10     DB 10, 13, 'STRING LENGTH IN DECIMAL IS : $'
    MES11     DB 10, 13, 'STRING IS NOT PALINDROME $'
    MES12     DB 10, 13, 'STRING IS PALINDROME $'

    FLAG      DB 0
    MES13     DB 10, 13, 'ENTER THE STRING: $'
    MES14     DB 10, 13, 'THE STRING IS : $'

    BUFF      DB 80
              DB 0
              DB 80 DUP(0)
    COUNTER1 DW 0
    COUNTER2 DW 0
```

```
        NUMBER   DW ?       ; define NUMBER
.CODE                       ; start code segment
START:  MOV AX, @DATA       ; [Initialize
        MOV DS, AX          ;  data segment]

BEGIN:  PROMPT MES8         ; Display MES8
        PROMPT MES8         ; Display MES8
        PROMPT MES1         ; Display MES1
        PROMPT MES2         ; Display MES2
        PROMPT MES3         ; Display MES3
        PROMPT MES4         ; Display MES4
        PROMPT MES5         ; Display MES5
        PROMPT MES6         ; Display MES6

AGAIN:  MOV      AH,01      ; [ READ
        INT      21H        ;   OPTION ]


        PROMPT   MES8       ; Display MES8

        CMP      AL,'5'     ; [ If choice is 5
        JZ       LAST       ;   exit ]

        CMP      AL,'1'     ; [ If choice is 1
        JNZ      NEXT1
        CALL     E_STR      ;  Enter the string
        PROMPT   MES8
        PROMPT   MES14
        CALL     D_STR      ;  Display the string
        JMP      BEGIN      ;   exit ]


NEXT1:  CMP      AL,'2'     ; [ If choice is 2
        JNZ      NEXT2
        CALL     L_STR      ;   Calculate the length of the string
        JMP      BEGIN      ;   exit ]

NEXT2:  CMP      AL,'3'     ; [ If choice is 3
        JNZ      NEXT3
        CALL     R_STR      ;   Reverse the string
```

```
        JMP     BEGIN    ;   exit ]

NEXT3:  CMP     AL,'4'   ; [ If choice is 4
        JNZ     NEXT4
        CALL    P_STR    ; Palindrome of the string
        JMP     BEGIN    ;   exit ]

NEXT4:  PROMPT  MES7     ; Display MES7
        JMP     AGAIN

LAST:   MOV     AH,4CH   ; Return to DOS
        INT     21H

E_STR PROC NEAR
        PROMPT  MES13    ; Display message MES13
        MOV     AH,0AH
        LEA     DX,BUFF  ; I/P the string.
        INT     21H
        MOV     FLAG,1
        RET
        ENDP

L_STR PROC NEAR
        CMP     FLAG,0
        JNZ     SKIP
        PROMPT  MES9
        RET
SKIP:   MOV     AL,BUFF+1
        PROMPT  MES10
        CALL    D_HEX
        RET
        ENDP

R_STR PROC NEAR
        CMP     FLAG,0
        JNZ     SKIP1
        PROMPT  MES9
        RET
```

```
SKIP1:  CALL    DR_STR
        RET
        ENDP

P_STR PROC NEAR
        LEA     BX,BUFF+2       ; Get starting address of string
        MOV     CH,00H
        MOV     CL,BUFF+1
        MOV     DI,CX
        DEC     DI
        SAR     CL,1
        MOV     SI,00H
BACK4:  MOV     AL,[BX + DI] ; Get the right most character
        MOV     AH,[BX + SI] ; Get the left most character
        CMP     AL,AH        ; Check for palindrome
        JNZ     LAST2        ; If not exit
        DEC     DI           ; Decrement end pointer
        INC     SI           ; Increment starting pointer
        DEC     CL           ; Decrement counter
        JNZ     BACK4        ; If count not = 0, repeat
        PROMPT  MES12        ; Display message 12
        RET
LAST2:  PROMPT  MES11        ; Display message 11
        RET
        ENDP


D_STR PROC NEAR
        LEA     BX,BUFF
        MOV     CH,00H       ; [ Take character
        MOV     CL,BUFF +1   ;   count in
        MOV     DI,00        ;   DI ]
BACK:   MOV     DL,[BX+DI+2] ; Point to the start
                             ; character and read it
        MOV     AH,02H
        INT     21H          ; Display the character
        INC     DI           ; Decrement count
        LOOP    BACK         ; Repeat until count is 0
        RET
        ENDP
```

```
DR_STR  PROC  NEAR
        LEA  BX,BUFF
        MOV  CH,00H        ; [ Take character
        MOV  CL,BUFF+1     ;    count in
        MOV  DI,CX         ;    DI ]
BACK3:  MOV  DL,[BX+DI+1]  ; Point to the start
                           ; character and read it
        MOV  AH,02H
        INT  21H           ; Display the character
        DEC  DI            ; Decrement count
        JNZ  BACK3         ; Repeat until count is 0
        RET
        ENDP

D_HEX  PROC  NEAR
        MOV AH, 00H        ; Clear AH
        AAM                ; Convert to BCD
        ADD AX, 3030H      ; Convert to ASCII
        MOV BX,AX          ; Save result
        MOV DL, BH         ; Load first digit (MSD)
        MOV AH, 02         ; Load function number
        INT 21H            ; Display first digit (MSD)
        MOV DL, BL         ; Load second digit (LSD)
        INT 21H            ; Display second digit (LSD)
        RET
        ENDP
END
```

Program 24 : String manipulations

Program Statement :

Write 8086 ALP to perform string manipulation. The strings to be accepted from the user is to be stored in code segment Module_1 and write FAR PROCEDURES in code segment Module_2 for following operations on the string.

a. Concatenation of two strings.

b. Compare two strings.

c. Number of occurrences of a sub-string in the given string.

d. Find number of words, characters and capital letters from the given text in the data segment.

Note : Use PUBLIC and EXTERN directive. Create •OBJ files of both the modules and link them to create an EXE file. Command : Tlink M1.0BJ M2.0BJ

In this experiment we have to write two •asm programs one for accepting strings and one for procedures.

Algorithm : Module_1

1. Display Menu
   a. Enter the strings.
   b. Concatenation of two strings.
   c. Compare two strings.
   d. Find number of occurrences of a substring.
   e. Find words, characters and capital letters.
   f. Exit.
2. Read option
   It's option is
   1. Read two strings.
   2. Concatenate two strings.
   3. Compare two strings.
   4. Find number of occurrences of a substring.
   5. Find words, characters and capital letters.
   6. Exit.
3. Stop

M1 : String operations

```
PROMPT MACRO MESSAGE ;Define macro with MESSAGE as a parameter
              PUSH    AX              ; save registers
              PUSH    DX
              MOV     AH, 09H         ; display message
              LEA     DX, MESSAGE
              INT     21H
              POP     DX              ; restore registers
              POP     AX
              ENDM


.MODEL SMALL                          ; select small model
.STACK 100
```

```
.DATA                ; start data segment
        PUBLIC BUFF1
        PUBLIC BUFF2
        PUBLIC BUFF3
    MES1     DB 10, 13, '1. ENTER THE STRING $'
    MES2     DB 10, 13, '2. CONCATENATION OF TWO STRINGS $'
    MES3     DB 10, 13, '3. COMPARE TWO STRINGS $'
    MES4     DB 10, 13, '4. NUMBER OF OCCURENCES OF A
                        SUBSTRING $'
    MES5     DB 10, 13, '5. FIND WORDS,CHARACTERS AND CAPITAL
                        LETTERS $'
    MES6     DB 10, 13, '6. EXIT $'
    MES7     DB 10, 13, 'ENTER THE CHOICE : $'
    MES8     DB 10, 13, 'ENTER CORRECT CHOICE : $'
    MES9     DB 10, 13,  '$'
    MES10    DB 10, 13, 'STRING IS MISSING - PLEASE ENTER
                        THE STRING$'
    MES11    DB 10, 13, 'CONCATENATED STRING IS : $'
    MES12    DB 10, 13, 'TWO STRINGS ARE SAME $'
    MES13    DB 10, 13, 'TWO STRINGS ARE NOT SAME $'

    FLAG     DB 0
    MES14    DB 10,13, 'ENTER THE STRING: $'
    MES15    DB 10,13, 'THE STRING IS : $'

    BUFF1    DB 80
             DB 0
             DB 80 DUP(0)
    BUFF2    DB 80
             DB 0
             DB 80 DUP(0)
    BUFF3    DB 80
             DB 0
             DB 80 DUP(0)
.CODE                     ; start code segment

     EXTRN CON_STR:FAR
     EXTRN COM_STR:FAR
     EXTRN SUB_STR:FAR
     EXTRN FWCC_STR:FAR
```

```
START:  MOV     AX, @DATA      ; [Initialize
        MOV     DS, AX         ;  data segment]
        MOV     ES, AX
BEGIN:  PROMPT  MES9           ; Display MES9
        PROMPT  MES9           ; Display MES9
        PROMPT  MES1           ; Display MES1
        PROMPT  MES2           ; Display MES2
        PROMPT  MES3           ; Display MES3
        PROMPT  MES4           ; Display MES4
        PROMPT  MES5           ; Display MES5
        PROMPT  MES6           ; Display MES6
        PROMPT  MES7           ; Display MES7

AGAIN:  MOV     AH,01          ; [ READ
        INT     21H            ;   OPTION ]

        CMP     AL,'6'         ; [ If choice is 6
        JZ      LAST           ;   exit ]

        MOV     BL,FLAG        ; Check for first occurrence
        CMP     BL,0
        JNZ     SKIP           ; if not skip
        CMP     AL,'1'         ; check if choice is 1
        JE      SKIP           ; if yes skip
        PROMPT  MES10          ; otherwise give error message
        JMP     BEGIN          ; and enter the strings

  SKIP: PROMPT  MES9           ; Display MES9

        CMP     AL,'1'         ; [ If choice is 1
        JNZ     NEXT1

        LEA     DX, BUFF1
        CALL    E_STR          ;  Enter the string1

        LEA     DX, BUFF2
        CALL    E_STR          ;  Enter the string2
        MOV     FLAG,1
        JMP     BEGIN          ;   exit ]
```

```
NEXT1:  CMP     AL,'2'   ; [ If choice is 2
        JNZ     NEXT2
        CALL    CON_STR  ;   Concatenate two strings
        JMP     LAST     ;   exit ]


NEXT2:  CMP     AL,'3'   ; [ If choice is 3
        JNZ     NEXT3
        CALL    COM_STR  ;   Compare two string
        JMP     BEGIN    ;   exit ]

NEXT3:  CMP     AL,'4'   ; [ If choice is 4
        JNZ     NEXT4
        CALL    SUB_STR  ; Find number of occurences of a
                         ; sub-string in the given string
        JMP     BEGIN    ;   exit ]
NEXT4:  CMP     AL,'5'   ; [ If choice is 4
        JNZ     NEXT5
        CALL    FWCC_STR ; Find word, character and capital
                         ; letters in the string
        JMP     BEGIN    ;   exit ]

NEXT5:  PROMPT  MES8     ; Display MES8
        JMP     AGAIN

LAST:   MOV     AH,4CH   ; Return to DOS
        INT     21H

E_STR PROC NEAR
        PROMPT  MES1     ; Display message MES1
        MOV     AH,0AH   ; I/P the string.
        INT     21H
        RET
        ENDP

END
```

M2 : For string operations

```
.MODEL SMALL
.STACK 100
.DATA
        EXTRN   BUFF1:BYTE
        EXTRN   BUFF2:BYTE
        EXTRN   BUFF3:BYTE
        MESS1   DB 10,13,'STRINGS ARE SAME $'
        MESS2   DB 10,13,'STRINGS ARE NOT SAME $'
        MESS3   DB 10,13,'NUMBER OF ALPHABETS IN THE STRING
                ARE:$'
        MESS4   DB 10,13,'NUMBER OF CAPITAL LETTERS IN THE STRING
                ARE:$'
        MESS5   DB 10,13,'NUMBER OF WORDS IN THE STRING ARE : $'
        MESS6   DB 10,13,'NUMBER OF OCCURRENCES OF SUBSTRING IN
                THE STRING ARE : $'
        WFLAG   DB   0
        ACOUNT  DB   0
        CCOUNT  DB   0
        WCOUNT  DB   1
        C_ADDR  DW   ? ; current address of pointer
        E_ADDR  DW   ? ; End address of string

.CODE

        PUBLIC CON_STR
        PUBLIC COM_STR
        PUBLIC SUB_STR
        PUBLIC FWCC_STR

CON_STR PROC FAR

        CLD
        MOV CH,00        ; copy string 1
        MOV CL, BUFF1+1
        LEA SI, BUFF1+2
        LEA DI, BUFF3+2
        REPZ MOVSB

        MOV CH,00        ; copy string 2
        MOV CL, BUFF2+1
```

```
        LEA  SI, BUFF2+2
        REPZ MOVSB

        MOV  CL, BUFF1+1    ; calculate and store length of
                              concatenated string
        ADD  CL, BUFF2+1    ;
        MOV  BUFF3+1,CL

        MOV  CH,00          ; Display concatenated string
        LEA  SI,BUFF3+2
DISNEXT: MOV AH,02H
        MOV  DL,[SI]
        INT  21H
        INC  SI
        LOOP DISNEXT
        RET
        CON_STR  ENDP

COM_STR PROC FAR

        MOV  CH,BUFF1+1     ; check two string character by character
        MOV  CL,BUFF2+1
        CMP  CH,CL
        JNZ  NOTEQ
        CLD
        MOV  CH,00
        LEA  SI,BUFF1+2
        LEA  DI,BUFF2+2
        REPE CMPSB
        JNZ  NOTEQ

        MOV  AH,09H         ; if equal display message accordingly
        LEA  DX,MESS1
        INT  21H
        JMP  RE
NOTEQ:  MOV  AH,09H         ; if not equal display message
                              accordingly
        LEA  DX,MESS2
        INT  21H
```

```
RE:       RET
          COM_STR   ENDP

 SUB_STR PROC FAR

          MOV BL,00
          LEA SI, BUFF1+2
          MOV C_ADDR,SI       ; Load current address
          MOV DL, BUFF1+1
          MOV DH,00
          MOV AX, SI
          ADD AX,DX
          MOV E_ADDR,AX       ; load end address
ST1:      MOV CH,0
          MOV CL,BUFF2+1      ; load length of substring
          LEA DI,BUFF2+2      ; initialize pointer to substring
BBB:      MOV BH,[SI]
          CMP BH,BYTE PTR [DI]  ; compare substring characters
          JNZ NNNEXT          ; if not equal go to NNNEXT
          INC SI              ; otherwise increment character pointers
          INC DI              ; and confine
          LOOP BBB
          INC BL              ; if substring occurs increment count
          CMP SI, E_ADDR      ; check for end of string
          JNZ ST1             ; if not zero go to check more
                                occurrences
          JMP LLLAST          ; if end of string go to display number
                                of occurrences
NNNEXT:   MOV SI,C_ADDR
          INC SI
          MOV C_ADDR,SI       ; modify current address
          CMP SI, E_ADDR
          JNZ ST1
LLLAST:
          MOV AH,09H          ; display number of occurrences of string
          LEA DX, MESS6
          INT 21H
          MOV AL,BL
          CALL DIS_HEX
          RET
```

```
        SUB_STR   ENDP

FWCC_STR PROC  FAR

        MOV  CH,  00
        MOV  CL,  BUFF1+1
        LEA  SI,  BUFF1+2
BB:     MOV  AL,[SI]         ; check of space
        CMP  AL,' '
        JNZ  NNEXT
        MOV  AL,WFLAG        ; if space occurs increment word count
        CMP  AL,0
        JZ   LLAST
        MOV  WFLAG,0
        INC  WCOUNT
        JMP  LLAST
NNEXT:  MOV  WFLAG,1
                             ; .IF AL >= 'A' && AL <= 'Z'
        CMP  AL,'A'
        JB   LLAST           ; check if alphabet
        CMP  AL,'Z'          ; if yes increment alphabet count
        JA   NNEXT1
        INC  ACOUNT
        INC  CCOUNT
                             ;.ENDIF

 NNEXT1:                     ;.IF AL >= 'a' && AL <= 'z'
        CMP  AL,'a'
        JB   LLAST           ; check if alphabet
        CMP  AL,'z'          ; if yes increment alphabet count
        JA   LLAST
        INC  ACOUNT
                             ;.ENDIF
LLAST:  INC SI
        LOOP BB

        MOV AH,09H           ; display alphabet count
        LEA DX,MESS3
        INT 21H
        MOV AL,ACOUNT
```

```
        CALL DIS_HEX

        MOV  AH,09H           ; display character count
        LEA  DX,MESS4
        INT  21H
        MOV  AL,CCOUNT
        CALL DIS_HEX

        MOV  AH,09H           ; display word count
        LEA  DX,MESS5
        INT  21H
        MOV  AL,WCOUNT
        CALL DIS_HEX

        MOV  ACOUNT,0
        MOV  CCOUNT,0
        MOV  WCOUNT,1

        RET
        FWCC_STR   ENDP

DIS_HEX PROC NEAR
    MOV AH, 00H                   ; Clear AH
    AAM
                                  ; Convert to BCD
    ADD AX, 3030H                 ; Convert to ASCII
    MOV BX,AX                     ; Save result
    MOV DL, BH                    ; Load first digit (MSD)
    MOV AH, 02                    ; Load function number
    INT 21H                       ; Display first digit (MSD)
    MOV DL, BL                    ; Load second digit (LSD)
    INT 21H                       ; Display second digit (LSD)
    RET
    ENDP
END
```

Program 25 : Sorting of array

Program Statement : Write 8086 ALP to arrange the numbers stored in the array in ascending as well as descending order. Assume that the first location in the array holds the number of elements in the array and successive memory locations will be actual array elements. Write separate subroutine to arrange the numbers in ascending and descending order. Accept key from the user.

If user enters 1 : Arrange in ascending order

If user enters 2 : Arrange in descending order

Sorting of Array

```
PROMPT MACRO MESSAGE              ; Define macro with MESSAGE as a
        PUSH     AX               ; parameter save register
        MOV      AH, 09H          ; display message
        LEA      DX, MESSAGE
        INT      21H
        POP      AX               ; restore register
        ENDM

.MODEL SMALL
.STACK 100
.DATA
    ARRAY    DB   10, 53H,20H,30H,25H,50H,09H,70H,13H,90H,00H
    MES1     DB   10,13, '1. SORT ARRAY IN THE ASCENDING ORDER $'

    MES2     DB   10,13, '2. SORT ARRAY IN THE DESCENDING ORDER $'
    MES3     DB   10,13, '3. EXIT $'
    MES4     DB   10,13, 'ENTER THE CHOICE :   $'
    MES5     DB   10,13, 'SORTED ARRAY IS : $'
    MES6     DB   10,13, 'ENTER CORRECT CHOICE : $'
    MES7     DB   10,13, '$'
.CODE
START:   MOV      AX,@data        ; [ Initialise
         MOV      DS,AX           ;   data segment ]

         PROMPT   MES1
         PROMPT   MES2
         PROMPT   MES3
         PROMPT   MES4
```

```
     ST1: MOV     AH,01H
          INT     21H

          CMP     AL,'3'
          JZ      LAST

          CMP     AL,'1'
          JNZ     NEXT
          PROMPT  MES7
          CALL    ASC
          JMP     LAST
NEXT:     CMP     AL,'2'
          JNZ     NEXT1
          PROMPT  MES7
          CALL    DSC
          JMP     LAST
NEXT1:    PROMPT  MES6
          JMP     ST1
LAST:     MOV     AH, 4CH
          INT     21H
ASC PROC NEAR
          MOV     CL,ARRAY        ; Initialise counter1
BBB1:     MOV     CH,ARRAY        ; Initialise counter2
          DEC     CH
          XOR     DI,DI           ; Initialise pointer
          LEA     BX,ARRAY        ; Initialise array base pointer
BACK1:    MOV     DL,[BX+DI+1]    ; Get the number
          CMP     DL,[BX+DI+2]    ; Compare it with next number
          JBE     SKIP1

          MOV     AH,[BX+DI+2]    ; Otherwise
          MOV     [BX+DI+2],DL    ; exchange
          MOV     [BX+DI+1],AH    ; two numbers

SKIP1:    INC     DI
          DEC     CH
          JNZ     BACK1
          DEC     CL
          JNZ     BBB1
```

```
        PROMPT  MES5
        MOV     CH,00
        MOV     CL,ARRAY
        LEA     DI,ARRAY
        INC     DI
AG1:    INC     DI
        MOV     AL,[DI]
        CALL    D_HEX2          ; Display sorted array
        PUSH    AX
        PUSH    DX
        MOV     AH,02H
        MOV     DL,' '
        INT     21H
        POP     DX
        POP     AX
        LOOP    AG1
        RET
        ENDP

DSC PROC NEAR
        MOV CL,ARRAY            ; Initialise counter1
BBB:    MOV CH,ARRAY            ; Initialise counter2
        DEC CH
        XOR DI,DI               ; Initialise pointer
        LEA BX,ARRAY            ; Initialise array base pointer
BACK:   MOV DL,[BX+DI+1]        ; Get the number
        CMP DL,[BX+DI+2]        ; Compare it with next number
        JAE SKIP

        MOV AH,[BX+DI+2]        ; Otherwise
        MOV [BX+DI+2],DL        ; exchange
        MOV [BX+DI+1],AH        ; two numbers

SKIP:   INC DI
        DEC CH
        JNZ BACK
        DEC CL
        JNZ BBB
```

```
        PROMPT  MES5
        MOV     CH,00
        MOV     CL,ARRAY
        LEA     DI,ARRAY
        INC     DI
AG:     INC     DI
        MOV     AL,[DI]
        CALL    D_HEX2          ; Display sorted array
        PUSH    AX
        PUSH    DX
        MOV     AH,02H
        MOV     DL,' '
        INT     21H
        POP     DX
        POP     AX
        LOOP    AG
        RET
        ENDP

D_HEX2 PROC NEAR
        PUSH    CX
        MOV     CL, 04H         ; Load rotate count
        MOV     CH, 02H         ; Load digit count
BAC:    ROL     AX, CL          ; rotate digits
        PUSH    AX              ; save contents of AX
        AND     AL, 0FH         ; [Convert
        CMP     AL,9            ;  number
        JBE     Add30           ; to
        ADD     AL, 37H         ; its
        JMP     DISP            ; ASCII
Add30:
        ADD     AL,30H          ; equivalent]
```

```
DISP:    MOV  AH,02H
         MOV  DL,AL          ; [Display the
         INT  21H            ;  number]
         POP  AX             ; restore contents of AX
         DEC  CH             ; decrement digit count
         JNZ  BAC            ; if not zero repeat
         POP  CX
         RET
         ENDP
         END
```

Program 26 : Program to search a given byte in the string

```
.MODEL SMALL
.DATA
        M1      DB   10, 13, 'ENTER THE STRING : $'
        M2      DB   10, 13, 'GIVEN BYTE IS NOT IN THE STRING $'
        CHAR    DB   0
        ADDR    DB   0
        BUFF    DB   80
                DB   0
                DB   80 DUP (0)
.CODE
START :         MOV  AX,@data       ; [ Initialise
                MOV  DS,AX          ;   data segment ]
                MOV  AH,09H         ; Display message1
                MOV  DX,OFFSET M1
                INT  21H
                MOV  AH,0AH         ; Input the string
                LEA  DX,BUFF
                INT  21H
                MOV  AH,01          ; [Read character
                INT  21H            ;  from keyboard]
                MOV  CHAR,AL        ; save character
                MOV  CH,00H
                MOV  CL,BUFF+1      ; Take character count in CX
                LEA  BX,BUFF+2
                MOV  DI,00H
BACK :          MOV  DL,[BX+DI]     ; point to the first character
                CMP  DL, CHAR       ; compare string character with
                                    ; given character
                JZ   NEXT           ; if match occurs go to next
                INC  DI
                DEC  CX             ; Decrement character counter
                JNZ  BACK           ; If not = 0, repeat
                MOV  AH,09H         ; [Display message M2
                LEA  DX,M2          ;  on the
                INT  21H            ;  monitor]
```

```
                JMP  LAST
NEXT:           MOV  ADDR,DI       ; save relative address of the
                                   ; byte from the starting
                                   ; location of the string
LAST:           MOV  AH,4CH        ; [ Terminate and
                INT  21H           ;   Exit to DOS ]
                END  START
```

Program 27 : Program to find LCM of two 16-bit unsigned numbers

(Softcopy of this program, P24.asm is available at www.vtubooks.com)

If we divide the first number by the second number and there is no remainder, then the first number is the LCM. In case of remainder, it is necessary to add first number to it to get the new first number. After addition we have to divide the new first number by the second number to check if the remainder is zero. If remainder is not zero again add the original first number to new one and repeat the process.

For examp'e, if two numbers are 20 and 15 then we get LCM as follows :

$$20 \div 15 = 1 \text{ Remainder } 5 \text{ i.e. } \neq 0$$

$$\therefore \quad 20 + 20 = 40 \div 15 = 2 \text{ Remainder } 10 \text{ i.e. } \neq 0$$

$$\therefore \quad 40 + 20 = 60 \div 15 = 4 \text{ Remainder } 0$$

$$\therefore \quad \text{LCM} = 60$$

```
NAME LCM
PAGE 60,80
TITLE program to find LCM of two 16-bit unsigned numbers
.MODEL SMALL
.STACK 64
.DATA
    NUMBERS   DW 0020, 0015
    LCM       DW 2 DUP (?)
.CODE
START:        MOV AX,@DATA       ; [Initialize
              MOV DS,AX          ;  data segment]
              MOV DX,0
              MOV AX,NUMBERS     ; Get the first number
              MOV BX,NUMBERS+2   ; Get the second number
BACK:         PUSH AX            ; [ Save the
              PUSH DX            ;   first number]
              DIV BX             ; Divide if by second number
              CMP DX,0           ; Check if remainder = 0
              JE EXIT            ; if remainder = 0 then exit
              POP DX
              POP AX
              ADD AX,NUMBERS     ; First number + first number
              JNC SKIP
              INC DX
```

```
SKIP:          JMP BACK          ; Goto BACK
EXIT:          POP LCM+2         ; [ Get
               POP LCM           ;   the LCM ]
               MOV AH,4CH        ; [ Terminate and
               INT 21H           ;   Exit to DOS ]
               END START
```

Program 28 : Program to find HCF of two numbers.

(Softcopy of this program, P25.asm is available at www.vtubooks.com)

To find the HCF of two numbers we have to divide greater number by smaller number, if remainder is zero, divisor is a HCF. If remainder is not zero, remainder becomes new divisor and previous divisor becomes dividend and this process is repeated until we get remainder 0.

For example, if numbers are 20 and 15 we can find HCF as follows :

$$20 \div 15 = 1 \text{ Remainder } 5 \text{ i.e. } \neq 0$$

$$\therefore \quad 15 \div 5 = 3 \text{ Remainder } 0$$

$$\therefore \quad \text{HCF} = 5$$

```
.model small
.stack 100
.data
     CR         EQU 0AH
     LF         EQU 0DH
     MES_1      DB   CR,LF,'ENTER 4-DIGIT FIRST HEX NO',CR,LF,'$'
     MES_2      DB   CR,LF,'ENTER 4-DIGIT SECOND HEX NO',CR,LF,'$'
     MES_3      DB   CR,LF,'INPUT IS INVALID BCD $'
     MES_4      DB   CR,LF,'THE HCF IS : $'
     MULTI           DW 1,10,100,1000
     RESULT          DW (00)
     DIVISOR         DW (00)
     DIVIDEND        DW (00)

     INP1            DB 05
                     DB 00
                     DB 05 DUP(0)
     INP2            DB 05
                     DB 00
                     DB 05 DUP(0)
.code
MAIN:          MOV AX,@data           ; [ Initialise
               MOV DS,AX              ;   data segment ]
               MOV AH,09H             ; [  Display
               MOV DX,OFFSET MES_1    ;    MES_1
               INT 21H                ;    on video screen ]
               LEA DX,INP1            ; [  Get the
               MOV AH,0AH             ;    First
```

```
          INT  21H                  ;     HEX number ]
          MOV  AH,09H               ; [   Display
          MOV  DX,OFFSET MES_2      ;     MES_2
          INT  21H                  ;     on video screen ]
          LEA  DX,INP2              ; [Get the
          MOV  AH,0AH               ;    Second
          INT  21H                  ;    HEX number ]
          MOV  CH,02H               ; Initialize buffer counter
          LEA  BX,INP1              ; Get the address of buffer
AGAIN:    INC  BX                   ; [   Adjust buffer
          INC  BX                   ;     pointer ]
          XOR  DI,DI                ; Clear pointer
          MOV  CL,04                ; Initialize counter for digits
BACK:     MOV  AL,[BX+DI]           ; Get the digit from buffer
          CMP  AL,39H               ; [ Convert
          JG    NEXT                ;    the ASCII
          SUB  AL,30H               ;    code of
          JMP  SKIP                 ;    the actual number
NEXT:     SUB  AL,37H               ;    and store it in the same
SKIP:     MOV  [BX+DI],AL           ;  position ]
          INC  DI                   ; Increment pointer
          DEC  CL                   ; Decrement digit counter
          JNZ  BACK                 ; If not zero goto BACK
          LEA  BX,INP2              ; Point to second buffer
          DEC  CH                   ; Decrement buffer counter
          JNZ  AGAIN                ; If not zero goto AGAIN
          MOV  CL,4                 ; Initialize rotation counter
          LEA  BX,INP1              ; Point to first buffer
          INC  BX                   ; [Adjust buffer
          INC  BX                   ;  pointer ]
          MOV  AH,[BX+0]            ; [Forms the
          SAL  AH,CL                ;  packed BCD
          AND  AH,0F0H              ;  Higher
          MOV  AL,[BX+1]            ;  Byte ]
          OR   AH,AL
          MOV  AL,[BX+2]            ; [Forms the
          SAL  AL,CL                ;  packed BCD
          AND  AL,0F0H              ;  Lower
          MOV  DH,[BX+3]            ;  byte ]
          OR   AL,DH
          MOV  RESULT,AX            ; Save packed word as a RESULT
          MOV  CL,4                 ; Initialize rotation counter
          LEA  BX,INP2              ; Point to second buffer
          INC  BX                   ; [   Adjust buffer
          INC  BX                   ;     pointer ]
          MOV  AH,[BX+0]            ; [Forms the
          SAL  AH,CL                ;  packed BCD
          AND  AH,0F0H              ;  Higher
          MOV  AL,[BX+1]            ;  byte]
          OR   AH,AL
```

```
          MOV AL,[BX+2]          ; [Forms the
          SAL AL,CL              ;  packed BCD
          AND AL,0F0H            ;  lower
          MOV DH,[BX+3]          ;  byte ]
          OR  AL,DH
          CMP AX,RESULT          ; Compare two packed words
          JNC NEXT1
          MOV DIVISOR,AX         ; Assign smaller word as a
          MOV CX,RESULT          ; DIVISOR and
          MOV DIVIDEND,CX        ; greater word as a DIVIDEND
          JMP SKIP1
NEXT1:    MOV DIVIDEND,AX        ; Assign greater word as a
          MOV CX,RESULT          ; DIVIDEND and
          MOV DIVISOR,CX         ; smaller word as a DIVISOR
SKIP1:    MOV DX,0
          MOV AX,DIVIDEND
          DIV DIVISOR            ; Perform division
          CMP DX,0               ; Check remainder for zero
          MOV CX,DIVISOR
          MOV DIVISOR,DX         ; Load remainder as a new
                                 ; DIVISOR
          MOV DIVIDEND,CX        ; Load previous DIVISOR as a
                                 ; new DIVIDEND
          JNZ SKIP1              ; If remainder is not zero
                                 ; goto SKIP1
          MOV AH,09H             ; [Display
          LEA DX,MES_4           ;  MES_4
          INT 21H                ;  on video screen ]
          ADD CL,30H             ; [Display the DIVISOR
          MOV DL,CL              ;  when remainder
          MOV AH,02H             ;  is zero
          INT 21H                ;  i.e. HCF ]
          MOV AH,4CH             ; [Terminate and
          INT 21H                ;   Exit to DOS ]
          END MAIN
          END
```

Program 29 : Program to find LCM of two given numbers.

(Softcopy of this program, P26.asm is available at www.vtubooks.com)

There is a one more method to find LCM of two number if HCF is known. We can find LCM as follows :

$$LCM = [number1 \times number\ 2] \div HCF$$

This program accepts two four digit numbers from keyboard, finds HCF first and using above equation it then finds LCM of the two numbers.

```
.model small
.stack 100
.data
        CR          EQU 0AH
        LF          EQU 0DH
        MES_1       DB CR,LF,'ENTER 4-DIGIT FIRST HEX NO',CR,LF,'$'
        MES_2       DB CR,LF,'ENTER 4-DIGIT SECOND HEX NO',CR,LF,'$'
        MES_3       DB CR,LF,'INPUT IS INVALID BCD $'
        MES_4       DB CR,LF,'THE HCF IS : $'
        MULTI           DW 1,10,100,1000
        RESULT          DW (00)
        DIVISOR         DW (00)
        DIVIDEND        DW (00)

        INP1            DB 05
                        DB 00
                        DB 05 DUP(0)
        INP2            DB 05
                        DB 00
                        DB 05 DUP(0)
.code
MAIN:   MOV AX,@data            ; [  Initialise
        MOV DS,AX               ;    data segment]
        MOV AH,09H              ; [  Display
        MOV DX,OFFSET MES_1     ;    MES_1
        INT 21H                 ;    on video screen ]
        LEA DX,INP1             ; [  Get the
        MOV AH,0AH              ;    First
        INT 21H                 ;    HEX number ]
        MOV AH,09H              ; [  Display
        MOV DX,OFFSET MES_2     ;    MES_2
        INT 21H                 ;    on video screen ]
        LEA DX,INP2             ; [  Get the
        MOV AH,0AH              ;    Second
        INT 21H                 ;    HEX number ]
        MOV CH,02H              ; Initialize buffer counter
        LEA BX,INP1             ; Get the buffer pointer
AGAIN:  INC BX                  ; [  Adjust buffer
        INC BX                  ;    pointer ]
        XOR DI,DI               ; Clear pointer
        MOV CL,04               ; Initialize counter for digits
BACK:   MOV AL,[BX+DI]          ; Get the digit from buffer
        CMP AL,39H              ; [ Convert
        JG   NEXT               ;    the ASCII
        SUB AL,30H              ;    code
        JMP SKIP                ;    the actual number
```

```
NEXT:   SUB AL,37H          ;     and store it in the same
SKIP:   MOV [BX+DI],AL      ;     position ]
        INC DI              ; Increment pointer
        DEC CL              ; Decrement digit counter
        JNZ BACK            ; If not zero goto BACK
        LEA BX,INP2         ; Point to second buffer
        DEC CH              ; Decrement buffer counter
        JNZ AGAIN           ; If not zero goto AGAIN
        MOV CL,4            ; Initialize rotation counter
        LEA BX,INP1         ; Point to first buffer
        INC BX              ; [   Adjust buffer
        INC BX              ;     pointer ]
        MOV AH,[BX+0]       ; [   Forms the
        SAL AH,CL           ;     packed BCD
        AND AH,0F0H         ;     Higher
        MOV AL,[BX+1]       ;     Byte ]
        OR  AH,AL           ;
        MOV AL,[BX+2]       ; [   Forms the
        SAL AL,CL           ;     packed BCD
        AND AL,0F0H         ;     Lower
        MOV DH,[BX+3]       ;     byte ]
        OR  AL,DH
        MOV RESULT,AX       ; Save the packed word as a
                            ; RESULT
        MOV CL,4            ; Initialize rotation counter
        LEA BX,INP2         ; Point to second buffer
        INC BX              ; [   Adjust buffer
        INC BX              ;     pointer ]
        MOV AH,[BX+0]       ; [   Forms the
        SAL AH,CL           ;     packed BCD
        AND AH,0F0H         ;     Higher
        MOV AL,[BX+1]       ;     byte ]
        OR  AH,AL
        MOV AL,[BX+2]       ; [   Forms the
        SAL AL,CL           ;     packed BCD
        AND AL,0F0H         ;     lower
        MOV DH,[BX+3]       ;     byte ]
        OR  AL,DH
        MOV RESULT1,AX      ; Save second pack word as
                            ; a RESULT2
        CMP AX,RESULT       ; Compare two packed words
        JNC NEXT1
```

```
        MOV  DIVISOR,AX          ; Assign smaller word as a
        MOV  CX,RESULT           ; DIVISOR and
        MOV  DIVIDEND,CX         ; greater word as a DIVIDEND
        JMP  SKIP1
NEXT1:
        MOV  DIVIDEND,AX         ; Assign greater word as a
        MOV  CX,RESULT           ; DIVIDEND and
        MOV  DIVISOR,CX          ; smaller word as a DIVISOR
SKIP1:  MOV  DX,0
        MOV  AX,DIVIDEND
        DIV  DIVISOR             ; Perform division
        CMP  DX,0                ; Check remainder for zero
        MOV  CX,DIVISOR
        MOV  DIVISOR,DX          ; Load remainder as a new
                                 ; DIVISOR
        MOV  DIVIDEND,CX         ; Load previous DIVISOR as a
                                 ; new DIVIDEND
        JNZ  SKIP1               ; If remainder is not zero
                                 ; goto SKIP1
        MOV  AH,09H              ; [  Display
        LEA  DX,OFFSET MES_3     ;    MES_3
        INT  21H                 ;    on video screen ]
; Number1 × Number2 = HCF × LCM ∴ LCM =(Number1 × Number2)/HCF
        MOV  HCF,CX
        MOV  DX,0
        MOV  AX,RESULT           ; Get the first number
        MUL  RESULT1             ; Multiply number1 and number2
        DIV  HCF                 ; Divide multiplication by HCF
        MOV  CL,4                ; Initialize rotation counter
        MOV  BX,AX               ; Save the quotient (LCM)
        AND  AH,0F0H             ; [  Display the LCM
        SAR  AH,CL               ;    on the video screen ]
        CMP  AH,09H
        JNC  SKIP2
        ADD  AH,30H
        JMP  NEXT2
SKIP2:  ADD  AH,37H
NEXT2:  MOV  DL,AH
        MOV  AH,02H
        INT  21H
        MOV  AX,BX
        AND  AH,0FH
        CMP  AH,09H
        JNC  SKIP3
        ADD  AH,30H
        JMP  NEXT3
```

```
SKIP3:   ADD AH,37H
NEXT3:    MOV DL,AH
         MOV AH,02H
         INT 21H
         MOV AX,BX
         AND AL,0F0H
         SAR AL,CL
         CMP AL,09H
         JNC SKIP4
         ADD AL,30H
         JMP NEXT4
SKIP4:   ADD AL,37H
NEXT4:   MOV DL,AL
         MOV AH,02H
         INT 21H
         MOV AX,BX
         AND AL,0FH
         CMP AL,09H
         JNC SKIP5
         ADD AL,30H
         JMP NEXT5
SKIP5:   ADD AL,37H
NEXT5:   MOV DL,AL
         MOV AH,02H
         INT 21H
         MOV  AH,4CH             ; [  Terminate and
         INT  21H                ;    Exit to DOS ]
         END MAIN
         END
```

❑❑❑

6

8086 Interrupts

6.1 Introduction

Sometimes it is necessary to have the computer automatically execute one of a collection of special routines whenever certain conditions exists within a program or in the microcomputer system. For example, it is necessary that microcomputer system should give response to devices such as keyboard, sensor and other components when they request for service.

The most common method of servicing such device is the **polled approach.** This is where the processor must test each device in sequence and in effect "ask" each one if it needs communication with the processor. It is easy to see that a large portion of the main program is looping through this continuous polling cycle. Such a method would have a serious and decremental effect on system throughput, thus limiting the tasks that could be assumed by the microcomputer and reducing the cost effectiveness of using such devices.

A more desirable method would be the one that allows the microprocessor to execute its main program and only stop to service peripheral devices when it is told to do so by the device itself. In effect, the method, would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is completed, the processor would resume exactly where it left off. This method is called **interrupt method.** It is easy to see that system throughput would drastically increase, and thus enhance its cost effectiveness. Most microprocessors allow execution of special routines by interrupting normal program execution. When a microprocessor is interrupted, it stops executing its current program and calls a special routine which "services" the interrupt. The event that causes the interruption is called **interrupt** and the special routine executed to service the interrupt is called **interrupt service routine/procedure.** Normal program can be interrupted by three ways :

1. By external signal
2. By a special instruction in the program or
3. By the occurrence of some condition.

An interrupt caused by an external signal is referred as a **hardware interrupt**. Conditional interrupts or interrupts caused by special instructions are called **software interrupts**.

6.2 Interrupt Cycle of 8086/88

An 8086 interrupt can come from any one the three sources :

- External signal
- Special instruction in the program
- Condition produced by instruction.

6.2.1 External Signal (Hardware Interrupt)

An 8086 can get interrupt from an external signal applied to the non maskable interrupt (NMI) input pin, or the interrupt (INTR) input pin.

6.2.2 Special Instruction

8086 supports a special instruction, INT to execute special program. At the end of the interrupt service routine, execution is usually returned to the interrupted program.

6.2.3 Condition Produced by Instruction

An 8086 is interrupted by some condition produced in the 8086 by the execution of an instruction. For example divide by zero : Program execution will automatically be interrupted if you attempt to divide an operand by zero.

At the end of each instruction cycle 8086 checks to see if there is any interrupt request. If so, 8086 responds to the interrupt by performing series of actions (Refer Fig. 6.1).

1. It decrements stack pointer by 2 and pushes the flag register on the stack .
2. It disables the INTR interrupt input by clearing the interrupt flag in the flag register.
3. It resets the trap flag in the flag register.
4. It decrements stack pointer by 2 and pushes the current code segment register contents on the stack.
5. It decrements stack pointer by 2 and pushes the current instruction pointer contents on the stack.
6. It does an indirect far jump at the start of the procedure by loading the CS and IP values for the start of the interrupt service routine (ISR).

An IRET instruction at the end of the interrupt service procedure returns execution to the main program.

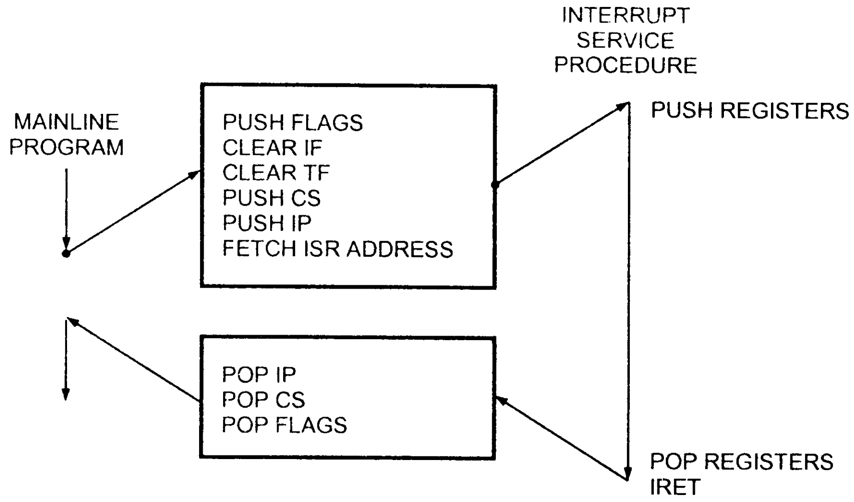


Fig. 6.1 8086 interrupt response

Now the question is "How to get the values of CS and IP register ?" The 8086 gets the new values of CS and IP register from four memory addresses. When it responds to an interrupt, the 8086 goes to memory locations to get the CS and IP values for the start of the interrupt service routine. In an 8086 system the first 1 kbyte of memory from 00000H to 003FFH is reserved for storing the starting addresses of interrupt service routines. This block of memory is often called the **interrupt vector table** or the **interrupt pointer table.** Since 4 bytes are required to store the CS and IP values for each interrupt service procedure, the table can hold the starting addresses for 256 interrupt service routines. Fig. 6.2 shows how the 256 interrupt pointers are arranged in the memory table.

Each interrupt type is given a number between 0 to 255 and the address of each interrupt is found by multiplying the type by 4 e.g. for type 11, interrupt address is $11 \times 4 = 44_{10} =$ 0002CH.

Only first five types have explicit definitions such as divide by zero and non maskable interrupt. The next 27 interrupt types, from 5 to 31, are reserved by Intel for use in future microprocessors. The upper 224 interrupt types, from 32 to 255, are available for user for hardware or software interrupts.

When the 8086 responds to an interrupt, it automatically goes to the specified location in the interrupt vector table to get the starting address of interrupt service routine. So user has to load these starting addresses for different routines at the start of the program.

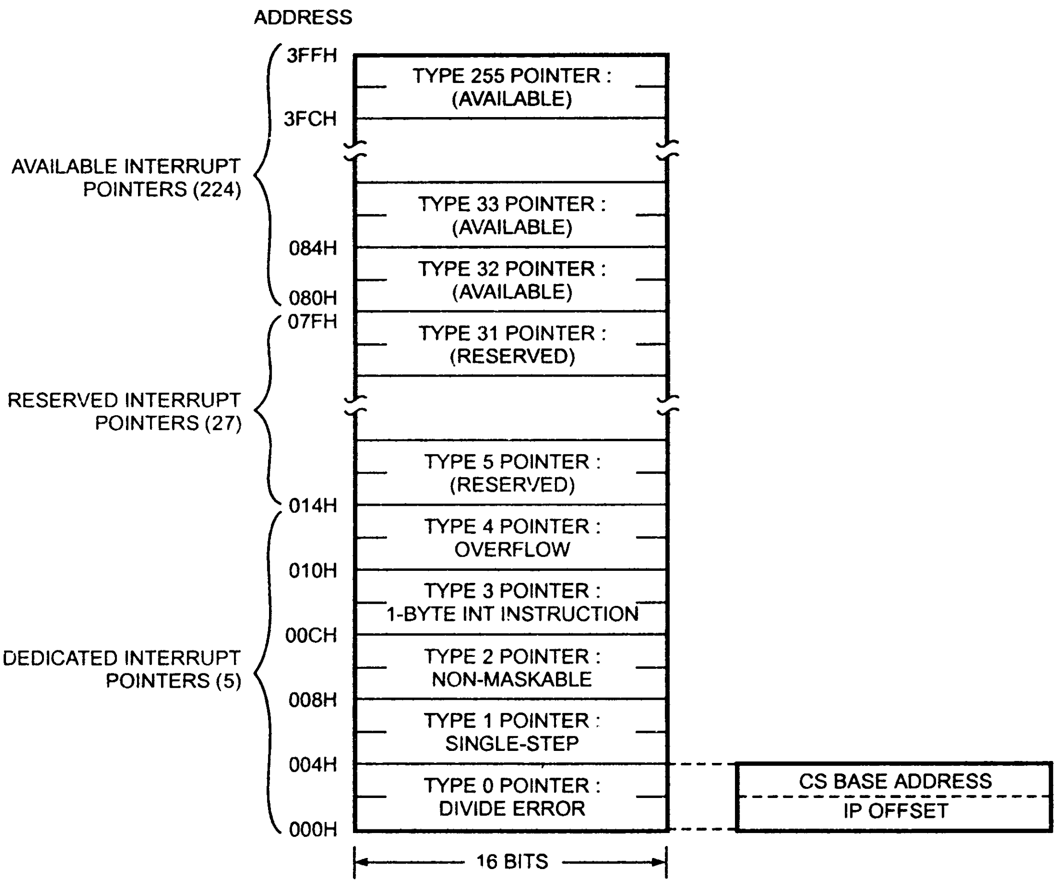


Fig. 6.2 8086 interrupt vector table

6.3 8086 Interrupt Types

6.3.1 Divide by Zero Interrupt (Type 0)

When the quotient from either a DIV or IDIV instruction is too large to fit in the result register; 8086 will automatically execute type 0 interrupt.

6.3.2 Single Step Interrupt (Type 1)

The type 1 interrupt is the single step trap. In the single step mode, system will execute one instruction and wait for further direction from user. Then user can examine the contents of registers and memory locations and if they are correct, user can tell the system to execute the next instruction. This feature is useful for debugging assembly language programs.

An 8086 system is used in the single step mode by setting the trap flag. If the trap flag is set, the 8086 will automatically execute a type 1 interrupt after execution of each instruction. But the 8086 has no such instruction to directly set or reset the trap flag. These operations can be performed by taking the flag register contents into memory, changing the memory contents so to set or reset trap flag and save the memory contents into flag register.

Assembly language program to set trap flag :

```
PUSHF                    ; save the contents of trap flag in
                         ; stack memory
MOV BP, SP               ; copy SP to BP for use as index
OR [ BP + 0 ], 0100H     ; set the Bit 8 in the memory pointed
                         ; by BP i.e. set TF bit
POPF                     ; Restore the flag register with TF = 1
```

To reset the trap flag we have to reset Bit 8. This can be done by using AND [BP + 0], 0FEFFH instruction instead of OR [BP + 0], 0100H.

6.3.3 Non Maskable Interrupt (Type 2)

As the name suggests, this interrupt cannot be disabled by any software instruction. This interrupt is activated by low to high transition on 8086 NMI input pin. In response, 8086 will do a type 2 interrupt.

6.3.4 Breakpoint Interrupt (Type 3)

The type 3 interrupt is used to implement **breakpoint** function in the system. The type 3 interrupt is produced by execution of the INT 3 instruction. Breakpoint function is often used as a debugging aid in cases where single stepping provides more detail than wanted. When you insert a breakpoint, the system executes the instructions upto the breakpoint, and then goes to the breakpoint procedure. In the breakpoint procedure you can write a program to display register contents, memory contents and other information that is required to debug your program. You can insert as many breakpoints as you want in your program.

6.3.5 Overflow Interrupt (Type 4)

The type 4 interrupt is used to check overflow condition after any signed arithmetic operation in the system. The 8086 overflow flag, OF, will be represented in the destination register or memory location.

For example, if you add the 8-bit signed number 0111 1000 (+ 120 decimal) and the 8-bit signed number 0110 1010 (+ 106 decimal), result is 1110 0010 (– 98 decimal). In signed numbers, MSB (Most Significant Bit) is reserved for sign and other bits represent magnitude of the number. In the previous example, after addition of two 8-bit signed numbers result is negative, since it is too large to fit in 7-bits. To detect this condition in the program, you can put interrupt on overflow instruction, INTO, immediately after the arithmetic instruction in the program. If the overflow flag is not set when the 8086 executes the INTO instruction, the instruction will simply function as an NOP (no operation). However, if the overflow flag is set, indicating an overflow error, the 8086 will execute a type 4 interrupt after executing the INTO instruction.

Another way to detect and respond to an overflow error in a program is to put the jump if overflow instruction, (JO) immediately after the arithmetic instruction. If the overflow flag is set as a result of arithmetic operation, execution will jump to the address specified in the JO instruction. At this address you can put an error routine which responds in the way you want to the overflow.

6.3.6 Software Interrupts

Type 0 - 255 :

The 8086 INT instruction can be used to cause the 8086 to do one of the 256 possible interrupt types. The interrupt type is specified by the number as a part of the instruction. You can use an INT2 instruction to send execution to an NMI interrupt service routine. This allows you to test the NMI routine without needing to apply an external signal to the NMI input of the 8086.

With the software interrupts you can call the desired routines from many different programs in a system e.g. BIOS in IBM PC. The IBM PC has in its ROM collection of routines, each performing some specific function such as reading character from keyboard, writing character to CRT. This collection of routines referred to as **Basic Input Output System** or **BIOS**.

The BIOS routines are called with INT instructions. We will summarize interrupt response and how it is serviced by going through following steps.

1. 8086 pushes the flag register on the stack.
2. It disables the single step and the INTR input by clearing the trap flag and interrupt flag in the flag register.
3. It saves the current CS and IP register contents by pushing them on the stack.
4. It does an indirect far jump to the start of the routine by loading the new values of CS and IP register from the memory whose address calculated by multiplying 4 to the interrupt type, For example, if interrupt type is 4 then memory address is $4 \times 4 = 10_{10}$ = 10H. So 8086 will read new value of IP from 00010H and CS from 00012H.
5. Once these values are loaded in the CS and IP, 8086 will fetch the instruction from the new address which is the starting address of interrupt service routine.
6. An IRET instruction at the end of the interrupt service routine gets the previous values of CS and IP by popping the CS and IP from the stack.
7. At the end the flag register contents are copied back into flag register by popping the flag register form stack.

6.3.7 Maskable Interrupt (INTR)

The 8086 INTR input can be used to interrupt a program execution. The 8086 is provided with a maskable handshake interrupt. This interrupt is implemented by using two pins - INTR and $\overline{INTA}$. This interrupt can be enabled or disabled by STI (IF=1) or CLI (IF=0), respectively. When the 8086 is reset, the interrupt flag is automatically cleared (IF=0). So after reset INTR is disabled. User has to execute STI instruction to enable INTR interrupt.

The 8086 responds to an INTR interrupt as follows :

1. The 8086 first does two interrupt acknowledge machine cycles as shown in the Fig. 6.3 to get the interrupt type from the external device. In the first interrupt acknowledge machine cycle the 8086 floats the data bus lines AD_0-AD_{15} and sends out an $\overline{INTA}$ pulse on its $\overline{INTA}$ output pin. This indicates an interrupt acknowledge cycle in progress and the system is ready to accept the interrupt type from the external device. During the second interrupt acknowledge machine cycle the 8086 sends out another pulse on its $\overline{INTA}$ output pin. In response to this second $\overline{INTA}$ pulse the external device puts the interrupt type on lower 8-bits of the data bus.

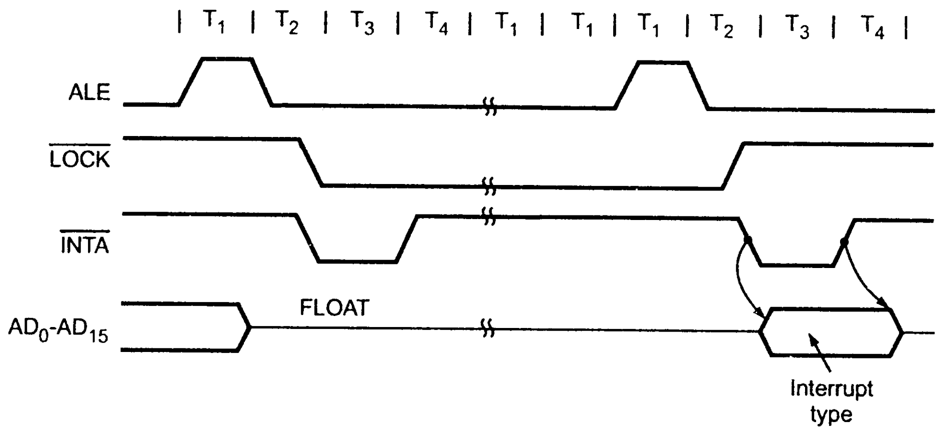


Fig. 6.3 Interrupt acknowledge machine cycle

2. Once the 8086 receives the interrupt type, it pushes the flag register on the stack, clears TF and IF, and pushes the CS and IP values of the next instruction on the stack.

3. The 8086 then gets the new value of IP from the memory address equal to 4 times the interrupt type (number), and CS value from memory address equal to 4 times the interrupt number plus 2.

6.4 Interrupt Priorities

As far as the 8086 interrupt priorities are concerned, software interrupts (All interrupts except single step, NMI and INTR interrupts) have the highest priority, followed by NMI followed by INTR. Single step has the least priority.

Interrupt	Priority
Divide Error, Int n, Int 0	HIGHEST
NMI	↓
INTR	↓
SINGLE - STEP	LOWEST

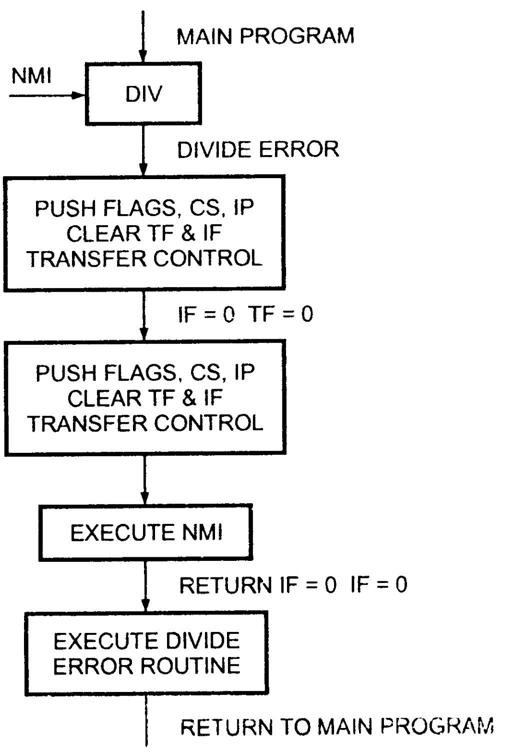


Fig. 6.4 Flowchart for divide error routine

The interrupt flag is automatically cleared as part of the response of an 8086 to an interrupt. This prevents a signal on the INTR input from interrupting a higher priority interrupt service routine. The 8086 allows NMI input to interrupt higher priority interrupt, for example suppose that a rising edge signal arrives at the NMI input while the 8086 is executing a DIV instruction, and that the division operation produces a divide error. Since the 8086 checks for internal interrupts before it checks for an NMI interrupt, the 8086 will push the flags on the stack, clear TF and IF, push the return address on the stack, and go to the start of the divide error service routine. The 8086 will then do an NMI interrupt response and execute non-maskable interrupt service routine. After completion of NMI service routine an 8086 will return to the divide error routine. it will execute divide error routine and then it will return to the main program (Refer Fig. 6.4).

6.5 Expanding Interrupt Structure using PIC 8259

Interrupts can be used for a variety of applications. Each of these interrupt applications requires a separate interrupt input. If we are working with an 8086, we get only two interrupt inputs INTR and NMI. For applications where we have multiple interrupt sources, we use external device called a priority interrupt controller (PIC). Fig. 6.5 shows the connection between 8086 and 8259.

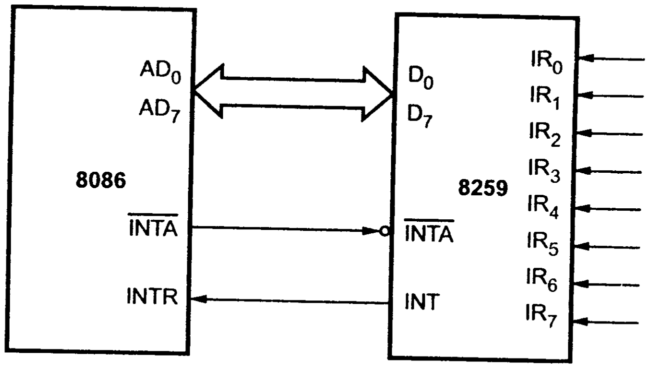


Fig. 6.5 Connection between 8086 and 8259

6.5.1 Features of 8259

1. It can manage eight priority interrupts. This is equivalent to provide eight interrupt pins on the processor in place of INTR pin.
2. It is possible to locate vector table for these additional interrupts any where in the memory map. However, all eight interrupts are spaced at the interval of either four or eight locations.
3. By cascading 8259s it is possible to get 64 priority interrupts.
4. Interrupt mask register makes it possible to mask individual interrupt request.
5. The 8259A can be programmed to accept either the level triggered or the edge triggered interrupt request.
6. With the help of 8259A user can get the information of pending interrupts, in-service interrupts and masked interrupts.
7. The 8259A is designed to minimize the software and real time overhead in handling multilevel priority interrupts.

6.5.2 Block Diagram of 8259A

Fig. 6.6 shows the internal block diagram of the 8259A. It includes eight blocks : data bus buffer, read/write logic, control logic, three registers (IRR, ISR and IMR), priority resolver, and cascade buffer.

Data Bus Buffer

The data bus allows the 8086 to send control words to the 8259A and read a status word from the 8259A and read a status word from the 8259A. The 8-bit data bus also allows the 8259A to send interrupt types to the 8086.

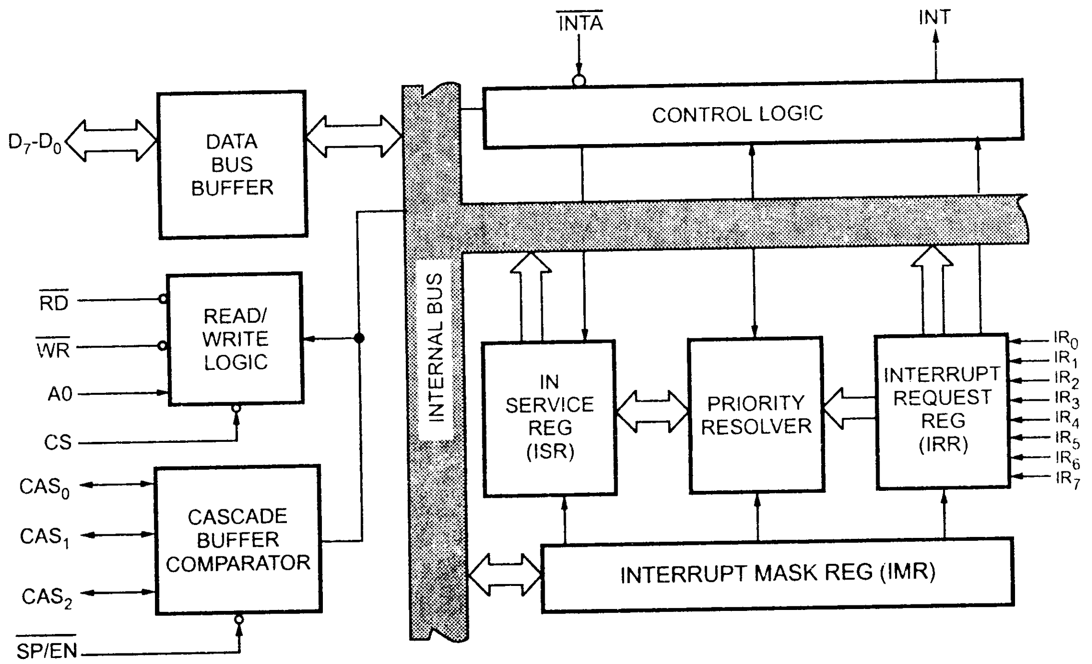


Fig. 6.6 Block diagram of 8259A

Read/Write Logic

The $\overline{RD}$ and $\overline{WR}$ inputs control the data flow on the data bus when the device is selected by asserting its chip select ($\overline{CS}$) input low.

Control Logic

This block has an input and an output line. If the 8259A is properly enabled the interrupt request will cause the 8259A to assert its INT output pin high. If this pin is connected to the INTR pin of an 8086 and if the 8086 interrupt flag is set, then this high signal will cause the 8086 to respond INTR as explained earlier.

Interrupt Request Register (IRR)

The IRR is used to store all the interrupt levels which are requesting service. The eight interrupt inputs set corresponding bits of the Interrupt Request Register.

Interrupt Service Register (ISR)

The Interrupt Service Register (ISR) stores all the levels that are currently being serviced.

Interrupt Mask Register (IMR)

Interrupt Mask Register (IMR) stores the masking bits of the interrupt lines to be masked. This register can be programmed by an OCW. An interrupt which is masked by software will not be recognised and serviced even if it set the corresponding bits in the IRR.

Priority Resolver

The priority resolver determines the priorities of the bits set in the IRR. The bit corresponding to the highest priority interrupt input is set in the ISR during the $\overline{INTA}$ input.

Cascade Buffer Comparator

This section generates control signals necessary for cascade operations. It also generates Buffer-Enable signals. As stated earlier, the 8259 can be cascaded with other 8259s in order to expand the interrupt handling capacity to sixty-four levels. In such a case, the former is called a **master**, and the latter are called **slaves**. The 8259 can be set up as a master or a slave by the $\overline{SP}$ / $\overline{EN}$ pin.

CAS 0 - 2

For a master 8259, the CAS_0-CAS_2 pins are outputs, and for slave 8259s, these are inputs. When the 8259 is a master (that is, when it accepts interrupt requests from other 8259s), the CALL opcode is generated by the Master in response to the first $\overline{INTA}$. The vectoring address must be released by the slave 8259. The master sends an identification code of three-bits (to select one out of the eight possible slave 8259s) on the CAS_0-CAS_2 lines. The slave 8259s accept these three signals as inputs (on their CAS_0-CAS_2 pins) and compare the code sent by the master with the codes assigned to them during initialisation. The slave thus selected (which had originally placed an interrupt request to the master 8259) then puts out the address of the interrupt service routine during the second and third $\overline{INTA}$ pulses from the CPU.

$\overline{SP}$ / $\overline{EN}$ (Slave Program /Enable Buffer)

The $\overline{SP}$ / $\overline{EN}$ signal is tied high for the master. However, it is grounded for the slave.

In large systems where buffers are used to drive the data bus, the data sent by the 8259 in response to $\overline{INTA}$ cannot be accessed by the CPU (due to the data bus buffer being disabled).

If an 8259 is used in the buffered mode (buffered or non-buffered modes of operation can be specified at the time of initialising the 8259), the $\overline{SP}$ / $\overline{EN}$ pin is used as an output which can be used to enable the system data bus buffer whenever the 8259's data bus outputs are enabled (when it is ready to send data).

Means, in non-buffered mode, the $\overline{SP}/\overline{EN}$ pin of an 8259 is used to specify whether the 8259 is to operate as a master or as a slave, and in the buffered mode, the $\overline{SP}/\overline{EN}$ pin is used as an output to enable the data bus buffer of the system.

6.5.3 Interrupt Sequence

The events occur as follows in an 8086 system :

1. One or more of the INTERRUPT REQUEST lines (IR0-IR7) are raised high, setting the corresponding IRR bit(s).

2. The priority resolver checks three registers : The IRR for interrupt requests, the IMR for masking bits, and the ISR for the interrupt request being served. It resolves the priority and sets the INT high when appropriate.
3. The CPU acknowledges the INT and responds with an $\overline{\text{INTA}}$ pulse.
4. Upon receiving an $\overline{\text{INTA}}$ from the CPU, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive data bus during this cycle.
5. A selection of priority modes is available to the programmer so that the manner in which the requests are processed by the 8259A can be configured to match his system requirements. The priority modes can be changed or reconfigured dynamically at any time during the main program. This means that the complete interrupt service structure can be defined as required, based on the total system environment.
6. The 8086 will initiate a second INTA pulse. During this pulse, the 8259A releases a 8-bit pointer (interrupt type) onto the Data Bus where it is read by the CPU.
7. This completes the interrupt cycle. In the AEOI mode the ISR bit is reset at the end of the second INTA pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt subroutine.

6.5.4 Priority Modes and Other Features

The various modes of operation of the 8259 are :

(a) Fully Nested Mode,

(b) Rotating Priority Mode,

(c) Special Masked Mode, and

(d) Polled Mode.

a) Fully Nested Mode :

After initialization, the 8259A operates in fully nested mode so it is called as default mode. The 8259 continues to operate in the Fully Nested Mode until the mode is changed through Operation Command Words. In this mode, IR0 has highest priority and IR7 has lowest priority. When the interrupt is acknowledged, it sets the corresponding bit in ISR. This bit will prevent all interrupts of the same or lower level, however it will accept higher priority interrupt requests. The vector address corresponding to this interrupt is then sent. The bit in the ISR will remain set until an EOI command is issued by the microprocessor at the end of interrupt service routine.

But if AEOI (Automatic End of Interrupt) bit is set, the bit in the ISR resets at the trailing edge of the last $\overline{\text{INTA}}$.

End of Interrupt (EOI)

The IS bit can be reset by an End of Interrupt command issued by the CPU, usually just before exiting from the interrupt routine.