**Savitribai Phule Pune University**
**Second Year of Computer Engineering (2015 Course)**
**210257: Microprocessor Lab**
**Assignment No. 1**

**Problem Statement: Write X86/64 ALP to count number of positive and negative numbers from the array.**

```
;*************************************************************
section .data

nline db 10,10
nline_len equ  $-nline

arr dd  -11111111H, 22222222H, -33333333H, -44444444H, -55555555H
arr_size equ 5

pmsg db   10,10,"The no. of Positive elements in 32-bit array:"
pmsg_len equ $-pmsg

nmsg  db  10,10,"The no. of Negative elements in 32-bit array:"
nmsg_len equ $-nmsg

;*************************************************************
section  .bss

p_count  resq   01
n_count  resq   01
dnumbuff resb   02


%macro display 2
    mov rax,01
    mov rdi,01
    mov rsi,%1
    mov rdx,%2
    syscall
%endmacro

;*************************************************************

section .text
global   _start
_start:

    mov    esi, arr
    mov    ecx,5         ;Arraay counter i.e.5
```

```
    mov    ebx,0;         ; counter for     +ve nos.
    mov    edx,0;         ; counter for    -ve nos.

next_num:
    mov    eax,[esi]     ; take no. in RAX
    rcl    eax,1         ; rotate left 1 bit to check for sign bit
    jc    negative

positive:
    inc    ebx            ; no carry, so no. is +ve
    jmp    next

negative:
    inc    edx            ; carry, so no. is -ve

next:
    add  esi,4                ; 32 bit nos i.e. 4 bytes
    loop next_num

    mov [p_count], ebx        ; store positive count
    mov [n_count], edx        ; store negative count

    display pmsg, pmsg_len
    mov ebx,[p_count]         ; load value of p_count in rax
    call disp8_proc           ; display p_count

    display nmsg, nmsg_len
    mov ebx,[n_count]        ; load value of n_count in rax
    call  disp8_proc         ; display n_count

    display    nline, nline_len

  exit:
    mov rax,60        ;Exit
    mov rbx,00
    syscall


;******************************************************
disp8_proc:
    mov edi,dnumbuff    ;point edi to buffer
    mov ecx,02        ;load number of digits to display

dispup1:
    rol bl,4        ;rotate number left by four bits
    mov dl,bl       ;move lower byte in dl
    and dl,0fh      ;mask upper digit of byte in dl
```

```
    add dl,30h      ;add 30h to calculate ASCII code
    cmp dl,39h      ;compare with 39h
    jbe dispskip1   ;if less than 39h akip adding 07 more
    add dl,07h      ;else add 07

dispskip1:
    mov [edi],dl     ;store ASCII code in buffer
    inc edi          ;point to next byte
    loop dispup1     ;decrement the count of digits to display
                     ;if not zero jump to repeat

    display dnumbuff,2
    ret

;****************Output*************************************
;[root@localhost A1]# nasm -f elf64 A1.asm
;[root@localhost A1]# ld -o A1 A1.o
;[root@localhost A1]# ./A1


;The no. of Positive elements in 32-bit array :    01

;The no. of Negative elements in 32-bit array :    04

;[root@localhost A1]#
```

**Assignment No.2A**

**Problem Statement: Write X86/64 ALP to perform non-overlapped and overlapped block transfer (with and without string specific instructions). Block containing data can be defined in the data segment.**

```
;*************************************************************
;Assignmnet No. 2A
;Write X86/64 ALP to perform non-overlapped and overlapped block ;transfer
(with and without string specific instructions). Block;containing data can
be defined in the data segment.
;.............................................................
; Non-overlapped Block Transfer
;*************************************************************
section .data
 menumsg db 10,'##Menu for Non-overlapped Block Transfer##',10
       db 10,'1.Block Transfer without using string instructions'
       db 10,'2.Block Transfer with using string instructions'
       db 10,'3.Exit',10
 menumsg_len equ $-menumsg

 blk_bfrmsg db 10,'Block contents before transfer'
 blk_bfrmsg_len equ $-blk_bfrmsg

 blk_afrmsg db 10,'Block contents after transfer'
 blk_afrmsg_len equ $-blk_afrmsg

 srcmsg db 10,'Source block contents::'
 srcmsg_len equ $-srcmsg

 dstmsg db 10,'Destination block contents::'
 dstmsg_len equ $-dstmsg

 srcblk db 01h,02h,03h,04h,05h
 dstblk db 00,00,00,00,00

 spacechar db 20h
 spchlength equ $-spacechar

;***************.bss Section*****************************
section .bss
    optionbuff resb 02
    dispbuff resb 02

%macro display 2
    mov rax,01
    mov rdi,01
```

```
    mov rsi,%1
    mov rdx,%2
    syscall
%endmacro

%macro accept 2
    mov rax,00
    mov rdi,00
    mov rsi,%1
    mov rdx,%2
    syscall
%endmacro

;******************.text Section*************************
section .text
    global _start
_start:

    display blk_bfrmsg,blk_bfrmsg_len

    call dispsrc_blk_proc

    call dispdest_blk_proc

   menu: display menumsg,menumsg_len

         accept optionbuff,02

         cmp byte [optionbuff],31h

         je wos

         cmp byte [optionbuff],32h

         je ws


exit:    mov rax,60        ;Exit
         mov rbx,00
         syscall

;**************Display Source Block Procedure***************

dispsrc_blk_proc:
        display srcmsg,srcmsg_len
        mov rsi,srcblk
        mov rcx,05h
```

```
    up1:push rcx
        mov bl,[rsi]
        push rsi

        call disp8_proc

        display spacechar,spchlength

pop rsi
        inc rsi
        pop rcx
        loop up1
        ret

;**************Display Destination Block Procedure*************

dispdest_blk_proc:
        display dstmsg,dstmsg_len
        mov rdi,dstblk
        mov rcx,05

    up2:push rcx
        mov bl,[rdi]
        push rdi

        call disp8_proc

        display spacechar,spchlength

pop rdi
        inc rdi
        pop rcx
        loop up2
        ret

;**************Without String Procedure************
wos:
    mov rsi,srcblk
    mov rdi,dstblk
    mov rcx,05

         again: mov bl,[rsi]
                mov [rdi],bl
                inc rsi
                inc rdi
                loop again
```

```
        display blk_afrmsg,blk_afrmsg_len
        call dispsrc_blk_proc
        call dispdest_blk_proc
        jmp menu

;************Using String Procedure***************

ws:
    mov rsi,srcblk
    mov rdi,dstblk
    mov rcx,05

     cld                                                                rep
movsb

    display blk_afrmsg,blk_afrmsg_len
    call dispsrc_blk_proc
    call dispdest_blk_proc
    jmp menu

;************Display Procedure*******************

disp8_proc:
    mov rsi,dispbuff
    mov rcx,02

dup1:
    rol bl,4
    mov dl,bl
    and dl,0Fh
    cmp dl,09H
    jbe dskip
    add dl,07h

dskip:add dl,30h
      mov [rsi],dl
      inc rsi
      loop dup1

      display dispbuff,02

      ret
;************Output*********************************
;[root@localhost A2]# nasm -f elf64 Ass2A.asm
;[root@localhost A2]# ld -o Ass2A Ass2A.o
;[root@localhost A2]# ./Ass2A
```

```
;Block contents before transfer
;Source block contents::01 02 03 04 05
;Destination block contents::00 00 00 00 00

;##### Menu for Non-overlapped Block Transfer #####

;1.Block Transfer without using string instructions
;2.Block Transfer with using string instructions
;3.Exit
;1

;Block contents after transfer
;Source block contents::01 02 03 04 05
;Destination block contents::01 02 03 04 05

;##### Menu for Non-overlapped Block Transfer #####

;1.Block Transfer without using string instructions
;2.Block Transfer with using string instructions
;3.Exit
;3
;[root@localhost A2]#
```

**Assignment No.2B**

**Problem Statement: Write X86/64 ALP to perform non-overlapped and overlapped block transfer (with and without string specific instructions). Block containing data can be defined in the data segment.**

```
;*********************************************************
;Assignmnet No. 2B
;Write X86/64 ALP to perform non-overlapped and overlapped block ;transfer
(with and without string specific instructions). Block ;containing data
can be defined in the data segment.
;.........................................................
; overlapped Block Transfer
;*********************************************************

section .data
    menumsg db 10,'##Menu for overlapped Block Transfer##',10
        db 10,'1.Block Overlap without using string instructions'
        db 10,'2.Block Overlap with using string instructions'
        db 10,'3.Exit',10,10
    menumsg_len equ $-menumsg


    blk_bfrmsg db 10,10,'Block contents before Overlap'
    blk_bfrmsg_len equ $-blk_bfrmsg

    blk_afrmsg db 10,'Block contents after Overlap',10
    blk_afrmsg_len equ $-blk_afrmsg

    srcmsg db 10,'Source block contents::'
    srcmsg_len equ $-srcmsg

    posmsg db 10,10,10,'Enter position to overlap::'
    posmsg_len equ $-posmsg

    spacechar db 20h
    spchlength equ $-spacechar

    srcblk db 01h,02h,03h,04h,05h,00h,00h,00h,00h,00h

;**********.bss Section*********************

section .bss
    optionbuff resb 02
    dispbuff resb 02
    numascii resb 03
    pos resb 00
```

```
%macro display 2
    mov rax,01
    mov rdi,01
    mov rsi,%1
    mov rdx,%2
    syscall
%endmacro

%macro accept 2
    mov rax,00
    mov rdi,00
    mov rsi,%1
    mov rdx,%2
    syscall
%endmacro

;**********Text Section*********************

section .text
    global _start
_start:

    display blk_bfrmsg,blk_bfrmsg_len

    call disp_src_blk_proc


    display posmsg,posmsg_len

    accept numascii,3

    call packnum_proc


menu:    display menumsg,menumsg_len

         accept optionbuff,02

         cmp byte [optionbuff],31H
         je wos

         cmp byte [optionbuff],32H
         je ws
exit:
    mov rax,60        ;Exit
    mov rbx,00
    syscall
```

```
;**********Display Block Procedure**********************

disp_src_blk_proc:
        display srcmsg,srcmsg_len
        mov rsi,srcblk
        mov rcx,05h
        add cl,[pos]

    up1:push rcx
        mov bl,[rsi]
        push rsi

        call disp8_proc

        display spacechar,spchlength
        pop rsi
        inc rsi
        pop rcx
        loop up1
        ret

;************Without String Procedure***************
wos:
    mov rsi,srcblk+4
    mov rdi,rsi
    add rdi,[pos]

    mov rcx,05
blkup1:
    mov al,[rsi]
    mov [rdi],al
    dec rsi
    dec rdi
    loop blkup1

    display blk_afrmsg,blk_afrmsg_len
    call disp_src_blk_proc

    jmp exit

;************Using String Procedure***************
ws:
    mov esi,srcblk+4
    mov edi,esi
    add edi,[pos]
```

```
    mov ecx,05

    std
    rep movsb

    display blk_afrmsg,blk_afrmsg_len
    call disp_src_blk_proc

    jmp exit

;************Display Procedure*******************
disp8_proc:
    mov ecx,2
    mov edi,dispbuff
 dup1:
       rol bl,4
       mov al,bl
       and al,0fh
       cmp al,09
       jbe dskip
       add al,07h
dskip: add al,30h
       mov [edi],al
       inc edi
       loop dup1

    display dispbuff,2
    ret

;************Packnum Procedure*******************
packnum_proc:
    mov bx,0
    mov ecx,2
    mov esi,numascii

up2:    rol bl,4
    mov al,[esi]
    sub al,30h
    cmp al,09h
    jbe skip5
    sub al,07h
skip5:
    add bl,al
    inc esi
    loop up2
    mov [pos],bl
    ret
```

```
;*************Output*********************************
;[root@localhost MIT2016]# nasm -f elf64 Ass2B.asm
;[root@localhost MIT2016]# ld -o Ass2B Ass2B.o
;[root@localhost MIT2016]# ./Ass2B


;Block contents before Overlap
;Source block contents::01 02 03 04 05


;Enter position to overlap::02

;## Menu for overlapped Block Transfer ##

;1.Block Overlap without using string instructions
;2.Block Overlap with using string instructions
;3.Exit

;1

;Block contents after Overlap

;Source block contents::01 02 01 02 03 04 05

[root@localhost MIT2016]#
```

**Assignment No.03**

**Problem Statement:** Write X86/64 ALP to convert 4-digit Hex number into its equivalent BCD number and 5-digit BCD number into its equivalent HEX number. Make your program user friendly to accept the choice from user for:
(a) HEX to BCD b) BCD to HEX (c) EXIT.
Display proper strings to prompt the user while accepting the input and displaying the result.(wherever necessary, use 64-bit registers)

```
;*************************************************************
section .data
    menumsg db 10,10,'###### Menu for Code Conversion ######'
        db 10,'1: Hex to BCD'
        db 10,'2: BCD to Hex'
        db 10,'3: Exit'
        db 10,10,'Please Enter Choice::'
    menumsg_len equ $-menumsg


    hexinmsg db 10,10,'Please enter 4 digit hex number::'
    hexinmsg_len equ $-hexinmsg

    bcdopmsg db 10,10,'BCD Equivalent::'
    bcdopmsg_len equ $-bcdopmsg

    bcdinmsg db 10,10,'Please enter 5 digit BCD number::'
    bcdinmsg_len equ $-bcdinmsg

    hexopmsg db 10,10,'Hex Equivalent::'
    hexopmsg_len equ $-hexopmsg

;*************************************************************

section .bss

    numascii resb 06  ;common buffer for choice, hex and bcd input
    outputbuff resb 02
    dispbuff resb 08

    %macro display 2
       mov rax,01
       mov rdi,01
       mov rsi,%1
       mov rdx,%2
       syscall
    %endmacro
```

```
       %macro accept 2
      mov rax,0
      mov rdi,0
      mov rsi,%1
      mov rdx,%2
      syscall
       %endmacro
;*************************************************************
section .text

    global _start
_start:

menu:    display menumsg,menumsg_len
    accept numascii,2

    cmp byte [numascii],'1'
    je hex2bcd_proc



    cmp byte [numascii],'2'
    je bcd2hex_proc


    cmp byte [numascii],'3'
    je exit
    jmp _start

exit:
    mov rax,60
    mov rbx,0
    syscall
;*************************************************************
hex2bcd_proc:
    display hexinmsg,hexinmsg_len
    accept numascii,5
    call packnum
    mov ax,bx
    mov rcx,0
    mov bx,10        ;Base of Decimal No. system
h2bup1:    mov dx,0
    div bx
    push rdx
    inc rcx
    cmp ax,0
    jne h2bup1
```

```
    mov rdi,outputbuff

h2bup2:    pop rdx
    add dl,30h
    mov [rdi],dl
    inc rdi
    loop h2bup2

    display bcdopmsg,bcdopmsg_len
    display outputbuff,5
    jmp menu
;*************************************************************
bcd2hex_proc:
    display bcdinmsg,bcdinmsg_len
    accept numascii,6

    display hexopmsg,hexopmsg_len

    mov rsi,numascii
    mov rcx,05
    mov rax,0
    mov ebx,0ah

b2hup1:    mov rdx,0
    mul ebx
    mov dl,[rsi]
    sub dl,30h
    add rax,rdx
    inc rsi
    loop b2hup1
    mov ebx,eax
    call disp32_num
    jmp menu
;*************************************************************
packnum:
    mov bx,0
    mov ecx,04
    mov esi,numascii
up1:
    rol bx,04
    mov al,[esi]
    cmp al,39h
    jbe skip1
    sub al,07h
skip1:    sub al,30h
    add bl,al
    inc esi
```

```
    loop up1
    ret

;*************************************************************
disp32_num:
    mov rdi,dispbuff    ;point esi to buffer
    mov rcx,08         ;load number of digits to display

dispup1:
    rol ebx,4         ;rotate number left by four bits
    mov dl,bl         ;move lower byte in dl
    and dl,0fh         ;mask upper digit of byte in dl
    add dl,30h         ;add 30h to calculate ASCII code
    cmp dl,39h         ;compare with 39h
    jbe dispskip1         ;if less than 39h akip adding 07 more
    add dl,07h         ;else add 07

dispskip1:
    mov [rdi],dl         ;store ASCII code in buffer
    inc rdi             ;point to next byte
    loop dispup1         ;decrement the count of digits to display
                ;if not zero jump to repeat

    display dispbuff+3,5    ;Dispays only lower 5 digits as upper three
are '0'

    ret
;******************OUTPUT****************************************
;###### Menu for Code Conversion ######
;1: Hex to BCD
;2: BCD to Hex
;3: Exit

;Please Enter Choice::1
;Please enter 4 digit hex number::000F

;BCD Equivalent::15
;###### Menu for Code Conversion ######
;1: Hex to BCD
;2: BCD to Hex
;3: Exit
;Please Enter Choice::2
;Please enter 5 digit BCD number::00015


;Hex Equivalent::0000F
```

**Assignment No.04**

**Problem Statement: Write X86/64 ALP to perform multiplication of two 8-bit hexadecimal numbers. Use successive addition and add and shift method. (use of 64-bit registers is expected)**

```
;*****************************************
;Assignment no:4A
;Title:Multiplication using successive addition
;*****************************************
section .data

welmsg db 10,'Multiplication using successive addition',10
welmsg_len equ $-welmsg

nummsg db 10,'Enter two digits of Number::'
nummsg_len equ $-nummsg

resmsg db 10,'Multiplication of elements::'
resmsg_len equ $-resmsg

blankmsg db 10,'',10
blank_len equ $-blankmsg

;**********.bss Section**********************

section .bss

    numascii resb 03
    num1 resb 02
    num2 resb 02
    result resb 01
    dispbuff resb 04

%macro display 2
    mov rax,01
    mov rdi,01
    mov rsi,%1
    mov rdx,%2
    syscall
%endmacro

%macro accept 2
    mov rax,00
    mov rdi,00
    mov rsi,%1
    mov rdx,%2
```

```
    syscall
%endmacro

;**********.text Section**********************
section .text
global _start
_start:

    display welmsg,welmsg_len

    display nummsg,nummsg_len
    accept numascii,3
    call packnum
    mov byte[num1],bl

    display nummsg,nummsg_len
    accept numascii,3
    call packnum
    mov byte[num2],bl


    mov cx,[num2]
         mov edx,00h            ;Temporary Addition
    mov eax,[num1]


addup:    add edx,eax
    loop addup

    mov [result],edx

    display resmsg,resmsg_len
    mov ebx,[result]

    call disp16_proc

    display blankmsg,blank_len

exit:    mov rax,60
    mov rbx,00
    syscall

;**********Packnum Procedure**********************
packnum:
    mov bl,0
    mov ecx,02
    mov esi,numascii
```

```
    up1:rol bl,04
    mov al,[esi]
    cmp al,39h
    jbe skip1
    sub al,07h
 skip1: sub al,30h
    add bl,al
    inc esi
    loop up1
    ret
;**********Display Procedure*********************
disp16_proc:
    mov ecx,4
    mov edi,dispbuff
 dup1:  rol bx,4
    mov al,bl
    and al,0fh
    cmp al,09
    jbe dskip
    add al,07h
 dskip: add al,30h
        mov [edi],al
        inc edi
        loop dup1
        display dispbuff,4
        ret
;**********Output**********************
;[root@localhost MIT2016]# nasm -f elf64 Ass3A.asm
;[root@localhost MIT2016]# ld -o Ass3A Ass3A.o
;[root@localhost MIT2016]# ./Ass3A

;Multiplication using successive addition

;Enter two digits of Number::04

;Enter two digits of Number::05

;Multiplication of elements::0014

;[root@localhost MIT2016]#

;*****************************************
;Assignment no:4B
;.....................................
;Title:Multiplication using Add & Shift method
;*****************************************
```

```
section .data

welmsg db 10,'Multiplication using Add & Shift method',10
welmsg_len equ $-welmsg

nummsg db 10,'Enter two digits of Number::'
nummsg_len equ $-nummsg

resmsg db 10,'Multiplication of elements::'
resmsg_len equ $-resmsg

blankmsg db 10,'',10
blank_len equ $-blankmsg

;**********.bss Section**********************

section .bss

    numascii resb 03
    num1 resb 02
    num2 resb 02
    result resb 02
    dispbuff resb 04

%macro display 2
    mov rax,01
    mov rdi,01
    mov rsi,%1
    mov rdx,%2
    syscall
%endmacro

%macro accept 2
    mov rax,00
    mov rdi,00
    mov rsi,%1
    mov rdx,%2
    syscall
%endmacro

;**********.text Section**********************
section .text
global _start
_start:

    display welmsg,welmsg_len
```

```
    display nummsg,nummsg_len
    accept numascii,3
    call packnum
    mov byte[num1],bl

    display nummsg,nummsg_len
    accept numascii,3
    call packnum
    mov byte[num2],bl

    mov al,[num1]

    mov cl,0
    mov edx,0
    mov edx,08h

    addup:
        rcr al,01
        jnc next1
        mov bh,00h
        shl bx,cl
        add [result],bx
        mov bl,[num2]
    next1:     inc cl
               dec edx
        jnz addup

    display resmsg,resmsg_len
    mov ebx,[result]
    call disp16_proc

display blankmsg,blank_len

exit:    mov rax,60
    mov rbx,00
    syscall

;**********Packnum Procedure**********************
packnum:
    mov bl,0
    mov ecx,02
    mov esi,numascii
    up1:rol bl,04
    mov al,[esi]
    cmp al,39h
    jbe skip1
    sub al,07h
```

```
 skip1: sub al,30h
    add bl,al
    inc esi
    loop up1
    ret
;**********Display Procedure**********************
disp16_proc:
    mov ecx,4
    mov edi,dispbuff
 dup1:  rol bx,4
    mov al,bl
    and al,0fh
    cmp al,09
    jbe dskip
    add al,07h
 dskip: add al,30h
        mov [edi],al
        inc edi
        loop dup1
        display dispbuff,4
        ret
;**********Output**********************
;[root@localhost MIT2016]# nasm -f elf64 Ass3B.asm
;[root@localhost MIT2016]# ld -o Ass3B Ass3B.o
;[root@localhost MIT2016]# ./Ass3B

;Multiplication using Add & Shift method

;Enter two digits of Number::04

;Enter two digits of Number::05

;Multiplication of elements::0014

;[root@localhost MIT2016]#
```

**Assignment No.05**

**Problem Statement:Write X86 ALP to find, a) Number of Blank spaces b) Number of lines c) Occurrence of a particular character. Accept the data from the text file. The text file has to be accessed during Program_1 execution and write FAR PROCEDURES in Program_2 for the rest of the processing. Use of PUBLIC and EXTERN directives is mandatory.**

# A5_file1.asm

```
;************************************************************

extern    far_proc            ; [ FAR PROCRDURE
                    ;   USING EXTERN DIRECTIVE ]

global    filehandle, char, buf, abuf_len

%include    "macro.asm"

;************************************************************
section .data
    nline        db    10
    nline_len    equ    $-nline

    ano        db     10,10,10,10,"ML assignment 05 :- String Operation
using Far Procedure"
           db           10,"-----------------------------------------
------",10
    ano_len    equ    $-ano

    filemsg    db    10,"Enter filename for string operation    : "
    filemsg_len    equ    $-filemsg

    charmsg    db    10,"Enter character to search    : "
    charmsg_len    equ    $-charmsg

    errmsg    db    10,"ERROR in opening File...",10
    errmsg_len    equ    $-errmsg

    exitmsg    db    10,10,"Exit from program...",10,10
    exitmsg_len    equ    $-exitmsg

;************************************************************
section .bss
    buf            resb    4096
```

```
    buf_len         equ    $-buf         ; buffer initial length

    filename         resb    50
    char             resb    2

    filehandle         resq    1
    abuf_len         resq    1         ; actual buffer length


;*************************************************************
section .text
    global _start

_start:
        display    ano,ano_len         ;assignment no.

        display    filemsg,filemsg_len
        accept     filename,50
        dec    rax
        mov    byte[filename + rax],0         ; blank char/null char

        display    charmsg,charmsg_len
        accept     char,2

        fopen    filename             ; on succes returns handle
        cmp    rax,-1H             ; on failure returns -1
        jle    Error
        mov    [filehandle],rax

        fread    [filehandle],buf, buf_len
        mov    [abuf_len],rax

        call    far_proc
        jmp    Exit

Error:    display    errmsg, errmsg_len

Exit:    display    exitmsg,exitmsg_len

    display nline,nline_len

    mov rax,60
    mov rdi,0
    syscall
```

# A5_file2.asm

```
;*************************************************************
global    far_proc

extern    filehandle, char, buf, abuf_len

%include "macro.asm"
;*************************************************************
section .data
    nline         db    10,10
    nline_len:     equ    $-nline

    smsg         db    10,"No. of spaces are    : "
    smsg_len:     equ    $-smsg

    nmsg         db    10,"No. of lines are    : "
    nmsg_len:     equ    $-nmsg

    cmsg         db    10,"No. of character occurances are    : "
    cmsg_len:     equ    $-cmsg


;*************************************************************
section .bss

    scount    resq    1
    ncount    resq    1
    ccount    resq    1

    dispbuff    resb    4

;*************************************************************
section .text
;    global    _main
;_main:

far_proc:                    ;FAR Procedure

        mov    rax,0
        mov    rbx,0
        mov    rcx,0
        mov    rsi,0
```

```
        mov    bl,[char]
        mov    rsi,buf
        mov    rcx,[abuf_len]

again:    mov    al,[rsi]

case_s:    cmp    al,20h          ;space : 32 (20H)
        jne    case_n
        inc    qword[scount]
        jmp    next

case_n:    cmp    al,0Ah          ;newline : 10(0AH)
        jne    case_c
        inc    qword[ncount]
        jmp    next

case_c:    cmp    al,bl              ;character
        jne    next
        inc    qword[ccount]

next:       inc    rsi
        dec    rcx             ;
        jnz    again             ;loop again

        display smsg,smsg_len
        mov    rbx,[scount]
        call    display16_proc

        display nmsg,nmsg_len
        mov    rbx,[ncount]
        call    display16_proc

        display cmsg,cmsg_len
        mov    rbx,[ccount]
        call     display16_proc

    fclose    [filehandle]
    ret


;*************************************************************
display16_proc:
    mov rdi,dispbuff    ;point esi to buffer
    mov rcx,4         ;load number of digits to display
dispup1:
```

```
    rol bx,4         ;rotate number left by four bits
    mov dl,bl         ;move lower byte in dl
    and dl,0fh         ;mask upper digit of byte in dl
    add dl,30h         ;add 30h to calculate ASCII code
    cmp dl,39h         ;compare with 39h
    jbe dispskip1         ;if less than 39h akip adding 07 more
    add dl,07h         ;else add 07

dispskip1:
    mov [rdi],dl         ;store ASCII code in buffer
    inc rdi             ;point to next byte
    loop dispup1         ;decrement the count of digits to display
                ;if not zero jump to repeat

    display dispbuff,4    ;

    ret
;*************************************************************
```

## macro.asm

```
;*************************************************************
;macro.asm
;macros as per 64 bit conventions

%macro accept 2
    mov    rax,0    ;read
    mov    rdi,0    ;stdin/keyboard
    mov    rsi,%1    ;buf
    mov    rdx,%2    ;buf_len
    syscall
%endmacro

%macro display 2
    mov    rax,1    ;print
    mov    rdi,1    ;stdout/screen
    mov    rsi,%1    ;msg
    mov    rdx,%2    ;msg_len
    syscall
%endmacro
```

```
%macro fopen 1
    mov     rax,2         ;open
    mov     rdi,%1         ;filename
    mov     rsi,2         ;mode RW
    mov     rdx,0777o     ;File permissions
    syscall
%endmacro

%macro fread 3
    mov     rax,0     ;read
    mov     rdi,%1     ;filehandle
    mov     rsi,%2     ;buf
    mov     rdx,%3     ;buf_len
    syscall
%endmacro

%macro fwrite 3
    mov     rax,1     ;write/print
    mov     rdi,%1     ;filehandle
    mov     rsi,%2     ;buf
    mov     rdx,%3     ;buf_len
    syscall
%endmacro

%macro fclose 1
    mov     rax,3         ;close
    mov     rdi,%1     ;file handle
    syscall
%endmacro
;*************************************************************
```

## myfile.txt

```
"Welcome!!!"
Computer Engineering
Sinhgad Institute of Technology & Science Narhe,Pune

 ;*******************Output***********************************
;[root@localhost A5_Far]# nasm -f elf64 A5_file1.asm
;[root@localhost A5_Far]# nasm -f elf64 A5_file2.asm
;[root@localhost A5_Far]# ld -o A5_file1 A5_file1.o A5_file2.o
;[root@localhost A5_Far]# ./A5_file1
```

```
;ML assignment 05 :- String Operation using Far Procedure
;-------------------------------------------------

;Enter filename for string operation    : myfile.txt

;Enter character to search    : e

;No. of spaces are    : 0007
;No. of lines are    : 0003
;No. of character occurances are    : 000B

;Exit from program...


;[root@localhost A5_Far]#
;*********************************************************
```

*Note:All files are save within single folder.*

**Assignment No.06**

**Problem Statement: Write X86/64 ALP to switch from real mode to protected mode and display the values of GDTR, LDTR, IDTR, TR and MSW Registers.**

```
;*********************************************************
section .data
rmodemsg db 10,"Processor is in Real Mode"
rmsg_len equ $-rmodemsg

pmodemsg db 10,"Processor is in Protected Mode"
pmsg_len equ $-pmodemsg

gdtmsg db 10,"GDT Contents are::"
gdtmsg_len equ $-gdtmsg

ldtmsg db 10,"LDT Contents are::"
ldtmsg_len equ $-ldtmsg

idtmsg db 10,"IDT Contents are::"
idtmsg_len equ $-idtmsg

trmsg db 10,"Task Register Contents are::"
trmsg_len equ $-trmsg

mswmsg db 10,"Machine Status Word::"
mswmsg_len equ $-mswmsg

colmsg db ":"

nwline db 10
;*********************************************************
section .bss

gdt resd 1
resw 1
ldt resw 1
idt resd 1
resw 1
tr  resw 1

cr0_data resd 1

dispbuff resb 04

%macro display 2
    mov    rax,1    ;print
```

```
    mov    rdi,1    ;stdout/screen
    mov    rsi,%1    ;msg
    mov    rdx,%2    ;msg_len
    syscall
%endmacro

;*****************************************************
section .text
global _start
_start:
    smsw eax    ;Stores the machine status word (bits 0
                ;through   15 of control register CR0) into eax.

    mov [cr0_data],eax

    bt eax,0    ;Checking PE(Protected Mode Enable) bit(LSB),
                ;if 1=Protected Mode, else Real Mode
    jc prmode
    display rmodemsg,rmsg_len
    jmp nxt1

prmode: display pmodemsg,pmsg_len

nxt1:sgdt [gdt]
     sldt [ldt]
     sidt [idt]
     str [tr]
    ;.......display gdt data...........

     display gdtmsg,gdtmsg_len

     mov bx,[gdt+4]
     call display16_proc

     mov bx,[gdt+2]
     call display16_proc

     display colmsg,1

     mov bx,[gdt]
     call display16_proc

    ;......display ldt data...........

     display ldtmsg,ldtmsg_len
     mov bx,[ldt]
     call display16_proc
```

```
    ;......display idt data...........

    display idtmsg,idtmsg_len

    mov bx,[idt+4]
    call display16_proc

    mov bx,[idt+2]
    call display16_proc

    display colmsg,1

    mov bx,[idt]
    call display16_proc

    ;....display task register data.........

    display trmsg,trmsg_len

    mov bx,[tr]
    call display16_proc

    ;....display machine status word data......

    display mswmsg,mswmsg_len

    mov bx,[cr0_data+2]
    call display16_proc

    mov bx,[cr0_data]
    call display16_proc

    display nwline,1

    mov rax,60
    mov rdi,0
    syscall
;*************************************************************
display16_proc:
    mov rdi,dispbuff    ;point esi to buffer
    mov rcx,4         ;load number of digits to display
dispup1:
    rol bx,4         ;rotate number left by four bits
    mov dl,bl         ;move lower byte in dl
    and dl,0fh         ;mask upper digit of byte in dl
    add dl,30h         ;add 30h to calculate ASCII code
```

```
    cmp dl,39h          ;compare with 39h
    jbe dispskip1          ;if less than 39h akip adding 07 more
    add dl,07h          ;else add 07

dispskip1:
    mov [rdi],dl          ;store ASCII code in buffer
    inc rdi              ;point to next byte
    loop dispup1          ;decrement the count of digits to display
                ;if not zero jump to repeat

    display dispbuff,4    ;

    ret
;******************Output*************************************
;[root@localhost MIT2016]# nasm -f elf64 proc.asm
;[root@localhost MIT2016]# ld -o proc proc.o
;[root@localhost MIT2016]# ./proc

;Processor is in Protected Mode
;GDT Contents are::B7304000:007F
;LDT Contents are::0000
;IDT Contents are::81BDD000:0FFF
;Task Register Contents are::0040
;Machine Status Word::8005FFFF
;[root@localhost MIT2016]#
```

**Assignment No.07**

**Problem Statement:Write X86 program to sort the list of integers in ascending/descending order. Read the input from the text file and write the sorted data back to the same text file using bubble sort**

# A7_bsort.asm

```
;*************************************************************

%include    "macro.asm"
;*************************************************************
section .data
    nline         db    10
    nline_len     equ    $-nline

    ano         db     10,10,10,10,"ML assignment 07 :- Bubble sort using
file operations"
            db           10,"-----------------------------------------
------",10
    ano_len     equ    $-ano


    filemsg     db    10,"Enter filename of input data    : "
    filemsg_len     equ    $-filemsg

    omsg         db    10,"Sorting using bubble sort Operation successful."
            db    10,"Output stored in same file...",10,10
    omsg_len     equ    $-omsg

    errmsg     db    10,"ERROR in opening/reading/writing File...",10
    errmsg_len     equ    $-errmsg

    ermsg         db    10,"ERROR in writing File...",10
    ermsg_len     equ    $-ermsg

    exitmsg     db    10,10,"Exit from program...",10,10
    exitmsg_len     equ    $-exitmsg

;*************************************************************
section .bss
    buf             resb    1024
    buf_len         equ    $-buf        ; buffer length

    filename         resb    50
```

```
    filehandle        resq    1
    abuf_len        resq    1        ; actual buffer length

    array            resb    10
    n            resq    1
;*************************************************************
section .text
    global _start

_start:
        display    ano,ano_len                ;assignment no.

        display    filemsg,filemsg_len
        accept     filename,50
        dec    rax
        mov    byte[filename + rax],0           ; blank char/null char

        fopen    filename            ; on succes returns handle
        cmp    rax,-1H                ; on failure returns -1
        je    Error
        mov    [filehandle],rax

        fread    [filehandle],buf, buf_len
        dec    rax                    ; EOF
        mov    [abuf_len],rax

        call    bsort_proc

        jmp    Exit

Error:    display    errmsg, errmsg_len

Exit:        display    exitmsg,exitmsg_len


    mov    rax,60    ;exit
    mov    rdi,0
    syscall
;*************************************************************
bsort_proc:                        ; Bubble sort procedure
        call    buf_array_proc

        mov    rax,0
        mov    rbp,[n]
        dec    rbp
```

```
        mov     rcx,0
        mov     rdx,0
        mov     rsi,0
        mov     rdi,0

        mov     rcx,0                   ; i=0

oloop:          mov     rbx,0                   ; j=0

        mov     rsi,array               ; a[j]

iloop:          mov     rdi,rsi                 ; a[j+1]
        inc     rdi

        mov     al,[rsi]
        cmp     al,[rdi]
        jbe     next

        mov     dl,0
        mov     dl,[rdi]                ; swap
        mov     [rdi],al
        mov     [rsi],dl

next:           inc     rsi
        inc     rbx                     ; j++
        cmp     rbx,rbp
        jb      iloop

        inc     rcx
        cmp     rcx,rbp
        jb      oloop

    fwrite    [filehandle],omsg, omsg_len
    fwrite    [filehandle],array,[n]

    fclose [filehandle]

    display     omsg, omsg_len
    display     array,[n]

    RET

Error1:
    display     ermsg, ermsg_len
    RET
;*************************************************************
buf_array_proc:
```

```
    mov     rcx,0
    mov     rsi,0
    mov     rdi,0

    mov     rcx,[abuf_len]
    mov     rsi,buf
    mov     rdi,array

next_num:
    mov     al,[rsi]
    mov     [rdi],al

    inc     rsi         ; number
    inc     rsi         ; newline
    inc     rdi

    inc     byte[n]         ; counter

    dec     rcx         ; number
    dec     rcx         ; newline
    jnz     next_num
    ret
;*********************************************************
```

## macro.asm

```
;*********************************************************
;macro.asm
;macros as per 64 bit conventions

%macro accept 2
    mov     rax,0         ;read
    mov     rdi,0         ;stdin/keyboard
    mov     rsi,%1     ;buf
    mov     rdx,%2     ;buf_len
    syscall
%endmacro

%macro display 2
    mov     rax,1         ;print
    mov     rdi,1         ;stdout/screen
    mov     rsi,%1     ;msg
    mov     rdx,%2     ;msg_len
    syscall
```

```
%endmacro

%macro fopen 1
    mov     rax,2         ;2 for open file
    mov     rdi,%1             ;filename
    mov     rsi,2         ;mode RW
    mov     rdx,0777o     ;File permissions
    syscall
%endmacro

%macro fread 3
    mov     rax,0     ;read
    mov     rdi,%1     ;filehandle
    mov     rsi,%2     ;buf
    mov     rdx,%3     ;buf_len
    syscall
%endmacro

%macro fwrite 3
    mov     rax,1     ;write/print
    mov     rdi,%1     ;filehandle
    mov     rsi,%2     ;buf
    mov     rdx,%3     ;buf_len
    syscall
%endmacro

%macro fclose 1
    mov     rax,3      ;close
    mov     rdi,%1      ;file handle
    syscall
%endmacro
;*************************************************************
```

## data.txt

```
4 5 1 2 3
;*************************************************************

after execution data.txt is:

;*************************************************************
Sorting using bubble sort Operation successful.
Output stored in same file...
12345
;*************************************************************
```

*Note:All files are save within single folder.*

**Assignment No.08**

**Problem Statement:Write X86 menu driven Assembly Language Program (ALP) to implement OS (DOS) commands TYPE, COPY and DELETE using file operations. User is supposed to provide command line arguments in all cases.**

```
%macro cmn 4            ;input/output
    mov rax,%1
    mov rdi,%2
    mov rsi,%3
    mov rdx,%4
    syscall
%endmacro
%macro exit 0
    mov rax,60
    mov rdi,0
    syscall
%endmacro

%macro fopen 1
    mov    rax,2        ;open
    mov    rdi,%1    ;filename
    mov    rsi,2        ;mode RW
    mov    rdx,0777o    ;File permissions
    syscall
%endmacro

%macro fread 3
    mov    rax,0        ;read
    mov    rdi,%1    ;filehandle
    mov    rsi,%2    ;buf
    mov    rdx,%3    ;buf_len
    syscall
%endmacro

%macro fwrite 3
    mov    rax,1        ;write/print
    mov    rdi,%1    ;filehandle
    mov    rsi,%2    ;buf
    mov    rdx,%3    ;buf_len
    syscall
%endmacro

%macro fclose 1
    mov    rax,3        ;close
```

```
    mov    rdi,%1    ;file handle
    syscall
%endmacro

section .data
    menu db 'MENU : ',0Ah
        db "1. TYPE",0Ah
        db "2. COPY",0Ah
        db "3. DELETE",0Ah
        db "4. Exit",0Ah
        db "Enter your choice : "
    menulen equ $-menu
    msg db "Command : "
    msglen equ $-msg
    cpysc db "File copied successfully !!",0Ah
    cpysclen equ $-cpysc
    delsc db 'File deleted successfully !!',0Ah
    delsclen equ $-delsc
    err db "Error ...",0Ah
    errlen equ $-err
    cpywr db 'Command does not exist',0Ah
    cpywrlen equ $-cpywr
    err_par db 'Insufficient parameter',0Ah
    err_parlen equ $-err_par

section .bss
    choice resb 2
    buffer resb 50
    name1 resb 15
    name2 resb 15
    cmdlen resb 1
    filehandle1 resq 1
    filehandle2 resq 1
    abuf_len        resq    1         ; actual buffer length
    dispnum resb 2
    buf resb    4096
    buf_len equ $-buf         ; buffer initial length

section .text
global _start
_start:
again:    cmn 1,1,menu,menulen
    cmn 0,0,choice,2
    mov al,byte[choice]
    cmp al,31h
```

```
    jbe op1
    cmp al,32h
    jbe op2
    cmp al,33h
    jbe op3
        exit
        ret
op1:
    call tproc
    jmp again
op2:
    call cpproc
    jmp again
op3:
    call delproc
    jmp again


;type command procedure
tproc:
    cmn 1,1,msg,msglen
    cmn 0,0,buffer,50
    mov byte[cmdlen],al
    dec byte[cmdlen]

    mov rsi,buffer
    mov al,[rsi]            ;search for correct type command
    cmp al,'t'
    jne skipt
    inc rsi
    dec byte[cmdlen]
    jz skipt
    mov al,[rsi]
    cmp al,'y'
    jne skipt
    inc rsi
    dec byte[cmdlen]
    jz skipt
    mov al,[rsi]
    cmp al,'p'
    jne skipt
    inc rsi
    dec byte[cmdlen]
    jz skipt
    mov al,[rsi]
```

```
    cmp al,'e'
    jne skipt
    inc rsi
    dec byte[cmdlen]
    jnz correctt
    cmn 1,1,err_par,err_parlen
    call exit

skipt:    cmn 1,1,cpywr,cpywrlen
    exit
correctt:
    mov rdi,name1             ;finding file name
    call find_name

    fopen name1             ; on succes returns handle
    cmp rax,-1H             ; on failure returns -1
    jle error
    mov [filehandle1],rax

    xor rax,rax
    fread [filehandle1],buf, buf_len
    mov [abuf_len],rax
    dec byte[abuf_len]

    cmn 1,1,buf,abuf_len        ;printing file content on screen

ret


;copy command procedure
cpproc:
    cmn 1,1,msg,msglen
    cmn 0,0,buffer,50         ;accept command
    mov byte[cmdlen],al
    dec byte[cmdlen]

    mov rsi,buffer
    mov al,[rsi]             ;search for copy
    cmp al,'c'
    jne skip
    inc rsi
    dec byte[cmdlen]
    jz skip
    mov al,[rsi]
    cmp al,'o'
```

```
    jne skip
    inc rsi
    dec byte[cmdlen]
    jz skip
    mov al,[rsi]
    cmp al,'p'
    jne skip
    inc rsi
    dec byte[cmdlen]
    jz skip
    mov al,[rsi]
    cmp al,'y'
    jne skip
    inc rsi
    dec byte[cmdlen]
    jnz correct
    cmn 1,1,err_par,err_parlen
    exit

skip:    cmn 1,1,cpywr,cpywrlen
    exit
correct:
    mov rdi,name1            ;finding first file name
    call find_name

    mov rdi,name2            ;finding second file name
    call find_name

skip3:    fopen name1            ; on succes returns handle
    cmp rax,-1H            ; on failure returns -1
    jle error
    mov [filehandle1],rax

    fopen name2            ; on succes returns handle
    cmp rax,-1H            ; on failure returns -1
    jle error
    mov [filehandle2],rax

    xor rax,rax
    fread [filehandle1],buf, buf_len
    mov [abuf_len],rax
    dec byte[abuf_len]

    fwrite [filehandle2],buf, [abuf_len]        ;write to file
```

```
    fclose    [filehandle1]
    fclose    [filehandle2]
    cmn 1,1,cpysc,cpysclen

    jmp again
error:
    cmn 1,1,err,errlen
    exit
ret



;delete command procedure
delproc:

    cmn 1,1,msg,msglen
    cmn 0,0,buffer,50        ;accept command
    mov byte[cmdlen],al
    dec byte[cmdlen]

    mov rsi,buffer
    mov al,[rsi]            ;search for copy
    cmp al,'d'
    jne skipr
    inc rsi
    dec byte[cmdlen]
    jz skipr
    mov al,[rsi]
    cmp al,'e'
    jne skipr
    inc rsi
    dec byte[cmdlen]
    jz skipr
    mov al,[rsi]
    cmp al,'l'
    jne skipr
    inc rsi
    dec byte[cmdlen]
    jnz correctr
    cmn 1,1,err_par,err_parlen
    exit

skipr:    cmn 1,1,cpywr,cpywrlen
    exit
```

```
correctr:
    mov rdi,name1             ;finding first file name
    call find_name

    mov rax,87              ;unlink system call
    mov rdi,name1
    syscall

    cmp rax,-1H              ; on failure returns -1
    jle errord
    cmn 1,1,delsc,delsclen
    jmp again

errord:
    cmn 1,1,err,errlen
    exit

ret


find_name:                  ;finding file name from command
    inc rsi
    dec byte[cmdlen]
cont1:    mov al,[rsi]
    mov [rdi],al
    inc rdi
    inc rsi
    mov al,[rsi]
    cmp al,20h              ;searching for space
    je skip2
    cmp al,0Ah              ;searching for enter key
    je skip2
    dec byte[cmdlen]
    jnz cont1
    cmn 1,1,err,errlen
    exit

skip2:
ret
```

**Assignment No.09**

**Problem Statement:** **Problem Statement: Write x86 ALP to find the factorial of a given integer number on a command line by using recursion. Explicit stack manipulation is expected in the code.**

```
;*************************************************************
section .data

    nummsg db "***Program to find Factorial of a number*** ",10
           db "Enter the number : ",
    nummsg_len equ $-nummsg

    resmsg db "Factorial is : "
    resmsg_len equ $-resmsg

    thankmsg db 10,"Thank you ",10
    thankmsg_len equ $-thankmsg

    zerofact db " 00000001 "
    zerofactlen equ $-zerofact

;*************************************************************
section .bss

    dispbuff resb 16
    result resb 4
    num resb 1
    num1 resb 1
    numascii resb 3

    %macro display 2
       mov rax,01
       mov rdi,01
       mov rsi,%1
       mov rdx,%2
       syscall
    %endmacro

       %macro accept 2
      mov rax,0
      mov rdi,0
      mov rsi,%1
      mov rdx,%2
      syscall
       %endmacro
;*************************************************************
section .text
```

```
global _start
_start:

    display nummsg,nummsg_len
    accept numascii,3            ;accept number from user
    call packnum8_proc            ;convert number from ascii to hex
    mov [num],bl

    display resmsg,resmsg_len

    mov al,[num]            ;store number in accumulator
    cmp al,01h
    jbe endfact

    mov bl,[num]
    call proc_fact
    mov rbx,rax
    call disp64_proc
    jmp exit

endfact:
    display zerofact,zerofactlen

exit:    display thankmsg,thankmsg_len

    mov rax,60
    mov rdi,0
    syscall
    ret

;*************************************************************
disp64_proc:
    mov rdi,dispbuff    ;point esi to buffer
    mov rcx,16        ;load number of digits to display
dispup1:
    rol rbx,4        ;rotate number left by four bits
    mov dl,bl        ;move lower byte in dl
    and dl,0fh        ;mask upper digit of byte in dl
    add dl,30h        ;add 30h to calculate ASCII code
    cmp dl,39h        ;compare with 39h
    jbe dispskip1        ;if less than 39h akip adding 07 more
    add dl,07h        ;else add 07

dispskip1:
    mov [rdi],dl        ;store ASCII code in buffer
    inc rdi            ;point to next byte
    loop dispup1        ;decrement the count of digits to display
```

```
                ;if not zero jump to repeat
    display dispbuff,16    ;
    ret
;*************************************************************
packnum8_proc:
    mov bx,0
    mov ecx,02
    mov esi,numascii
up1:
    rol bl,04
    mov al,[esi]
    cmp al,39h
    jbe skip1
    sub al,07h
skip1:    sub al,30h
    add bl,al
    inc esi
    loop up1
    ret
;***********************Recursion************************************
;There are two kinds of recursion: direct and indirect.
;In direct recursion, the procedure calls itself and
;in indirect recursion, the first procedure calls a second
;procedure,which in turn, calls the first procedure.

proc_fact:
    cmp bl, 1
    jne do_calculation
    mov ax, 1
    ret
do_calculation:
    push rbx
    dec bl
    call proc_fact
    pop rbx
    mul bl
    ret
;***************************************8
;[root@localhost vikas]# cd MIT2016/
;[root@localhost MIT2016]# nasm -f elf64 factorial.asm
;[root@localhost MIT2016]# ld -o factorial factorial.o
;[root@localhost MIT2016]# ./factorial
;***Program to find Factorial of a number***
;Enter the number : 04
;Factorial is : 0000000000000018
;Thank you
;[root@localhost MIT2016]#
```

**Assignment No.12**

**Problem Statement:Write 80387 ALP to obtain: i) Mean ii) Variance iii) Standard Deviation.**

```
section .data

msg1 db 10, 'mean is:  '
msg1len equ $- msg1

msg2 db 10, 'std deviation is:'
msg2len equ $- msg2

msg3 db 10, 'variance is:'
msg3len equ $- msg3

data dd 9.0,1.0
datacnt dw 02
hdec dq 100

decpt db '.'

section .bss

res rest 01
mean resd 01
var resd 01
dispbuff resb 01

%macro disp 2
    mov eax,04
    mov ebx,01
    mov ecx,%1
    mov edx,%2
    int 80h
%endmacro
%macro accept 2
    mov eax,03
    mov ebx,00
    mov ecx,%1
    mov edx,%2
    int 80h
%endmacro

section .text
```

```
global _start
_start:

disp msg1,msg1len

       finit
    fldz
    mov rbx,data
    mov rsi,00
    xor rcx,rcx
    mov cx,[datacnt]

bk:    fadd dword [rbx+rsi*4]
    inc rsi
    loop bk

    fidiv word[datacnt]
    fst dword[mean]
    call dispres
    MOV RCX,00
    MOV CX,[datacnt]
    MOV RBX,data
    MOV RSI,00
    FLDZ
up1:    FLDZ
    FLD DWORD[RBX+RSI*4]
    FSUB DWORD[mean]
    FST ST1
    FMUL
    FADD
    INC RSI
    LOOP up1
    FIDIV word[datacnt]
    FST dWORD[var]
    FSQRT
    disp msg2,msg2len
    CALL dispres
    FLD dWORD[var]
    disp msg3,msg3len
    CALL dispres


exit:     mov eax,01
    mov ebx,00
    int 80h
```

```
disp8_proc:
    mov rdi,dispbuff
    mov rcx,02
back:    rol bl,04
    mov dl,bl
    and dl,0FH
    cmp dl,09
    jbe next1
    add dl,07H
next1:  add dl,30H
    mov [rdi],dl
    inc rdi
    loop back
    ret

dispres:
    fimul dword[hdec]
    fbstp tword[res]
    xor rcx,rcx
    mov rcx,09H
    mov rsi,res+9
up2:    push rcx
    push rsi
    mov bl,[rsi]
    call disp8_proc
    disp dispbuff,2
    pop rsi
    dec rsi
    pop rcx
    loop up2
    disp decpt,1
    mov bl,[res]
    call disp8_proc
    disp dispbuff,2
    ret
output:
student@student-OptiPlex-390:~$ nasm -f elf64 mean.nasm
student@student-OptiPlex-390:~$ ld -o mean  mean.o
student@student-OptiPlex-390:~$ ./mean
mean is:  000000000000000005.00
std deviation is:000000000000000004.00
variance is:000000000000000016.00
student@student-OptiPlex-390:~$ ^C
student@student-OptiPlex-390:~$
```

**Assignment No.13**

**Problem Statement: Write a TSR to generate the pattern of the frequency tones by reading the Real Time Clock (RTC). The duration of the each tone is solely decided by the programmer.**

```
;*************************************************************

code segment
assume cs:code
org 100h                ;prog seg prefix addrss
jmp initze              ;hex no of 256
savint dd ?             ;for saving address of es:bx
count dw 0000h          ;count of tics

hours db ?
mins db ?
sec db ?

testnum:
        push ax    ;store all the contents of register
        push bx    ;(not to change original values of register)
        push cx
        push dx
        push cs
        push es
        push si
        push di

        mov ax,0b800h   ;starting address of display
        mov es,ax
        mov cx,count
        inc cx
        mov count,cx
        cmp cx,011h

      jne exit

        mov cx,0000h
        mov count,cx
        call time
exit:
        pop di
        pop si
        pop es
        pop ds
```

```
        pop dx
        pop cx
        pop bx
        pop ax
        jmp cs:savint   ;jump to normal isr

;------------------convert procedure--------------------
convert proc
        and al,0f0h
        ror al,4
        add al,30h
        call disp
        mov al,dh
        and al,0fh
        add al,30h
        call disp
        ret
endp

;-------------------------time procedure----------------
time proc
        mov ah,02h      ;getting current time system clk
        int 1ah
        mov hours,ch    ;HH->ch, MM->cl, SS->dh
        mov mins,cl
        mov sec,dh

       ; mov bx,0E00h    ;location for displaying clk

       mov bx,3984
        mov al,hours       ;Display Hours
        mov dh,hours
        call convert
        mov al,':'
        call disp

        mov al,mins       ;Display Mins
        mov dh,mins
        call convert
        mov al,':'
        call disp


        mov al,sec       ;Display Seconds
        mov dh,sec
        call convert
```

```
        call tone

        ret
          endp


;-----------------------display procedue----------------
disp proc
         mov ah,9Ch      ;for setting attribute
                  ;ATTRIBUTE BYTE  BL  R  G  B  I  R  G  B
                                      ;BACKGROUND FOREGROUND
        mov es:bx,ax    ;write into vedio buffer
         inc bx
         inc bx
         ret
endp

;--------------- frequency tone procedure------------------

tone proc

        mov     al, 182         ; Prepare the speaker for the
         out     43h, al         ;  note.
         mov     ax, 4560        ; Frequency number (in
decimal)
         out     42h, al         ; Output low byte.
         mov     al, ah          ; Output high byte.
         out     42h, al
         in      al, 61h         ; Turn on note (get value from
                                 ;  port 61h).
         or      al, 00000011b   ; Set bits 1 and 0.
         out     61h, al         ; Send new value.
         mov     bx, 25          ; Pause for duration of note.
.pause1:
         mov     cx, 65535
.pause2:
         dec     cx
         jne     .pause2
         dec     bx
         jne     .pause1
         in      al, 61h         ; Turn off note (get value from
                                 ;  port 61h).
         and     al, 11111100b   ; Reset bits 1 and 0.
         out     61h, al         ; Send new value.
        ret
endp
;------------------initialization-----------------------
```

```
initze:
        push cs
        pop ds
        cli             ;clear int flag

        mov ah,35h      ;35 for get orignal add
        mov al,08h      ;intrrupt no
        int 21h
        mov word ptr savint,bx
        mov word ptr savint+2,es

        mov ah,25h              ;25 for set int add
        mov al,08h
        mov dx,offset testnum   ;new add for intrrupt
        int 21h

        mov ah,31h              ;make prog resident(request tsr)
        mov dx,offset initze    ;size of program

        sti                    ;set intrrupt flag
        int 21h
code ends
end
```

**Assignment No.14**

**Problem Statement: Write 80386 ALP to implement multitasking. Where each task is supposed to change the color of the text displayed at the center of the screen.**

```
;*************************************************************

code segment

     assume cs:code,ds:code
start: mov ax,cs
     mov ds,ax
       mov cl,0
       mov ch,0

task1: inc cl
     cmp cl,09
       je exit
     mov ax,0b800h
       mov es,ax
       mov si,1830

       mov al,'*'
     mov ah,93h
     mov es:[si],ax
     inc bl
     mov al,bl
     add al,30h
     mov ah,93h
     inc si
      inc si

       mov es:[si],ax
       call delay

     jmp task2



task2: inc ch
     cmp ch,09
     je exit
     mov ax,0b800h
     mov es,ax
     mov si,840
     mov al,'$'
      mov ah,0A1h
```

```
     mov es:[si],ax
     inc bh
     mov al,bh
     add al,30h
     mov ah,0A1h
     inc si
     inc si

     mov es:[si],ax
       call delay

    jmp task1


    delay proc near
    mov ax,0fffh
    d2:mov dx,0fffh
   d1:nop
     nop
     nop
     nop
     dec dx
    jnz d1
    dec ax
    jnz d2
    ret
    endp



exit:
    mov ah,4ch
    int 21h

code ends
end start
```