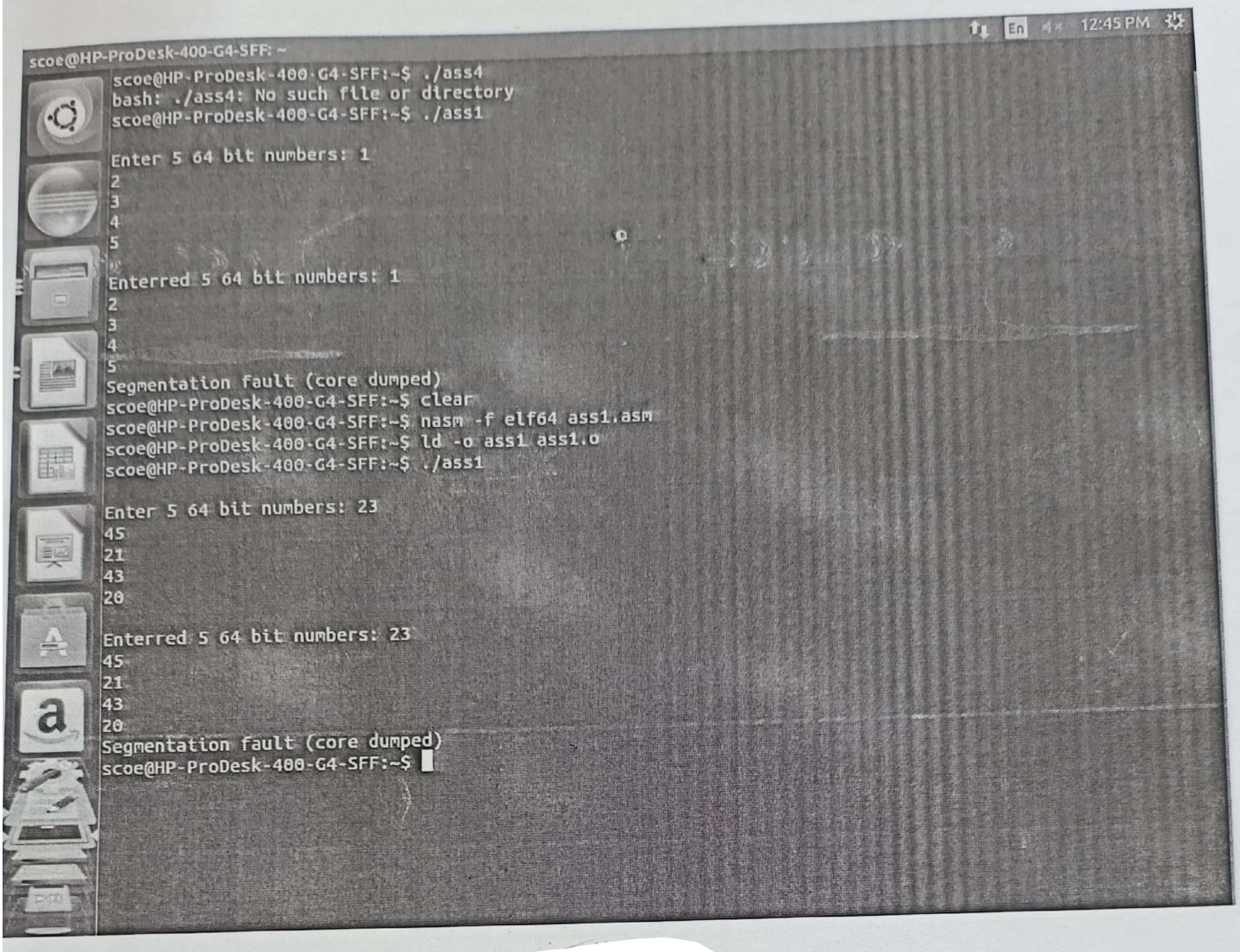
Assignment 1:

```
section .data
msg1 db 10,13,"Enter 5 64 bit numbers: "
len1 equ $-msg1
msg2 db 10,13,"Enterred 5 64 bit numbers: "
len2 equ $-msg2

section .bss
array resd 200
counter resb 1

section .text
global _start
_start:

;display
mov rax,1
mov rdi,1
mov rsi,msg1
mov rdx,len1
syscall

;accept
mov byte[counter],05
mov rbx,00

loop1:
mov rax,0
mov rdi,0
mov rsi,array
add rsi,rbx
mov rdx,17
syscall
add rbx,17
dec byte[counter]
jnz loop1

;display
mov rax,1
mov rdi,1
mov rsi,msg2
mov rdx,len2
syscall

;display
mov byte[counter],05
mov rbx,00

loop2:
mov rax,1
```

```
mov rdi,1
mov rsi,array
add rsi,rbx
mov rdx,17
syscall
add rbx,17
dec byte[counter]
jnz loop2
```

Output:



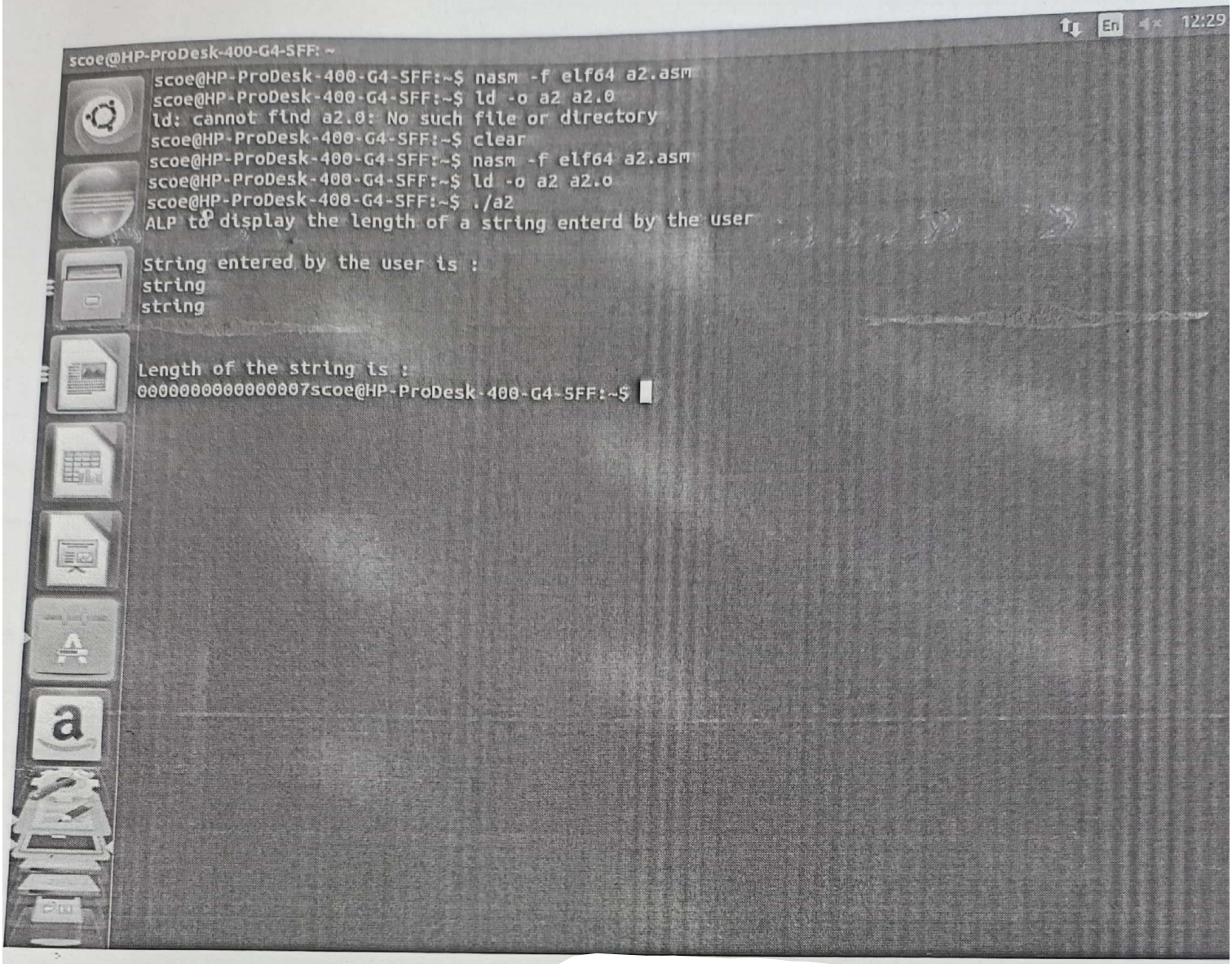
Assignment 2:

```
section .data
   msg db "ALP to display the length of a string enterd by the user",10
   msg_len equ $ - msg

   msg1 db 10, "String entered by the user is : ",10
   msg1_len equ $ - msg1

   msgop db 10, 10,"Length of the string is : " ,10
   msgop_len equ $ - msgop


section .bss
   string resb 50
   strl equ $-string

   result resb 50

%macro write 2
   mov rax,1
   mov rdi,1section .data
   msg db "ALP to display the length of a string enterd by the user",10
   msg_len equ $ - msg

   msg1 db 10, "String entered by the user is : ",10
   msg1_len equ $ - msg1

   msgop db 10, 10,"Length of the string is : " ,10
   msgop_len equ $ - msgop


section .bss
   string resb 50
   strl equ $-string

   result resb 50

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

section .text
   global _start

_start:
```

```
    write msg, msg_len
    write msg1, msg1_len

    mov rax, 0
    mov rdi, 0
    mov rsi, string
    mov rdx, 200
    syscall

    call disp

    mov rax, 60
    mov rdi, 0
    syscall

disp:

    mov rbx,rax ;store number in rbx
    mov rdi, result ;point rdi to result variable
    mov cx,16 ;load count of rotation in cl
up1:
    rol rbx,04 ;rotate number left by four bits
    mov al,bl ;move lower byte in dl
    and al,0fh ; get only LSB
    cmp al,09h ;compare with 39h
    jg add_37 ;if grater than 39h skip add 37
    add al,30h
    jmp skip ;else add 30
add_37 : add al,37h
skip:mov [rdi],al ;store ascii code in result variable
    inc rdi ;point to next byte
    dec cx ;decrement the count of digits to display
    jnz up1 ;if not zero jump to repeat
    write string, 50
    write msgop,msgop_len

    write result,16 ;call to macro
    ret
    mov rsi,%1
    mov rdx,%2
    syscall
%endmacro

section .text
    global _start

_start:

    write msg, msg_len
    write msg1, msg1_len
```

```
    mov rax, 0
    mov rdi, 0
    mov rsi, string
    mov rdx, 200
    syscall

    call disp

    mov rax, 60
    mov rdi, 0
    syscall

disp:

    mov rbx,rax ;store number in rbx
    mov rdi, result ;point rdi to result variable
    mov cx,16 ;load count of rotation in cl
up1:
    rol rbx,04 ;rotate number left by four bits
    mov al,bl ;move lower byte in dl
    and al,0fh ; get only LSB
    cmp al,09h ;compare with 39h
    jg add_37 ;if grater than 39h skip add 37
    add al,30h
    jmp skip ;else add 30
add_37 : add al,37h
skip:mov [rdi],al ;store ascii code in result variable
    inc rdi ;point to next byte
    dec cx ;decrement the count of digits to display
    jnz up1 ;if not zero jump to repeat
    write string, 50
    write msgop,msgop_len

    write result,16 ;call to macro
    ret
```

Output:



Practical No_3

```
section .data
   array db 11h,59h,33h,22h,44h

   msg1 db 10,"ALP to find the largest number in an array",10
   msg1_len equ $ - msg1

   msg2 db 10,"The Array contains the elements : ",10
   msg2_len equ $ - msg2

   msg3 db 10,10, "The Largest number in the array is : ",10
   msg3_len equ $ - msg3

section .bss
   counter resb 1
   result resb 4

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

section .text
   global _start

_start:
   write msg1 , msg1_len
   write msg2 , msg2_len

   mov byte[counter],05
   mov rsi,array
next:  mov al,[rsi]
   push rsi
   call disp
   pop rsi
   inc rsi
   dec byte[counter]
   jnz next

   write msg3 , msg3_len

   mov byte[counter],05
   mov rsi, array
   mov al, 0      ; al is an 8 bit register , al stores max
repeat: cmp al,[rsi]    ;cmp opr1 , opr2  : opr1 - opr2
   jg skip
   mov al,[rsi]
skip:  inc rsi
   dec byte[counter]
   Jnz repeat

   call disp

   mov rax,60
```

```
    mov rdi,1
    syscall

disp:
    mov bl,al ;store number in bl
    mov rdi, result ;point rdi to result variable
    mov cx,02 ;load count of rotation in cl
up1:
    rol bl,04 ;rotate number left by four bits
    mov al,bl ;move lower byte in dl
    and al,0fh ; get only LSB
    cmp al,09h ;compare with 39h
    jg add_37 ;if grater than 39h skip add 37
    add al,30h
    jmp skip1 ;else add 30
add_37: add al,37h
skip1:  mov [rdi],al ;store ascii code in result variable
    inc rdi ;point to next byte
    dec cx ;decrement the count of digits to display
    jnz up1 ;if not zero jump to repeat

    write result , 4

    ret
```

Output

```
ALP to find the largest number in an array

The Array contains the elements :
1159332244

The Largest number in the array is :
59
[Execution complete with exit code 1]
```

Practical No 4

```
%macro IO 4
mov rax,%1
mov rdi,%2
mov rsi,%3
mov rdx,%4
syscall
%endmacro
section .data
        m1 db "enter choice (+,-,*, /)" ,10 ; 10d -> line feed
  l1 equ $-m1
  m2 db "Write a switch case driven X86/64 ALP to perform 64-bit hexadecimal arithmetic operations (+,-,*, /) using suitable macros. Define procedure for each operation." ,10
  l2 equ $-m2
  m3 db "rahul ghosh 3236" ,10
  l3 equ $-m3
  madd db "addition here" ,10
  l4 equ $-madd
  msub db "subtraction here" ,10
  l5 equ $-msub
  mmul db "multiplication here" ,10
  l6 equ $-mmul
  mdiv db "division here" ,10
  l7 equ $-mdiv
  mspace db 10
  m_result db "result is "
  m_result_l equ $-m_result
  m_qou db "qoutient is "
  m_qou_l equ $-m_qou
  m_rem db "remainder is "
  m_rem_l equ $-m_rem
```

```
    m_default db "enter correct choice",10
    m_default_l equ $-m_default
section .bss
    choice resb 2
    _output resq 1
    _n1 resq 1
    _n2 resq 1
    temp_1 resq 1
    temp_2 resq 1
section .text
        global _start
_start:
    IO 1,1,m2,l2
    IO 1,1,m3,l3
    IO 1,1,m1,l1
    IO 0,0,choice,2
    cmp byte [choice],'+'
    jne case2
    call add_fun
    jmp exit
case2:
    cmp byte [choice],'-'
    jne case3
    call sub_fun
    jmp exit
case3:
    cmp byte [choice],'*'
    jne case4
    call mul_fun
    jmp exit
case4:
    cmp byte [choice],'/'
    jne case5
    call div_fun
    jmp exit
case5:
    cmp byte [choice],'a'
    jne error
    call add_fun
    call sub_fun
    call mul_fun
    call div_fun
    jmp exit
error:
    IO 1,1,m_default,m_default_l
    jmp exit
exit:
        mov rax, 60
        mov rdi, 0
        syscall
add_fun:
    IO 1,1,madd,l4
    mov qword[_output],0
    IO 0,0,_n1,17
    IO 1,1,_n1,17
    call ascii_to_hex
    add qword[_output],rbx
    IO 0,0,_n1,17
    IO 1,1,_n1,17
    call ascii_to_hex
    add qword[_output],rbx
    mov rbx,[_output]
```

```
    IO 1,1,mspace,1
    IO 1,1,m_result,m_result_l
    call hex_to_ascii
    ret
sub_fun:
    IO 1,1,msub,l5
    mov qword[_output],0
    IO 0,0,_n1,17
    IO 1,1,_n1,17
    ;IO 1,1,mspace,1
    call ascii_to_hex
    add qword[_output],rbx
    IO 0,0,_n1,17
    IO 1,1,_n1,17
    ;IO 1,1,mspace,1
    call ascii_to_hex
    sub qword[_output],rbx
    mov rbx,[_output]
    IO 1,1,mspace,1
    IO 1,1,m_result,m_result_l
    call hex_to_ascii

    ret
mul_fun:
    IO 1,1,mmul,l6 ; message
    IO 0,0,_n1,17    ; n1 input
    IO 1,1,_n1,17
    call ascii_to_hex; conversion returns hex
value in rbx
    mov [temp_1],rbx ; storing hex in temp_1
    IO 0,0,_n1,17    ;n2 input
    IO 1,1,_n1,17
    call ascii_to_hex
    mov [temp_2],rbx ; putting hex of n2 in
temp_2
    mov rax,[temp_1] ; temp_1->rax
    mov rbx,[temp_2] ;temp_2->rbx
    mul rbx        ; multiplication
    push rax
    push rdx
    IO 1,1,mspace,1
    IO 1,1,m_result,m_result_l
    pop rdx
    mov rbx,rdx; setting rbx value for
conversion
    call hex_to_ascii
    pop rax
    mov rbx,rax; setting rbx value for
conversion
    call hex_to_ascii  ; final output
ret
    _fun:
    IO 1,1,mdiv,l7
    IO 0,0,_n1,17    ; n1 input
    IO 1,1,_n1,17
    call ascii_to_hex; conversion returns hex
value in rbx
    mov [temp_1],rbx ; storing hex in temp_1
    IO 0,0,_n1,17    ;n2 input
    IO 1,1,_n1,17
    call ascii_to_hex
    mov [temp_2],rbx ; putting hex of n2 in
temp_2
    mov rax,[temp_1] ; temp_1->rax
```

```
mov rbx,[temp_2] ;temp_2->rbx
xor rdx,rdx
mov rax,[temp_1] ; temp_1->rax
mov rbx,[temp_2] ; temp_2->rbx
div rbx ; div
push rax
push rdx
IO 1,1,mspace,1
O 1,1,m_rem,m_rem_l
op rdx
ov rbx,rdx
ll hex_to_ascii; remainder output
 1,1,mspace,1
 1,1,m_qou,m_qou_l
op rax
ov rbx,rax
  hex_to_ascii; quotient output

    hex:
      _n1
        rcx, 16
        ox, rbx
    next1:
          rol rbx, 4
          mov al, [rsi]
          cmp al,47h
    jge error
          cmp al, 39h
          jbe sub30h
          sub al, 7
          sub30h:
                sub al, 30h
          add bl, al
          inc rsi
          loop next1

ret
hex_to_ascii:
      mov rcx, 16
      mov rsi,_output
      next2:
            rol rbx, 4
            mov al, bl
            and al, 0Fh
            cmp al, 9
            jbe add30h
            add al, 7
      add30h:
            add al, 30h
            mov [rsi], al
            inc rsi
            loop next2
            IO 1,1,_output,16
            IO 1,1,mspace,1
```

Output : -

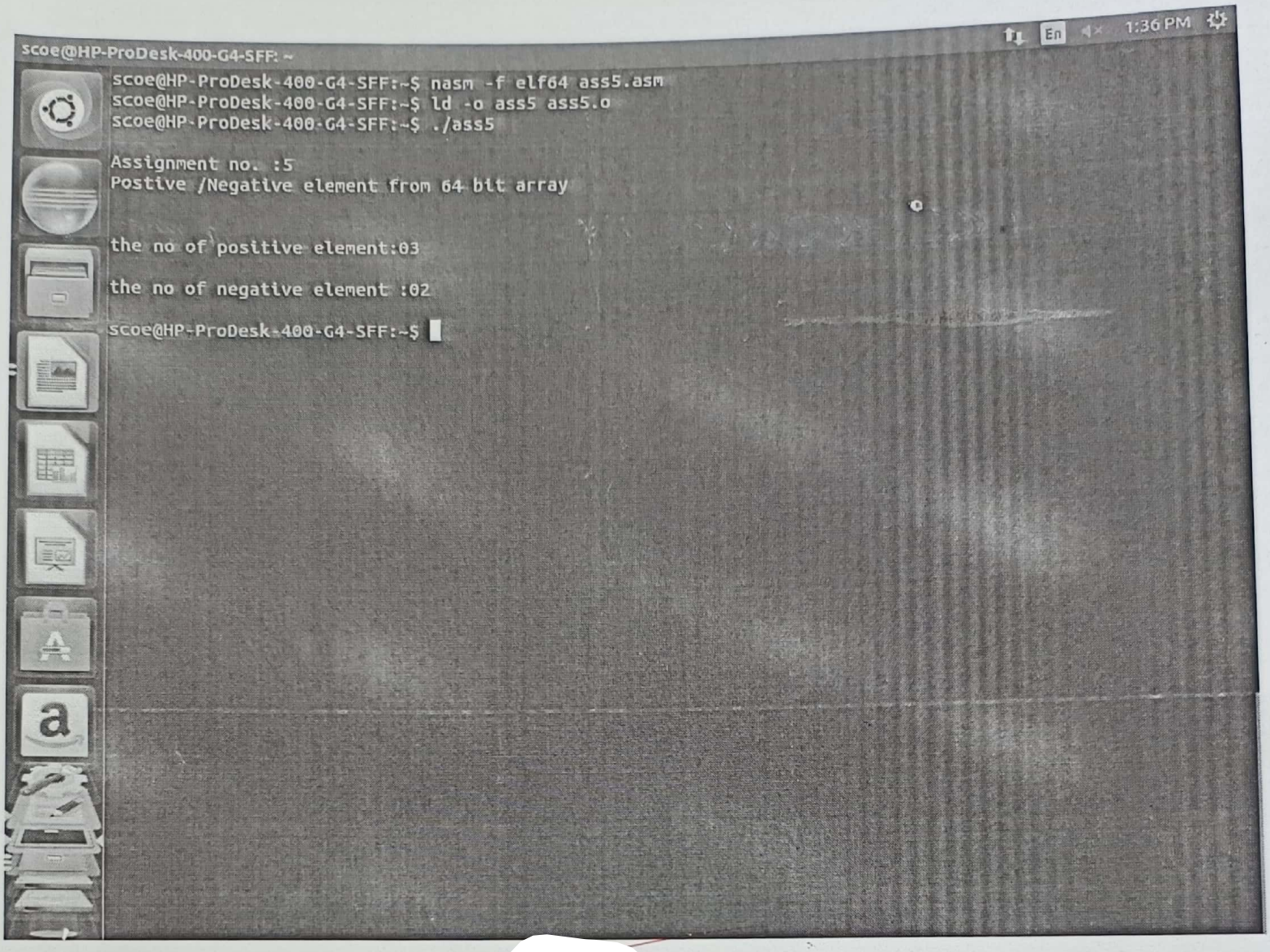
```
Write a switch case driven X86/64
exadecimal arithmetic operations
e macros. Define procedure for ea
rahul ghosh 3236
enter choice (+,-,*, /)
multiplication here
0000000000000004
0000000000000002
result is 0000000000000000
0000000000000008

[Execution complete with exit cod
```

Assignment : 5

```
section .data
        nline db 10,10
        nline_len equ $-nline
        ano db 10,"Assignment no. :5",10
        db "Postive /Negative element from 64 bit array",10
        ano_len equ $-ano

        arr64 dq -21H,5FH,-33H,2AH,62H
        n equ 5
        pmsg db 10,10,"the no of positive element:"
        pmsg_len equ $-pmsg
        nmsg db 10,10,"the no of negative element :"
        nmsg_len equ $-nmsg
section .bss
        p_count resq 1
        n_count  resq 1
        char_ans resb 16
%macro Print 2
        mov rax,1
        mov rdi,1
        mov rsi,%1
        mov rdx,%2
        syscall

%endmacro

%macro Read 2
        mov rax,0
        mov rdi,0
        mov rsi,%1
        mov rdx,%2
        syscall
%endmacro

%macro Exit 0
        mov rax,60
        mov rdi,0
        syscall
%endmacro

section .text
        global _start
        _start:
        Print ano, ano_len
        mov rsi,arr64
        mov rcx,n
        mov rbx,0
        mov rdx,0
next_num:
        mov rax,[rsi]
```

```
        Rol rax,1
        jc negative
Positive:
        Inc rbx
        jmp next
negative:
        inc rdx
next:
        add rsi,8
        dec rcx
        jnz next_num
        mov [p_count],rbx
        mov [n_count], rdx

        Print pmsg,pmsg_len
        mov rax, [p_count]
        call disp64_proc

        Print nmsg,nmsg_len
        mov rax,[n_count]
        call disp64_proc
        Print nline,nline_len
        Exit
disp64_proc:
        mov rbx,16
        mov rcx,2
        mov rsi,char_ans+1
cnt:
        mov rdx,0
        div rbx
        cmp dl,09h
        jbe add30
        add dl,07h
add30:
        add dl,30h
        mov [rsi],dl
        dec rsi
        dec rcx
        jnz cnt
        Print char_ans,2
ret
```

Output:



Assignment 6 :

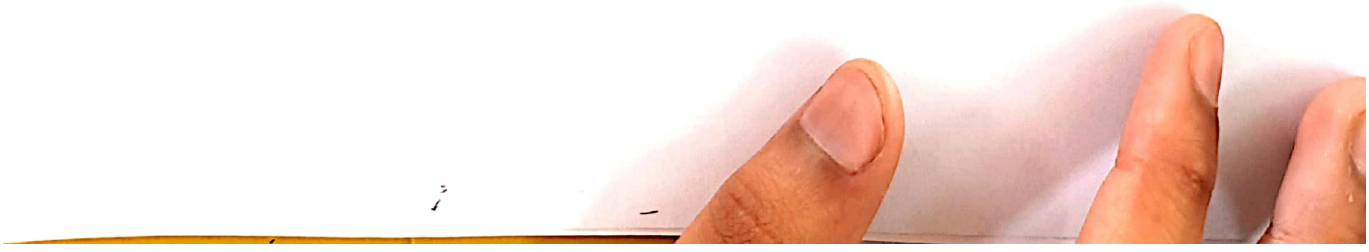
```
section .data
    nline           db      10,10
        nline_len   equ     $-nline

        ano         db      10,"    Assignment no         :6",
                    db      10,"-------------------------------------------------------",
                    db      10,"    Assignment Name:Conversion From HEX to BCD and BCD to
HEX Number.",
                    db      10,"-------------------------------------------------------",10
        ano_len             equ     $-ano

        menu        db      10,"1.Hex To BCD.",
                    db      10,"2.BCD To Hex.",
                    db      10,"3.Exit."
                    db      10,"Enter Your Choice::"
        menu_len    equ     $-menu

        hmsg        db      10,"Enter 4 digit Hex Number::"
        hmsg_len    equ     $-hmsg

        bmsg        db      10,"Enter 5 digit BCD Number::"
        bmsg_len    equ     $-bmsg

        ebmsg       db      10,"The Equivalent BCD Number is::"
        ebmsg_len   equ     $-ebmsg

        ehmsg       db      10,"The Equivalent Hex Number is::"
        ehmsg_len   equ     $-ehmsg

        emsg        db      10,"INVALID NUMBER INPUT",10
        emsg_len    equ     $-emsg
;-----------------------------------------------------------------------------
section .bss
        buf         resb    6       ;a variable which will store the accepted number from the user. 6
bytes are reserved for it. 5 digits of BCD+enter key
        char_ans    resb    4
        ans         resw    1

;-----------------------------------------------------------------------------

%macro      Print   2
        MOV RAX,1
        MOV RDI,1
      MOV   RSI,%1
      MOV   RDX,%2
   syscall
%endmacro

%macro      Read    2
        MOV RAX,0
```

```
        MOV RDI,0
      MOV   RSI,%1
      MOV   RDX,%2
   syscall
%endmacro


%macro Exit 0
        Print   nline,nline_len
        MOV  RAX,60
      MOV    RDI,0
   syscall
%endmacro
;-----------------------------------------------------------

section .text
        global _start

_start:
        Print   ano,ano_len

MENU:       Print   menu,menu_len
        Read    buf,2           ;accept choice i.e 1 digit+enter

        mov     al,[buf]        ;contains only digit character

c1:     cmp     al,'1'
        jne     c2
        call    HEX_BCD
        jmp     MENU

c2:     cmp     al,'2'
        jne     c3
        call    BCD_HEX
        jmp     MENU

c3:     cmp     al,'3'
        jne     invalid
        Exit

invalid:
        Print   emsg,emsg_len
        jmp     MENU
;-------------------------------------------------------------

HEX_BCD:                        ; procedure to convert Hex to BCD
        Print   hmsg,hmsg_len
        call    Accept_16               ;accept 4 digit hex number
        mov     ax,bx                   ;mov hex number in ax

        mov     bx,10                   ;for divide hex number by 10
        xor     bp,bp                   ;counter
```

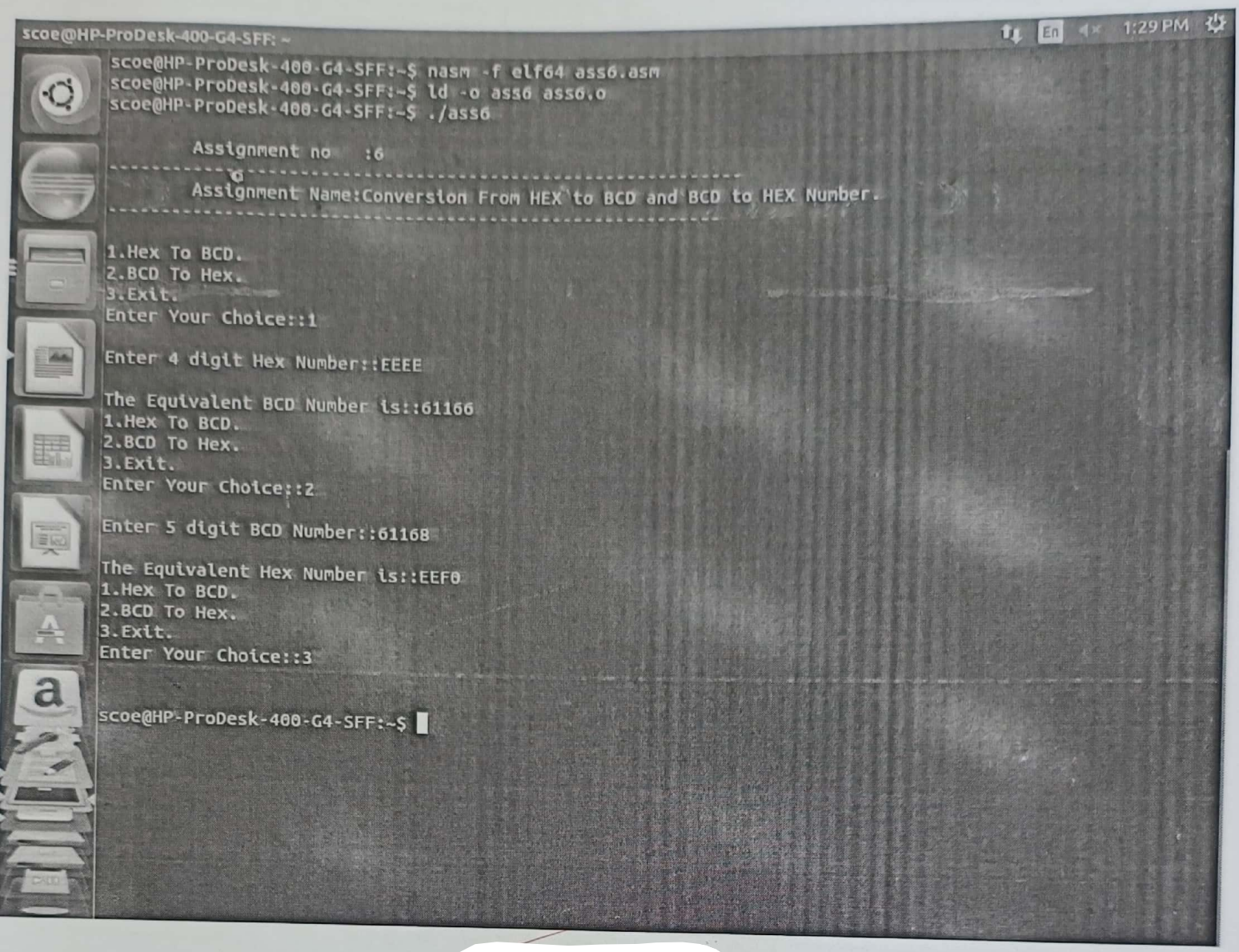
```
back:   xor     dx,dx           ;as dx each time contains remainder
        div     bx              ;divide ax by 10 ax=Q,dx=R
        push    dx              ;push dx on stack as it is bcd
        inc     bp              ;inc counter by 1. It counts the no of digits pushed on the stack

        cmp     ax,0            ;compare whether Q is 0 if 0 means number get over
        jne     back            ;mov to conversion of quotient

        Print   ebmsg,ebmsg_len

back1:  pop     dx              ;pop last digit pushed on stack
        add     dl,30h          ;add 30 to digit to make them decimal
        mov     [char_ans],dl   ;print individual digit
        Print   char_ans,1

        dec     bp              ;counter decrement . it checks whehter all contents of stack are
popped out
        jnz     back1           ;mov to next digit

RET
;-------------------------------------------------------

BCD_HEX:
        Print   bmsg,bmsg_len
        Read    buf,6           ;5 digit + 1 enter

        mov     rsi,buf         ;Points at the start of buffer
        xor     ax,ax           ;Previous digit =0
        mov     rbp,5           ;counter
        mov     rbx,10          ;multiplier

next:   xor     cx,cx           ;contains next digit each time
        mul     bx              ;(ax*bx)+cl
        mov     cl,[rsi]
        sub     cl,30h
        add     ax,cx

        inc     rsi             ;Point at the next digit
        dec     rbp
        jnz     next

        mov     [ans],ax        ;store ax in ans because ax get change in Print macro
        Print   ehmsg,ehmsg_len

        mov     ax,[ans]
        call    Disp_16         ;Print hex number


RET
;-------------------------------------------------------
```

```
Disp_16:                        ;Hex to Ascii(character) display
        MOV  RSI,char_ans+3
        MOV  RCX,4          ;counter
        MOV  RBX,16                     ;Hex no

next_digit:
        XOR  RDX,RDX
        DIV  RBX

        CMP  DL,9
        JBE  add30
        ADD  DL,07H

add30 :
        ADD  DL,30H
        MOV  [RSI],DL

        DEC  RSI
        DEC  RCX
        JNZ  next_digit

        Print  char_ans,4
ret
;-----------------------------------------------
Accept_16:                              ;Ascii(character) to hex number input
        Read  buf,5    ;4 digits of Hex and one enter key

        MOV  RCX,4     ; rcx is a counter for 4 digits of Hex
        MOV  RSI,buf  .; base address
        XOR  BX,BX     ;making the contents of bx =0

next_byte:
        SHL  BX,4     ; the earlier group of 4 bits will get discarded and new group of 4 bits will
get added in on right side
        MOV  AL,[RSI]  ; content of hex no goes to AL

        CMP  AL,'0'   ; the hex no gets compared with 0
        JB     error   ; jump if below i.e. if a negative no go to error
        CMP  AL,'9'   ; hex no is compared with 9, if its less than or equal to it, then its a valid hex
no.
        JBE    sub30    ; hence subtract 30

        CMP  AL,'A'   ; hex no gets compared with 'A'
        JB     error
        CMP  AL,'F'
        JBE    sub37    ; subtract 37 if below F, sice its a valid hex no.

        CMP  AL,'a'
        JB     error
        CMP  AL,'f'
        JBE    sub57    ; 57 will get subtracted if the hex no. is between 'a' to'f'
```

```
error:
        Print   emsg,emsg_len
        Exit

sub57: SUB   AL,20H   ; this and the following 2 instructions will get executed serially. Hence split up of 57 as 20, 7 and 30.
sub37: SUB   AL,07H
sub30: SUB   AL,30H

        ADD   BX,AX    ; this now valid hex no is added to contents of bx. after 4 such iterations we will get a binary equivalent of the hex no

        INC   RSI      ; increment rsi so that it points to the next hex digit
        DEC   RCX      ; decrement the counter rcx since its a counter for the no of digits of hex
        JNZ   next_byte ; keep going in a loop till the counter=0

RET
```



Output:



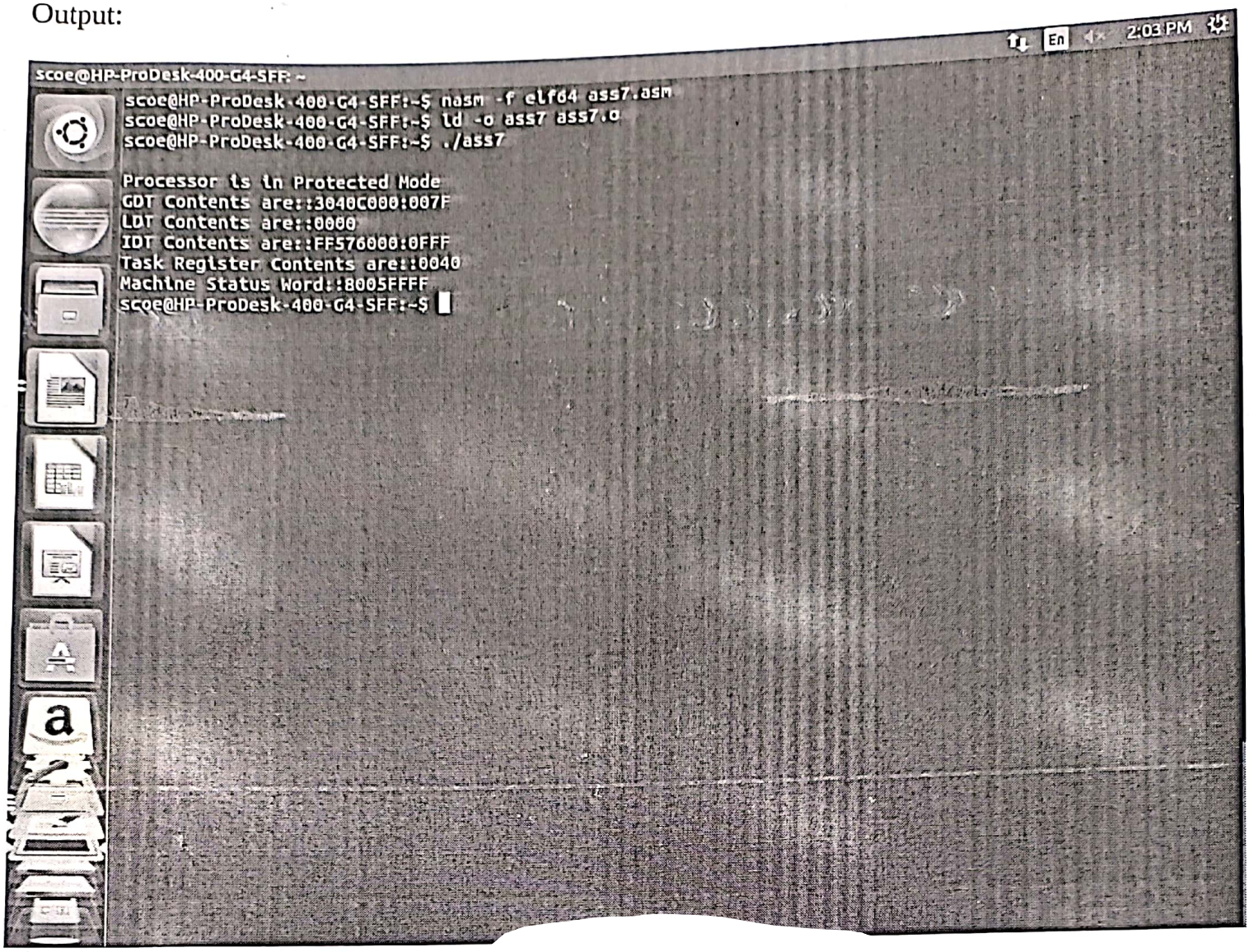
Assignment 7 :

```
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::'
        gmsg_len:equ $-gdtmsg

        ldtmsg db 10,'LDT Contents are::'
        lmsg_len:equ $-ldtmsg

        idtmsg db 10,'IDT Contents are::'
        imsg_len:equ $-idtmsg

        trmsg db 10,'Task Register Contents are::'
        tmsg_len: equ $-trmsg

        mswmsg db 10,'Machine Status Word::'
        mmsg_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

        dnum_buff resb 04

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

section .text
global _start
_start:
        smsw eax                ;Reading CR0. As MSW is 32-bit cannot use RAX register.
```

```
        mov [cr0_data],rax

        bt rax,1                ;Checking PE bit, if 1=Protected Mode, else Real Mode
        jc prmode
        print rmodemsg,rmsg_len
        jmp nxt1

prmode: print pmodemsg,pmsg_len

nxt1:   sgdt [gdt]
        sldt [ldt]
        sidt [idt]
        str [tr]
        print gdtmsg,gmsg_len

        mov bx,[gdt+4]
        call print_num

        mov bx,[gdt+2]
        call print_num

        print colmsg,1

        mov bx,[gdt]
        call print_num

        print ldtmsg,lmsg_len
        mov bx,[ldt]
        call print_num

        print idtmsg,imsg_len

        mov bx,[idt+4]
        call print_num

        mov bx,[idt+2]
        call print_num

        print colmsg,1

        mov bx,[idt]
        call print_num

        print trmsg,tmsg_len

        mov bx,[tr]
        call print_num

        print mswmsg,mmsg_len

        mov bx,[cr0_data+2]
```

```
        call print_num

        mov bx,[cr0_data]
        call print_num

        print nwline,1


exit:   mov rax,60
        xor rdi,rdi
        syscall

print_num:
        mov rsi,dnum_buff     ;point esi to buffer

        mov rcx,04            ;load number of digits to printlay

up1:
        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 skip1             ;if less than 39h skip adding 07 more
        add dl,07h            ;else add 07
skip1:
        mov [rsi],dl          ;store ASCII code in buffer
        inc rsi               ;point to next byte
        loop up1              ;decrement the count of digits to printlay
                              ;if not zero jump to repeat

        print dnum_buff,4     ;printlay the number from buffer

        ret
```

Output:

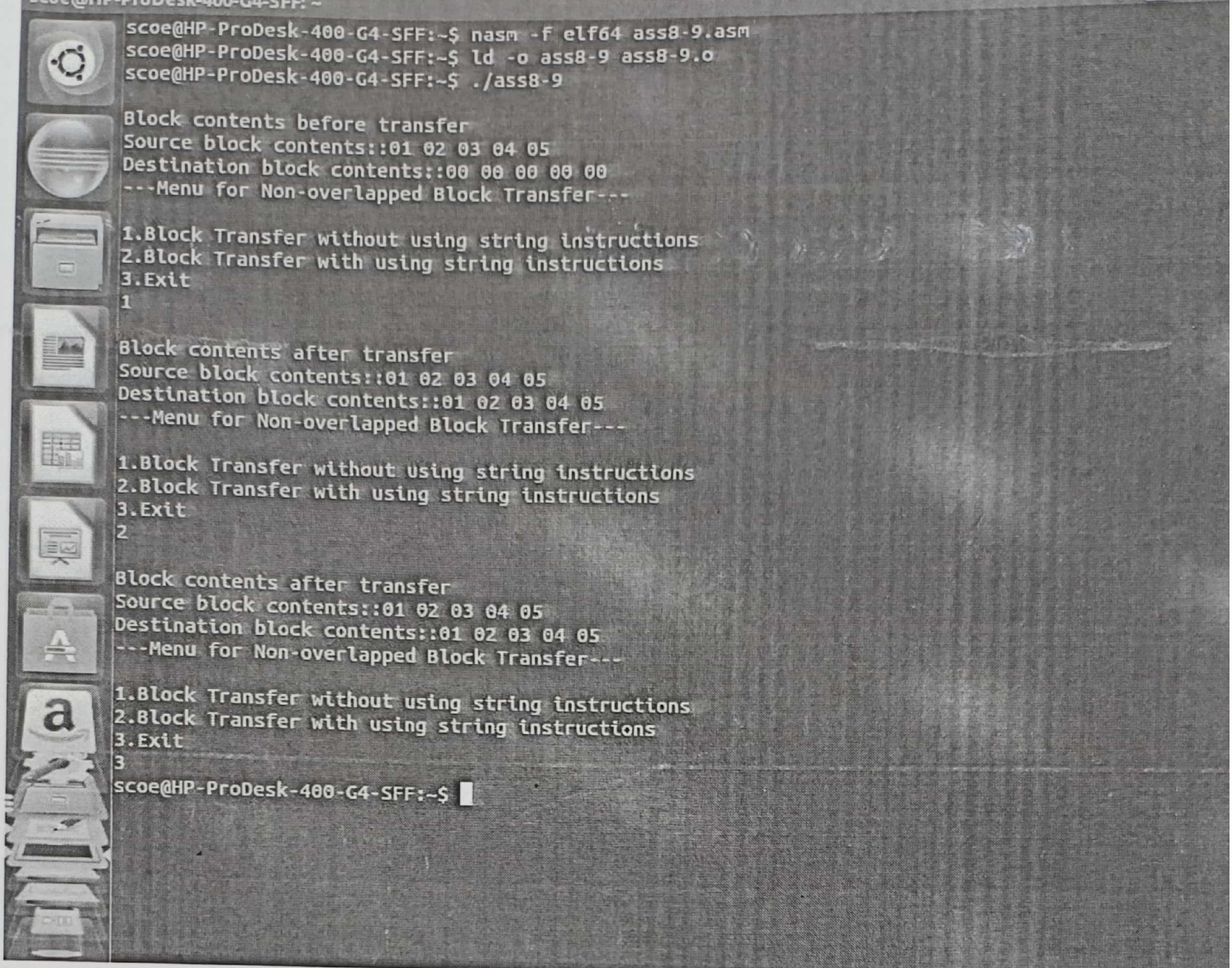


Assignment 8-9 :

```
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_
        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
```



```
scoe@HP-ProDesk-400-G4-SFF:~$ nasm -f elf64 ass8-9.asm
scoe@HP-ProDesk-400-G4-SFF:~$ ld -o ass8-9 ass8-9.o
scoe@HP-ProDesk-400-G4-SFF:~$ ./ass8-9

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
2

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
scoe@HP-ProDesk-400-G4-SFF:~$
```

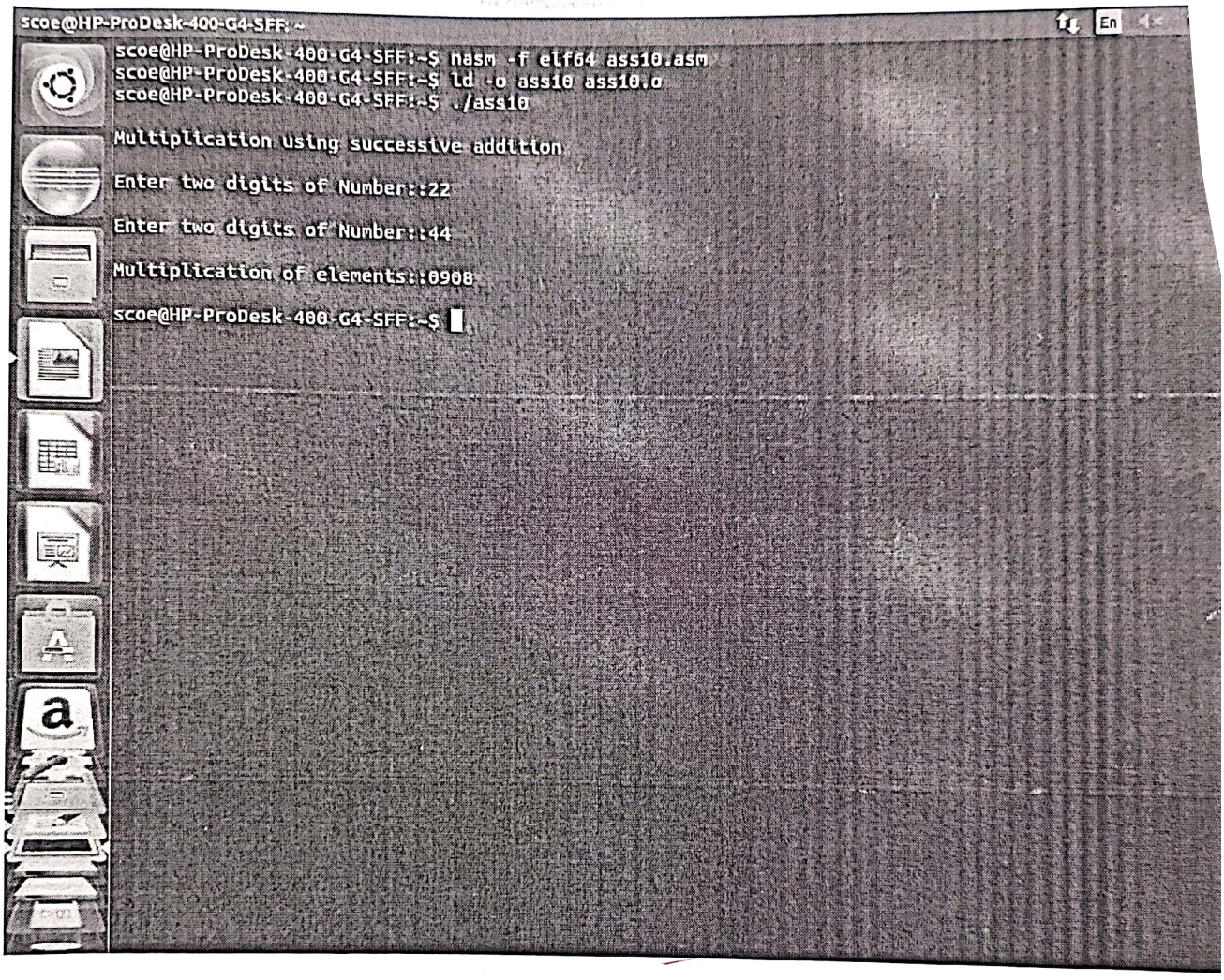
Assignment 10:

```
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:



```
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: