**Practical – 1**

**Problem statement:** Write an X86/64 ALP to accept five 64-bit Hexadecimal numbers from user and

store them in an array and display the accepted numbers.

**Program:**

section .data

mesg1 db "Enter the numbers ",10,13

mesg1len equ $-mesg1

mesg2 db "Display the numbers ",10,13

mesg2len equ $-mesg2

section .bss

num resb 10

section .text

global \_start

\_start:

mov rax,1;

mov rdi,1;

mov rsi,mesg1

mov rdx,mesg1len

syscall

mov rax,0

mov rdi,0

mov rsi,num

syscall

mov rax,1;

mov rdi,1;

mov rsi,mesg2

mov rdx,mesg2len

syscall

mov rax,1

mov rdi,1

mov rsi,num

syscall

mov rax,60

mov rdi,0

syscall

**Output:**

atharva@atharva :~$ nasm -f elf64 lab1.asm

atharva@atharva :~$ ld -o lab1 lab1.o

atharva@atharva :~$./lab1

Enter the numbers

8 4 2 3

Display the numbers

8 4 2 3

**Practical – 2**

**Problem statement:** Write an X86/64 ALP to accept a string and to display its length.

**Program code:**

section .data

mesg1 db "enter the string",10,13

mesg1len equ $-mesg1

mesg2 db"length of the string is:",10,13

mesg2len equ $-mesg2

section .bss

strbuff resb 200

strlen:equ $-strbuff

alen resq 1

num resb 16

%macro print 2

mov rax,1

mov rdi,1

mov rsi,%1

mov rdx,%2

syscall

%endmacro

section .text

global \_start

\_start:

print mesg1,mesg1len

mov rax,0

mov rdi,0

mov rsi,strbuff

mov rdx,strlen

syscall

dec rax

mov[alen],rax

print mesg2,mesg2len

call disp64\_proc

mov rax,60

mov rdi,0

syscall

disp64\_proc:

mov rdi,num

mov rdi,num

mov rcx,16

mov rbx,[alen]

dispupl:

rol rbx,4

mov al,bl;

and al,0fh

cmp al,09

jbe dispskipl

add al,07

dispskipl:

add al,30h

mov [rdi],al

inc rdi

loop dispupl

print num,16

ret

mov rax,60

mov rdi,0

syscall

**Output:**

atharva@atharva:~$ gedit lab2.asm

atharva@atharva:~$ nasm -f elf64 lab2.asm

atharva@atharva:~$ ld -o lab2 lab2.o

**Practical - 3**

**Problem statement:** Write an X86/64 ALP to find the largest of given Byte/Word/Dword/64-bit numbers.

**Program:**

section .data

    arr\_msg db &#39;Array Elements Are:: &#39;,10

    dq &#39;0fa10001h&#39;,10

    dq &#39;0b200002h&#39;,10

    dq &#39;0fff0003h&#39;,10

    dq &#39;0d400004h&#39;,10

    dq &#39;0fffffffh&#39;,10

    arr\_len : equ $-arr\_msg

    larg\_msg db &#39;Largest Number is::&#39;

    larg\_len: equ $-larg\_msg

    nwline db 10

    array dq 0fa10001h,0b200002h,0fff0003h,0d400004h,0fffffffh   ;array elements

    arrcnt dd 05h

section .bss

    arr\_num resb 16

    large resq 1

%macro dispmsg 2

    mov rax,1    ;System call for write

    mov rdi,1    ;standard output stream

    mov rsi,%1    ;message start address

    mov rdx,%2    ;message length

    syscall

%endmacro

section .text

    global \_start

\_start:

    dispmsg arr\_msg,arr\_len

    mov rsi,array

    mov rcx,[arrcnt]

    mov rax,[rsi]

    dec rcx

lup1:    add rsi,08    ;Point to next element

    cmp rax,[rsi]

    ja lskip1

    xchg rax,[rsi]

lskip1:    loop lup1

    mov [large],rax

    dispmsg larg\_msg,larg\_len

    mov rbx,[large]

    call disp\_num

    dispmsg nwline,1

exit:    mov rax,60

    mov rdi,0

    syscall

disp\_num:

    mov rdi,arr\_num    ;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

    dispmsg arr\_num,16

    ret

**Output:**

atharva@atharva:~$ gedit ass3.asm

atharva@atharva:~$ nasm -f elf64 ass3.asm

atharva@atharva:~$ ld -o ass3 ass3.o

atharva@atharva:~$ ./ass3

Array Elements Are::

0fa10001h

0b200002h

0fff0003h

0d400004h

0fffffffh

Largest Number is::000000000fffffffh

**Practical – 4**

**Problem statement:** Write a switch case driven X86/64 ALP to perform 64-bit hexadecimal arithmetic operations (+,-,\*, /) using suitable macros. Define procedure for each operation.

**Program:**

%macro scall 4

mov rax,%1

mov rdi,%2

mov rsi,%3

mov rdx,%4

syscall

%endmacro

;-------------------------.data section------------------------------

section .data

arr dq 000000000000003h,0000000000000002h

n equ 2

menu db 10d,13d,"\*\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*"

db 10d,13d,"1. Addition"

db 10d,13d,"2. Subtraction"

db 10d,13d,"3. Multiplication"

db 10d,13d,"4. Division"

db 10d,13d,"5. Exit"

db 10d,13d,"Enter your Choice: "

menu\_len equ $-menu

m1 db 10d,13d,"Addition: "

l1 equ $-m1

m2 db 10d,13d,"Substraction: "

l2 equ $-m2

m3 db 10d,13d,"Multiplication: "

l3 equ $-m3

m4 db 10d,13d,"Division: "

l4 equ $-m4

;-------------------------.bss section------------------------------

section .bss

answer resb 16 ;to store the result of operation

choice resb 2

;------------------------.text section -----------------------------

section .text

global \_start:

\_start:

up: scall 1,1,menu,menu\_len

scall 0,0,choice,2

cmp byte[choice],'1'

je case1

cmp byte[choice],'2'

je case2

cmp byte[choice],'3'

je case3

cmp byte[choice],'4'

je case4

cmp byte[choice],'5'

je case5

case1: scall 1,1,m1,l1

call addition

jmp up

case2: scall 1,1,m2,l2

call substraction

jmp up

case3: scall 1,1,m3,l3

call multiplication

jmp up

case4: scall 1,1,m4,l4

call division

jmp up

case5: mov rax,60

mov rdi,0

syscall

;procedures for arithmetic and logical operations

addition:

mov rcx,n

dec rcx

mov rsi,arr

mov rax,[rsi]

up1: add rsi,8

mov rbx,[rsi]

add rax,rbx

loop up1

call display

ret

substraction:

mov rcx,n

dec rcx

mov rsi,arr

mov rax,[rsi]

up2: add rsi,8

mov rbx,[rsi]

sub rax,rbx

loop up2

call display

ret

multiplication:

mov rcx,n

dec rcx

mov rsi,arr

mov rax,[rsi]

up3: add rsi,8

mov rbx,[rsi]

mul rbx

loop up3

call display

ret

division:

mov rcx,n

dec rcx

mov rsi,arr

mov rax,[rsi]

up4: add rsi,8

mov rbx,[rsi]

mov rdx,0

div rbx

loop up4

call display

ret

or:

mov rcx,n

dec rcx

mov rsi,arr

mov rax,[rsi]

up6: add rsi,8

mov rbx,[rsi]

or rax,rbx

loop up6

call display

ret

xor:

mov rcx,n

dec rcx

mov rsi,arr

mov rax,[rsi]

up7: add rsi,8

mov rbx,[rsi]

xor rax,rbx

loop up7

call display

ret

and:

mov rcx,n

dec rcx

mov rsi,arr

mov rax,[rsi]

up8: add rsi,8

mov rbx,[rsi]

and rax,rbx

loop up8

call display

ret

display:

mov rsi,answer+15

mov rcx,16

cnt: mov rdx,0

mov rbx,16

div rbx

cmp dl,09h

jbe add30

add dl,07h

add30: add dl,30h

mov [rsi],dl

dec rsi

dec rcx

jnz cnt

scall 1,1,answer,16

ret

**Output:**

atharva@atharva:~$ gedit lab4.asm

atharva@atharva:~$ nasm -f elf64 lab4.asm

atharva@atharva:~$ ld -o lab4 lab4.o

atharva@atharva:~$ ./lab4

\*\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*

1. Addition

2. Subtraction

3. Multiplication

4. Division

5. Exit

Enter your Choice: 1

Addition: 0000000000000005

\*\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*

1. Addition

2. Subtraction

3. Multiplication

4. Division

5. Exit

Enter your Choice: 2

Substraction: 0000000000000001

\*\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*

1. Addition

2. Subtraction

3. Multiplication

4. Division

5. Exit

Enter your Choice: 3

Multiplication: 0000000000000006

\*\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*

1. Addition

2. Subtraction

3. Multiplication

4. Division

5. Exit

Enter your Choice: 4

Division: 0000000000000001

\*\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*

1. Addition

2. Subtraction

3. Multiplication

4. Division

5. Exit

Enter your Choice: 5

**Practical – 5**

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

**Program:**

section .data

welmsg db 10,'Welcome to count +ve and -ve numbers in an array',10

welmsg\_len equ $-welmsg

pmsg db 10,'Count of +ve numbers::'

pmsg\_len equ $-pmsg

nmsg db 10,'Count of -ve numbers::'

nmsg\_len equ $-nmsg

nwline db 10

array dw 8505h,90ffh,0087h,0088h,8a9fh,00adh,0002h

arrcnt equ 7

pcnt db 0

ncnt db 0

section .bss

dispbuff resb 2

%macro print 2

mov rax, 1

mov rdi, 1

mov rsi, %1

mov rdx, %2

syscall

%endmacro

section .text

global \_start

\_start:

print welmsg,welmsg\_len

mov rsi,array

mov rcx,arrcnt

up1:

bt word[rsi],15

jnc pnxt

inc byte[ncnt]

jmp pskip

pnxt: inc byte[pcnt]

pskip: inc rsi

inc rsi

loop up1

print pmsg,pmsg\_len

mov bl,[pcnt]

call disp8num

print nmsg,nmsg\_len

mov bl,[ncnt]

call disp8num

print nwline,1 ;New line char

exit:

mov rax,60

mov rdi,0

syscall

disp8num:

mov rcx,2 ;Number digits to display

mov rdi,dispbuff ;Temp buffer

dup1:

rol bl,4 ;Rotate number from bl to get MS digit to LS digit

mov al,bl ;Move rotated number to AL

and al,0fh ;Mask upper digit

cmp al,09 ;Compare with 9

jbe dskip ;If number below or equal to 9 go to add only 30h

add al,07h ;Else first add 07h

dskip: add al,30h ;Add 30hWrite an ALP to count no. of positive and

negative numbers from the array.

mov [edi],al ;Store ASCII code in temp buff

inc rdi ;Increment pointer to next location in temp buff

loop dup1 ;repeat till ecx becomes zero

print dispbuff,2 ;display the value from temp buff

ret ;return to calling program

**Output:**

atharva@atharva:~$ gedit lab5.asm

atharva@atharva:~$ nasm -f elf64 lab5.asm

atharva@atharva:~$ ld -o lab5 lab5.o

atharva@atharva:~$ ./lab5

Welcome to count +ve and -ve numbers in an array

Count of +ve numbers::04

Count of -ve numbers::03

**Practical – 6**

**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).

**Program:**

section .data

nline db 10,10

nline\_len: equ $-nline

menu db 10,"-----------Menu----------"

db 10,"1. Hex to BCD "

db 10,"2. BCD to Hex"

db 10,"3. Exit "

db 10

db 10,"Enter your choice: "

menu\_len: equ $-menu

h2bmsg db 10,"Hex to BCD "

db 10,"Enter 4-digit Hex number: "

h2bmsg\_len: equ $-h2bmsg

b2hmsg db 10,"BCD to Hex "

db 10,"enter 5-digit BCD number: "

b2hmsg\_len: equ $-b2hmsg

hmsg db 10,13,"Equivalent Hex number is: "

hmsg\_len: equ $-hmsg

bmsg db 10,13,"Equivalent BCD number is: "

bmsg\_len: equ $-bmsg

emsg db 10,"You entered Invalid Data!!!",10

emsg\_len: equ $-emsg

section .bss

buf resb 6

buf\_len: equ $-buf

digitcount resb 1

ans resw 1

char\_ans resb 4

;macros as per 64-bit convensions

%macro print 2

mov rax,1 ; Function 1 - write

mov rdi,1 ; To stdout

mov rsi,%1 ; String address

mov rdx,%2 ; String size

syscall ; invoke operating system to WRITE

%endmacro

%macro read 2

mov rax,0 ; Function 0 - Read

mov rdi,0 ; from stdin

mov rsi,%1 ; buffer address

mov rdx,%2 ; buffer size

syscall ; invoke operating system to READ

%endmacro

%macro exit 0

print nline, nline\_len

mov rax, 60 ; system call 60 is exit

xor rdi, rdi ; we want return code 0

syscall ; invoke operating system to exit

%endmacro

section .text

global \_start

\_start:

print menu, menu\_len

read buf,2 ; choice + enter

mov al,[buf]

c1: cmp al,'1'

jne c2

call hex\_bcd

jmp \_start

c2: cmp al,'2'

jne c3

call bcd\_hex

jmp \_start

c3: cmp al,'3'

jne err

exit

err: print emsg,emsg\_len

jmp \_start

hex\_bcd:

print h2bmsg, h2bmsg\_len

call accept\_16

mov ax,bx

mov rbx,10

back:

xor rdx,rdx

div rbx

push dx

inc byte[digitcount]

cmp rax,0h

jne back

print bmsg, bmsg\_len

print\_bcd:

pop dx

add dl,30h ; possible digits are 0-9 so add 30H only

mov [char\_ans],dl ; store character in char\_ans

print char\_ans,1 ; print on screen in reverse order

dec byte[digitcount]

jnz print\_bcd

ret

bcd\_hex:

print b2hmsg, b2hmsg\_len

read buf,buf\_len ; buflen = 5 + 1

mov rsi,buf ; load bcd pointer

xor rax,rax ; sum

mov rbx,10

mov rcx,05 ; digit\_count

back1: xor rdx,rdx

mul ebx ; previous digit \* 10 = ans (rax\*rbx = rdx:rax)

xor rdx,rdx

mov dl,[rsi] ; Take current digit

sub dl,30h ; accepted digit is Decimal, so Sub 30H only

add rax,rdx

inc rsi

dec rcx

jnz back1

mov [ans],ax

print bmsg, bmsg\_len

mov ax,[ans]

call display\_16

ret

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

accept\_16:

read buf,5 ; buflen = 4 + 1

xor bx,bx

mov rcx,4

mov rsi,buf

next\_digit:

shl bx,04

mov al,[rsi]

cmp al,"0" ; "0" = 30h or 48d

jb error ; jump if below "0" to error

cmp al,"9"

jbe sub30 ; subtract 30h if no is in the range "0"-"9"

cmp al,"A" ; "A" = 41h or 65d

jb error ; jump if below "A" to error

cmp al,"F"

jbe sub37 ; subtract 37h if no is in the range "A"-"F"

cmp al,"a" ; "a" = 61h or 97d

jb error ; jump if below "a" to error

cmp al,"f"

jbe sub57 ; subtract 57h if no is in the range "a"-"f"

error: print emsg,emsg\_len ; "You entered Invalid Data!!!"

exit

sub57: sub al,20h ; subtract 57h if no is in the range "a"-"f"

sub37: sub al,07h ; subtract 37h if no is in the range "a"-"f"

sub30: sub al,30h ; subtract 30h if no is in the range "0"-"9"

add bx,ax ; prepare number

inc rsi ; point to next digit

loop next\_digit

ret

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

display\_16:

mov rsi,char\_ans+3 ; load last byte address of char\_ans in rsi

mov rcx,4 ; number of digits

cnt: mov rdx,0 ; make rdx=0 (as in div instruction rdx:rax/rbx)

mov rbx,16 ; divisor=16 for hex

div rbx

cmp dl, 09h ; check for remainder in RDX

jbe add30

add dl, 07h

add30:

add dl,30h ; calculate ASCII code

mov [rsi],dl ; store it in buffer

dec rsi ; point to one byte back

dec rcx ; decrement count

jnz cnt ; if not zero repeat

print char\_ans,4 ; display result on screen

ret

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

**Output:**

atharva@atharva:~$ gedit lab6.asm

atharva@atharva:~$ nasm -f elf64 lab6.asm

atharva@atharva:~$ ld -o lab6 lab6.o

atharva@atharva:~$ ./lab6

-----------Menu----------

1. Hex to BCD

2. BCD to Hex

3. Exit

Enter your choice: 1

Hex to BCD

Enter 4-digit Hex number: 8A9F

Equivalent BCD number is: 35487

-----------Menu----------

1. Hex to BCD

2. BCD to Hex

3. Exit

Enter your choice: 2

BCD to Hex

enter 5-digit BCD number: 35487

Equivalent BCD number is: 8A9F

-----------Menu----------

1. Hex to BCD

2. BCD to Hex

3. Exit

Enter your choice: 3

**Practical – 7**

**Problem Statement:** Write X86/64 ALP to detect protected mode and display the values of DTR, LDTR, IDTR, TR and MSW Registers also identify CPU type using CPUID instruction.

**Program:**

section .data

nline db 10,10

nline\_len: equ $-nline

colon db ":"

rmsg db 10,'Processor is in Real Mode...'

rmsg\_len: equ $-rmsg

pmsg db 10,'Processor is in Protected Mode...'

pmsg\_len: equ $-pmsg

gmsg db 10,"GDTR (Global Descriptor Table Register)

: "

gmsg\_len: equ $-gmsg

imsg db 10,"IDTR (Interrupt Descriptor Table Register)

: "

imsg\_len: equ $-imsg

lmsg db 10,"LDTR (Local Descriptor Table Register) : "

lmsg\_len: equ $-lmsg

tmsg db 10,"TR (Task Register) : "

tmsg\_len: equ $-tmsg

mmsg db 10,"MSW (Machine Status Word) : "

mmsg\_len: equ $-mmsg

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

Section .bss

GDTR resw 3 ; 48 bits, so 3 words

IDTR resw 3

LDTR resw 1 ; 16 bits, so 1 word

TR resw 1

MSW resw 1

char\_sum resb 4 ; 16-bits, so 4 digits

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

;you can change the macros as per 64-bit convensions

%macro print 2

mov rax, 1

mov rdi, 1

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro exit 0

mov rax, 60

mov rdi, 0

syscall

%endmacro

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

; If U ARE MODIFYING 32-BIT PROGRAM then

; Check line by line and make all 'e' as 'r' and other modifications

; for 64-bit numbers

section .text

global \_start

\_start:

SMSW [MSW]

mov rax,[MSW]

ror rax,1 ; Check PE bit, if 1=Protected

Mode, else Real Mode

jc p\_mode

print rmsg,rmsg\_len

jmp next

p\_mode:

print pmsg,pmsg\_len

next:

SGDT [GDTR]

SIDT [IDTR]

SLDT [LDTR]

STR [TR]

; SMSW [MSW]

print gmsg, gmsg\_len ;GDTR (Global Descriptor

Table Register)

; LITTLE ENDIAN SO

TAKE LAST WORD FIRST

mov ax,[GDTR+4] ; load value of GDTR[4,5] in ax

call disp16\_proc ; display GDTR contents

mov ax,[GDTR+2] ; load value of GDTR[2,3] in ax

call disp16\_proc ; display GDTR contents

print colon,1

mov ax,[GDTR+0] ; load value of GDTR[0,1] in ax

call disp16\_proc ; display GDTR contents

print imsg, imsg\_len ;IDTR (Interrupt Descriptor Table

Register)

mov ax,[IDTR+4]

call disp16\_proc

mov ax,[IDTR+2]

call disp16\_proc

print colon,1

mov ax,[IDTR+0]

call disp16\_proc

print lmsg, lmsg\_len ;LDTR (Local Descriptor Table

Register)

mov ax,[LDTR]

call disp16\_proc

print tmsg, tmsg\_len ;TR (Task Register)

mov ax,[TR]

call disp16\_proc

print mmsg, mmsg\_len ;MSW (Machine Status

Word)

mov ax,[MSW]

call disp16\_proc

print nline, nline\_len

exit

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

disp16\_proc:

mov rsi,char\_sum+3 ; load last byte address of char\_sum

buffer in rsi

mov rcx,4 ; number of digits

cnt: mov rdx,0 ; make rdx=0 (as in div instruction

rdx:rax/rbx)

mov rbx,16 ; divisor=16 for hex

div rbx

cmp dl, 09h ; check for remainder in RDX

jbe add30

add dl, 07h

add30:

add dl,30h ; calculate ASCII code

mov [rsi],dl ; store it in buffer

dec rsi ; point to one byte back

dec rcx ; decrement count

jnz cnt ; if not zero repeat

print char\_sum,4 ; display result on screen

ret

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

**Output:**

atharva@atharva:~$ gedit lab7.asm

atharva@atharva:~$ nasm -f elf64 lab7.asm

atharva@atharva:~$ ld -o lab7 lab7.o

atharva@atharva:~$ ./lab7

Processor is in Protected Mode...

GDTR (Global Descriptor Table Register) : 00082000:007F

IDTR (Interrupt Descriptor Table Register) : 00000000:0FFF

LDTR (Local Descriptor Table Register) : 0000

TR (Task Register) : 0040

MSW (Machine Status Word) : 0033

**Practical – 8**

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

**Program:**

section .data

menumsg db 10,10,'##### 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

wrchmsg db 10,10,'Wrong Choice Entered....Please try again!!!',10,10

wrchmsg\_len equ $-wrchmsg

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 times 5 db 0

cnt equ 05

spacechar db 20h

lfmsg db 10,10

section .bss

optionbuff resb 02

dispbuff resb 02

%macro dispmsg 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

section .text

global \_start

\_start:

dispmsg blk\_bfrmsg,blk\_bfrmsg\_len

call showblks

menu: dispmsg menumsg,menumsg\_len

accept optionbuff,02

cmp byte [optionbuff],'1'

jne case2

call blkxferwo\_proc

jmp exit1

case2: cmp byte [optionbuff],'2'

jne case3

call blkxferw\_proc

jmp exit1

case3: cmp byte [optionbuff],'3'

je exit

dispmsg wrchmsg,wrchmsg\_len

jmp menu

exit1:

dispmsg blk\_afrmsg,blk\_afrmsg\_len

call showblks

dispmsg lfmsg,2

exit:

mov rax , 60 ;Exit

mov rdi , 0

syscall

dispblk\_proc:

mov rcx,cnt

rdisp:

push rcx

mov bl,[esi] ;Read ASCII value char by char

push rsi

call disp8\_proc ;& Display

;Point to next char

dispmsg spacechar,1 ;Display space

pop rsi

pop rcx

inc esi

loop rdisp ;Decrement count

;Repeat display process till actual count becomes zero

ret

blkxferwo\_proc:

mov esi,srcblk

mov edi,dstblk

mov ecx,cnt

blkup1:

mov al,[esi]

mov [edi],al

inc esi

inc edi

loop blkup1

ret

blkxferw\_proc:

mov esi,srcblk

mov edi,dstblk

mov ecx,cnt

cld

rep movsb

ret

showblks:

dispmsg srcmsg,srcmsg\_len

mov esi,srcblk

call dispblk\_proc

dispmsg dstmsg,dstmsg\_len

mov esi,dstblk

call dispblk\_proc

ret

disp8\_proc:

mov ecx,02

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

dispmsg dispbuff,03

ret

**Output:**

atharva@atharva:~$ gedit lab8.asm

atharva@atharva:~$ nasm -f elf64 lab8.asm

atharva@atharva:~$ ld -o lab8 lab8.o

atharva@atharva:~$ ./lab8

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

atharva@atharva:~$ nasm -f elf64 ass8.asm

atharva@atharva:~$ ld -o ass8 ass8.o

atharva@atharva:~$ ./ass8

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

2

Block contents after transfer

Source block contents::01 02 03 04 05

Destination block contents::01 02 03 04 05

**Practical – 9**

**Problem Statement:** Write X86/64 ALP to perform overlapped block transfer with string specific instructions Block containing data can be defined in the data segment.

**Program:**

section .data

menumsg db 10,10,'##### Menu for 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

wrchmsg db 10,10,'Wrong Choice Entered....Please try again!!!',10,10

wrchmsg\_len equ $-wrchmsg

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 times 5 db 0

cnt equ 05

spacechar db 20h

lfmsg db 10,10

section .bss

optionbuff resb 02

dispbuff resb 02

%macro dispmsg 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

section .text

global \_start

\_start:

dispmsg blk\_bfrmsg,blk\_bfrmsg\_len

call showblks

menu: dispmsg menumsg,menumsg\_len

accept optionbuff,02

cmp byte [optionbuff],'1'

jne case2

call blkxferwo\_proc

jmp exit1

case2: cmp byte [optionbuff],'2'

jne case3

call blkxferw\_proc

jmp exit1

case3: cmp byte [optionbuff],'3'

je exit

dispmsg wrchmsg,wrchmsg\_len

jmp menu

exit1:

dispmsg blk\_afrmsg,blk\_afrmsg\_len

call showblks

dispmsg lfmsg,2

exit:

mov rax , 60 ;Exit

mov rdi , 0

syscall

dispblk\_proc:

mov rcx,cnt

rdisp:

push rcx

mov bl,[esi] ;Read ASCII value char by char

push rsi

call disp8\_proc ;& Display

;Point to next char

dispmsg spacechar,1 ;Display space

pop rsi

pop rcx

inc esi

loop rdisp ;Decrement count

;Repeat display process till actual count becomes zero

ret

blkxferwo\_proc:

mov esi,srcblk+3

mov edi,dstblk

mov ecx,cnt

blkup1:

mov al,[rsi]

mov [rdi],al

inc rsi

inc rdi

loop blkup1

ret

blkxferw\_proc:

mov rsi,srcblk+3

mov rdi,dstblk

mov rcx,cnt

cld

rep movsb

ret

showblks:

dispmsg srcmsg,srcmsg\_len

mov esi,srcblk

call dispblk\_proc

dispmsg dstmsg,dstmsg\_len

mov esi,dstblk

call dispblk\_proc

ret

disp8\_proc:

mov ecx,02

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

dispmsg dispbuff,03

ret

**Output:**

atharva@atharva:~$ gedit lab9.asm

atharva@atharva:~$ nasm -f elf64 lab9.asm

atharva@atharva:~$ ld -o lab9 lab9.o

atharva@atharva:~$ ./lab9

Block contents before transfer

Source block contents::01 02 03 04 05

Destination block contents::00 00 00 00 00

##### Menu for 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::04 05 04 05 04

atharva@atharva:~$ nasm -f elf64 lab9.asm

atharva@atharva:~$ ld -o lab9 lab9.o

atharva@atharva:~$ ./lab9

Block contents before transfer

Source block contents::01 02 03 04 05

Destination block contents::00 00 00 00 00

##### Menu for 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::04 05 04 05 04

**Practical - 10**

**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).

**Program:**

%macro dispmsg 2

mov rax,1

mov rdi,1

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 .data

msg db 10,'Enter two digit Number::'

msg\_len equ $-msg

res db 10,'Multiplication of elements is::'

res\_len equ $-res

choice db 10,13,'Enter your Choice:'

db 10,13,'1.Successive Addition'

db 10,13,'2.Add and Shift method'

db 10,13,'3.Exit',10

choice\_len equ $-choice

section .bss

num resb 03

num1 resb 01

result resb 04

cho resb 2

section .text

global \_start

\_start:

mov rax,00

mov rbx,00

mov rcx,00

mov rdx,00

mov byte[result],0

mov byte[num],0

mov byte[num1],0

dispmsg choice,choice\_len

accept cho,2

cmp byte[cho],31h

je a

cmp byte[cho],32h

je b

jmp exit

a:call Succe\_addition

jmp \_start

b:call Add\_shift

jmp \_start

exit:

mov rax,60

mov rdi,0

syscall

convert:

mov rbx,00

mov rcx,00

mov rax,00

mov rcx,02

mov rsi,num

up1:

rol bl,04

mov al,[rsi]

cmp al,39h

jg p1

sub al,30h

jmp p2

p1: sub al,37h

p2:add bl,al

inc rsi

loop up1

ret

display:

mov rcx,4

mov rdi,result

dup1:

rol bx,4

mov al,bl

and al,0fh

cmp al,09h

jg p3

add al,30h

jmp p4

p3: add al,37h

p4:mov [rdi],al

inc rdi

loop dup1

dispmsg result,4

ret

Succe\_addition:

dispmsg msg,msg\_len

accept num,3

call convert

mov [num1],bl

dispmsg msg,msg\_len

accept num,3

call convert

mov rcx,00

mov rax,00

mov rax,[num1]

repet:

add rcx,rax

dec bl

jnz repet

mov [result],rcx

dispmsg res,res\_len

mov rbx,[result]

call display

ret

Add\_shift:

dispmsg msg,msg\_len

accept num,3

call convert

mov [num1],bl

dispmsg msg,msg\_len

accept num,3

call convert

mov [num],bl

mov rbx,00

mov rcx,00

mov rdx,00

mov rax,00

mov dl,08

mov al,[num1]

mov bl,[num]

p11:shr bx,01

jnc p

add cx,ax

p:shl ax,01

dec dl

jnz p11

mov [result],rcx

dispmsg res,res\_len

mov rbx,[result]

call display

ret

**Output:**

atharva@atharva:~$ gedit lab10.asm

atharva@atharva:~$ nasm -f elf64 lab10.asm

atharva@atharva:~$ ld -o lab10 lab10.o

atharva@atharva:~$ ./lab10

Enter your Choice:

1.Successive Addition

2.Add and Shift method

3.Exit

1

Enter two digit Number::40

Enter two digit Number::12

Multiplication of elements is::0480

Enter your Choice:

1.Successive Addition

2.Add and Shift method

3.Exit

2

Enter two digit Number::10

Enter two digit Number::10

Multiplication of elements is::0100

Enter your Choice:

1.Successive Addition

2.Add and Shift method

3.Exit

3