Practical 1:

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
msg1 db 10,13, "enter 5 64 bit numbers"
len1 equ $-msg1
msg2 db 10,13, "Enter 5 64 bit numbers"
len2 equ $-msg2

section .bss
array resd 200
counter resb 1

section .text
global _start
_start:
;display
mov eax,1
mov edi,1
mov esi, msg1
mov edx, len1
syscall
;accept
mov byte [counter],05
mov ebx,00
loop1:
mov eax,0
mov edi,0
```

```
 mov esi, array
add esi,ebx
mov edx,17
syscall
add ebx,17
                    dec byte [counter]
JNZ loop1
;display
mov eax,1
mov edi,1
mov esi, msg2
mov edx, len2
syscall
;display
mov byte [counter],05
mov ebx,00
loop2:
mov eax,1
mov edi,1
mov esi, array
add esi,ebx
mov edx,17
syscall
add ebx,17
          dec byte [counter]
JNZ loop2
;exit system call
mov eax,60
mov edi,0
```

syscall

Practical 2:

Write an X86/64 ALP to accept a string and to display its length.

Program:

```
section .data
        msg1 db 10,13,"Enter a string:"
        len1 equ $-msg1

section .bss
        str1 resb 200
        result resb 16

section .text
        global _start
        _start:
;display
        mov eax,1
        mov edi,1
        mov esi, msg1
        mov edx, len1
        syscall

;store string
        mov eax, 0
        mov edi,0
        mov esi,str1
        mov edx,200
        syscall
```

```
call display

;exit system call
        mov eax,60
        mov edi,0
        syscall

%macro dispmsg 2
        mov eax,1
        mov edi,1
        mov esi, %1
        mov edx, %2
        syscall
%endmacro

display:
        mov ebx,eax
        mov edi,result
        mov cx,16

        up1:
                rol ebx,04
                mov al,bl
                and al,0fh
                cmp al,09h
                jg add_37
                add al,30h
                jmp skip
        add_37:
                add al,37h
        skip:
                mov [edi],al
                inc edi
                dec cx
                jnz up1
                dispmsg result,08
ret
```

Practical 3:

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

Program:

```
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
repeat: cmp al,[rsi]
        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
        mov rdi, result
        mov cx,02
up1:
        rol bl,04
        mov al,bl
        and al,0fh
        cmp al,09h
        jg add_37
        add al,30h
        jmp skip1
add_37: add al,37h
skip1:  mov [rdi],al
        inc rdi
        dec cx
        jnz up1
        write result , 4
        ret
```

-

Practical 4:

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

Program:

```
section .data
welmsg db 10,'Welcome to count positive and negative 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,87h,88h,8a9fh,0adh,02h,8507h
arrcnt equ 8
pcnt db 0
ncnt db 0

section .bss
dispbuff resb 2

%macro print 2
mov eax, 4
mov ebx, 1
mov ecx, %1
mov edx, %2
```

```
int 80h
%endmacro

section .text
global _start
_start:
print welmsg,welmsg_len
mov esi,array
mov ecx,arrcnt

up1:
bt word[esi],15
jnc pnxt
inc byte[ncnt]
jmp pskip
pnxt: inc byte[pcnt]
pskip: inc esi
inc esi
loop up1

print pmsg,pmsg_len
mov bl,[pcnt]
call disp8num
print nmsg,nmsg_len
mov bl,[ncnt]
call disp8num

print nwline,1
```

```
exit:

mov eax,01
mov ebx,0
int 80h

disp8num:
mov ecx,2
mov edi,dispbuff

dup1:
rol bl,4
mov al,bl
and al,0fh
cmp al,09
jbe dskip
dskip:
add al,30h
mov [edi],al
inc edi

loop dup1
print dispbuff,2
ret
```

Practical 5:

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
msg1 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,'Enter Choice:'
msg1length equ $-msg1
msg2 db 10,10,'Enter 4 digit hex number::'
msg2length equ $-msg2
msg3 db 10,10,'BCD Equivalent:'
msg3length equ $-msg3
msg4 db 10,10,'Enter 5 digit BCD number::'
msg4length equ $-msg4
msg5 db 10,10,'Wrong Choice Entered....Please try again!!!',10,10
msg5length equ $-msg5
msg6 db 10,10,'Hex Equivalent::'
msg6length equ $-msg6
cnt db 0
section .bss
arr resb 06
```

```
dispbuff  resb 08
ans resb 01
%macro disp 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:
disp msg1,msg1length
accept arr,2
cmp byte [arr],'1'
jne l1
call hex2bcd_proc
jmp menu
l1: cmp byte [arr],'2'
jne l2
call bcd2hex_proc
```

```
jmp menu
l2: cmp byte [arr],'3'
je exit
disp msg5,msg5length
jmp menu
exit:
mov rax,60
mov rbx,0
syscall
hex2bcd_proc:
disp msg2,msg2length
accept arr,5
call conversion
mov rcx,0
mov ax,bx
mov bx,10
l33: mov dx,0
div bx
push rdx
inc rcx
inc byte[cnt]
cmp ax,0
jne l33
disp msg3,msg3length
l44: pop rdx
add dl,30h
mov [ans],dl
disp ans,1
dec byte[cnt]
```

```
jnz l44
ret
bcd2hex_proc:
disp msg4,msg4length
accept arr,6
disp msg6,msg6length
mov rsi,arr
mov rcx,05
mov rax,0
mov ebx,0ah
l55: mov rdx,0
mul ebx
mov dl,[rsi]
sub dl,30h
add rax,rdx
inc rsi
dec rcx
jnz l55
mov ebx,eax
call disp32_num

ret
conversion:
mov bx,0
mov ecx,04
mov esi,arr
up1:
rol bx,04
mov al,[esi]
```

```
cmp al,39h

jbe l22

sub al,07h

l22: sub al,30h

add bl,al

inc esi

loop up1

ret


disp32_num:

mov rdi,dispbuff

mov rcx,08

l77:

rol ebx,4

mov dl,bl

and dl,0fh

add dl,30h

cmp dl,39h

jbe l66

add dl,07h

l66:

mov [rdi],dl

inc rdi

dec rcx

jnz l77

disp dispbuff+3,5

ret
```

Practical 6:

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

Program:

```
section .data
rmodemsg db 10,"Processor is in Real Mode."
rmsg_len equ $-rmodemsg
pmodemsg db 10,"Processor is in Protected Mode.",10
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 Contents are :- "
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 disp 2
mov eax,01
mov edi,01
mov esi,%1
mov edx,%2
syscall
%endmacro

section .text
global _start
_start:
smsw eax
mov [cr0_data],eax
bt eax,1
jc prmode
disp rmodemsg,rmsg_len

jmp nxt1
prmode:disp pmodemsg,pmsg_len
nxt1:sgdt  [gdt]
sldt [ldt]
sidt [idt]
str [tr]
disp gdtmsg,gmsg_len
```

```
mov bx,[gdt+4]

call disp_num

mov bx,[gdt+2]

call disp_num

disp colmsg,1

mov bx,[gdt]

call disp_num

disp ldtmsg,lmsg_len

mov bx,[ldt]

call disp_num

disp idtmsg,imsg_len

mov bx,[idt+4]

call disp_num

mov bx,[idt+2]

call disp_num

disp colmsg,1

mov bx,[idt]

call disp_num

disp trmsg,tmsg_len

mov bx,[tr]

call disp_num

disp mswmsg,mmsg_len

mov bx,[cr0_data+2]

call disp_num

mov bx,[cr0_data]

call disp_num

disp nwline,1

exit:mov eax,60

xor edi,edi
```

```
syscall

disp_num:
mov esi,dnum_buff
mov ecx,04
up1:rol bx,4
mov dl,bl
and dl,0fh
add dl,30h
cmp dl,39h
jbe skip1
add dl,07h
skip1:mov [esi],dl
inc esi
loop up1

disp dnum_buff,4
ret
```

Practical 7:

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

walmag db "Welcome:",10

walmsglen equ $-walmag

msg0 db "Overlaped and non overlaped block transfer:",10

msg0len equ $-msg0


srcblk db 10,"Source block is: 01h 02h 03h 04h 05h "

srcblklen equ $-srcblk

dstblk db 10,"Destanition after transfer is:"

dstblklen equ $-dstblk

nw db 10

nwlen equ $-nw

spacebar db 20h

msg db 10,"*************************menu*******************************"

msglen equ $-msg

msg1 db 10,"1. Non overlaped block transfer without string instruction:"

msg1len equ $-msg1

msg2 db 10,"2. Non overlaped block transfer with string instruction:"

msg2len equ $-msg2

msg3 db 10,"3. Exit:"

msg3len equ $-msg3

msg4 db 10,"Enter your choice:"

msg4len equ $-msg4

array db 01h,02h,03h,04h,05h

cnt equ 5
```

```
newarray times 5 db 0
section .bss
dispbuff  resb 2

choice resb 2
dest resb 8
%macro read 2
mov rax,0
mov rdi,0
mov rsi,%1
mov rdx,%2
syscall
%endmacro
%macro print 2
mov rax,1
mov rdi,1
mov rsi,%1
mov rdx,%2
syscall
%endmacro
%macro exit 0
exit:mov rax,60
xor rdi,rdi
syscall
%endmacro

section .text
global _start
_start:
```

```
print walmag, walmsglen

print msg0 ,msg0len

menu:print msg,msglen

print msg1 ,msg1len

print msg2 ,msg2len

print msg3 ,msg3len

print msg4 ,msg4len

read choice,2

cmp byte[choice],'1'

je case1

cmp byte[choice],'2'

je case2

cmp byte[choice],'3'

je exit1

case1:call wo

jmp menu

case2:call with

jmp menu

wo:mov esi,array

mov edi,dest

mov ecx,cnt

again:mov al,[esi]

mov [edi],al

inc esi

inc edi

loop again

print srcblk,srcblklen

print dstblk,dstblklen

mov rdi,dest
mov rcx,cnt
nxtnun:push  rcx
mov bl,[rdi]
push rdi
call disp8
pop rdi
push rdi
print spacebar,1
pop rdi
inc rdi
pop rcx
loop nxtnun
jmp menu
ret
with:mov esi,array
mov edi,dest
mov ecx,cnt
cld
rep movsb
print srcblk,srcblklen
print dstblk,dstblklen
mov rdi,dest
mov rcx,cnt
nxtnum1:push  rcx
mov bl,[rdi]
push rdi
call disp8
pop rdi
push rdi
```

```
print spacebar,1
pop rdi
inc rdi
pop rcx
loop nxtnum1
jmp menu
ret
exit1:mov rax,60
xor rdi,rdi
syscall
disp8:
mov rcx,02
mov rsi,dispbuff
back1:rol bl,04
mov al,bl
and al,0Fh
cmp al,09
jbe add30
add al,07h
add30:add al,30h
mov [rsi],al
inc rsi
loop back1
print dispbuff,2
ret
```

Practical 8:

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
walmag db "Welcome:",10
walmsglen equ $-walmag
msg0 db "Overlaped and non overlaped block transfer:",10
msg0len equ $-msg0
srcblk db 10,"Source block is: 01h 02h 03h 04h 05h "
srcblklen equ $-srcblk
dstblk db 10,"Destanition after transfer is:"
dstblklen equ $-dstblk
nw db 10
nwlen equ $-nw
spacebar db 20h
msg db 10,"*************************menu******************************"
msglen equ $-msg
msg1 db 10,"1. overlaped block transfer without string instruction:"
msg1len equ $-msg1
msg2 db 10,"2. overlaped block transfer with string instruction:"
msg2len equ $-msg2
msg3 db 10,"3. Exit:"
msg3len equ $-msg3
msg4 db 10,"Enter your choice:"
msg4len equ $-msg4
array db 01h,02h,03h,04h,05h
cnt equ 5
```

```
newarray times 5 db 0

section .bss
dispbuff  resb 2
choice resb 2
dest resb 8
%macro read 2
mov eax,0
mov edi,0
mov esi,%1
mov edx,%2
syscall
%endmacro
%macro print 2
mov rax,1
mov rdi,1
mov rsi,%1
mov rdx,%2
syscall
%endmacro
%macro exit 0
exit:mov rax,60 ;
xor rdi,rdi
syscall
%endmacro
section .text

global _start
_start:
```

```
print walmag, walmsglen

print msg0 ,msg0len

menu: print msg,msglen

print msg1 ,msg1len

print msg2 ,msg2len

print msg3 ,msg3len

print msg4 ,msg4len

read choice,2

cmp byte[choice],'1'

je case1

cmp byte[choice],'2'

je case2

cmp byte[choice],'3'

je exit1

case1:call wo

jmp menu

case2:call with

jmp menu

wo:mov esi,array

mov edi,array+2

mov ecx,cnt

again:mov al,[esi]

mov [edi],al

inc esi

inc edi

loop again

print srcblk,srcblklen

print dstblk,dstblklen

mov rdi,array
mov rcx,cnt
nxtnun:push  rcx
mov bl,[rdi]
push rdi
call disp8
pop rdi
push rdi
print spacebar,1
pop rdi
inc rdi
pop rcx
loop nxtnun
jmp menu
ret
with:mov esi,array
mov edi,dest
mov ecx,cnt
cld
rep movsb
print srcblk,srcblklen
print dstblk,dstblklen
mov rdi,dest
mov rcx,cnt
nxtnum1:push  rcx
mov bl,[rdi]
push rdi
call disp8
pop rdi
push rdi
```

```
print spacebar,1
pop rdi
inc rdi

pop rcx
loop nxtnum1
jmp menu
ret
exit1:mov rax,60
xor rdi,rdi
syscall
disp8:
mov rcx,02
mov rsi,dispbuff
back1:rol bl,04
mov al,bl
and al,0Fh
cmp al,09
jbe add30
add al,07h
add30:add al,30h
mov [rsi],al
inc rsi
loop back1
print dispbuff,2
ret
```

Practical 9:

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 for multiplication of two 8 bit numbers by successive addition:

```
section .data
msg1 db "enter first 8 bit hex no:",10
len1: equ $-msg1
msg2 db "enter second 8 bit hex no:",10
len2: equ $-msg2
msg3 db "Multiplication of two hex no is:",10
len3: equ $-msg3

section .bss

arr1 resb 3
arr2 resb 3
arr3 resb 4

%macro disp 2
mov eax,01h
mov edi,01h
mov esi,%1
mov edx,%2
syscall
%endmacro

%macro inn 2
```

```
mov eax,00h

mov edi,00h

mov esi,%1

mov edx,%2

syscall

%endmacro


section .text

global _start

_start:


disp msg1,len1

inn arr1,03


disp msg2,len2

inn arr2,03


mov esi,arr1

mov cl,04

xor bx,bx

mov ch,02

up:

cmp byte[esi],39h

jng sk

sub byte[esi],07h

sk:

sub byte[esi],30h

shl bx,cl

add bl,[esi]
```

```
inc esi
dec ch
jnz up

xor dx,dx
mov esi,arr2
mov cl,04
mov ch,02
up1:
cmp byte[esi],39h
jng sk1
sub byte[esi],07h
sk1:
sub byte[esi],30h
shl dx,cl
add dl,[esi]

inc esi
dec ch
jnz up1

xor ax,ax
xor cl,cl
cmp dl,bl
jng sph
mov cl,bl
mov bl,dl
jmp outt
sph:
```

```
mov cl,dl
outt:
add ax,bx
dec cl
jnz outt


mov esi,arr3
mov ch,04
mov cl,04
again1:
rol ax,cl
mov bl,al
and bl,0fh
cmp bl,09h
jng skip2
add bl,07h
skip2:
add bl,30h
mov [esi],bl
inc esi
dec ch
jnz again1

disp arr3,04

mov eax,3ch
mov edi,00
syscall
```

Program for multiplication of two 8 bit numbers using add and shift method:

```
section .data
msg1 db "enter first 8 bit hex no:",10
len1: equ $-msg1
msg2 db "enter second 8 bit hex no:",10
len2: equ $-msg2
msg3 db "Multiplication of two hex no is:",10
len3: equ $-msg3

section .bss
arr1 resb 3
arr2 resb 3
arr3 resb 4
```

```
%macro disp 2
mov eax,01h
mov edi,01h
mov esi,%1
mov edx,%2
syscall
%endmacro

%macro inn 2
mov eax,00h
mov edi,00h
mov esi,%1
mov edx,%2
syscall
%endmacro

section .text
global _start
_start:

disp msg1,len1
inn arr1,03

disp msg2,len2
inn arr2,03

mov esi,arr1
mov cl,04
xor bx,bx
```

```
mov ch,02
up:
cmp byte[esi],39h
jng sk
sub byte[esi],07h
sk:
sub byte[esi],30h
shl bx,cl
add bl,[esi]
inc esi
dec ch
jnz up

xor dx,dx
mov esi,arr2
mov cl,04
mov ch,02
up1:
cmp byte[esi],39h
jng sk1
sub byte[esi],07h
sk1:
sub byte[esi],30h
shl dx,cl
add dl,[esi]

inc esi
dec ch
jnz up1
```

```
xor ax,ax

mov cl,00h

mov ch,08h

again:

shr dl,01

jnc xx

shl bx,cl

add ax,bx

shr bx,cl

xx:

inc cl

dec ch

jnz again


mov esi,arr3

mov ch,04

mov cl,04

again1:

rol ax,cl

mov bl,al

and bl,0fh

cmp bl,09h

jng skip2

add bl,07h

skip2:

add bl,30h

mov [esi],bl

inc esi
```

```
dec ch

jnz again1


disp arr3,04


mov eax,3ch

mov edi,00

syscall
```