Experiment no1: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.

section .data

msg1 db 10,13,"Enter 5 64 bit numbers"

len1 equ $-msg1

msg2 db 10,13,"Entered 5 64 bit numbers"

len2 equ $-msg2

section .bss

array resd 200

counter resb 1

section .text

global \_start

\_start:

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 ; 0 for read

mov rdi,0 ; 0 for keyboard

mov rsi, array ;move pointer to start of array

add rsi,rbx

mov rdx,17

syscall

add rbx,17 ;to move counter

dec byte[counter]

JNZ loop1;

mov Rax,1

mov Rdi,1

mov Rsi,msg2

mov Rdx,len2

syscall;

mov byte[counter],05

mov rbx,00

loop2:

mov rax,1 ;1 for write

mov rdi, 1 ;1 for monitor

mov rsi, array

add rsi,rbx

mov rdx,17

syscall;

add rbx,17

dec byte[counter]

JNZ loop2;

exit:

syscall;

mov rax ,60

mov rdi,0

syscall ;

Output:

(base) kse@kse-Veriton-Series:~$ cd Desktop/

(base) kse@kse-Veriton-Series:~/Desktop$ nasm -f elf64 exp1.asm

(base) kse@kse-Veriton-Series:~/Desktop$ ld -oapp exp1.o

(base) kse@kse-Veriton-Series:~/Desktop$ ./app

Enter 5 64 bit numbers11111

22222

33333

44444

55555

Entered 5 64 bit numbers11111

22222

33333

44444

55555

Experiment no 2:Write an X86/64 ALP to accept a string and to display its length.

%macro IO 4 ; simple macro ,

mov rax,%1 ; param 1 -> system call number

mov rdi,%2 ; param 2 -> file descriptor

mov rsi,%3 ; param 3 -> message

mov rdx,%4 ; param 4 -> length

syscall

%endmacro

section .data

m1 db "enter string",10 ;10 ->line feed

l1 equ $-m1

m2 db "Entered",10

l2 equ $-m2

m3 db "Length is ",10

l3 equ $-m3

m4 db "Write an X86/64 ALP to accept a string and to display its length" ,10

l4 equ $-m4

m5 db "Exiting now" ,10

l5 equ $-m5

m6 db "rahul ghosh 3236" ,10

l6 equ $-m6

m7 db 10

section .bss

string resb 50 ;string array of size 50

strl equ $-string

count resb 1

\_output resb 20

section .text

global \_start

\_start:

IO 1,1,m6,l6 ; display

IO 1,1,m4,l4 ; display

IO 1,1,m1,l1 ; display

input:

IO 0,0,string,strl

mov [count],rax ;count now points to rax

output:

IO 1,1,m2,l2 ; display

IO 1,1,string,strl

IO 1,1,m3,l3 ; display

mov rax,[count] ; value of count passed into rax

call hex\_to\_dec

IO 1,1,\_output,16

IO 1,1,m7,1

exit:

IO 1,1,m5,l5

mov rax, 60 ; system call for exit

mov rdi, 0 ; exit code 0

syscall

hex\_to\_dec:

mov rsi,\_output+15 ; max size of display , for convinience set to 16 and rsipoints to output[16]

mov rcx,2 ; loop count , or number of digits displayed , 2 digits (unlikely we will enter string > 99)(prints preceding zeros otherwise)

letter2:

xor rdx,rdx ; setting rdx to null without setting a null byte (a tip i saw on reddit) needed to clean dl for use

mov rbx,10 ; conversion base

div rbx ; dividing by conversion base

cmp dl,09h ; comparing 09h with dl , since the division remainder ends up in dx and since its one digits its in dl

jbe add30 ; if value under in 0-9 , we directly add 30h to get ascii 0-9

add30:

add dl,30h ; adding 30h

mov [rsi],dl ; moving the ascii we generated to rsi

dec rsi ; rsi now points to the next place in display or output[n-1]

dec rcx ; loop

jnz letter2

ret

output:

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

enter string microprocessor

Entered

microprocessorLength is

14

Experiment no 3: Write an X86/64 ALP to find the largest of given Byte/Word/Dword/64-bit numbers.

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

Experiment no 4: Write a switch case driven X86/64 ALP to perform 64-bit hexadecimal arithmetic operations (+,-,\*, /) using suitable macros. Define procedure for each operation.

%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

div\_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

IO 1,1,m\_rem,m\_rem\_l

pop rdx

mov rbx,rdx

call hex\_to\_ascii; remainder output

IO 1,1,mspace,1

IO 1,1,m\_qou,m\_qou\_l

pop rax

mov rbx,rax

call hex\_to\_ascii; quotient output

ret

ascii\_to\_hex:

mov rsi, \_n1

mov rcx, 16

xor rbx, 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

ret

output:

addition here

0000000000000004

0000000000000003

result is 0000000000000007

subtraction here

0000000000000004

0000000000000003

result is 0000000000000001

multiplication

0000000000000004

0000000000000003

result is 000000000000000C

division here

0000000000000004

0000000000000003

remainder is 0000000000000001

qoutient is 0000000000000001

Experiment no 5: Count number of positive and negative number from array.

section .data

msg1 db "Count of Positive numbers:"

len1 equ $-msg1

msg2 db "Count of negative numbers:"

len2 equ $-msg2

array db 10,12,-21,-12,-19,-34,41

%macro print 2

mov rax,01

mov rdi,01

mov rsi,%1

mov rdx,%2

syscall

%endmacro

section .bss

count resb 2

pcount resb 2

ncount resb 2

totalcount resb 2

section .text

global \_start

\_start:

mov byte[count],07

mov byte[pcount],00

mov byte[ncount],00

mov rsi,array

Up:

mov al,00

add al,[rsi]

js neg

inc byte[pcount]

jmp Down

neg:

inc byte[ncount]

Down:

add rsi,01

dec byte[count]

jnz Up

mov bl,[pcount]

mov dl,[ncount]

b1:

print msg1,len1

mov bh,[pcount]

call disp

print msg2,len2

mov bh,[ncount]

call disp

mov rax,60

mov rdi,00

syscall

disp:

mov byte[count],02

loop:

rol bh,04

mov al,bh

AND al,0FH

cmp al,09

jbe l1

add al,07h

l1:add al,30h

mov[totalcount],al

print totalcount,02

dec byte[count]

jnz loop

ret

output:

kse@kse-Veriton-Series:~$ cd Desktop/

kse@kse-Veriton-Series:~/Desktop$ nasm -f elf64 exp5.asm

kse@kse-Veriton-Series:~/Desktop$ ld -oapp exp5.o

kse@kse-Veriton-Series:~/Desktop$ ./app

Count of Positive numbers:03Count of negative numbers:04

Experiment no 6: Hex to BCD &BCD to Hex Conversion.

%macro print 2

mov rax,1

mov rdi,0

mov rsi,%1

mov rdx,%2

syscall

%endmacro

%macro read 2

mov rax,0

mov rdi,1

mov rsi,%1

mov rdx,%2

syscall

%endmacro

section .data

menu: db "=====================================",10

db " MENU ",10

db "=====================================",10

db "Enter your choice",10

db "1. Hex to BCD",10

db "2.BCD to hex",10

db "3.Exit",10

menulen: equ $-menu

msg1: db "Enter the number",10

len1: equ $-msg1

blank: db "",10

blen: equ $-blank

section .bss

choice resb 2

hexcode resb 5

bcdcode resb 5

count resb 1

ascdigit resb 1

asciicode resb 4

section .text

global \_start:

\_start:

print menu,menulen

read choice,2

cmp byte[choice],31h

je hod

cmp byte[choice],32h

je doh

cmp byte[choice],33h

je exit

jmp \_start ;jump to start if invalid choice is entered

hod: ;hex to bcd

print msg1,len1

read hexcode,5

mov byte[count],4

call hextoascii ;value at dx is hexadecimal ie binary

mov byte[count],0

mov rax,0h

mov ax,dx

mov rdx,0h

mov rbx,0Ah

back:

div rbx

push dx

mov rdx,0

inc byte[count]

cmp ax,0h

jnz back

repeat:

pop dx

add dl,30h

mov byte[ascdigit],dl

print ascdigit,1

dec byte[count]

jnz repeat

print blank,blen

jmp \_start

doh: ;bcd to hex

print msg1,len1

read bcdcode,6

mov rbx,10

mov rdx,0

mov rax,0

mov byte[count],5

mov rsi,bcdcode

;converting to decimal

unpack:

mul rbx

sub byte[rsi],30h

movsx cx,byte[rsi] ;move with sign extended

add ax,cx

inc rsi

dec byte[count]

jnz unpack

;now we have the decimal value. Convert we have hex val

mov byte[count],4

mov rsi,asciicode

again:

rol ax,4

mov bl,al

and bl,0Fh

cmp bl,9

jbe nocorrection2

add bl,7

nocorrection2:

add bl,30h

mov byte[rsi],bl

inc rsi

dec byte[count]

jnz again

print asciicode,4

print blank,blen

jmp \_start

exit: ;exit system call

mov rax,60

mov rdx,0h

syscall

hextoascii:

mov ax,0h

mov dx,0h

mov rsi,hexcode

back2:

rol dx,4

mov al,byte[rsi]

sub al,30h

cmp al,9

jbe nocorrection

sub al,7h

nocorrection:

add dx,ax

add rsi,1

dec byte[count]

jnz back2

ret

**Output:**

=====================================

MENU

=====================================

Enter your choice

1. Hex to BCD

2.BCD to hex

3.Exit

1

Enter the number

23

12537

=====================================

MENU

=====================================

Enter your choice

1. Hex to BCD

2.BCD to hex

3.Exit

2

Enter the number

32

6C18

=====================================

MENU

=====================================

Enter your choice

1. Hex to BCD

2.BCD to hex

3.Exit

3

Experiment no 7: 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

msg1:db 'GDTR contents :',0xa

len1:equ $-msg1

msg2:db 'LDTR contents:',0xa

len2:equ $-msg2

msg3:db 'IDTR contents :',0xa

len3:equ $-msg3

msg4:db 'TR contents:',0xa

len4:equ $-msg4

msg5:db 'MSW contents:',0xa

len5:equ $-msg5

msg6:db 'We are in protected mode.!!',0xa

len6:equ $-msg6

msg7:db ' ',0xa

len7:equ $-msg7

msg8:db 'We are not in protected mode.!!',0xa

len8:equ $-msg8

msg9:db ' : ',0xa

len9:equ $-msg9

section .bss

gdt:resd 01

resw 01

ldt:resw 01

idt: resd 01

resw 01

tr:resw 01

msw:resw 01

result: resw 01

section .text

global \_start

\_start:

smsw [msw]

sgdt [gdt]

sldt [ldt]

sidt [idt]

str [tr]

mov ax,[msw]

bt ax,0

jc next

mov rax,1

mov rdi,1

mov rsi,msg8

mov rdx,len8

syscall

jmp exit

next:

mov rax,1

mov rdi,1

mov rsi,msg6

mov rdx,len6

syscall

;GDTR

mov rax,1

mov rdi,1

mov rsi,msg1

mov rdx,len1

syscall

mov bx,word[gdt+4]

call HtoA

mov bx,word[gdt+2]

call HtoA

mov rax,1

mov rdi,1

mov rsi,msg9

mov rdx,len9

syscall

mov bx,word[gdt]

call HtoA

;LDTR

mov rax,1

mov rdi,1

mov rsi,msg7

mov rdx,len7

syscall

mov rax,1

mov rdi,1

mov rsi,msg2

mov rdx,len2

syscall

mov bx,word[ldt]

call HtoA

;IDTR

mov rax,1

mov rdi,1

mov rsi,msg7

mov rdx,len7

syscall

mov rax,1

mov rdi,1

mov rsi,msg3

mov rdx,len3

syscall

mov bx,word[idt+4]

call HtoA

mov bx,word[idt+2]

call HtoA

mov rax,1

mov rdi,1

mov rsi,msg9

mov rdx,len9

syscall

mov bx,word[idt]

call HtoA

;TR

mov rax,1

mov rdi,1

mov rsi,msg7

mov rdx,len7

syscall

mov rax,1

mov rdi,1

mov rsi,msg4

mov rdx,len4

syscall

mov bx,word[tr]

call HtoA

;MSW

mov rax,1

mov rdi,1

mov rsi,msg7

mov rdx,len7

syscall

mov rax,1

mov rdi,1

mov rsi,msg5

mov rdx,len5

syscall

mov bx,word[msw]

call HtoA

;EXIT

exit:

mov rax,60

mov rdi,0

syscall

HtoA:

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

mov rax,1

mov rdi,1

mov rsi,result

mov rdx,4

syscall

ret

output:

We are in protected mode.!!

GDTR contents :

00001000 :

007F

LDTR contents:

0000

IDTR contents :

00000000 :

0FFF

TR contents:

0040

MSW contents:

FFFF

Experiment no 8: Non-overlapped and overlapped block transfer.

section .data

sourceBlock db 12h,45h,87h,24h,97h

count equ 05

msg db "ALP for non overlapped block transfer using string instructions : ",10

msg\_len equ $ - msg

msgSource db 10,"The source block contains the elements : ",10

msgSource\_len equ $ - msgSource

msgDest db 10,10,"The destination block contains the elements : ",10

msgDest\_len equ $ - msgDest

bef db 10, "Before Block Transfer : ",10

beflen equ $ - bef

aft db 10,10 ,"After Block Transfer : ",10

aftlen equ $ - aft

section .bss

destBlock resb 5

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 msg , msg\_len

write bef , beflen

write msgSource , msgSource\_len

mov rsi,sourceBlock

call dispBlock

write msgDest , msgDest\_len

mov rsi,destBlock

call dispBlock

mov rsi,sourceBlock

mov rdi,destBlock

mov rcx, count

cld

rep movsb

write aft , aftlen

write msgSource , msgSource\_len

mov rsi,sourceBlock

call dispBlock

write msgDest , msgDest\_len

mov rsi,destBlock

call dispBlock

mov rax,60

mov rdi,0

syscall

dispBlock:

mov rbp,count

next:mov al,[rsi]

push rsi

call disp

pop rsi

inc rsi

dec rbp

jnz next

ret

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 for non overlapped block transfer using string instructions :

Before Block Transfer :

The source block contains the elements :

1245872497

The destination block contains the elements :

0000000000

After Block Transfer :

The source block contains the elements :

1245872497

The destination block contains the elements :

1245872497

Experiment no 9: Multiplication of two 8 bit nos. using Successive addition and Shift and add method.

section .data

msg db 'Enter two digit Number::',0xa

msg\_len equ $-msg

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

res\_len equ $-res

choice db 'Enter your Choice:',0xa

db'1.Successive Addition',0xa

db '2.Add and Shift method',0xa

db '3.Exit',0xa

choice\_len equ $-choice

section .bss

num resb 03

num1 resb 01

result resb 04

cho resb 2

section .text

global \_start

\_start:

xor rax,rax

xor rbx,rbx

xor rcx,rcx

xor rdx,rdx

mov byte[result],0

mov byte[num],0

mov byte[num1],0

mov rax,1

mov rdi,1

mov rsi,choice

mov rdx,choice\_len

syscall

mov rax,0 ;; read choice

mov rdi,0

mov rsi,cho

mov rdx,2

syscall

cmp byte[cho],31h ;; comparing choice

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: ;; ASCII to Hex conversion

xor rbx,rbx

xor rcx,rcx

xor rax,rax

mov rcx,02

mov rsi,num

up1:

rol bl,04

mov al,[rsi]

cmp al,39h

jbe p1

sub al,07h

jmp p2

p1: sub al,30h

p2: add bl,al

inc rsi

loop up1

ret

display: ;; Hex to ASCII conversion

mov rcx,4

mov rdi,result

dup1:

rol bx,4

mov al,bl

and al,0fh

cmp al,09h

jbe p3

add al,07h

jmp p4

p3: add al,30h

p4:mov [rdi],al

inc rdi

loop dup1

mov rax,1

mov rdi,1

mov rsi,result

mov rdx,4

syscall

ret

Succe\_addition:

mov rax,1

mov rdi,1

mov rsi,msg

mov rdx,msg\_len

syscall

mov rax,0

mov rdi,0

mov rsi,num

mov rdx,3

syscall

call convert

mov [num1],bl

mov rax,1

mov rdi,1

mov rsi,msg

mov rdx,msg\_len

syscall

mov rax,0

mov rdi,0

mov rsi,num

mov rdx,3

syscall

call convert

xor rcx,rcx

xor rax,rax

mov rax,[num1]

repet:

add rcx,rax

dec bl

jnz repet

mov [result],rcx

mov rax,1

mov rdi,1

mov rsi,res

mov rdx,res\_len

syscall

mov rbx,[result]

call display

ret

Add\_shift:

mov rax,1

mov rdi,1

mov rsi,msg

mov rdx,msg\_len

syscall

mov rax,0

mov rdi,0

mov rsi,num

mov rdx,3

syscall

call convert

mov [num1],bl

mov rax,1

mov rdi,1

mov rsi,msg

mov rdx,msg\_len

syscall

mov rax,0

mov rdi,0

mov rsi,num

mov rdx,3

syscall

call convert

mov [num],bl

xor rbx,rbx

xor rcx,rcx

xor rdx,rdx

xor rax,rax

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

mov rax,1

mov rdi,1

mov rsi,res

mov rdx,res\_len

syscall

;dispmsg res,res\_len

mov rbx,[result]

call display

ret

;; output:

;;Enter your Choice:

;;1.Successive Addition

;;2.Add and Shift method

;;3.Exit

;;1

;;Enter two digit Number::

;;02

;;Enter two digit Number::

;;02

;;Multiplication of elements is::0004Enter your Choice:

;;1.Successive Addition

;;2.Add and Shift method

;;3.Exit

;;2

;;Enter two digit Number::

;;03

;;Enter two digit Number::

;;03

;;Multiplication of elements is::0009Enter your Choice:

;;1.Successive Addition

;;2.Add and Shift method

;;3.Exit

;;3

Experiment no 10: Write x86 ALP to find the factorial of a given integer number on a command line by using recursion.

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

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

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

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

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

output

\*\*\*Program to find Factorial of a number\*\*\*

Enter the number :6

Factorial is : 0000000000000006

Thank you

[Execution complete with exit code 0]