1,12) write a menu drivenALP for the following 64 bit arithmatic operarions and display the result accept the number from user.1)Addition 2)Division 3)Exit

```
; 64-bit Linux Assembly Program for Addition/Division
%macro READ 2
   mov rax, 0
   mov rdi, 0
   mov rsi, %1
   mov rdx, %2
   syscall
%endmacro

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

section .data
msg1 db "Enter first number: ",10
len1 equ $-msg1
msg2 db "Enter second number: ",10
len2 equ $-msg2
menu db "1. Addition",10,"2. Division",10,"3. Exit",10
menuLen equ $-menu
select db "Enter your choice: ",0
```

```
selectLen equ $-select
newline db 10

section .bss
a resq 1
b resq 1
char_buff resb 17
choice resb 2

section .text
global _start

_start:
menu_loop:
   WRITE menu, menuLen
   WRITE select, selectLen
   READ choice, 2
   mov al, [choice]
   cmp al, '1'
   je do_add
   cmp al, '2'
   je do_div
   cmp al, '3'
   je exit_prog
   jmp menu_loop

read_a:
   WRITE msg1, len1
```

```
    READ char_buff, 17
    dec rax
    mov rcx, rax
    call accept
    mov [a], rbx
    ret

read_b:
    WRITE msg2, len2
    READ char_buff, 17
    dec rax
    mov rcx, rax
    call accept
    mov [b], rbx
    ret

do_add:
    call read_a
    call read_b
    mov rbx, [a]
    add rbx, [b]
    call display
    WRITE newline, 1
    jmp menu_loop

do_div:
    call read_a
    call read_b
```

```
    mov rax, [a]
    xor rdx, rdx
    idiv qword [b]
    mov rbx, rax
    call display
    WRITE newline, 1
    jmp menu_loop

exit_prog:
    mov rax, 60
    xor rdi, rdi
    syscall

accept:
    push rcx
    push rsi
    push rax
    mov rsi, char_buff
    xor rbx, rbx
    xor rax, rax
.accept_loop:
    mov al, [rsi]
    cmp al, '0'
    jb .done
    cmp al, '9'
    ja .done
    sub al, '0'
    imul rbx, 10
```

```
    add rbx, rax
    inc rsi
    loop .accept_loop
.done:
    pop rax
    pop rsi
    pop rcx
    ret

display:
    push rbx
    push rcx
    push rdx
    push rsi
    push rdi
    mov rsi, char_buff
    add rsi, 16
    mov byte [rsi], 0
    mov rax, rbx
    mov rcx, 10
    mov rbx, 0
.convert_loop:
    xor rdx, rdx
    div rcx
    add dl, '0'
    dec rsi
    mov [rsi], dl
    inc rbx
```

```
    test rax, rax
    jnz .convert_loop
.display_output:
    WRITE rsi, rbx
    pop rdi
    pop rsi
    pop rdx
    pop rcx
    pop rbx
    ret
```

2,13) write a menu drivenALP for the following 64 bit arithmatic operarions and display the result accept the number from user.1)subtraction 2)multillpcation 3)Exit

```
; 64-bit Linux Assembly Program for Subtraction/Multiplication
%macro READ 2
    mov rax, 0
    mov rdi, 0
    mov rsi, %1
    mov rdx, %2
    syscall
%endmacro
%macro WRITE 2
    mov rax, 1
    mov rdi, 1
    mov rsi, %1
    mov rdx, %2
    syscall
```

```
%endmacro

section .data
msg1 db "Enter first number: ",10
len1 equ $-msg1
msg2 db "Enter second number: ",10
len2 equ $-msg2
menu db 10,"1. Subtraction",10,"2. Multiplication",10,"3. Exit",10
menuLen equ $-menu
select db "Enter your choice: ",0
selectLen equ $-select
newline db 10
invalid db "Invalid choice, please try again.",10
lenInvalid equ $-invalid

section .bss
a resq 1
b resq 1
char_buff resb 17
choice resb 2

section .text
global _start

_start:
menu_loop:
   WRITE menu, menuLen
   WRITE select, selectLen
```

```
    READ choice, 2
    mov al, [choice]
    cmp al, '1'
    je do_sub
    cmp al, '2'
    je do_mul
    cmp al, '3'
    je exit_prog
    WRITE invalid, lenInvalid
    jmp menu_loop

read_a:
    WRITE msg1, len1
    READ char_buff, 17
    dec rax
    mov rcx, rax
    call accept
    mov [a], rbx
    ret

read_b:
    WRITE msg2, len2
    READ char_buff, 17
    dec rax
    mov rcx, rax
    call accept
    mov [b], rbx
    ret
```

```
do_sub:
    call read_a
    call read_b
    mov rbx, [a]
    sub rbx, [b]
    call display
    WRITE newline, 1
    jmp menu_loop

do_mul:
    call read_a
    call read_b
    mov rax, [a]
    imul qword [b]
    mov rbx, rax
    call display
    WRITE newline, 1
    jmp menu_loop

exit_prog:
    mov rax, 60
    xor rdi, rdi
    syscall

accept:
    push rcx
    push rsi
```

```
    push rax
    mov rsi, char_buff
    xor rbx, rbx
    xor rax, rax
.accept_loop:
    mov al, [rsi]
    cmp al, '0'
    jb .done
    cmp al, '9'
    ja .done
    sub al, '0'
    imul rbx, 10
    add rbx, rax
    inc rsi
    loop .accept_loop
.done:
    pop rax
    pop rsi
    pop rcx
    ret

display:
    push rbx
    push rcx
    push rdx
    push rsi
    push rdi
    mov rsi, char_buff
```

```
    add rsi, 16
    mov byte [rsi], 0
    mov rax, rbx
    mov rcx, 10
    mov rbx, 0
.convert_loop:
    xor rdx, rdx
    div rcx
    add dl, '0'
    dec rsi
    mov [rsi], dl
    inc rbx
    test rax, rax
    jnz .convert_loop
.display_output:
    WRITE rsi, rbx
    pop rdi
    pop rsi
    pop rdx
    pop rcx
    pop rbx
    ret
```

3,14)Write an ALP to convert 64 bit HEX number to BCD number

```
section .data
    msg1    db "Enter HEX number: ", 0
    msg1_len equ $ - msg1
```

```
    msg2    db 10, "Result (in BCD): ", 0
    msg2_len equ $ - msg2

    newline  db 10

section .bss
    input   resb 20
    len     resq 1
    num     resq 1

section .text
    global _start

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

%macro WRITE 2
    mov rax, 1
    mov rdi, 1
    mov rsi, %1
    mov rdx, %2
    syscall
```

```
%endmacro

_start:
    WRITE msg1, msg1_len
    READ input, 20
    dec rax
    mov [len], rax

    mov rcx, [len]
    call accept

    WRITE msg2, msg2_len
    mov rax, rbx
    call print_bcd
    jmp exit

accept:
    mov rsi, input
    xor rbx, rbx
    mov rcx, [len]

hex_to_num:
    movzx rdx, byte [rsi]
    cmp dl, '9'
    jbe num_digit
    cmp dl, 'F'
    jbe upper_case
    cmp dl, 'f'
```

```
    jbe lower_case
    jmp invalid_char

upper_case:
    sub dl, 'A' - 10
    jmp got_digit
lower_case:
    sub dl, 'a' - 10
    jmp got_digit
num_digit:
    sub dl, '0'
got_digit:
    shl rbx, 4
    add rbx, rdx
    inc rsi
    dec rcx
    jnz hex_to_num
    ret

invalid_char:
    xor rbx, rbx
    ret

print_bcd:
    mov rbx, 10
    xor rcx, rcx

.divide_loop:
```

```
    xor rdx, rdx
    div rbx
    add dl, '0'
    push rdx
    inc rcx
    test rax, rax
    jnz .divide_loop

.print_loop:
    pop rax
    mov [input], al
    WRITE input, 1
    dec rcx
    jnz .print_loop
    WRITE newline, 1
    ret

exit:
    mov rax, 60
    xor rdi, rdi
    syscall
```

4 ,15)Write an ALP to convert 64 bit BCD into its equivalent HEX numbers.

```
; BCD TO HEX CONVERSION *
%macro write 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

section .data
    newline db 10
    menu db 10, "Enter BCD Number : "
    menu_len equ $ - menu

    msg2 db 10, "The Hex equivalent is : "
    len2 equ $ - msg2

section .bss
    char_buff resb 17    ; buffer for input and output
    choice resb 2
    cnt resq 1
    char resb 1

section .text
```

```
global _start

_start:
    write menu, menu_len
    read char_buff, 17

    ; Calculate actual length of input
    dec rax
    mov rcx, rax
    mov rbx, 0
    mov rsi, char_buff

bcd_loop:
    mov rax, 10
    mul rbx
    mov rbx, rax
    mov dl, byte [rsi]
    sub dl, '0'
    movzx rdx, dl
    add rbx, rdx
    inc rsi
    loop bcd_loop

    write msg2, len2
    call display_hex

    ; Exit program instead of infinite loop
    jmp Exit
```

```
; --------- Display hex (prints 16 hex digits) ---------
display_hex:
   mov rcx, 16
   mov rdi, char_buff    ; Use RDI as destination pointer

display_loop:
   rol rbx, 4
   mov al, bl
   and al, 0Fh
   cmp al, 9
   jbe add_30
   add al, 7

add_30:
   add al, '0'
   mov byte [rdi], al
   inc rdi
   dec rcx
   jnz display_loop

   write char_buff, 16
   write newline, 1
   ret

; --------- Exit program ---------
Exit:
   mov rax, 60
```

```
    xor rdi, rdi
    syscall
```

5,16,24) Write a menu driven ALP to perform multiplication of two 64 bit hexa-decimal numbers using successive addition.

```
section .data
    menu_msg     db 10, "1. Multiplication by Successive Addition", 10, "2. Exit", 10, 0
    menuLen      equ $-menu_msg

    prompt_choice db "Enter your choice: ", 0
    promptLen    equ $-prompt_choice

    prompt_m1    db "Enter multiplicand (hex): ", 0
    m1Len        equ $-prompt_m1

    prompt_m2    db "Enter multiplier (hex): ", 0
    m2Len        equ $-prompt_m2

    result_msg   db "Product (hex): ", 0
    resultLen    equ $-result_msg

    newline      db 10

section .bss
    inbuf1   resb 18
    inbuf2   resb 18
```

```
   temp    resb 2
   buf     resb 18
   m1      resq 1
   m2      resq 1
   prod    resq 1

%macro WRITE 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

section .text
   global _start

_start:

main_menu:
```

```
    WRITE menu_msg, menuLen
    WRITE prompt_choice, promptLen

    READ temp, 2        ; Read choice + newline
    mov al, [temp]

    cmp al, '1'
    je multiply
    cmp al, '2'
    je exit_program

    jmp main_menu       ; Invalid input, show menu again

multiply:
    ; --- Get Multiplicand ---
    WRITE prompt_m1, m1Len
    READ inbuf1, 18
    dec rax             ; Decrement length to ignore newline
    mov rcx, rax        ; Length for htoi
    mov rsi, inbuf1     ; Source buffer
    call htoi
    mov [m1], rbx

    ; --- Get Multiplier ---
    WRITE prompt_m2, m2Len
    READ inbuf2, 18
    dec rax             ; Decrement length to ignore newline
    mov rcx, rax
```

```
    mov rsi, inbuf2
    call htoi
    mov [m2], rbx

    ; --- Perform Multiplication (Successive Addition) ---
    mov rax, 0
    mov rbx, [m1]        ; Multiplicand
    mov rcx, [m2]        ; Multiplier (Counter)
    mov rdx, 0           ; Accumulator (Result)

    cmp rcx, 0           ; Handle multiply by zero case
    je done_mul

mul_add_loop:
    add rdx, rbx         ; Add multiplicand to result
    dec rcx              ; Decrement counter
    jnz mul_add_loop     ; Continue if counter not zero

done_mul:
    mov [prod], rdx

    ; --- Print Result ---
    WRITE result_msg, resultLen
    mov rbx, [prod]
    call itohex          ; Convert RBX to hex string in 'buf'

    WRITE newline, 1     ; Print trailing newline
    jmp main_menu
```

```
exit_program:
   mov rax, 60
   xor rdi, rdi
   syscall

; ------------------------------------------------------
; Procedure: htoi
; Converts ASCII Hex string to Integer
; Inputs:  RSI = pointer to string, RCX = length
; Output:  RBX = integer value
; ------------------------------------------------------
htoi:
   xor rbx, rbx         ; Clear result
.hnext:
   cmp rcx, 0
   je .hdone

   xor rax, rax
   mov al, [rsi]

   cmp al, 10           ; Safety check for newline
   je .hdone

   shl rbx, 4           ; Shift previous result left by 4 bits

   cmp al, '0'
   jb .skip
```

```
    cmp al, '9'
    jbe .is_digit

    cmp al, 'A'
    jb .skip
    cmp al, 'F'
    jbe .is_upper

    cmp al, 'a'
    jb .skip
    cmp al, 'f'
    jbe .is_lower

    jmp .skip           ; Invalid character

.is_digit:
    sub al, '0'
    jmp .store
.is_upper:
    sub al, 'A' - 10
    jmp .store
.is_lower:
    sub al, 'a' - 10

.store:
    add rbx, rax        ; Add new digit to result
.skip:
    inc rsi
```

```
    dec rcx
    jmp .hnext
.hdone:
    ret

; ------------------------------------------------------------
; Procedure: itohex
; Converts Integer to Hex ASCII string
; Input:  RBX = Integer value
; Output: Buffer 'buf' filled with 16 hex chars
; ------------------------------------------------------------
itohex:
    mov rsi, buf
    mov rcx, 16          ; Process 16 nibbles (64 bits)
    mov rax, rbx         ; Copy number to RAX for processing

.hex_loop:
    cmp rcx, 0
    je .done

    mov rdx, rax
    and rdx, 0x0F        ; Mask all but the last 4 bits

    cmp dl, 9
    jbe .number
    add dl, 87           ; Convert 10-15 to 'a'-'f' (87+10=97='a')
    jmp .store_char
```

```
.number:
   add dl, 48           ; Convert 0-9 to '0'-'9'

.store_char:
   mov [rsi+rcx-1], dl    ; Store backwards (at end of buffer)
   shr rax, 4           ; Shift right to get next nibble
   dec rcx
   jmp .hex_loop

.done:
   WRITE buf, 16        ; Print the 16-char buffer
   Ret
```

6,17)Write a menu driven ALP program to implement following string operatons.1)String length 2)string compare 3)string palindrome 4)Exit

```
%macro WRITE 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

section .data
    menu_msg db 10, "Select option:", 10, "1. String length", 10, "2. String compare", 10, \
            "3. String palindrome", 10, "4. Exit", 10, "Enter choice: ", 0
    menu_len equ $-menu_msg
    msg1 db "Enter string 1: ", 0
    len1 equ $-msg1
    msg2 db "Enter string 2: ", 0
    len2 equ $-msg2
    msgLen db "Length: ", 0
    lenLen equ $-msgLen
    eqMsg db "Strings are equal.", 10
    eqLen equ $-eqMsg
    neMsg db "Strings are not equal.", 10
    neLen equ $-neMsg
    palMsg db "Palindrome.", 10
    palLen equ $-palMsg
    npalMsg db "Not palindrome.", 10
    npalLen equ $-npalMsg
    newline db 10

section .bss
    string1 resb 40
```

```
    string2 resb 40
    temp resb 4
    strLen1 resq 1
    strLen2 resq 1
    num_buffer resb 20

section .text
global _start

_start:
    call main_menu

main_menu:
    WRITE menu_msg, menu_len
    READ temp, 2
    mov al, [temp]
    cmp al, '1'
    je strlen
    cmp al, '2'
    je strcmp
    cmp al, '3'
    je strpal
    cmp al, '4'
    je exit
    jmp main_menu

strlen:
    WRITE msg1, len1
```

```
    READ string1, 40
    dec rax
    mov [strLen1], rax
    WRITE msgLen, lenLen
    mov rbx, [strLen1]
    call showint
    WRITE newline, 1
    jmp main_menu

strcmp:
    WRITE msg1, len1
    READ string1, 40
    dec rax
    mov [strLen1], rax
    WRITE msg2, len2
    READ string2, 40
    dec rax
    mov [strLen2], rax

    mov rcx, [strLen1]
    mov rdx, [strLen2]
    cmp rcx, rdx
    jne notequal

    mov rsi, string1
    mov rdi, string2
    mov rcx, rdx
    repe cmpsb
```

```
    jne notequal

    WRITE eqMsg, eqLen
    jmp main_menu

notequal:
    WRITE neMsg, neLen
    jmp main_menu

strpal:
    WRITE msg1, len1
    READ string1, 40
    dec rax
    mov [strLen1], rax

    mov rsi, string1
    mov rdi, string1
    mov rcx, [strLen1]
    add rdi, rcx
    dec rdi

check_pal:
    cmp rsi, rdi
    jge is_pal
    mov al, [rsi]
    mov dl, [rdi]
    cmp al, dl
    jne not_pal
```

```
    inc rsi
    dec rdi
    jmp check_pal

is_pal:
    WRITE palMsg, palLen
    jmp main_menu

not_pal:
    WRITE npalMsg, npalLen
    jmp main_menu

exit:
    mov rax, 60
    xor rdi, rdi
    syscall

; Function to display number in RBX as decimal
showint:
    mov rdi, num_buffer
    mov rcx, 0
    mov rax, rbx

.convert_loop:
    xor rdx, rdx
    mov rbx, 10
    div rbx
    add dl, '0'
```

```
    push rdx
    inc rcx
    test rax, rax
    jnz .convert_loop

    mov rbx, rcx  ; save length

.pop_loop:
    pop rax
    mov [rdi], al
    inc rdi
    dec rcx
    jnz .pop_loop

    ; Write the number
    mov rax, 1
    mov rdi, 1
    mov rsi, num_buffer
    mov rdx, rbx
    syscall

    ret
```

7,18)Write a menu driven ALP program to implement following string operations. 1)string copy 2)String reverse 3)check for sunbstring 4)Exit

```
%macro WRITE 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

section .data
    menu_msg db 10, "Select option:", 10, "1. String copy", 10, "2. String reverse", 10, \
            "3. Check substring", 10, "4. Exit", 10, "Enter choice: ", 0
    menuLen  equ $-menu_msg
    msg1     db "Enter string 1: ", 0
    len1     equ $-msg1
    msg2     db "Enter string 2: ", 0
    len2     equ $-msg2
    copyMsg  db "Copied string: ", 0
    copyLen  equ $-copyMsg
    revMsg   db "Reversed string: ", 0
    revLen   equ $-revMsg
    subMsg   db "Substring found.", 10
```

```
    subLen   equ $-subMsg
    nsubMsg  db "Not a substring.", 10
    nsubLen  equ $-nsubMsg
    newline  db 10

section .bss
    string1 resb 40
    string2 resb 40
    string3 resb 40
    temp    resb 2
    l1      resq 1
    l2      resq 1

section .text
    global _start

_start:
main_menu:
    WRITE menu_msg, menuLen
    READ temp, 2
    mov al, [temp]
    cmp al, '1'
    je strcpy
    cmp al, '2'
    je strrev
    cmp al, '3'
    je strsub
    cmp al, '4'
```

```
    je exit
    jmp main_menu  ; invalid input

strcpy:
    WRITE msg1, len1
    READ string1, 40
    dec rax
    mov [l1], rax
    mov rsi, string1
    mov rdi, string3
    mov rcx, [l1]
    cld
    rep movsb              ; copy string1 to string3
    WRITE copyMsg, copyLen
    WRITE string3, [l1]
    WRITE newline, 1
    jmp main_menu

strrev:
    WRITE msg1, len1
    READ string1, 40
    dec rax
    mov [l1], rax
    mov rcx, [l1]
    mov rsi, string1
    add rsi, rcx
    dec rsi                ; now rsi points to last char of string1
    mov rdi, string3
```

```
    mov rcx, [l1]

rev_loop:
    mov al, [rsi]
    mov [rdi], al
    dec rsi
    inc rdi
    dec rcx
    jnz rev_loop

    WRITE revMsg, revLen
    WRITE string3, [l1]
    WRITE newline, 1
    jmp main_menu

strsub:
    WRITE msg1, len1
    READ string1, 40
    dec rax
    mov [l1], rax
    WRITE msg2, len2
    READ string2, 40
    dec rax
    mov [l2], rax

    mov rbx, [l1]          ; length of main string
    mov rdx, [l2]          ; length of substring
```

```
    cmp rbx, rdx
    jl not_sub              ; if substring longer than main string

    mov rsi, string1        ; main string pointer
    mov rdi, string2        ; substring pointer

find_sub:
    mov rcx, rdx            ; substring length
    push rsi
    push rdi
    repe cmpsb
    pop rdi
    pop rsi
    je is_sub

    inc rsi
    dec rbx
    cmp rbx, rdx
    jge find_sub

not_sub:
    WRITE nsubMsg, nsubLen
    jmp main_menu

is_sub:
    WRITE subMsg, subLen
    jmp main_menu
```

```
exit:
   mov rax, 60
   xor rdi, rdi
   syscall
```

8,19) Write a menu driven ALP program to implement following string operations. 1)string concantenate 2) string compare 3) string length 4)exit

```
%macro WRITE 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

section .data
   menu_msg db 10, "Select:", 10, "1. String Concatenate", 10, "2. String Compare", 10, \
```

```
        "3. String Length", 10, "4. Exit", 10, "Enter choice: ", 0
    menuLen  equ $-menu_msg

    msg1    db "Enter string 1: ", 0
    msg1Len  equ $-msg1
    msg2    db "Enter string 2: ", 0
    msg2Len  equ $-msg2
    catMsg   db "Concatenated string: ", 0
    catLen   equ $-catMsg
    eqMsg    db "Strings equal.", 10
    eqLen    equ $-eqMsg
    neMsg    db "Strings not equal.", 10
    neLen    equ $-neMsg
    lenMsg   db "String length: ", 0
    lenLen   equ $-lenMsg
    newline  db 10

section .bss
    string1    resb 40
    string2    resb 40
    string3    resb 80     ; for concatenation result
    temp       resb 2
    l1         resq 1
    l2         resq 1
    l3         resq 1
    num_buffer resb 20     ; for number display

section .text
```

```
    global _start


_start:
main_menu:
    WRITE menu_msg, menuLen
    READ temp, 2
    mov al, [temp]
    cmp al, '1'
    je strcat
    cmp al, '2'
    je strcmp
    cmp al, '3'
    je strlen
    cmp al, '4'
    je exit
    jmp main_menu

strcat:
    WRITE msg1, msg1Len
    READ string1, 40
    dec rax
    mov [l1], rax

    WRITE msg2, msg2Len
    READ string2, 40
    dec rax
    mov [l2], rax
```

```
    mov rsi, string1
    mov rdi, string3
    mov rcx, [l1]
    cld
    rep movsb           ; copy string1 to string3

    mov rsi, string2
    mov rcx, [l2]
    rep movsb           ; append string2 to string3

    mov rbx, [l1]
    add rbx, [l2]
    mov [l3], rbx

    WRITE catMsg, catLen
    WRITE string3, [l3]
    WRITE newline, 1
    jmp main_menu

strcmp:
    WRITE msg1, msg1Len
    READ string1, 40
    dec rax
    mov [l1], rax

    WRITE msg2, msg2Len
    READ string2, 40
    dec rax
```

```
    mov [l2], rax

    mov rbx, [l1]
    cmp rbx, [l2]
    jne not_equal

    mov rsi, string1
    mov rdi, string2
    mov rcx, rbx
    cld
    repe cmpsb
    jne not_equal

    WRITE eqMsg, eqLen
    jmp main_menu

not_equal:
    WRITE neMsg, neLen
    jmp main_menu

strlen:
    WRITE msg1, msg1Len
    READ string1, 40
    dec rax
    mov [l1], rax

    WRITE lenMsg, lenLen
    mov rbx, [l1]
```

```
    call display_len
    WRITE newline, 1
    jmp main_menu

exit:
    mov rax, 60
    xor rdi, rdi
    syscall

; Display value in rbx as ASCII decimal
display_len:
    mov rdi, num_buffer
    mov rcx, 0
    mov rax, rbx

.convert_loop:
    xor rdx, rdx
    mov rbx, 10
    div rbx
    add dl, '0'
    push rdx
    inc rcx
    test rax, rax
    jnz .convert_loop

    mov rbx, rcx  ; save length

.pop_loop:
```

```
    pop rax
    mov [rdi], al
    inc rdi
    dec rcx
    jnz .pop_loop

    ; Write the number using direct syscall
    mov rax, 1
    mov rdi, 1
    mov rsi, num_buffer
    mov rdx, rbx
    syscall
    ret
```

11,23) Write an ALP to find the roots of quadratic equation.

ROOTS OF QUADRATIC EQUATION

```
%macro scan 2
    push rbp
    mov rax,0
    mov rdi,%1
    mov rsi,%2
    call scanf
    pop rbp
%endmacro

%macro printfloat 2
```

```
    push rbp
    mov rax,1
    mov rdi,%1
    movsd xmm0,%2
    call printf
    pop rbp
%endmacro

section .data
    m1 db "%lf",0
    m2 db "%s",0
    msg1 db 10,"Enter the values of a, b, & c",0
    msg2 db 10,"Roots are",0

section .bss
    a resb 8
    b resb 8
    c resb 8
    temp resw 1
    t1 resb 8
    t2 resb 8
    t3 resb 8
    t4 resb 8
    r1 resb 10
    r2 resb 10

section .text
    global main
```

```
    extern printf, scanf

main:
    ; print message
    mov rdi, m2
    mov rsi, msg1
    xor rax, rax
    call printf

    ; input values
    scan m1, a
    scan m1, b
    scan m1, c

    finit
    fld qword [b]
    fmul st0, st0
    fstp qword [t1]
    fld qword [a]
    fmul qword [c]
    mov word [temp], 4
    fimul word [temp]
    fstp qword [t2]
    fld qword [t1]
    fsub qword [t2]
    fstp qword [t4]
    fld qword [t4]
    fabs
```

```
    fsqrt
    fstp qword [t1]
    fld qword [b]
    fchs
    fstp qword [t2]
    fld qword [a]
    mov word [temp], 2
    fimul word [temp]
    fstp qword [t3]
    cmp qword [t4], 0
    je equal_root
    fld qword [t2]
    fadd qword [t1]
    fdiv qword [t3]
    fstp qword [r1]
    printfloat m1, [r1]
equal_root:
    fld qword [t2]
    fsub qword [t1]
    fdiv qword [t3]
    fstp qword [r2]
    printfloat m1, [r2]
    mov rax, 0
    ret
```

25,26) Write a menu driven ALP to perform multiplication of two 64-bit hexa-decimal number using add and shift method.

```
section .data
    menu db 10, "--- Multiply 2 Hexadecimal 64-bit Numbers ---", 10
        db "1. Multiplication using Add and Shift", 10
        db "2. Exit", 10
        db "Enter your choice: ", 0
    menu_len equ $-menu

    msg_multiplicand db "Enter multiplicand (hex): ", 0
    msg_multiplier   db "Enter multiplier (hex): ", 0
    msg_product      db "Product (hex): ", 0
    newline          db 10
    hex_chars        db "0123456789ABCDEF"

section .bss
    choice       resb 4
    input_buffer  resb 20
    multiplicand  resq 1
    multiplier    resq 1
    product       resq 1

section .text
    global _start

_start:
main_menu:
    ; Display menu
    mov rax, 1
    mov rdi, 1
```

```
    mov rsi, menu
    mov rdx, menu_len
    syscall

    ; Read choice
    mov rax, 0
    mov rdi, 0
    mov rsi, choice
    mov rdx, 4
    syscall

    ; Check choice
    mov al, [choice]
    cmp al, '1'
    je multiplication
    cmp al, '2'
    je exit
    jmp main_menu

multiplication:
    ; Get multiplicand
    mov rax, 1
    mov rdi, 1
    mov rsi, msg_multiplicand
    mov rdx, 21
    syscall

    call read_hex
```

```
mov [multiplicand], rax

; Get multiplier
mov rax, 1
mov rdi, 1
mov rsi, msg_multiplier
mov rdx, 19
syscall

call read_hex
mov [multiplier], rax

; Perform multiplication using add and shift
call multiply_add_shift

; Display result
mov rax, 1
mov rdi, 1
mov rsi, msg_product
mov rdx, 15
syscall

mov rax, [product]
call print_hex

mov rax, 1
mov rdi, 1
mov rsi, newline
```

```
    mov rdx, 1
    syscall

    jmp main_menu

exit:
    mov rax, 60
    xor rdi, rdi
    syscall

; Read hexadecimal number from input
read_hex:
    mov rax, 0
    mov rdi, 0
    mov rsi, input_buffer
    mov rdx, 16
    syscall

    ; Convert hex string to number
    mov rsi, input_buffer
    xor rax, rax
    xor rcx, rcx

.convert_loop:
    mov cl, [rsi]
    cmp cl, 10    ; newline
    je .done
    cmp cl, 13    ; carriage return
```

```
    je .done
    cmp cl, '0'
    jb .next_char
    cmp cl, '9'
    ja .check_upper

    ; Digit 0-9
    sub cl, '0'
    jmp .add_digit

.check_upper:
    cmp cl, 'A'
    jb .check_lower
    cmp cl, 'F'
    ja .check_lower

    ; Upper case A-F
    sub cl, 'A' - 10
    jmp .add_digit

.check_lower:
    cmp cl, 'a'
    jb .next_char
    cmp cl, 'f'
    ja .next_char

    ; Lower case a-f
    sub cl, 'a' - 10
```

```
.add_digit:
    shl rax, 4
    add rax, rcx

.next_char:
    inc rsi
    jmp .convert_loop

.done:
    ret

; Print hexadecimal number
print_hex:
    mov rdi, input_buffer
    mov rcx, 16    ; 16 hex digits for 64-bit

.convert:
    rol rax, 4
    mov rbx, rax
    and rbx, 0xF
    mov bl, [hex_chars + rbx]
    mov [rdi], bl
    inc rdi
    loop .convert

    ; Print the hex string
    mov rax, 1
```

```
    mov rdi, 1
    mov rsi, input_buffer
    mov rdx, 16
    syscall
    ret

; Multiplication using add and shift method
multiply_add_shift:
    mov rax, [multiplicand]
    mov rbx, [multiplier]
    xor rdx, rdx    ; Clear product

.multiply_loop:
    test rbx, 1     ; Check if LSB is 1
    jz .no_add
    add rdx, rax    ; Add multiplicand to product

.no_add:
    shl rax, 1      ; Shift multiplicand left
    shr rbx, 1      ; Shift multiplier right
    jnz .multiply_loop

    mov [product], rdx
    ret
```