q.1.Write 64-bit ALP to “Hello World” in NASM

```
section .data
    hello db 'Hello, World!', 0x0A  ; The string to print with newline
    len equ $ - hello               ; Length of the string

section .text
    global _start                   ; Entry point for the program

_start:
    ; Write system call
    mov rax, 1                      ; sys_write (system call number)
    mov rdi, 1                      ; file descriptor (stdout)
    mov rsi, hello                  ; address of the string
    mov rdx, len                    ; length of the string
    syscall                         ; invoke the system call

    ; Exit system call
    mov rax, 60                     ; sys_exit (system call number)
    xor rdi, rdi                    ; exit status (0)
    syscall                         ; invoke the system call
```

q.2.Write 64-bit ALP to accept number and display it on screen.

```
section .bss
   num resb 20             ; Reserve space for the input (max 20 bytes)

section .text
   global _start           ; Declare the entry point

_start:
   ; Prompt the user
   mov rax, 1              ; sys_write system call
   mov rdi, 1              ; File descriptor (stdout)
   mov rsi, prompt         ; Address of the prompt message
   mov rdx, prompt_len     ; Length of the prompt message
   syscall                 ; Call the kernel

   ; Read input from user
   mov rax, 0              ; sys_read system call
   mov rdi, 0              ; File descriptor (stdin)
   mov rsi, num            ; Buffer to store the input
   mov rdx, 20             ; Maximum number of bytes to read
   syscall                 ; Call the kernel

   ; Display the number back to the user
   mov rax, 1              ; sys_write system call
   mov rdi, 1              ; File descriptor (stdout)
   mov rsi, num            ; Address of the input buffer
   mov rdx, 20             ; Length of the input
   syscall                 ; Call the kernel

   ; Exit program
   mov rax, 60             ; sys_exit system call
   xor rdi, rdi            ; Exit code 0
   syscall                 ; Call the kernel

section .data
   prompt db 'Enter a number: ', 0x0A  ; Prompt message
   prompt_len equ $ - prompt           ; Length of the prompt message
```

q.3.Write 64-bit ALP to convert HEX 4-digit input to BCD 5-digit output.

```
%macro scall 4
mov rax,%1
mov rdi,%2
mov rsi,%3
mov rdx,%4
syscall
%endmacro
section .data
m1 db 10,"enter 4 digit Hex Number"
l1 equ $-m1
m2 db 10,13,"Equivalent BCD number is:"
l2 equ $-m2
section .bss
buf resb 6
digitcount resb 1
char_ans resb 4
section .text
global _start
_start:
scall 01,01,m1,l1
scall 0,0,buf,5
call accept_proc
mov ax,bx
call h2bproc
;----------------Exit------------
mov rax,60
mov rdi,0
syscall
;------------HEX TO BCD PROCEDURE---------
h2bproc:
mov rbx, 0Ah
back:
xor rdx,rdx
div rbx
push dx
inc byte[digitcount]
cmp rax,0h
jne back
scall 01,01,m2,l2
print_bcd:
pop dx
add dl,30h
mov [char_ans],dl
scall 01,01,char_ans,1
```

```
dec byte[digitcount]
jnz print_bcd
ret
;----------------Accept Procedure------------
accept_proc:
xor bx,bx
mov rcx,4
mov rsi,buf
next_digit:
shl bx,04
mov al,[rsi]
cmp al,39h
jbe label1
sub al,07h
label1:
sub al,30h
add bx,ax
inc rsi
loop next_digit
ret
```

Netraj

```
section .data
   msg db "Enter a Hex number : "
   len equ $ - msg
   decimal dw 0

section .bss
   hex resb 5
   integer resb 10

section .text
   global _start
_start:
   ;input number
   mov rax, 1
   mov rdi, 1
   mov rsi, msg
   mov rdx, len
   syscall

   mov rax, 0
   mov rdi, 0
   mov rsi, hex
   mov rdx, 5
```

```
    syscall
    ;-----------------------------------------
    ; convert hex to decimal
    mov rsi, hex
    call hexStr_to_decimal
    mov [decimal], rax

    ;-----------------------------------------------
    ; convert decimal to integer string
    mov rsi, integer
    call int_to_str

    ; reverse
    dec rcx
    mov rsi, integer
    call reverse

    ;-----------------------------------------------
    ; print
    mov rax, 1
    mov rdi, 1
    mov rsi, integer
    mov rdx, 10
    syscall

    ; -----------------------------------------------
    ; end program
    mov rax, 60
    xor rdi, rdi
    syscall
;=============================================================
hexStr_to_decimal:
    xor rcx, rcx
    xor rax, rax
l1:
    movzx rbx, byte [rsi + rcx]
    cmp bl, 0
    je done1
    cmp bl, 10
    je done1

    sub bl, '0'
    cmp bl, 9
    jng addNum
    sub bl, 7
addNum:
```

```
    imul rax, rax, 16
    add rax, rbx
    inc rcx
    jmp l1
done1:
    ret
;-------------------------------------------------------
; int_to_str
int_to_str:
    xor rcx, rcx
    xor rbx, rbx
    mov rbx, 10
    test rax, rax
    mov byte[rsi], '0'
    jz zero
l2:
    test rax, rax
    jz done2
    xor rdx, rdx
    div rbx
    add dl, '0'
    mov byte[rsi + rcx], dl
    inc cx
    jmp l2
done2:
    mov byte[rsi + rcx], 0
    ret
zero:
    inc rcx
    mov byte[rsi + rcx], 0
    ret

reverse:
    xor rax, rax
l3:
    cmp rcx, 0
    jle done3
    mov ah, byte[rsi]
    mov al, byte[rsi + rcx]
    mov byte[rsi], al
    mov byte[rsi + rcx], ah
    sub rcx, 2
    inc rsi
    jmp l3
done3:
    ret
```

q.4.Write 64-bit ALP to accept the numbers from user and perform addition of 2 numbers and display the result on screen.

```
section .data
   msg db "Enter number: ", 0
   msg_len equ $ - msg
   result_msg db "Result: ", 0
   result_msg_len equ $ - result_msg
   newline db 10

section .bss
   var1 resb 20
   var2 resb 20
   result resb 20
   num1 resq 1
   num2 resq 1


section .text
   global _start

_start:
   ; First number
   mov rax, 1
   mov rdi, 1
   mov rsi, msg
   mov rdx, msg_len
   syscall

   mov rax, 0
   mov rdi, 0
   mov rsi, var1
   mov rdx, 20
   syscall

   ; Second number
   mov rax, 1
   mov rdi, 1
   mov rsi, msg
   mov rdx, msg_len
   syscall

   mov rax, 0
   mov rdi, 0
   mov rsi, var2
   mov rdx, 20
   syscall

   ; Convert strings to integers
```

```
    mov rsi, var1
    call str_to_int
    mov [num1], rax

    mov rsi, var2
    call str_to_int
    mov [num2], rax

    ; Add numbers
    mov rax, [num1]
    add rax, [num2]

    ; Convert result to string
    mov rsi, result
    call int_to_str

    ; Print result
    mov rax, 1
    mov rdi, 1
    mov rsi, result_msg
    mov rdx, result_msg_len
    syscall

    mov rax, 1
    mov rdi, 1
    mov rsi, result
    mov rdx, 20
    syscall

    ; Print newline
    mov rax, 1
    mov rdi, 1
    mov rsi, newline
    mov rdx, 1
    syscall

    ; Exit
    mov rax, 60
    xor rdi, rdi
    syscall

str_to_int:
    xor rax, rax
    xor rcx, rcx

.loop:
    movzx rdx, byte [rsi + rcx]
    cmp dl, 10
    je .done
    cmp dl, 0
```

```
    je .done

    sub dl, '0'
    imul rax, 10
    add rax, rdx

    inc rcx
    jmp .loop

.done:
    ret

int_to_str:
    mov rdi, rsi
    add rdi, 19
    mov byte [rdi], 0
    mov rcx, 10

.loop:
    xor rdx, rdx
    div rcx
    add dl, '0'
    dec rdi
    mov [rdi], dl
    test rax, rax
    jnz .loop

    mov rax, rdi
    ret
```

Netraj

```
section .data
    num1 db 0
    num2 db 0
    msg db "Enter number ",0
    msg_len equ $ - msg

section .bss
    str1 resb 5
    str2 resb 5
    result resb 5

section .text
    global _start
_start:
    ; first number
    mov rax, 1
    mov rdi, 1
    mov rsi, msg
```

```
mov rdx, msg_len
syscall

mov rax, 0
mov rdi, 0
mov rsi, str1
mov rdx, 5
syscall

; call str_to_int
mov rsi, str1
call str_to_int
mov [num1], al
;-----------------------------------------------

; second number
mov rax, 1
mov rdi, 1
mov rsi, msg
mov rdx, msg_len
syscall

mov rax, 0
mov rdi, 0
mov rsi, str2
mov rdx, 5
syscall

; call str_to_int
mov rsi, str2
call str_to_int
mov [num2], al
;-----------------------------------------------
; add the numbers
mov al, [num1]
mov bl, [num2]
add al, bl
;-----------------------------------------------
; convert int to string
mov rsi, result
call int_to_str
;-----------------------------------------------
; print
mov rax, 1
mov rdi, 1
mov rsi, result
mov rdx, 5
syscall
;-----------------------------------------------
; exit program
```

```
    mov rax, 60
    xor rdi, rdi
    syscall
;=============================================================
str_to_int:
    xor rax, rax
    xor rcx, rcx
    xor rbx, rbx

l1:
    cmp byte[rsi + rcx], 0
    je done1
    cmp byte[rsi + rcx], 10
    je done1

    movzx bx, byte [rsi + rcx]
    sub bl, '0'
    imul rax, rax, 10
    add rax, rbx
    inc rcx
    jmp l1
done1:
    ret
;-------------------------------------------------------
int_to_str:
    xor rcx, rcx
    mov rbx, 10

    ; Handle zero case
    test rax, rax
    jnz l2
    mov byte [rsi], '0'
    inc rsi
    mov byte [rsi], 0
    mov rcx, 1
    jmp done2
l2:
    xor rdx, rdx
    div rbx
    add dl, '0'
    mov byte [rsi + rcx], dl
    inc rcx
    test rax, rax
    jnz l2

reverse:
    mov byte [rsi + rcx], 0
    dec rcx

l3:
```

```
    cmp rcx, 0
    jle done2
    mov ah, byte [rsi]
    mov al, byte [rsi + rcx]
    mov byte [rsi], al
    mov byte [rsi + rcx], ah

    sub rcx, 2
    inc rsi
    jmp l3
done2:
    ret
```

q.5.Write 64-bit ALP to perform following string operations i) Length of String ii) Reverse of String

```
%macro print 2
	mov rax,1
	mov rdi,1
	mov rsi,%1
	mov rdx,%2
	syscall
%endmacro
%macro read 2
	mov rax,0
	mov rdi,0
	mov rsi,%1
	mov rdx,%2
	syscall
%endmacro
section .data
	m1 db "Enter the string :"
	l1 equ $-m1
	m2 db "Length of the string:"
	l2 equ $-m2
	m3 db "Reverse of the string :"
	l3 equ $-m3
section .bss
	string resb 50
	string2 resb 50
	length resb 16
	answer resb 16
section .text
	global _start
_start:
	print m1,l1
	read string,50
	dec rax
	mov [length],rax
	print m2,l2
	mov rax,[length]
	mov rsi,answer+15
	mov rcx,16
loop1:
	mov rdx,0
	mov rbx,10
	div rbx
	cmp dl,09h
	jbe skip1
	add dl,07h
skip1:
	add dl, 30h
	mov [rsi],dl
```

```
        dec rsi
        dec rcx
        jnz loop1
        print answer,16
        mov rsi,string
  mov rdi,string2
        mov rcx,[length]
        add rdi,rcx
        dec rdi
loop2:
        mov al,[rsi]
        mov [rdi],al
        inc rsi
        dec rdi
        loop loop2
        print m3,l3
        print string2,[length]
        mov rax,60
        mov rdi,0
        syscall
```

q.6.Write 64-bit ALP to perform multiplication of two 8-bit hexadecimal number with successive addition

```
section .bss
   num1    resb 5
   num2    resb 5
   result  resb 20

section .data
   msg1        db 'Enter first number: ', 0
   msg2        db 'Enter second number: ', 0
   msg_result  db 'Result: ', 0
   newline     db 0xA
   ten         dq 10

%macro scall 4
   mov rax, %1
   mov rdi, %2
   mov rsi, %3
   mov rdx, %4
   syscall
%endmacro


section .text
   global _start

_start:

   scall 1,1,msg1,19

   scall 0,0,num1,5

   call str_to_int
   mov rbx, rax



   scall 1,1,msg2,20


   scall 0,0,num2,5



   call str_to_int
   mov rcx, rax

   xor rdx, rdx
```

```
multiply:
    cmp rcx, 0
    je done
    add rdx, rbx
    dec rcx
    jmp multiply

done:
    mov rax, rdx
    call int_to_str
    scall 1,1,msg_result,8

    scall 1,1,result,20

    scall 1,1,newline,1

    mov rax, 60
    xor rdi, rdi
    syscall

str_to_int:
    xor rax, rax
    xor rdi, rdi

next_digit:
    mov dl, [rsi + rdi]
    cmp dl, 0xA
    je done_conv
    sub dl, '0'
    imul rax, rax, 10
    add rax, rdx
    inc rdi
    jmp next_digit

done_conv:
    ret

int_to_str:
    mov rbx, 0
    mov rdi, result + 19
    mov byte [rdi], 0
    dec rdi

    test rax, rax
    jz int_zero

int_loop:
    xor rdx, rdx
    div qword [ten]
    add dl, '0'
```

```
    mov [rdi], dl
    dec rdi
    inc rbx
    test rax, rax
    jnz int_loop

jmp int_done

int_zero:
    mov byte [rdi], '0'
    dec rdi
    inc rbx

int_done:
    mov rsi, rdi
    inc rsi
    ret
```