Assign 2 : length of string

```
section .data
msg db"This is the assembly language code",10
msg_len equ $-msg
msg1 db"Enter the string:",10
msg1_len equ $-msg1
msg2 db"Entered string is: "
msg2_len equ $-msg2
msg3 db"The length of string is: "
msg3_len equ $-msg3
msg4 db" ",10
msg4_len equ $-msg4

%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


section .bss
buffer resb 20
buffer_len equ $-buffer
charans resb 2
result resq 1

section .text

global _start
_start:

print msg,msg_len
print msg1,msg1_len
read buffer, buffer_len
```

```
dec rax
call Display
print msg2,msg2_len
print buffer,buffer_len

mov rax,60
mov rdi,0
syscall


Display:

mov rbx,16
mov rcx,2
mov rsi,charans+1

next:
mov rdx,0
div rbx
cmp dl,09h
jbe add30
add dl,07h

add30:
add dl,30h

mov [rsi], dl
dec rsi
dec rcx
jnz next
print msg3,msg3_len
print charans,2
print msg4, msg4_len
ret
```

ASSIGN 3 : To count positive negative numbers

```
section .data
arr dq 123H, -120H, 12H, -20H, -20H, 66H, -918737H, 9922H, -293H, -987H
n equ 10
msg db"Assignment no.2 : Calculate the count of positive and negative numbers in array",10
msg_len equ $-msg
```

```
msg1 db"The positive numbers are: "
msg1_len equ $-msg1
msg2 db 10,"The negative numbers are: "
msg2_len equ $-msg2
msg3 db 10," "
msg3_len equ $-msg3

%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

section .bss
charans resb 4
result resq 1
p_count resb 8
n_count resb 8

section .text
global _start
_start:

print msg,msg_len

mov rsi,arr
mov rcx,n
mov rbx,0;
mov rdx,0;

next_num:

mov rax,[rsi]
Rol rax,1
jc negative
```

```
positive:

inc rbx
jmp next1

negative:
inc rdx

next1:
add rsi,8
dec rcx
jnz next_num

mov [p_count],rbx
mov [n_count],rdx

print msg1,msg1_len
mov rax,[p_count]
call Display

print msg2,msg2_len
mov rax,[n_count]
call Display

print msg3,msg3_len
mov rax,60
mov rdi,0
syscall

Display:

mov rbx,16
mov rcx,2
mov rsi,charans+1

next:
mov rdx,0
div rbx
cmp dl,09h
jbe add30
add dl,07h

add30:
add dl,30h

mov [rsi], dl
dec rsi
```

```
dec rcx
jnz next
print charans,2
ret
```

ASSIGN 4 : BCD TO HEX

```
section .data
msg1 db "Enter the 5 digit bcd number:"
msg1_len equ $-msg1
msg2 db 10,"The equivalent hexadecimal number is : "
msg2_len equ $-msg2
msg3 db "",10
msg3_len equ $-msg3
msg4 db 10,"Invalid input.Try again!"
msg4_len equ $-msg4

%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

section .bss
buf resb 6
charans resb 4
result resq 1
ans resb 2

section .text
global _start
```

```
_start:

call Bcd_Hex

print msg3,msg3_len

mov rax,60
mov rdi,0
syscall

Bcd_Hex:
print msg1,msg1_len
read buf,6

mov rsi,buf
xor ax,ax
mov rbp,5
mov rbx,10

next:
xor cx,cx
mul bx
mov cl,[rsi]

cmp cl,'0'
jb error

cmp cl,'9'
ja error

cmp cl,'9'
jbe sub30

sub30: sub cl,30h

add ax,cx

inc rsi
dec rbp
jnz next

mov [ans],ax
print msg2,msg2_len
mov ax,[ans]
call Display
ret

error:
```

```
print msg4,msg4_len
print msg3,msg3_len
mov rax,60
mov rdi,0
syscall

Display:

mov rbx,16
mov rcx,4
mov rsi,charans+3

next1:
mov rdx,0
div rbx
cmp dl,09h
jbe add30
add dl,07h

add30:
add dl,30h

mov [rsi], dl
dec rsi
dec rcx
jnz next1
print charans,4
ret
```

ASSIGN 5 - HEX TO BCD

```
section .data
   msg1 db "Enter a 4 digit HEX number: "
   msg1_len equ $-msg1
   msg2 db 10,"The equivalent BCD number: "
   msg2_len equ $-msg2
   err db 10,"Please enter valid hex number",10
   err_len equ $-err
%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
mov rax,60
mov rdi,0
syscall
%endmacro
section .bss
buf resb 5
charAns resb 1
ans resw 1


section .text
    global _start

_start:

 call hex_bcd

exit

hex_bcd:
print msg1,msg1_len
call accept

mov ax,bx
mov bx,10
xor bp,bp ; or use mov bp,0
back:
xor dx,dx
div bx
push dx
```

```
inc bp

cmp ax,0
jne back

print msg2,msg2_len
back1:
pop dx
add dl,30h  ; as the popped number is of one byte we will use dl
mov [charAns],dl
print charAns,1

dec bp
jnz back1
ret
accept:
read buf,5 ; 4 numbers + 1 enter
mov rcx,4
mov rsi,buf ; to access elements we need to store base address in rsi
xor bx,bx

next_byte:
shl bx,4
mov al,[rsi]

cmp al,'0'  ; to check whether it is less than 0
jb error
cmp al,'9'
jbe sub30

cmp al,'A'
jb error
cmp al,'F'
ja error
cmp al,'F'
jbe sub37

cmp al,'a'
jb error
cmp al,'f'
ja error
cmp al,'f'
jbe sub57

sub57: sub al,20h  ; due to serial execution from sub30 it becomes 57h
sub37: sub al,07h
sub30: sub al,30h
```

```
add bx,ax
inc rsi
dec rcx
jnz next_byte
ret
error:
print err,err_len
exit
```

ASSIGN 6 - BLOCK TRANSFER (NON - OVERLAPPED)

```
; ALP PROGRAM FOR BLOCK TRANSFER NON OVERLAP WITHOUT STRING
INSTRUCTIONS
; ROLL : SYCOD272


section .data
sblock db 10h,20h, 30h, 40h,50h
dblock db 0h,0h,0h,0h,0h
msg db "The source array is: "
msg_len equ $-msg
msg1 db 10, "The destination array is: "
msg1_len equ $-msg1
msg2 db "",10
msg2_len equ $-msg2
msg3 db "After block transfer, the arrays are :",10
```

```
msg3_len equ $-msg3
space db " "


%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

section .bss
charans resb 4
result resq 1

section .text
global _start
_start:
print msg,msg_len
mov rsi,sblock
call Display_block

print msg1, msg1_len
mov rsi,dblock
call Display_block

print msg2,msg2_len

call Block_transfer

print msg3,msg3_len

print msg,msg_len
mov rsi,sblock
call Display_block
```

```
print msg1, msg1_len
mov rsi,dblock
call Display_block

print msg2,msg2_len

mov rax,60
mov rdi,0
syscall

Display:

mov rbx,16
mov rcx,2
mov rsi,charans+1

next:
mov rdx,0
div rbx
cmp dl,09h
jbe add30
add dl,07h

add30:
add dl,30h

mov [rsi], dl
dec rsi
dec rcx
jnz next
print charans,2
ret

Display_block :
mov rbp,5
 next_num:
 mov al,[rsi]
 push rsi
 call Display
 print space,1
 pop rsi
 inc rsi
 dec rbp
 jnz next_num
 ret

Block_transfer:
mov rsi,sblock
```

```
mov rdi,dblock
mov rcx,5

nextb:
mov al,[rsi]
mov [rdi],al

inc rsi
inc rdi
dec rcx
jnz nextb
ret

—----------------------------------------------------

; ALP PROGRAM FOR BLOCK TRANSFER NON OVERLAP WITH STRING
INSTRUCTIONS
; ROLL : SYCOD272


section .data
sblock db 60h,70h, 80h, 90h,10h
dblock db 0h,0h,0h,0h,0h
mesg db "ALP PROGRAM FOR BLOCK TRANSFER NON OVERLAP WITH STRING
INSTRUCTIONS",10
mesg_len equ $-mesg
msg db "The source array is: "
msg_len equ $-msg
msg1 db 10, "The destination array is: "
msg1_len equ $-msg1
msg2 db "",10
msg2_len equ $-msg2
msg3 db "After block transfer, the arrays are :",10
msg3_len equ $-msg3
space db " "


%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

section .bss
charans resb 4
result resq 1

section .text
global _start
_start:

print mesg,mesg_len

print msg,msg_len
mov rsi,sblock
call Display_block

print msg1, msg1_len
mov rsi,dblock
call Display_block

print msg2,msg2_len

call Block_transfer

print msg3,msg3_len

print msg,msg_len
mov rsi,sblock
call Display_block

print msg1, msg1_len
mov rsi,dblock
call Display_block

print msg2,msg2_len

mov rax,60
mov rdi,0
syscall

Display:
```

```
mov rbx,16
mov rcx,2
mov rsi,charans+1

next:
mov rdx,0
div rbx
cmp dl,09h
jbe add30
add dl,07h

add30:
add dl,30h

mov [rsi], dl
dec rsi
dec rcx
jnz next
print charans,2
ret

Display_block :
mov rbp,5
 next_num:
 mov al,[rsi]
 push rsi
 call Display
 print space,1
 pop rsi
 inc rsi
 dec rbp
 jnz next_num
 ret

Block_transfer:
mov rsi,sblock
mov rdi,dblock
mov rcx,5

rep movsb
ret
```

ASSIGN 7 - OVERLAPPED

A) Without string instructions

```
section .data

sblock db 11h,12h,13h,14h,15h
dblock times 5 db 0

smsg db 10,"Source block is: "
smsg_len equ $-smsg
msg db 10,"After block transfer"
msg_len equ $-msg
dmsg db 10,"Destination block is: "
dmsg_len equ $-dmsg

space db " "


%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
mov rax,60
mov rdi,0
syscall
%endmacro

section .bss
charans resb 2

section .text
global _start
_start:
print smsg,smsg_len
mov rsi,sblock
call Display_block
```

```
print dmsg,dmsg_len
mov rsi,dblock-2
call Display_block

call Block_transfer

print msg,msg_len
print dmsg,dmsg_len
mov rsi,dblock-2
call Display_block

exit

Block_transfer:

mov rsi,sblock+4
mov rdi,dblock+2
mov rcx,5

back:
mov al,[rsi]
mov [rdi],al
dec rsi
dec rdi

dec rcx
jnz back
ret

Display:
mov rbx,16
mov rcx,2
mov rsi,charans+1

next:
mov rdx,0
div rbx
cmp dl,09h
jbe add30
add dl,07h

add30:
add dl,30h
mov [rsi], dl
dec rsi
dec rcx
jnz next
```

```
print charans,2
ret

Display_block :
mov rbp,5
 next_num:
 mov al,[rsi]
 push rsi
 call Display
 print space,1
 pop rsi
 inc rsi
 dec rbp
 jnz next_num
 ret
```

B) With string instructions

```
section .data

sblock db 11h,12h,13h,14h,15h
dblock times 5 db 0

smsg db 10,"Source block is: "
smsg_len equ $-smsg
msg db 10,"After block transfer"
msg_len equ $-msg
dmsg db 10,"Destination block is: "
dmsg_len equ $-dmsg

space db " "


%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
mov rax,60
mov rdi,0
syscall
%endmacro

section .bss
charans resb 2

section .text
global _start
_start:
print smsg,smsg_len
mov rsi,sblock
call Display_block

print dmsg,dmsg_len
mov rsi,dblock-2
call Display_block

call Block_transfer

print msg,msg_len
print dmsg,dmsg_len
mov rsi,dblock-2
call Display_block

exit

Block_transfer:

mov rsi,sblock+4
mov rdi,dblock+2
mov rcx,5
std
rep movsb
ret

Display:
mov rbx,16
mov rcx,2
mov rsi,charans+1

next:
```

```
mov rdx,0
div rbx
cmp dl,09h
jbe add30
add dl,07h

add30:
add dl,30h
mov [rsi], dl
dec rsi
dec rcx
jnz next
print charans,2
ret

Display_block :
mov rbp,5
 next_num:
 mov al,[rsi]
 push rsi
 call Display
 print space,1
 pop rsi
 inc rsi
 dec rbp
 jnz next_num
 ret
```

ASSIGN 8 - PROTECTED MODE

```
section .data
msg1 db "Processor is in protected mode",10
msg1_len equ $-msg1
msg2 db "Processor is not in protected mode",10
msg2_len equ $-msg2
mgdt db "Value of GDTR: "
mgdt_len equ $-mgdt
mline db "",10
mline_len equ $-mline
mldt db "Value of LDTR: "
mldt_len equ $-mldt
midt db "Value of IDTR: "
midt_len equ $-midt
mmsw db "Value of MSW: "
```

```
mmsw_len equ $-mmsw


%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
mov rax,60
mov rdi,0
syscall
%endmacro

section .bss
LDTR resw 1
GDTR resw 3
MSW resw 1
IDTR resw 3
TR resw 1
charans resb 4

section .text
global _start
_start:
SMSW [MSW]
mov ax,[MSW]
shr ax,1
jc pmode
print msg2,msg2_len

pmode: print msg1,msg1_len
jmp next
next:
SGDT [GDTR]
SLDT [LDTR]
STR [TR]
```

```
SIDT [IDTR]
SMSW [MSW]

print mgdt,mgdt_len
mov ax,[GDTR+4]
call display
mov ax,[GDTR+2]
call display
mov ax,[GDTR]
call display
print mline,mline_len
print mldt,mldt_len
mov ax,[LDTR]
call display
print mline,mline_len
print mmsw,mmsw_len
mov ax,[MSW]
call display

exit

display:
mov rbx,16
mov rcx,2
mov rsi,charans+1

next1:
mov rdx,0
div rbx
cmp dl,09h
jbe add30
add dl,07h

add30:
add dl,30h
mov [rsi], dl
dec rsi
dec rcx
jnz next1
print charans,2
ret
```