Lab Task - 04

Write an assembly program to print all the ASCII code from 0 to 255. Hints: use jnz and dec instructions

```
.model small
.stack 100H
.data
.code
main proc
   mov ah, 2
   mov cx, 256
   mov dl, 0
print_loop:
   int 21h
   inc dl
   dec cx
   jnz print_loop
exit:
   mov ah,4ch
   int 21h
   main endp
end main
```

Put the sum of the first 50 terms of the arithmetic sequence 1, 5, 9, 13, ... in DX. Hints: Employ LOOP instructions to do the following.

```
.model small
.stack 100H
.data
.code
main proc
   mov ax, 1
   mov bx, 0
   mov cx, 50
sum_loop:
   add bx, ax
   add ax, 4
   loop sum_loop
   mov dx, bx
exit:
   mov ah,4ch
   int 21h
   main endp
end main
```

Put the sum 100 + 95 + 90 + ... + 5 in AX. Hints: Employ LOOP instructions to do the following.

```
.model small
.stack 100H
.data
.code
main proc
   mov ax, 100
   mov bx, 0
   mov cx, 19
sum_loop:
   add bx, ax
   sub ax, 5
   loop sum_loop
   mov ax, bx
exit:
   mov ah,4ch
   int 21h
   main endp
end main
```

Read a character and display it 50 times on the next line. Hints: use LOOP instructions and put cx = 50

```
.model small

.stack 100H

.data

.code

main proc

   mov ah, 1

   int 21h

   mov bl, al


   mov ah, 2

   mov dl, 10

   int 21h

   mov dl, 13

   int 21h


   mov ah, 2

   mov cx, 50

   mov dl, bl


print_loop:

   int 21h

   loop print_loop


exit:

   mov ah,4ch

   int 21h
```

```
    main endp
end main
```

Lab Task – 05

Write a program to check whether a given input character is a vowel or not.

```
.model small
.stack 100h
.data

input_m db 'Enter a Character to check VOWEL : $'
is_v db ' is a VOWEL$'
not_v db ' is NOT a VOWEL$'

.code
main proc
    mov ax, @data
    mov ds, ax

    mov ah, 9
    lea dx, input_m
    int 21h

    mov ah, 1
    int 21h
    mov bl, al

    mov ah, 2
    mov dl, 10
    int 21h
    mov dl, 13
    int 21h

    cmp bl, 'a'
    je is_vowel
    cmp bl, 'e'
    je is_vowel
    cmp bl, 'i'
    je is_vowel
    cmp bl, 'o'
    je is_vowel
    cmp bl, 'u'
    je is_vowel

    cmp bl, 'A'
    je is_vowel
    cmp bl, 'E'
    je is_vowel
    cmp bl, 'I'
    je is_vowel
    cmp bl, 'O'
    je is_vowel
    cmp bl, 'U'
    je is_vowel

    not_vowel:
        mov ah, 2
        mov dl, bl
        int 21h

        mov ah, 9
        lea dx, not_v
        int 21h

        jmp exit



     is_vowel:
        mov ah, 2
        mov dl, bl
        int 21h

        mov ah, 9
        lea dx, is_v
        int 21h

    exit:
     mov ah,4ch
     int 21h
     main endp
end main
```

Take an input character from user. Check it for letter and convert upper to lower or lower to upper using logical instructions.

```
.model small

.stack 100h

.data

   input_m db 'Enter a character to convert to Upper or Lower: $'

   result_m db 'Converted Character: $'

   error_m db 'Invalid Character.$'

   input db ?

   output db ?


.code

main proc

   mov ax, @data

   mov ds, ax


   mov ah, 9

   lea dx, input_m

   int 21h


   mov ah, 1

   int 21h

   mov input, al


   cmp al, 'A'

   jl check_lowercase

   cmp al, 'Z'

   jg check_lowercase
```

```
    or al, 20h
    mov output, al
    jmp print_result

check_lowercase:
    cmp al, 'a'
    jl not_char
    cmp al, 'z'
    jg not_char

    and al, 5Fh
    mov output, al
    jmp print_result

not_char:
    mov ah, 2
    mov dl, 10
    int 21h
    mov dl, 13
    int 21h

    lea dx, error_m
    mov ah, 9
    int 21h
    jmp exit

print_result:
    mov ah, 2
    mov dl, 10
    int 21h
    mov dl, 13
    int 21h

    lea dx, result_m
    mov ah, 9
    int 21h

    mov dl, output
    mov ah, 2
    int 21h

exit:
    mov ah,4ch
    int 21h
    main endp
end main
```

Take an input character from user. Check it for number and find whether it is odd or even using TEST instruction.

```
.model small

.stack 100h

.data

   input_m db 'Enter a Number to Check EVEN or ODD: $'

   even_m db ' is EVEN.$'

   odd_m db ' is ODD.$'

   error_m db ' is not a NUMBER.$'

   input db ?


.code

main proc

   mov ax, @data

   mov ds, ax


   mov ah, 9

   lea dx, input_m

   int 21h


   mov ah, 1

   int 21h

   mov input, al


   mov ah, 2

   mov dl, 10

   int 21h

   mov dl, 13
```

```
    int 21h

    mov al, input
    cmp al, '0'
    jl not_number
    cmp al, '9'
    jg not_number

    sub al, '0'

    mov bl, al
    test al, 1
    jz even_number

odd_number:
    add bl, '0'
    mov dl, bl
    mov ah, 2
    int 21h

    lea dx, odd_m
    mov ah, 9
    int 21h
    jmp exit

even_number:
    add bl, '0'
    mov dl, bl
    mov ah, 2
    int 21h

    lea dx, even_m
    mov ah, 9
    int 21h
    jmp exit

not_number:
    mov dl, input
    mov ah, 2
    int 21h

    lea dx, error_m
    mov ah, 9
    int 21h

exit:
    mov ah,4ch
    int 21h
    main endp
end main
```

<u>Lab Task – 06</u>

Write an assembly language program for Binary Input and Output.

```
.model small
.stack 100h
.data
   input_msg db 'Enter a Binary Number : $'
   output_msg db 'The Binary Number is : $'
.code
main proc
   mov ax, @data
   mov ds, ax
   mov ah, 9
   lea dx, input_msg
   int 21h

   xor bx, bx
   mov cl, 8
next_bit:
   mov ah, 1
   int 21h
   cmp al, 0Dh
   je display_result
   sub al, '0'
   cmp al, 1
   ja invalid_input
   shl bx, 1
   or bl, al
   dec cl
```

```
    jz display_result
    jmp next_bit

invalid_input:
    mov ah, 9
    lea dx, output_msg
    int 21h
    mov ah, 4Ch
    int 21h

display_result:
    mov ah, 2
    mov dl, 10
    int 21h
    mov dl, 13
    int 21h
    mov ah, 9
    lea dx, output_msg
    int 21h

    mov cx, 8
print_bit:
    mov dx, bx
    shr dx, 7
    and dl, 1
    add dl, '0'
    mov ah, 2
    int 21h
    shl bx, 1
    loop print_bit


    exit:
        mov ah, 4Ch
        int 21h
        main endp
end main
```

Write an assembly language program for Hex input and Output.

```
.model small
.stack 100h
.data
    input_msg db 'Enter a Hexadecimal Number : $'
    output_msg db 'The Hexadecimal Number is : $'
.code
main proc
    mov ax, @data
    mov ds, ax


    mov ah, 9
    lea dx, input_msg
    int 21h
    xor bx, bx
next_digit:
    mov ah, 1
    int 21h
    cmp al, 0Dh
    je display_result
    shl bx, 4
    cmp al, 'A'
    jl digit_num
    sub al, 37h
    jmp store_digit

digit_num:
    sub al, '0'

store_digit:
    or bl, al
    jmp next_digit

display_result:
    mov ah, 2
    mov dl, 10
    int 21h
    mov dl, 13
    int 21h

    mov ah, 9
    lea dx, output_msg
    int 21h

    mov cx, 4
print_digit:
    mov dx, bx
    shr dx, 12
    and dl, 0Fh
    cmp dl, 9
    jg letter_hex
    add dl, '0'
    jmp print_char

letter_hex:
    add dl, 37h

print_char:
    mov ah, 2
    int 21h
    shl bx, 4
    loop print_digit

    exit:
        mov ah, 4Ch
        int 21h
        main endp
end main
```

Write an assembly language program that binary number to hexadecimal number.

```
.model small
.stack 100h
.data
   input_msg db 'Enter a Binary Number : $'
   output_msg db 'The Hexadecimal Number is : $'
.code

main proc
   mov ax, @data
   mov ds, ax

   mov ah, 9
   lea dx, input_msg
   int 21h

   xor bx, bx
next_bit:
   mov ah, 1
   int 21h
   cmp al, 0Dh
   je display_result
   sub al, '0'
   shl bx, 1
   or bl, al
   jmp next_bit
```

```
display_result:
    mov ah, 2
    mov dl, 10
    int 21h
    mov dl, 13
    int 21h

    mov ah, 9
    lea dx, output_msg
    int 21h

    mov cx, 4

print_digit:
    mov dx, bx
    shr dx, 12
    and dl, 0Fh
    cmp dl, 9
    jg letter_hex
    add dl, '0'
    jmp print_char

letter_hex:
    add dl, 37h

print_char:
    mov ah, 2
    int 21h
    shl bx, 4
    loop print_digit

    exit:
        mov ah, 4Ch
        int 21h
        main endp
end main
```

Write an assembly language program that hexadecimal number to binary number.

```
.model small
.stack 100h
.data
    input_msg db 'Enter a hexadecimal number (0-F): $'
    output_msg db 'The binary number is: $'
.code

main proc
    mov ax, @data
    mov ds, ax

    mov ah, 9
    lea dx, input_msg
    int 21h

    xor bx, bx
```

```
next_digit:
    mov ah, 1
    int 21h
    cmp al, 0Dh
    je display_result
    shl bx, 4
    cmp al, 'A'
    jl digit_num
    sub al, 37h
    jmp store_digit

digit_num:
    sub al, '0'

store_digit:
    or bl, al
    jmp next_digit

display_result:
    mov ah, 2
    mov dl, 10
    int 21h
    mov dl, 13
    int 21h

    mov ah, 9
    lea dx, output_msg
    int 21h
    mov cx, 16

print_binary_bit:
    mov dx, bx
    shr dx, 15
    and dl, 1
    add dl, '0'
    mov ah, 2
    int 21h
    shl bx, 1
    loop print_binary_bit

    exit:
        mov ah, 4Ch
        int 21h
main endp
end main
```

<u>Lab Task -07</u>

Suppose the register ax = 5, bx =6, Swap the numbers of ax and bx so that ax gets 6 and bx gets 5. use the concept of Stack. Push and Pop instructions must use.

```
.model small

.stack 100h

.data

.code


main proc

   mov ax, 5

   mov bx, 6


   push ax

   push bx


   pop ax

   pop bx


   exit:

   mov ah,4ch

   int 21h

   main endp

end main
```

Suppose that AX= 1234h, BX= 5678h, CX = 9ABCh, and SP= 1 00h. Write an assembly program to find out the contents of AX, BX, CX, and SP after executing the following instructions:

```
.model small

.stack 100h

.data

.code


main proc

   mov ax, 1234h

   mov bx, 5678h

   mov cx, 9ABCh

   mov sp, 100h


   push ax

   push bx

   xchg ax, cx

   pop cx

   push ax

   pop bx


   exit:

   mov ah,4ch

   int 21h

   main endp

end main
```

Reverse three characters 123, output should look like as follows: Hints, use the concept of push and pop.

```
.model small
.stack 100h
.data
   before db "Before Reverse: $"
   after db "After Reverse: $"
.code

main proc
   mov ax, @data
   mov ds, ax

   lea dx, before
   mov ah, 9
   int 21h

   xor cx, cx

read_input:
   mov ah, 1
   int 21h
   cmp al, 0Dh
   je done_input
   push ax
   inc cx
   jmp read_input
```

```
done_input:
    mov ah,2
    mov dl,10
    int 21h
    mov dl,13
    int 21h

    lea dx, after
    mov ah, 9
    int 21h

    mov cx, cx

print_reverse:
    pop ax
    mov dl, al
    mov ah, 2
    int 21h
    loop print_reverse

    exit:
    mov ah,4ch
    int 21h
    main endp
end main
```

Take a string from user. Once user hits enters reverse the string given by the user. Must take input from user. Hints: Use the concept of push and pop.

```
.model small
.stack 100h
.data
   msg1 db 'Before Reverse: $'
   msg2 db 'After Reverse: $'

.code
main proc
   mov ax, @data
   mov ds, ax

   lea dx, msg1
   mov ah, 09h
   int 21h

   mov ah, 2
   mov dl, 0Ah
   int 21h
   mov dl, 0Dh
   int 21h

   mov dl, '?'
   int 21h

   xor cx, cx
```

```
read_loop:
    mov ah, 1
    int 21h
    cmp al, 0Dh
    je reverse_loop

    push ax
    inc cx
    jmp read_loop

reverse_loop:

    mov ah, 2
    mov dl, 0Ah
    int 21h
    mov dl, 0Dh
    int 21h

    lea dx, msg2
    mov ah, 09h
    int 21h

    mov ah, 2
    mov dl, 0Ah
    int 21h
    mov dl, 0Dh
    int 21h

print_loop:
    pop ax
    mov dl, al
    mov ah, 2
    int 21h
    loop print_loop

    exit:
    mov ah,4ch
    int 21h
    main endp
end main
```

write a procedure named sub that subtract the variables and show print the value.

```
.model small
.stack 100h
.data
a db "Enter two values: $"
b db "Result: $"
.code

main proc
    mov ax,@data
    mov ds,ax
```

```
proc1 proc
    mov ah,9
    lea dx,a
    int 21h

    mov ah,1
    int 21h
    mov bl,al
    int 21h
    mov bh,al
    call proc2
    ret

proc2 proc
    mov ah,2
    mov dl,10
    int 21h
    mov dl,13
    int 21h

    mov ah,9
    lea dx,b
    int 21h
    sub bl,bh
    add bl,'0'
    mov ah,2
    mov dl,bl
    int 21h
```

```
        cmp bl,13
        jmp exit

    exit:
    mov ah,4ch
    int 21h
end main
```