**1] Write an ALP to input and display array elements**

**INPUT –**

%macro print 2

    mov eax, 4

    mov ebx, 1

    mov ecx, %1

    mov edx, %2

    int 80h

%endmacro

%macro read 2

    mov eax, 3

    mov ebx, 0

    mov ecx, %1

    mov edx, %2

    int 80h

%endmacro

section .data

    msg db 'Enter the elements of the array : ', 0xA

    plen equ $ - msg

    msg1 db 'The array elements are: '

    mlen equ $ - msg1

    space db ' '

    slen equ $ - space

    newline db 0xA

    nlen equ $ - newline

section .bss

    array resb 5

    num resb 2

section .text

    global \_start

\_start:

    mov ecx, 5

    mov esi, 0

    push ecx

    print msg, plen

    pop ecx

input\_loop:

    push ecx

    read num, 2

    mov al, [num]

    sub al, '0'

    mov [array + esi], al

    inc esi

    pop ecx

    loop input\_loop

    print msg1, mlen

    mov ecx, 5

    mov esi, 0

display\_loop:

    push ecx

    mov al, [array + esi]

    add al, '0'

    mov [num], al

    print num, 1

    cmp ecx, 1

    je skip\_space

    print space, slen

skip\_space:

    inc esi

    pop ecx

    loop display\_loop

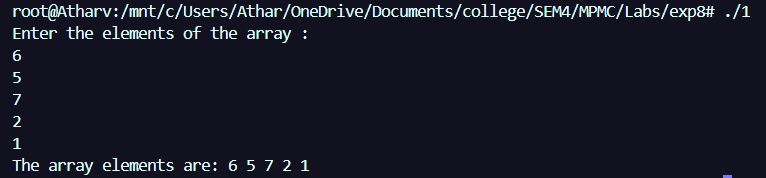
    print newline, nlen

    mov eax, 1

    mov ebx, 0

    int 80h

**OUTPUT –**

****

**2]Write an ALP to count number of positive and negative numbers in an array**

**INPUT-**

%macro print 2

    mov eax, 4

    mov ebx, 1

    mov ecx, %1

    mov edx, %2

    int 80h

%endmacro

%macro read 2

    mov eax, 3

    mov ebx, 0

    mov ecx, %1

    mov edx, %2

    int 80h

%endmacro

section .data

    msg db 'Enter the elements (with sign): '

    plen equ $ - msg

    pos\_msg db 'Positive numbers: '

    pos\_len equ $ - pos\_msg

    neg\_msg db 'Negative numbers: '

    neg\_len equ $ - neg\_msg

    newline db 0xA

    nlen equ $ - newline

section .bss

    array resb 5

    num resb 3

    pos\_count resb 1

    neg\_count resb 1

section .text

    global \_start

\_start:

    mov ecx, 5

    mov esi, 0

    mov byte[pos\_count], 0

    mov byte[neg\_count], 0

    push ecx

    print msg, plen

    print newline, nlen

    pop ecx

input\_loop:

    push ecx

    read num, 3

    mov al, [num]

    cmp al, '-'

    je negative

    sub al, '0'

    inc byte[pos\_count]

    jmp store

negative:

    mov al, [num + 1]

    sub al, '0'

    neg al

    inc byte[neg\_count]

store:

    mov [array + esi], al

    inc esi

    pop ecx

    loop input\_loop

    print pos\_msg, pos\_len

    mov al, [pos\_count]

    add al, '0'

    mov [num], al

    print num, 1

    print newline, nlen

    print neg\_msg, neg\_len

    mov al, [neg\_count]

    add al, '0'

    mov [num], al

    print num, 1

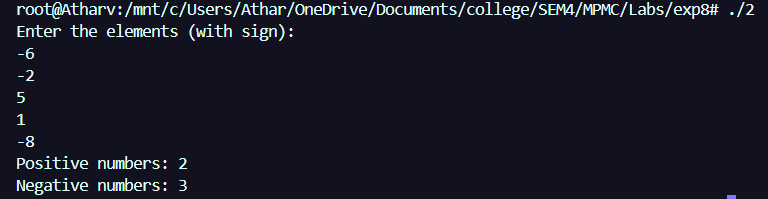
    print newline, nlen

    mov eax, 1

    mov ebx, 0

    int 80h

**OUTPUT –**

****

**3] Write an ALP to count number of even and odd numbers in the array**

**INPUT –**

%macro print 2

    mov eax, 4

    mov ebx, 1

    mov ecx, %1

    mov edx, %2

    int 80h

%endmacro

%macro read 2

    mov eax, 3

    mov ebx, 0

    mov ecx, %1

    mov edx, %2

    int 80h

%endmacro

section .data

    msg db 'Enter the elements of the array: ', 0xA

    plen equ $ - msg

    msg1 db 'Number of even numbers: '

    mlen1 equ $ - msg1

    msg2 db 'Number of odd numbers: '

    mlen2 equ $ - msg2

    newline db 0xA

    nlen equ $ - newline

section .bss

    array resb 5

    num resb 2

    even\_count resb 1

    odd\_count resb 1

section .text

    global \_start

\_start:

    mov ecx, 5

    mov esi, 0

    mov byte[even\_count], 0

    mov byte[odd\_count], 0

    push ecx

    print msg, plen

    pop ecx

input\_loop:

    push ecx

    read num, 2

    mov al, [num]

    sub al, '0'

    mov [array + esi], al

    inc esi

    pop ecx

    loop input\_loop

    mov ecx, 5

    mov esi, 0

count\_loop:

    mov al, [array + esi]

    mov ah, 0

    mov bl, 2

    div bl

    cmp ah, 0

    jz even\_number

    inc byte[odd\_count]

    jmp next\_number

even\_number:

    inc byte[even\_count]

next\_number:

    inc esi

    loop count\_loop

    print msg1, mlen1

    mov al, [even\_count]

    add al, '0'

    mov [num], al

    print num, 1

    print newline, nlen

    print msg2, mlen2

    mov al, [odd\_count]

    add al, '0'

    mov [num], al

    print num, 1

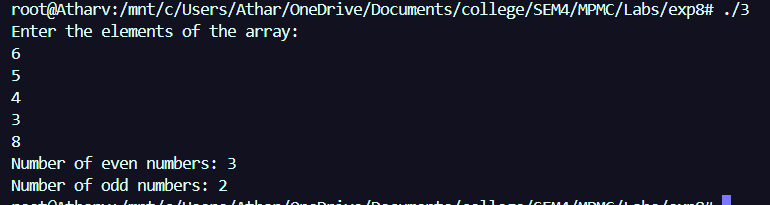
    print newline, nlen

    mov eax, 1

    mov ebx, 0

    int 80h

**OUTPUT –**

****

**4] Write an ALP to calculate elements less than 5 in the array**

**INPUT –**

%macro print 2

    mov eax, 4

    mov ebx, 1

    mov ecx, %1

    mov edx, %2

    int 80h

%endmacro

%macro read 2

    mov eax, 3

    mov ebx, 0

    mov ecx, %1

    mov edx, %2

    int 80h

%endmacro

section .data

    msg db 'Enter the elements of the array : ', 0xA

    plen equ $ - msg

    msg1 db 'Count of elements less than 5: '

    mlen equ $ - msg1

    msg2 db 'Count of elements greater than 5: '

    m2len equ $ - msg2

    msg3 db 'Count of elements equal to 5: '

    m3len equ $ - msg3

    newline db 0xA

    nlen equ $ - newline

section .bss

    array resb 5

    num resb 2

    countLess resb 1

    countGreater resb 1

    countEqual resb 1

section .text

    global \_start

\_start:

    mov ecx, 5

    mov esi, 0

    push ecx

    print msg, plen

    pop ecx

input\_loop:

    push ecx

    read num, 2

    mov al, [num]

    sub al, '0'

    mov [array + esi], al

    inc esi

    pop ecx

    loop input\_loop

    mov ecx, 5

    mov esi, 0

    mov byte[countLess], 0

    mov byte[countGreater], 0

    mov byte[countEqual], 0

count\_loop:

    mov al, [array + esi]

    cmp al, 5

    je equal\_to\_5

    jg greater\_than\_5

    inc byte[countLess]

    jmp continue\_count

greater\_than\_5:

    inc byte[countGreater]

    jmp continue\_count

equal\_to\_5:

    inc byte[countEqual]

continue\_count:

    inc esi

    loop count\_loop

    print msg1, mlen

    mov al, [countLess]

    add al, '0'

    mov [num], al

    print num, 1

    print newline, nlen

    print msg2, m2len

    mov al, [countGreater]

    add al, '0'

    mov [num], al

    print num, 1

    print newline, nlen

    print msg3, m3len

    mov al, [countEqual]

    add al, '0'

    mov [num], al

    print num, 1

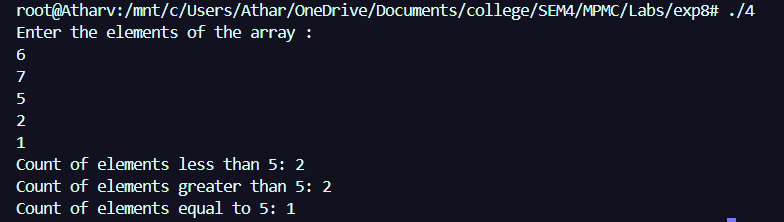
    print newline, nlen

    mov eax, 1

    mov ebx, 0

    int 80h

**OUTPUT –**

****

**5] Write an ALP to find the sum of elements of the array**

**INPUT –**

%macro print 2

    mov eax, 4

    mov ebx, 1

    mov ecx, %1

    mov edx, %2

    int 80h

%endmacro

%macro read 2

    mov eax, 3

    mov ebx, 0

    mov ecx, %1

    mov edx, %2

    int 80h

%endmacro

section .data

    msg db 'Enter the elements of the array: ', 0xA

    plen equ $ - msg

    msg1 db 'The array elements are: '

    mlen equ $ - msg1

    msg2 db 'Sum of elements: '

    slen equ $ - msg2

    space db ' '

    splen equ $ - space

    newline db 0xA

    nlen equ $ - newline

section .bss

    array resd 5

    num resb 6

    sum resd 1

    dispbuf resb 6

section .text

    global \_start

\_start:

    mov ecx, 5

    mov esi, 0

    push ecx

    print msg, plen

    pop ecx

input\_loop:

    push ecx

    read num, 6

    mov ebx, 0

    mov ecx, 0

convert:

    mov al, [num + ecx]

    cmp al, 0xA

    je done\_convert

    sub al, '0'

    imul ebx, 10

    add bl, al

    inc ecx

    jmp convert

done\_convert:

    mov [array + esi\*4], ebx

    add esi, 1

    pop ecx

    loop input\_loop

    mov ecx, 5

    mov esi, 0

    mov ebx, 0

sum\_loop:

    add ebx, [array + esi\*4]

    add esi, 1

    loop sum\_loop

    mov [sum], ebx

    print msg1, mlen

    mov ecx, 5

    mov esi, 0

display\_loop:

    push ecx

    mov eax, [array + esi\*4]

    mov ecx, dispbuf

    add ecx, 5

    mov byte [ecx], 0

    mov ebx, 10

convert\_to\_ascii:

    dec ecx

    xor edx, edx

    div ebx

    add dl, '0'

    mov [ecx], dl

    test eax, eax

    jnz convert\_to\_ascii

    print ecx, 6

    print space, splen

    add esi, 1

    pop ecx

    loop display\_loop

    print newline, nlen

    print msg2, slen

    mov eax, [sum]

    mov ecx, dispbuf

    add ecx, 5

    mov byte [ecx], 0

    mov ebx, 10

convert\_sum:

    dec ecx

    xor edx, edx

    div ebx

    add dl, '0'

    mov [ecx], dl

    test eax, eax

    jnz convert\_sum

    print ecx, 6

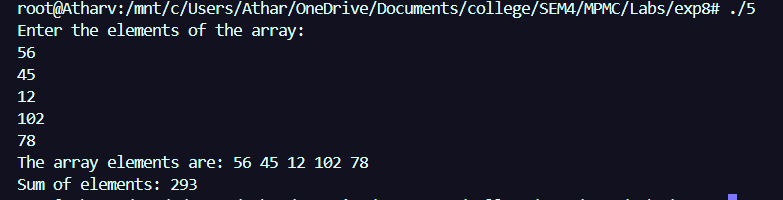
    print newline, nlen

    mov eax, 1

    mov ebx, 0

    int 80h

**OUTPUT –**

****

**CONCLUSION –**

**Array operations for inputing ,display and counting were successfully implemented using NASM .**