Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year:2025), B.Sc. in CSE (Day)

Lab Report NO : 4

Course Title: Microprocessor and Microcontroller lab Course Code:

CSE304 Section: 231D4

Lab Experiment Name: Implement Procedure in Assembly Language Programming & Implement Macro in Assembly Language Programming

Student Details

Name	ID
Promod Chandra Das	231002005

Lab Date : 21/04/2025

Submission Date : 28/04/2025

Course Teacher's Name : Md. Romzan Alom

Lab Report Status

Marks: …… **Signature:.....**

Comments:..... **Date:.....**

✧ Introduction:

In assembly language programming, **procedures** and **macros** are essential tools that help organize and simplify code. A **procedure** is a block of reusable code that performs a specific task and can be called from different parts of a program using the CALL instruction, returning with RET. It promotes modular programming and reduces redundancy. On the other hand, a **macro** is a code template that expands at compile time, allowing commonly used code patterns to be reused without the overhead of a procedure call. Both features make assembly programs easier to manage, more readable, and efficient in execution.

✧ Objective:

The objective of this study is to understand and implement **procedures** and **macros** in assembly language programming. It aims to highlight how procedures promote modularity and code reuse by allowing defined blocks of code to be called multiple times, while macros simplify repetitive coding tasks by enabling inline code expansion. By learning these concepts, students will be able to write more organized, efficient, and readable assembly programs, making the development process easier and more maintainable.

❖ Problem 1:

1. Write an assembly language code to take natural number series as input, and as output show the following
using two different procedures:
i. The summation of odd numbers.
ii. The summation of even numbers.
Code:

```
.model small
.stack 100h
.data
sum_odd dw 0
sum_even dw 0
.code
main:
    mov ax, @data
    mov ds, ax
    mov cx, 5
input:
    mov ah, 1
    int 21h
    sub al, '0'
    test al, 1
    jz even
    add sum_odd, ax
    jmp next
even:
    add sum_even, ax
next:
    loop input

    ; Print sum_odd
    mov ax, sum_odd
    call print
    ; Print sum_even
    mov ax, sum_even
    call print

    mov ah, 4ch
    int 21h
```

```
print proc
    mov bx, 10
    xor cx, cx
div_loop:
    xor dx, dx
    div bx
    push dx
    inc cx
    test ax, ax
    jnz div_loop
print_loop:
    pop dx
    add dl, '0'
    mov ah, 2
    int 21h
    loop print_loop
    ret
print endp

end main
```

Problem 2: Average, Largest, and Smallest of 5 Numbers

```
.model small
.stack 100h
.data
arr db 5 dup(?), avg db 0, large db 0, small
db 9
.code
main:
    mov ax, @data
    mov ds, ax
    lea si, arr
    mov cx, 5
input_loop:
    mov ah, 1
    int 21h
    sub al, '0'
    mov [si], al
    inc si
    loop input_loop

    call calculate_avg
    call find_largest
    call find_smallest

    mov ah, 09h
    lea dx, result_msg
    int 21h
    mov al, avg
    call print_num
    lea dx, large_msg
    int 21h
    mov al, large
    call print_num
    lea dx, small_msg
    int 21h
    mov al, small
    call print_num

    mov ah, 4Ch
    int 21h
calculate_avg proc
    lea si, arr
    xor bx, bx
    mov cx, 5
avg_loop:
    mov al, [si]
    add bx, ax
    inc si
    loop avg_loop
    mov ax, bx
    mov cx, 5
    div cx
    mov avg, al
    ret
calculate_avg endp
```

```
find_largest proc
    lea si, arr
    mov al, [si]
    mov large, al
    inc si
    mov cx, 4
largest_loop:
    mov al, [si]
    cmp al, large
    jg set_large
    inc si
    loop largest_loop
    ret
set_large:
    mov large, al
    inc si
    loop largest_loop
    ret
find_largest endp

find_smallest proc
    lea si, arr
    mov al, [si]
    mov small, al
    inc si
    mov cx, 4
smallest_loop:
    mov al, [si]
    cmp al, small
    jl set_small
    inc si
    loop smallest_loop
    ret
set_small:
    mov small, al
    inc si
    loop smallest_loop
    ret
find_smallest endp

print_num proc
    add al, '0'
    mov dl, al
    mov ah, 02h
    int 21h
    ret
print_num endp

.end main
```

❖ **Problem 3: Rearranging 7 Digits in Ascending and Descending Order**
Code:

```
.model small
.stack 100h
.data
arr db 7 dup(?)
.code
main:
    mov ax, @data
    mov ds, ax
    lea si, arr
    mov cx, 7
input_loop:
    mov ah, 1
    int 21h
    sub al, '0'
    mov [si], al
    inc si
    loop input_loop

    call sort_ascending
    lea dx, .asc_msg
    mov ah, 09h
    int 21h
    call print_array

    call sort_descending
    lea dx, .desc_msg
    mov ah, 09h
    int 21h
    call print_array

    mov ah, 4Ch
    int 21h

sort_ascending proc
    lea si, arr
    mov cx, 6
asc_loop:
    lea di, arr
    mov al, [di]
    cmp al, [di+1]
    jg swap_asc
    inc di
    loop asc_loop
    ret
swap_asc:
    mov al, [di]
    xchg al, [di+1]
    mov [di], al
    jmp asc_loop
sort_ascending endp
```

```
sort_descending proc
    lea si, arr
    mov cx, 6
desc_loop:
    lea di, arr
    mov al, [di]
    cmp al, [di+1]
    jl swap_desc
    inc di
    loop desc_loop
    ret
swap_desc:
    mov al, [di]
    xchg al, [di+1]
    mov [di], al
    jmp desc_loop
sort_descending endp

print_array proc
    lea si, arr
    mov cx, 7
print_loop:
    mov al, [si]
    call print_num
    inc si
    loop print_loop
    ret
print_array endp

print_num proc
    add al, '0'
    mov dl, al
    mov ah, 02h
    int 21h
    ret
print_num endp

.end main
```

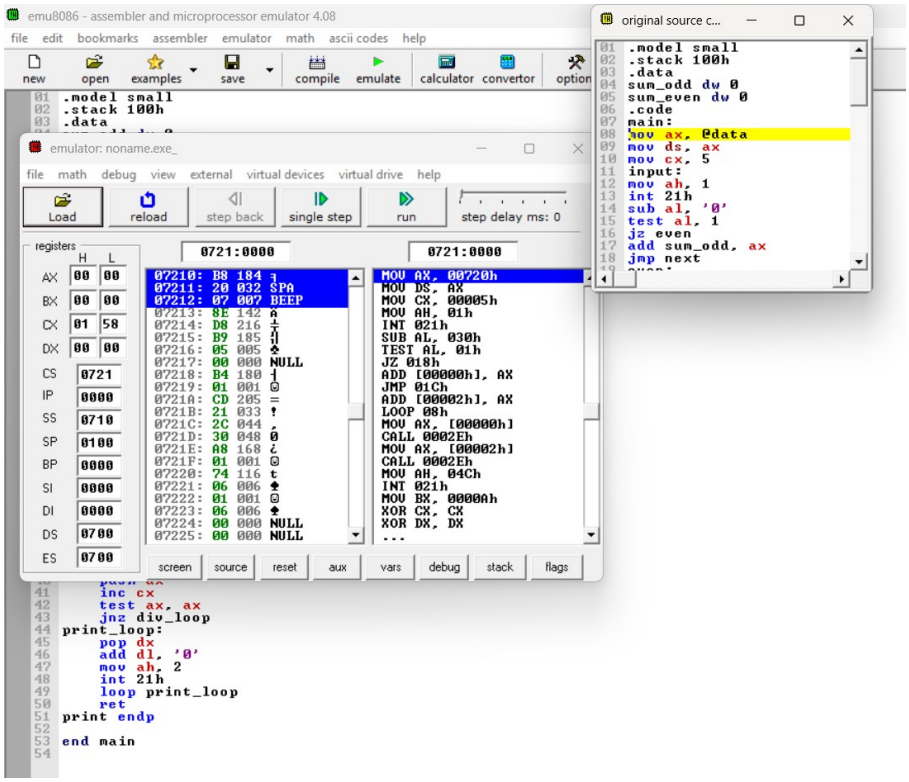
❖ **Problem 4:**

Write an Assembly Language code that takes an input ARRAY and passes the array values and address to a MACRO. Now produce the summation of odd digits and even digits as output.

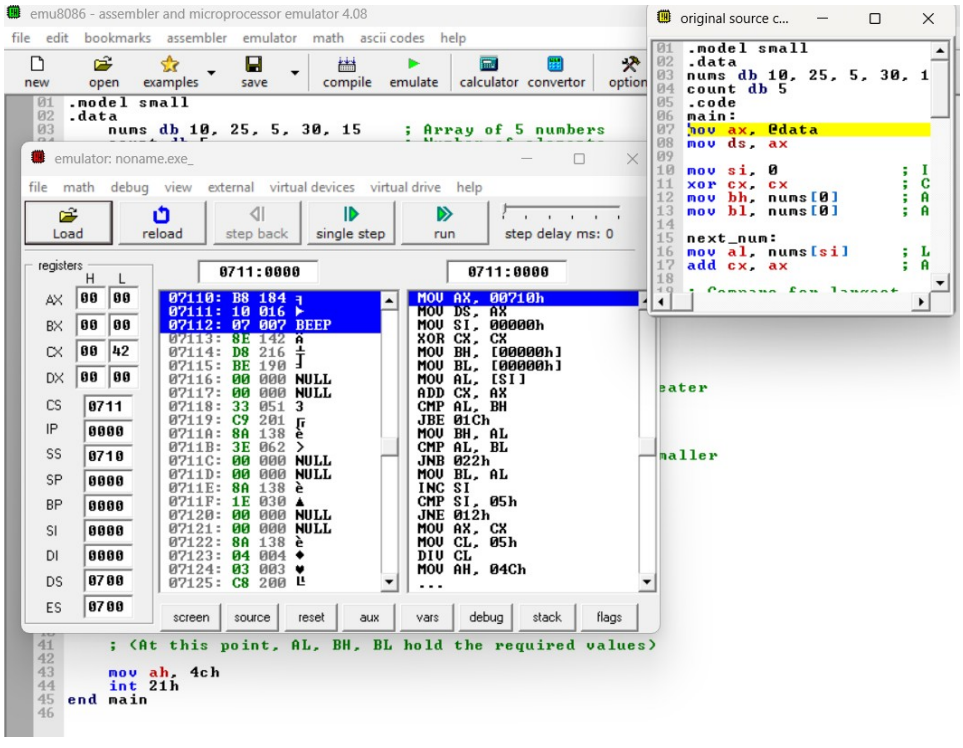
```
.model small
.stack 100h
.data
arr db 3, 1, 4, 5, 1, 6, 8, 7
odd_sum db 0
even_sum db 0
.code

; MACRO to calculate odd and even sum
SumOddEven MACRO arr, oddSum, evenSum
    lea si, arr
    mov al, [si]
    cmp al, 0
    je done
    ; Check for odd or even
    test al, 1
    jz even
odd:
    add oddSum, al
    jmp next
even:
    add evenSum, al
next:
    inc si
    mov al, [si]
    jmp $-8
done:
ENDM

main:
    mov ax, @data
    mov ds, ax

    ; Call the macro
    SumOddEven arr, odd_sum, even_sum

    ; Display the results
    lea dx, odd_msg
    mov ah, 09h
    int 21h
    mov al, odd_sum
    call PrintNum
```

```
 lea dx, even_msg
    mov ah, 09h
    int 21h
    mov al, even_sum
    call PrintNum

    ; Exit program
    mov ah, 4Ch
    int 21h

PrintNum proc
    add al, '0'
    mov dl, al
    mov ah, 02h
    int 21h
    ret
PrintNum endp

odd_msg db 'ODD Digits: $'
even_msg db 13, 10, 'EVEN Digits: $'

.end main
```

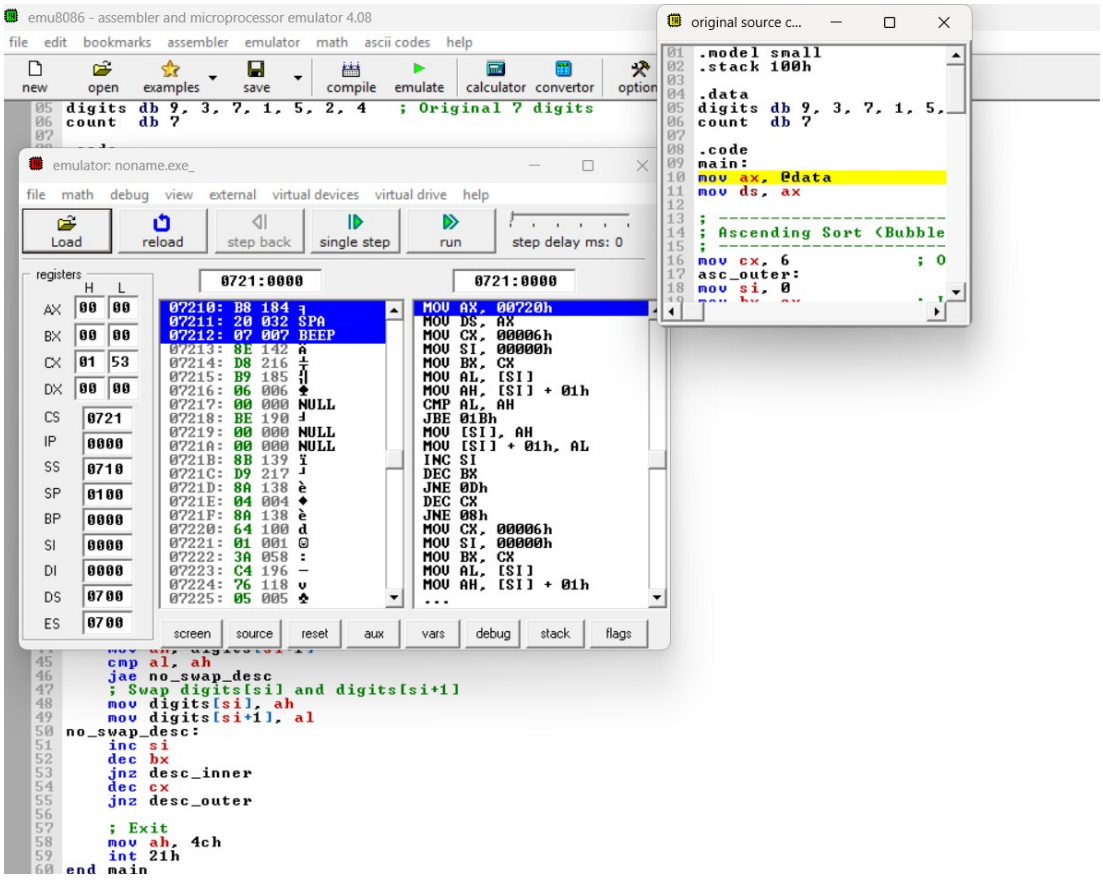
- **Problem 1(Output):**



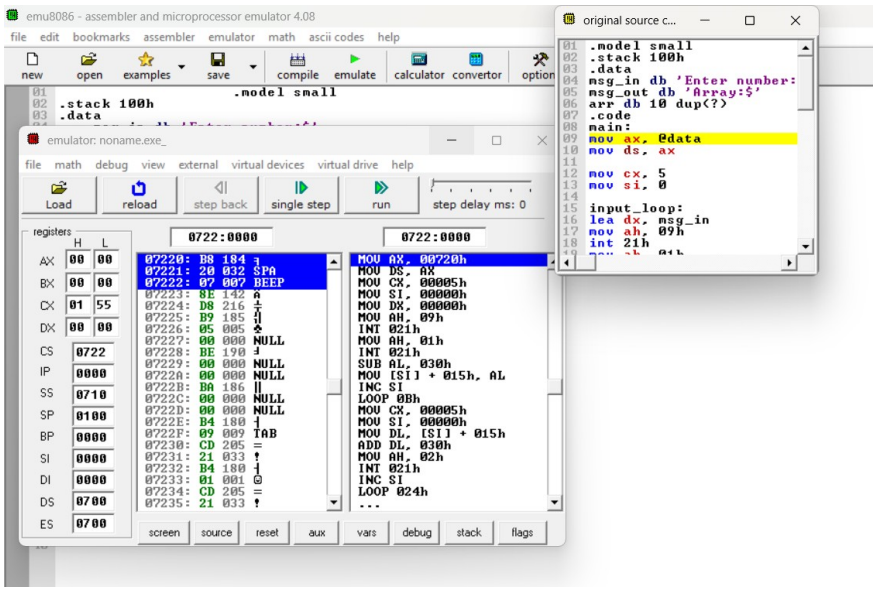
- **Problem 2(Output):**



- **Problem 3(Output):**



- **Problem 4(Output):**



Discussion:

Procedures in Assembly are reusable code blocks that perform a specific task. They improve code organization and reusability, and manage control flow with `CALL` and `RET`. However, they introduce overhead due to function calls and stack management.

Macros, on the other hand, are code templates expanded inline at compile time, reducing execution overhead. They are faster since there's no call mechanism, but they increase code size due to duplication. Procedures are ideal for complex or repetitive tasks, while macros excel in optimizing small, repetitive code sections. Both serve different purposes, balancing flexibility, performance, and code size.