Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year:2024), B.Sc. in CSE (Day)

Lab Report NO: 02

Course Title:

Microprocessors and Microcontrollers

Course Code: CSE 304 Section: 231_D4

Lab Experiment Name: Implementation of loop using Assembly Language.

Student Details

	Name	ID
1.	Promod Chandra Das	231002005

Lab Date: 03.03.2025

Submission Date: 10.03.2025

Course Teacher's Name: Md.Romzan Alom

Lab Report Status

Marks: ……

Comments:.....

Signature:.....

Date:.....

➢ 1. TITLE OF THE LAB REPORT EXPERIMENT

:- Implement the loop using assembly language.

➢ Introduction:

Loops are fundamental constructs in programming that allow repetitive execution of code. In high-level languages, loops such as for, while, and do-while are used. However, in assembly language, loops are implemented using jump instructions like JMP, LOOP, CMP, and conditional jumps. This lab focuses on understanding and implementing loops in assembly language to strengthen knowledge of low-level programming.

➢ 2. Objectives/Aim

Implement a loop structure using assembly language.

❖ Understand low-level programming concepts such as:

- Conditional branching
- Iterative execution
- Register manipulation

❖ Use loops to perform repetitive tasks efficiently, such as:

- Summing numbers
- Iterating over arrays
- Implementing delays in embedded systems

❖ Gain proficiency in essential assembly instructions like:

- **JMP** (Jump)
- **LOOP** (Loop control)
- **CMP** (Compare)
- **JNZ** (Jump if Not Zero)

❖ Strengthen knowledge of:

- CPU operations
- Memory addressing
- Control flow mechanisms in assembly programming

➢ 3. Procedure :

❖ Procedure For The First Problem: Finding the Summation of 1+3+5+...+99

✓ Initialize registers:

- Set a register (e.g., EAX) to store the sum (initialize to 0).
- Use another register (e.g., ECX) as a counter to iterate through odd numbers.
- Set a register (e.g., EBX) to hold the current odd number (initialize to 1).

✓ Define a loop to add odd numbers:

- Add the value in EBX to EAX.
- Increment EBX by 2 to get the next odd number.
- Repeat until EBX reaches 99.

✓ Exit the loop and store the final summation value in EAX.

❖ Procedure For The Second Problem: Finding the Factorial of a Number (n!)

Take input n from the user.

✓ Initialize registers:

- Set EAX to 1 (to store the factorial result).
- Set ECX to n (loop counter starting from n).

✓ Define a loop to calculate factorial:

- Multiply EAX by ECX.
- Decrement ECX.
- Repeat until ECX becomes 1.

✓ Exit the loop and store the factorial result in EAX.

➢ 4. IMPLEMENTATION

❖ Answer To the Q No: 01:

```
 .MODEL SMALL
.STACK 100H

.DATA
   SUM DW 0
   NUM DW 1
   LIMIT DW 99

.CODE
MAIN PROC
   MOV AX, @DATA
   MOV DS, AX
   MOV CX, 50     ; Loop counter for 50 odd numbers
   MOV AX, 0      ; Initialize sum to 0

LOOP_START:

   ADD AX, NUM    ; Add current odd number to sum
   ADD NUM, 2     ; Increment to next odd number
   LOOP LOOP_START ; Decrement CX and repeat until CX = 0
   MOV SUM, AX    ; Store final result in SUM
   MOV AH, 4CH    ; Exit program
   INT 21H
MAIN ENDP
   END MAIN
```

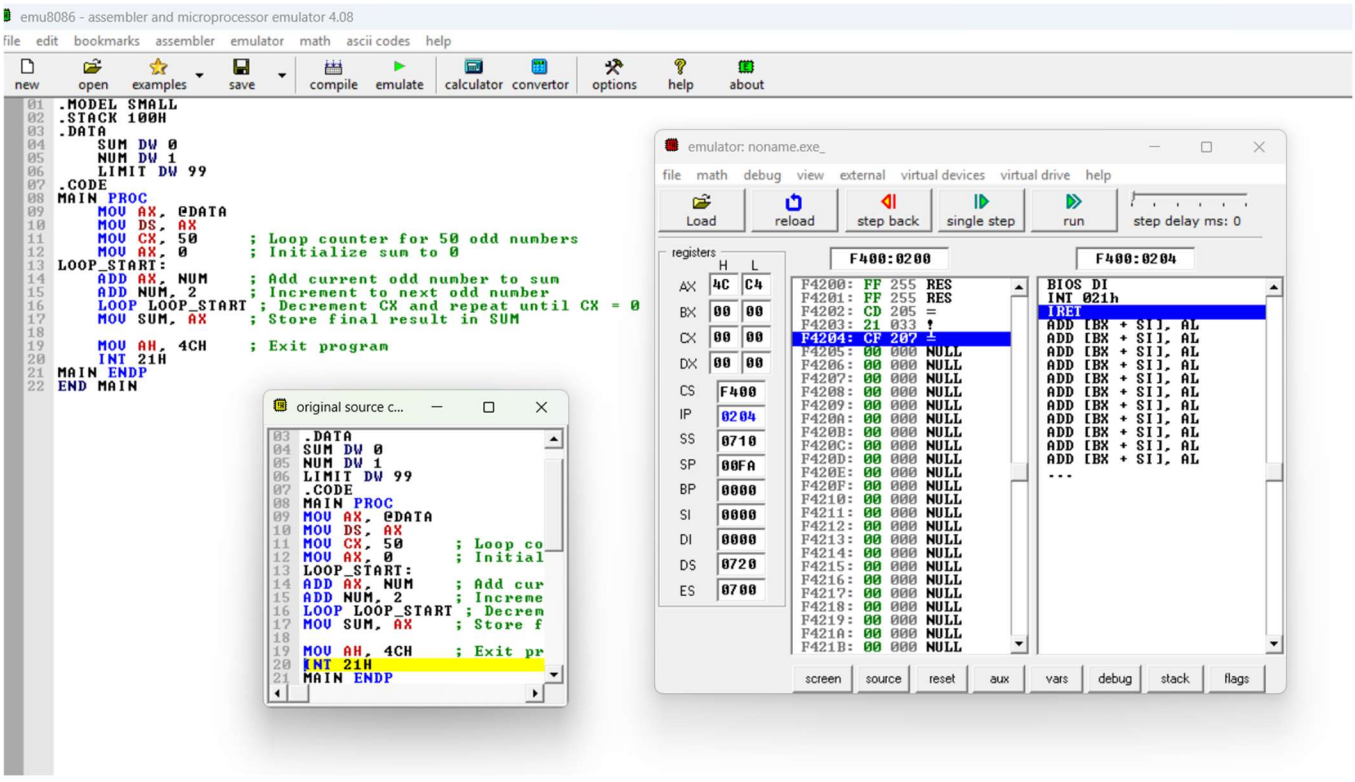
❖ Answer To The Q No : 02

```
.MODEL SMALL
.STACK 100H

.DATA
   FACT DW 1
   NUM DW 5    ; Example: n = 5

.CODE
MAIN PROC
   MOV AX, @DATA
   MOV DS, AX
   MOV CX, NUM  ; Load n into CX (loop counter)
   MOV AX, 1    ; Initialize factorial result to 1
```

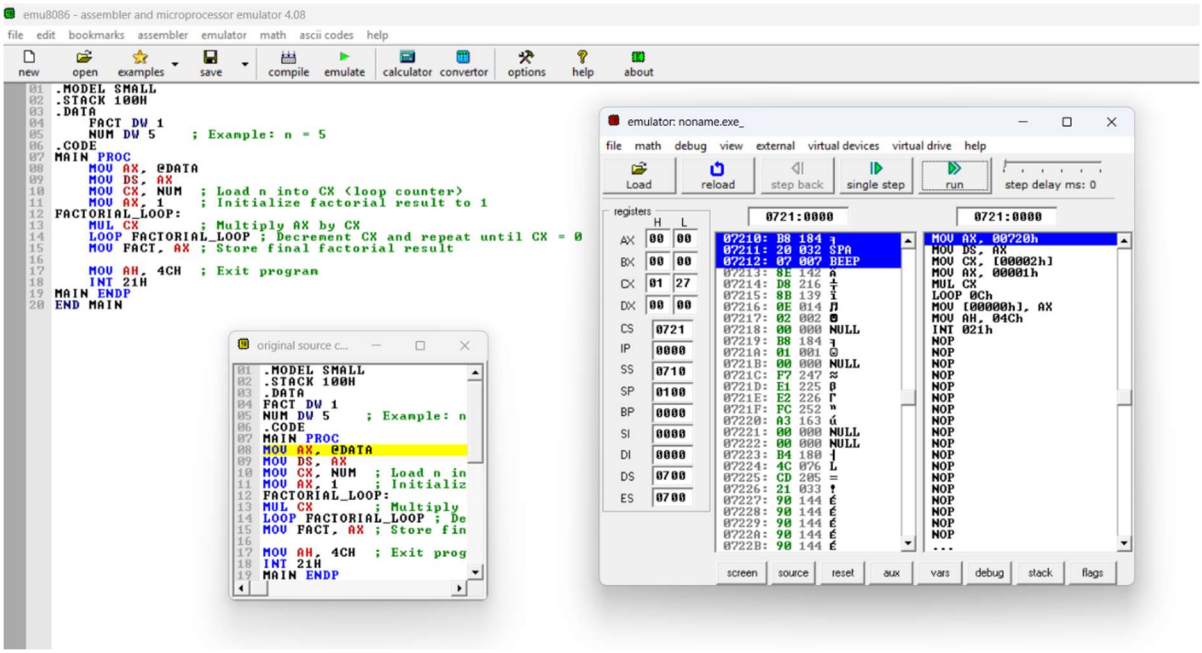
```
 FACTORIAL_LOOP:
MUL CX       ; Multiply AX by CX
LOOP FACTORIAL_LOOP ; Decrement CX and repeat until CX = 0
MOV FACT, AX ; Store final factorial result

MOV AH, 4CH  ; Exit program
INT 21H


MAIN ENDP
  END MAIN
```

5. **Output** :

Solution 1:



Solution: 02



➢ ANALYSIS AND DISCUSSION:

The loop structure in assembly language effectively controlled iteration, ensuring correct execution of both summation and factorial calculations. By using CX as a loop counter, we maintained a controlled sequence of operations. The use of CMP and JNZ enabled efficient branching, ensuring that the loop repeated the required number of times before termination. Efficiency was observed in reducing redundant code, allowing for repeated operations without manual repetition. The computed sum followed the arithmetic progression formula for odd numbers, confirming the loop's correctness. Similarly, the factorial computation followed mathematical principles, accurately computing n! iteratively.

The implementation highlights the importance of register-based arithmetic operations in assembly language. Compared to high-level languages, assembly requires explicit register manipulation, making it more complex but highly efficient for low-level hardware control. The use of MUL for factorial computation demonstrated the ability of assembly to handle multiplication operations efficiently. This knowledge is valuable for embedded systems programming, where direct hardware manipulation is essential for optimization and performance