Course Name:	**Microprocessors and Microcontrollers Laboratory**	**Semester:**	**IV**
Date of Performance:	**5 / 02 / 2024**	**Batch No:**	**A - 2**
Faculty Name:	**Kirti Sawlani**	**Roll No:**	**16014022050**
Faculty Sign & Date:		**Grade / Marks:**	**___ / 25**

Experiment No: 2
Title: Generation of Fibonacci Series

Aim and Objective of the Experiment:

Write an 8086 based Assembly Language Program to find the first 10 Fibonacci series numbers and store them in the data segment.

COs to be achieved:

CO2: Develop 8086 based assembly language programs.

Theory:

To study basic instructions and addressing modes of 8086. Understand assembler directives and concept of data and code segment

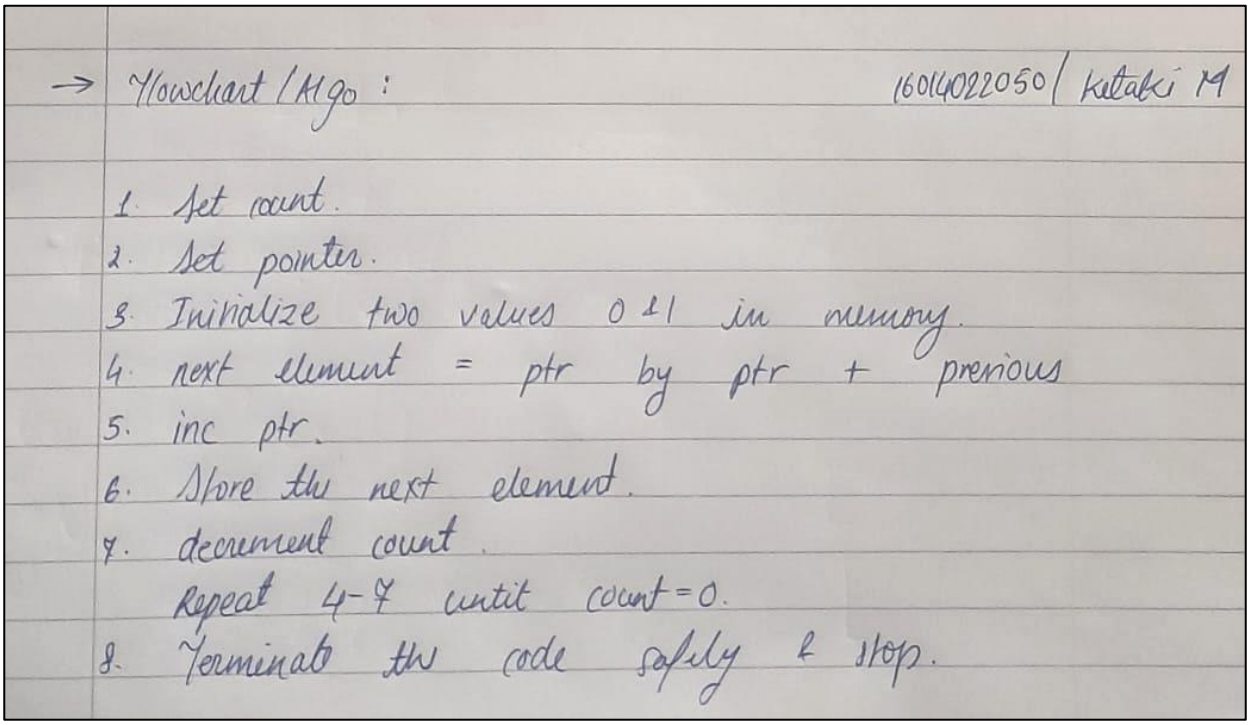
This experiment covers following instructions groups.

a. Data transfer
b. Arithmetic (Multiply Instructions)
c. Branch instructions

Stepwise-Procedure:

1. Open EMU8086 and write your ASM code in the empty workspace.
2. Click on emulate button and it should open the emulator window.
3. You can run the code using single step execution and monitor the internal registers / flags.

Algorithm / Flowchart:



Assembly Language Program:

Write comments on List file and copy paste the list file contents here.

EMU8086 GENERATED LISTING. MACHINE CODE <- SOURCE.

noname.exe_ -- emu8086 assembler version: 4.08

[05-02-2024 -- 17:00:23]

==========

[LINE] LOC: MACHINE CODE SOURCE

==========

```
[  1]      :                        ; MPMC LAB - Experiment 3 | 16014022050
[  2]      :
[  3]      :                        data segment
[  4]      :                        ; Define a byte variable 'cnt' with an initial value of 10
[  5]   0000: 0A                        cnt db 10
[  6]      :
[  7]      :                        ; Allocate space for a byte array named 'fibo' with 12 elements, initialized to zero
[  8]   0001: 00 00 00 00 00 00 00 00 00 00 00 00   fibo db 12 dup(00)

[  9]      :
[ 10]      :                        ; End of the data segment
[ 11]      :                        data ends
[ 12]      :
[ 13]      :                        code segment
[ 14]      :                        ; Set up the segment registers
[ 15]      :                        assume cs: code, ds: data
[ 16]      :
[ 17]   0010:                          start:
[ 18]      :                        ; Load the data segment address into the AX register
[ 19]   0010: B8 00 00                  mov ax, data
[ 20]      :                        ; Set DS to the address of the data segment
[ 21]   0013: 8E D8                     mov ds, ax
[ 22]      :
[ 23]      :                        ; Load the address of the 'fibo' array into the SI register
[ 24]   0015: BE 01 00                  lea si, fibo
[ 25]      :
[ 26]      :                        ; Initialize the loop counter CL with the value of 'cnt'
[ 27]   0018: 8A 0E 00 00               mov cl, cnt
[ 28]      :                        ; Clear CH to ensure that the loop counter is 16 bits wide
[ 29]   001C: B5 00                     mov ch, 0
[ 30]      :                        ; Store 0 in the first element of the 'fibo' array
[ 31]   001E: B0 00                     mov al, 0
[ 32]   0020: 88 04                    mov [si], al
[ 33]      :
[ 34]      :                        ; Point to the next element in the 'fibo' array
[ 35]   0022: 46                        inc si
[ 36]      :
```

```
[ 37]      :                    ; Store 1 in the second element of the 'fibo' array
[ 38]   0023: B0 01                mov al, 01
[ 39]   0025: 88 04                mov [si], al
[ 40]      :
[ 41]      :                    ; Start of the Fibonacci sequence calculation loop
[ 42]   0027:                      up:
[ 43]      :                    ; Load the current element of the Fibonacci sequence into AL
[ 44]   0027: 8A 04                mov al, [si]
[ 45]      :                    ; Add the previous element to the current element
[ 46]   0029: 02 44 FF              add al, [si - 1]
[ 47]      :                    ; Adjust the result for BCD addition
[ 48]   002C: 27                   daa
[ 49]      :                    ; Move to the next position in the 'fibo' array
[ 50]   002D: 46                   inc si
[ 51]      :                    ; Store the calculated Fibonacci number
[ 52]   002E: 88 04                mov [si], al
[ 53]      :                    ; Decrement the loop counter and continue if it's not zero
[ 54]   0030: E2 F5                loop up
[ 55]      :
[ 56]      :                    ; Terminate the program
[ 57]   0032: B4 4C                mov ah, 4ch
[ 58]   0034: CD 21                int 21h
[ 59]      :
[ 60]      :                    ; End of the code segment
[ 61]      :                    code ends
[ 62]      :
[ 63]      :                    ; Program entry point
[ 64]      :                    end start
[ 65]      :
```

==========

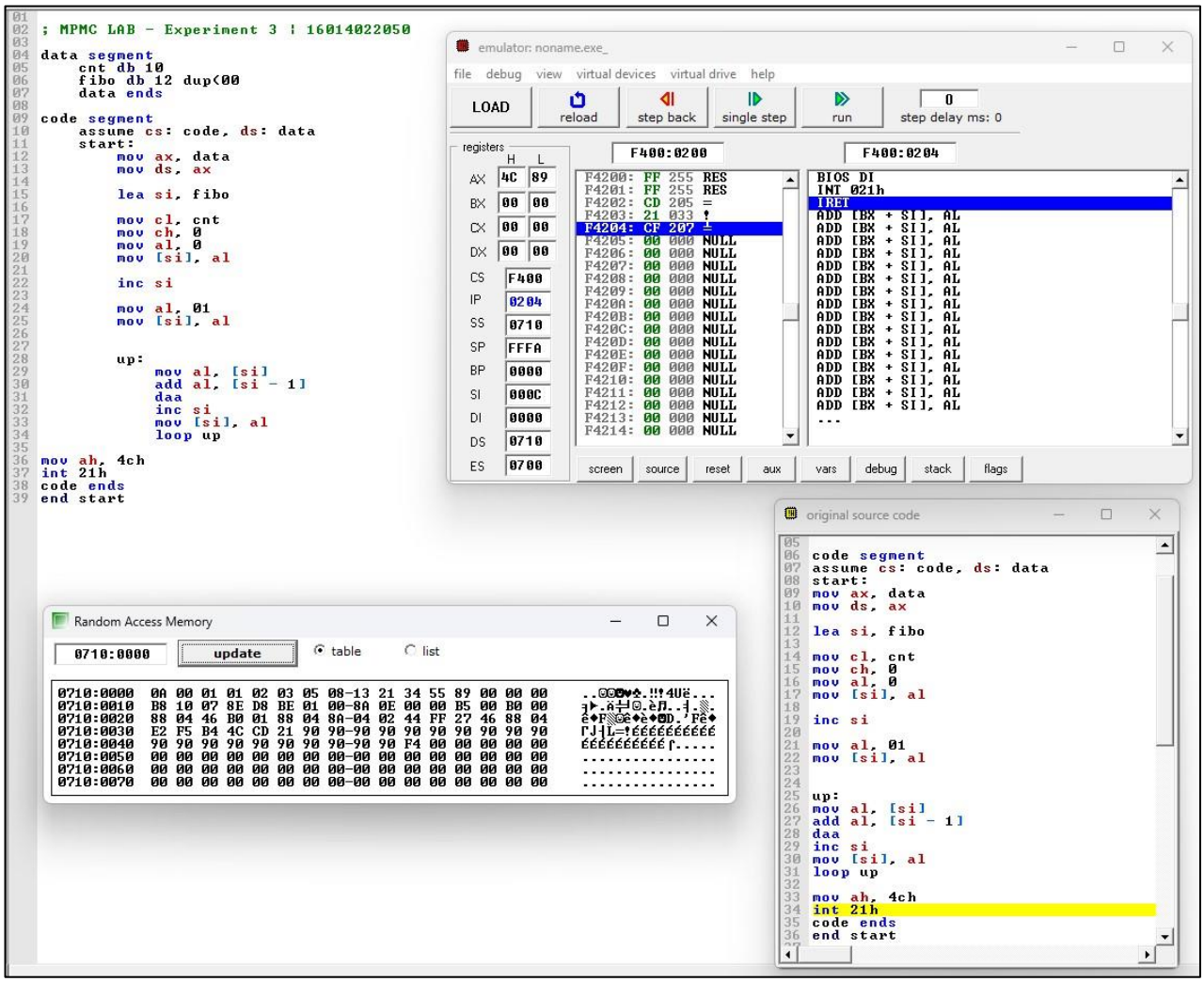
EXE HEADER - bytes from 0000 to 01FF inclusive.

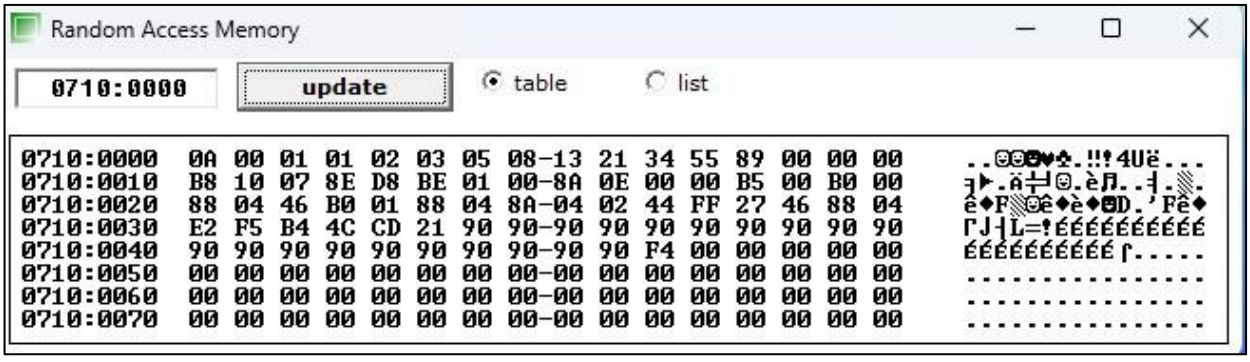
0000: 4D - exe signature (M)
0001: 5A - exe signature (Z)
0002: 36 - bytes on last page (l.byte)

0003: 00 - bytes on last page (h.byte)
0004: 02 - 512 byte pages in file (l.byte)
0005: 00 - 512 byte pages in file (h.byte)
0006: 01 - relocations (l.byte)
0007: 00 - relocations (h.byte)
0008: 20 - paragraphs in header (l.byte)
0009: 00 - paragraphs in header (h.byte)
000A: 00 - minimum memory (l.byte)
000B: 00 - minimum memory (h.byte)
000C: FF - maximum memory (l.byte)
000D: FF - maximum memory (h.byte)
000E: 00 - SS - stack segment (l.byte)
000F: 00 - SS - stack segment (h.byte)
0010: 00 - SP - stack pointer (l.byte)
0011: 00 - SP - stack pointer (h.byte)
0012: 81 - check sum (l.byte)
0013: D0 - check sum (h.byte)
0014: 00 - IP - instruction pointer (l.byte)
0015: 00 - IP - instruction pointer (h.byte)
0016: 01 - CS - code segment (l.byte)
0017: 00 - CS - code segment (h.byte)
0018: 1E - relocation table adress (l.byte)
0019: 00 - relocation table adress (h.byte)
001A: 00 - overlay number (l.byte)
001B: 00 - overlay number (h.byte)
001C: 01 - signature (l.byte)
001D: 00 - signature (h.byte)
001E: 01 - relocation table - offset inside segment (l.byte)
001F: 00 - relocation table - offset inside segment (h.byte)
0020: 01 - relocation table - segment anchor (l.byte)
0021: 00 - relocation table - segment anchor (h.byte)
0022 to 01FF - reserved relocation area (00)

==========

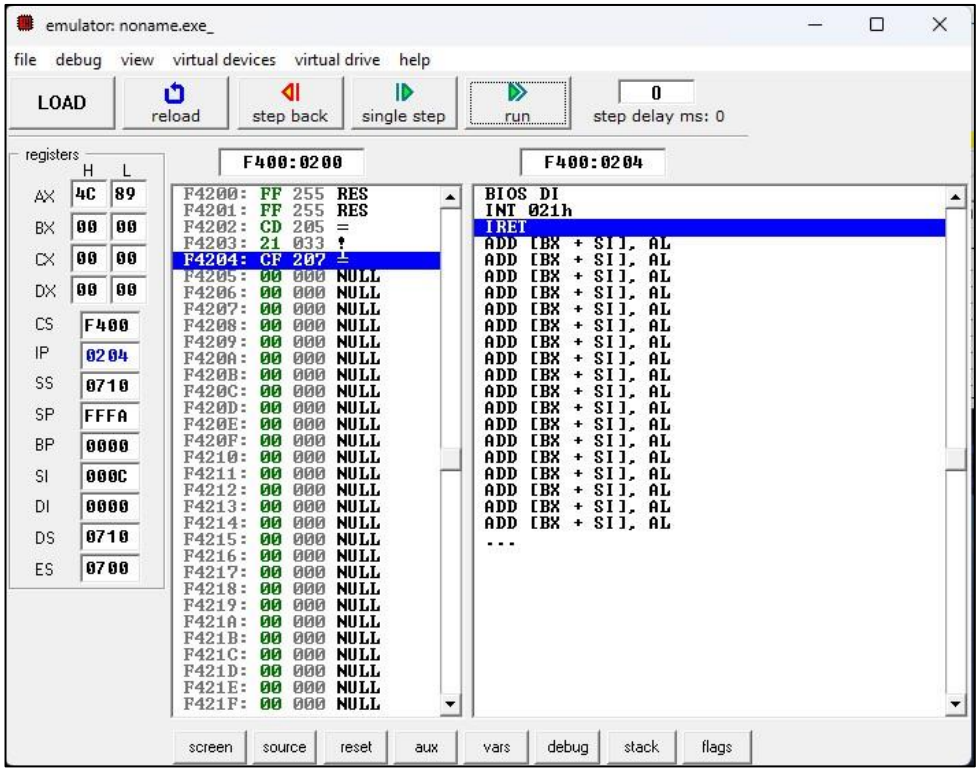
Output Screenshots:





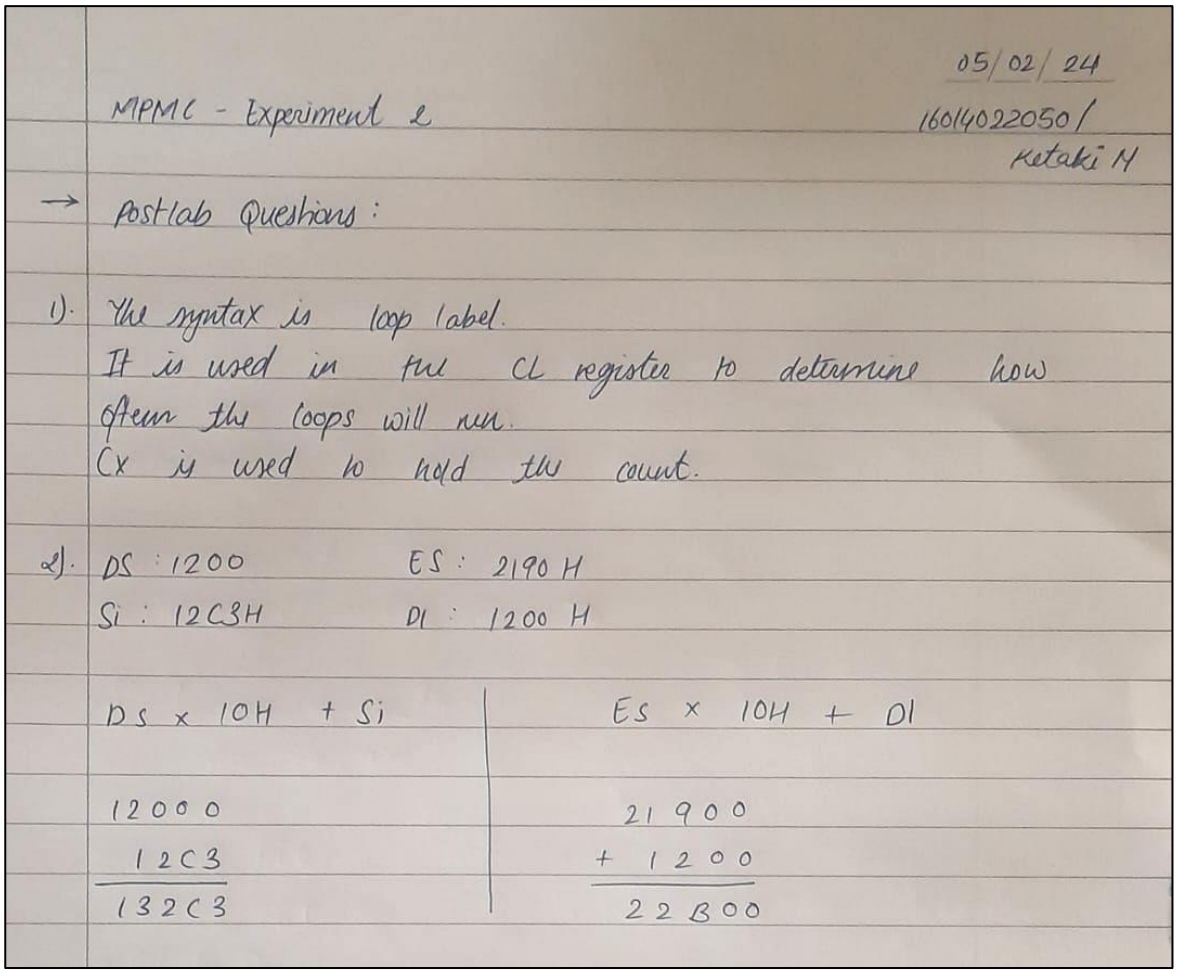
original source code

```
05
06  code segment
07  assume cs: code, ds: data
08  start:
09  mov ax, data
10  mov ds, ax
11
12  lea si, fibo
13
14  mov cl, cnt
15  mov ch, 0
16  mov al, 0
17  mov [si], al
18
19  inc si
20
21  mov al, 01
22  mov [si], al
23
24
25  up:
26  mov al, [si]
27  add al, [si - 1]
28  daa
29  inc si
30  mov [si], al
31  loop up
32
33  mov ah, 4ch
34  int 21h
35  code ends
36  end start
```



Post Lab Subjective/Objective type Questions:

1. **What is the LOOP instruction? Explain use of CX register in the same.**
2. **What addresses will be generated in following instruction execution?**
   If DS = 3200H, SI = 12C3H, DI = 1200H, ES = 2190H
   MOVSB



05/02/24

MPMC - Experiment 2

16014022050 / Ketaki M

→ Postlab Questions :

1). The syntax is loop label.
It is used in the CL register to determine how often the loops will run.
Cx is used to hold the count.

2). DS : 1200 ES : 2190 H
SI : 12C3H DI : 1200 H

DS x 10H + SI	ES x 10H + DI
12000	21900
12C3	+ 1200
132C3	22B00

3. **Which of the following combination of segment register and offset is not calculating address 23410H**
   a. **DS: 2000 H and SI: 3410 H**
   b. **DS: 2300 H and SI: 0410 H**
   c. **DS: 2341 H and SI: 0010 H**
   d. **DS: 2241 H and SI: 1000 H**

Conclusion:

In conclusion, the 8086 Assembly Language Program effectively generates and stores the initial 10 Fibonacci numbers within the data segment. This experiment showcases the successful implementation of the Fibonacci sequence algorithm on the x86 architecture, highlighting the program's ability to handle numerical computations and memory management.

Signature of faculty in-charge with Date: