Experiment No. 04

4.1 Experiment Name

Experimental Study of Stack Operation and Introduction to Procedures

4.2 Objectives

- To understand the structure of Assembly Language programming
- To learn about the microprocessor emulator "Emu 8086" and its operation
- To get acquainted with the stack operation and its implementation

4.3 Theory

The Assembly language is a low-level computer programming language that consists primarily of symbolic versions of a computer's machine language.

A stack is essentially a storage device that stores data in the LIFO (last in, first out) order. It is a memory unit with a Stack Pointer address register (SP). Stacking is broken down into two operations:

- **POP:** Deletion (decrements SP); removes the top word to the stack
- **PUSH:** Insertion (increments SP); adds a new top word to the stack

Addresses and data are saved in the stack when the CPU branches to a procedure. In RAM, a LIFO (last in, first out) data structure is implemented. This stack was then pushed to the top with the return address. In addition, the stack is used to swap the values of two registers and register pairs.

To write a program, a stack segment is required after ".MODEL" segment.

A ".STACK" segment is used to set a block of memory to store the stack.

After that a data segment ".DATA" is declared followed by a ".CODE" segment which by the way contains instructions of the program.

Then "MAIN PROC" segment acts as an entry point to the program. In some other cases, they may call other procedures, or themselves. Theses are mainly of two kinds: NEAR and FAR. NEAR segment is the same code segment as the calling program. But FAR is in different segment.

This segment is ended with "MAIN ENDP" where other procedures go. Both PROC and ENDP are pseudo-ops that delineate the procedure.

"END MAIN" segment ends the program and must be followed by the name of the main procedure.

4.4 Apparatus

- Emu 8086 - Microprocessor Emulator

4.5 Emulator Code

```
.MODEL SMALL
.STACK 100
.DATA
.VAR1 DB 12H
.CODE
MAIN PROC
MOV SP, 0101H
MOV AX, 1234H
PUSH AX
MOV BX, 4321H
PUSH AX
POP AX
POP BX
MAIN ENDP
END MAIN
```

4.6 Output

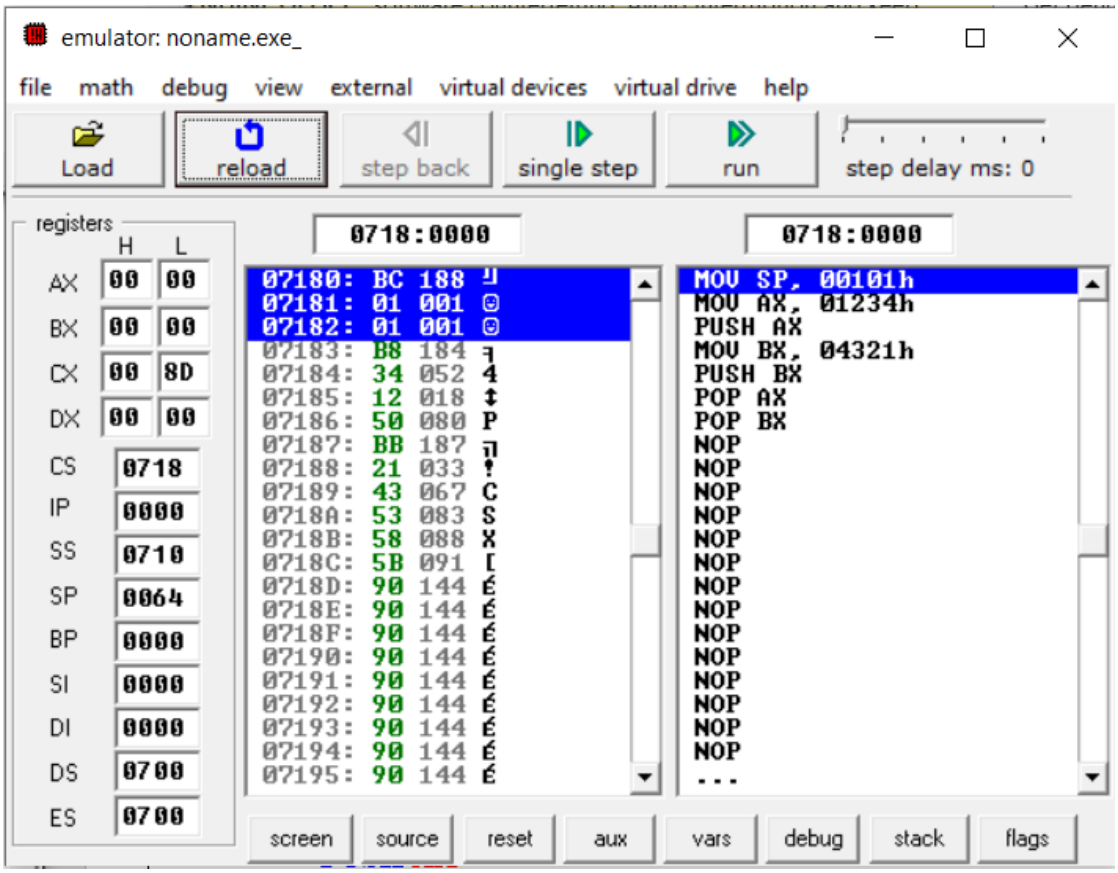

Fig 4.1: Emulator output for ALU for Stack operation (After Step 0)

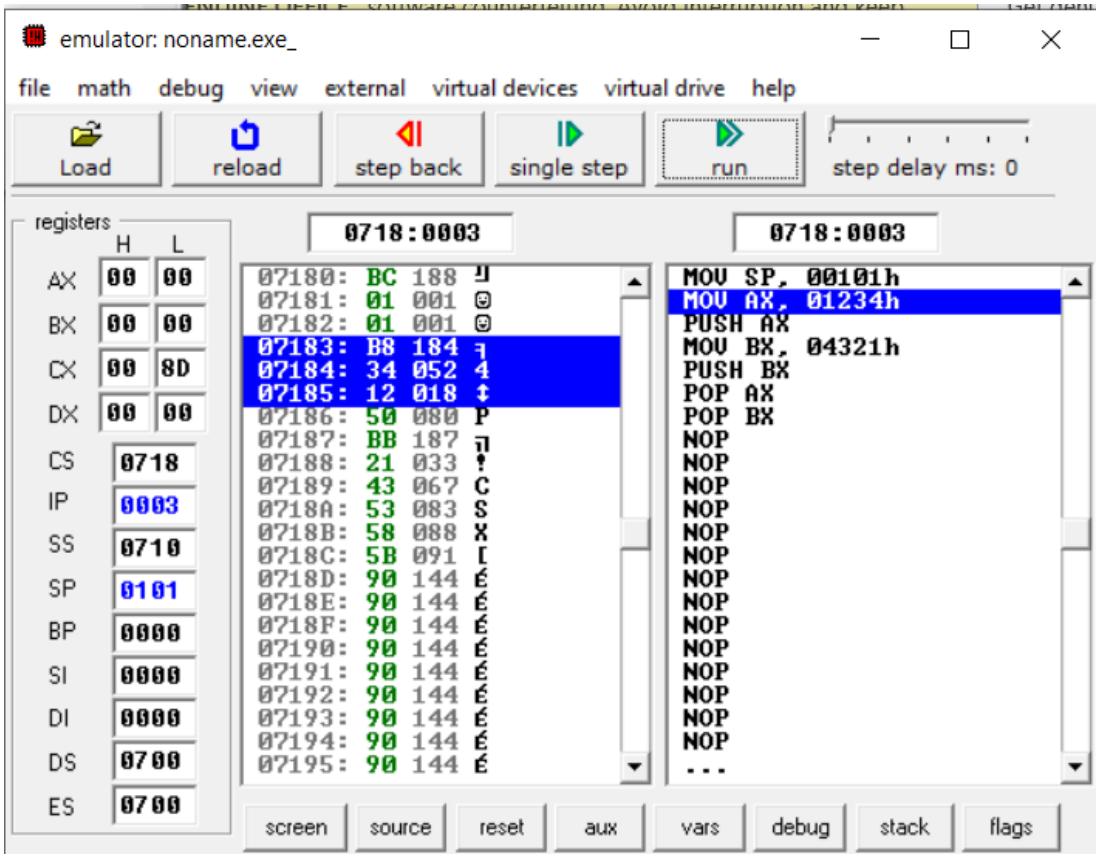

Fig 4.2: Emulator output for ALU for Stack operation (After Step 1)

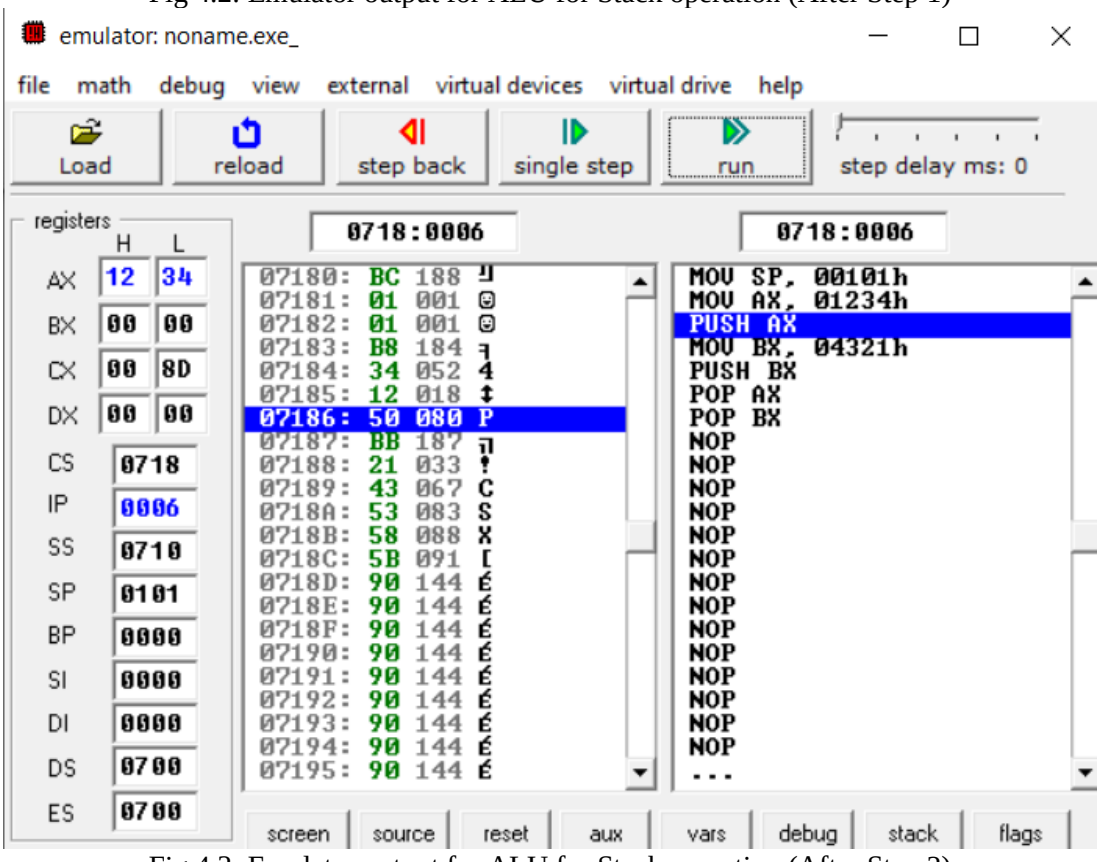

Fig 4.3: Emulator output for ALU for Stack operation (After Step 2)

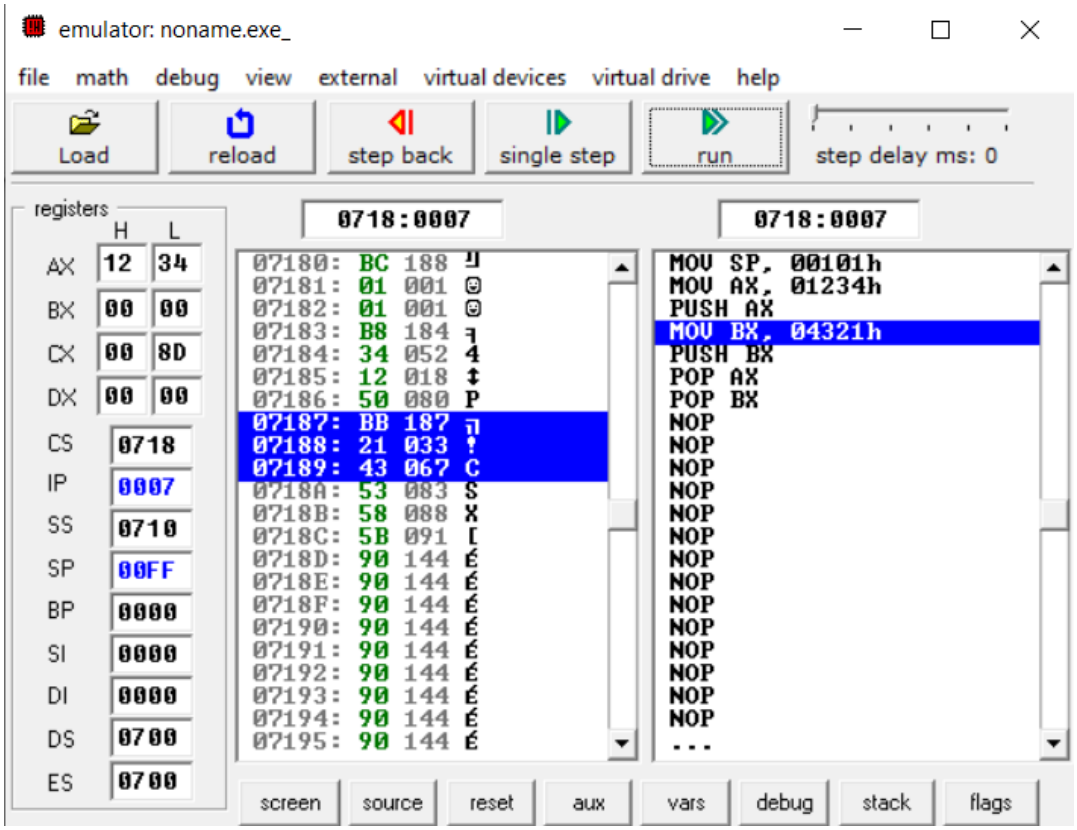

Fig 4.4: Emulator output for ALU for Stack operation (After Step 3)

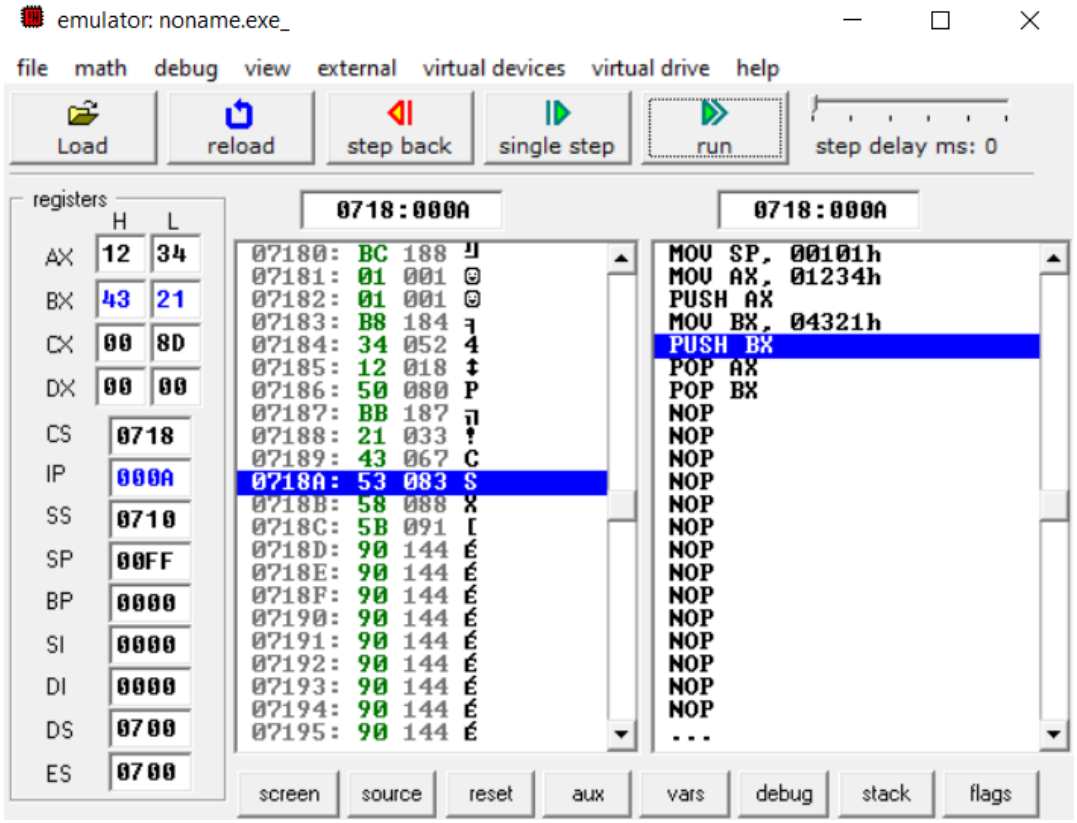

Fig 4.5: Emulator output for ALU for Stack operation (After Step 4)

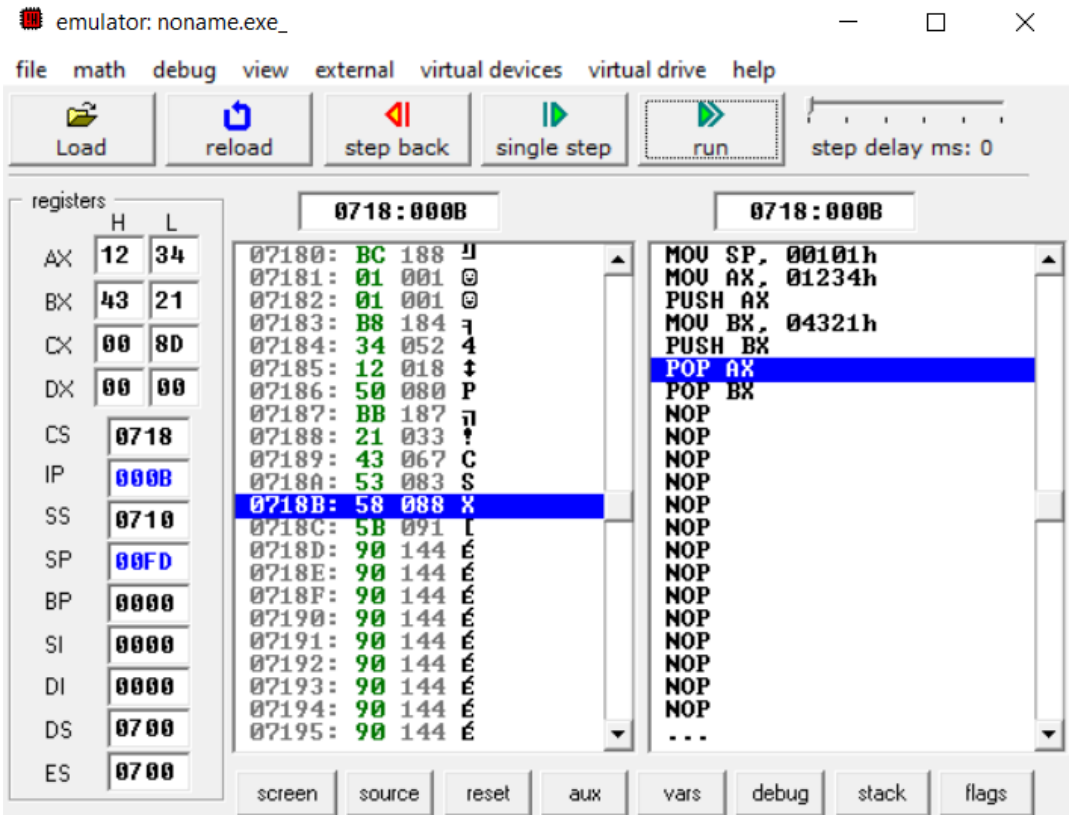

Fig 4.6: Emulator output for ALU for Stack operation (After Step 5)

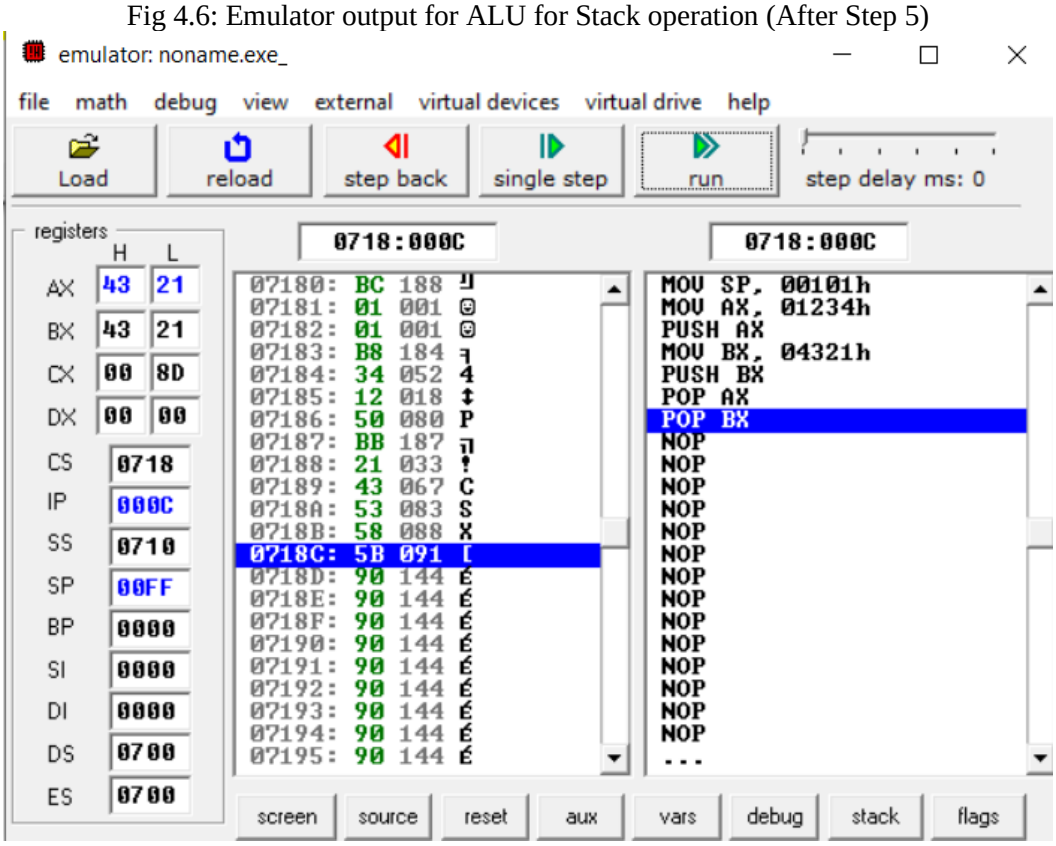

Fig 4.7: Emulator output for ALU for Stack operation (After Step 6)

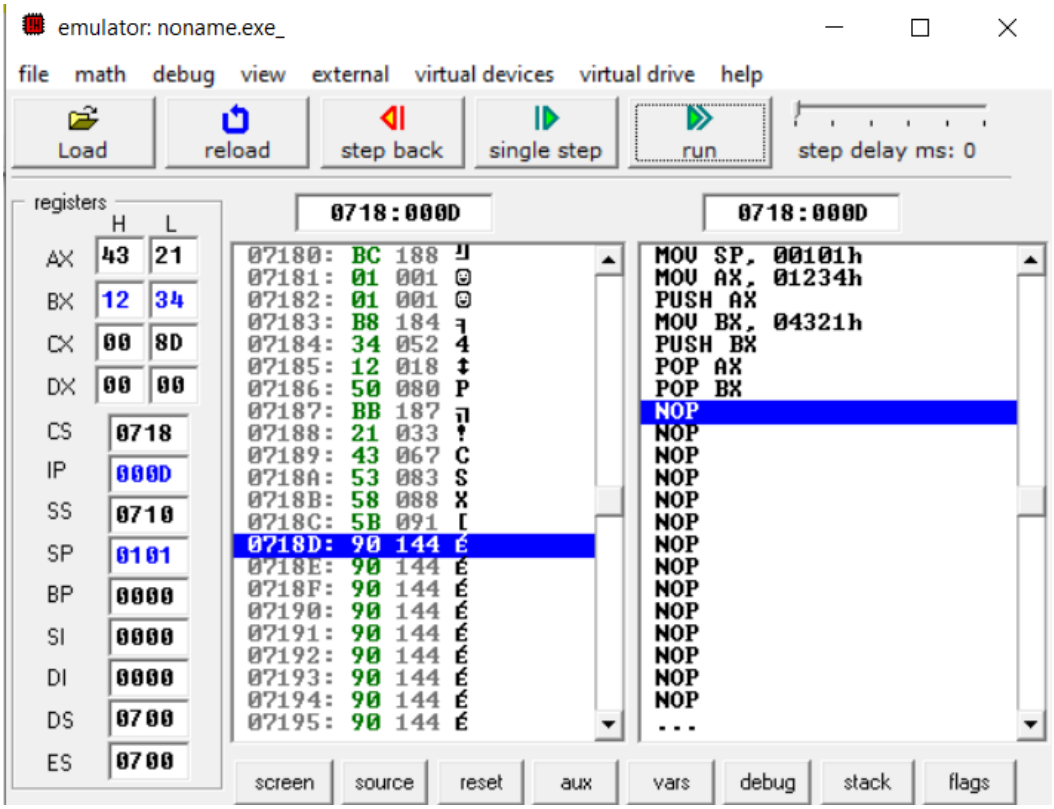

Fig 4.8: Emulator output for ALU for Stack operation (After Step 7)

4.7 Discussion & Conclusion

In this experiment, we learned about the techniques for building an assembly language program. We understand the procedure code and how the stack operation works.

Inserting an element into the stack is known as a "push" & "pop" operation and the new element is positioned at the top of the stack since there is only one other location where it can be added: the very top of the stack.

We also become acquainted with the term 'pseudo-ops' and their operations. Finally, we utilized all of these practically. Thus, the objective of the experiment was achieved.