Experiment No. 05

5.1 Experiment Name

Experimental study of String operation using assembly language

5.2 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 string operation and its implementation

5.3 Theory

A memory string or string is essentially a byte or word array in the 8086-assembly language. String instructions are thus intended for array processing. The string instructions' tasks can be completed out using the register indirect addressing mode. Each string instruction may require a source, a destination, or both operands.

In string operations, the direction of action is chosen by the direction flag (DF). The direction flag determines the action direction in string operations (DF). These operations are carried out by SI and DI in this location.

If DF = 0, the instruction "CLD" is executed, and SI and DI proceed in the direction of increasing memory addresses (from left to right across the string), If DF = 1, the instruction STD is issued, and SI and DI proceed in the direction of decreasing memory addresses (from right to left),

MOVSB, MOVSW, REP, STOSB, LODSB, STASB, and CMPSB are some more string commands.

- **MOVSB**: This MOV instruction is used to copy the contents of one string into another string. The MOVSB instruction copies the contents of the byte addressed by DS:SI, to the byte addressed by ES:DI.
- **MOVSW**: MOVSW is a function that moves a word from the source string to the destination string. It expects DS:DI to point to a source string word and ES:DS to point to a destination string, just like MOVSB. SI and DI are both increased by 2 if DF is 0 and decreased by 2 if DF is 1.
- **REP**: MOVSB transfers only one byte from the source string to the destination string. Before executing the REP prefix, we must first initialize CX to the number n of bytes in the source string, which causes MOVSB to be executed n times in order to relocate the entire string. After each MOVSB, CX is reduced until it reaches zero.
- **CLD**: For a clear direction flag, set DF to 0, which causes the operation to run from left to right.
- **STD**: For set direction flagging, DF = 1, causing the procedure to proceed from right to left.
- **STOSB**: Stores string bytes and transfers the contents of the AL register to the byte specified by ES:DI. DI is calculated as increment DF = 0 DF=0 or decrement DF = 1 DF=1.

- **STOSW**: Saves a string word and copies the contents of the AL register to the byte specified by ES:DI. DI is calculated as increment DF = 0 DF=0 or decrement DF = 1 DF=1.
- **LODSB**: This instruction is used to transfer the byte addressed by DS:SI into AL. Then, depending on whether DF is zero or one, SI is increased or decreased. The word form converts the word addressed by DS, SI, into AX, with SI increasing by 2 if DF is zero and decreasing by 2 if DF is one. LODSB may analyze the characters in a string. loading a byte string
- **LODSW**: Load string word; the other operation is well known to LODSB.
- **CMPSW**: String words are compared.
- **CMPSB**: string byte comparison

5.4 Apparatus

- Emu 8086 - Microprocessor Emulator

5.5 Emulator Code & Output

5.5.1 Duplicate a string and print it

```
.MODEL SMALL
.STACK 100
.DATA
STRING1 DB 'ASHRAF'
STRING2 DB 6 DUP (?)
.CODE
    MAIN PROC NEAR
        MOV AX, @DATA
        MOV DS, AX
        MOV ES, AX
        LEA SI, STRING1
        LEA DI, STRING2
        CLD
        MOV CL,6
        REP MOVSB
        MAIN ENDP
    END MAIN
```

Output

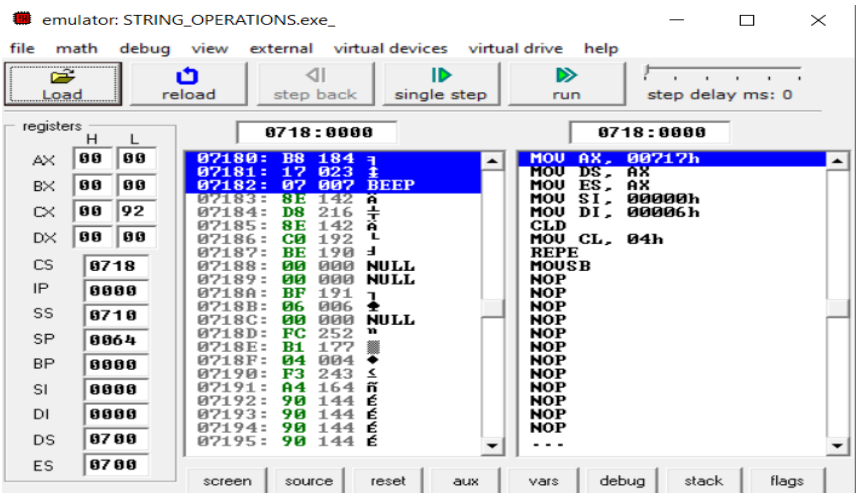

Fig 5.1.1: Emulator output for ALU for string operation (After Step 0)

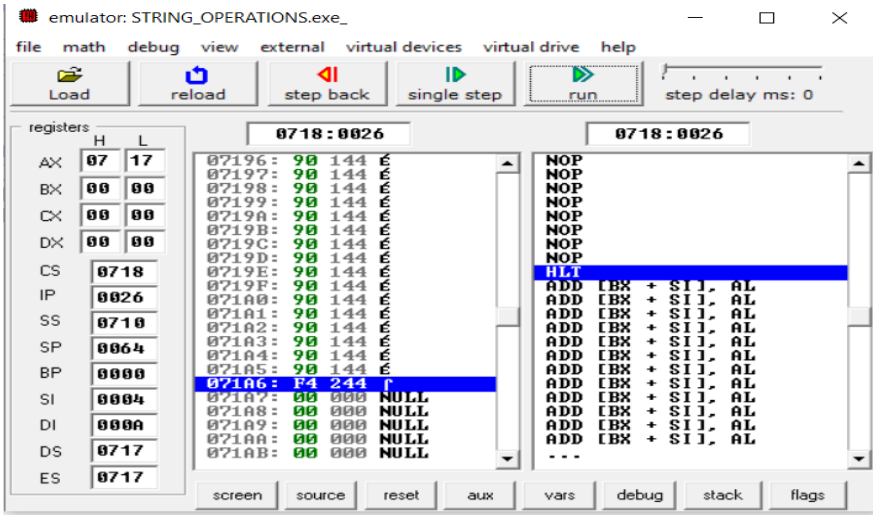

Fig 5.1.6: Emulator output for ALU for string operation (After Step 5)

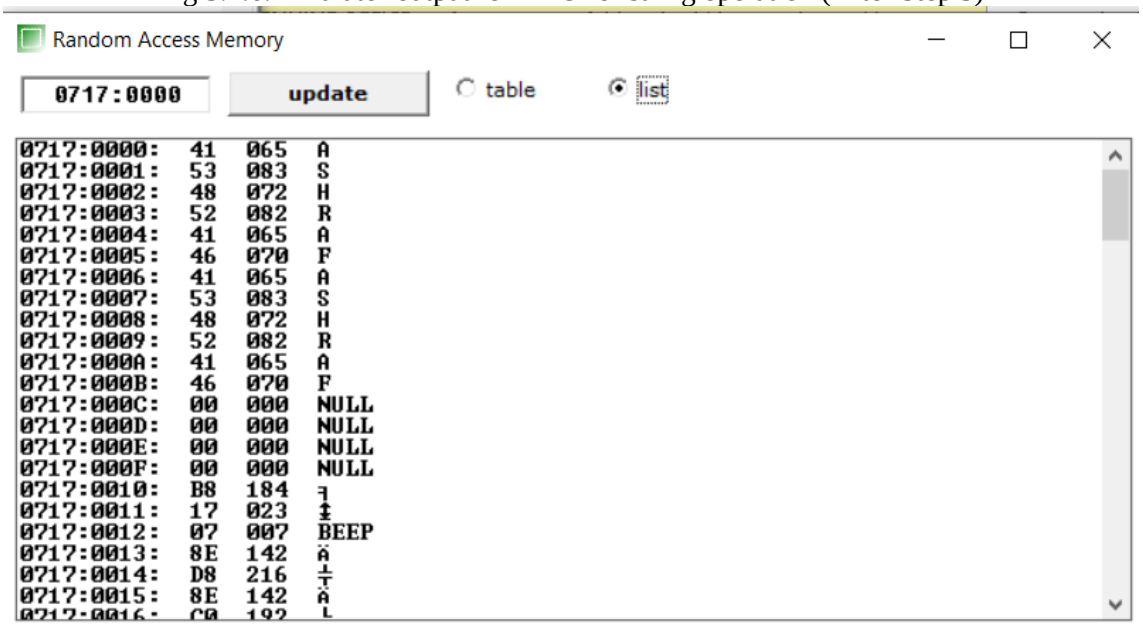

Fig 5.1.3: Memory output for ALU for string operation (final)

5.5.2 The same as Task Just this time, the copied phrase will begin with 0000H and the term that will be duplicated will begin after the copied term.

```
.MODEL SMALL
.STACK 100
.DATA
STRING2 DB 8 DUP(?)
STRING1 DB 'EEE_RUET'
.CODE
    MAIN PROC NEAR
        MOV AX,@DATA
        MOV DS,AX
        MOV ES,AX
        LEA SI,STRING1
        LEA DI,STRING2
        CLD
        MOV CL,8
```

```
        REP
        MOVSB
        MAIN ENDP
    END MAIN
```

Output

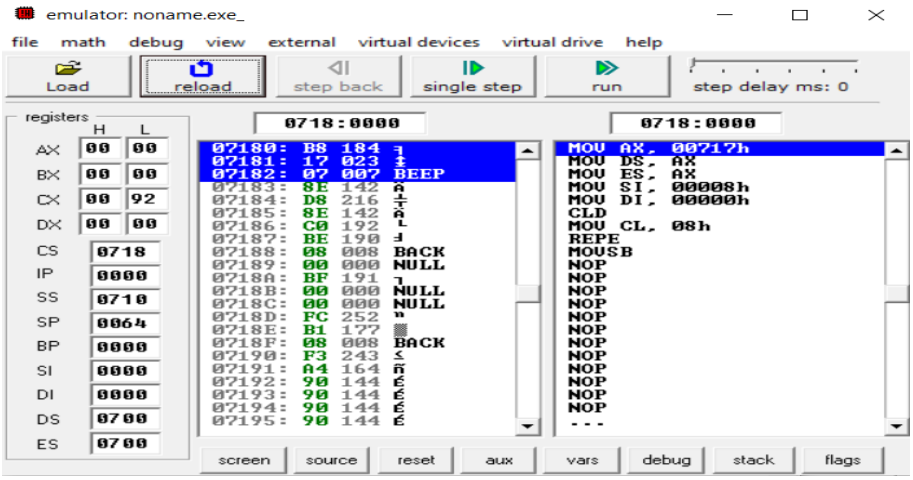

Fig 5.2.1: Emulator output for ALU for string operation (After Step 0)

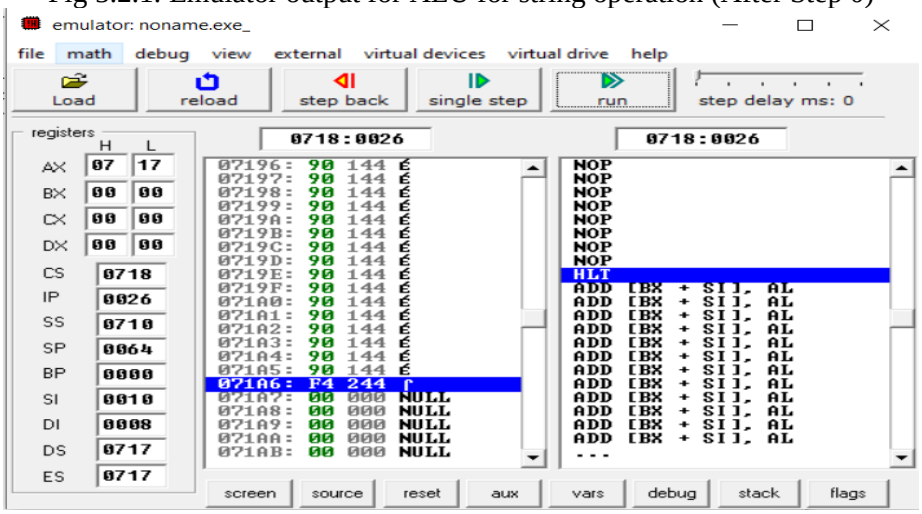

Fig 5.2.2: Emulator output for ALU for string operation (After final step)

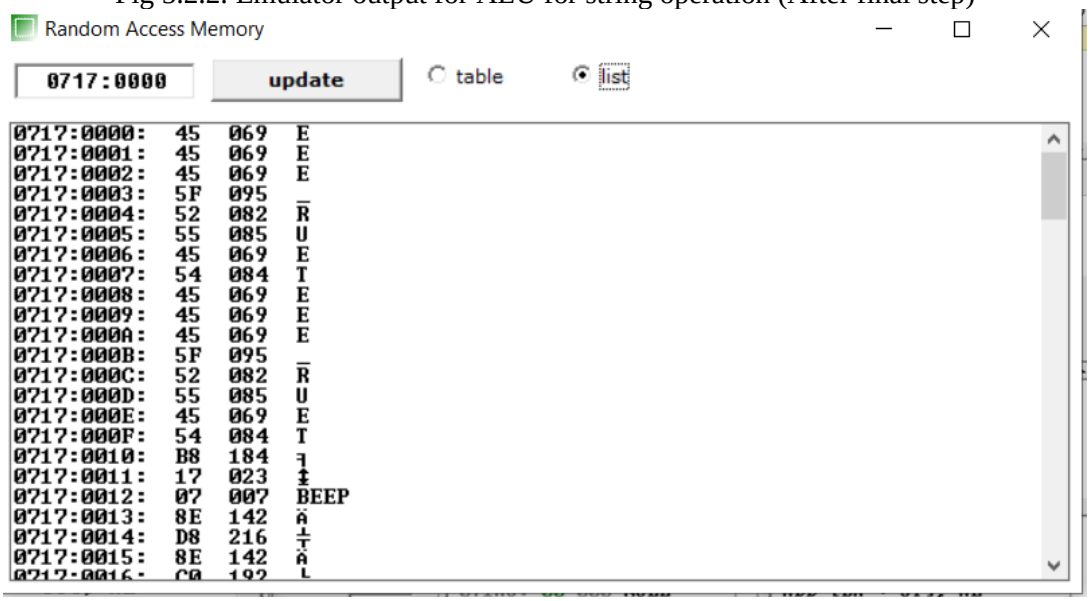

Fig 5.2.3: Memory output for ALU for string operation (final)

5.5.3 String correction of 'HLLO' to 'HELLO'

```
.MODEL SMALL
.STACK 100
.DATA
STRING1 DB 'HLLO'
.CODE
    MAIN PROC NEAR
        MOV AX,@DATA
        MOV DS,AX
        MOV ES,AX
        LEA SI,STRING1+3
        LEA DI,STRING2+4
        STD
        MOV CL,3
       REP MOVSB
       MOV AL,'E'
        STOSB
        MAIN ENDP
     END MAIN
```

Output

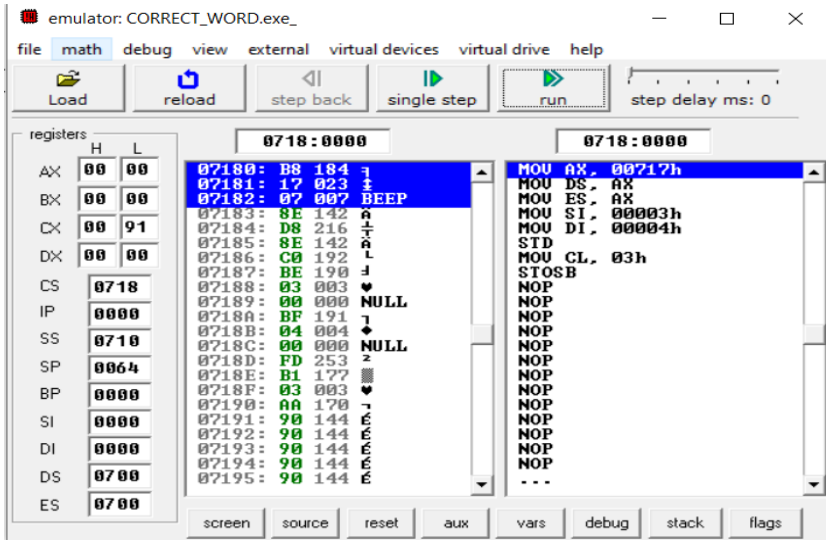

Fig 5.3.1: Emulator output for ALU for string operation (After Step 0)

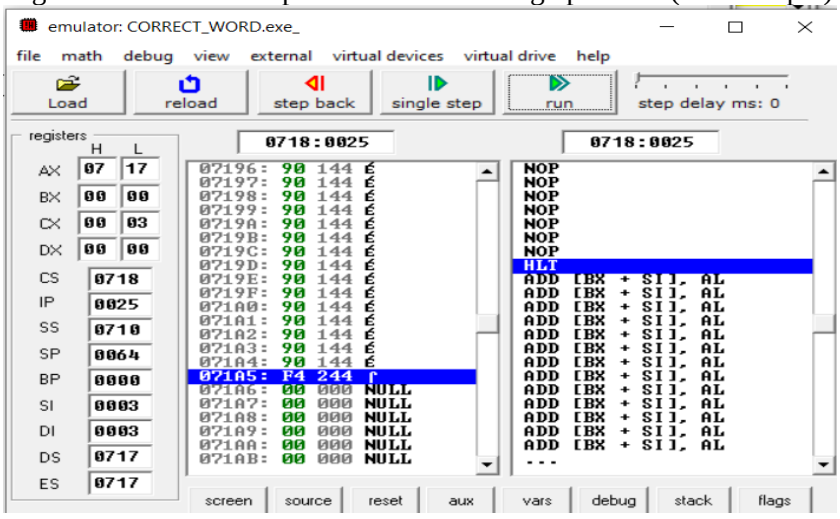

Fig 5.3.2: Emulator output for ALU for string operation (After final step)

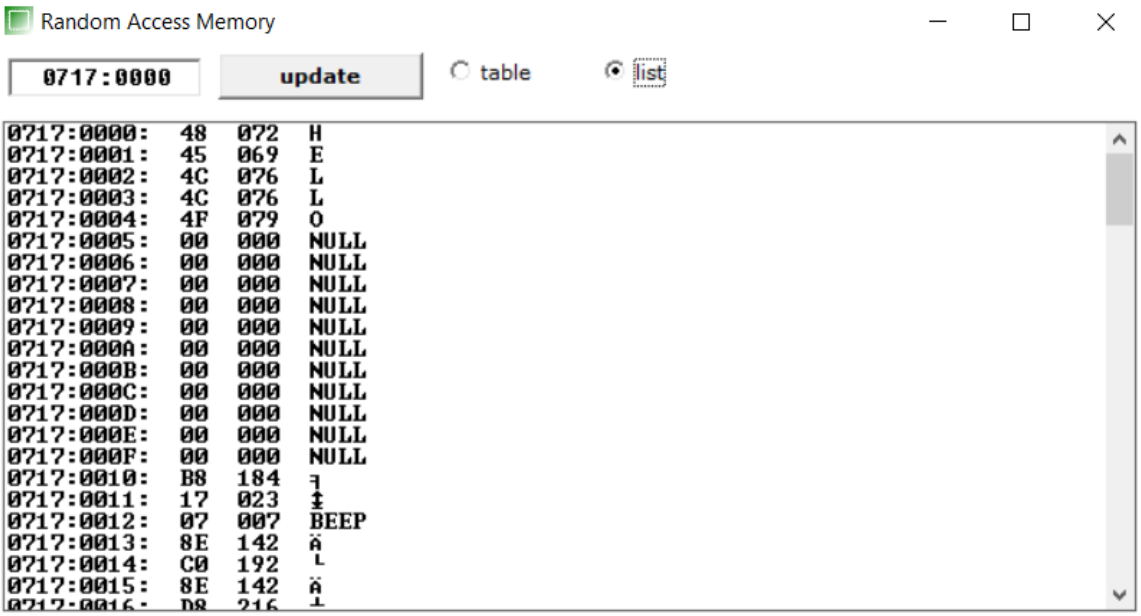

Fig 5.3.3: Memory output for ALU for string operation (final)

5.6 Discussion & Conclusion

In this experiment, we learned about the techniques for building an assembly language program. We understand the procedure code and how the string operation and some of its instructions work. An experimental test of the string's functionality was performed using the well-known application emu8086.

Finally, we utilized all of these practically. The results of the experiment and our manual computation were comparable. Thus, the objective of the experiment was achieved.