Name:- Sravan Chittupalli

VJTI ID: 181060018

EXPERIMENT - 2

AIM

1. Design an 8086-assembly language program to sort an array of 8-bit integers in ascending order. The number of integers available in array is available as a 8 bit variable **Number.** The 8-bit integers are placed at address **source**. The sorted array is stored at address **destination.**

2. Design an 8086-assembly language program to sort an array of 8-bit integers in descending order. The number of integers available in array is available as a 8 bit variable **Number.** The 8-bit integers are placed at address **source**. The sorted array is stored at address **destination.**

SOFTWARE

- EMU8086 emulator

EXPERIMENTS

1. **ASCENDING SORT:** -

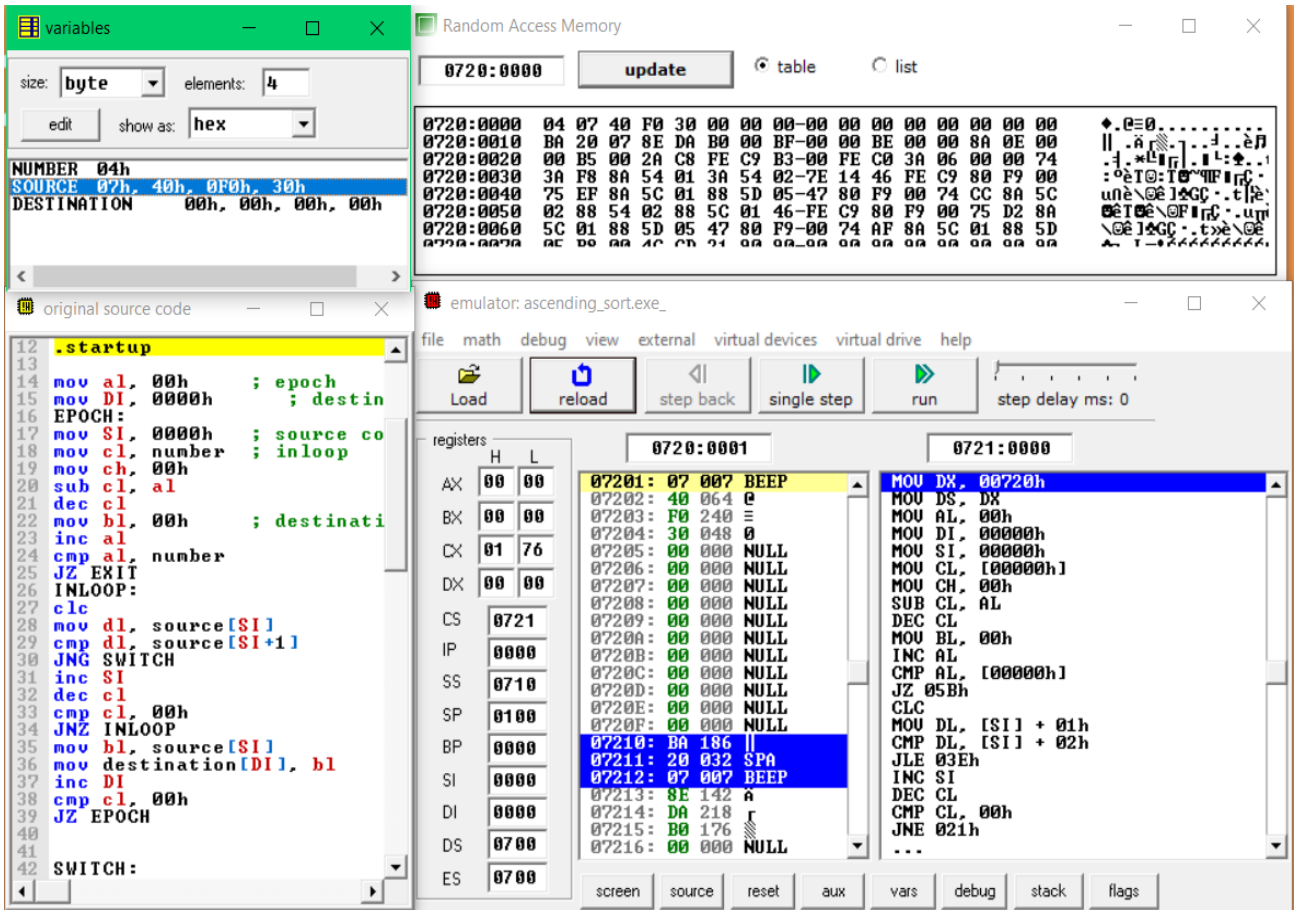
- **OVERVIEW**
  Here we are sorting the number in bubble sorting technique. In this sorting technique there will be n passes for n different numbers. In i^{th} pass the i^{th} smallest element will be placed at the end. This is comparison-based sort. We taking two consecutive numbers, compare them, and then swap them if the numbers are not in correct order.

- **PROCEDURE**
  1. Setup an outer loop that shows the number of times to perform ascending operation to sort the memory. I have assigned `ax` register to do this.
  2. Next setup an inner loop that runs (epoch - `ax`) times.
  3. Compare the location pointed by SI with the next number. If the second number is greater than the first then perform the SWITCH operation else increment SI.
  4. Store the last element in the destination location.
  5. Repeat this process ‘ax’ times.
  6. Stop and terminate the program

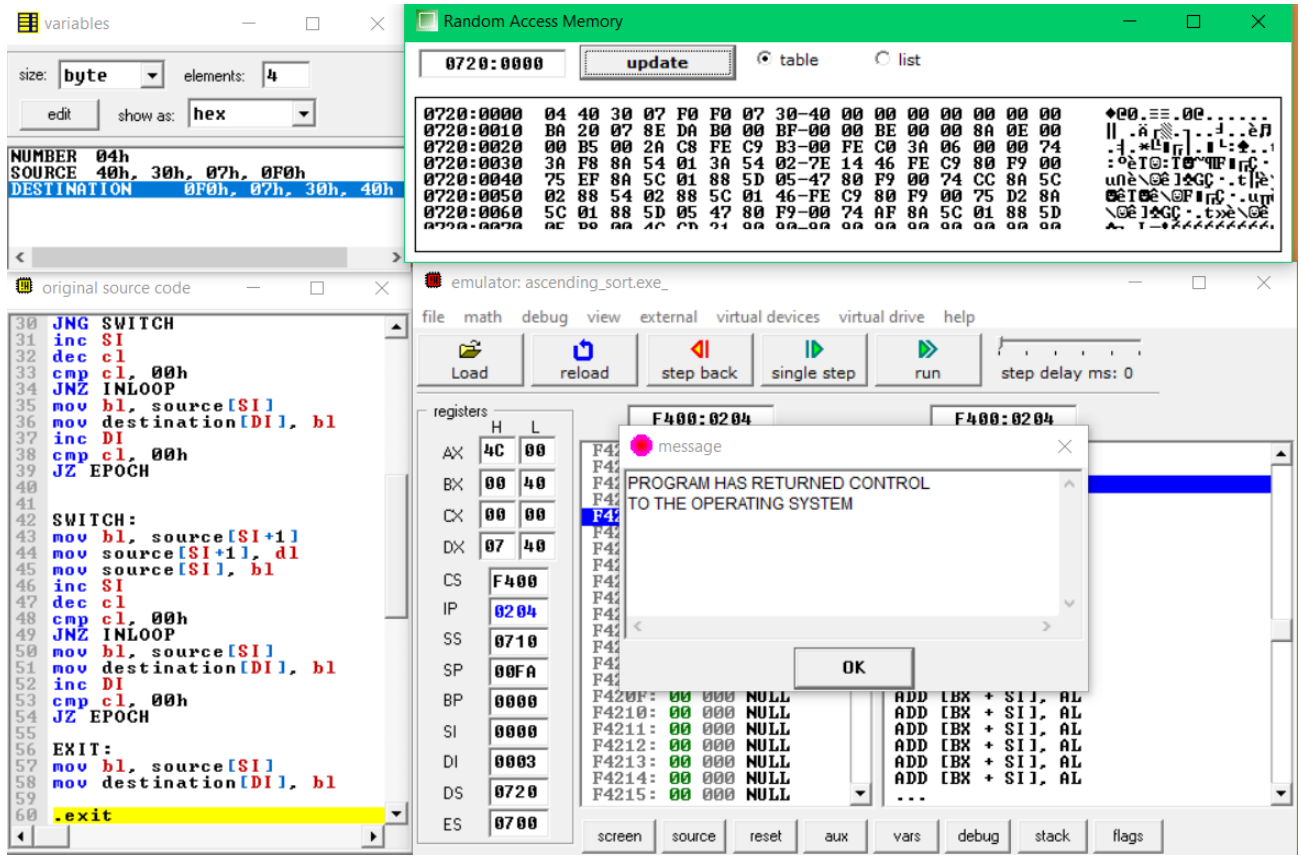
- **CODE: -**

```
.model small
.stack

.data
    number db 04h
    source db ?
    destination db ?

.code
    ascending proc  ; smallest to largest

        .startup

        mov al, 00h     ; epoch
        mov cl, number  ; inloop
        mov ch, 00h
        mov DI, cx      ; destination count
        dec DI
        EPOCH:
        mov SI, 0000h   ; source count
        mov cl, number  ; inloop
        mov ch, 00h
        sub cl, al
        dec cl
        mov bl, 00h     ; destination count
        inc al
        cmp al, number
        JZ EXIT
        INLOOP:
        clc
        mov dl, source[SI]
        cmp dl, source[SI+1]
        JNG SWITCH
        inc SI
        dec cl
        cmp cl, 00h
        JNZ INLOOP
        mov bl, source[SI]
        mov destination[DI], bl
        inc DI
        cmp cl, 00h
        JZ EPOCH


        SWITCH:
        mov bl, source[SI+1]
        mov source[SI+1], dl
        mov source[SI], bl
        inc SI
        dec cl
        cmp cl, 00h
        JNZ INLOOP
        mov bl, source[SI]
        mov destination[DI], bl
        inc DI
        cmp cl, 00h
        JZ EPOCH

        EXIT:
        mov bl, source[SI]
        mov destination[DI], bl

        .exit

    ascending endp

end ascending
```

- **INPUT**



- **OUTPUT**



2. **DESCENDING SORT:** -

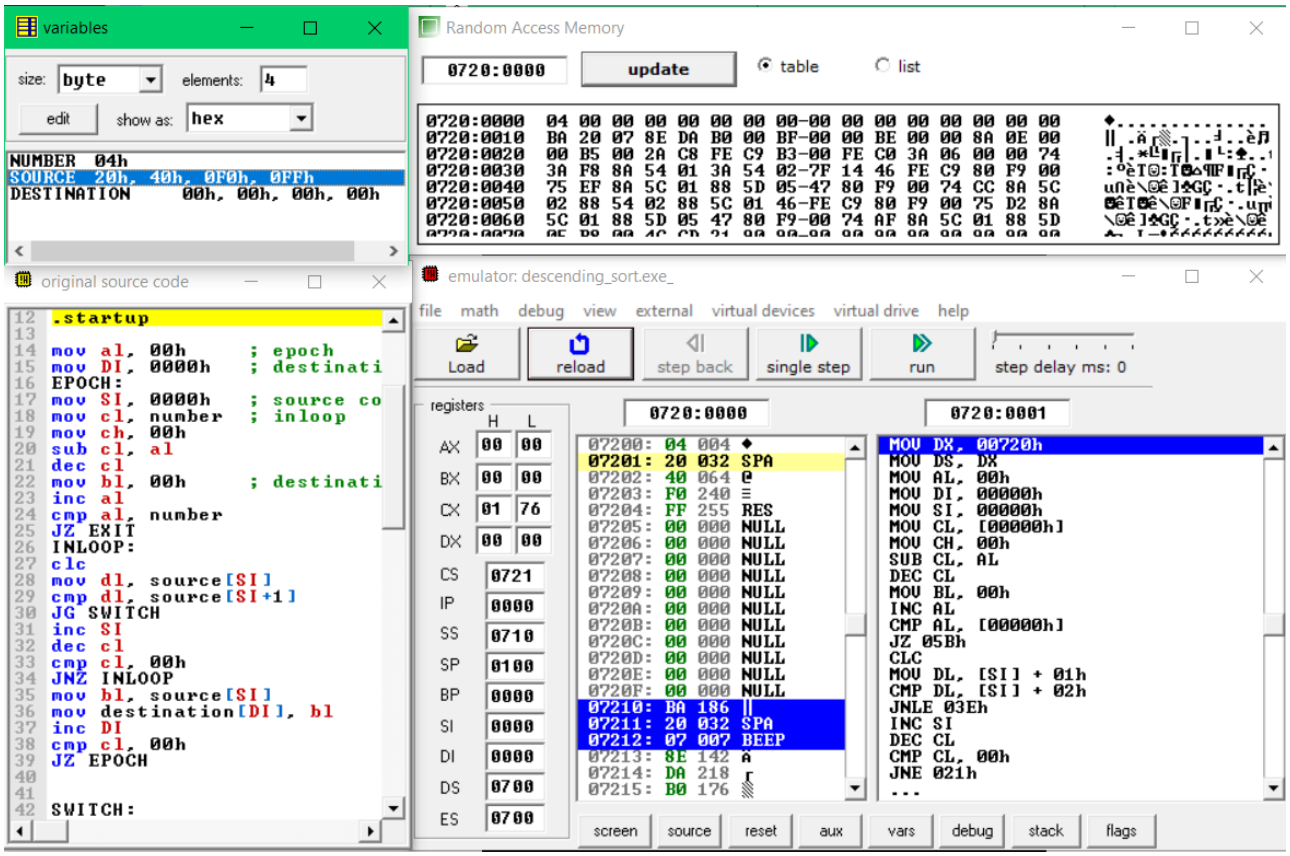
- **OVERVIEW**
  Here we are sorting the number in bubble sorting technique. In this sorting technique there will be n passes for n different numbers. In i^{th} pass the i^{th} smallest element will be placed at the end. This is comparison-based sort. We taking two consecutive numbers, compare them, and then swap them if the numbers are not in correct order.

- **PROCEDURE**
  1. Initialize the data segment memory.
  2. Setup an outer loop that shows the number of times to perform ascending operation to sort the memory. I have assigned `ax` register to do this.
  3. Next setup an inner loop that runs (epoch - `ax`) times.
  4. Compare the location pointed by SI with the next number. If the second number is greater than the first then perform the SWITCH operation else increment SI.
  5. Store the last element in the destination location.
  6. Repeat this process ‘ax’ times.
  7. Stop and terminate the program

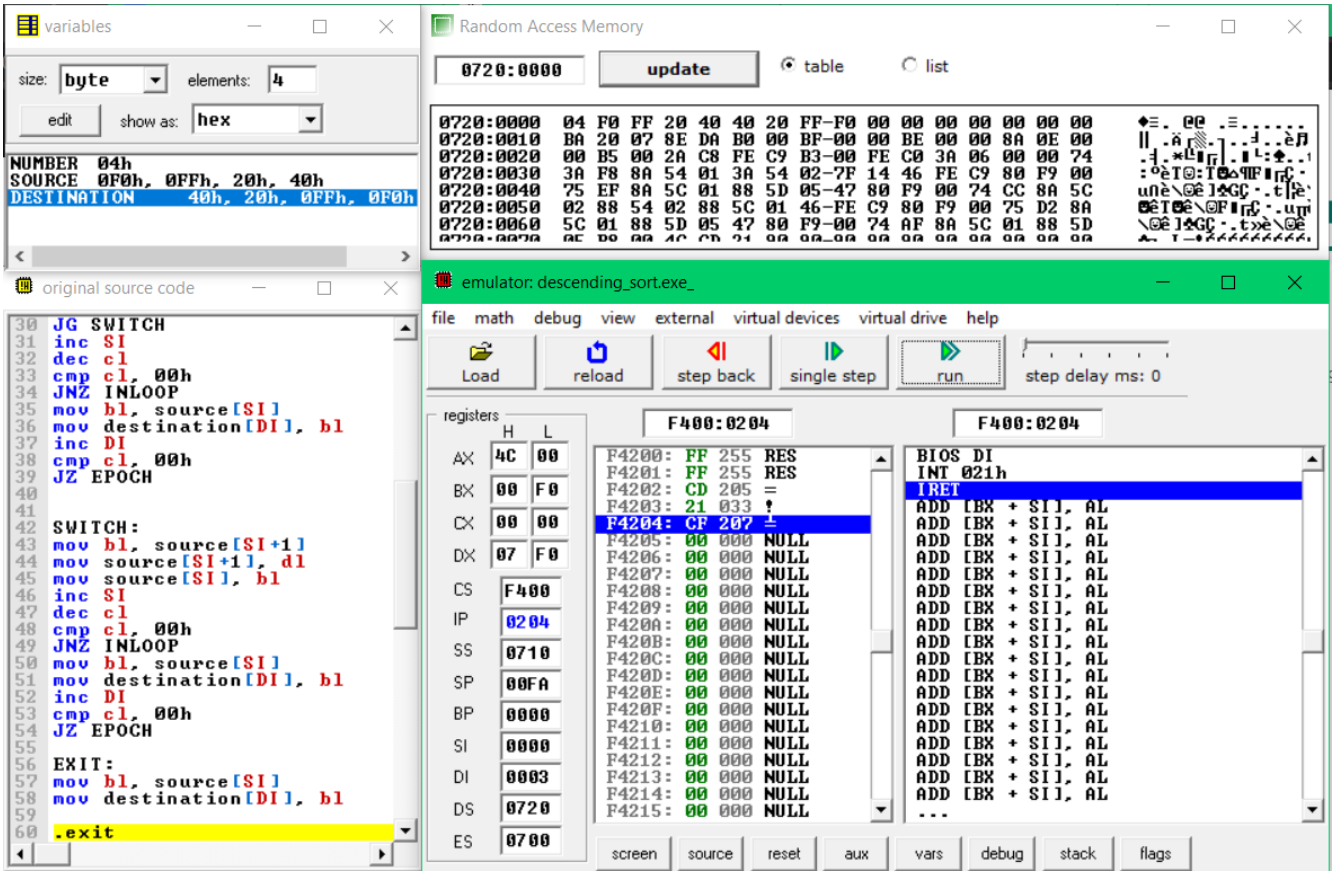
- **CODE**

```
.model small
.stack

.data
    number db 04h
    source db 04 dup (?)
    destination db 04 dup (?)

.code
    descending proc  ; largest to smallest

        .startup

        mov al, 00h     ; epoch
        mov DI, 0000h   ; destination count
        EPOCH:
        mov SI, 0000h   ; source count
        mov cl, number  ; inloop
        mov ch, 00h
        sub cl, al
        dec cl
        mov bl, 00h     ; destination count
        inc al
        cmp al, number
        JZ EXIT
        INLOOP:
        clc
        mov dl, source[SI]
        cmp dl, source[SI+1]
        JG SWITCH
        inc SI
        dec cl
        cmp cl, 00h
        JNZ INLOOP
        mov bl, source[SI]
        mov destination[DI], bl
        inc DI
        cmp cl, 00h
        JZ EPOCH


        SWITCH:
        mov bl, source[SI+1]
        mov source[SI+1], dl
        mov source[SI], bl
        inc SI
        dec cl
        cmp cl, 00h
        JNZ INLOOP
        mov bl, source[SI]
        mov destination[DI], bl
        inc DI
        cmp cl, 00h
        JZ EPOCH

        EXIT:
        mov bl, source[SI]
        mov destination[DI], bl

        .exit

    descending endp

end descending
```

- **INPUT**



- **OUTPUT**



CONCLUSION

In this program we have used the data transfer and loop control instructions. Using these instructions, we have learnt how to initialise and work with arrays.