

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:
            cld
            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

variables

size: **byte** elements: **4**

edit show as: **hex**

NUMBER 04h
SOURCE 07h, 40h, 0F0h, 30h
DESTINATION 00h, 00h, 00h, 00h

original source code

```

12 .startup
13
14 mov al, 00h ; epoch
15 mov di, 0000h ; destin
16 EPOCH:
17 mov si, 0000h ; source co
18 mov cl, number ; inloop
19 mov ch, 00h
20 sub cl, al
21 dec cl
22 mov bl, 00h ; destinati
23 inc al
24 cmp al, number
25 JZ EXIT
26 INLOOP:
27 clc
28 mov dl, source[SI]
29 cmp dl, source[SI+1]
30 JNG SWITCH
31 inc SI
32 dec cl
33 cmp cl, 00h
34 JNZ INLOOP
35 mov bl, source[SI]
36 mov destination[DI], bl
37 inc DI
38 cmp cl, 00h
39 JZ EPOCH
40
41 SWITCH:
42

```

Random Access Memory

0720:0000 update table list

0720:0000	04 07 40 F0 30 00 00 00 00 00 00 00 00 00 00 00	04 07 40 F0 30 00 00 00 00 00 00 00 00 00 00
0720:0010	BA 20 07 8E DA B0 00 BF 00 00 BE 00 00 8A 0E 00	BA 20 07 8E DA B0 00 BF 00 00 BE 00 00 8A 0E 00
0720:0020	00 B5 00 2A C8 FE C9 B3 00 FE C0 3A 06 00 00 74	00 B5 00 2A C8 FE C9 B3 00 FE C0 3A 06 00 00 74
0720:0030	3A F8 8A 54 01 3A 54 02 7E 14 46 FE C9 80 F9 00	3A F8 8A 54 01 3A 54 02 7E 14 46 FE C9 80 F9 00
0720:0040	75 EF 8A 5C 01 88 5D 05 47 80 F9 00 74 CC 8A 5C	75 EF 8A 5C 01 88 5D 05 47 80 F9 00 74 CC 8A 5C
0720:0050	02 88 54 02 88 5C 01 46 FE C9 80 F9 00 75 D2 8A	02 88 54 02 88 5C 01 46 FE C9 80 F9 00 75 D2 8A
0720:0060	5C 01 88 5D 05 47 80 F9 00 74 AF 8A 5C 01 88 5D	5C 01 88 5D 05 47 80 F9 00 74 AF 8A 5C 01 88 5D
0720:0070	02 88 54 02 88 5C 01 46 FE C9 80 F9 00 75 D2 8A	02 88 54 02 88 5C 01 46 FE C9 80 F9 00 75 D2 8A

emulator: ascending_sort.exe_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	00	00
BX	00	00
CX	01	76
DX	00	00
CS	0721	
IP	0000	
SS	0710	
SP	0100	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

0720:0001

0721:0000

screen source reset aux vars debug stack flags

• OUTPUT

variables

size: **byte** elements: **4**

edit show as: **hex**

NUMBER 04h
SOURCE 40h, 30h, 07h, 0F0h
DESTINATION 0F0h, 07h, 30h, 40h

original source code

```

30 JNG SWITCH
31 inc SI
32 dec cl
33 cmp cl, 00h
34 JNZ INLOOP
35 mov bl, source[SI]
36 mov destination[DI], bl
37 inc DI
38 cmp cl, 00h
39 JZ EPOCH
40
41 SWITCH:
42 mov bl, source[SI+1]
43 mov source[SI+1], dl
44 mov source[SI], bl
45 inc SI
46 dec cl
47 cmp cl, 00h
48 JNZ INLOOP
49 mov bl, source[SI]
50 mov destination[DI], bl
51 inc DI
52 cmp cl, 00h
53 JZ EPOCH
54
55 EXIT:
56 mov bl, source[SI]
57 mov destination[DI], bl
58
59 .exit
60

```

Random Access Memory

0720:0000 update table list

0720:0000	04 40 30 07 F0 F0 07 30 40 00 00 00 00 00 00 00	04 40 30 07 F0 F0 07 30 40 00 00 00 00 00 00 00
0720:0010	BA 20 07 8E DA B0 00 BF 00 00 BE 00 00 8A 0E 00	BA 20 07 8E DA B0 00 BF 00 00 BE 00 00 8A 0E 00
0720:0020	00 B5 00 2A C8 FE C9 B3 00 FE C0 3A 06 00 00 74	00 B5 00 2A C8 FE C9 B3 00 FE C0 3A 06 00 00 74
0720:0030	3A F8 8A 54 01 3A 54 02 7E 14 46 FE C9 80 F9 00	3A F8 8A 54 01 3A 54 02 7E 14 46 FE C9 80 F9 00
0720:0040	75 EF 8A 5C 01 88 5D 05 47 80 F9 00 74 CC 8A 5C	75 EF 8A 5C 01 88 5D 05 47 80 F9 00 74 CC 8A 5C
0720:0050	02 88 54 02 88 5C 01 46 FE C9 80 F9 00 75 D2 8A	02 88 54 02 88 5C 01 46 FE C9 80 F9 00 75 D2 8A
0720:0060	5C 01 88 5D 05 47 80 F9 00 74 AF 8A 5C 01 88 5D	5C 01 88 5D 05 47 80 F9 00 74 AF 8A 5C 01 88 5D
0720:0070	02 88 54 02 88 5C 01 46 FE C9 80 F9 00 75 D2 8A	02 88 54 02 88 5C 01 46 FE C9 80 F9 00 75 D2 8A

emulator: ascending_sort.exe_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	4C	00
BX	00	40
CX	00	00
DX	07	40
CS	F400	
IP	0204	
SS	0710	
SP	00FA	
BP	0000	
SI	0000	
DI	0003	
DS	0720	
ES	0700	

F400:0204

F400:0204

screen source reset aux vars debug stack flags

message
PROGRAM HAS RETURNED CONTROL TO THE OPERATING SYSTEM

OK

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:
        cld
        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**

variables

size: byte
elements: 4

edit
show as: hex

NUMBER	04h
SOURCE	20h, 40h, 0F0h, 0FFh
DESTINATION	00h, 00h, 00h, 00h

original source code

```

12 .startup
13
14 mov al, 00h ; epoch
15 mov DI, 0000h ; destination
16 EPOCH:
17 mov SI, 0000h ; source counter
18 mov cl, number ; inloop
19 mov ch, 00h
20 sub cl, al
21 dec cl
22 mov bl, 00h ; destination
23 inc al
24 cmp al, number
25 JZ EXIT
26 INLOOP:
27 cld
28 mov dl, source[SI]
29 cmp dl, source[SI+1]
30 JG SWITCH
31 inc SI
32 dec cl
33 cmp cl, 00h
34 JNZ INLOOP
35 mov bl, source[SI]
36 mov destination[DI], bl
37 inc DI
38 cmp cl, 00h
39 JZ EPOCH
40
41 SWITCH:
42

```

Random Access Memory

0720:0000
update
table
list

0720:0000	04 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00	♦
0720:0010	BA 20 07 8E DA B0 BF-00 BE 00 00 8A 0E 00
0720:0020	00 B5 00 2A C8 FE C9 B3-00 FE C0 3A 06 00 00 74
0720:0030	3A F8 8A 54 01 3A 54 02-7F 14 46 FE C9 80 F9 00
0720:0040	75 EF 8A 5C 01 88 5D 05-47 80 F9 00 74 CC 8A 5C	une
0720:0050	02 88 54 02 88 5C 01 46-FE C9 80 F9 00 75 D2 8A	0ET
0720:0060	5C 01 88 5D 05 47 80 F9-00 74 AF 8A 5C 01 88 5D	GE
0720:0070	0E 00 00 00 00 00 00 00-00 00 00 00 00 00 00

emulator: descending_sort.exe_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	00	00
BX	00	00
CX	01	76
DX	00	00
CS	0721	
IP	0000	
SS	0710	
SP	0100	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

0720:0000
0720:0001

07200: 04 004	♦
07201: 20 032 SPA	
07202: 40 064	e
07203: F0 240	=
07204: FF 255 RES	
07205: 00 000 NULL	
07206: 00 000 NULL	
07207: 00 000 NULL	
07208: 00 000 NULL	
07209: 00 000 NULL	
0720A: 00 000 NULL	
0720B: 00 000 NULL	
0720C: 00 000 NULL	
0720D: 00 000 NULL	
0720E: 00 000 NULL	
0720F: 00 000 NULL	
07210: BA 186	
07211: 20 032 SPA	
07212: 07 007 BEEP	
07213: 8E 142	A
07214: DA 218	r
07215: B0 176	

MOV DX, 00720h
MOV DS, DX
MOV AL, 00h
MOV DI, 00000h
MOV SI, 00000h
MOV CL, [00000h]
MOV CH, 00h
SUB CL, AL
DEC CL
MOV BL, 00h
INC AL
CMP AL, [00000h]
JZ 05Bh
CLC
MOV DL, [SI] + 01h
CMP DL, [SI] + 02h
JNLE 03Eh
INC SI
DEC CL
CMP CL, 00h
JNE 021h
...

screen source reset aux vars debug stack flags

- **OUTPUT**

The screenshot displays the Immunity Debugger interface with the following components:

- Variables Window:** Shows a variable named `NUMBER` of type `04h` (byte) with a value of `0F0h`. The source is `0F0h, 0FFh, 20h, 40h` and the destination is `40h, 20h, 0FFh, 0F0h`.
- Random Access Memory Window:** Displays a memory dump starting at address `0720:0000`. The data is shown in hexadecimal and ASCII. The ASCII column contains the string `... ..`.
- Emulator Window:** Shows the execution of `emulator: descending_sort.exe`.
- Assembly View:** Displays the assembly code for the `descending_sort.exe` process. The code is as follows:


```

30 JG SWITCH
31 inc SI
32 dec cl
33 cmp cl, 00h
34 JNZ INLOOP
35 mov bl, source[SI]
36 mov destination[DI], bl
37 inc DI
38 cmp cl, 00h
39 JZ EPOCH
40
41
42 SWITCH:
43 mov bl, source[SI+1]
44 mov source[SI+1], dl
45 mov source[SI], bl
46 inc SI
47 dec cl
48 cmp cl, 00h
49 JNZ INLOOP
50 mov bl, source[SI]
51 mov destination[DI], bl
52 inc DI
53 cmp cl, 00h
54 JZ EPOCH
55
56 EXIT:
57 mov bl, source[SI]
58 mov destination[DI], bl
59
60 .exit
      
```
- Registers Window:** Shows the current state of the registers. The `AX` register contains `4C 00`, `BX` contains `00 F0`, `CX` contains `00 00`, `DX` contains `07 F0`, `CS` contains `F400`, `IP` contains `0204`, `SS` contains `0710`, `SP` contains `00FA`, `BP` contains `0000`, `SI` contains `0000`, `DI` contains `0003`, `DS` contains `0720`, and `ES` contains `0700`.
- Memory Dump:** Shows the memory dump for the `BIOS DI` segment. The data is shown in hexadecimal and ASCII. The ASCII column contains the string `... ..`.

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.