LAB - 3

Aim of this lab: Analyse and evaluate the array operations using 8086 microprocessors.

OBJECTIVE - ① : (A) find the largest Numbers (8 bit Numbers) from a given array of size N.

① ASSEMBLY CODE :

```
1). data
2) Count db 04h      ; Count = array size (N=4)
3) value db 13h, 0ch, 05h, 15h      ; array elements
4) res db ?      ; variable to store the result
5). Code
6) MAIN PROC
7)    mov AX, data    ;  AX ← address of data segment
8)    mov ds, AX      ;  DS ← AX
9)    mov CL, Count   ;  CL ← Count (load array size)
10)   DEC CL;         ;  CL ← CL - 1 {since 1st element is
                                      loaded }
11)   LEA SI, value  ;  SI ← offset of value {point to
                                      start of array}
12)   mov AL, [SI]    ;  AL ← value [0]

13)   up: inc SI      ;  SI ← SI + 1
14)   Cmp AL, [SI]    ;  Compare AL with value [SI]
15)   JNC nxt         ;  if AL >= value [SI], jump to nxt
16)   mov AL, [SI]    ;  AL ← value [SI]  (new max found)
```

```
17) nxt : dec  CL      ; CL ← CL - 1
18) jnz  up            ; if CL != 0, jump to up (repeat loop)
19) LEA  DI, res       ; DI ← offset of res
20) mov  [DI], al      ; res ← AL (store max value in res)
21) END  MAIN          ; end of program.
```

② INPUT / OUTPUT ANALYSIS :

(a) Input :

Count = 4

value = [09h, 10h, 05h, 03h] → [9, 16, 5, 3] (in decimal)

(b) working :

initial AL = 9

Compare with next values :

9 < 16 → AL = 16

16 > 5 → no change

16 > 3 → no change

(c) output :

res = 10h → 16 (maximum value)

③ OBSERVATION TABLE

⊙ Input:

Sl. No.	Memory location	(Data) Operand
1	1001h	04(h)
2.	1002h	10h(16)
3.	1003h	05h(5)
4.	1004h	03h(3)

⊙ output

S.NO	Memory location	(Data) operand
1	1005h	10h(16)

OBJECTIVE - ② - Arrange the elements (8 bit number) of a given array of size N in ascending/descending order.

① ASSEMBLY CODE:

```
1).DATA
2) count DB 06      ; Define count = 6 (number of elements)
3) value DB 0FH, 09H, 14H, 45H, 3FH, 24H;  array of 8 bit elements

4) .CODE
6) MAIN PROC
7)   mov AX, DATA   ; AX ← Address of data segment
8)   mov DS, AX     ; DS ← AX (initialise Data Segment Register)
9)   LEA DI, count  ; DI ← offset address of count variable.
10)  mov CH, [DI]   ; CH ← value at [DI] ⇒ CH ← count (06H)
```

11) DEC CH ; CH ← CH - 1 ⇒ CH ← 05H

12) UP2 : MOV CL, CH ; CL ← CH (inital loop counter)

13) LEA SI, value ; SI ← offset address of every 1st array element.

14) UP1 : mov AL, [SI] ; AL ← value at [SI] (current element)

15) CMP AL, [SI+1] ; Compare AL with [SI+1] (next element)

16) JC DOWN ; if AL < [SI+1], jump to DOWN (no swap)

above for ascending ORDER.

* JNC DOWN ; if AL ≥ = [SI+1], jump to DOWN (no swap)

↳ for DECENDING ORDER

17) MOV DL, [SI+1] ; DL ← [SI+1] (stores nt element in DL)

18) XCHG [SI], DL ; Exchange DL with [SI] ⇒ [SI] ⇔ DL

19) MOV [SI+1], DL ; [SI+1] ← DL (original [SI])

20) DOWN : INC SI ; SI ← SI + 1 (move to next element)

21) DEC CL ; CL ← CL - 1 (decrease inner loop counter)

22) JNZ UP1 ; if CL ≠ 0, jump to UP1 (cont. inner loop)

23) DEC CH ; CH ← CH - 1 (dec. outer loop counter)

24) JNZ UP2 ; If CH ≠ 0, jump to UP2 (nxt outer loop path)

25) END MAIN ;

② INPUT / OUTPUT ANALYSIS :

(a) Input :

- Count = 6
- Value = [0FH, 09h, 14h, 45h, 3Fh, 24h]
  → [15, 9, 20, 69, 63, 36] (in decimal)

(b) Program Action :

- The program uses bubble sort logic
- Compares adjacent elements
- if current element is greater than next, they are swapped. (asc order)
  OR
- if current element is smaller than next, they are swapped (descending order)
- This is repeated (count - 1) times to sort the entire array in required order.

(c) Output :

- Value - (for ascending order)
  [09h, 0FH, 14h, 24h, 3Fh, 45h] → [9, 15, 20, 36, 63, 69] (in decimal)

- Value - for descending order)
  [45h, 3Fh, 24h, 14h, 0Fh, 09h] → [69, 63, 36, 20, 15, 9] (in decimal)

③ OBSERVATION TABLE: (a) for ascending orders

Input

S.No	Memory location	Data
①	0710 : 0000	06
②	0710 : 0001	0f
③	0710 : 0002	09
④	0710 : 0003	14
⑤	0710 : 0004	45
⑥	0710 : 0005	3f
⑦	0710 : 0006	24

Output

S.No.	Memory location	Data
①	0710: 0000	06
②	0710: 0001	09
③	0710: 0002	0f
④	0710: 0003	14
⑤	0710: 0004	24
⑥	0710: 0005	3f
⑦	0710: 0006	45

(b) for Descending order:

Input

S.No.	Memory location	Data
①	0710: 0000	06
②	0710 : 0001	0f
③	0710 : 0002	09
④	0710 : 0003	14
⑤	0710: 0004	45
⑥	0710: 0005	3f
⑦	0710: 0006	24

Output

S.No.	Memory location	Data
①	0710: 0000	06
②	0710:0001	45
③	0710:0002	3f
④	0710:0003	24
⑤	0710:0004	14
⑥	0710: 0005	0f
⑦	0710: 0006	09

CONCLUSION: In this lab, we implemented fundamental 8086 assembly program like finding the largest / smallest number and arranging the elements in ascending / descending order. We learnt how data processed using registers & memory, improving our understanding on assembly programming.

POST LAB:

(Q1) What are the directives available for data declaration in 8086 microprocessors ?

Soln) The directives available are :

DB (Define Byte): Allocate 1 Byte of memory
DD (Define Doubleword): Allocate 4 bytes
DQ (Define Quadword): Allocate 8 bytes
DT (Define Ten bytes): Allocate 10 bytes
DW (Define word): Allocate 2 byte.

(Q2) State the difference b/w END, ENDP, ENDS directives ?

Soln)

END	ENDP	ENDS
Indicates the end of assembly program. It's placed at the last line of Program - END Program (whole).	marks the end of a procedure within the program. It used when defining the procedures - END Procedure)	Signifies the end of a segment (a block of code, data / stack) - END segment

(Q3) find the sum & average of a given array of size N.

Sol:)

```
.DATA
ARRAY DB 10, 20, 30, 40, 50
N DB 5
SUM DW 0
AVG DW 0

.CODE
MAIN:
MOV AX, @DATA
MOV DS, AX
MOV SI, 0
MOV CX, N
MOV BX, 0

FIND_SUM:
MOV AL, ARRAY[SI]
ADD BL, AL
INC SI
LOOP FIND_SUM
MOV SUM, BX
MOV AX, BX
MOV CX, N
MOV DX, 0
DIV CX.
MOV AVG, AX
END MAIN.
```