

Experiment 07

Experiment 7: Loop operations in 8086 Assembly language programming

1. Program to move a set of numbers from one memory block to another.
2. Program to count the number of 1's and 0's in a given 16 bit number.
3. Write an ALP to find the smallest number from the given array.
4. Write an ALP to sort a given set of 16bit unsigned integers into ascending order using a bubble sort algorithm.

--

Roll No.	17
Name	Manav Jawrani
Class	D10-A
Subject	Microprocessor Lab
LO Mapped	LO4: Develop the assembly level programming using 8086 loop instruction set

Aim: Experiment 7: Loop operations in 8086 Assembly language programming

1. Program to move a set of numbers from one memory block to another.
2. Program to count the number of 1's and 0's in a given 16 bit number.
3. Write an ALP to find the smallest number from the given array.
4. Write an ALP to sort a given set of 16bit unsigned integers into ascending order using a bubble sort algorithm.

Introduction:

A **loop** is a block of statements that are repeatedly executed until a condition is satisfied. The assembly language uses **JMP** instruction to implement loops. However, the processor set can use the **LOOP** instruction to implement loops conveniently.

Syntax and explanation:

The following code snippet illustrates how a loop is implemented through JMP instruction:

```
mov AL, 5    ; store the number of iteration in AL
```

```
L1: (loop code)
```

```
DEC AL      ; decrement AL
```

```
JNZ L1
```

- The number of iterations of the loop is stored in AL.
- At the end of each iteration, the code decrements AL, then takes a conditional jump if AL is not zero.

Theory:

1. Program to move a set of numbers from one memory block to another.

Algorithm:

Step 1 - Start the program and create a set of numbers which is to be moved from one memory block to another.

Step 2 - Create an empty array of 5 elements.

Step 3 - Move content of DATA to register 'ax' and content of register 'ax' to 'ds'.

Step 4 - Initialize the loop counter and move the 'cx' register to the source of the empty array (i.e 5).

Step 5 - Move the 'SI' to address locations of 'B1' array and the 'DI' to address locations of 'B2' array by using 'offset' command.

Step 6 - Move the 'ax' register to 'SI' which will copy the contents of 'B1' array to 'B2' array and move the content of 'DI' to 'ax' register.

Step 7 - Now increase the index of 'SI' and 'DI'.

Step 8 - Terminate the program.

Code:

```
.MODEL SMALL
.STACK 100H
.DATA
    B1 DW 2110H , 3112H , 4113H , 5114H , 6115H
    B2 DW 5 DUP(0)
.CODE
    MOV AX , @DATA
    MOV DS , AX
    MOV CX , 5
    MOV SI , OFFSET B1
    MOV DI , OFFSET B2
UP:
    MOV AX , [SI]
    MOV [DI] , AX
    ADD SI , 2
    ADD DI , 2
    LOOP UP
    MOV AH , 4CH
    INT 21H
END
```

Input

Set of numbers - 2110H , 3112H , 4113H , 5114H , 6115H

Output:

```

File View Run Breakpoints Data Options Window Help
[ ]=CPU 80486
44AD:0000 B8AF44      mov     ax,44AF
44AD:0003 8ED8          mov     ds,ax
44AD:0005 B90500          mov     cx,0005
44AD:0008 BE0000          mov     si,0000
44AD:000B BF0A00          mov     di,000A
44AD:000E 8B04          mov     ax,[si]
44AD:0010 8905          mov     [di],ax
44AD:0012 83C602          add     si,0002
44AD:0015 83C702          add     di,0002
44AD:0018 E2F4          loop    000E
44AD:001A B44C          mov     ah,4C
44AD:001C CD21          int     21
44AD:001E 0000          add     [bx+si],al
449D:0000 CD 20 FF 9F 00 EA FF FF = f 0
449D:0008 AD DE E5 01 00 15 AF 01  ; 0 0 15 AF 01
449D:0010 00 15 7D 02 1C 0F 92 01  ; 0 15 7D 02 1C 0F 92 01
449D:0018 FF FF FF FF FF FF FF FF

ax 0192  c=1
bx 094F  z=0
cx CD8C  s=1
dx F68D  o=0
si 0142  p=0
di F690  a=0
bp 0100  i=1
sp 0106  d=1
ds 1D09
es F69E
ss 0192
cs 0000
ip 0000

44B1:0102 0000
44B1:0100 0000

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

```

2. Program to count the number of 1's and 0's in a given 16 bit number.

Algorithm:

Step 1 - Start the program and consider a 16 bit number.

Step 2 - Create variables z and o to store the zero's and one's respectively.

Step 3 - Move content of DATA to register 'ax' and content of register 'ax' to 'ds'.

Step 4 - Move the 'ax' register to 'no'.

Step 5 - Point the 'bx', 'cx' and 'dx' towards the index source of '00H', '01H', '02H' respectively.

Step 6 - Now perform the left rotation of 'ax' register and perform a conditional jump to level 1.

Step 7 - Create a loop to satisfy the given condition,

Step 8 - Move the number of zero's(i.e z) to 'bx' register and the number of one's(i.e o) to 'dx',

Step 9 - Terminate the program.

Code:

```
data segment
no dw 5000h
z dw ?
o dw ?
data ends
code segment
assume cs:code, ds:data
start:
mov ax,data
mov ds,ax
mov ax,no
mov bx,00h
mov cx,10h
mov dx,00h
up:
rol ax,1
jc one
inc bx
jmp nxt
one:
inc dx
nxt:
dec cx
jnz up
mov z, bx
mov o, dx
int 3
code ends
```

end start

Input:

16 bit number = 5000h

Binary representation - 0101000000000000

No. of zero's = 14

No. of one's = 2

Output:

```
File View Run Breakpoints Data Options Window Help
[ ]=CPU 80486 ds:0004 = 0000 1=[ ]
cs:0011 D1C0 rol ax,1 ax 5000 c=0
cs:0013 7204 jb 0019 bx 000E z=1
cs:0015 43 inc bx cx 0000 s=0
cs:0016 EB02 jmp 001A dx 0002 o=0
cs:0018 90 nop si 0000 p=1
cs:0019 42 inc dx di 0000 a=0
cs:001A 49 dec cx bp 0000 i=1
cs:001B 75F4 jne 0011 sp 0000 d=0
cs:001D 891E0200 mov [0002],bx ds 44AD
cs:0021 89160400 mov [0004],dx es 449D
cs:0025 CC int 03 ss 44AC
cs:0026 0000 add [bx+si],al cs 44AE
cs:0028 0000 add [bx+si],al ip 0021
es:0000 CD 20 FF 9F 00 EA FF FF = f 0
es:0008 AD DE E5 01 00 15 AF 01 : 0 0 0 0
es:0010 00 15 7D 02 1C 0F 92 01 0 0 0 0
es:0018 01 01 01 00 02 FF FF FF 0 0 0 0
ss:0002 6568
ss:0000 6474
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu
```

3. Write an ALP to find the smallest number from the given array.

Algorithm:

Step 1 - Start the program and create an array of numbers.

Step 2 - Create a variable 'SMALLEST' to store the number that is to be found.

Step 3 - Move content of DATA to register 'ax' and content of register 'ax' to 'ds'.

Step 4 - Assign '0' to 'bx' register.

Step 5 - Move the content of 'al' to array[bx] and 'SMALLEST' to 'al' register.

Step 6 - Compare 'bx' and exit if the jump is equal.

Step 7 - Now again move the content of 'al' to array[bx] and compare with the value stored in 'SMALLEST'.

Step 8 - Update the 'SMALLEST' and perform jumps and compare it with a given array to find the smallest number, and do this until the smallest number is found.

Step 9 - Move the 'SMALLEST' to 'al' register and terminate the program.

Code:

```
.MODEL SMALL
```

```
.STACK 100H
```

```
.DATA
```

```
ARRAY DB 55, 32, 98, 21, 13, 16, 38, 25, 56, 12
```

```
SMALLEST DB ?
```

```
.CODE
```

```
MAIN PROC
```

```
    MOV AX, @DATA
```

```
    MOV DS, AX
```

```
    MOV BX, 0
```

```
    MOV AL, ARRAY[BX]
```

```
    MOV SMALLEST, AL
```

```
COMPARE:
```

```
    INC BX
```

```
    CMP BX, 10
```

```
    JE EXIT
```

```
    MOV AL, ARRAY[BX]
```

```
    CMP AL, SMALLEST
```

```

JL UPDATE_SMALLEST
JMP COMPARE
UPDATE_SMALLEST:
MOV SMALLEST, AL
JMP COMPARE
EXIT:
MAIN ENDP
END MAIN

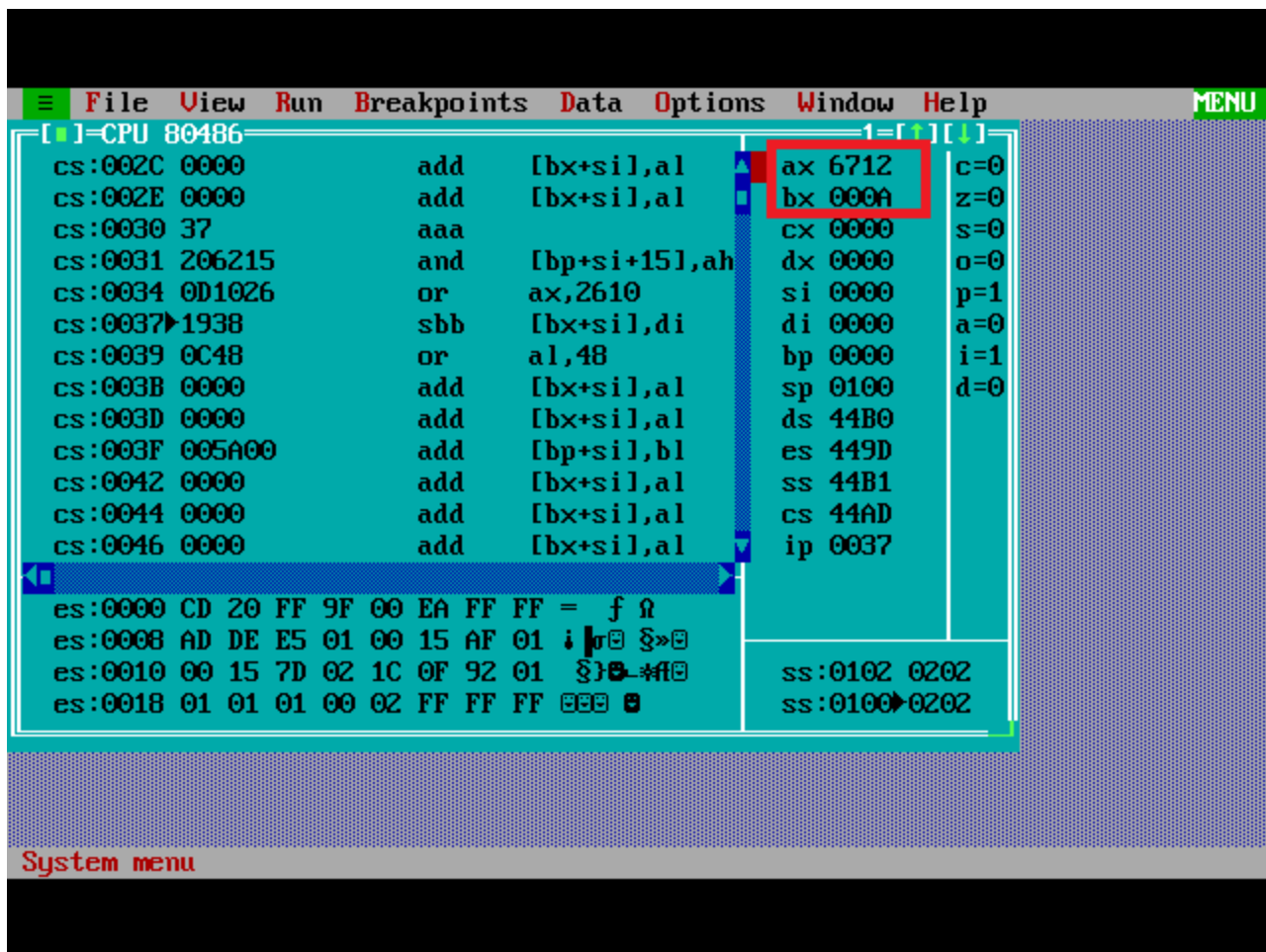
```

Input:

Array of numbers - 55, 32, 98, 21, 13, 16, 38, 25, 56, 12

Here the smallest number is '12' and it's position is 10(i.e A)

Output:



4. Write an ALP to sort a given set of 16bit unsigned integers into ascending order using a bubble sort algorithm.

Algorithm:

Step 1 - Start the program and write a set of 16bit unsigned integers.

Step 2 - Move content of DATA to register 'ax' and content of register 'ax' to 'ds'.

Step 3 - Move 'SI' towards '0000H' index source and 'bx' to 'a[SI]'.

Step 4 - Move the content of 'cx' register to 'bx' register and 'SI' towards '02H' index source.

Step 5 - Now move the content of 'ax' to 'a[SI]' and do an increment in the index 'SI'.

Step 6 - Compare 'ax' and 'a[SI]'. Perform an above jump and exchange the content of 'ax' and 'a[SI]'.

Step 7 - Move 'A[SI-2]' to 'ax' and create a loop.

Step 8 - Do a decrement in 'bx' and do a conditional jump.

Step 9 - Terminate the program.

Code:

DATA SEGMENT

A DW 0005H, 0ABCDH, 5678H, 1234H, 0EFCDH, 45EFH

DATA ENDS

ASSUME CS:CODE,DS:DATA

CODE SEGMENT

START: MOV AX,DATA

MOV DS,AX

MOV SI,0000H

MOV BX,A[SI]

DEC BX

X2: MOV CX,BX

MOV SI,02H

X1: MOV AX,A[SI]

INC SI

INC SI

```
CMP AX,A[SI]
JA X3
XCHG AX,A[SI]
MOV A[SI-2],AX
X3: LOOP X1
DEC BX
JNZ X2
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

Input:

Numbers - 0005H, 0ABCDH, 5678H, 1234H, 0EFCDH, 45EFH

Output:

The screenshot displays an 8086 assembly debugger interface. The main window shows a list of instructions being executed. The instruction at address CS:002D is highlighted, showing 'int 21'. To the right, a register window shows the current values of various registers: ax=4CCD, bx=0000, cx=0000, dx=0000, si=0004, di=0000, bp=0000, sp=0000, ds=44AD, es=449D, ss=44AC, cs=44AE, and ip=002D. Below the instruction list, a memory dump shows the contents of memory locations starting from es:0000. At the bottom, a status bar provides function key shortcuts: F1-Help, F2-Bkpt, F3-Mod, F4-Here, F5-Zoom, F6-Next, F7-Trace, F8-Step, F9-Run, and F10-Menu.

Address	Instruction	Register/Value
cs:002B B44C	mov ah,4C	ax 4CCD
cs:002D CD21	int 21	bx 0000
cs:002F 0000	add [bx+si],al	cx 0000
cs:0031 0000	add [bx+si],al	dx 0000
cs:0033 0000	add [bx+si],al	si 0004
cs:0035 0000	add [bx+si],al	di 0000
cs:0037 0000	add [bx+si],al	bp 0000
cs:0039 0000	add [bx+si],al	sp 0000
cs:003B 0000	add [bx+si],al	ds 44AD
cs:003D 0000	add [bx+si],al	es 449D
cs:003F 0000	add [bx+si],al	ss 44AC
cs:0041 0000	add [bx+si],al	cs 44AE
cs:0043 0000	add [bx+si],al	ip 002D

Memory Dump:

```

es:0000 CD 20 FF 9F 00 EA FF FF = f 0
es:0008 AD DE E5 01 00 15 AF 01 ; 0 0 0
es:0010 00 15 7D 02 1C 0F 92 01 0 0 0
es:0018 01 01 01 00 02 FF FF FF 0 0 0
  
```

Status Bar: F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

Conclusion:

We have successfully understood and performed all the programs using 8086 Assembly language programming.