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).

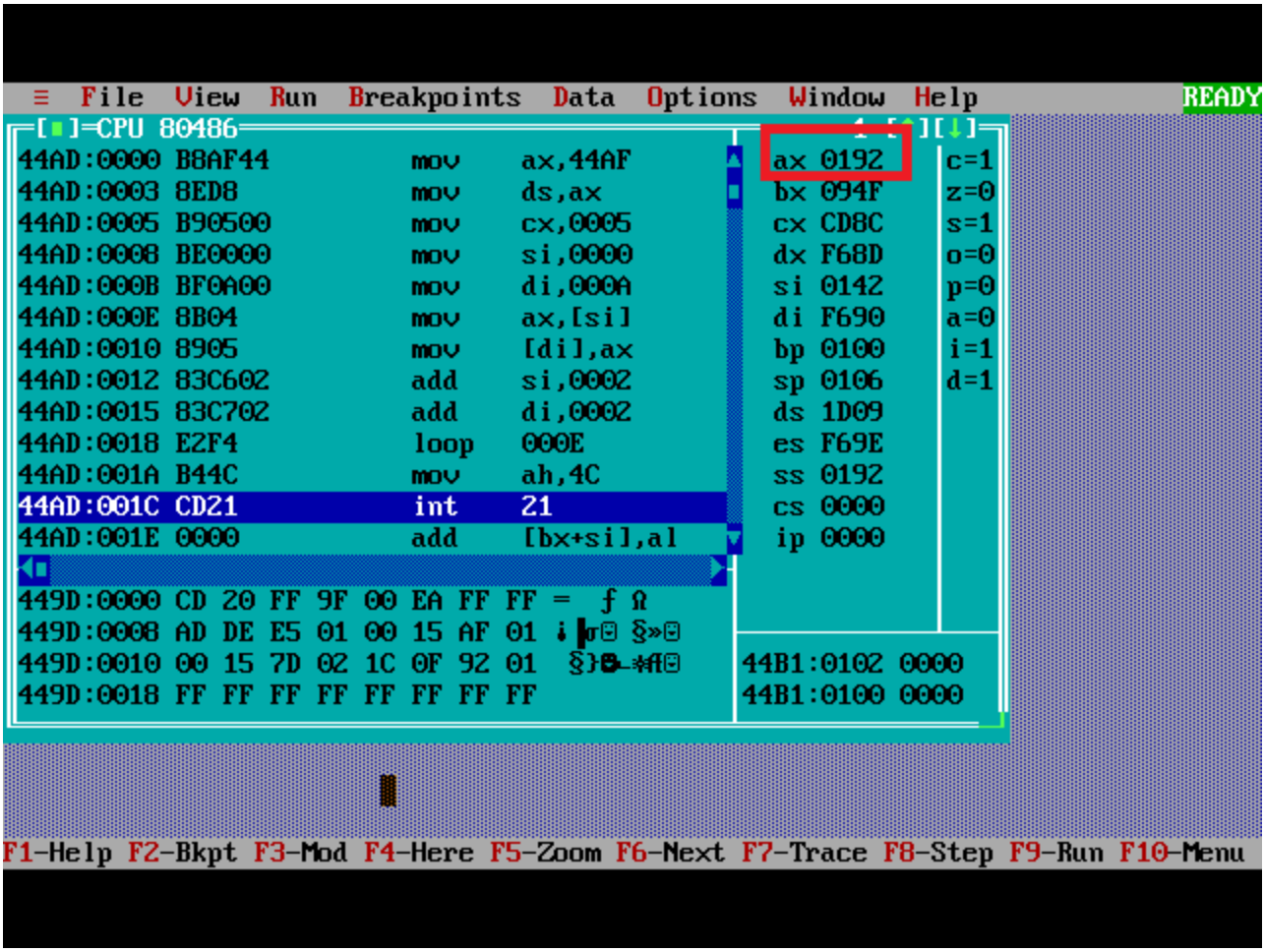
Step 5 - Move the ‘SI’ to address locations of ‘B1’ array and the ‘DI’ to address locations of ‘B2’ array by using ‘offset’ command.
Step 6 - Move the ‘ax’ register to ‘SI’ which will copy the contents of ‘B1’ array to ‘B2’ array and move the content of ‘DI’ to ‘ax’ register.
Step 7 - Now increase the 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:



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:

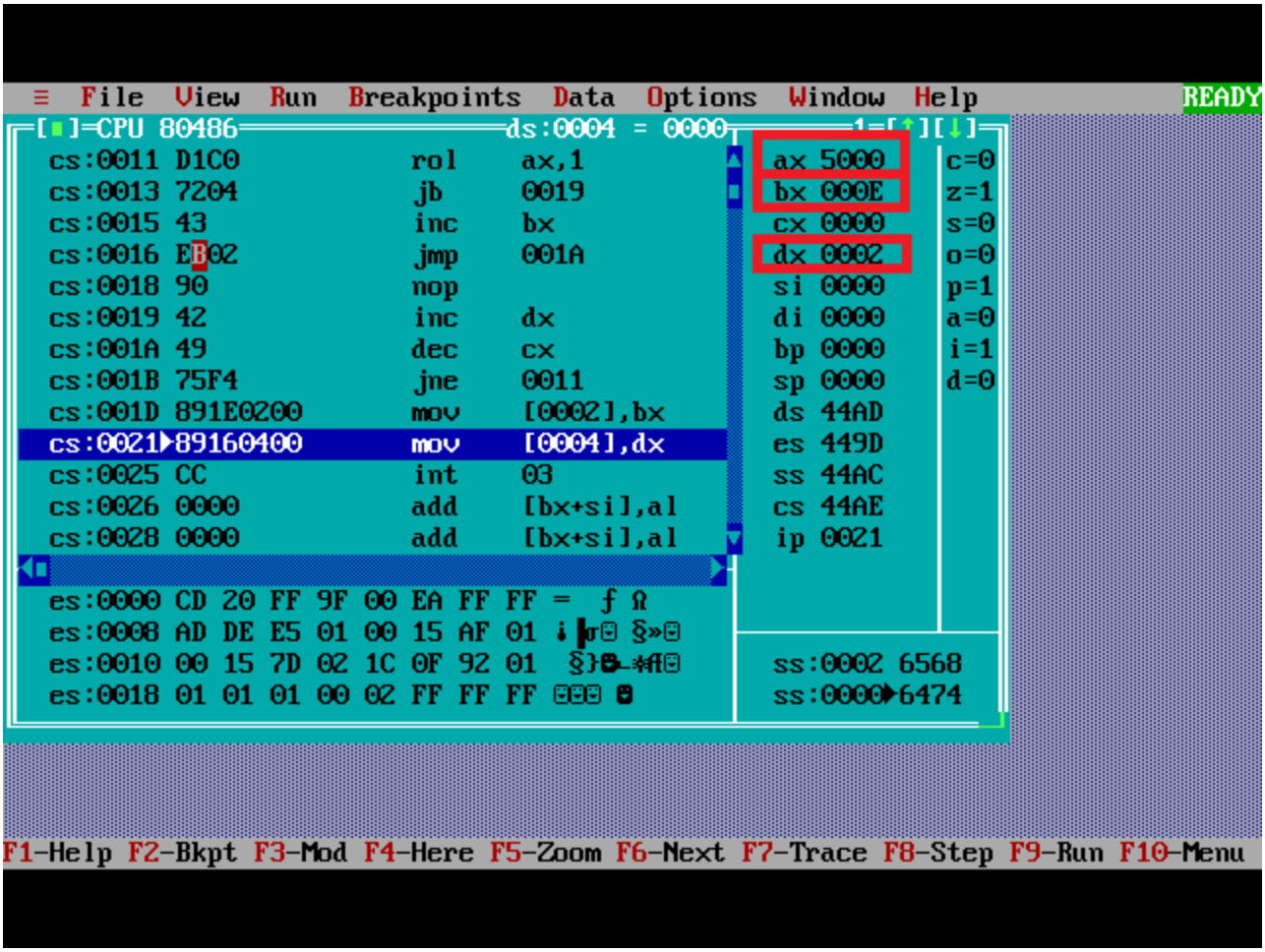
16 bit number = 5000h

Binary representation - 0101000000000000

No. of zero's = 14

No. of one's = 2

Output:



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
```

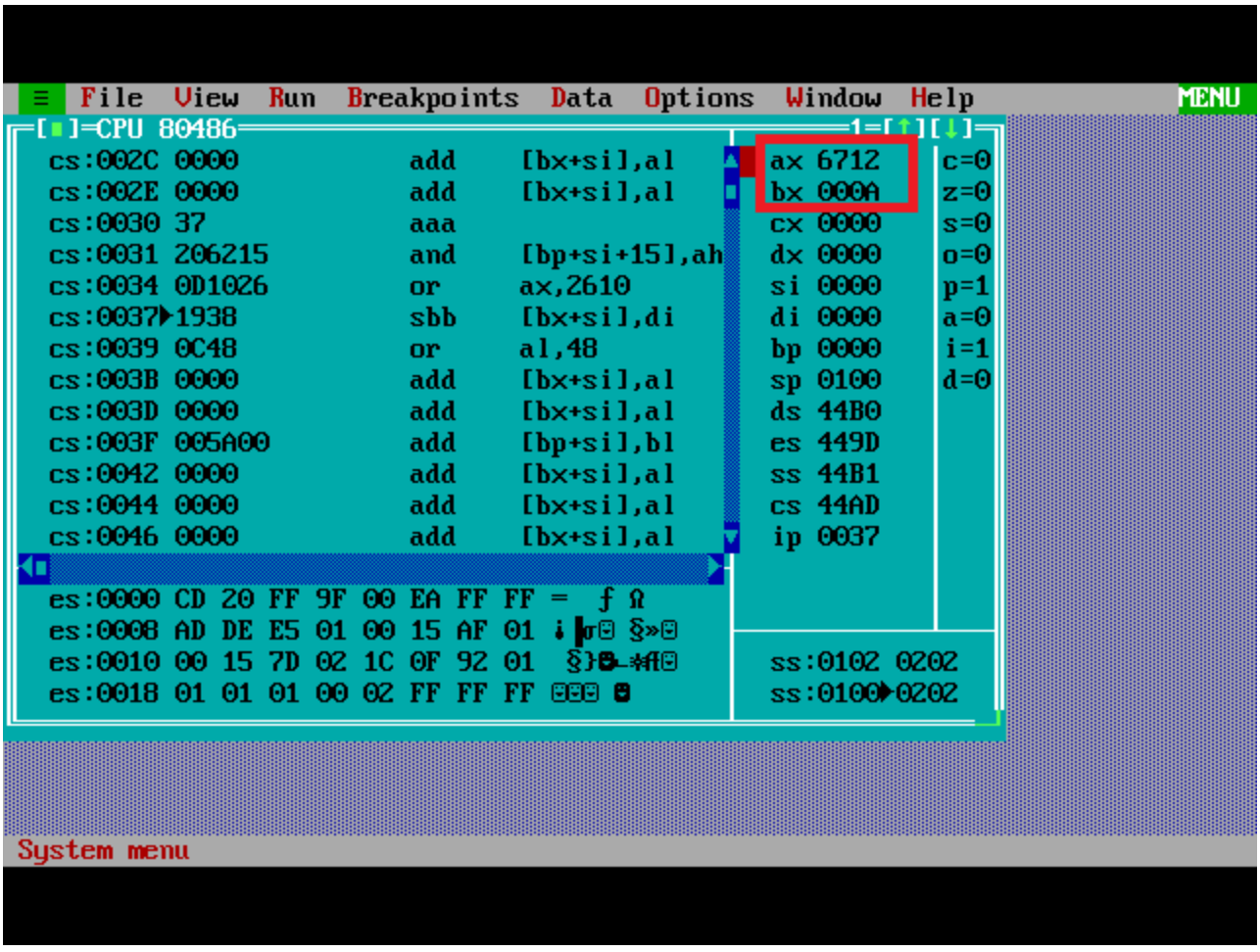
```
    JL UPDATE_SMALLEST
    JMP COMPARE
    UPDATE_SMALLEST:
    MOV SMALLEST, AL
    JMP COMPARE
    EXIT:
    MAIN ENDP
END MAIN
```

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.

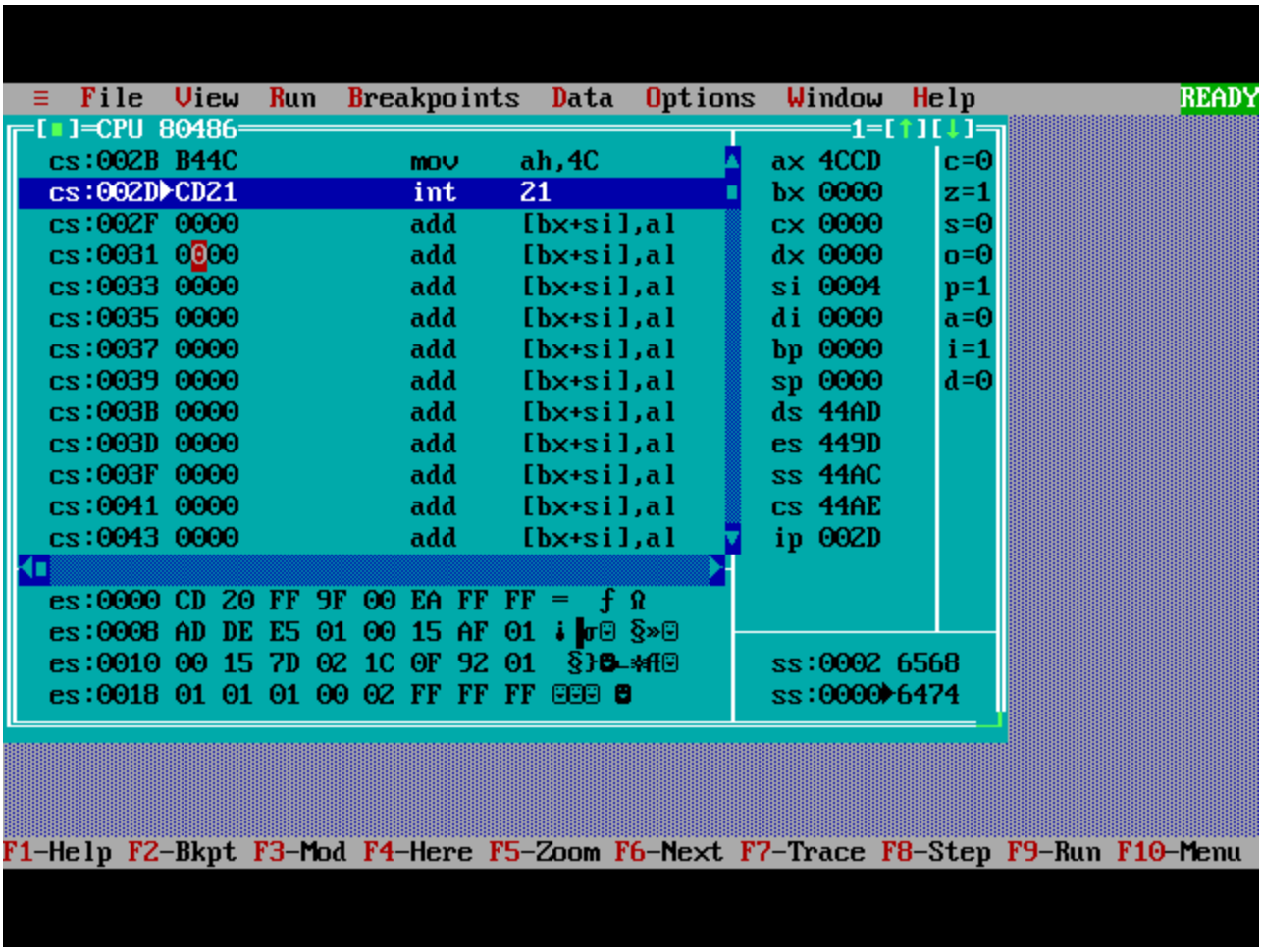
Step 9 - Terminate the program.

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:



Conclusion:

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