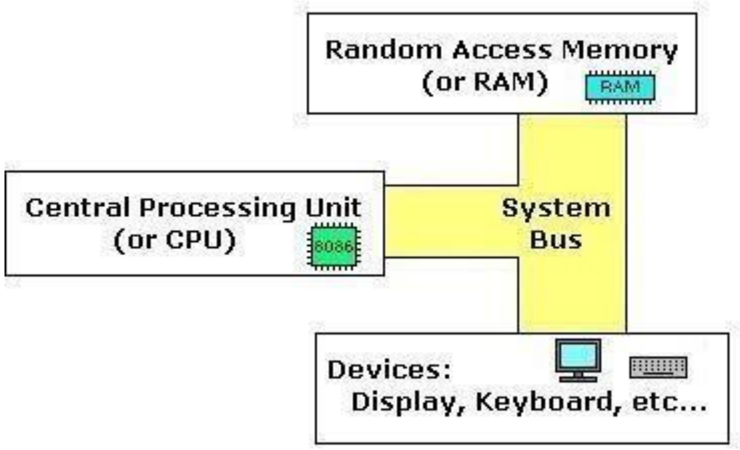
PRACTICAL - 1

AIM: Introduction to 8086 Microprocessor & Assembly Language Programming.

THEORY: Assembly language is a low level programming language. you need to get some knowledge about computer structure in order to understand anything. the simple computer model as i see it:



the **system bus** (shown in yellow) connects the various components of a computer.

The **CPU** is the heart of the computer, most of computations occur inside the **CPU**.

RAM is a place to where the programs are loaded in order to be executed.

general purpose registers

8086 CPU has 8 general purpose registers, each register has its own name:

- **AX** - the accumulator register (divided into **AH / AL**).
- **BX** - the base address register (divided into **BH / BL**).
- **CX** - the count register (divided into **CH / CL**).
- **DX** - the data register (divided into **DH / DL**).
- **SI** - source index register.
- **DI** - destination index register.
- **BP** - base pointer.
- **SP** - stack pointer.

despite the name of a register, it's the programmer who determines the usage for each general purpose register. the main purpose of a register is to keep a number (variable). the size of the above registers is 16 bit, it's something like: **0011000000111001b** (in binary form), or **12345** in decimal (human) form.

4 general purpose registers (AX, BX, CX, DX) are made of two separate 8 bit registers, for example if

AX= **0011000000111001b**, then AH=**00110000b** and AL=**00111001b**. therefore, when you modify any of the 8 bit registers 16 bit register is also updated, and vice-versa. the same is for other 3 registers, "H" is for high and "L" is for low part.

because registers are located inside the cpu, they are much faster than memory. accessing a memory location requires the use of a system bus, so it takes much longer. accessing data in a register usually takes no time. therefore, you should try to keep variables in the registers. register sets are very small and most registers have special purposes which limit their use as variables, but they are still an excellent place to store temporary data of calculations.

segment registers

- **CS** - points at the segment containing the current program.
- **DS** - generally points at segment where variables are defined.
- **ES** - extra segment register, it's up to a coder to define its usage.
- **SS** - points at the segment containing the stack

although it is possible to store any data in the segment registers, this is never a good idea. the segment registers have a very special purpose - pointing at accessible blocks of memory.

special purpose registers

- **IP - the instruction pointer.**
- **flags register - determines the current state of the microprocessor.**

IP register always works together with CS segment register and it points to currently executing instruction.

CONCLUSION: In this practical we get to know about the basics of 8086 Microprocessor and Assembly Programming Language.

PRACTICAL – 2

AIM: Store the data byte 32H into memory location 4000H.

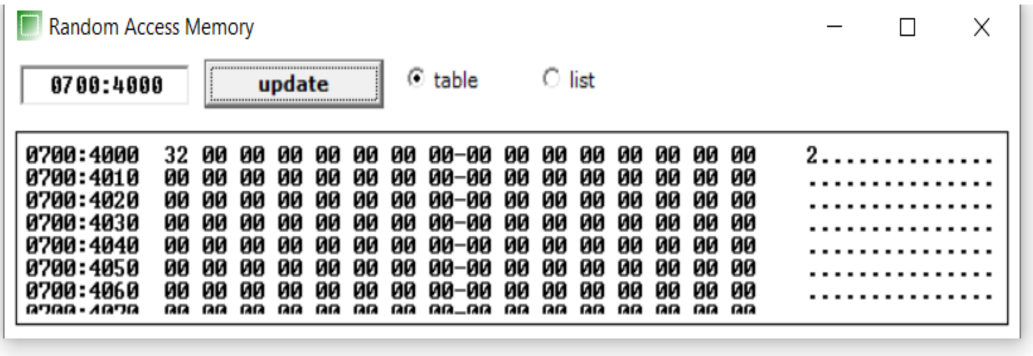
CODE:

org 100h

MOV [4000H],32H

ret

OUTPUT:



CONCLUSION: In this practical we learned how to store data at specific location.

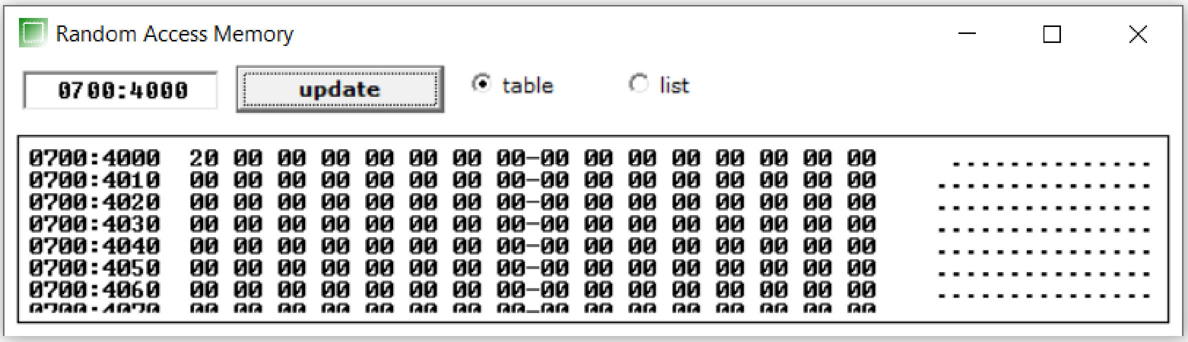
PRACTICAL – 3

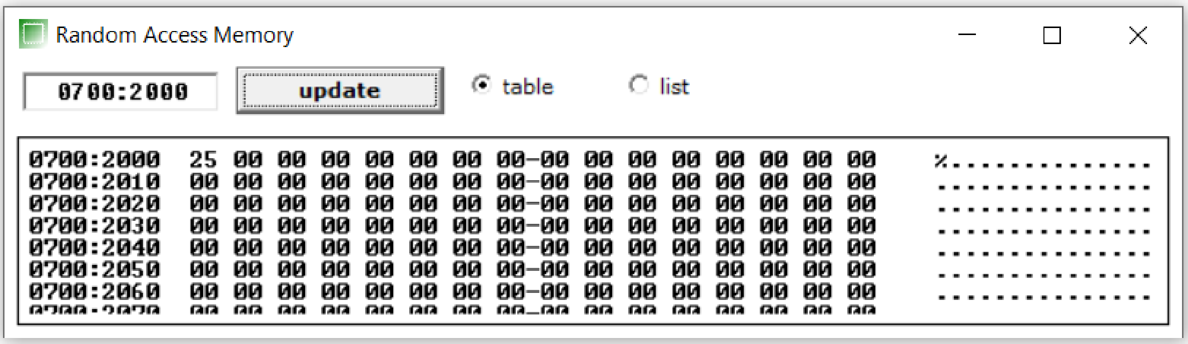
AIM: Exchange the contents of memory locations 2000H and 4000H.

CODE:

```
org 100h
MOV  [2000H],25H
MOV   AL,[2000H]
MOV  [4000H],20H
MOV AH,[4000H]

MOV  AL,[4000H]
MOV AH,[2000H]
ret
```

OUTPUT:





CONCLUSION: From this practical we learn how to transfer data on different memory location.

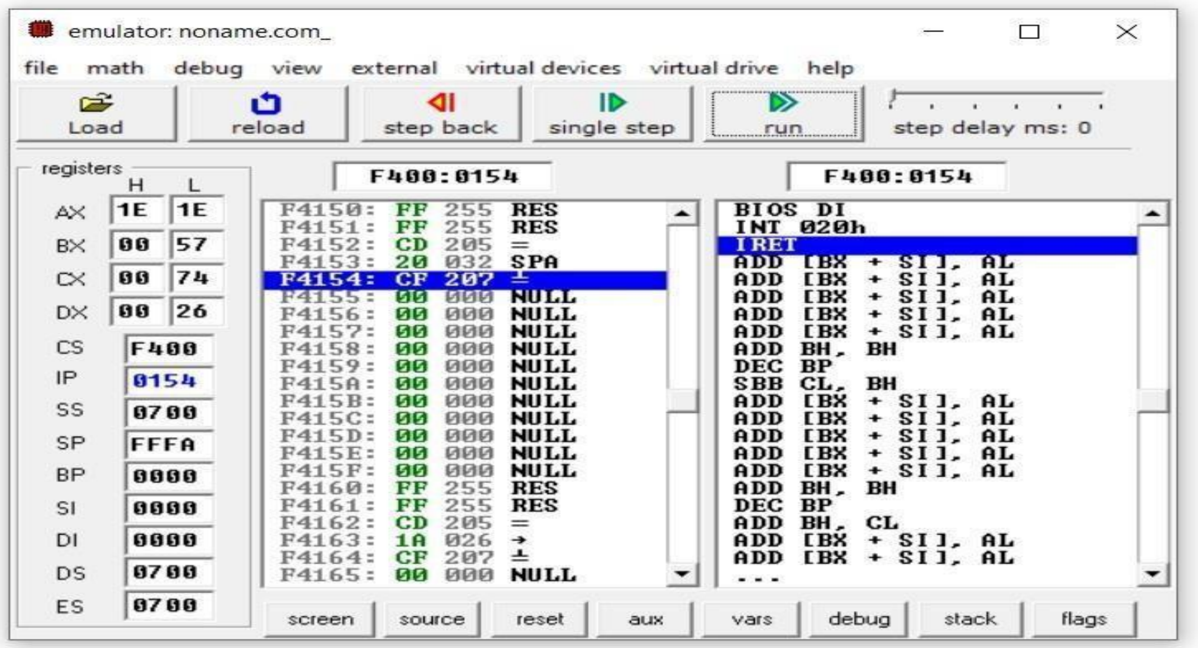
PRACTICAL – 4

AIM: Convert the below given C Program into Assembly Language.

CODE:

```
org 100h
MOV AL,15H
MOV BL,57H
MOV CL,74H
MOV DL,26H
ADD AL,BL
SUB AL,CL
ADD AL,DL
MOV AH,AL
ret
```

OUTPUT:



CONCLUSION: From this practical we learn how to convert programming language into assembly language.

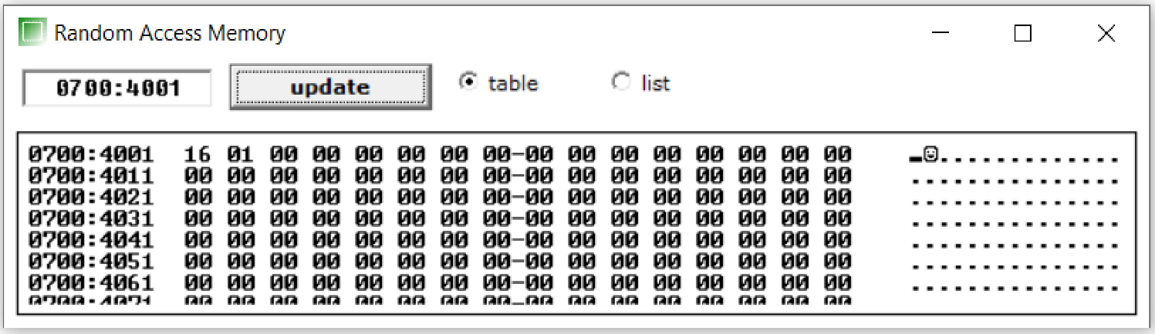
PRACTICAL - 5

AIM: Subtract the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H.

CODE:

```
org 100h


MOV [4001H],16H

MOV AL,[4001H]

MOV [2000H],15H

MOV BL,[2000H]

SUB AL,BL

MOV [4002H],AL

ret
```

OUTPUT:



CONCLUSION: From this practical we learn how to perform certain operation.

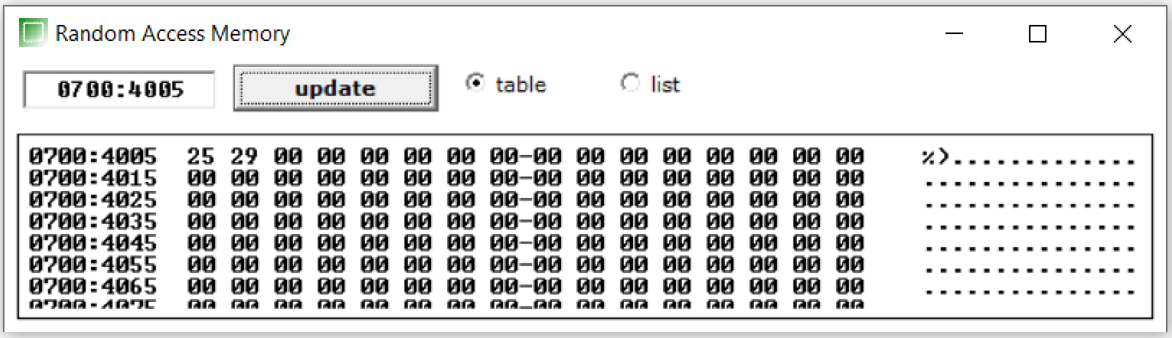
PRACTICAL - 6

AIM: Add the 16-bit number in memory locations 4000H and 4001H to the 16-bit number in memory locations 4002H and 4003H. The most significant eight bits of the two numbers tobe added are in memory locations 4001H and 4003H. Store the result in memory locations 4004H and 4005H with the most significant byte in memory location 4005H.

CODE:

```
org 100h

MOV [4001H],10H

MOV [4002H],09H

MOV AX,[4001H]

MOV [4003H],15H

MOV [4004H],20H

MOV BX,[4003H]

ADD AX,BX

MOV [4005H],AX

ret
```

OUTPUT:



CONCLUSION: From this practical we learn how to perform certain operation like addition on several memory location via registers.

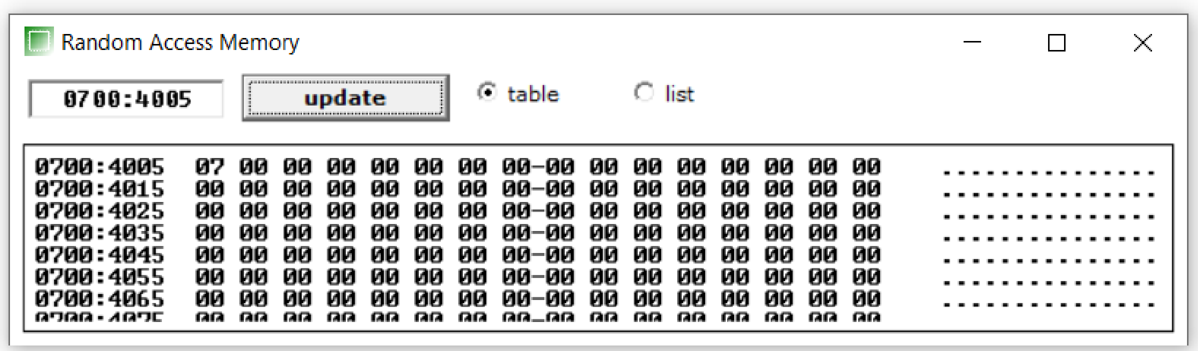
PRACTICAL - 7

AIM: Subtract the 16-bit number in memory locations 4002H and 4003H from the 16-bit number in memory locations 4000H and 4001H. The most significant eight bits of the two numbers are in memory locations 4001H and 4003H. Store the result in memory locations 4004H and 4005H with the most significant byte in Memory location 4005H.

CODE:

```
org 100h

MOV [4000H],7410H

MOV [4002H],9909H

MOV AX,[4000H]

MOV BX,[4002H]

SUB AL,BL

MOV [4005H],AL

ret
```

OUTPUT:



CONCLUSION: From this practical we learn how to use certain operation like sub in several memory location via transferring data into registers.

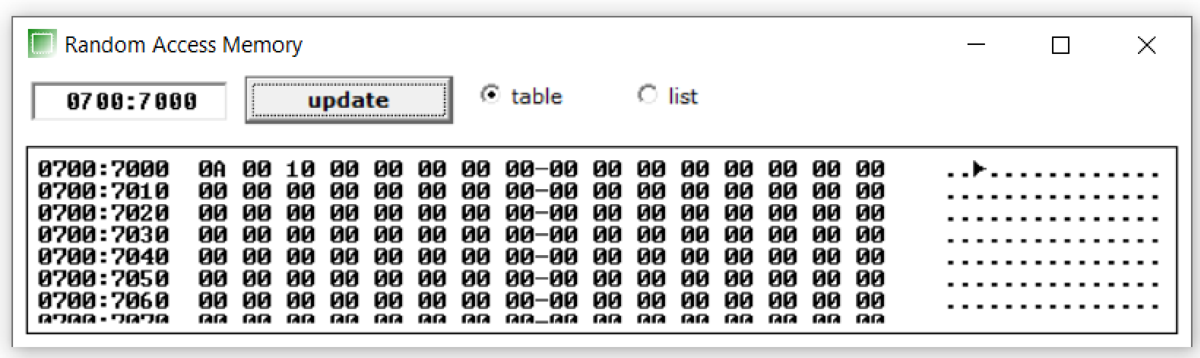
PRACTICAL - 8

AIM: Add Two 32-bit numbers stored in consecutive memory locations and store the result in memory locations starting from 7000H.

CODE:

```
org 100h
MOV [4000H],0002H
MOV [4002H],0009H
MOV AX,[4000H]
MOV BX,[4002H]
MOV [4004H],0008H
MOV [4006H],0007H
MOV CX,[4004H]
MOV DX,[4006H]
ADC AX,CX
ADC BX,DX
MOV [7000H],AX
MOV [7002H],BX
ret
```

OUTPUT:



CONCLUSION: In this practical we learned about how we can store and add 32 bit numbers using 2 16 bit registers and store them to the memory.

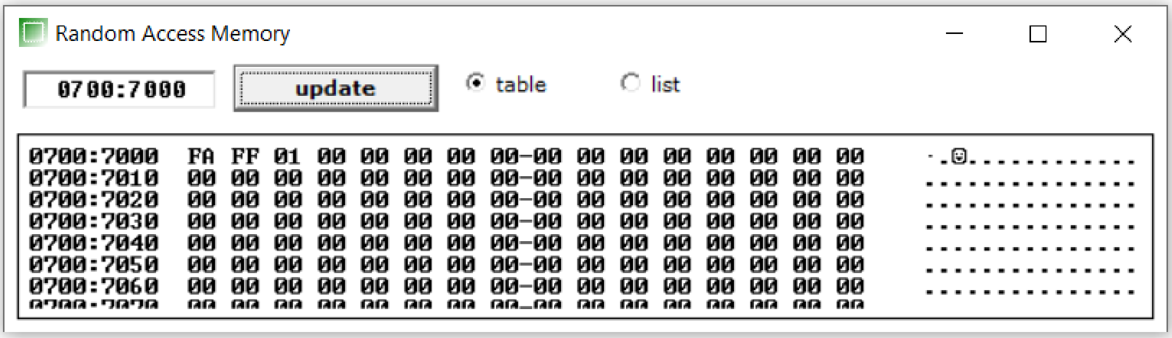
PRACTICAL - 9

AIM: Subtract Two 32-bit numbers stored in consecutive memory locations and store the result in memory locations starting from 7000H.

CODE:

```
org 100h
MOV [4000H],0002H
MOV [4002H],0009H
MOV AX,[4000H]
MOV BX,[4002H]
MOV [4004H],0008H
MOV [4006H],0007H
MOV CX,[4004H]
MOV DX,[4006H]
SBB AX,CX
SBB  BX,DX
MOV [7000H],AX
MOV [7002H],BX
ret
```

OUTPUT:



CONCLUSION: In this practical we learned about how to subtract 2 32 bit numbers using 2 16 bit registers and store them at a particular location.

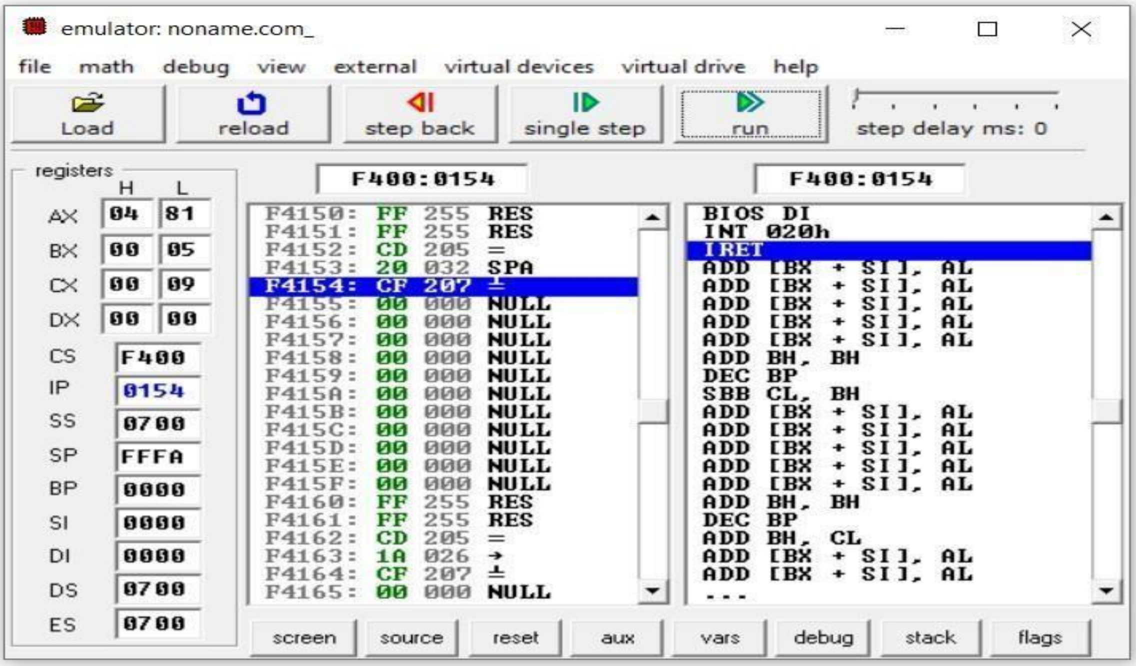
PRACTICAL - 10

AIM: Write an assembly language program to convert temperature in F to C. C=(F-32) * 5/9.

CODE:

```
org 100h

MOV AL,9

SUB AL,32

MOV BL,05

MOV CL,09

MUL BL

DIV CL

ret
```

OUTPUT:



CONCLUSION: In this practical we learned about how we can multiply and divide the data using the registers.

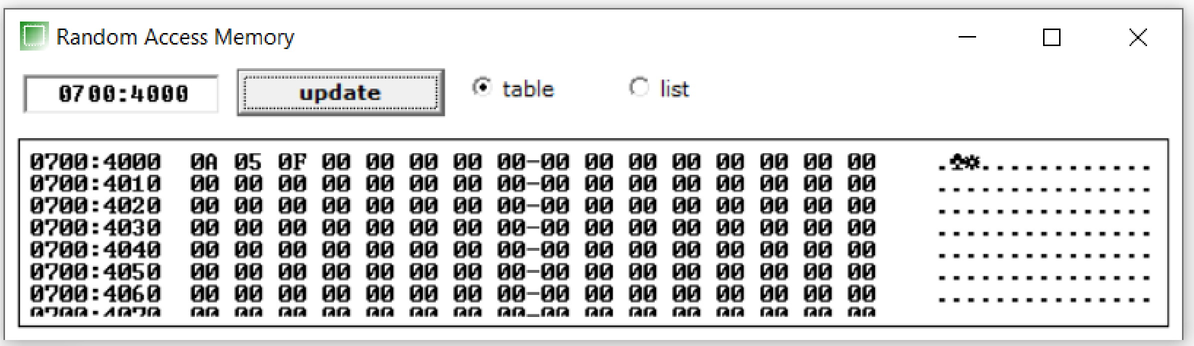
PRACTICAL - 11

AIM: Write a program to perform selective set operation on data stored at 4000H with the data stored at 4001H and store the result at 4002H. Verify the result and write bite wise operation of this program.(OR)

CODE:

```
org 100h
MOV [4000H],1010B;
MOV [4001H],0101B;
MOV AL,[4000H];
MOV BL,[4001H];
OR AL,BL;
MOV [4002H],AL;
ret
```

OUTPUT:



CONCLUSION: In this practical we learned about how the OR operation can be performed in the EMU 8086 register.

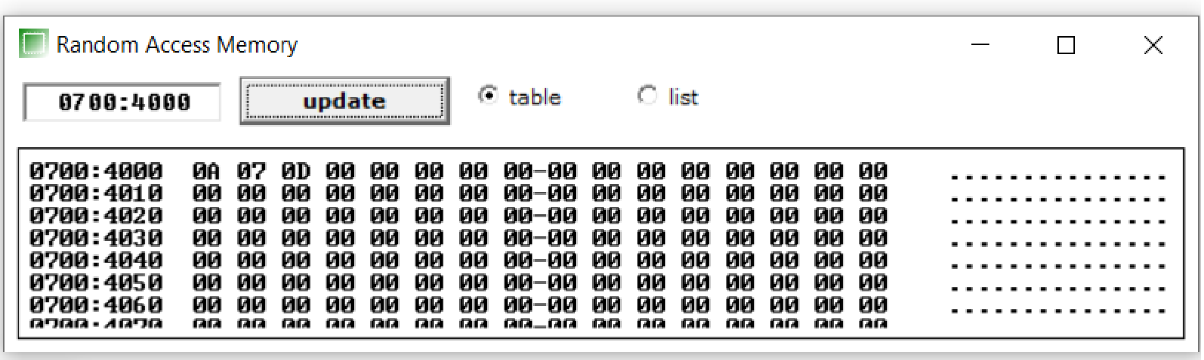
PRACTICAL - 12

AIM: Write a program to perform selective compliment operation on data stored at 4000H corresponding to the data stored at 4001H and store the result at 4002H. Verify the result and write bite wise operation of this program.(XOR)

CODE:

```
org 100h

MOV [4000H],1010B;

MOV [4001H],0101B;

MOV AL,[4000H];

MOV BL,[4001H];

XOR AL,BL;

MOV [4002H],AL;

ret
```

OUTPUT:



CONCLUSION: In this practical we learned about how we can perform selective compliment or XOR operation in 8086 registers.

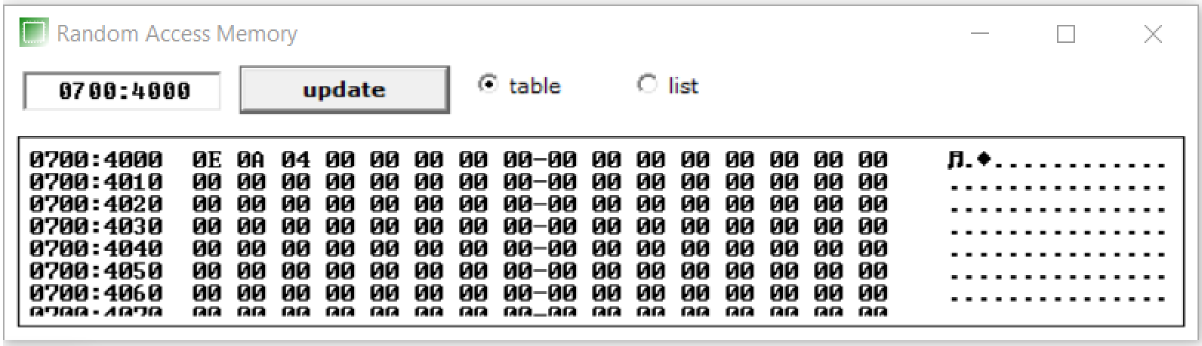
PRACTICAL - 13

AIM: Write a program to perform selective clear operation on data stored at 4000H corresponding to the data stored at 4001H and store the result at 4002H.Verify the result and write bite wise operation of this program. (A AND B')

CODE:

```
org 100h

mov [4000H],1110B;

MOV [4001H],1010B;

MOV AL,[4000H];

MOV BL,[4001H];

NOT BL;

AND AL,BL;

MOV [4002H],AL;

ret
```

OUTPUT:



CONCLUSION: In this practical we learned about using AND and Compliment operation in 8086 register.

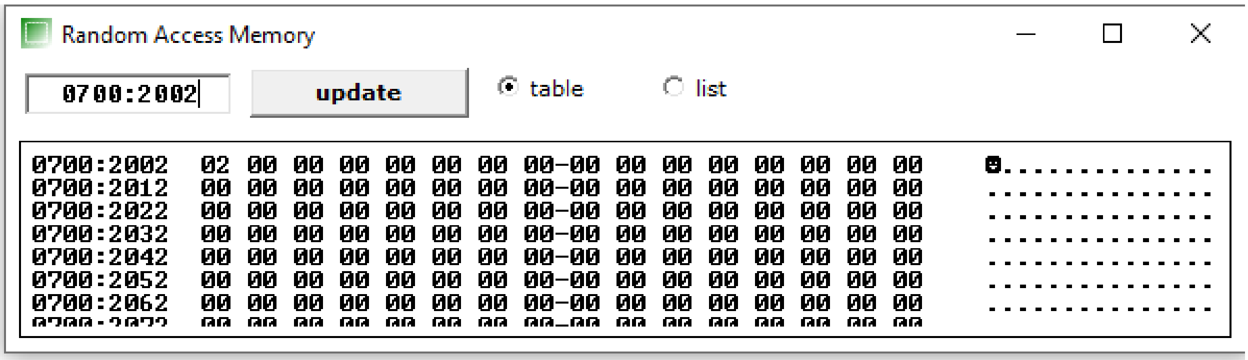
PRACTICAL - 14

AIM: Write an assembly language program the data at memory locations 2000H & 2001H. (Use XOR)

CODE:

```
org 100h
MOV  [2000H],00001111B
MOV  [2001H],00001101B
MOV AL,[2000H]
MOV BL,[2001H]
XOR AL,BL
MOV [2002H],AL
RET
```

OUTPUT:



CONCLUSION: Here we learned about how to perform XOR operation between data of 2 memory locations.

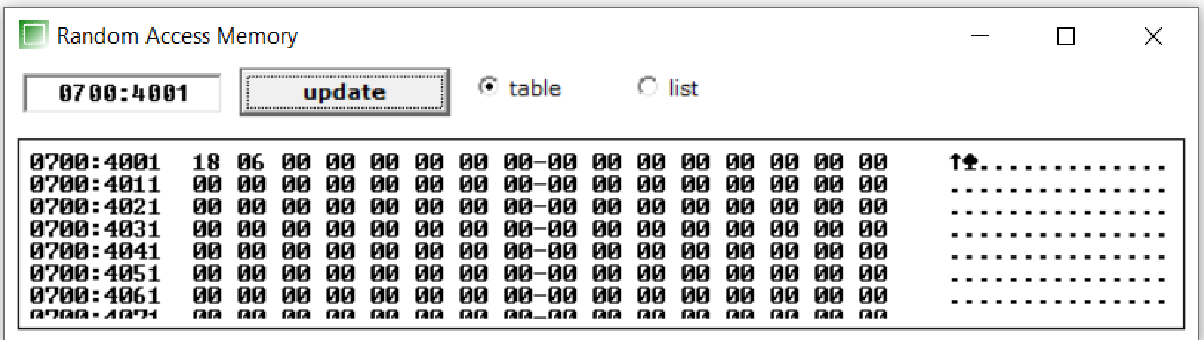
PRACTICAL - 15

AIM: Write a program to multiply & divide the number stored at 4000H by 2 and store the result at 4001H & 4002H. (Use Shift instructions).

CODE:

```
org 100h
MOV [4000H],1100B;
MOV AL,[4000H];
MOV BL,[4000H];
SHL AL,1;
SHR BL,1;
MOV [4001H],AL;
MOV [4002H],BL
ret
```

OUTPUT:



CONCLUSION: From this practical we learned about the shift instructions and how to use them to shift the bits.

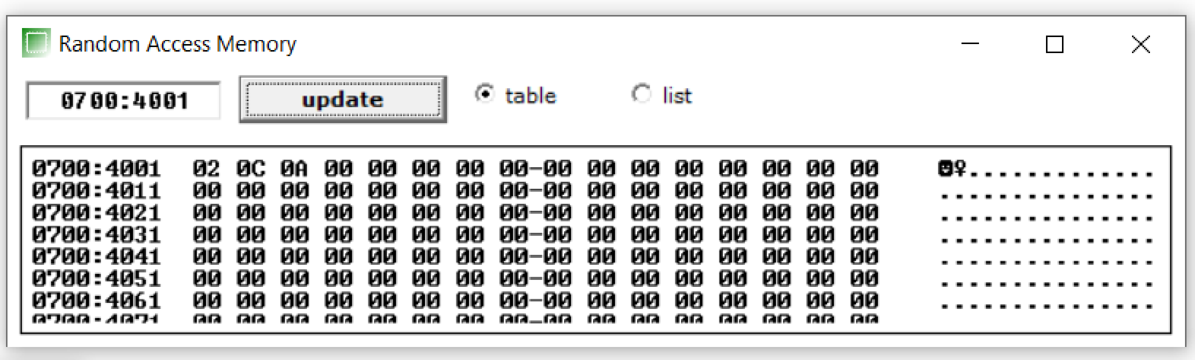
PRACTICAL - 16

AIM: Write a Program to subtract the contents of memory location 4001H from the memory location 4002H and place the result in memory location 4003H without SUB instruction.

CODE:

```
org 100h

MOV [4002H],1100B;

MOV [4001H],0010B;

MOV AL,[4002H];

MOV BL,[4001H];

NEG BL;

ADD AL,BL;

MOV [4003H],AL;

ret
```

OUTPUT:



CONCLUSION: From this practical we learned alternate method to subtract data from two memory locations or we can say that we learned about the "NEG" function in 8086 Emulator.

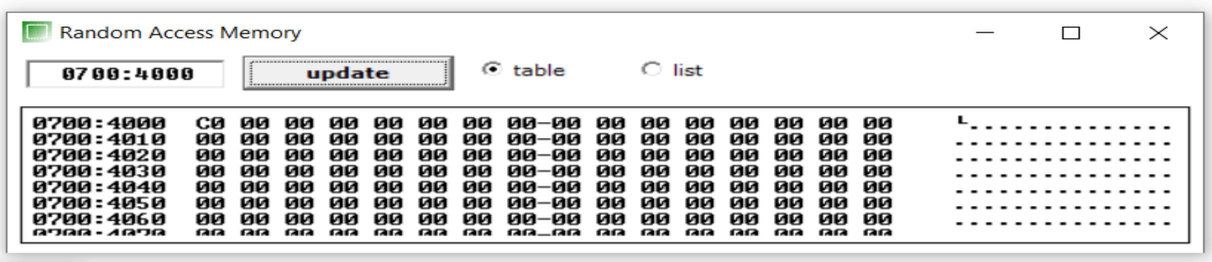
PRACTICAL - 17

AIM: Implement a program to mask the lower four bits of content of the memory location.

CODE:

```
org 100h

MOV    [4000H],11001010B;
MOV AL,[4000H];
AND AL,11110000B;
MOV [4000H],AL;

ret
```

OUTPUT:



CONCLUSION: From this practical we learned about how to mask the bits of any data from any memory location.

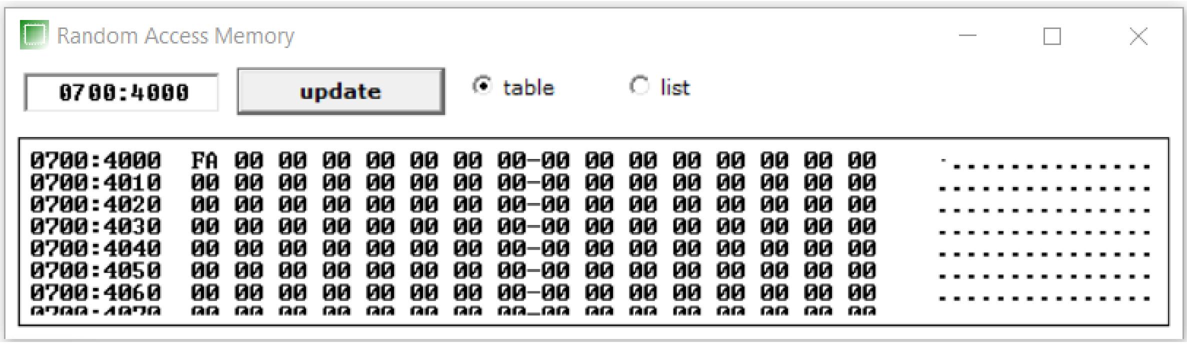
PRACTICAL - 18

AIM: Implement a program to set higher four bits of content of the memory location to 1.

CODE:

```
org 100h

MOV [4000H],11001010B;
MOV AL,[4000H];
OR AL,11110000B;
MOV [4000H],AL;

ret
```

OUTPUT:



CONCLUSION: From this practical we learned about the set operation that can be done using the OR operation.

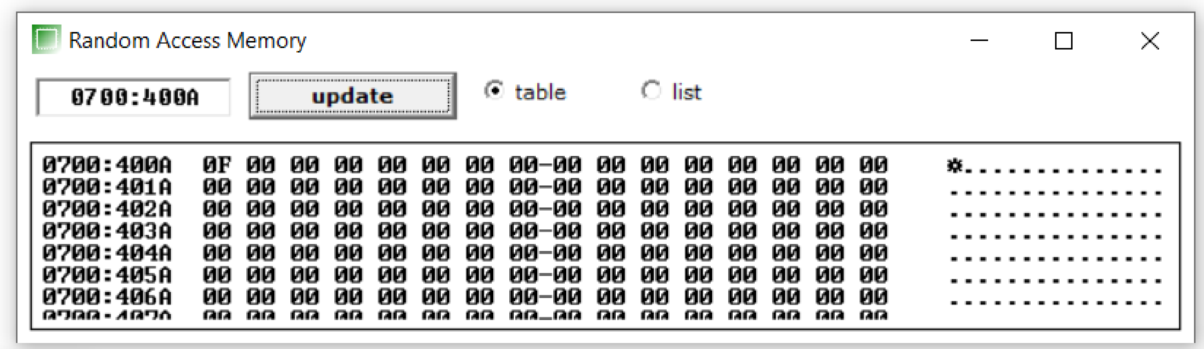
PRACTICAL - 19

AIM: Calculate the sum of series of numbers (Data set-1) from the memory location listed below & store the result at 400AH location.

CODE:

```
org 100h

MOV [4000H],05H
MOV [4001H],04H
MOV [4002H],03H
MOV [4003H],02H
MOV [4004H],01H
MOV SI,4000H
A:
MOV AL,[SI]
ADD [400AH],AL
INC SI
CMP SI,4005H
JNE A
ret
```

OUTPUT:



CONCLUSION: From this practical we learned about how to perform operations on continuous dataset without writing the same operation again using labels.

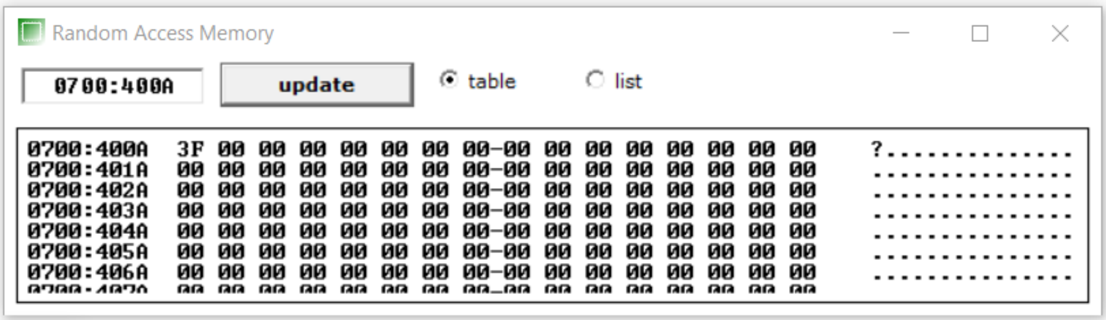
PRACTICAL - 20

AIM: Modify above the program such a way that it halts the execution if carry generated & stores the intermediate result at 400AH location. (Data set-2) (Note: Student need to implement FOR loop in this program: initialization, Compare, Decrement/Increment; also need to use JMP, JMX instructions.)

CODE:

```
org 100h

MOV [4000H],15H
MOV [4001H],14H
MOV [4002H],09H
MOV [4003H],08H
MOV [4004H],05H
MOV SI,4000H
MOV CX,5
A:
MOV AL,[SI]
ADD [400AH],AL
JC QUIT
INC SI
LOOP A
QUIT:
ret
```

OUTPUT:



CONCLUSION: From this practical we learned about the working and usage of "LOOP" and "JC" instructions.

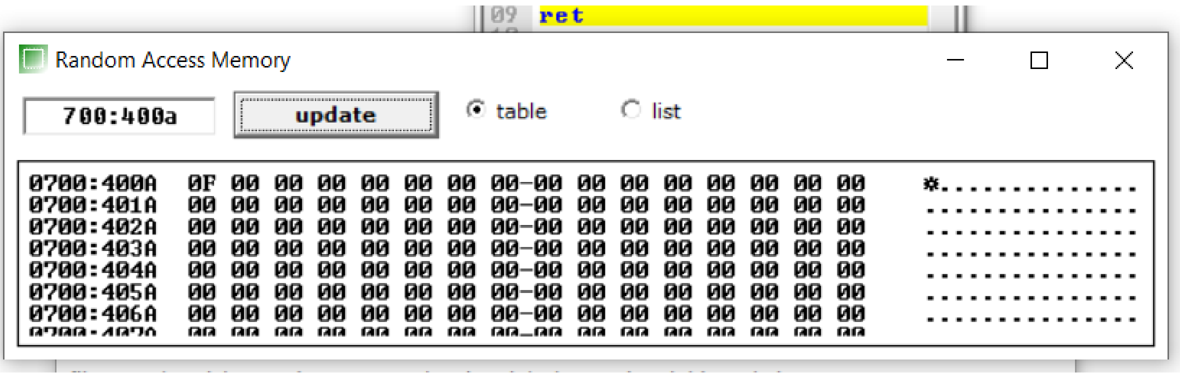
PRACTICAL - 21

AIM: Multiply two 8-bit numbers stored in memory locations 4001H and 4006H by repetitive addition and store the result at 400AH location.(Use Data Set -3) (Note: Student need to implement FOR loop in this program: initialization, Compare, Decrement/Increment; also need to use JMP, JMX instructions.)

CODE:

```
org 100h

MOV [4000h],05h

MOV [4001h],03h

MOV CL,[4001H]

MOV BL,[4000H]

abc:

ADD [400AH],BL

LOOP abc

ret
```

OUTPUT:



CONCLUSION: In this Practical we have learn JMP, JMX instruction and Multiplication Without MUL instruction.

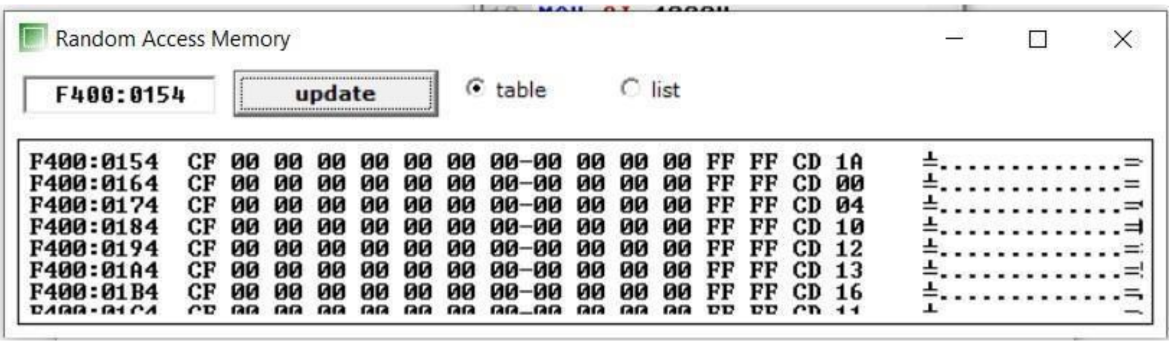
PRACTICAL - 22

AIM: Program to find average of n number.

CODE:

```
org 100h
MOV [4001H],01H
MOV [4002H],02H
MOV [4003H],03H
MOV [4004H],04H
MOV [4005H],05H
MOV [4006H],06H
MOV [4007H],07H
MOV [4008H],08H
MOV SI,4000H
MOV CL,0AH
MOV Dl,CL
I1:
MOV BL,[SI]
ADD AL,BL
INC  SI
LOOP  I1
DIV DL

ret
```

OUTPUT:



CONCLUSION: In this Practical we have learn how to find average of N number.

PRACTICAL - 23

AIM: Write an assembly language program to find the no. of odd numbers and even numbers, given an array of n numbers.

CODE:

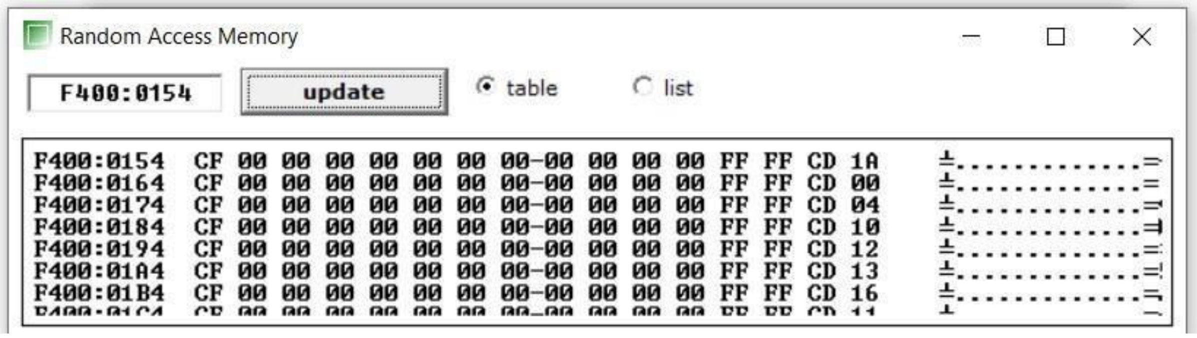
```
org 100h
mov [4000h],01h
mov [4001h],03h
mov [4002h],05h
mov [4003h],02h
mov [4004h],04h
mov [4005h],06h
mov [4006h],08h
mov [4007h],0ah
mov [4008h],07h
mov [4009h],02h
mov cl,0ah
mov si,4000h
mov dl,00h
mov bl,00h
i1:
mov al,[si]
shr al,1
inc si
jc odd
jnc even
odd:
inc dl
loop i1
ret
even:
```

inc bl

loop i1

ret

OUTPUT:



CONCLUSION: In this Practical we have learn find odd or even number out of N numbers of array. Here (1, 3, 5, 2, 4, 6, 8, 0A, 7, 2) Odd DL = 04.

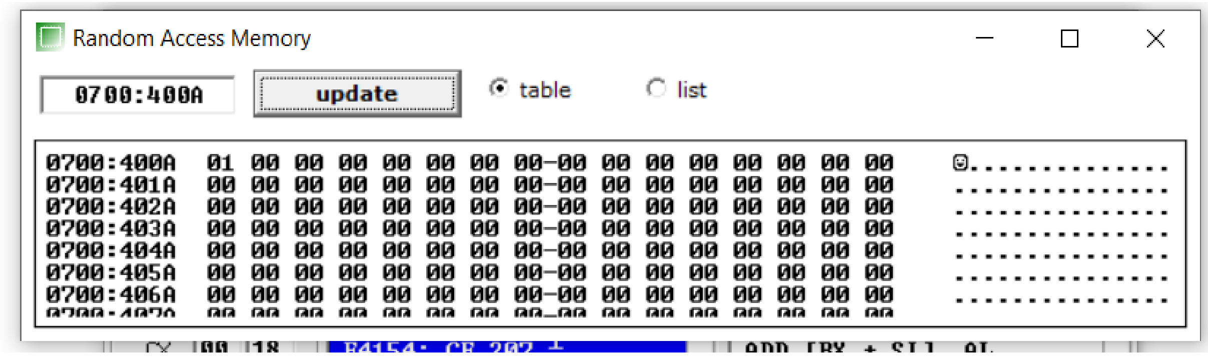
PRACTICAL - 24

AIM: Divide 8-bit number stored in memory locations 4009H by data stored at memory location 4001H & store result of division at memory location 400AH. (Use Data Set -4).

CODE:

```
org 100h

MOV [4001],56

MOV [4009],5

MOV AX,[4001]

MOV BL,[4009]

DIV BL

MOV [400AH],AH ;Division

ret
```

OUTPUT:



CONCLUSION: In this Practical we have learn perform division and store at specific memory location. Here 56 divide by 5.

PRACTICAL - 25

AIM: Divide 8-bit number stored in memory locations 4009H by data stored at memory location 4001H & store result of module operation at memory location 400AH. (Use Data Set - 2,4).

CODE:

```
org 100h

mov [4009h],0ah

mov [4001h],03h

mov ax,[4009h]

mov bl,[4001h]


div bl

mov [400ah],ah
ret
```

; NOTE: ALWAYS REMEMBER IN DIVISION IF AFTER DIVIION WE SEE AL GIVES ANSWER AFTER DIVISION LIKE 10/3=3 WHILE AH GIVES THE REMAINDER 10%3=1

OUTPUT:

CONCLUSION: From this practical we learned about how we can fetch the quotient and remainder while using the DIV instruction.

PRACTICAL - 26

AIM: Write an assembly language program to find the largest number in an array.

CODE:

```
org 100h
mov [4000h],10
mov [4001h],20
mov [4002h],45
mov [4003h],80
mov [4004h],65
mov [4005h],19
mov [4006h],44
mov [4007h],56
mov [4008h],29
mov [4009h],31

mov cl,10
mov si,4000h
mov al,[si]
l1:
cmp [si],al
jnc  high
jc repeat

repeat:
cmp si,4009h
jz exit
inc si
jmp l1
```

high:

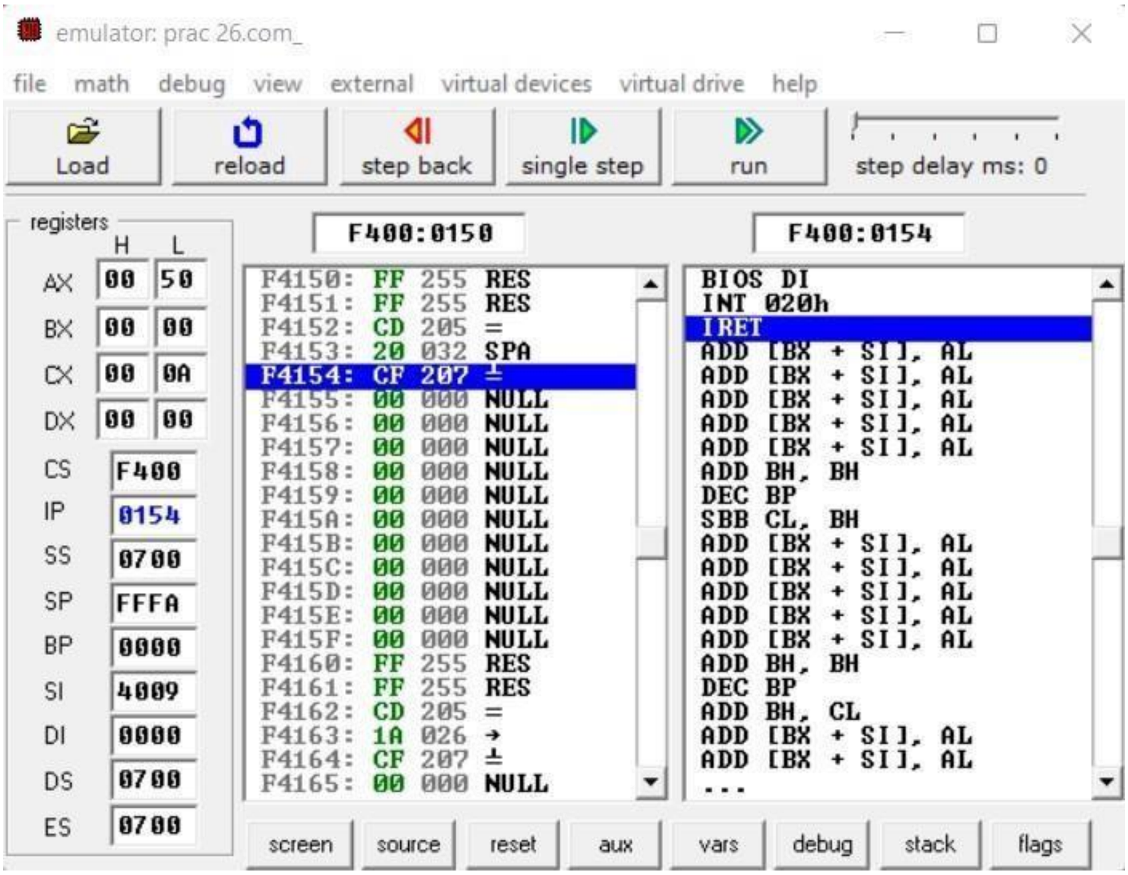
mov al,[si]

jmp repeat

exit:

ret

OUTPUT:



Note: The Highest number is seen in AL register.

CONCLUSION: From this practical we learned about the use of different types of jump instructions as well as how to find the highest number from given data.

PRACTICAL - 27

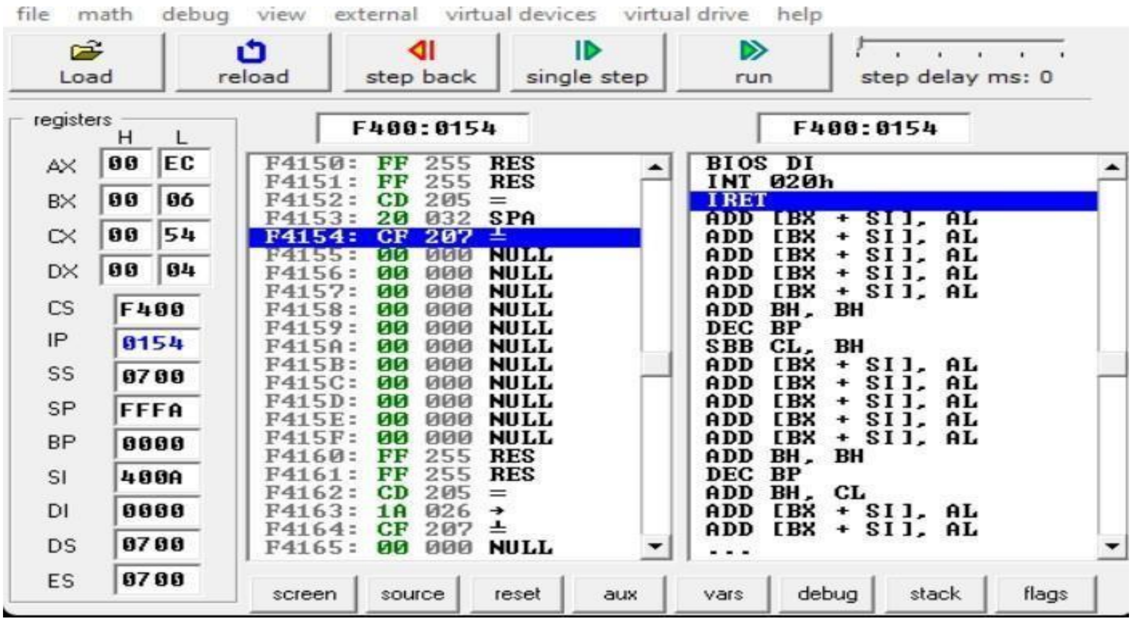
AIM: Write an assembly language program to count the numbers in an array (negative & positive)

CODE:

```
org 100h
mov  [4000h],10
mov [4001h],-20
mov  [4002h],45
mov  [4003h],80
mov [4004h],-65
mov  [4005h],19
mov  [4006h],44
mov [4007h],-56
mov [4008h],-29
mov [4009h],31
mov si,4001h
mov al,[si]
mov bl,00h
mov dl,00h

l1:
cmp [si],0h
js  nega
jmp pos
repeat:
cmp si,400ah
jz exit
```

```
inc si
jmp l1

pos:
inc bl
jmp repeat

nega:
inc dl
jmp repeat

exit:
ret
```

OUTPUT:



CONCLUSION: From this practical we learned about the use of new jump instruction i.e. JS which is jump when sign flag is 1 so by using that and comparing with zero we are able to distinguish between positive and negative numbers.

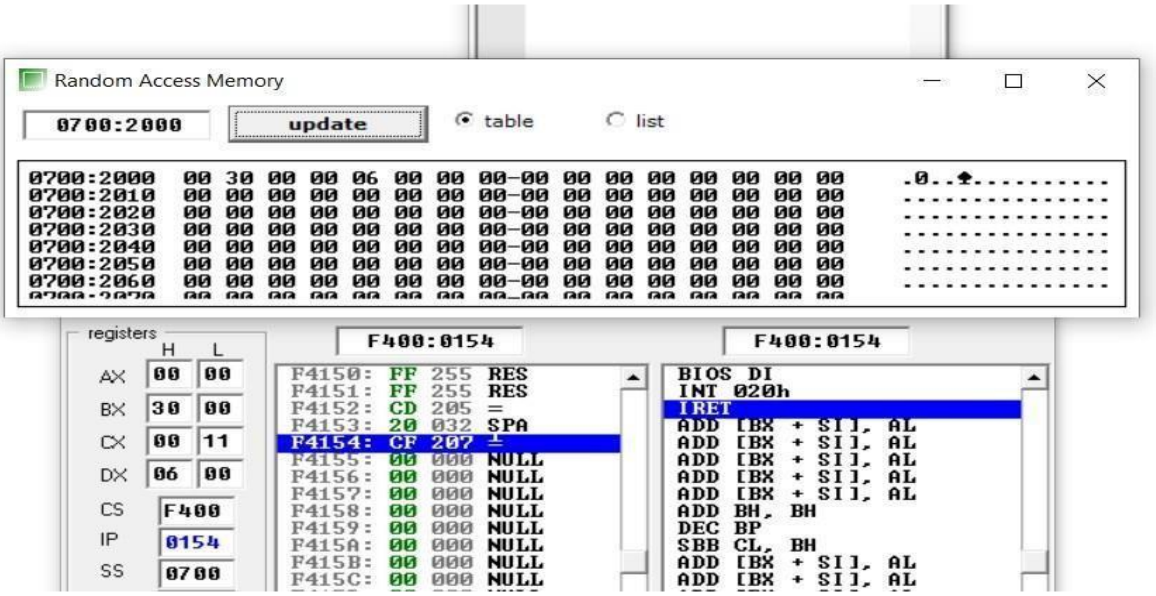
PRACTICAL - 28

AIM: Write an assembly language program to multiply two 16-bit numbers in memory and store the result in memory.

CODE:

```
org 100h
MOV  AX,2000H
MOV  BX,3000H
MUL BX
MOV  [2000H],BX
MOV [2003H],DX
ret
```

OUTPUT:



CONCLUSION: From this practical, we learned about how to multiply 2 16-bit numbers and where we can get its output.

PRACTICAL - 29

AIM: Write a program with nested loop which will display the decimal down counter. on Port 1 with a one second delay between each count.

CODE:

```
org 100h
MOV AX,9H
L2:
OUT 110,AX
MOV CX,9CH
L1:
LOOP L1
DEC  AX
JNZ L2
ret
```

OUTPUT:

CONCLUSION: From this practical, we learned about how we can generate a delay between some processes.

PRACTICAL - 30

AIM: Write an assembly language program to Display Digits 0 1 2 3 4 5 6 7 8 9 A B C D E F on port 01H with 500ms of delay

CODE:

```
org 100h
mov dx,'0'
mov bx,3Ah

rep:
mov ah, 6
int  21h
mov cx,70h
d:
dec cx
jnz d

inc  dl
cmp dx,bx
jnz rep

mov dl,'A'
mov bl,47h
rep2:
mov ah, 6
int  21h
mov cx,70h
d2:
```

```
dec cx
jnz d2


inc dl
cmp dl,bl
jnz rep2
jz exit
exit:
ret
```

OUTPUT:

CONCLUSION: From this practical, we learned about printing the numbers to the output screen using the interrupts with the use of given delays.

PRACTICAL - 31

AIM: Write an assembly language program to take input from memory address starting from 2000 h to 2009 h.

Display ‘1’ in LED port 199 if data <= 50h

Display ‘10’ in LED port 199 if data > 50h and data <= A0h

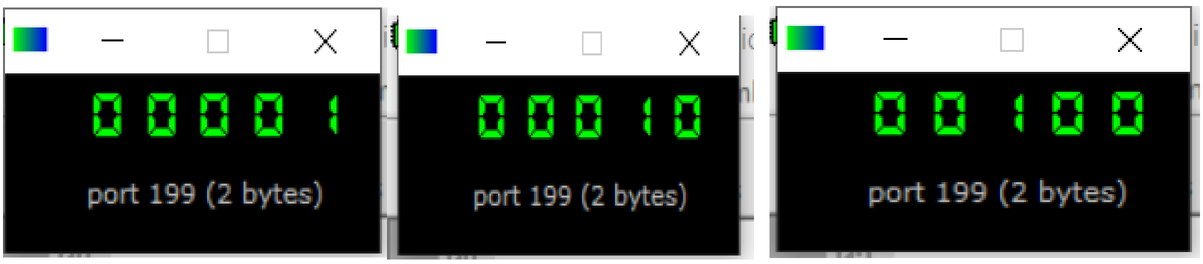
Display ‘100’ in LED port 199 if data > A0h

CODE:

```
org 100h
MOV [2000H],09H
MOV [2001H],78H
MOV [2002H],79H
MOV [2003H],59H
MOV [2004H],95H
MOV [2005H],25H
MOV [2006H],08H
MOV [2007H],21H
MOV [2008H],74H
MOV [2009H],39H
MOV SI,2000H

REP:
MOV BL,[SI]
CMP BL,50H
JBE LAB1
CMP BL,0A0H
JBE LAB10
CMP BL,0A0H
JG LAB100

LAB1:
MOV AL,1
```

```
OUT 199,AL
INC SI
CMP SI,200AH
JNZ REP
JMP EXIT

LAB10:
MOV  AL,10
OUT  199,AL
INC SI
CMP  SI,200AH
JNZ REP
JMP EXIT

LAB100:
MOV AX,100
OUT 199,AL
INC SI
CMP SI,200AH
JNZ REP
JMP EXIT

EXIT:
ret
```

OUTPUT:



CONCLUSION: From this practical, we learned about port and how to use LED .

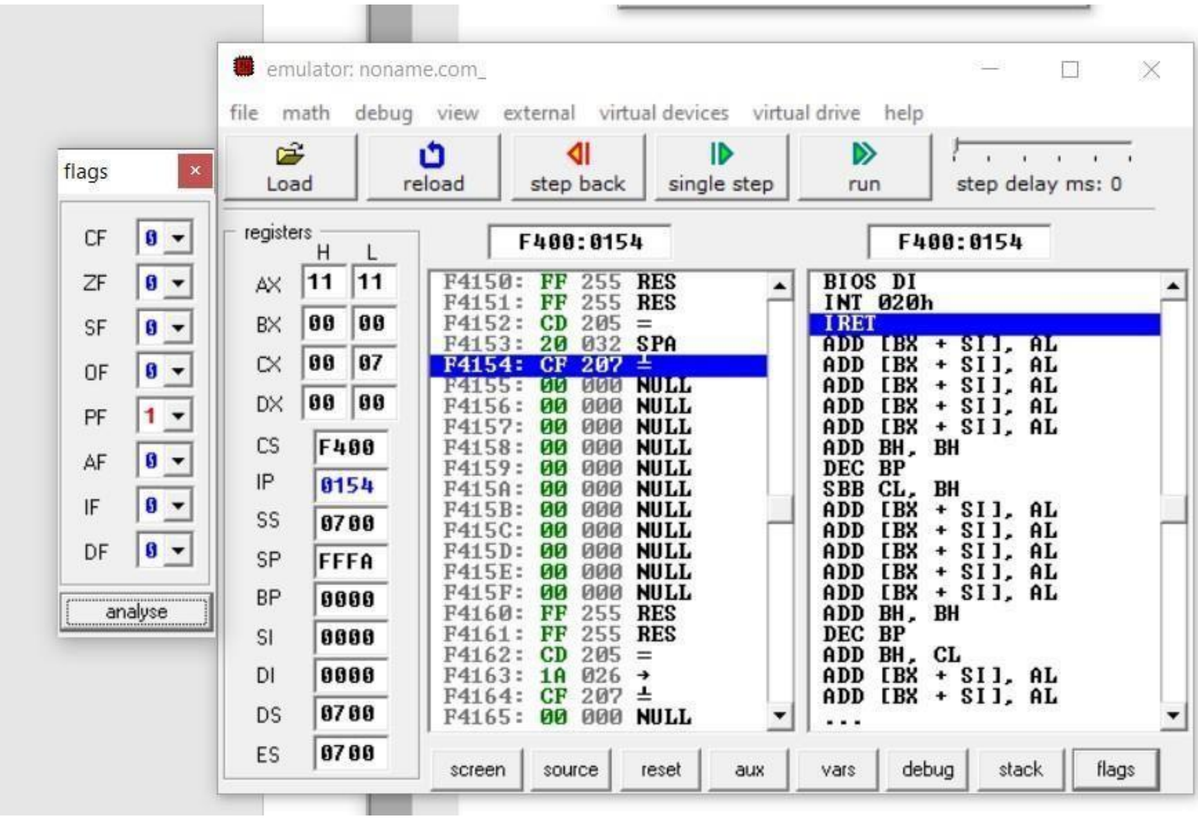
PRACTICAL - 32

AIM: Write a program which sets the parity bit.

CODE:

```
org 100h

MOV AX,1110H
ADD AX,01

ret
```

OUTPUT:



CONCLUSION: From this practical, we learned how to implement parity fag and what is the main use of it.

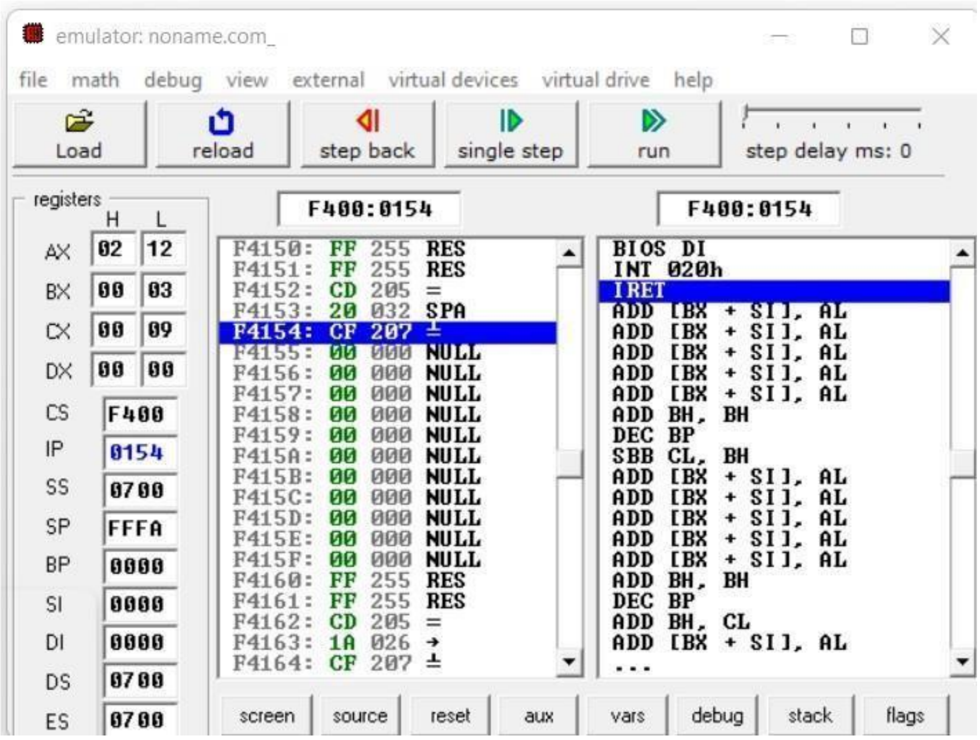
PRACTICAL - 33

AIM: Write a program which transfers content of Flags toRegister

CODE:

```
org 100h
mov al,1101b
mov bl,0011b
add al,bl
pushf
pop ax
ret
```

OUTPUT:



CONCLUSION: From this practical, we learned how to transfer content of flag to register.

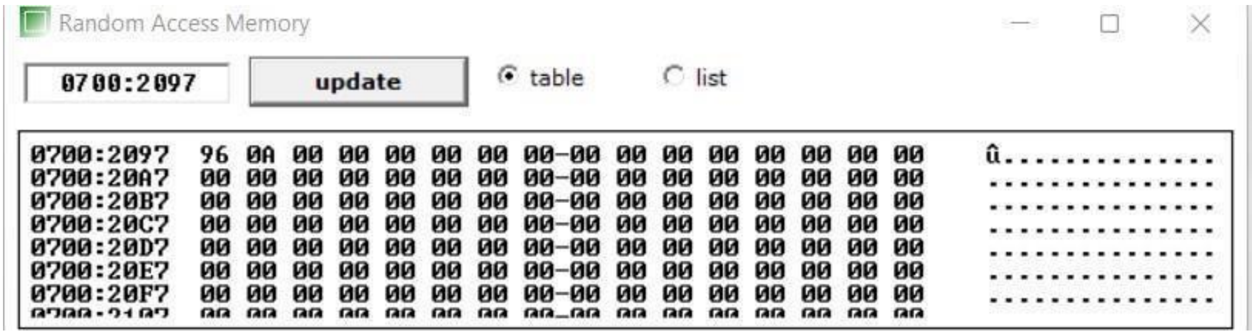
PRACTICAL - 34

AIM: Write a program to add the two Hex Numbers 7AH and 46H and to store the sum at memory location 2098 and flags status at 2097 location.

CODE :

```
org 100h

mov  al,7ah
mov  bl,46h
add al,bl
mov [2098h],al
pushf
pop cx
mov [2097h],cx
ret
```

OUTPUT:



CONCLUSION: From this practical we learn about how to crossover flag and operation and to transfer flag into register.

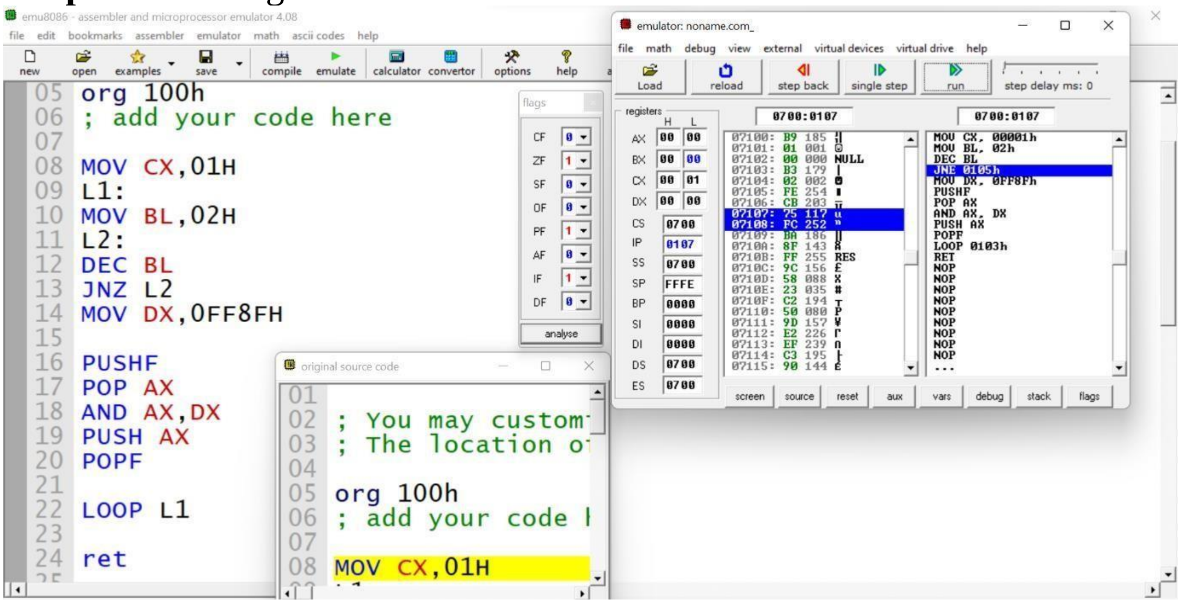
PRACTICAL – 35

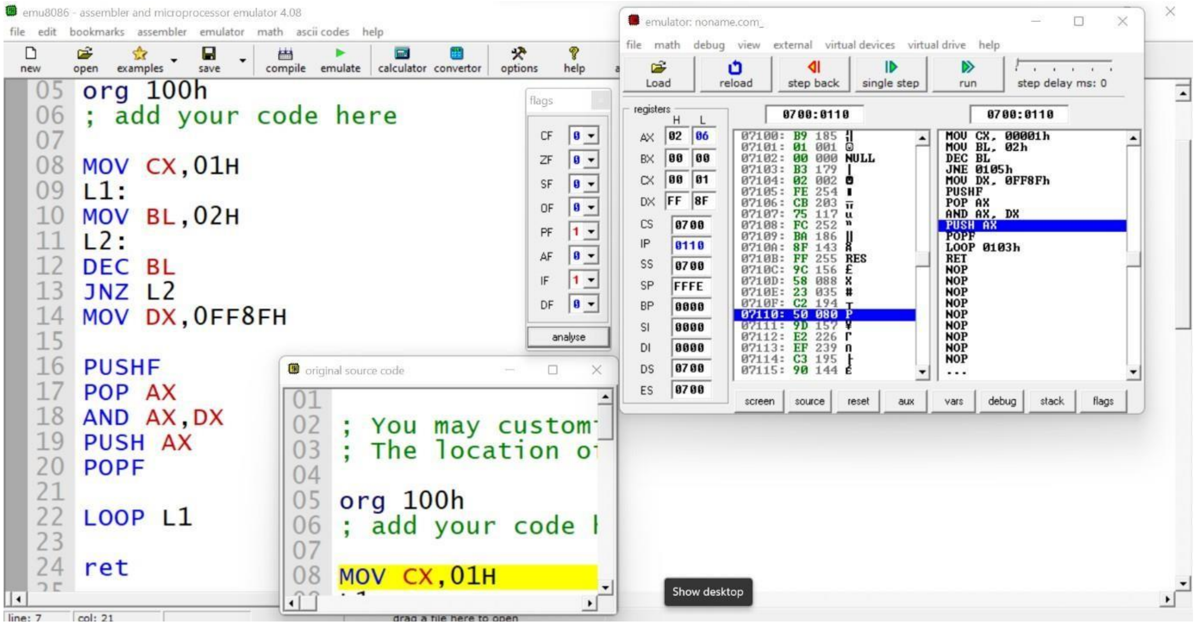
Aim: Write a 20 MS time delay subroutine using register pair BC. At the end of subroutine, clear the flag Z without affecting other flags and return to main program.

Program:

```
MOV CX,01H
L1:
MOV BL,02H
L2:
DEC BL
JNZ L2
MOV DX,0FF8FH
PUSHF
POP AX
AND AX,DX
PUSH AX
POPF
LOOP L1
Ret
```

Output: Zero Flag Set To 1:





Conclusion: From this practical we learn about how we reset/clear flag without affect other flag include time delay using loop.

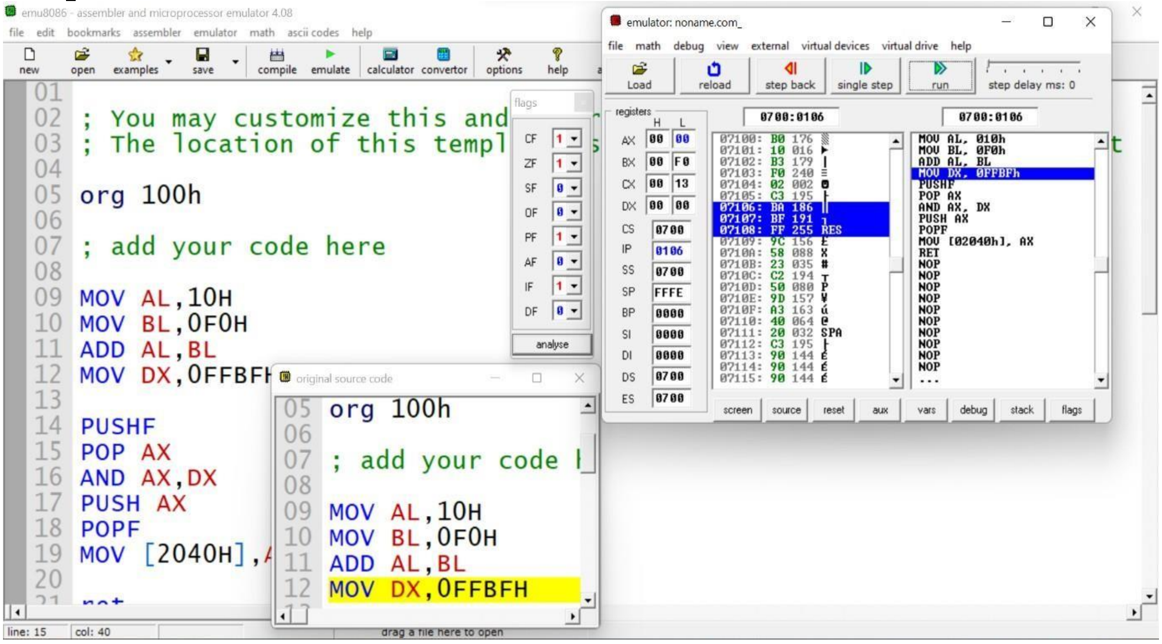
PRACTICAL – 36

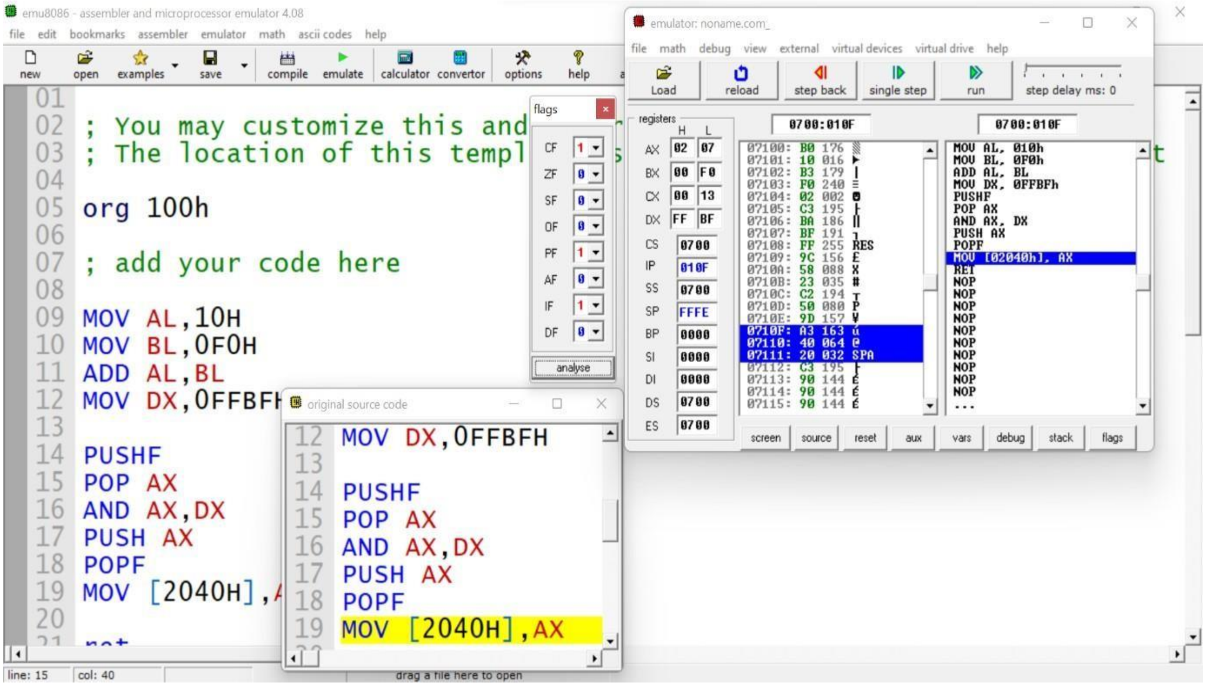
Aim: Using a Subroutine, write a program which adds two hex number 10H and F0H and store result at 2040H location in memory. At the end of subroutine, clear the flag Z without affecting other flags and return to main program.

Program:

```
MOV AL,10H
MOV BL,0F0H
ADD  AL,BL
MOV DX,0FFBFH
PUSHF
POP AX
AND AX,DX
PUSH AX
POPF
MOV [2040H],AX
Ret
```

Output:





Conclusion: In this practical we have written a program which adds two hex number & stored a result in a memory location. At the end of subroutine, clear the flag Z without affecting other flags and return to main program.

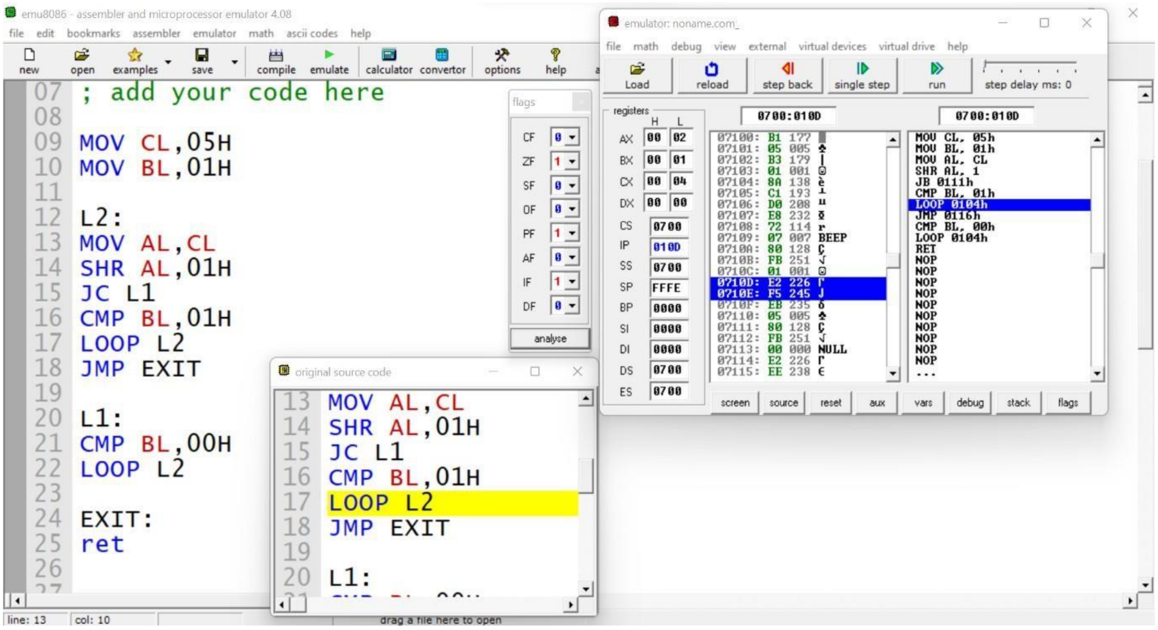
PRACTICAL – 37

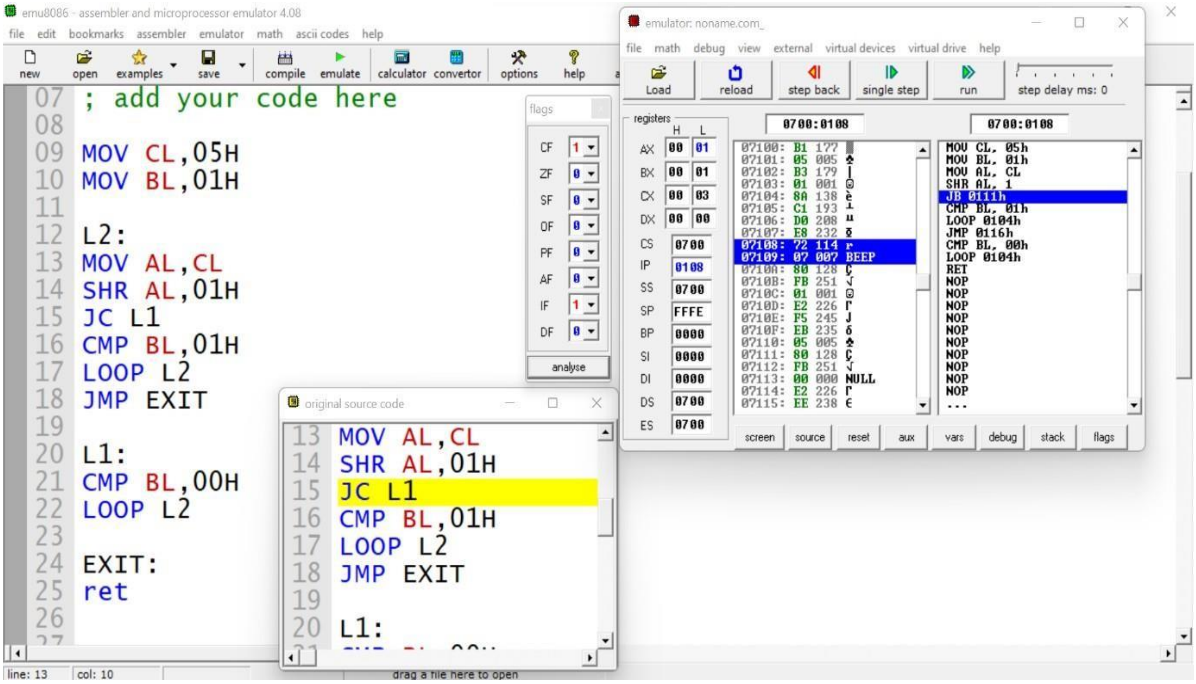
Aim: Write a program which set and resets zero flag at next iteration. (Take number of iteration equal to 5)

Program:

```
MOV CL,05H
MOV BL,01H

L2:
MOV  AL,CL
SHR  AL,01H
JC L1
CMP BL,01H
LOOP L2
JMP EXIT

L1:
CMP BL,00H
LOOP L2
EXIT:
ret
```

Output:





Conclusion: In this practical we have written a program which set and resets zero flag at next iteration.

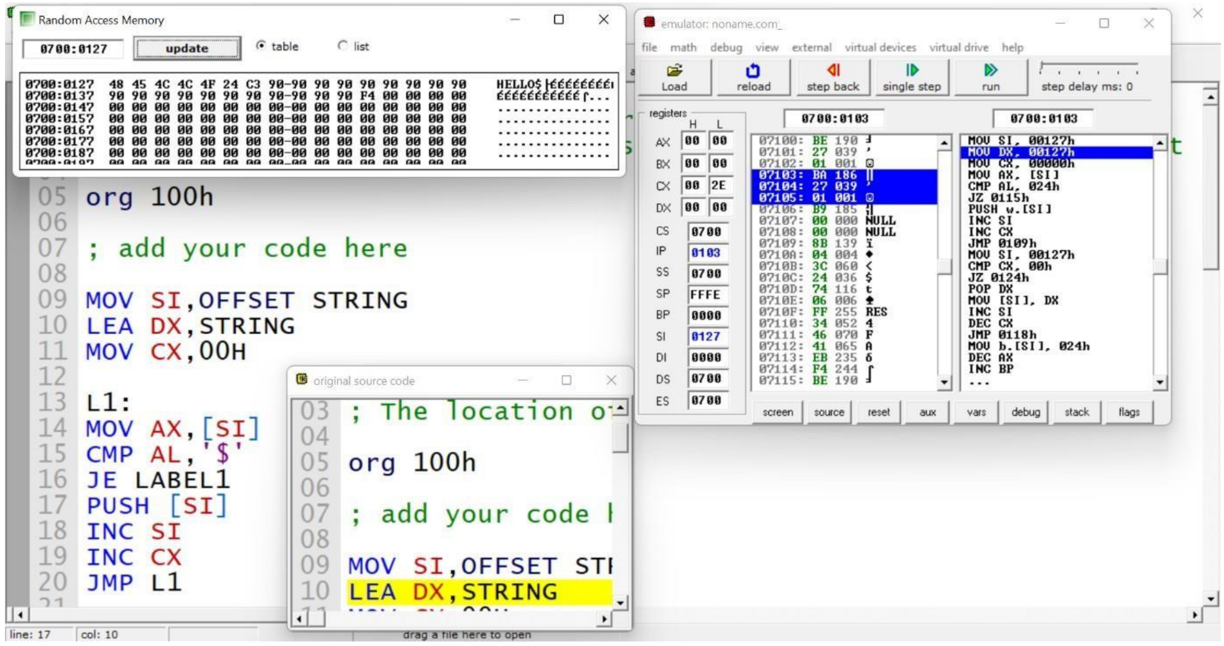
PRACTICAL – 38

Aim: Implement a program to reverse a string using stack operations and stored in same memory area.

Program:

```
MOV SI,OFFSET STRING
LEA DX,STRING
MOV CX,00H

L1:
MOV AX,[SI]
CMP AL,'$'
JE LABEL1
PUSH [SI]
INC SI
INC CX
JMP L1

LABEL1:
MOV SI,OFFSET STRING

LOOP2:
CMP CX,00H
JE EXIT
POP DX
MOV [SI],DX
INC SI
DEC CX
JMP LOOP2

EXIT:
MOV [SI],'$'

STRING DB 'HELLO','$'
ret
```

Output:



Conclusion: In this practical we have reversed a string using stack operations and stored in same memory area.

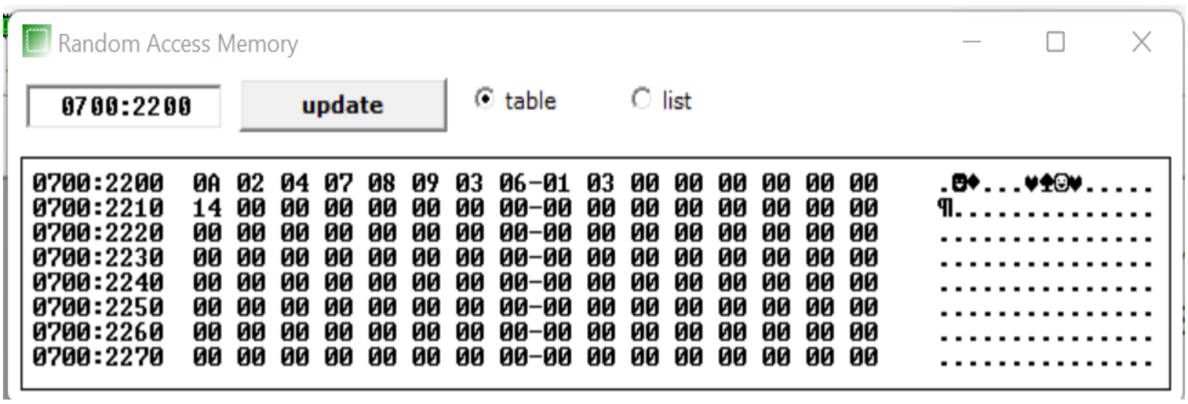
PRACTICAL – 39

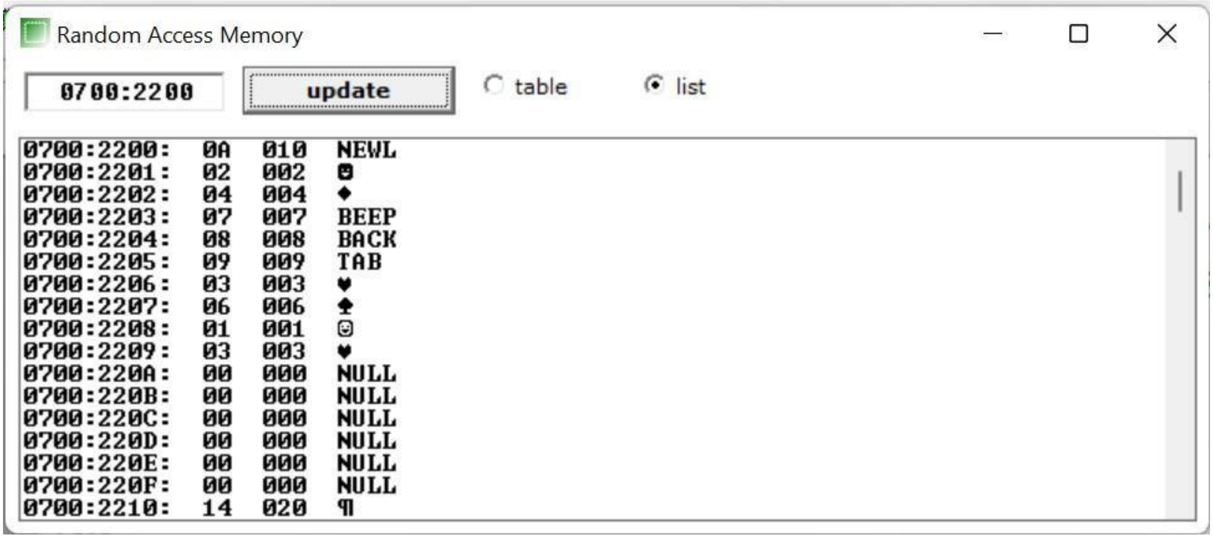
Aim: Calculate the sum of series of even numbers from the list of numbers. The length of the list is in memory location 2200H and the series itself begins from memory location 2201H. Assume the sum to be 8 bit number so you can ignore carries and store the sum at memory location 2210H.

Program:

```
org 100h
MOV [2200H],0AH
MOV [2201H],02H
MOV [2202H],04H
MOV [2203H],07H
MOV [2204H],08H
MOV [2205H],09H
MOV [2206H],03H
MOV [2207H],06H
MOV [2208H],01H
MOV [2209H],03H
MOV CL,[2200H]
MOV SI,2201H
MOV BL,00H

L1:
MOV  AL,[SI]
SHR  AL,01H
JC L2
ADD BL,[SI]
L2:
INC SI
LOOP L1
MOV [2210H],BL
ret
```

Output:





Conclusion: In this practical we have Calculate the sum of series of even numbers from the list of numbers.

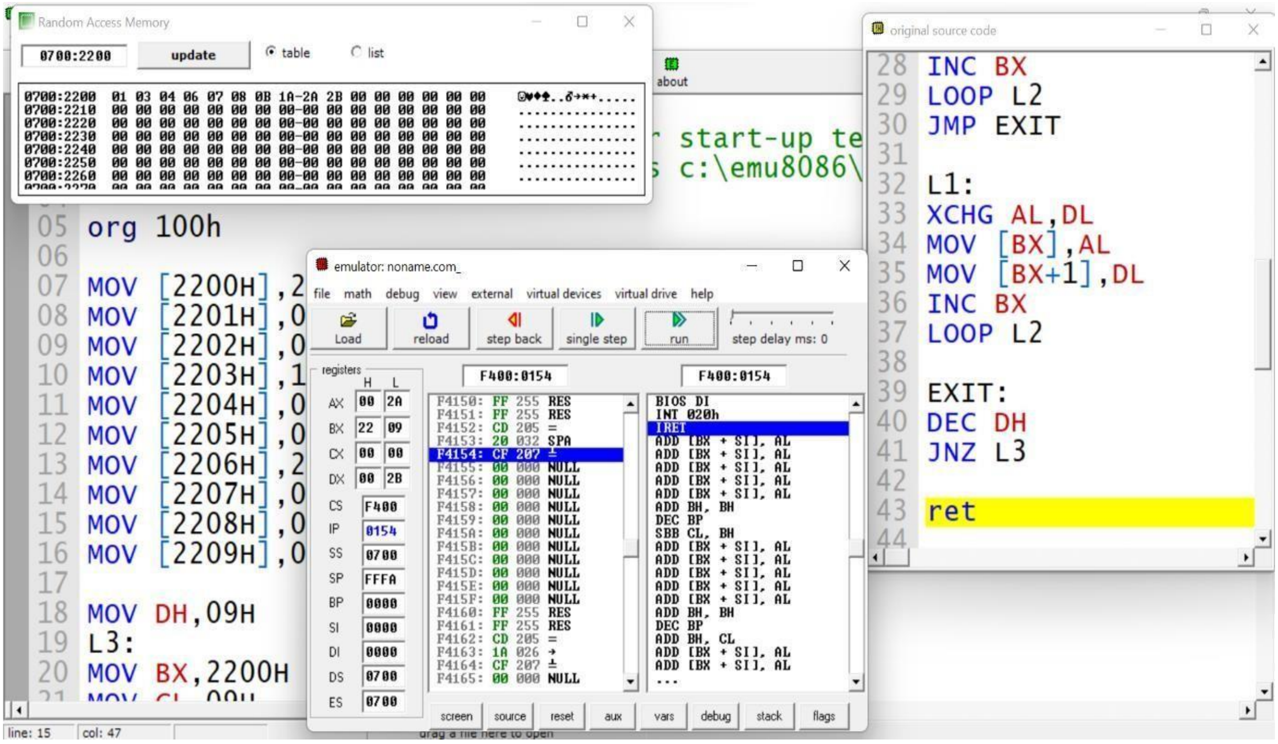
PRACTICAL – 40

Aim: Write an assembly language program to arrange an array of 10 data in ascending order. The length of the list is in memory location 2200H and the series itself begins from memory location 2201H

Program:

```
org 100h
MOV [2200H],2AH
MOV  [2201H],08H
MOV  [2202H],03H
MOV [2203H],1AH
MOV  [2204H],07H
MOV  [2205H],04H
MOV [2206H],2BH
MOV  [2207H],06H
MOV [2208H],0BH
MOV  [2209H],01H
MOV DH,09H
L3:
MOV BX,2200H
MOV CL,09H
L2:
MOV AL,[BX]
MOV DL,[BX+1]
CMP  AL,DL
JNC L1
INC BX
LOOP L2
JMP EXIT
L1:
XCHG AL,DL
MOV [BX],AL
MOV [BX+1],DL
INC BX
LOOP L2
EXIT:
DEC DH
JNZ L3
Ret
```

Output:



Conclusion: In this practical we have written an assembly language program to arrange an array of 10 data in ascending order. The length of the list is in memory location and the series itself begins from next memory location. In this practical we have used the concept of bubble sort.

PRACTICAL – 41

Aim: Write an assembly language program to fill the memory locations starting from 3000h, with n Fibonacci numbers.

Program:
```
mov ax,01h
mov dx,00h
mov cx,0ah
mov [3000h],dx
mov [3001h],ax
mov bx,3002h
l1:
add ax,dx
mov [bx],ax
mov dx,[bx-1]
inc bx
loop l1
ret
```

Output:

Conclusion: From this practical we learn how to get Fibonacci series using assembly language