Microprocessors and Microcontrollers
Lab File

ELE-306L

Submitted by-

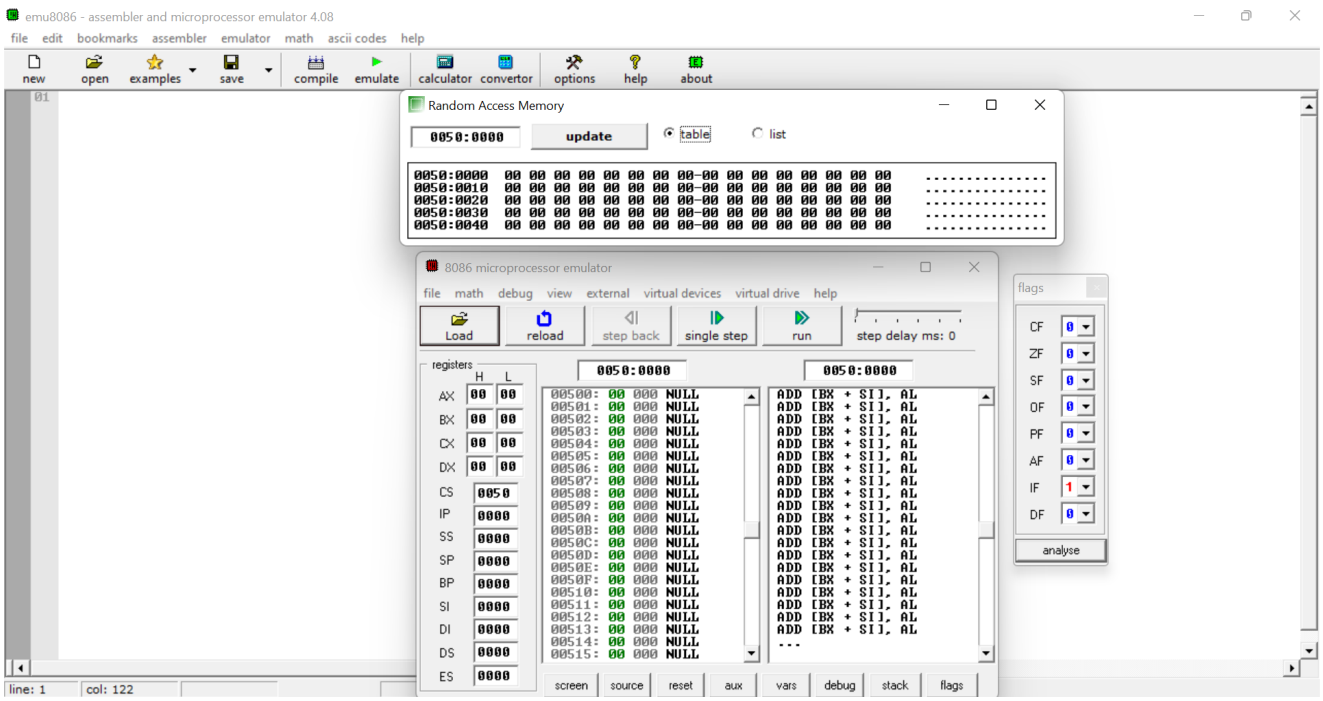
Name- Shreya Sheel Sachan
Section - C
Batch- C1
Roll No - 1913114
Id- BTBTC19088
Branch-Computer Science
Semester - VIth

Table of Contents

- ## Introduction of Emulator 8086

8086 Microprocessor Emulator, also known as EMU8086, is an emulator of the program 8086 microprocessor. It is developed with a built-in 8086 assembler. This application is able to run programs on both PC desktops and laptops. This tool is primarily designed to copy or emulate hardware. These include the memory of a program, CPU, RAM, input and output devices, and even the display screen.

The user interface of 8086 Microprocessor Emulator is simple and easy to manage. There are five major buttons with icons and titles included. These are “Load”, “Reload”, “Step Back”, “Single Step”, and “Run”. Above those buttons is the menu that includes “File”, “View”, “Virtual Devices”, “Virtual Drive”, and “Help”. Below the buttons is a series of choices that are usually in numbers and codes. At the leftmost part is an area called “Registers” with an indication of either “H” or “L”. The other side is divided into two, which enables users to manually reset, debug, flag, etc.



Running the Emulator

- Download and install emu8086(www.emu8086.com) It is usually installed in C:\EMU8086 subfolder in the “Windows” directory
- Run d emu8086 icon (on the desktop or in the c:\EMU8086 folder of window) It has green color
- If it requests for “Registration key, just ignore it by closing that window You will be left with the emulator (emu8086) IDE
- Copy and paste or type an Assembly Language program on editor of the emulator .
- Compile 6. Run (once there is no syntax error)
- Click OK to see/view the output of your program on the Emulator screen.
- After running the program, another menu screen will be displayed, where you have the option to “View” symbol table, variables, listing (containing the object code and source code), emulator screen, etc

● Instruction set: Programming and Illustration

❖ Data Transfer Instructions

These instructions are used to transfer the data from the source operand to the destination operand. Following are the examples of some instructions under this group -

1. MOV − Used to copy the byte or word from the provided source to the provided destination.
2. XCHG − Used to exchange the data from two locations.
3. XLAT − Used to translate a byte in AL using a table in the memory.
4. LEA − Used to load the address of operand into the provided register.

- **Arithmetic Instructions**

  These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc.

  Following are the examples of some instructions under this group -

  1. ADD − Used to add the provided byte to byte/word to word.
  2. ADC − Used to add with carry.
  3. INC − Used to increment the provided byte/word by 1.
  4. SUB − Used to subtract the byte from byte/word from word.
  5. SBB − Used to perform subtraction with borrow.
  6. MUL − Used to multiply unsigned byte by byte/word by word.
  7. DIV − Used to divide the unsigned word by byte or unsigned double word by word.

- **Bit Manipulation Instructions**

  These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc. Following are the examples of some instructions under this group -

  1. NOT − Used to invert each bit of a byte or word.
  2. AND − Used for adding each bit in a byte/word with the corresponding bit in another byte/word.
  3. OR − Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.
  4. XOR − Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.
  5. TEST − Used to add operands to update flags, without affecting operands

- **Program Execution Transfer Instructions (Branch and Loop Instructions)**

  These instructions are used to transfer/branch the instructions during an execution. Following are the examples of some instructions under this group -

1. RET − Used to return from the procedure to the main program.
2. JMP − Used to jump to the provided address to proceed to the next instruction.
3. JC − Used to jump if carry flag CF = 1
4. JNC − Used to jump if no carry flag (CF = 0)
5. JNE/JNZ − Used to jump if not equal/zero flag ZF = 0
6. LOOP − Used to loop a group of instructions until the condition satisfies, i.e., CX = 0

NOTE :There are some other instruction sets too in 8086 microprocessor but the ones discussed above are the common ones.

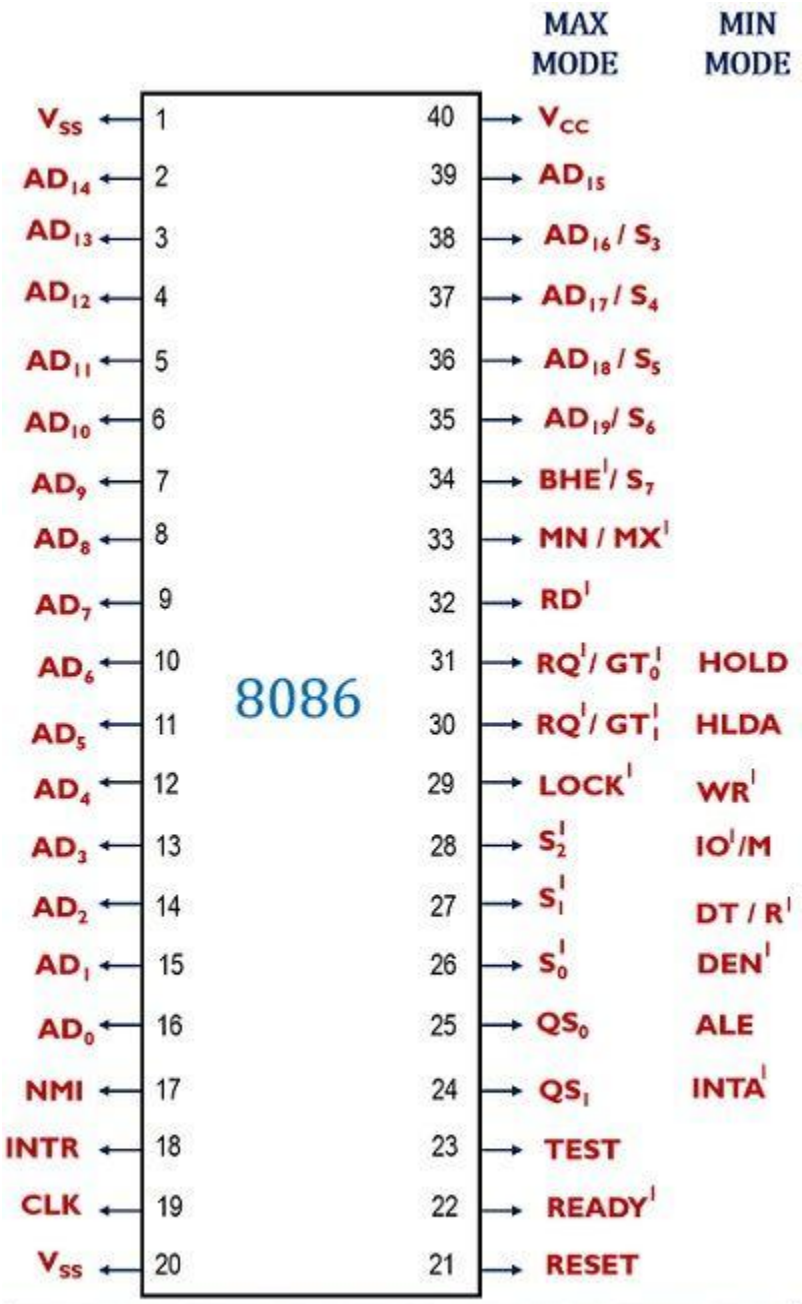
- ## General purpose registers in 8086 microprocessor

  The general purpose registers are used to store temporary data in the time of different operations in microprocessor. 8086 has eight general purpose registers.

Register	Function
AX	This is the accumulator. It is 16-bit registers, but it is divided into two 8-bit registers. These registers are AH and AL. AX is generally used for arithmetic or logical instructions, but it is not mandatory in 8086.
BX	BX is another register pair consisting of BH and BL. This register is used to store the offset values.
CX	CX is generally used as a control register. It has two parts CH and CL. For different looping and counting purposes these are used.

DX	DX is a data register. The two parts are DH and DL. This register can be used in Multiplication, Input/output addressing etc.
SP	This is the stack pointer. The stack pointer points to the top most element of the stack. For empty stack SP will be at position FFFEH.
BP	BP is another 16-bit register. This is a base pointer register. This register is primarily used in accessing the parameters passed by the stack. Its offset address is relative to the stack segment.
SI	This is the Source Index register. This is used to point the source in some string related operations. Its offset is relative to the data segment.
DI	This is the destination index register. This is used to point destinations in some string related operations. Its offset is relative to the extra segment.

- **Pin diagram of 8086 microprocessor**



Pin diagram of 8086 Microprocessor

Electronics Desk

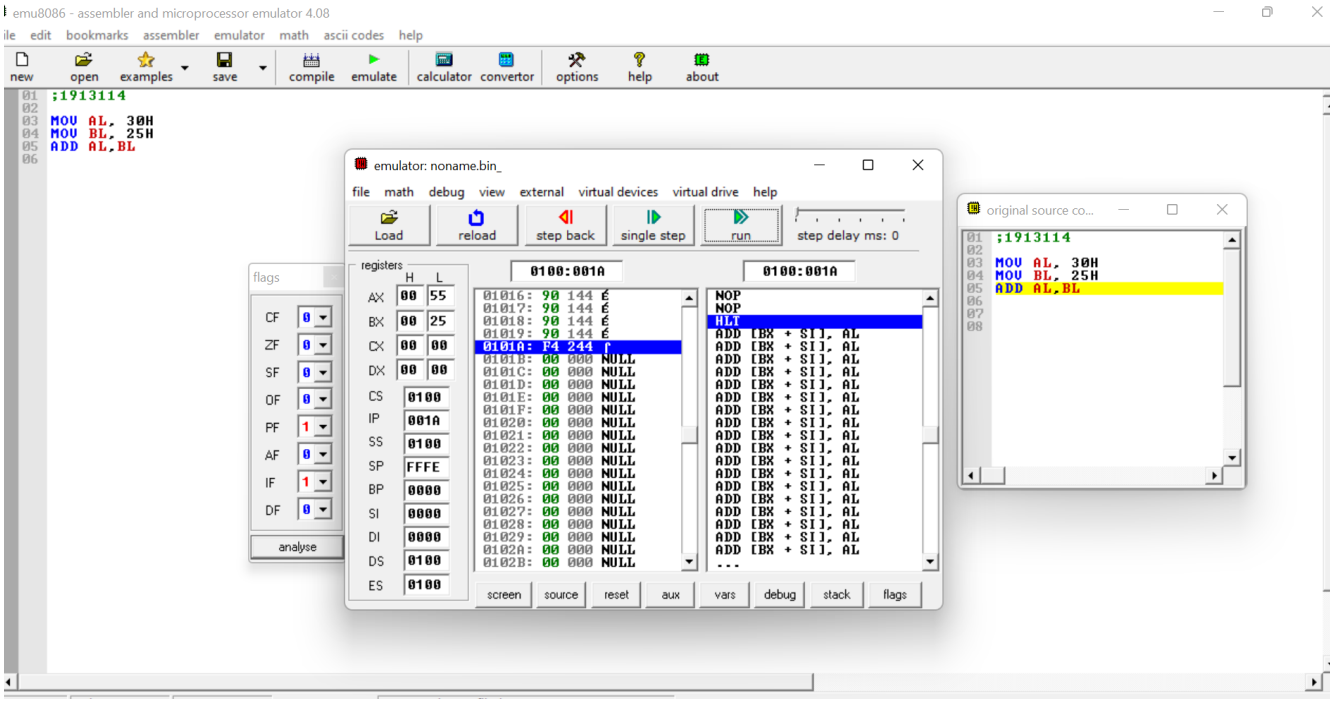
- **Program for addition of 8-bit numbers**

AIM- Addition of two 8-bit numbers.

CODE-

```
MOV AL, 30H
MOV BL, 25H
ADD AL,BL
```

OUTPUT-



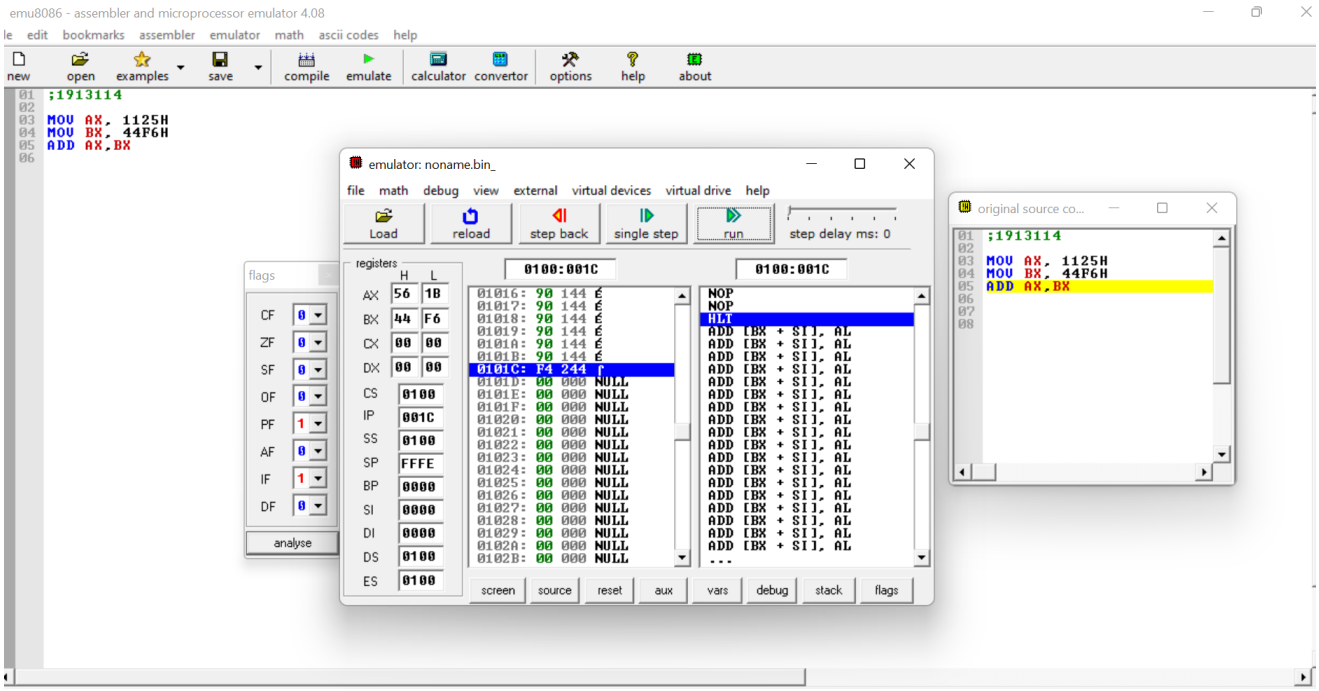
- **Program for addition of 16-bit numbers**

AIM- Addition of two 16-bit numbers.

CODE-

```
MOV AX, 1125H
MOV BX, 44F6H
ADD AX,BX
```

OUTPUT-



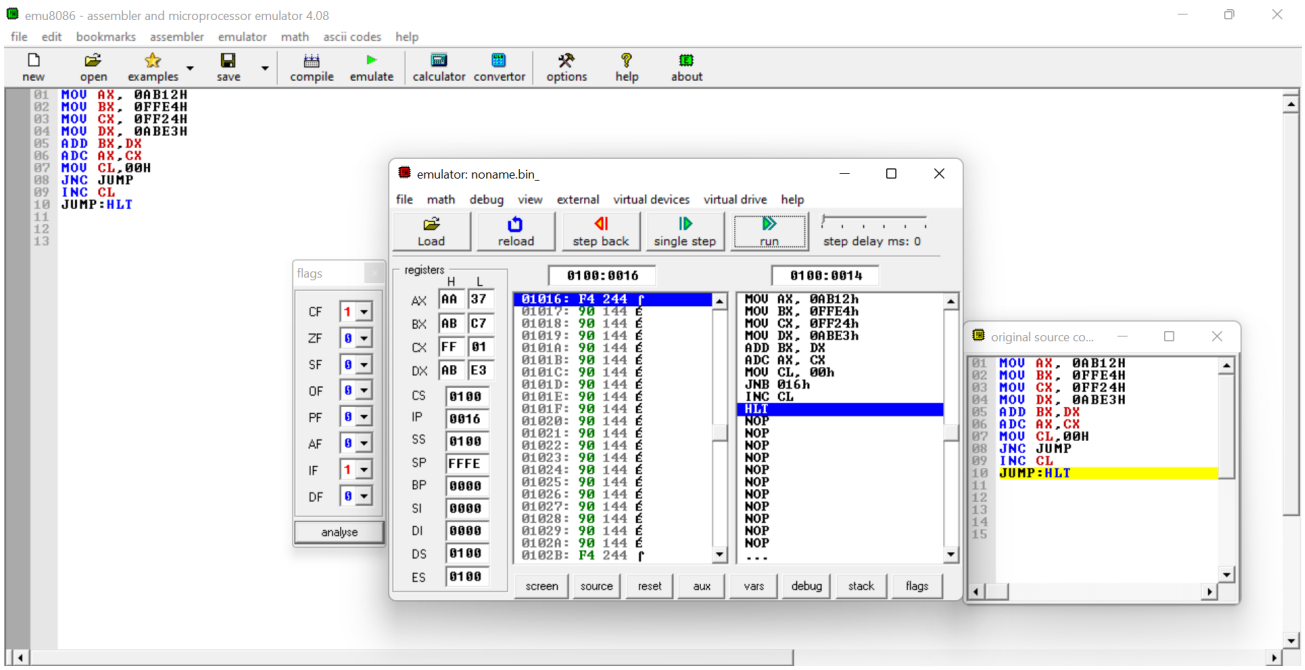
- **Program for addition of 32-bit numbers**

AIM- Addition of two 32-bit numbers.

CODE-

```
MOV AX, 0AB12H
MOV BX, 0FFE4H
MOV CX, 0FF24H
MOV DX, 0ABE3H
ADD BX,DX
ADC AX,CX
MOV CL,00H
JNC JUMP
INC CL
JUMP:HLT
```

OUTPUT-



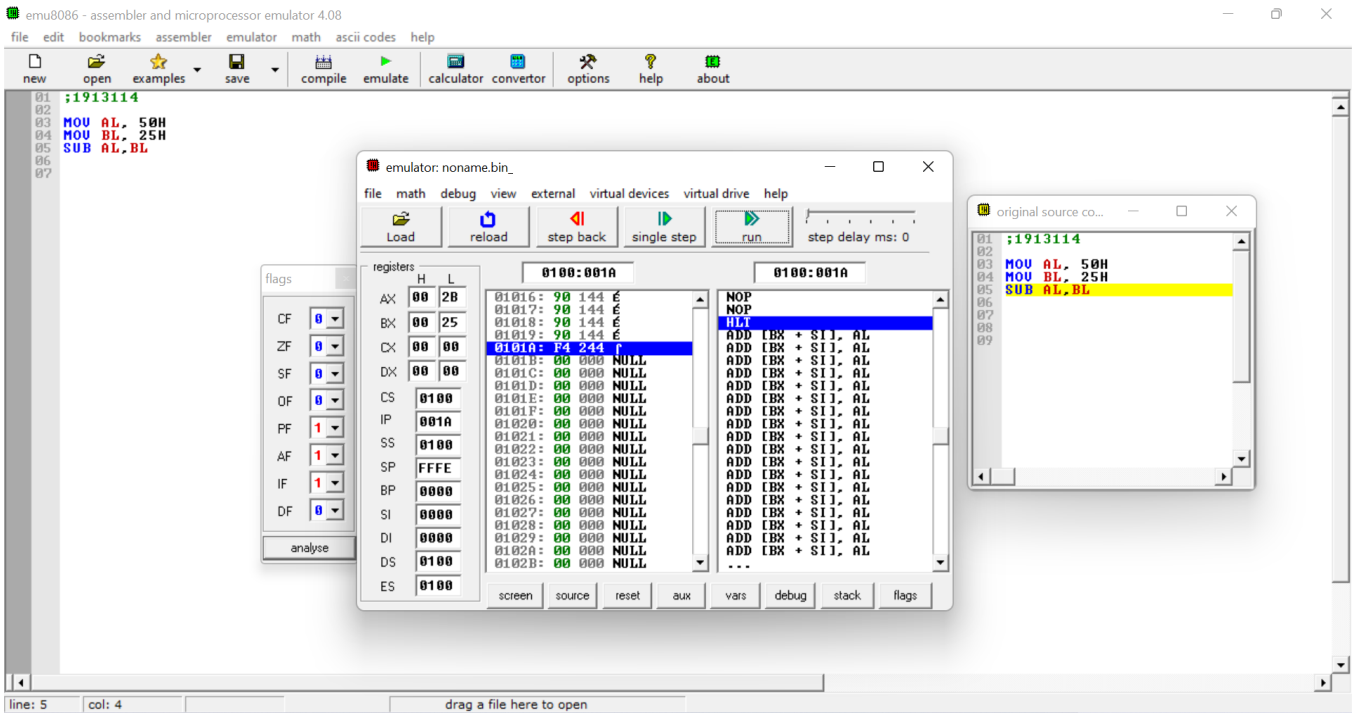
- **Program for subtraction of 8-bit numbers**

AIM- Subtraction of two 8-bit numbers.

CODE-

```
MOV AL, 50H
MOV BL, 25H
SUB AL,BL
```

OUTPUT-



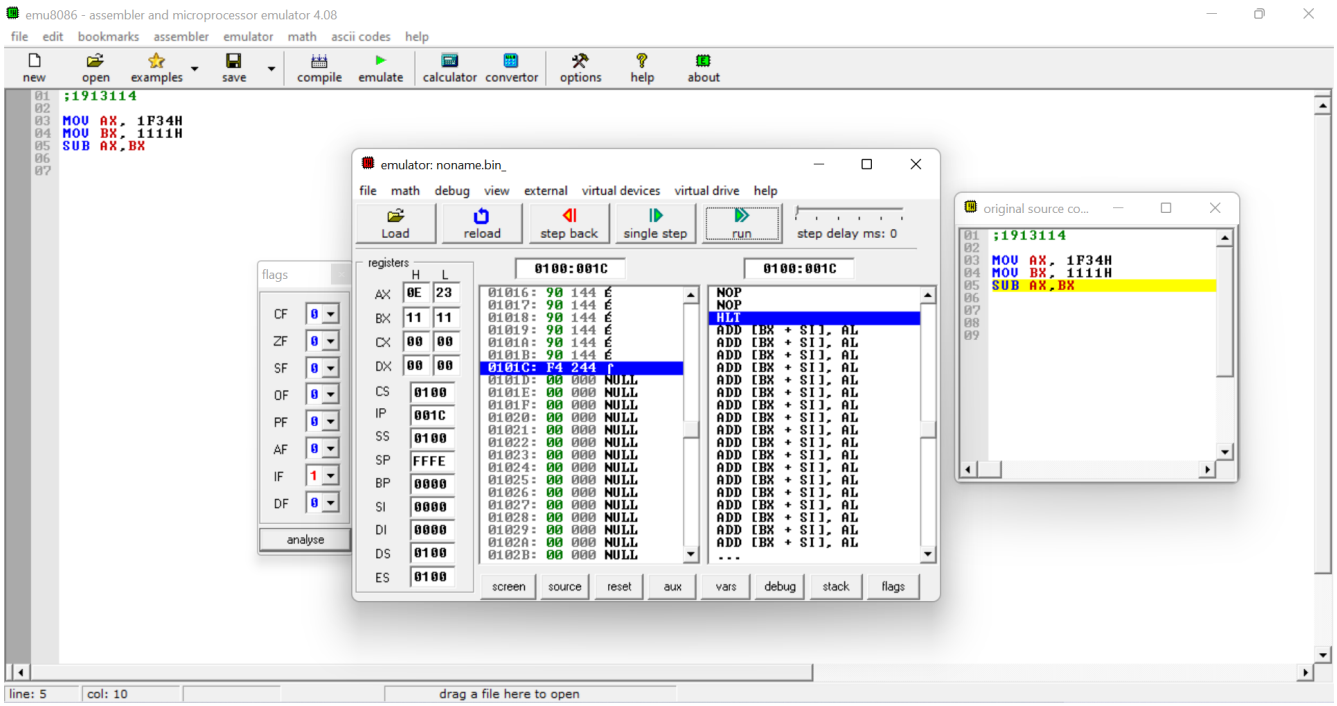
- **Program for subtraction of 16-bit numbers**

AIM- Subtraction of two 16-bit numbers.

CODE-

```
MOV AX, 1F34H
MOV BX, 1111H
SUB AX,BX
```

OUTPUT-



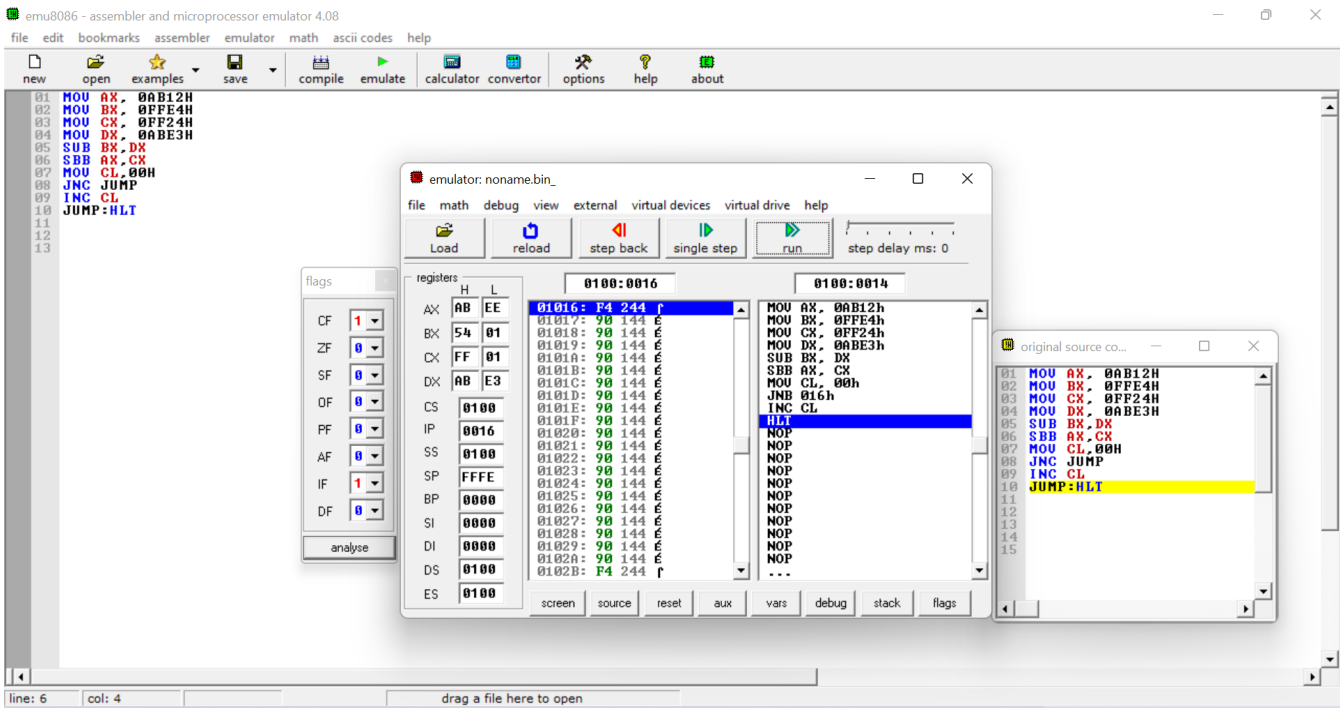
- **Program for subtraction of 32-bit numbers**

AIM- Subtraction of two 32-bit numbers.

CODE-

```
MOV AX, 0AB12H
MOV BX, 0FFE4H
MOV CX, 0FF24H
MOV DX, 0ABE3H
SUB BX,DX
SBB AX,CX
MOV CL,00H
JNC JUMP
INC CL
JUMP:HLT
```

OUTPUT-



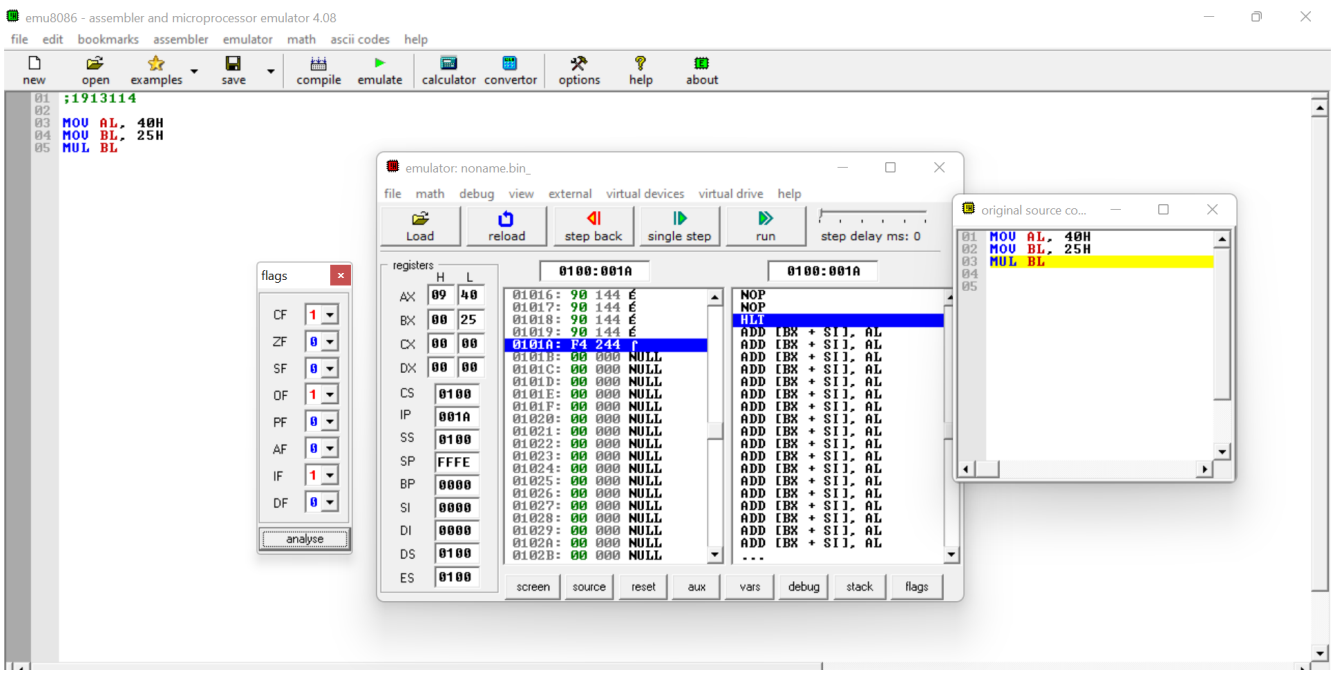
- **Program for multiplication of 8-bit numbers**

AIM- Multiplication of two 8-bit numbers.

CODE-

```
MOV AL, 40H
MOV BL, 25H
MUL BL
```

OUTPUT-



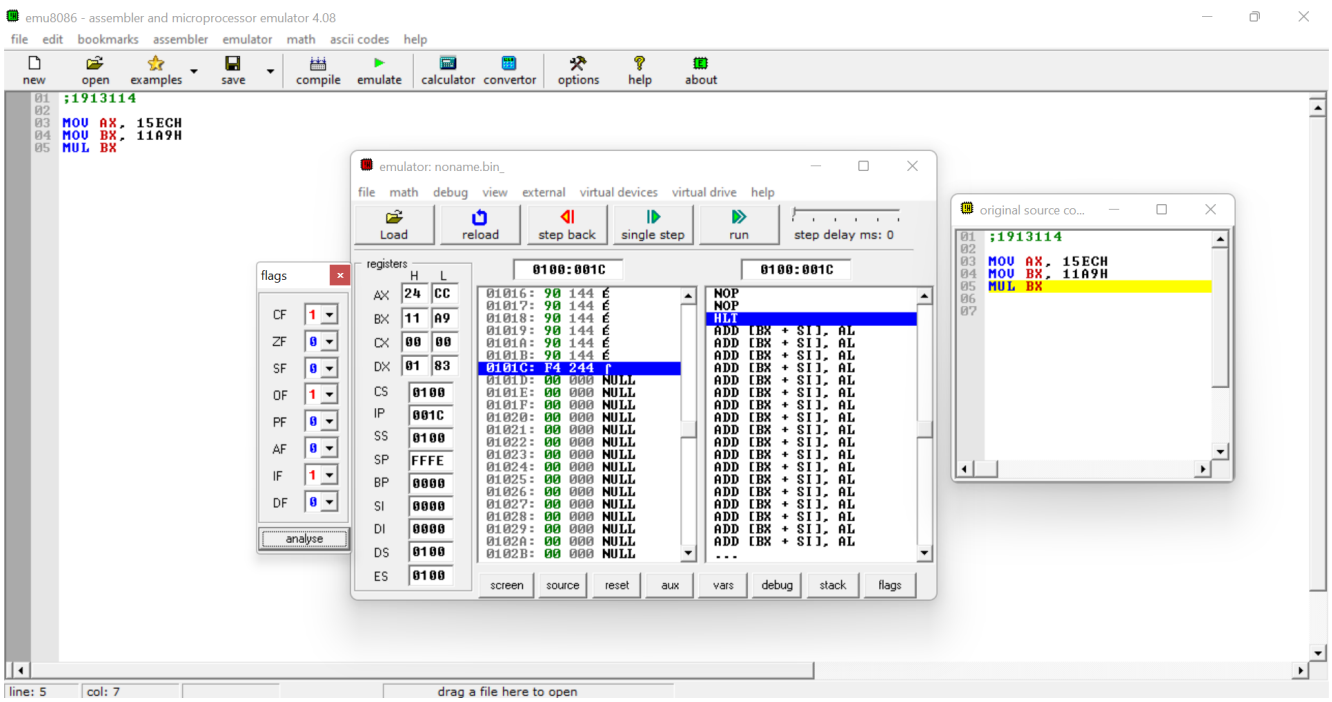
- **Program for multiplication of 16-bit numbers**

AIM- Multiplication of two 16-bit numbers.

CODE-

```
MOV AX, 15ECH
MOV BX, 11A9H
MUL BX
```

OUTPUT-



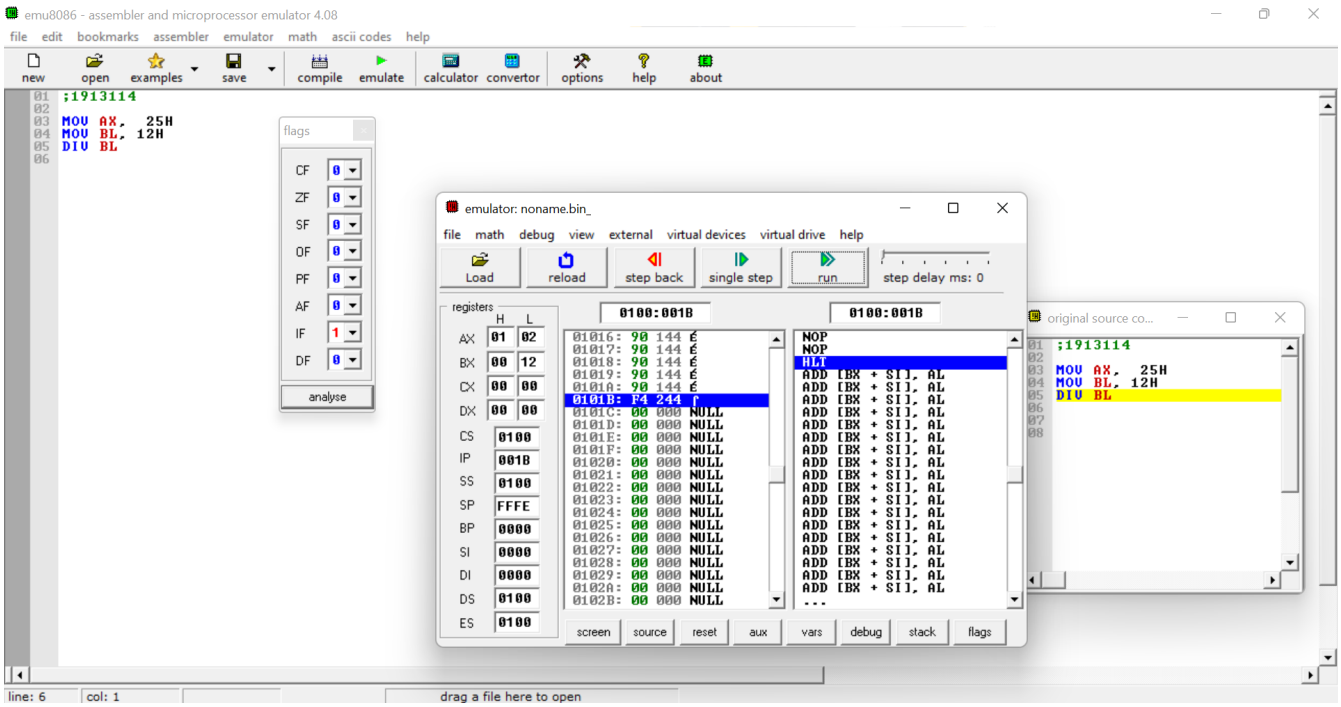
- **Program for division of 8-bit numbers**

AIM- Division of two 8-bit numbers.

CODE-

```
MOV AX,  25H
MOV BL, 12H
DIV BL
```

OUTPUT-



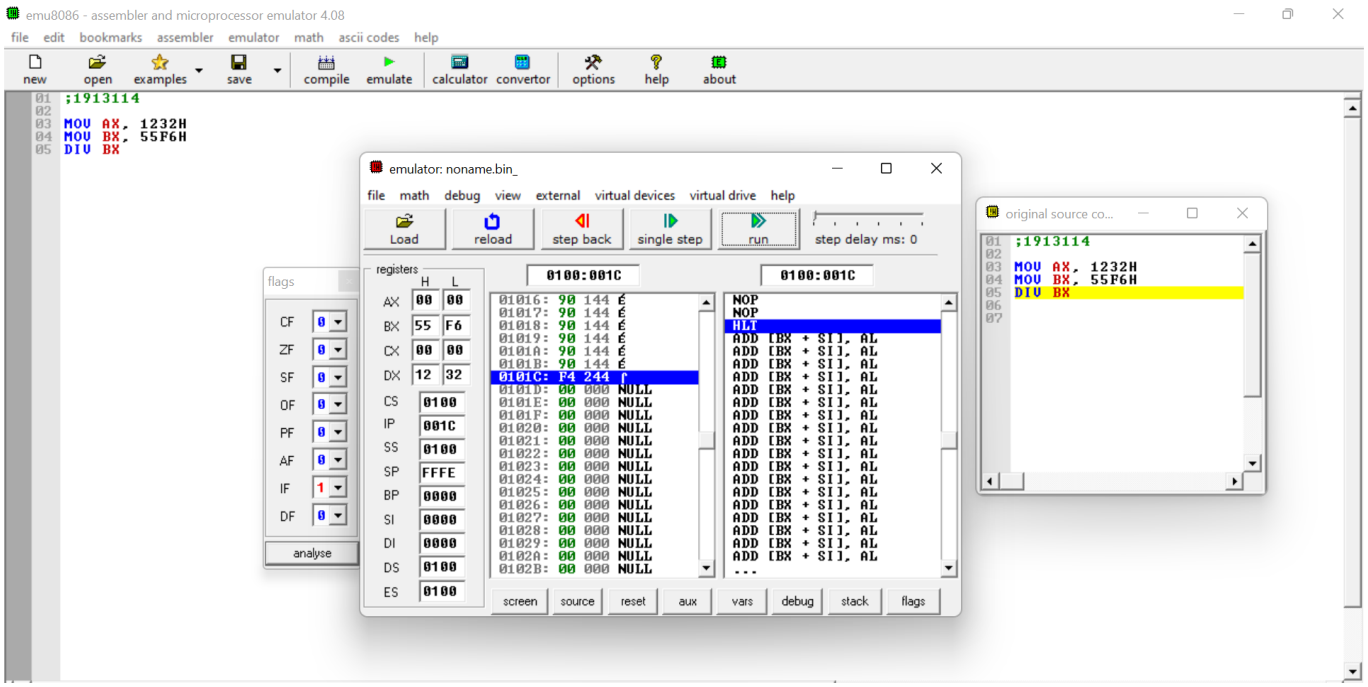
- **Program for division of 16-bit numbers**

AIM- Division of two 16-bit numbers.

CODE-

```
MOV AX,  1232H
MOV BX, 55F6H
DIV BX
```

OUTPUT-



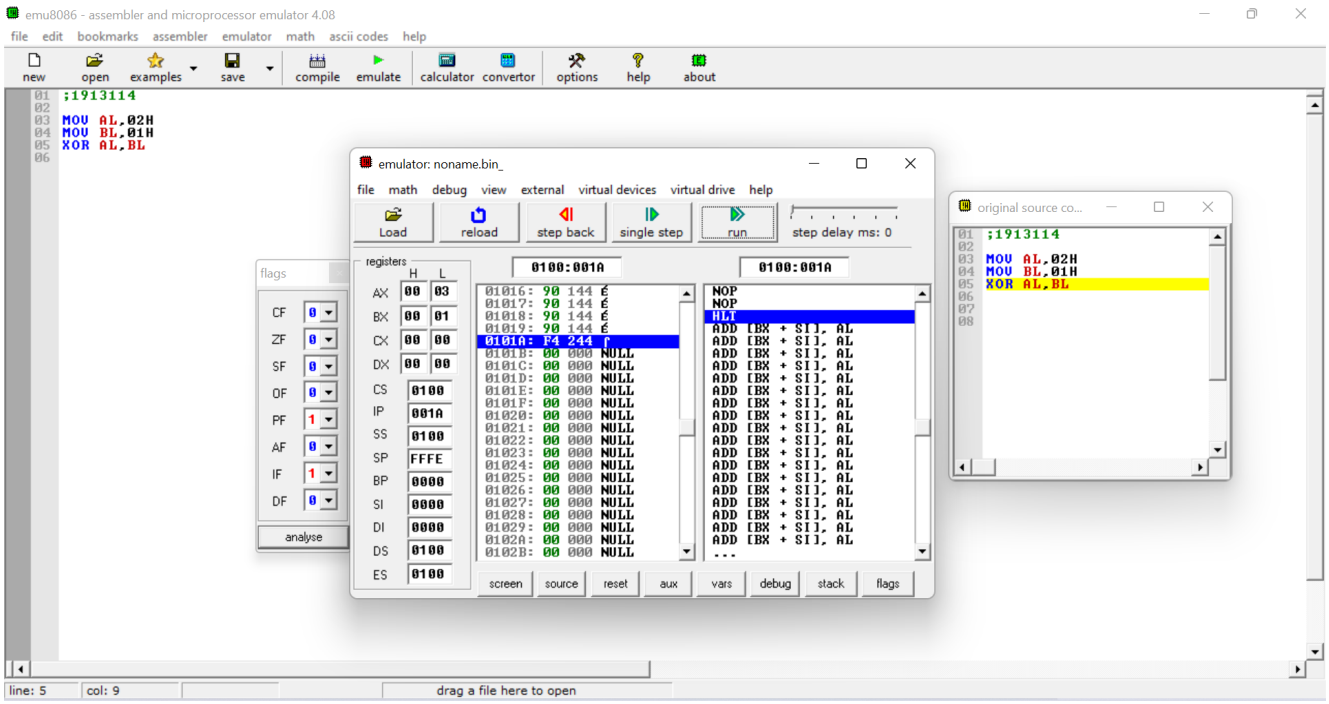
- **Program for XOR of 8-bit numbers**

AIM- XOR of two 8-bit numbers.

CODE-

```
MOV AL, 02H
MOV BL, 03H
XOR AL,BL
```

OUTPUT-



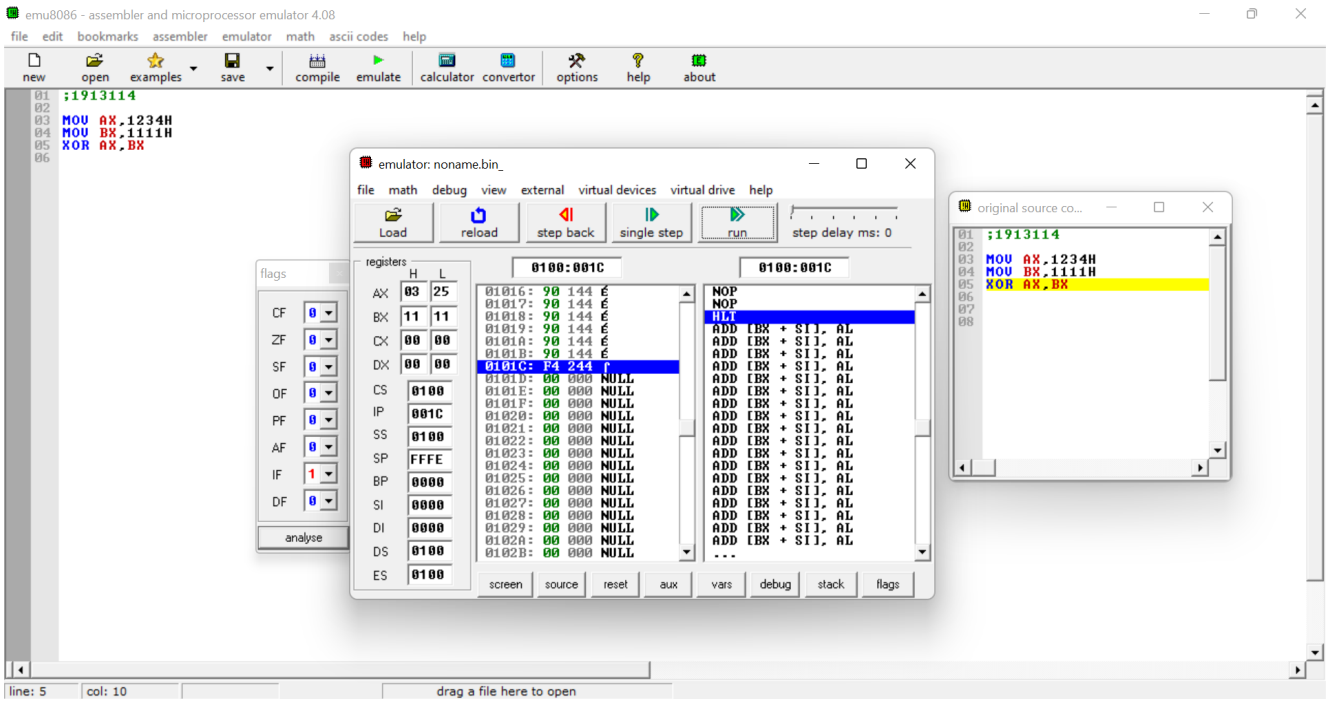
- **Program for XOR of 16-bit numbers**

AIM- XOR of two 16-bit numbers.

CODE-

```
MOV AX,1234H
MOV BX,1111H
XOR AX,BX
```

OUTPUT-



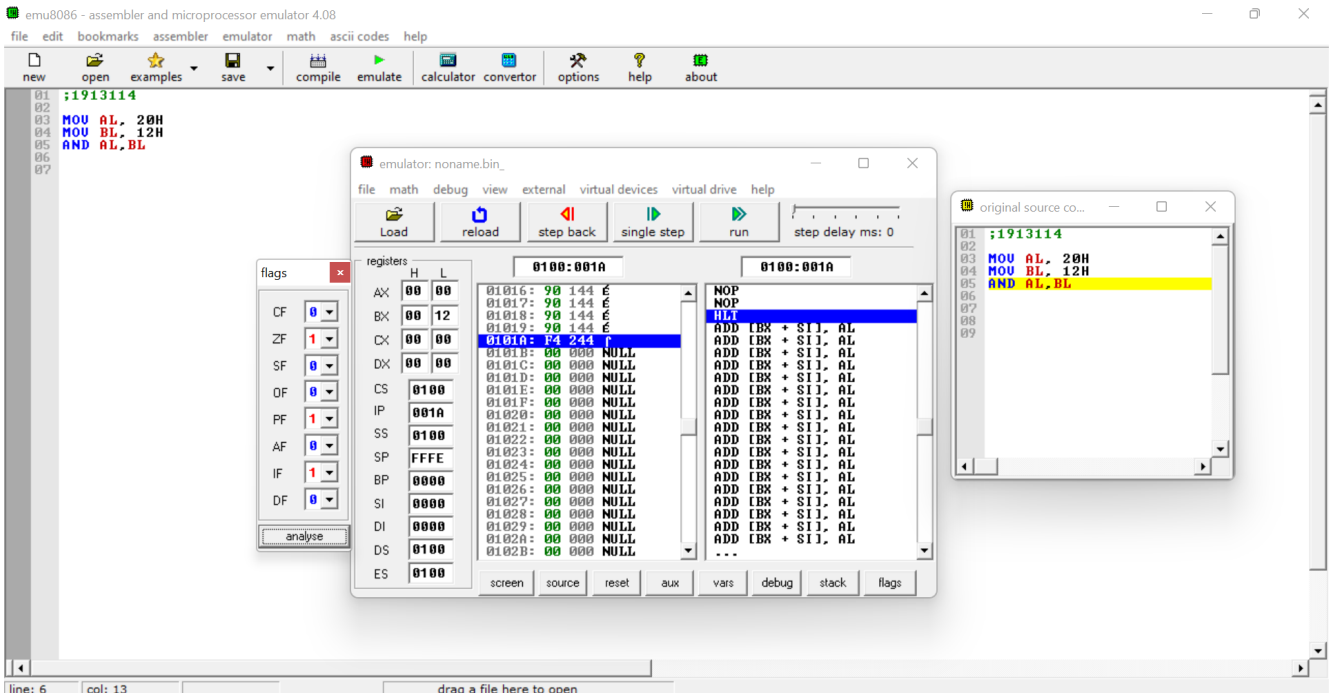
- **Program for AND of 8-bit numbers**

AIM- AND of two 8-bit numbers.

CODE-

```
MOV AL, 20H
MOV BL, 12H
AND AL,BL
```

OUTPUT-



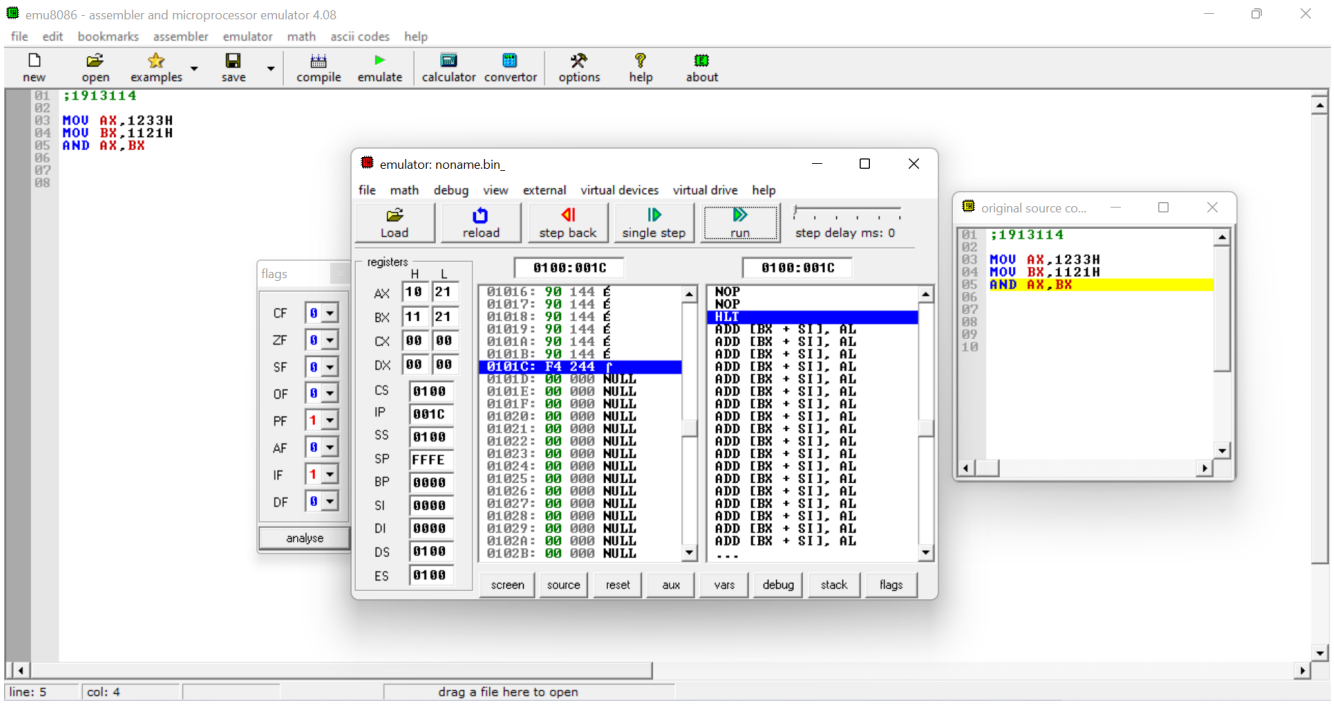
- **Program for AND of 16-bit numbers**

AIM- AND of two 16-bit numbers.

CODE-

```
MOV AX,1233H
MOV BX,1121H
AND AX,BX
```

OUTPUT-



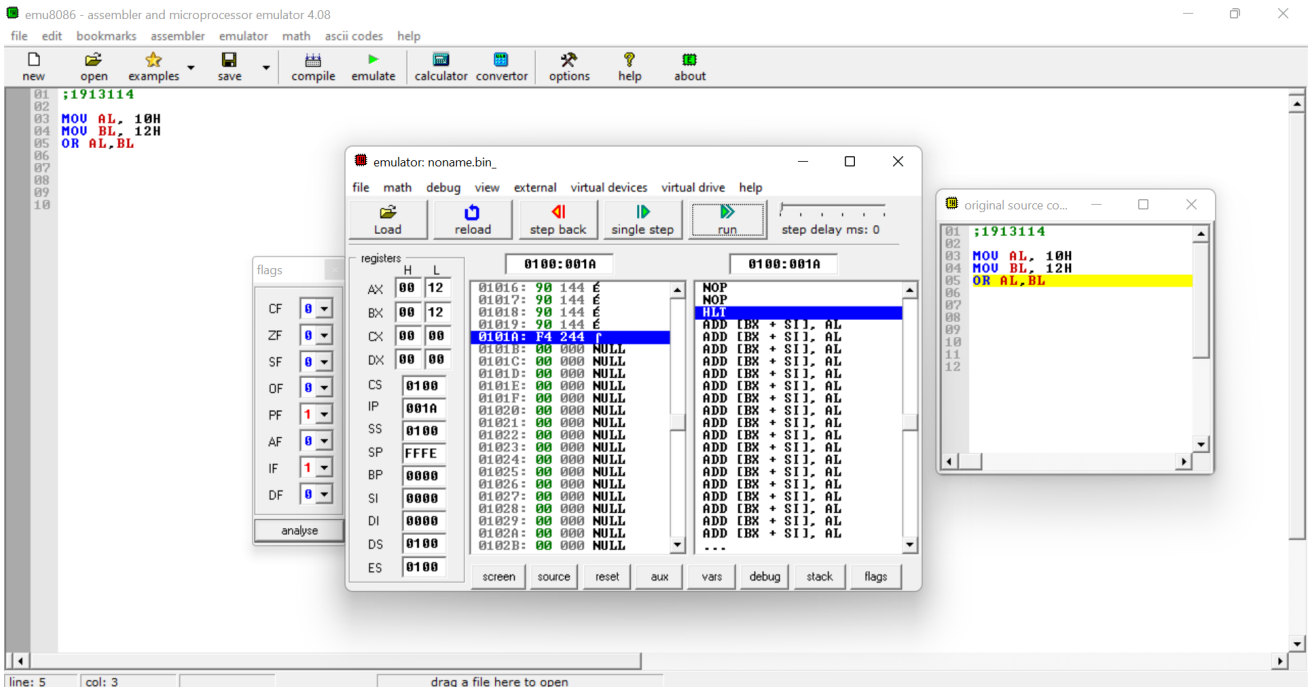
- **Program for OR of 8-bit numbers**

AIM- OR of two 8-bit numbers.

CODE-

```
MOV AL, 10H
MOV BL, 12H
OR AL,BL
```

OUTPUT-



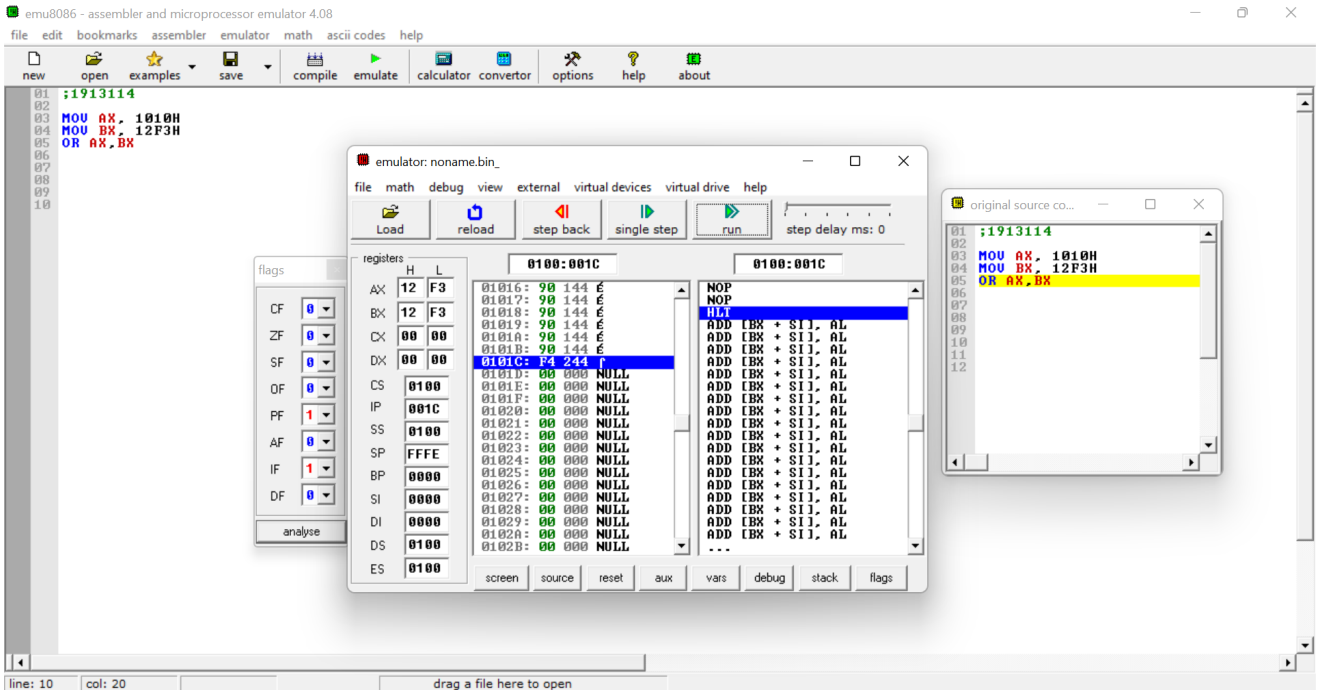
- **Program for OR of 16-bit numbers**

AIM- OR of two 16-bit numbers.

CODE-

```
MOV AX, 1010H
MOV BX, 12F3H
OR AX,BX
```

OUTPUT-



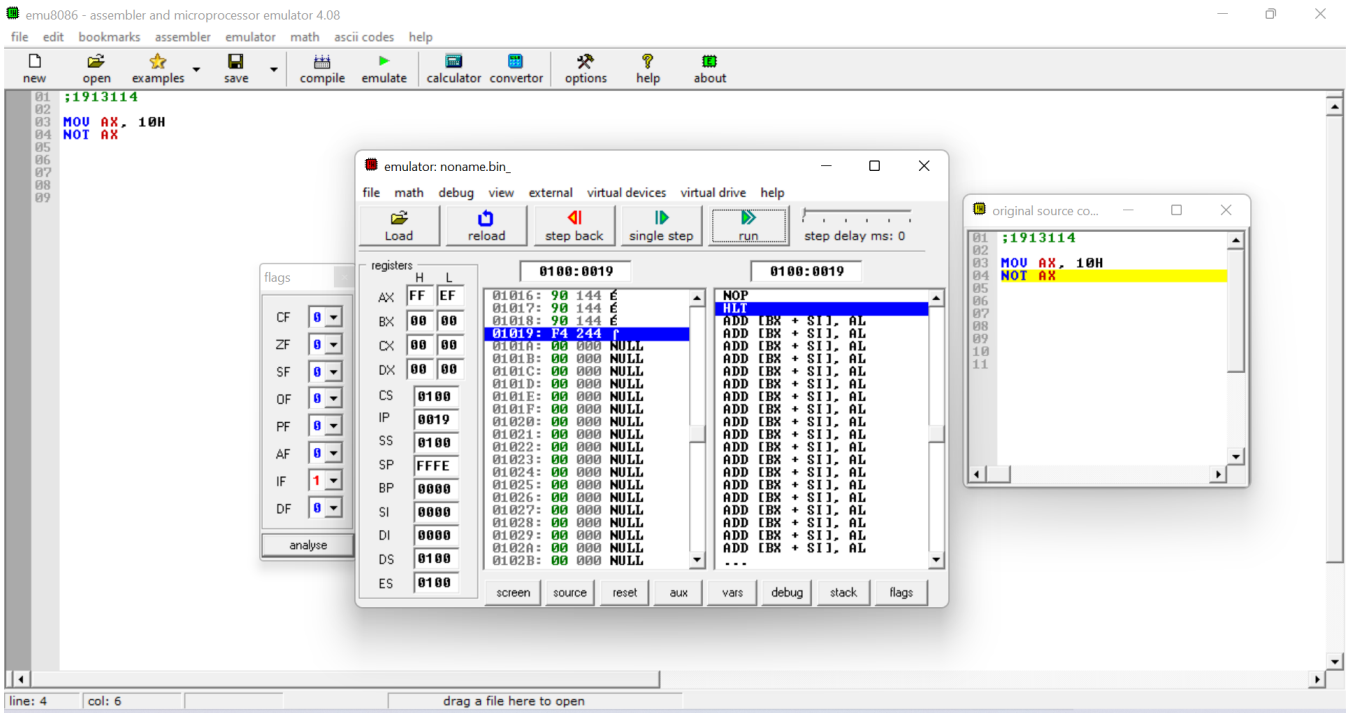
- **Program for NOT of 8-bit numbers**

AIM- NOT of two 8-bit numbers.

CODE-

```
MOV AX, 10H
NOT AX
```

OUTPUT-



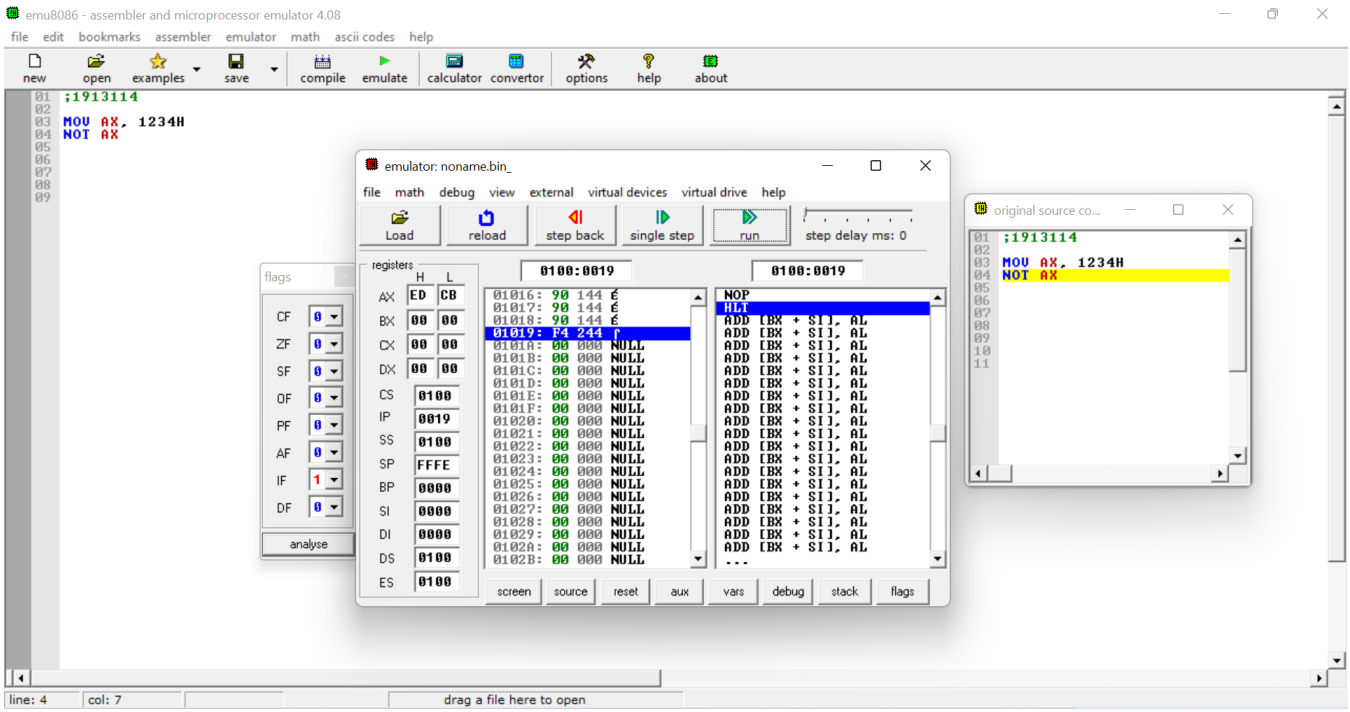
- **Program for NOT of 16-bit numbers**

AIM- NOT of two 16-bit numbers.

CODE-

```
MOV AX, 1234H
NOT AX
```

OUTPUT-



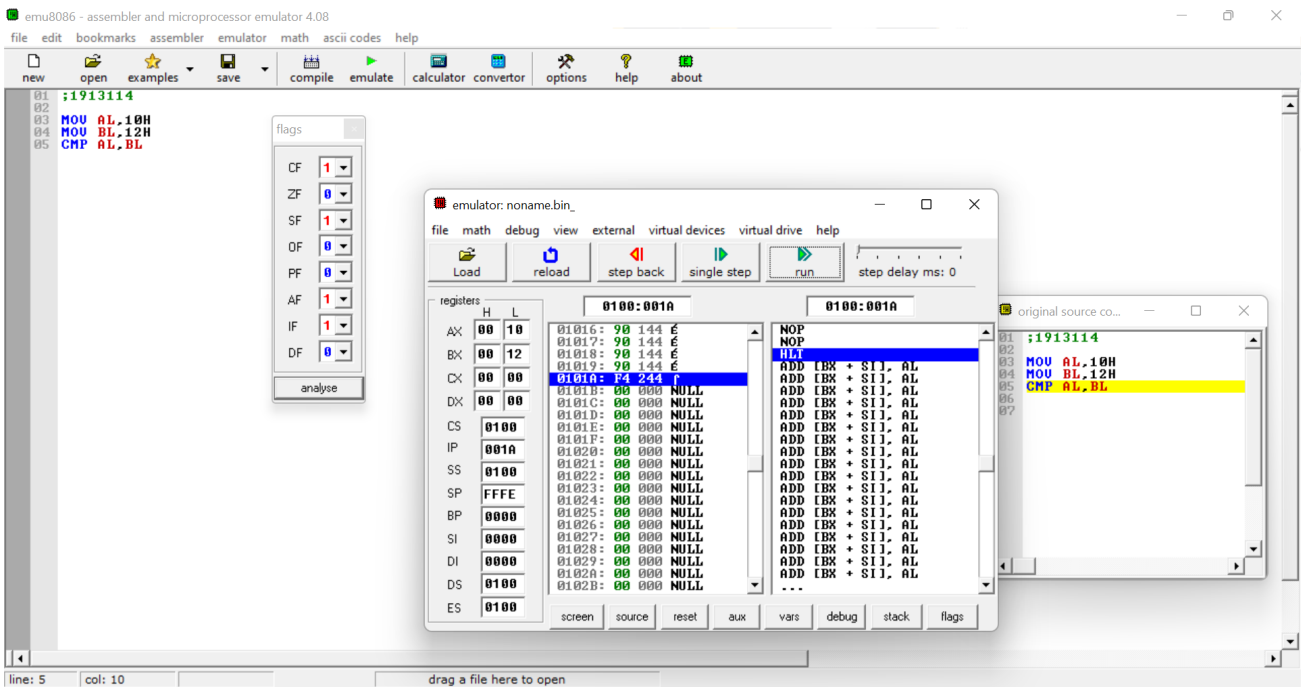
- **Program for comparison of 8-bit numbers**

AIM- Comparison of two 8-bit numbers.

CODE-

```
MOV AL,10H
MOV BL,12H
CMP AL,BL
```

OUTPUT-



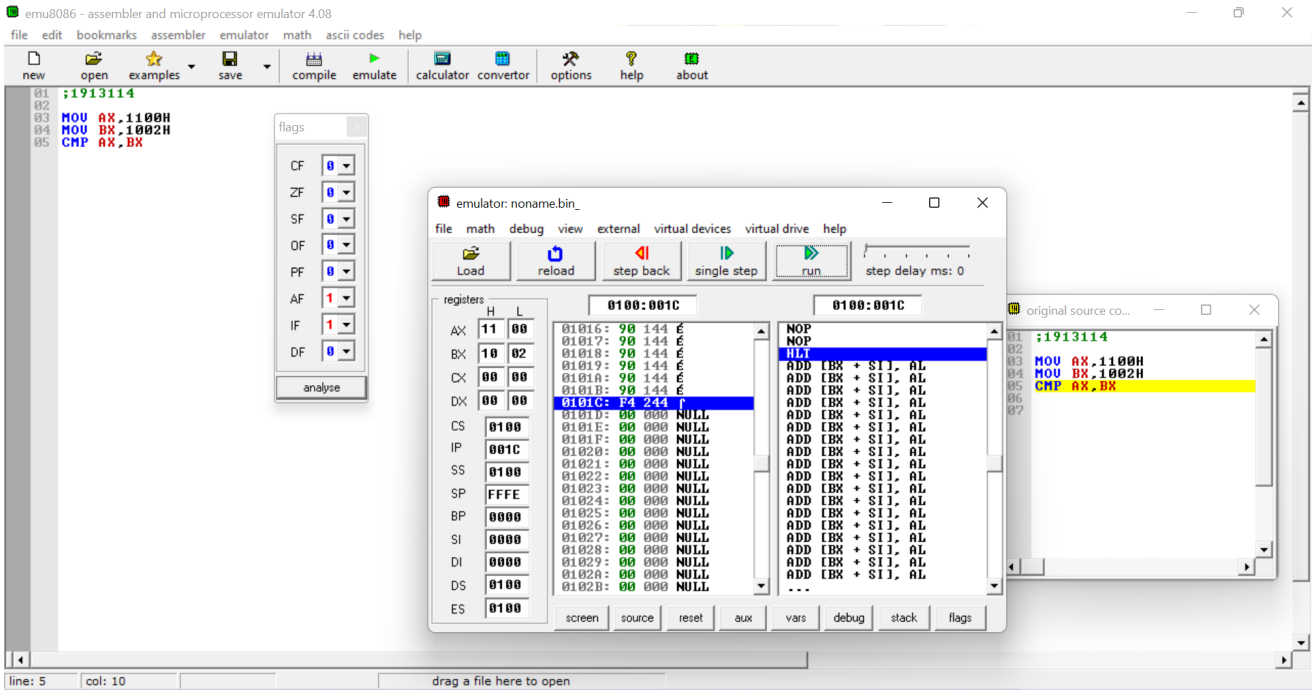
- **Program for comparison of 16-bit numbers**

AIM- Comparison of two 16-bit numbers.

CODE-

```
MOV AX,1100H
MOV BX,1002H
CMP AX,BX
```

OUTPUT-



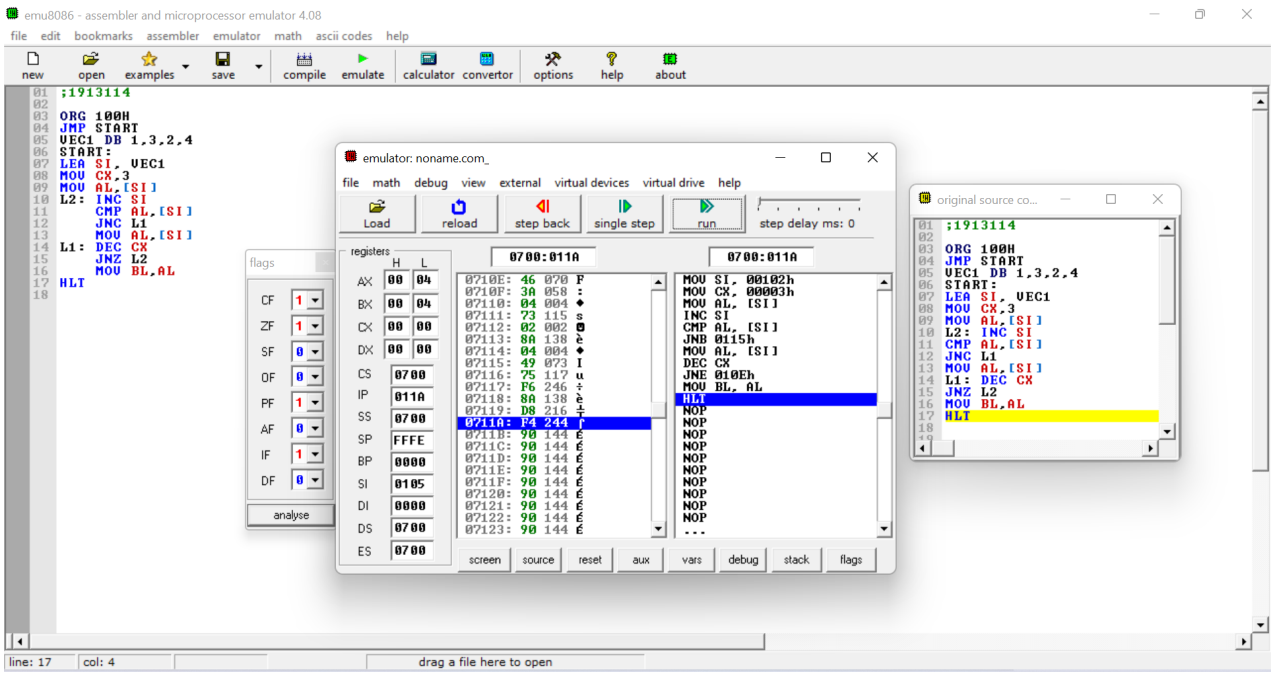
- **Program to find the maximum of N given numbers**

AIM- Find the maximum of N given numbers

CODE-

```
ORG 100H
JMP START
VEC1 DB 1,3,2,4
START:
LEA SI, VEC1
MOV CX,3
MOV AL,[SI]
L2: INC SI
    CMP AL,[SI]
    JNC L1
    MOV AL,[SI]
L1: DEC CX
   JNZ L2
    MOV BL,AL
HLT
```

OUTPUT-



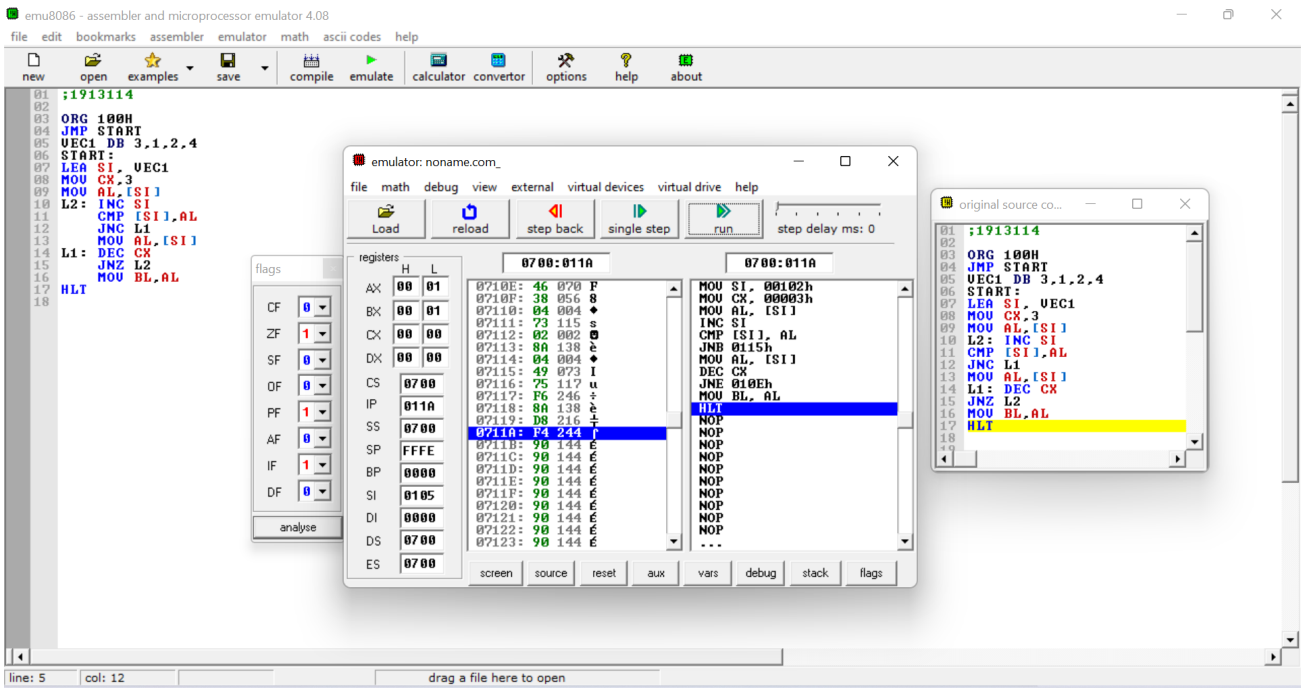
- **Program to find the minimum of N given numbers**

AIM- Find the minimum of N given numbers

CODE-

```
ORG 100H
JMP START
VEC1 DB 3,1,2,4
START:
LEA SI, VEC1
MOV CX,3
MOV AL,[SI]
L2: INC SI
    CMP [SI],AL
    JNC L1
    MOV AL,[SI]
L1: DEC CX
   JNZ L2
    MOV BL,AL
HLT
```

OUTPUT-



- **Program to arrange a given numbers in ascending order**

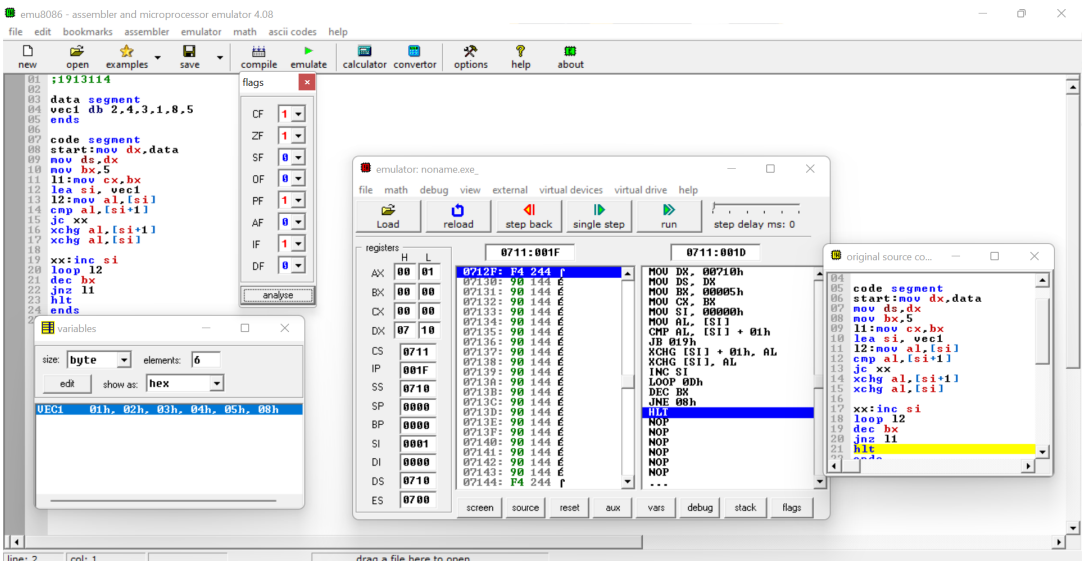
AIM- Arrange a given numbers in ascending order

CODE-

```
data segment
vec1 db 2,4,3,1,8,5
ends
code segment
start:mov dx,data
mov ds,dx
mov bx,5
l1:mov cx,bx
lea si, vec1
l2:mov al,[si]
cmp al,[si+1]
jc xx
xchg al,[si+1]
xchg al,[si]

xx:inc si
loop l2
dec bx
jnz l1
hlt
ends
end start
```

OUTPUT-



- **Program to arrange a given numbers in descending order**

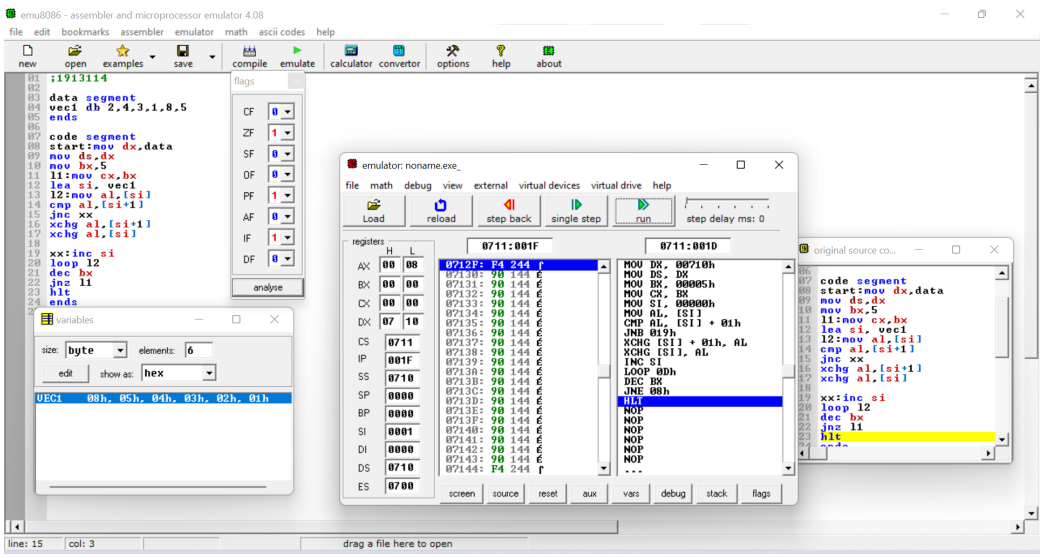
AIM- Arrange a given numbers in descending order

CODE-

```
data segment
vec1 db 2,4,3,1,8,5
ends
code segment
start:mov dx,data
mov ds,dx
mov bx,5
l1:mov cx,bx
lea si, vec1
l2:mov al,[si]
cmp al,[si+1]
jnc xx
xchg al,[si+1]
xchg al,[si]

xx:inc si
loop l2
dec bx
jnz l1
hlt
ends
end start
```

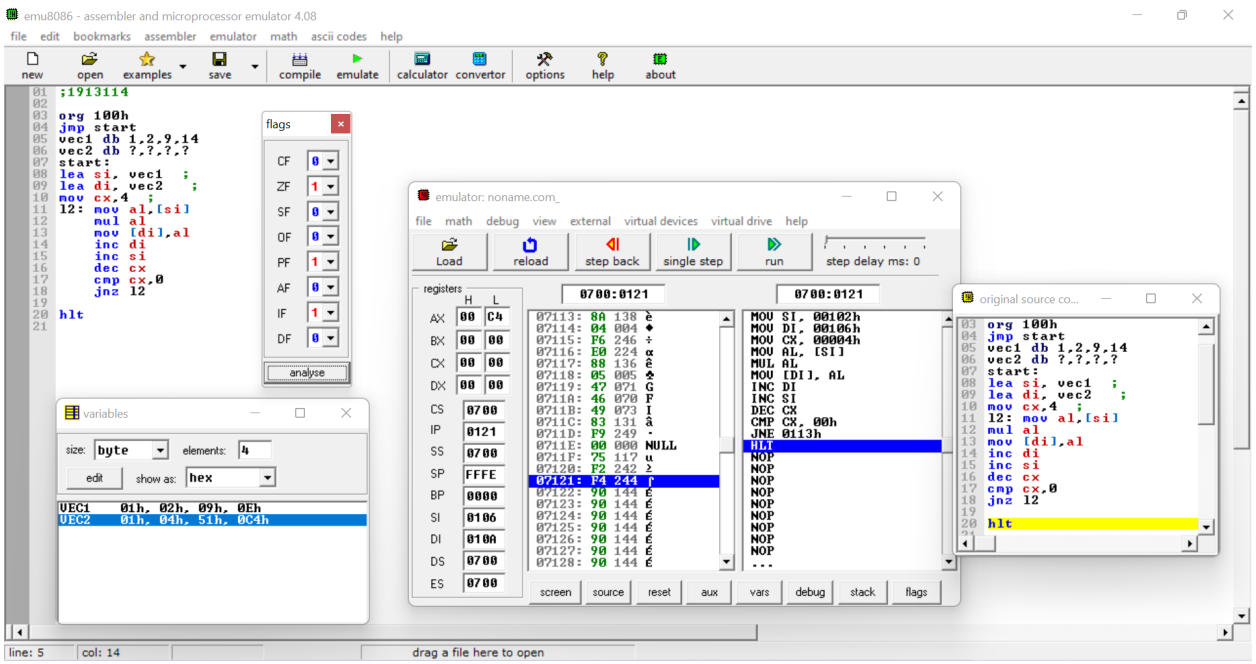
OUTPUT-



- **Program to do square of the given series**

AIM- Square of given series

CODE-
```
org 100h
    jmp start
    vec1 db 1,2,9,14
    vec2 db ?,?,?,?
    start:
    lea si, vec1  ;
    lea di, vec2   ;
    mov cx,4  ;
    l2: mov al,[si]
        mul al
        mov [di],al
        inc di
        inc si
        dec cx
        cmp cx,0
        jnz l2
    hlt
```
OUTPUT-



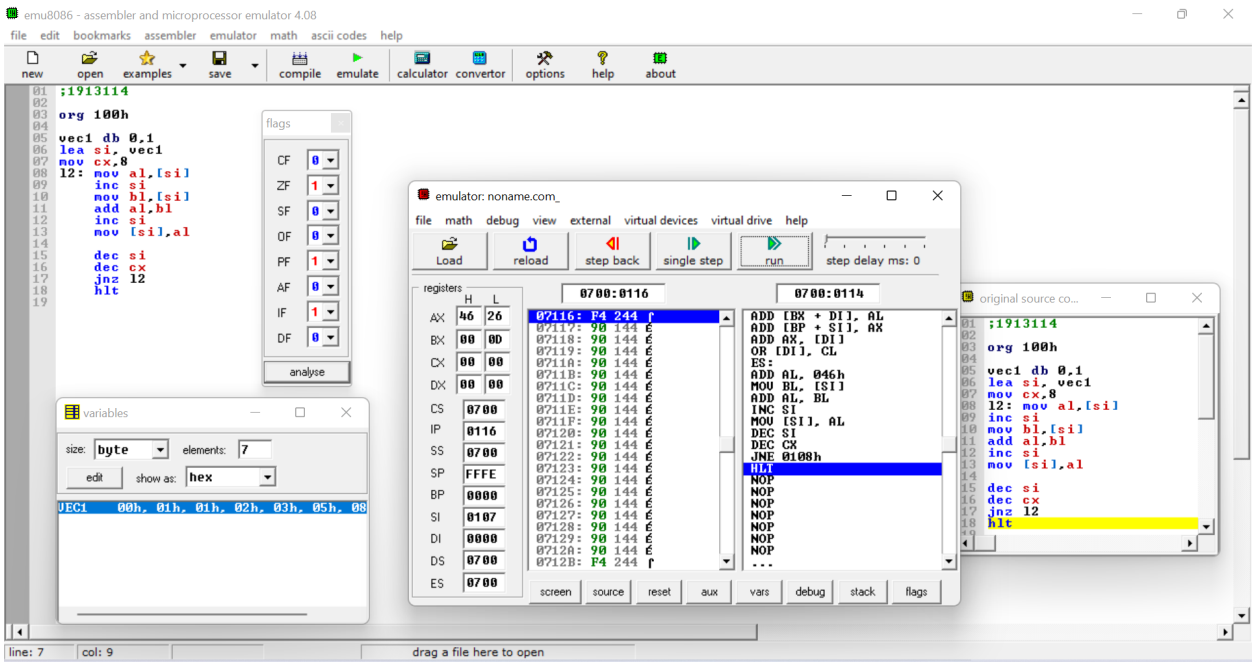
- **Program to generate the Fibonacci series**

AIM- Generate Fibonacci series

CODE-
```
vec1 db 0,1
lea si, vec1
mov cx,8
l2: mov al,[si]
    inc si
    mov bl,[si]
    add al,bl
    inc si
    mov [si],al

    dec si
    dec cx
    jnz l2
    hlt
```

OUTPUT-

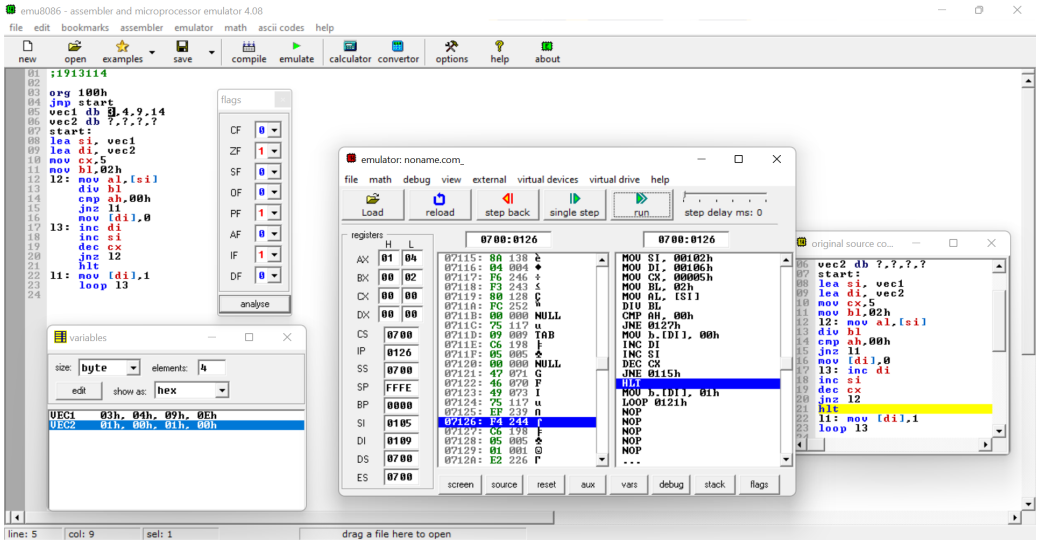


- **Program to find out EVEN and ODD numbers in a series**

AIM- Find odd numbers in a series

CODE-
```
org 100h
    jmp start
    vec1 db 3,4,9,14
    vec2 db ?,?,?,?
    start:
    lea si, vec1
    lea di, vec2
    mov cx,5
    mov bl,02h
    l2: mov al,[si]
        div bl
        cmp ah,00h
        jnz l1
        mov [di],0
    l3: inc di
        inc si
        dec cx
        jnz l2
        hlt
    l1: mov [di],1
        loop l3
```

OUTPUT-



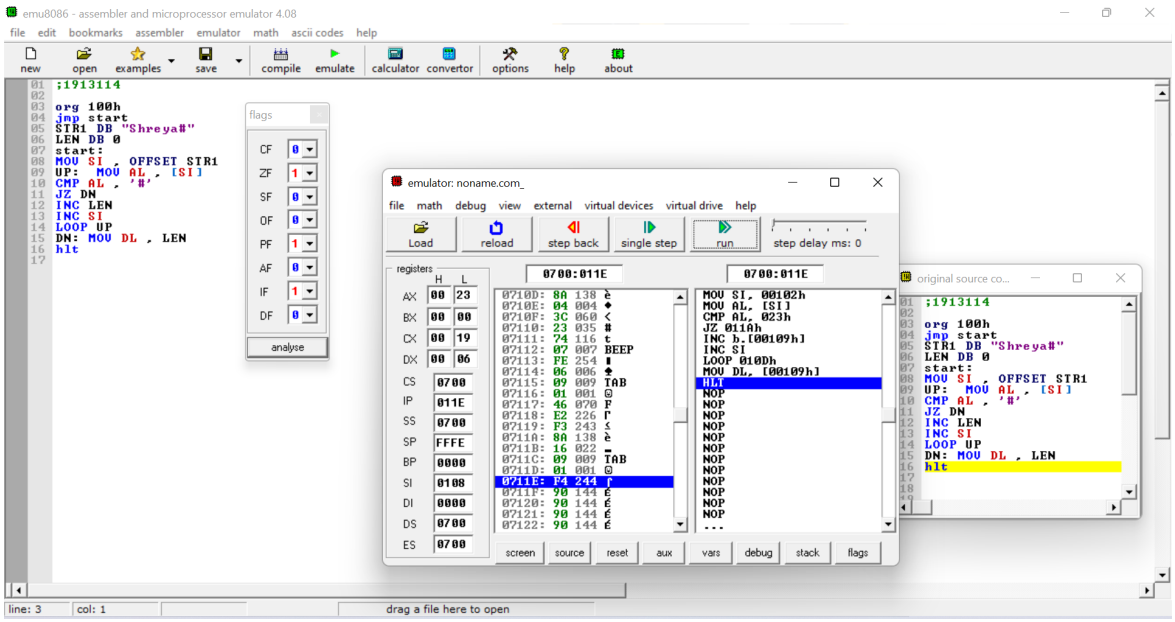
- **Program to count the length of a string**

AIM- Count the length of a string

CODE-

```
org 100h
jmp start
STR1 DB "Shreya#"
LEN DB 0
start:
MOV SI , OFFSET STR1
UP:  MOV AL , [SI]
CMP AL , '#'
JZ DN
INC LEN
INC SI
LOOP UP
DN: MOV DL , LEN
hlt
```

OUTPUT-



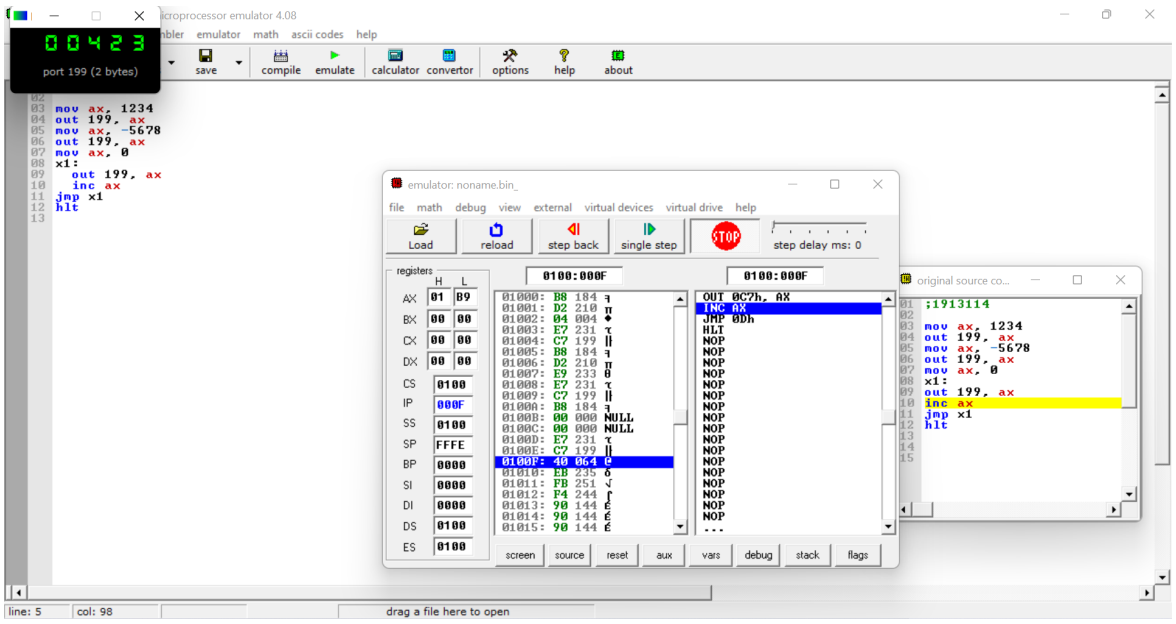
- **Program to display data on LED**

AIM- Display data on led

CODE-

```
mov ax, 1234
out 199, ax
mov ax, -5678
out 199, ax
mov ax, 0
x1:
 out 199, ax
 inc ax
jmp x1
hlt
```

OUTPUT-



- **Program to transfer the content of one memory location to another memory location**

AIM- Transfer content from one memory location to another

CODE- org 100h
```
    jmp start
    vec1 db 3,7,1,7
    vec2 db ?,?,?,?
    start:
    lea si, vec1
    lea bx, vec2
    mov cx,5
    l1: mov al,[si]
        mov [bx],al
        inc si
        inc bx
        dec cx
        jnz l1
        hlt
```

OUTPUT-