BANASTHALI VIDYAPEETH

Department of Computer Science and Engineering

Microprocessors and Microcontrollers

Lab File (ELE306L)

Submitted To –

Dr. Urvashi Prakash Shukla

Submitted By :

Shruti Goel

B.Tech Computer Science

6th Semester

1913120

BTBTC19145

Section:C

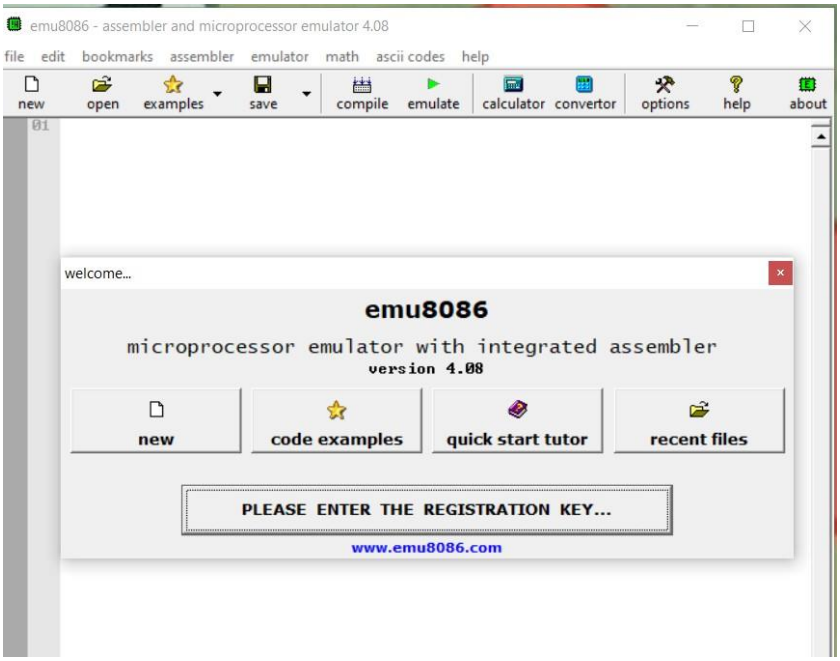
Experiment 1

AIM: Introduction to Emulator 8086

This is a microprocessor emulator with an integrated 8086 Assembler. The emulator can runprograms on a Virtual Machine, and emulate real hardware including screen, memory, and input and output devices. It

helps you program in assembly language. The source code is compiled by assembler and then executed o n Emulator step-by-step, allowing you to watch registers, flags, and memory while your program runs.

To speed up the process operations, the processor includes some internal memory storage locations called registers.



How to access the register?

8086 CPU has 4 general purpose registers, each register has its own name. General purpose registers are used to store temporary data within the microprocessor. There are 8 general purpose registers in 8086 microprocessors.

GENERAL PURPOSE REGISTER

1. **AX** – This is the accumulator. It is of 16 bits and is divided into two 8-bit registers AH and AL to also perform 8-bit instructions. It is generally used for arithmetical and logical instructions but in 8086 microprocessor it is not mandatory to have accumulator as the destination operand.

2. **BX** – This is the base register. It is of 16 bits and is divided into two 8-bit registers BH and BL to also perform 8-bit instructions. It is used to store the value of the offset.

3. **CX** – This is the counter register. It is of 16 bits and is divided into two 8- bit registers CH and CL to also perform 8-bit instructions. It is used in looping and rotation.

4. **DX** – This is the data register. It is of 16 bits and is divided into two 8-bit registers DH and DL to also perform 8-bit instructions. It is used in multiplication an input/output port addressing.

POINTER REGISTER:

1. **SP** – This is the stack pointer. It is of 16 bits. It points to the topmost item of the stack. If the stack is empty the stack pointer will be (FFFE)H. It's offset address relative to stack segment.

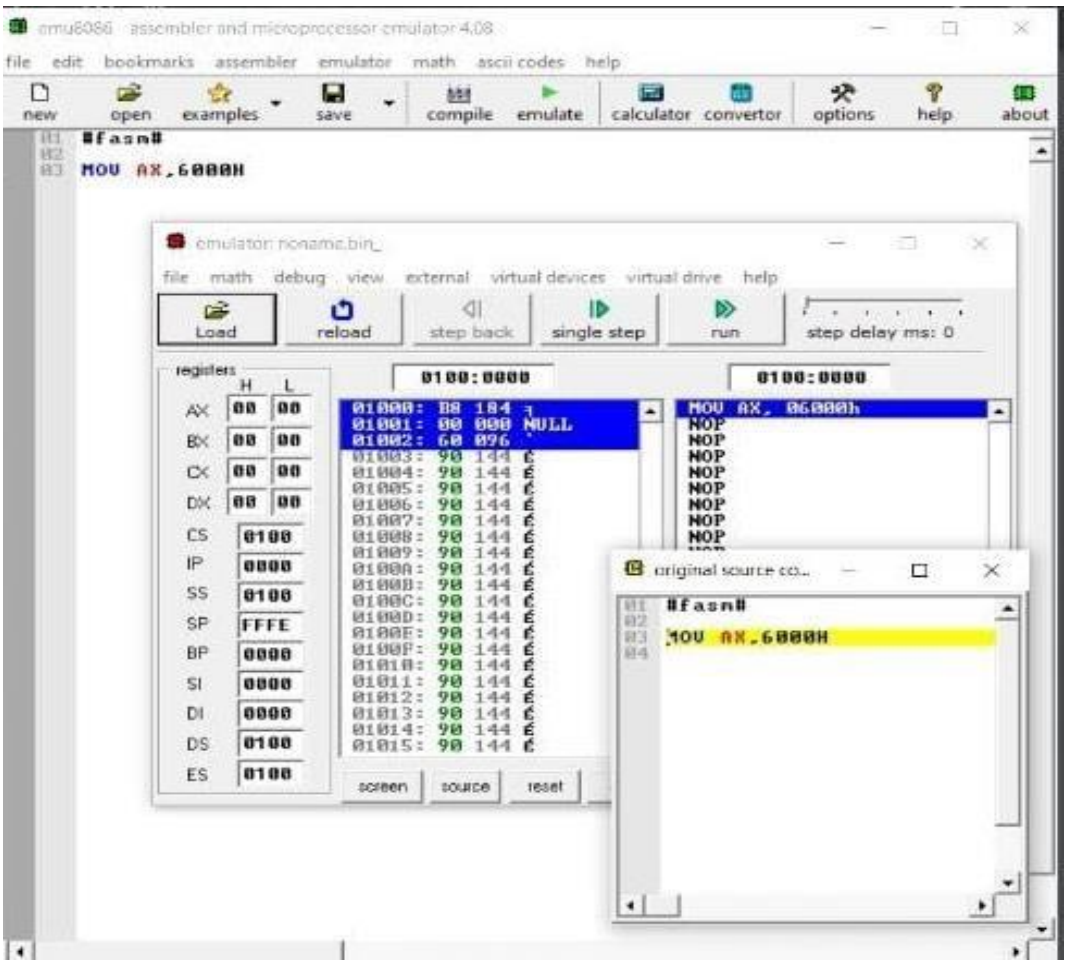
2. **BP** – This is the base pointer. It is of 16 bits. It is primary used in accessing parameters passed by the stack. It's offset address relative to stack segment.

INDEX REGISTER:

1. **SI** – This is the source index register. It is of 16 bits. It is used in the pointer addressing of data and as a source in some string related operations. It is offset is relative to data segment.

2. **DI** – This is the destination index register. It is of 16 bits. It is used in the pointer addressing of data and as a destination in some string related operations. It is offset is relative to extra segment.

The main purpose of a register is to keep a number (variable). Therefore, when we 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. 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, it is preferred 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.



Experiment 2

AIM: Instruction set: Programming and Illustration

The 8086 microprocessor supports 8 types of instructions −

1.Data Transfer Instructions

2.Arithmetic Instructions

3.Bit Manipulation Instructions

4.String Instructions

5.Program Execution Transfer Instructions (Branch & Loop Instructions)

6.Processor Control Instructions

7.Iteration Control Instructions

8.Interrupt Instructions

1. Data Transfer Instructions

These instructions are used to transfer the data from the source operand to the destination operand. Following are the list of instructions under this group −

i. Instruction to transfer a word

●MOV − Used to copy the byte or word from the provided source to the provided destination.

●PPUSH − Used to put a word at the top of the stack.

●POP − Used to get a word from the top of the stack to the provided location.

●PUSHA − Used to put all the registers into the stack.

●POPA − Used to get words from the stack to all registers.

●XCHG − Used to exchange the data from two locations.

●XLAT − Used to translate a byte in AL using a table in the memory. ii.

Instructions for input and output port transfer

● IN − Used to read a byte or word from the provided port to the accumulator. ●OUT

− Used to send out a byte or word from the accumulator to the provided port.

iii. Instructions to transfer the address

●LEA − Used to load the address of operand into the provided register. ●LDS −

Used to load DS register and other provided register from the memory

●LES − Used to load ES register and other provided register from the memory.

iv. Instructions to transfer flag registers

●LAHF − Used to load AH with the low byte of the flag register.

●SAHF − Used to store AH register to low byte of the flag register.

●PUSHF − Used to copy the flag register at the top of the stack.

●POPF − Used to copy a word at the top of the stack to the flag register.

2. Arithmetic Instructions

These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc. Following is the list of instructions under this group −

i. Instructions to perform addition

●ADD − Used to add the provided byte to byte/word to word.

●ADC − Used to add with carry.

●INC − Used to increment the provided byte/word by 1.

●AAA − Used to adjust ASCII after addition.

●DAA − Used to adjust the decimal after the addition/subtraction operation. ii.

Instructions to perform subtraction

●SUB − Used to subtract the byte from byte/word from word.

●SBB − Used to perform subtraction with borrow.

●DEC − Used to decrement the provided byte/word by 1.

●NPG − Used to negate each bit of the provided byte/word and add 1/2's complement.

●CMP − Used to compare 2 provided byte/word.

●AAS − Used to adjust ASCII codes after subtraction.

●DAS − Used to adjust decimal after subtraction.

iii. Instruction to perform multiplication

●MUL − Used to multiply unsigned byte by byte/word by word.

●IMUL − Used to multiply signed byte by byte/word by word.

●AAM − Used to adjust ASCII codes after multiplication. iv.

Instructions to perform division

●DIV − Used to divide the unsigned word by byte or unsigned double word by word.

●IDIV − Used to divide the signed word by byte or signed double word by word.

●AAD − Used to adjust ASCII codes after division.

●CBW − Used to fill the upper byte of the word with the copies of sign bit of the lower byte.

●CWD − Used to fill the upper word of the double word with the sign bit of the lower word.

3. Bit Manipulation Instructions

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

i. Instructions to perform logical operation

●NOT − Used to invert each bit of a byte or word.

●AND − Used for adding each bit in a byte/word with the corresponding bit in another byte/word.

●OR − Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.

●XOR − Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.

●TEST − Used to add operands to update flags, without affecting operands. ii.

Instructions to perform shift operations

●SHL/SAL − Used to shift bits of a byte/word towards left and put zero(S) in LSBs.

●SHR − Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.

●SAR − Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB. iii.

Instructions to perform rotate operations

●ROL − Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF].

●ROR − Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].

●RCR − Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.

●RCL − Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

4. String Instructions

String is a group of bytes/words and their memory is always allocated in a sequential order. Following is the list of instructions under this group −

●REP − Used to repeat the given instruction till CX ≠ 0.

●REPE/REPZ − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.

●REPNE/REPNZ − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.

●MOVS/MOVSB/MOVSW − Used to move the byte/word from one string to another.

●COMS/COMPSB/COMPSW − Used to compare two string bytes/words.

●INS/INSB/INSW − Used as an input string/byte/word from the I/O port to the provided memory location.
●OUTS/OUTSB/OUTSW − Used as an output string/byte/word from the provided memory location to the I/O port.

●SCAS/SCASB/SCASW − Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.

●LODS/LODSB/LODSW − Used to store the string byte into AL or string word into AX.

5. Program Execution Transfer Instructions (Branch and Loop Instructions)

These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions −

i. Instructions to transfer the instruction during an execution without any condition −

●CALL − Used to call a procedure and save their return address to the stack.

●RET − Used to return from the procedure to the main program.

●JMP − Used to jump to the provided address to proceed to the next instruction.

ii. Instructions to transfer the instruction during an execution with some conditions −

●JA/JNBE − Used to jump if above/not below/equal instruction satisfies.

●JAE/JNB − Used to jump if above/not below instruction satisfies.

●JBE/JNA − Used to jump if below/equal/ not above instruction satisfies.

●JC − Used to jump if carry flag CF = 1

●JE/JZ − Used to jump if equal/zero flag ZF = 1

●JG/JNLE − Used to jump if greater/not less than/equal instruction satisfies.

●JGE/JNL − Used to jump if greater than/equal/not less than instruction satisfies.

●JL/JNGE − Used to jump if less than/not greater than/equal instruction satisfies.

●JLE/JNG − Used to jump if less than/equal/if not greater than instruction satisfies.

●JNC − Used to jump if no carry flag (CF = 0)

●JNE/JNZ − Used to jump if not equal/zero flag ZF = 0

●JNO − Used to jump if no overflow flag OF = 0

●JNP/JPO − Used to jump if not parity/parity odd PF = 0

●JNS − Used to jump if not sign SF = 0

●JO − Used to jump if overflow flag OF = 1

●JP/JPE − Used to jump if parity/parity even PF = 1

●JS − Used to jump if sign flag SF = 1

6. Processor Control Instructions

These instructions are used to control the processor action by setting/resetting the flag values. Following are the instructions under this group −

●STC − Used to set carry flag CF to 1

●CLC − Used to clear/reset carry flag CF to 0

●CMC − Used to put complement at the state of carry flag CF.

●STD − Used to set the direction flag DF to 1

●CLD − Used to clear/reset the direction flag DF to 0

●STI − Used to set the interrupt enable flag to 1, i.e., enable INTR input.

●CLI − Used to clear the interrupt enable flag to 0, i.e., disable INTR input.

7. Iteration Control Instructions

These instructions are used to execute the given instructions for number of times. Following is the list of instructions under this group −

●LOOP − Used to loop a group of instructions until the condition satisfies, i.e., CX = 0

●LOOPE/LOOPZ − Used to loop a group of instructions till it satisfies ZF = 1 & CX = 0

●LOOPNE/LOOPNZ − Used to loop a group of instructions till it satisfies ZF = 0 & CX = 0 ●JCXZ

− Used to jump to the provided address if CX = 0

8. Interrupt Instructions

These instructions are used to call the interrupt during program execution.

●INT − Used to interrupt the program during execution and calling service specified.

●INTO − Used to interrupt the program during execution if OF = 1

●IRET − Used to return from interrupt service to the main program

Experiment 3

AIM:

- **Program for addition of 8/16/32-bit numbers**
- **Program for subtraction of 8/16/32-bit number.**
- **Program for multiplication of 8/16-bit number.**
- **Program for division of 8/16-bit number.**

CODE:

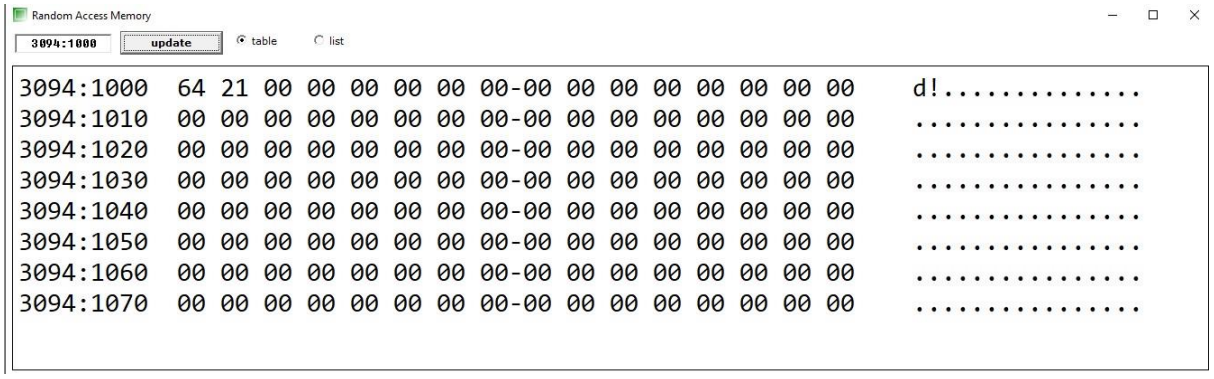
- ADDITION OF 8-BIT WITHOUT CARRY:

```
MOV AX, 3094H       ; Storing 3094H in AX Register
MOV DS, AX          ; Initializing DS with the help of AX
MOV SI  1000H       ; Storing the offset value in SI as 1000H(Assumed)
MOV AL, [SI]        ; Moving the content of 31940 Memory Location to AL
MOV BL, [SI+1]      ; Moving the content of 31941 Memory Location to BL
ADD AL, BL          ; Add AL and BL
MOV [SI+2], AL      ; Storing the value of the result in 31942 Memory Location HLT
```

OUTPUT:

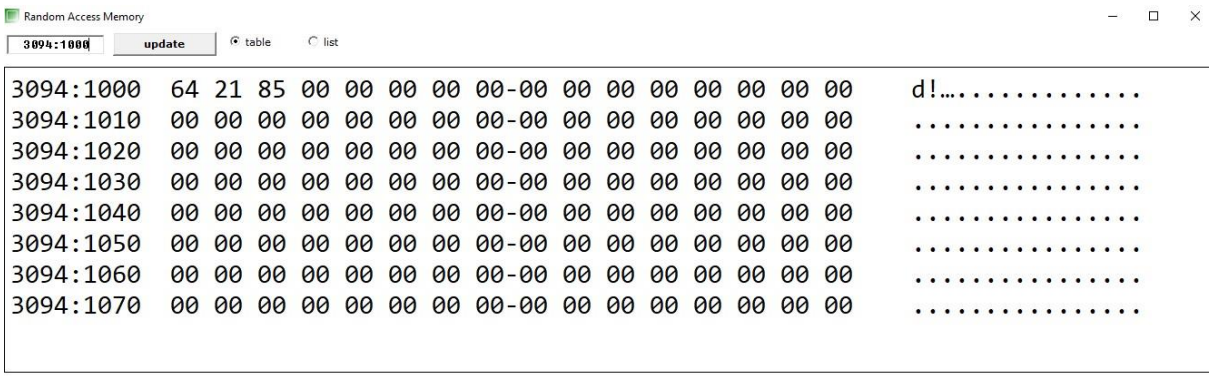
Values in memory location (Values to be added)

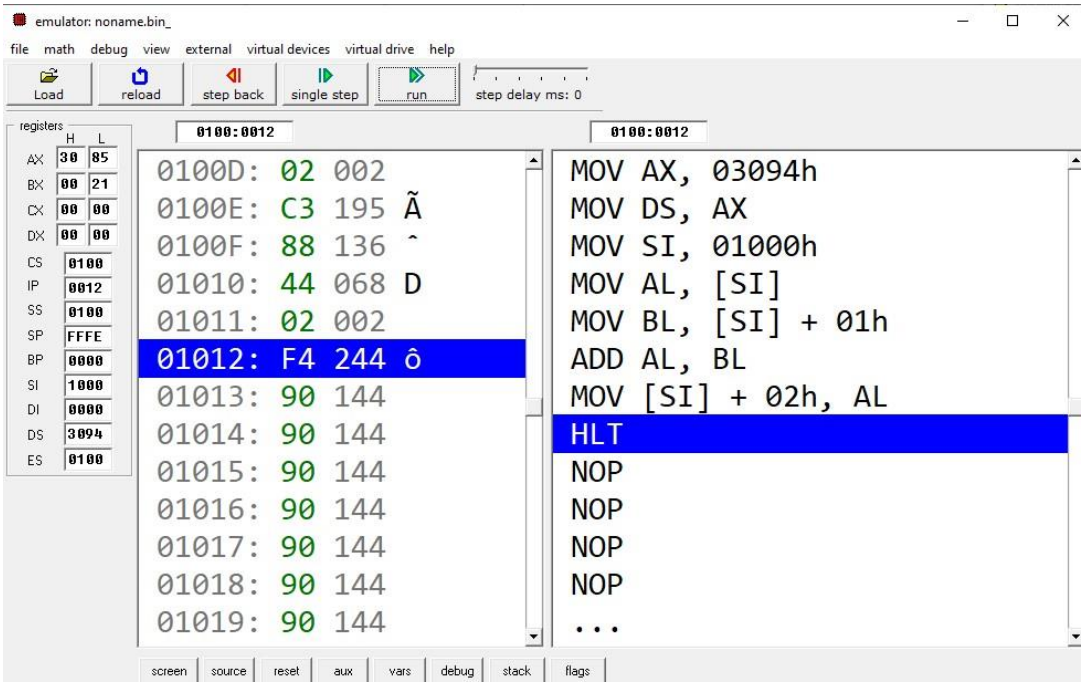
[31940] = 64H , [31941] = 21H



After running the program the result is

[31942] = 85H (Addition of 64H and 21H)





- ADDITION OF 16-BIT WITHOUT CARRY:

```
MOV AX, 3094H     ; Storing 3094H in AX Register
MOV DS, AX        ; Initializing DS with the help of AX
MOV SI, 1000H     ; Storing the offset value in SI as 1000H(Assumed)
MOV AX, [SI]      ; Moving the content of 31940 and 31941 Memory Location to
                    AX
MOV BX, [SI+2]    ; Moving the content of 31942 and 31943 Memory Location to
                    BX
ADD AX, BX        ; Add AX and BX
MOV [SI+4], AX    ; Storing the result in 31944 and 31945 Memory Location
HLT
```

OUTPUT:

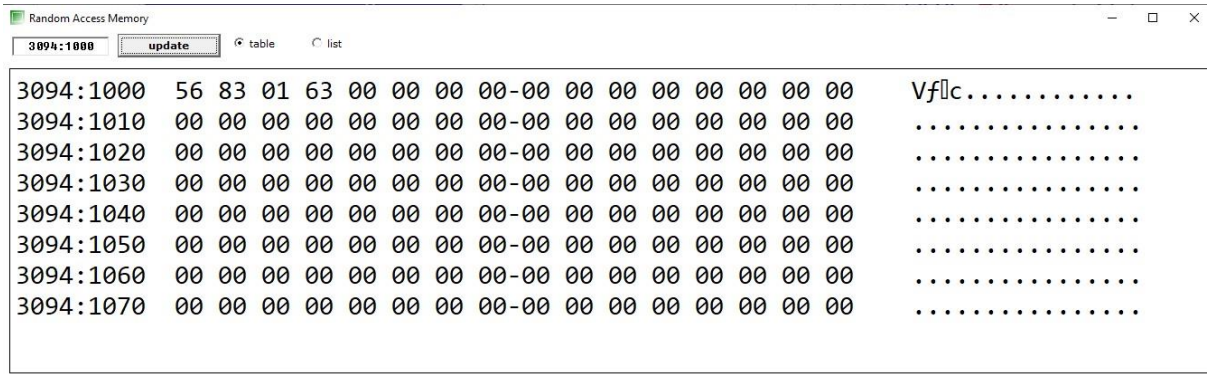
Values in memory location (Values to be added)

[31940] = 56H

[31941] = 83H

[31942] = 01H

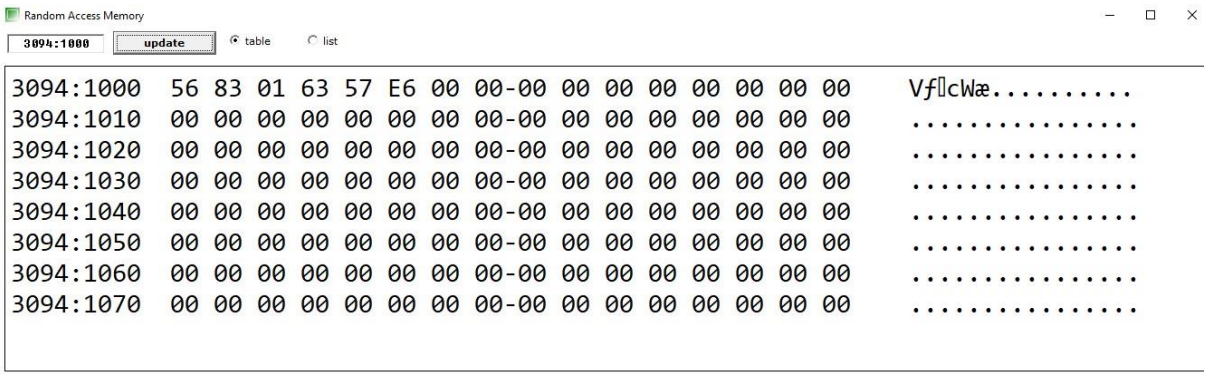
[31943] = 63H



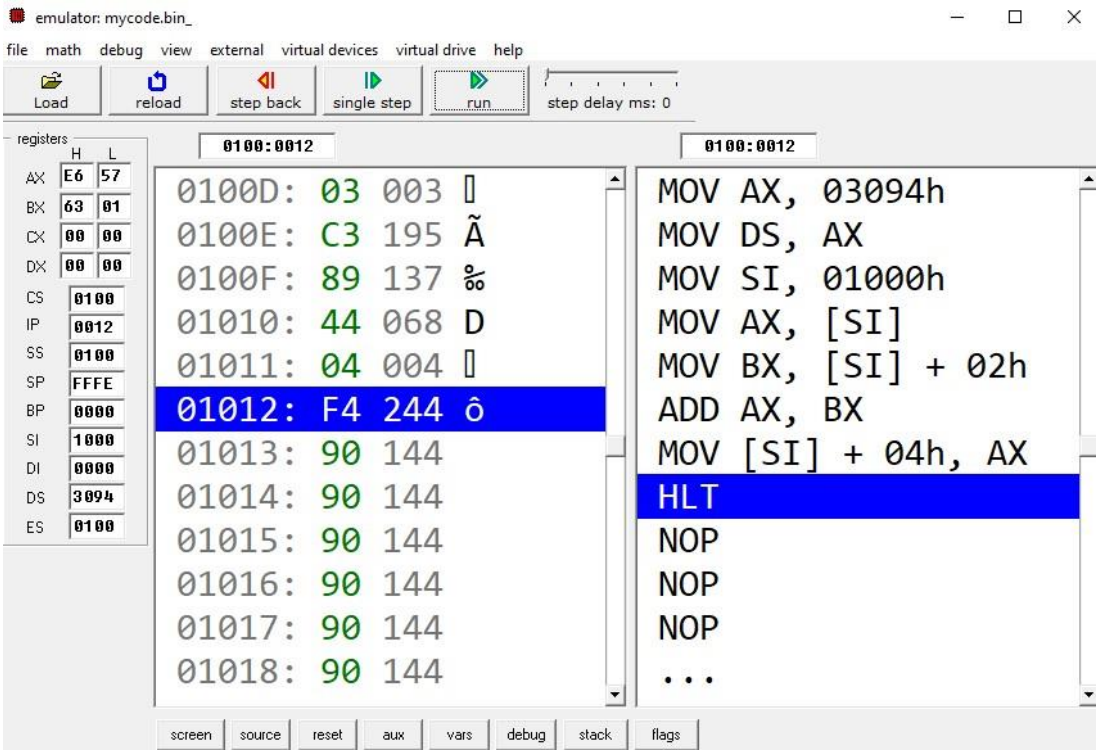
After running the program the result is

[31944] = 57H (Result of 56H+01H)

[31945] = E6H (Result of 83H+63H)



The emulator result window with updated values of all the registers is shown as



- ADDITION OF 32-BIT:

```
MOV AX , 3094H          ; Storing 3094H in AX Register
MOV DS , AX             ; Initializing DS with the help of AX
MOV SS , AX             ; Initializing SS with the help of AX
MOV SI , 1000H          ; Storing 1000H in SI
MOV DI , 2000H          ; Storing 2000H in DI
MOV BP , 3000H          ; Storing 3000H in BP
MOV AX , [SI]           ; Storing the content of 31940 and 31941 memory location
                            to AH and AL resp.
MOV BX , [SI+2]         ; Storing the content of 31942 and 31943 memory location
                            to BH and BL resp.
MOV CX , [DI]           ; Storing the content of 32940 and 32941 memory location
                            to CH and CL resp.
MOV DX , [DI+2]         ; Storing the content of 31942 and 31943 memory location
                            to DH and DL resp.
ADD AX , CX             ; Adding the values of AX and CX i.e, the lower byte of
```

```
                        data
ADC BX , DX             ; Adding the values of BX and DX i.e, the higher byte of
                        data
MOV [BP] , AX           ; Moving the lower byte of the result to 33940 and 33941
                        memory Location
MOV [BP+2] , BX         ; Moving the lower byte of the result to 33940 and 33942
                        memory Location
HLT
```

OUTPUT:

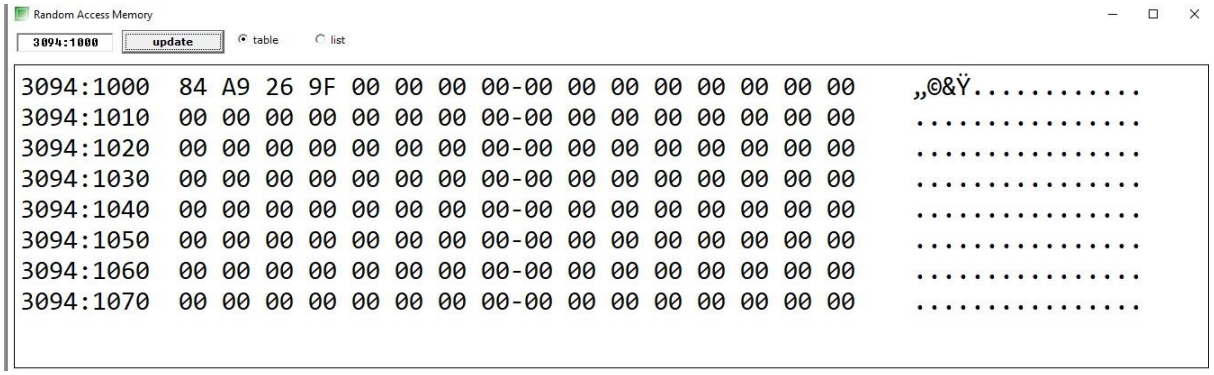
Values stored in memory location (Values for Addition)

[31940] = 84H

[31941] = A9H

[31942] = 26H

[31943] = 9FH



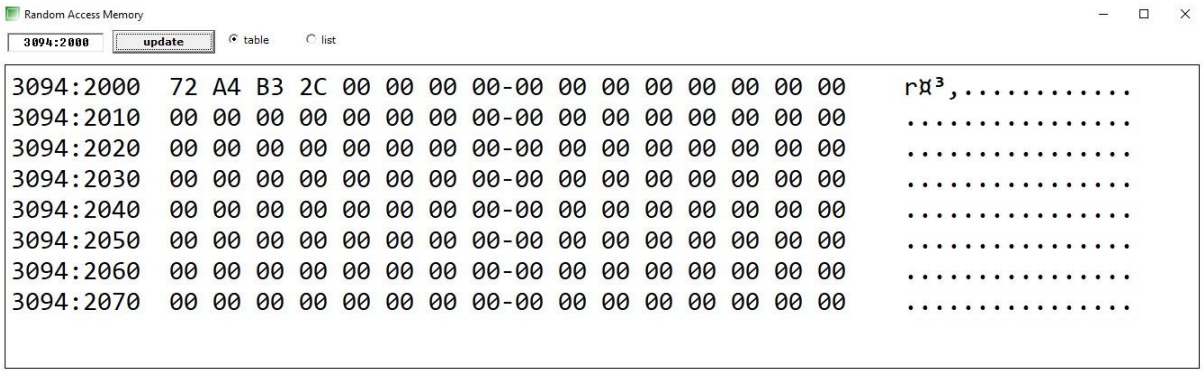
Values stored in memory location (Values for Addition)

[32940] = 72H

[32941] = A4H

[32942] = B3H

[32943] = 2CH



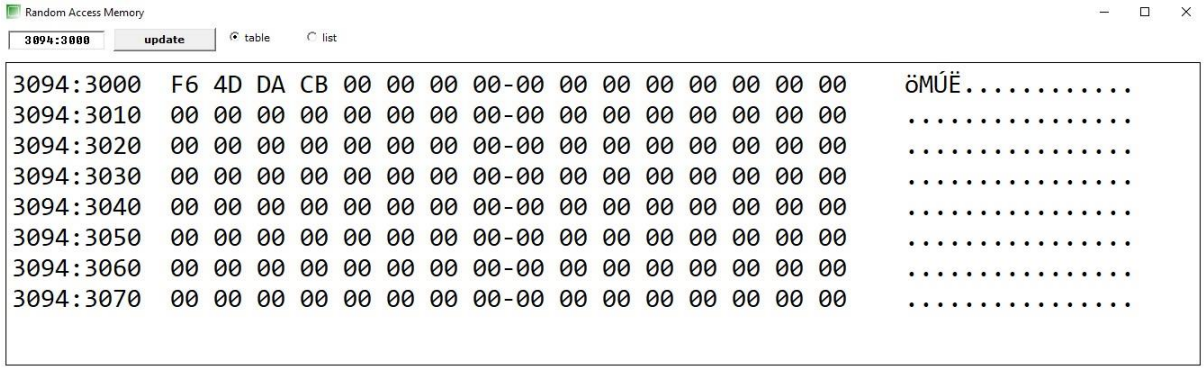
Values stored in memory location after execution (Addition has been performed)

[33940] = F6H

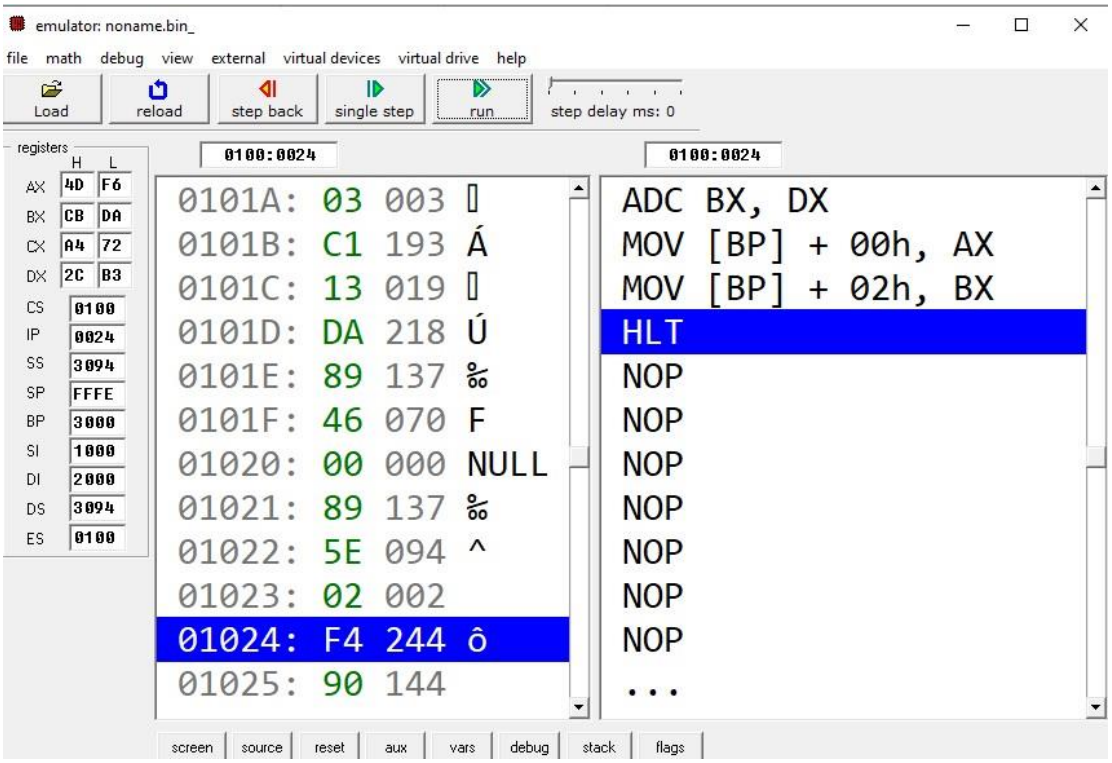
[33941] = 4DH

[33942] = DAH

[33943] = CBH



Emulator window after execution



- <u>SUBSTRACTION OF 8-BIT WITHOUT BORROW:</u>

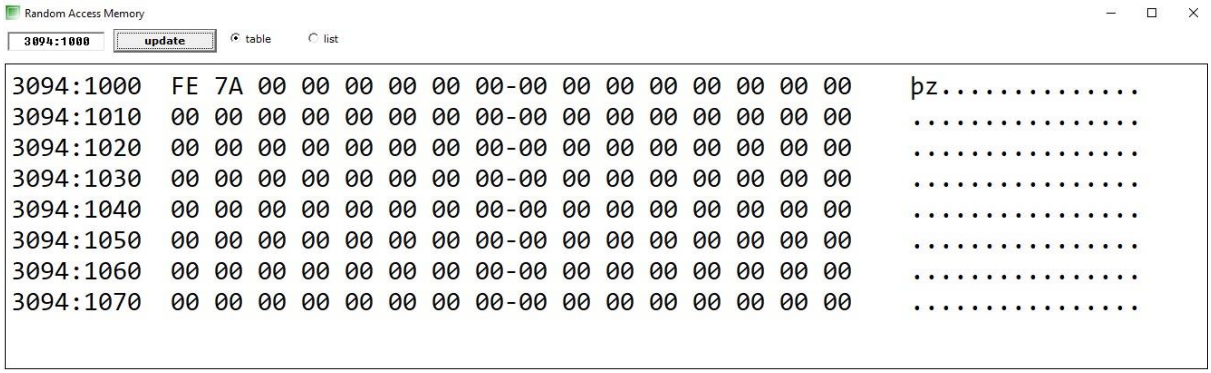
```
MOV AX, 3094H   ; Storing 3094H in AX Register
MOV DS, AX      ; Initializing DS with the help of AX
MOV SI, 1000H   ; Storing the offset value in SI as 1000H(Assumed)
MOV AL, [SI]    ; Moving the content of 31940 Memory Location to AL
MOV BL, [SI+1]  ; Moving the content of 31941 Memory Location to BL
SUB AL, BL      ; Subtract BL from AL
MOV [SI+2], AL  ; Storing the value of the result in 31942 Memory Location HLT
```

<u>OUTPUT:</u>

Values in memory location (Values to be subtracted)

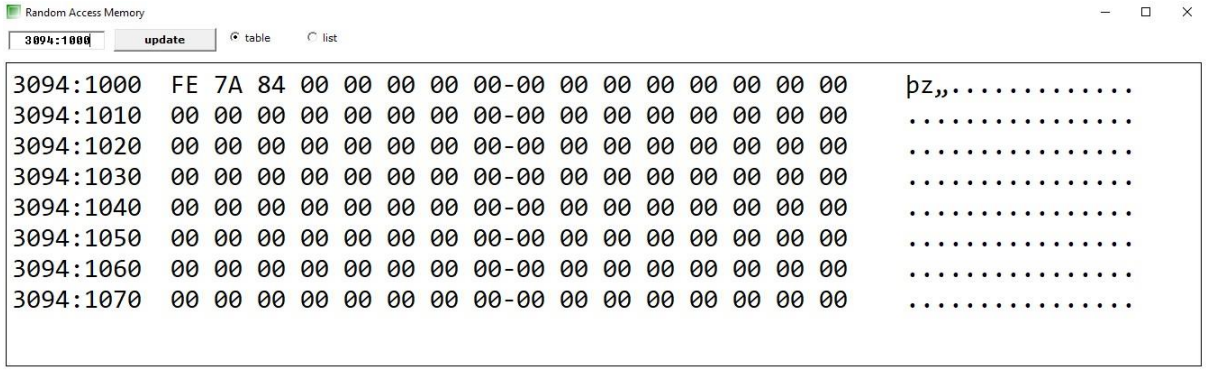
[31940] = FEH

[31941] = 7AH

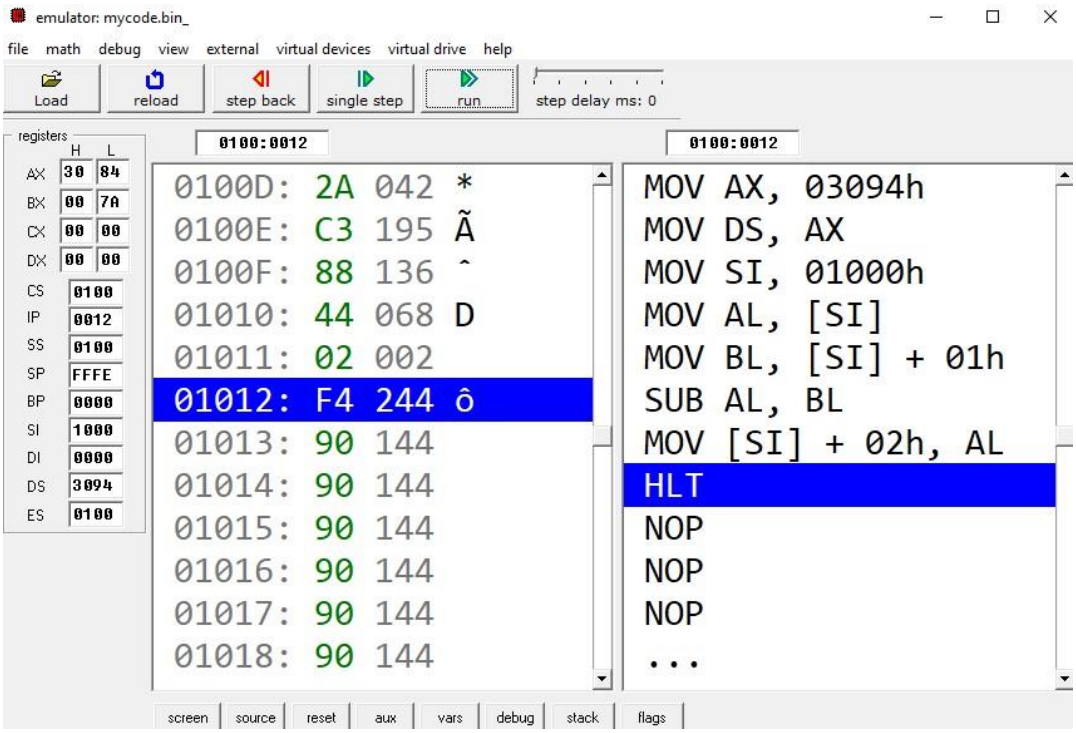


<u>After running the program the result is</u>

[31942] = 84H (Result of FEH – 7AH)



The emulator result window with updated values of all the registers is shown as



- SUBSTRACTION OF 16-BIT WITHOUT BORROW:

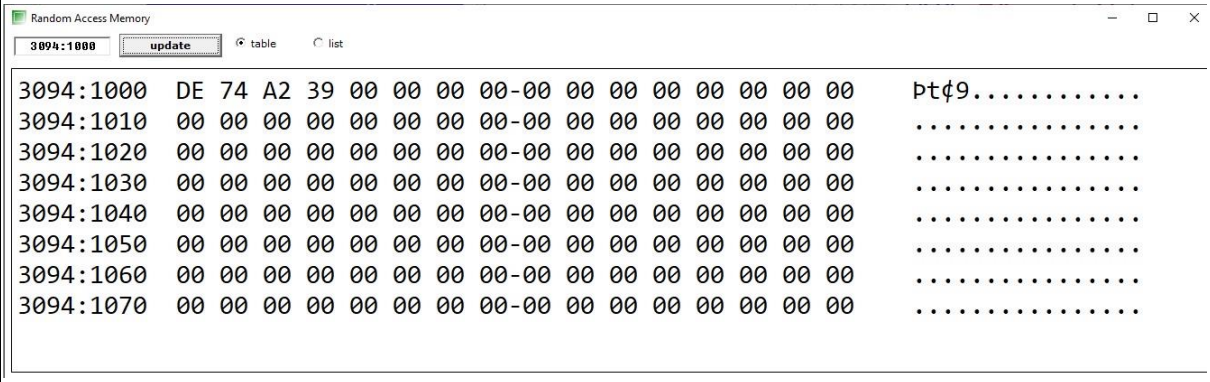
```
MOV AX, 3094H    ; Storing 3094H in AX Register
MOV DS, AX       ; Initializing DS with the help of AX
MOV SI, 1000H    ; Storing the offset value in SI as 1000H(Assumed)
MOV AX, [SI]     ; Moving the content of 31940 and 31941 Memory Location to
                   AL and AH resp.
MOV BX, [SI+2]   ; Moving the content of 31942 and 31943 Memory Location to
                   BL and BH resp.
SUB AX, BX       ; Subtract BX from AX
MOV [SI+4], AX   ; Storing the value of the result in 31944 and 31945 Memory
                   Location
HLT
```

OUTPUT:

Values in memory location (Values to be subtracted)

[31940] = DEH , [31941] = 74H

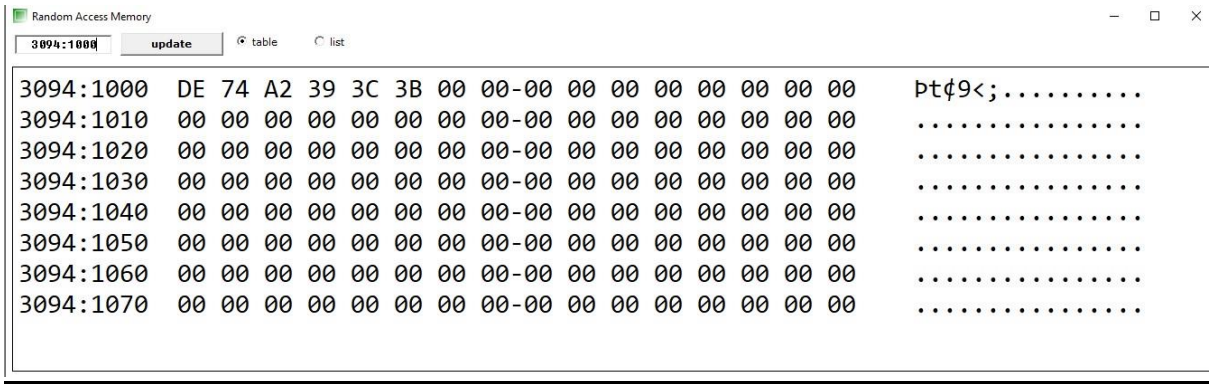
[31942] = A2H , [31943] = 39H



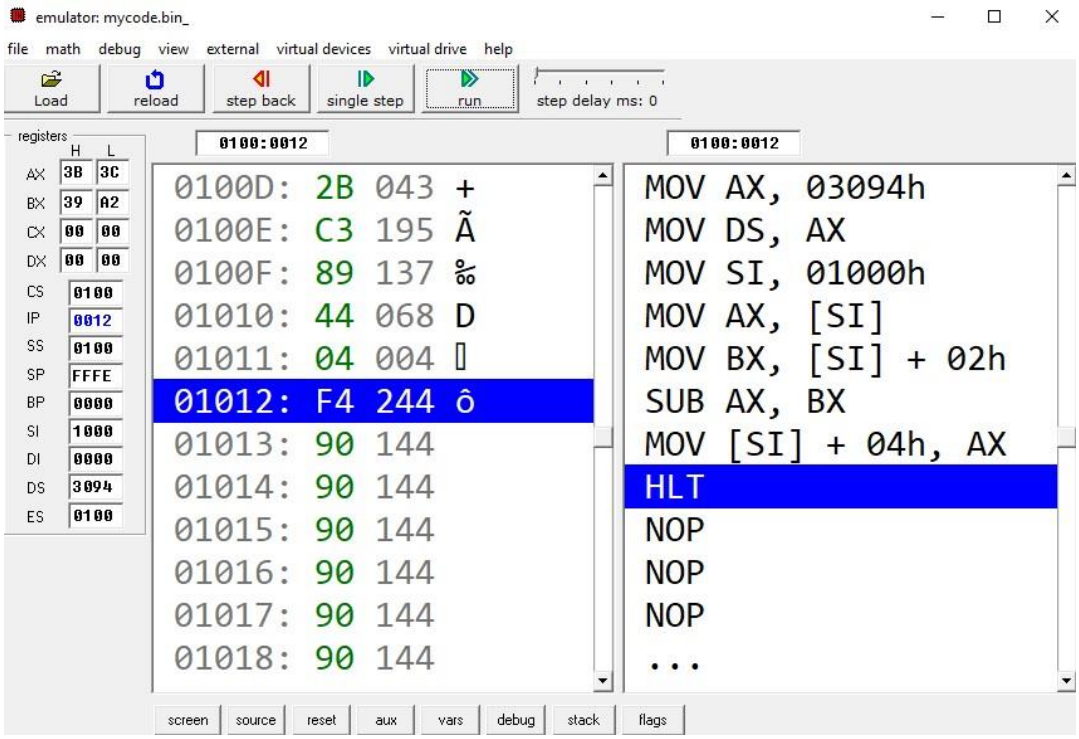
After running the program the result is

[31944] = 3CH (Result of DEH – A2H)

[31945] = 3BH (Result of 74H – 39H)



The emulator result window with updated values of all the registers is shown as



- <u>SUBSTRACTION OF 32-BIT:</u>

```
MOV AX , 3094H        ; Storing 3094H in AX Register
MOV DS , AX           ; Initializing DS with the help of AX
MOV SS , AX           ; Initializing SS with the help of AX
MOV SI , 1000H        ; Storing 1000H in SI
MOV DI , 2000H        ; Storing 2000H in DI
MOV BP , 3000H        ; Storing 3000H in BP
```

```
MOV AX , [SI]          ; Storing the content of 31940 and 31941 memory location to AX
MOV BX , [SI+2]        ; Storing the content of 31942 and 31943 memory location to BX
MOV CX , [DI]          ; Storing the content of 32940 and 32941 memory location to CX
MOV DX , [DI+2]        ; Storing the content of 31942 and 31943 memory location to DX SUB
AX , CX                ; Subtracting the values of CX from AX i.e, the lower byte of the
                         data
SBB BX , DX            ; Subtracting the values of DX from BX i.e, the higher byte of
                         data
MOV [BP] , AX          ; Moving the lower byte of the result to 33940 and 33941
                         memory Location
MOV [BP+2] , BX        ; Moving the higher byte of the result to 33942 and 33943
                          memory Location
HLT
```

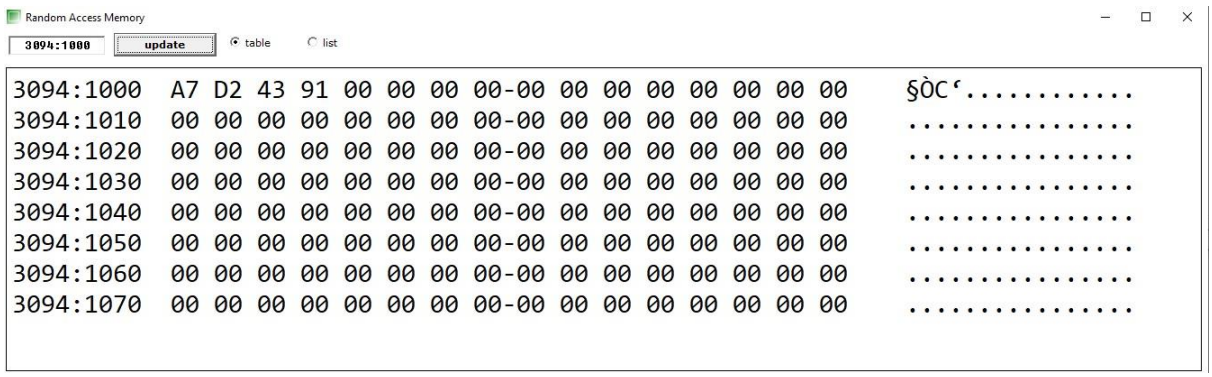
OUTPUT:

Values stored in memory location (Values for Subtraction)

[31940] = A7H

[31941] = D2H

[31942] = 43H [31943] = 91H



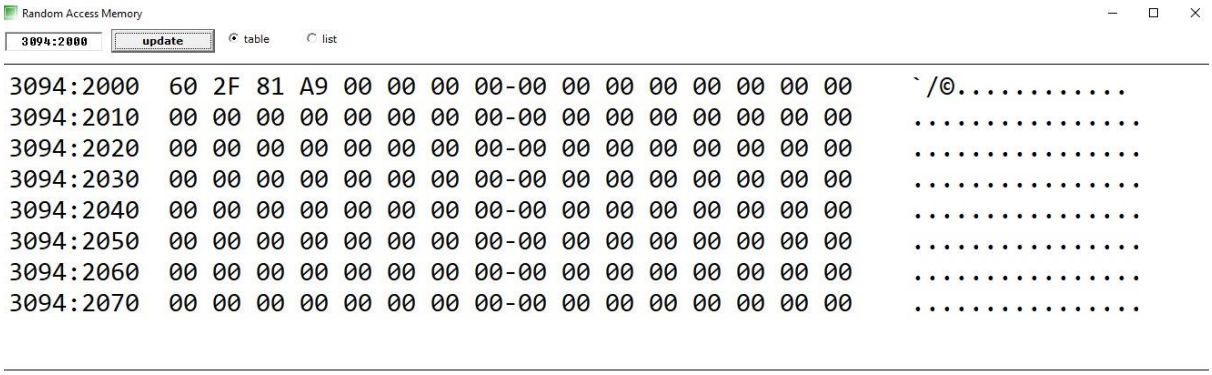
Values stored in memory location (Values for Subtraction)

[32940] = 61H

[32941] = 2FH

[32942] = 81H

[32943] = A9H



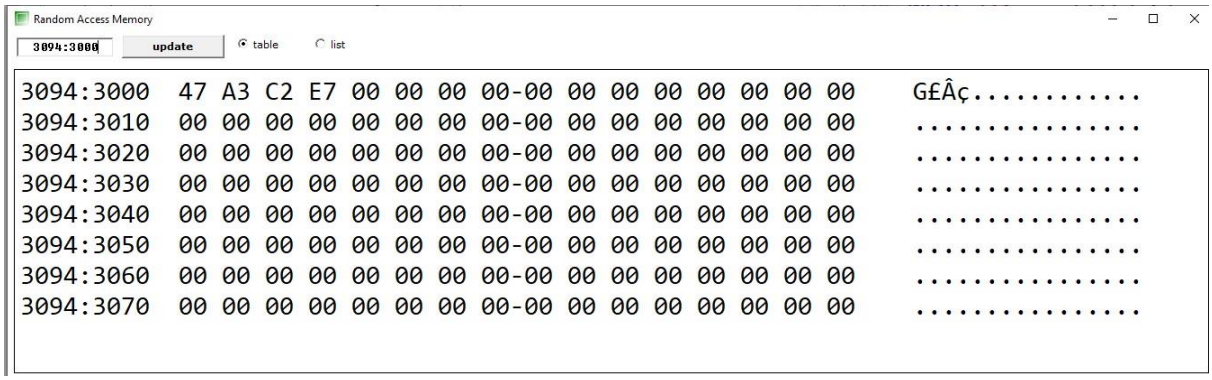
Values stored in memory location after execution

[33940] = 47H

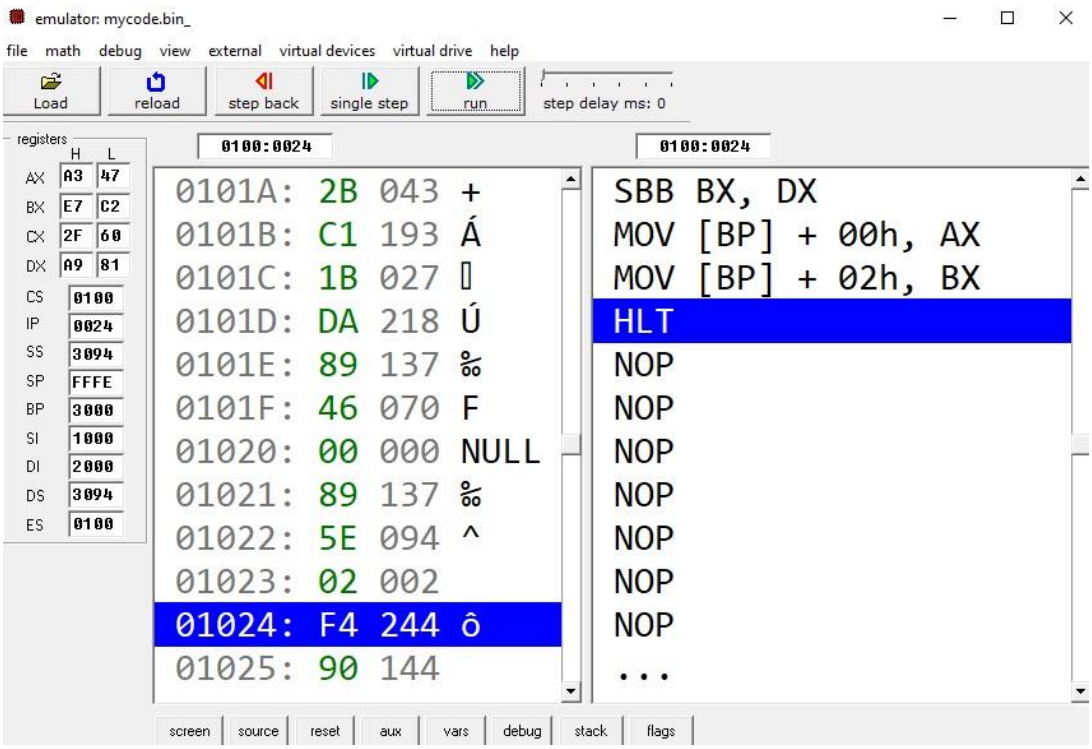
[33941] = A3H

[33942] = C2H

[33943] = E7H



The emulator screen after the program has been executed



- MULTIPLICATION OF 8-BIT :

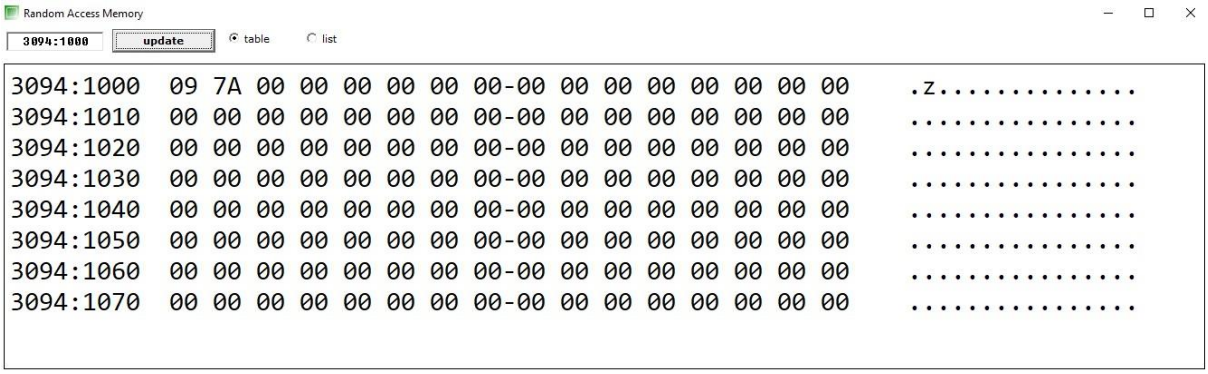
```
MOV AX , 3094H      ; Storing 3094H in AX Register
MOV DS , AX         ; Initializing DS with the help of AX
MOV SI , 1000H      ; Setting the value of SI to 1000H
MOV DI , 2000H      ; Setting the value of DI to 2000H
MOV AL , [SI]       ; Move the content of 31940 memory location to AL
MOV BL , [SI+1]     ; Move the content of 31941 memory location to BL
MUL BL              ; Multiply values of AL and BL
MOV [DI] , AX       ; Store the result in 32940 and 32941 memory location HLT
```

OUTPUT:

Values stored in memory location (Values to be multiplied)

[31940] = 09H

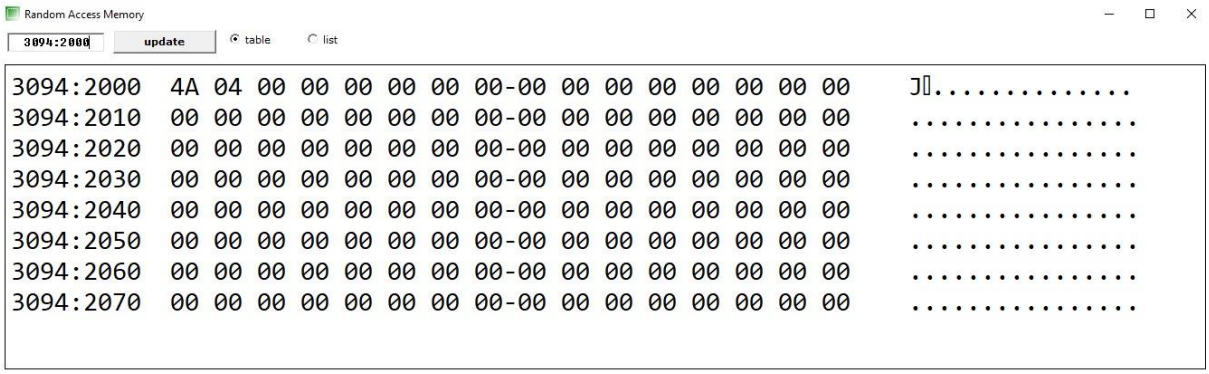
[31941] = 7AH



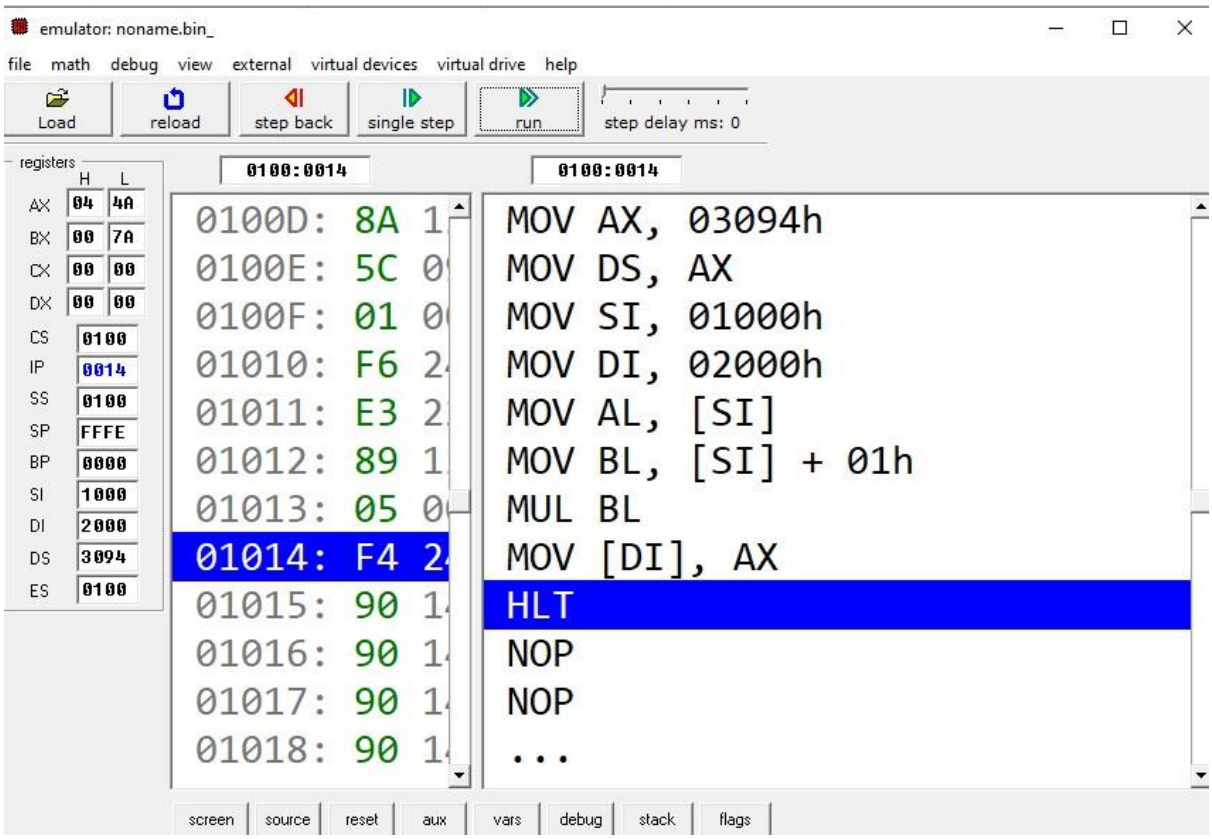
Values after execution (Result of Multiplication of the given values)

[32940] = 4AH

[32941] = 04H



Emulator window after execution of the program



- MULTIPLICATION OF 32-BIT :

```
MOV AX , 3094H        ; Storing 3094H in AX Register
MOV DS , AX           ; Initializing DS with the help of AX
MOV SI , 1000H        ; Setting the value of SI to 1000H
MOV DI , 2000H        ; Setting the value of DI to 2000H
MOV AX , [SI]         ; Move the content of 31940 and 31941 memory location to
                        AL
MOV BX , [SI+2]       ; Move the content of 31942 and 31943 memory location to
                        BL
IMUL BX               ; Multiply values of AL and BL
MOV [DI] , AX         ; Store the lower byte of result in 32940 and 32941 memory
                        location
MOV [DI+2] , DX       ; Store the higher byte of result in 32942 and 32943 memory
                        location
HLT
```

OUTPUT:

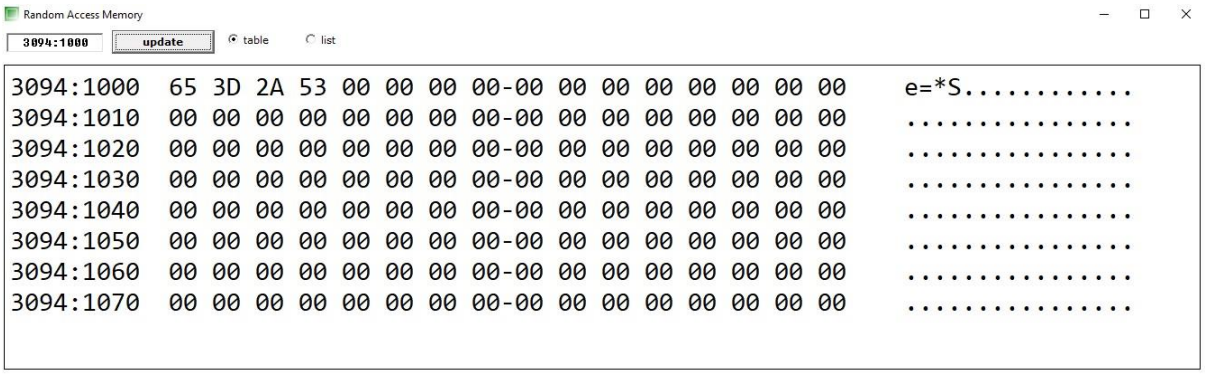
Values stored in memory location (Values to be Multiplied)

[31940] = 65H

[31941] = 3DH

[31942] = 2AH

[31943] = 53H



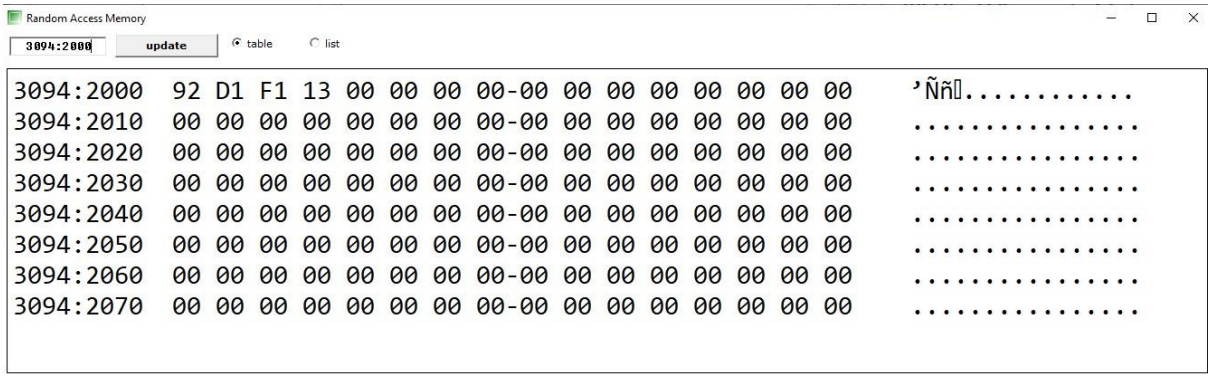
Values after execution (Result of multiplication of the given values)

[32940] = 92H

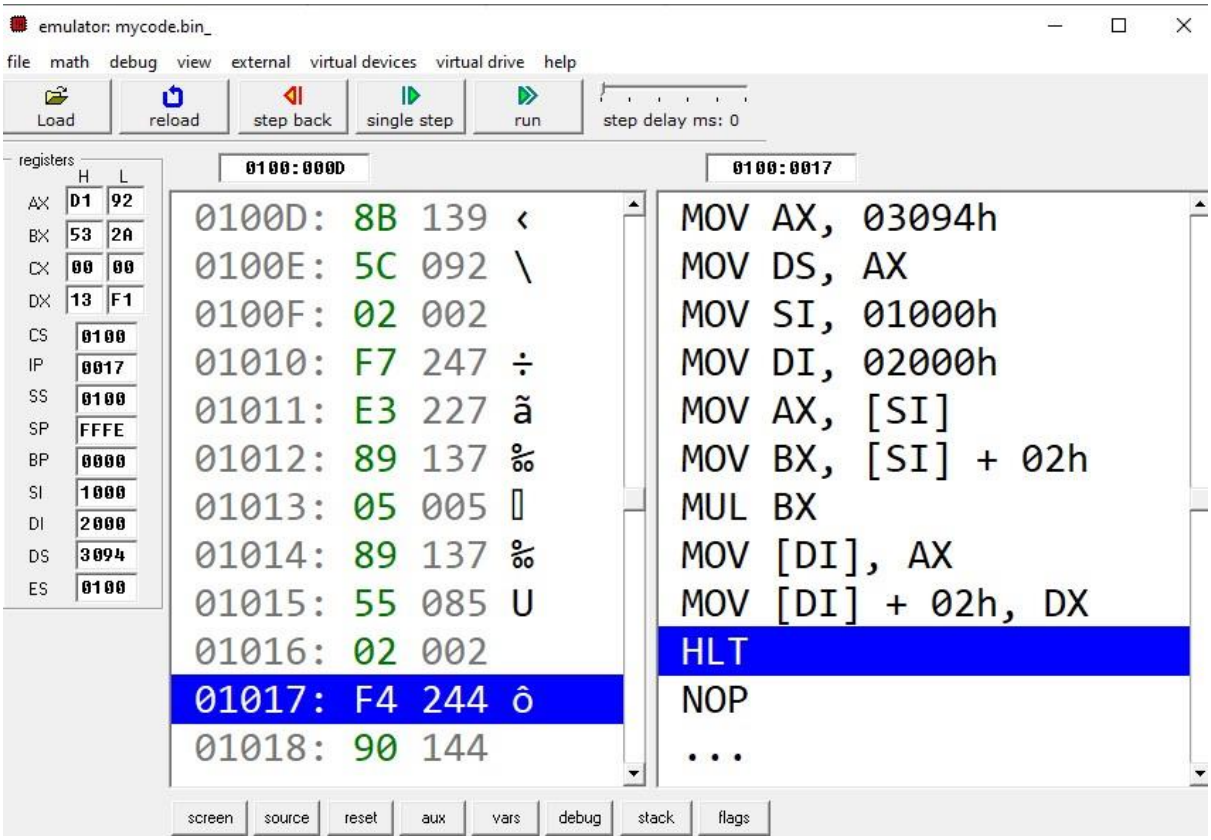
[32941] = D1H

[32942] = F1H

[32943] = 13H



Emulator window after execution of the program



- DIVISION OF 8/16-BIT :

```
MOV AX , 3094H      ; Storing 3094H in AX Register
MOV DS , AX         ; Initializing DS with the help of AX
MOV SI , 1000H      ; Setting the value of SI to 1000H
MOV DI , 2000H      ; Setting the value of DI to 2000H
MOV AX , [SI]       ; Move the content of 31940 and 31941 memory location to AX
MOV BL , [SI+2]     ; Move the content of 31942 memory location to BL
DIV BL              ; Divide values of AX and BL
MOV [DI] , AX       ; Store the result i.e, quotient in 32940 and remainder in 32941
                      memory location
HLT
```

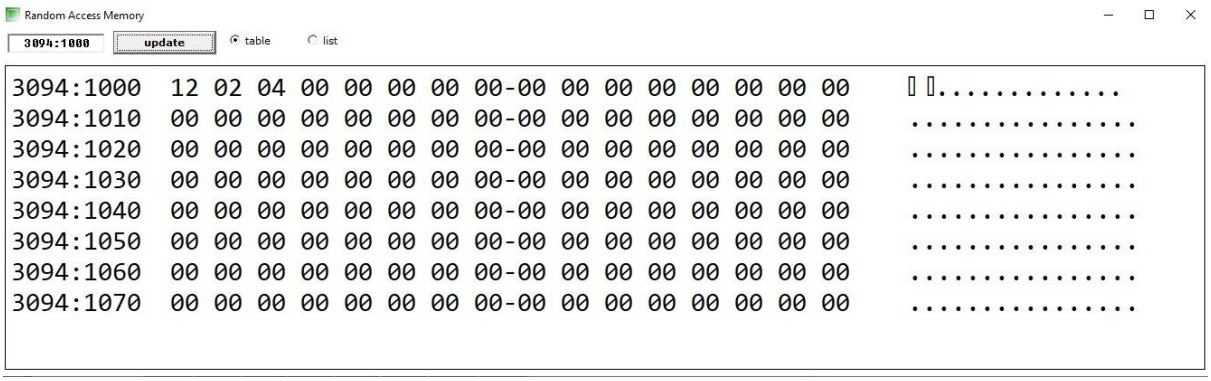
OUTPUT:

Values stored in memory location (Values to be divided)

[31940] = 12H (Dividend Lower bit)

[31941] = 02H (Dividend Higher bit)

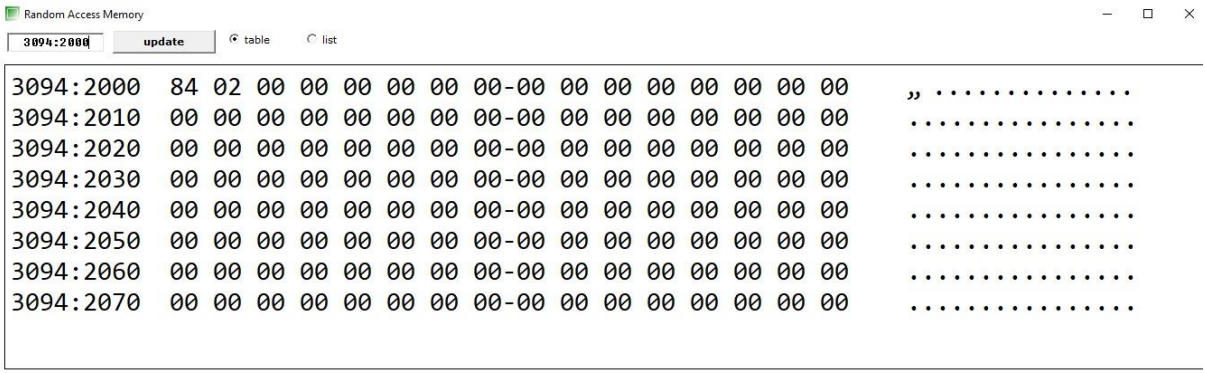
[31942] = 04H (Divisor)



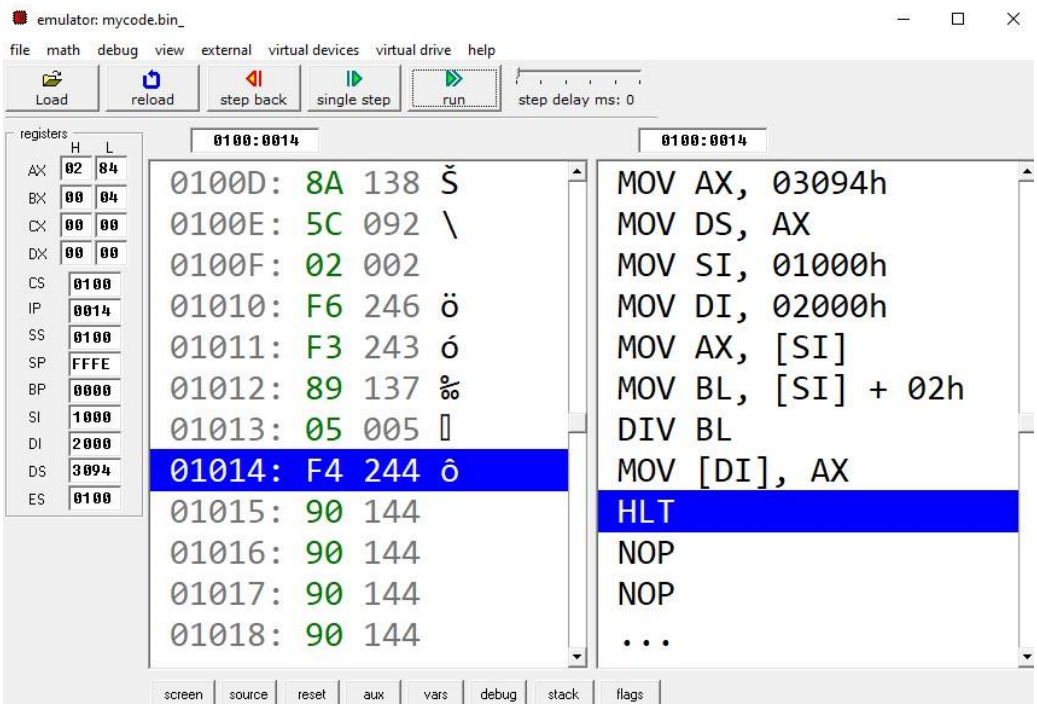
Values stored in memory location

[32940] = 84H (Quotient)

[32941] = 02H (Remainder)



Emulator window after execution of the program



- DIVISION OF 16/32-BIT :

```
MOV AX , 3094H     ; Storing 3094 in AX Register
MOV DS , AX        ; Initializing DS with the help of AX
MOV SI , 1000H     ; Setting the value of SI to 1000H
MOV DI , 2000H     ; Setting the value of DI to 2000H
MOV AX , [SI]      ; Move the content of 31940 and 31941 memory location to
                     AX
MOV DX , [SI+2]    ; Move the content of 31942 and 31943 memory location to
                     BL
MOV BX , [SI+4]    ; Move the content of 31944 and 31945 memory location to
                     BL
IDIV BX            ; Divide the 32 bit data values of AX and DX  BX
MOV [DI] , DX      ; Store the result i.e, quotient in 32940 and 32941 memory
location
MOV [DI+2] , AX    ; Store the remainder in 32942 and 32943 memory location HLT
```

OUTPUT:

Values stored in memory location (Values to be Divided)

[31940] = 62H

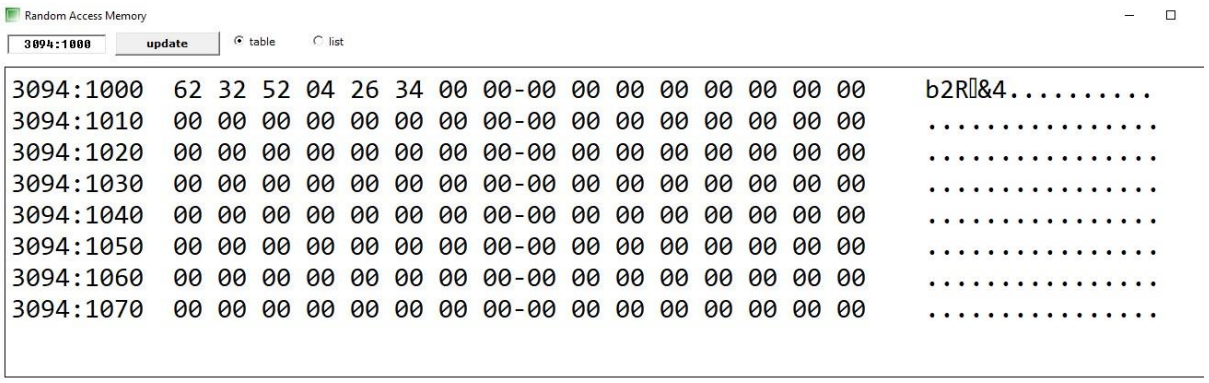
[31941] = 32H

[31942] = 52H

[31943] = 04H

[31944] = 26H

[31945] = 34H

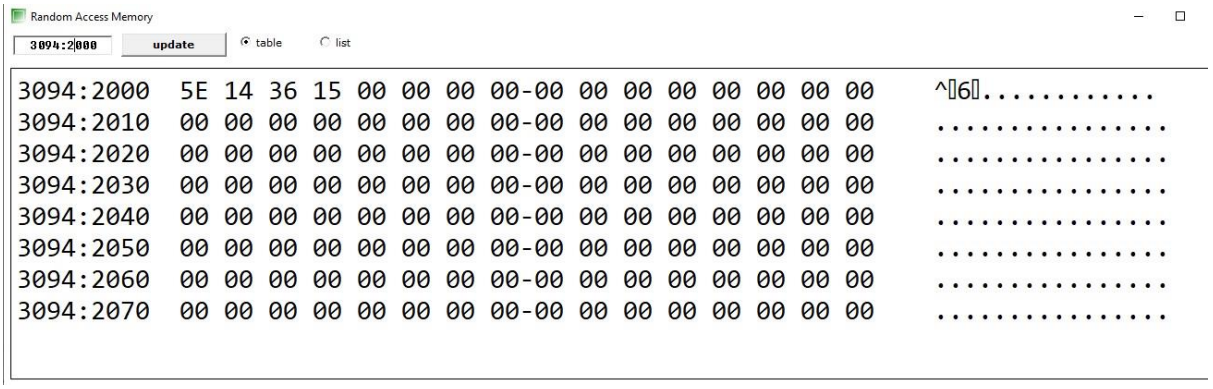


Values after execution (Result of DIVISION of the given values)

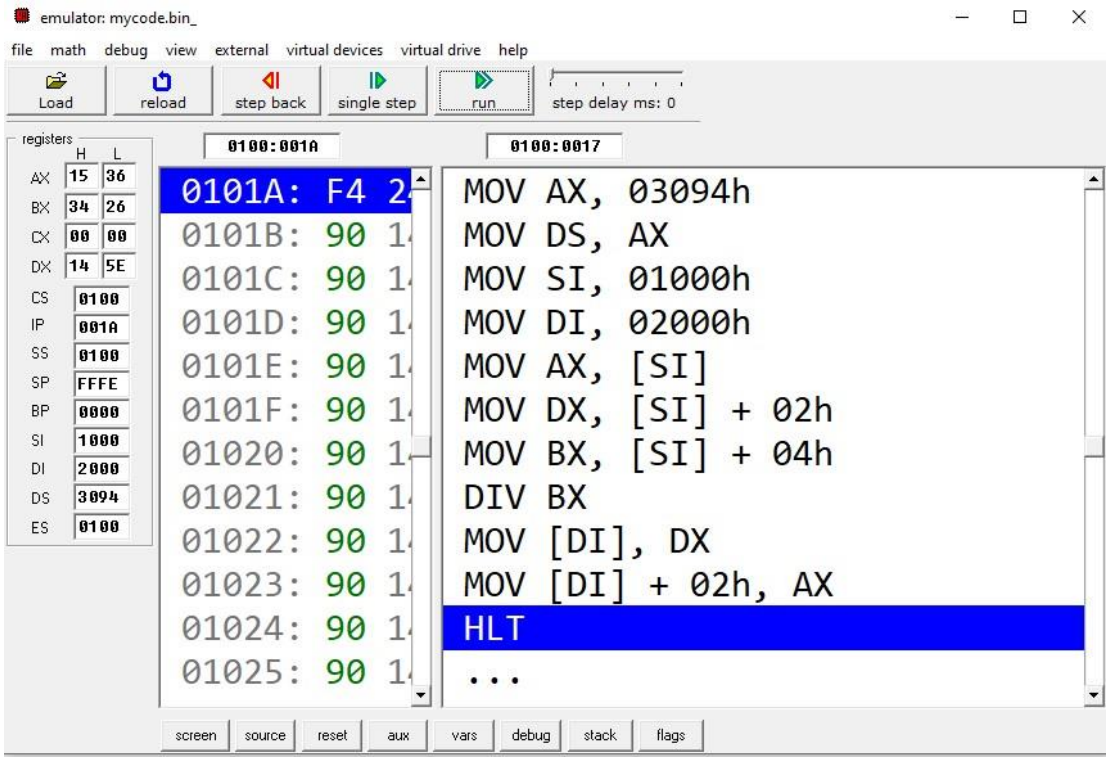
[32940] = 5EH

[32941] = 14H

[32942] = 36H [32943] = 15H



Emulator window after execution of the program


emulator: mycode.bin_
file math debug view external virtual devices virtual drive help
Load
reload
step back
single step
run
step delay ms: 0
registers
H L
AX 15 36
BX 34 26
CX 00 00
DX 14 5E
CS 0100
IP 001A
SS 0100
SP FFFE
BP 0000
SI 1000
DI 2000
DS 3094
ES 0100
0100:001A
0100:0017
0101A: F4 2
0101B: 90 1
0101C: 90 1
0101D: 90 1
0101E: 90 1
0101F: 90 1
01020: 90 1
01021: 90 1
01022: 90 1
01023: 90 1
01024: 90 1
01025: 90 1
MOV AX, 03094h
MOV DS, AX
MOV SI, 01000h
MOV DI, 02000h
MOV AX, [SI]
MOV DX, [SI] + 02h
MOV BX, [SI] + 04h
DIV BX
MOV [DI], DX
MOV [DI] + 02h, AX
HLT
...
screen source reset aux vars debug stack flags

Experiment 4

AIM:

- **Program for logical operation (XOR, OR, AND, NOT) and comparison of 8/16-bit number.**

CODE:

- **XOR:** org

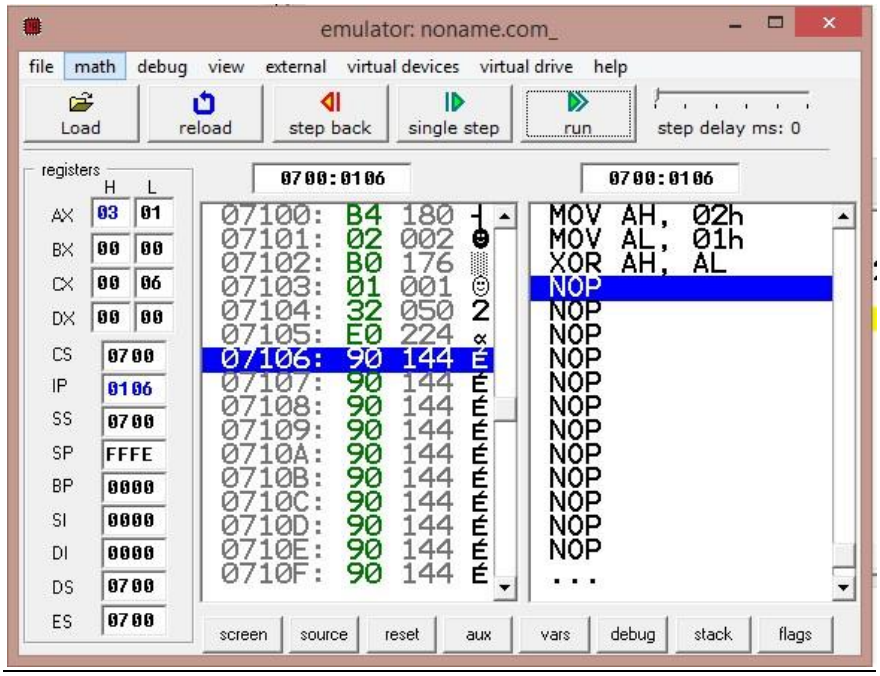
100h

MOV AH, 02H

MOV AL, 01H

XOR AH,AL

OUTPUT



- **OR:** org

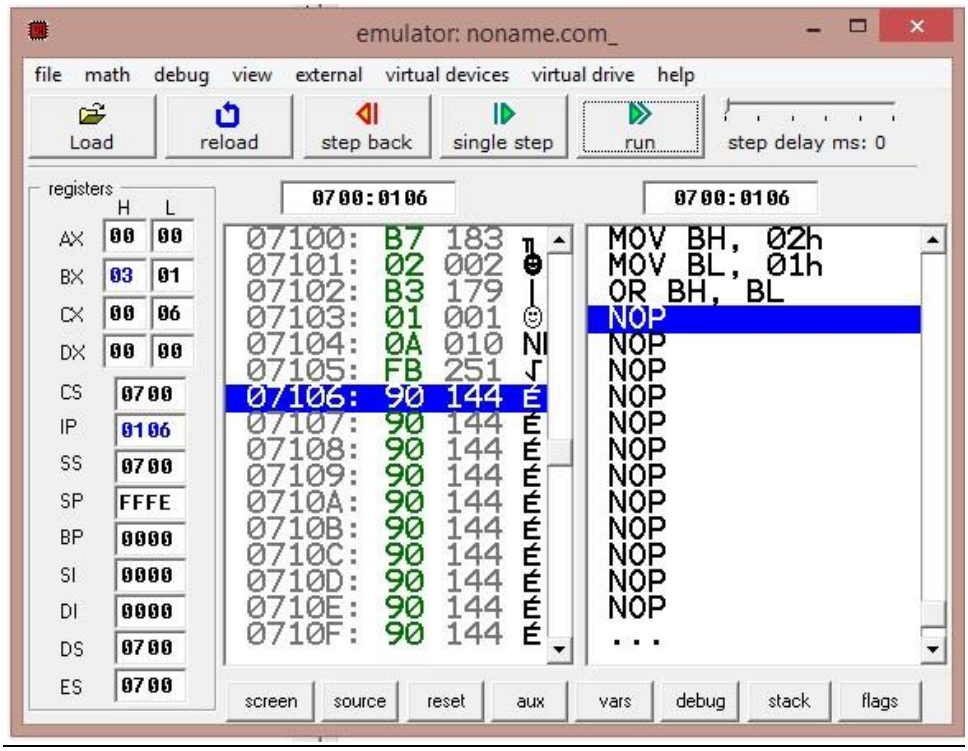
100h

MOV BH, 02H

MOV BL, 01H

OR BH,BL

OUTPUT:



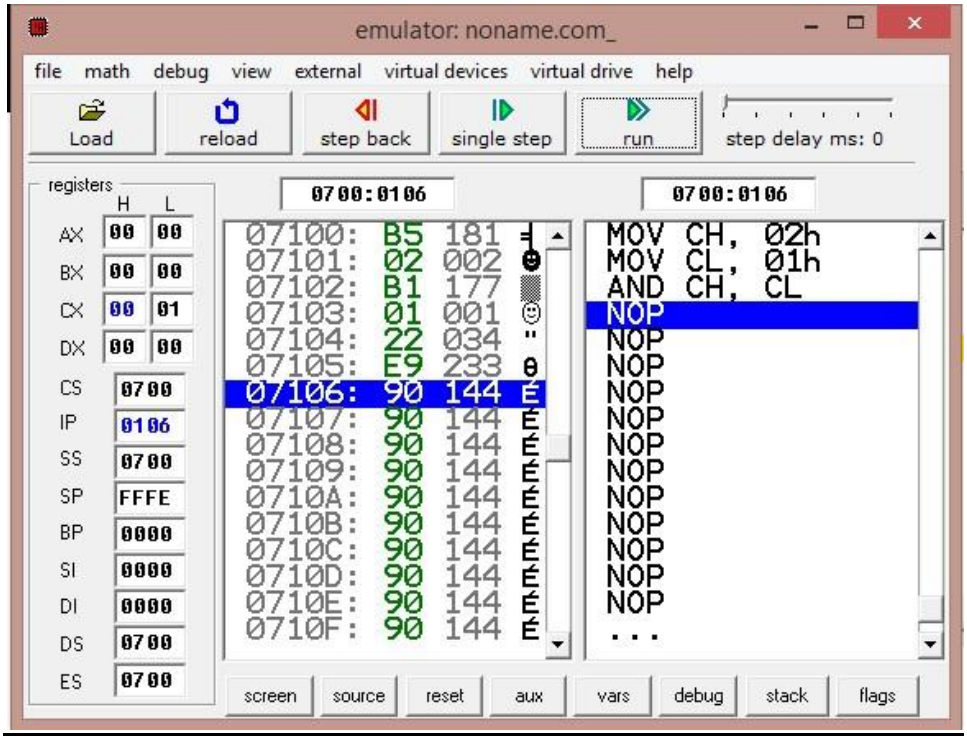
- **AND:** org

100h

MOV CH, 02H

MOV CL, 01H

AND CH,CL **OUTPUT:**



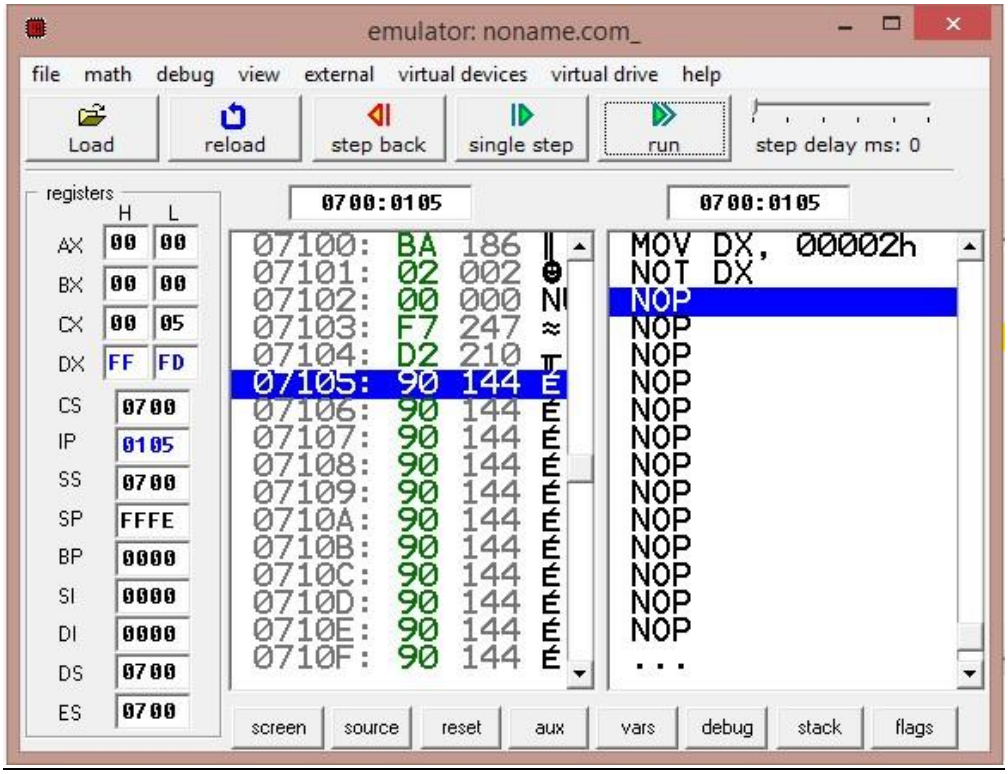
NOT: org

100h

MOV DX, 02H

NOT DX

OUTPUT:



AIM:

- **Program for logical operation (XOR, OR, AND, NOT) and comparison of 8/16-bit number.**

FINDING THE SMALLEST DATA FROM THE GIVEN DATA

CODE:

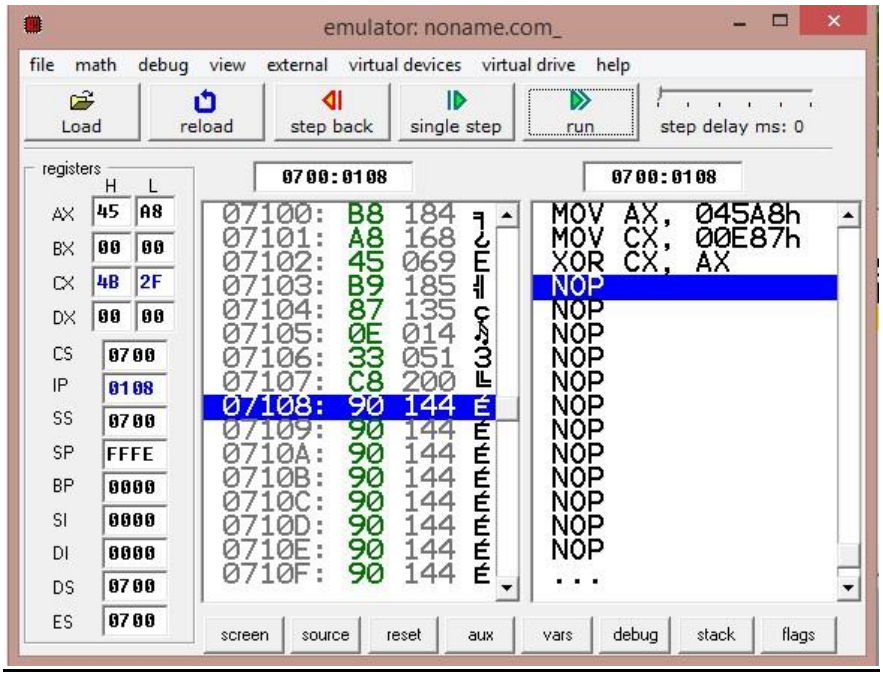
- **XOR:** org

100h

MOV AX, 45A8H

MOV CX, 0E87H

XOR CX,AX **OUTPUT:**



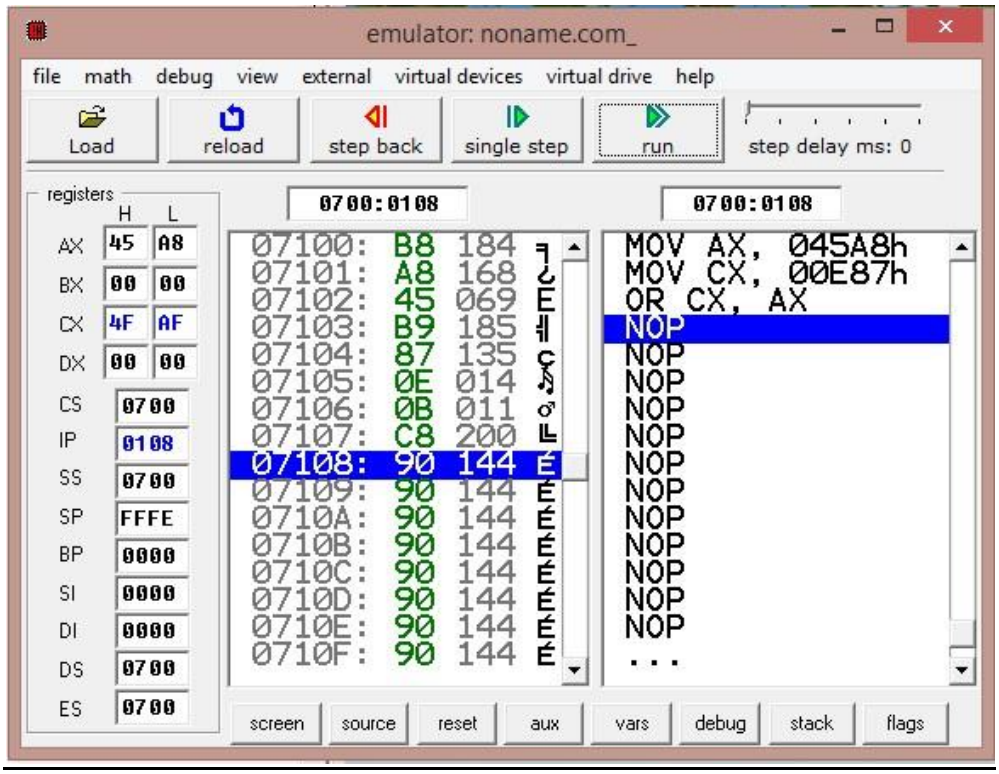
- **OR:** org

100h

MOV AX, 45A8H MOV

CX, 0E87H

OR CX,AX **OUTPUT:**



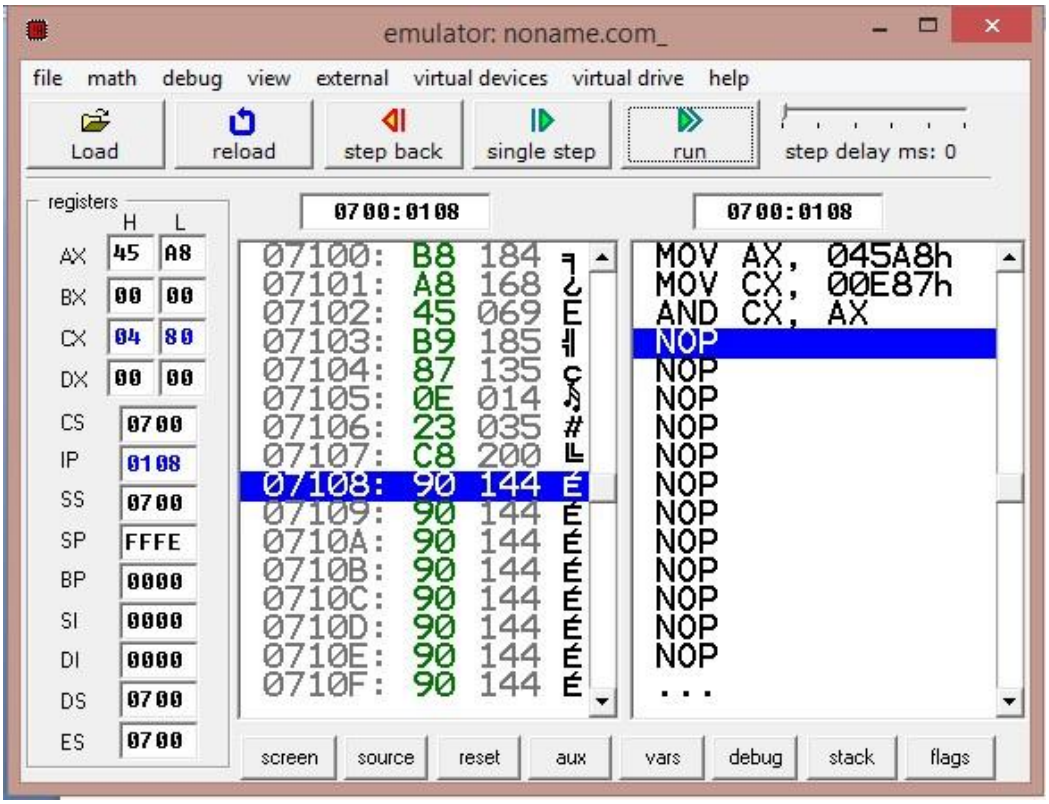
- **AND:** org

100h MOV AX,

45A8H

MOV CX, 0E87H

AND CX,AX **OUTPUT:**



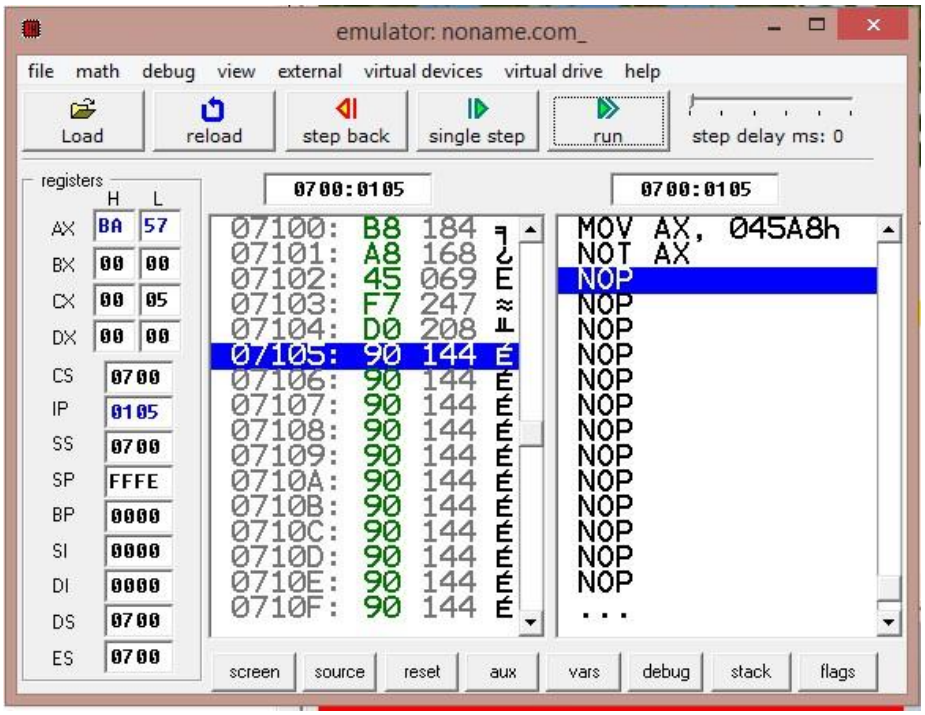
- **NOT:**

org 100h

MOV AX, 45A8H

NOT AX

OUTPUT:


emulator: noname.com_
file math debug view external virtual devices virtual drive help
Load
reload
step back
single step
run
step delay ms: 0
registers
H L
AX BA 57
BX 00 00
CX 00 05
DX 00 00
CS 0700
IP 0105
SS 0700
SP FFFE
BP 0000
SI 0000
DI 0000
DS 0700
ES 0700
0700:0105
0700:0105
07100: B8 184
07101: A8 168
07102: 45 069 E
07103: F7 247
07104: D0 208
07105: 90 144 É
07106: 90 144
07107: 90 144
07108: 90 144
07109: 90 144
0710A: 90 144
0710B: 90 144
0710C: 90 144
0710D: 90 144
0710E: 90 144
0710F: 90 144
MOV AX, 045A8h
NOT AX
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
screen
source
reset
aux
vars
debug
stack
flags

5

AIM: To find the minimum of N given numbers.

FINDING THE SMALLEST DATA FROM THE GIVEN DATA

CODE:

```
MOV AX , 3094H          ; Storing 3094H in AX
MOV DS,AX               ; Initializing DS with the help of AX
START: MOV SI , 1000H   ; Storing 1000H in SI
MOV DI , 2000H          ; Storing 2000H in DI
MOV CL , [SI]           ; Move the content of SI to CL register
INC SI                  ; Increment the value of SI
MOV AL , [SI]           ; Move the content of SI to AL
DEC CL                  ; Decrement the value of CL
AGAIN: INC SI           ; Increment SI
MOV BL , [SI]           ; Store the content of SI to BL
CMP AL , BL             ; Compare the content of AL and BL
JC AHEAD                ; Jump if carry set to AHEAD
MOV AL , BL             ; Move BL to AL
AHEAD: DEC CL           ; Decrement CL if carry is set
JNZ AGAIN               ; Jump if zero flag not set to AGAIN
MOV [DI],AL             ; Move the value of AL to content of DI
HLT
```

OUTPUT:

Values stored in memory location (Values to find the smallest from)

[31940] = 05H (Count)

[31941] = F4H

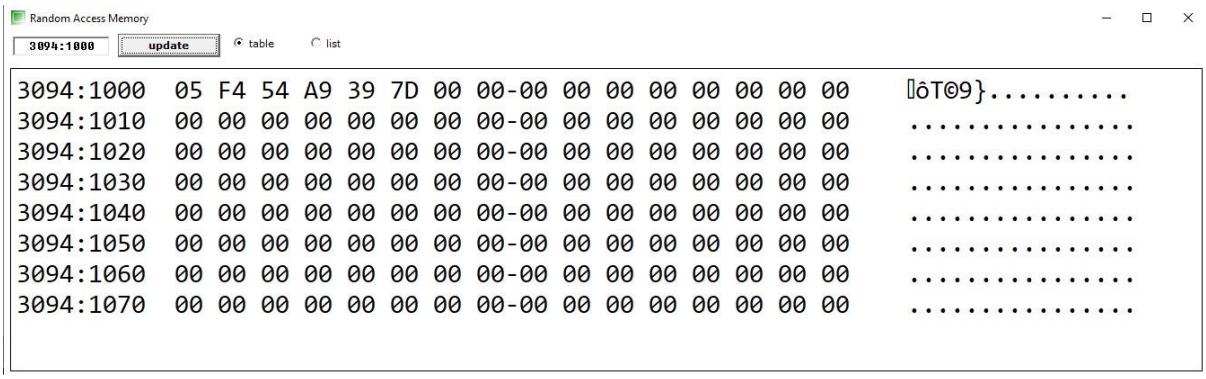
[31942] = 54H

[31943] = A9H

[31944] = 39H

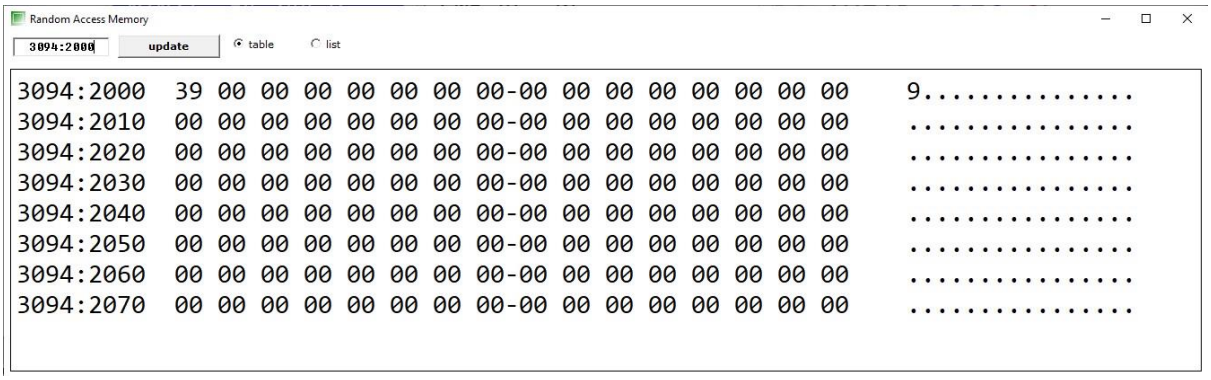
Experiment

[31945] = 7DH

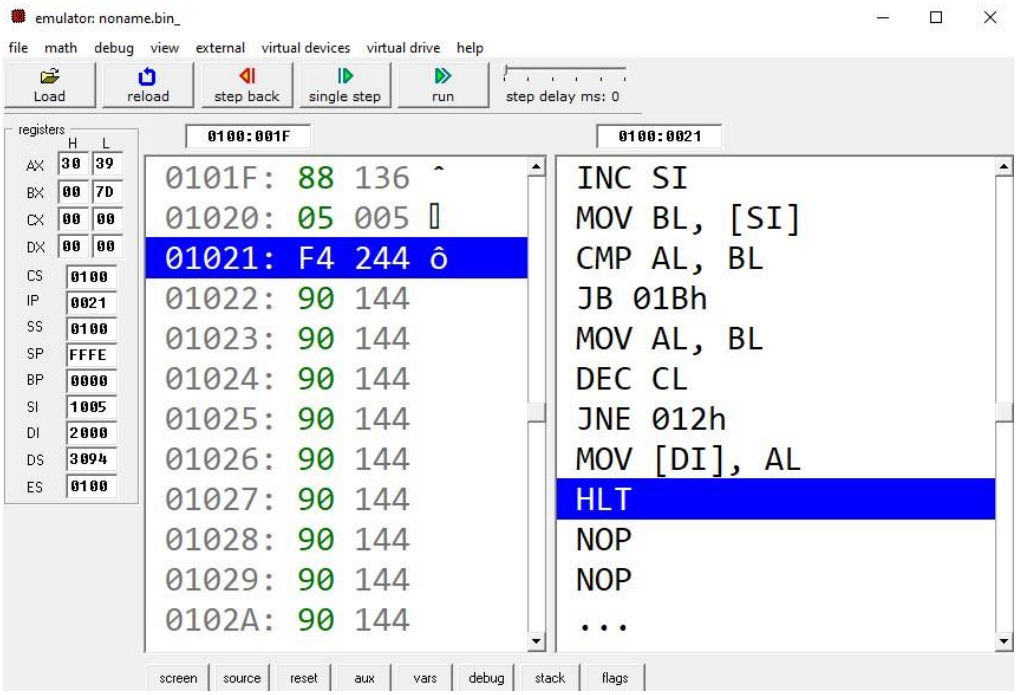


Values stored in memory location (Smallest Value) [32940]

= 39H



Emulator window after execution



6

AIM: To find the maximum of N given numbers.

FINDING THE LARGEST DATA FROM THE GIVEN DATA

CODE:

```
MOV AX , 3094H          ; Storing 3094H in AX
MOV DS,AX               ; Initializing DS with the help of AX
START: MOV SI , 1000H   ; Storing 1000H in SI
MOV DI , 2000H          ; Storing 2000H in DI
MOV CL , [SI]           ; Move the content of SI to CL register
INC SI                  ; Increment the value of SI
MOV AL , [SI]           ; Move the content of SI to AL
DEC CL                  ; Decrement the value of CL
AGAIN: INC SI           ; Increment SI
MOV BL , [SI]           ; Store the content of SI to BL
CMP AL , BL             ; Compare the content of AL and BL
JNC AHEAD               ; Jump if carry not set to AHEAD
MOV AL , BL             ; Move BL to AL
AHEAD: DEC CL           ; Decrement CL if carry is set
```

Experiment

```
JNZ AGAIN          ; Jump if zero flag not set to AGAIN
MOV [DI],AL        ; Move the value of AL to content of DI
HLT
```

OUTPUT:

Values stored in memory location (Values to find the largest from)

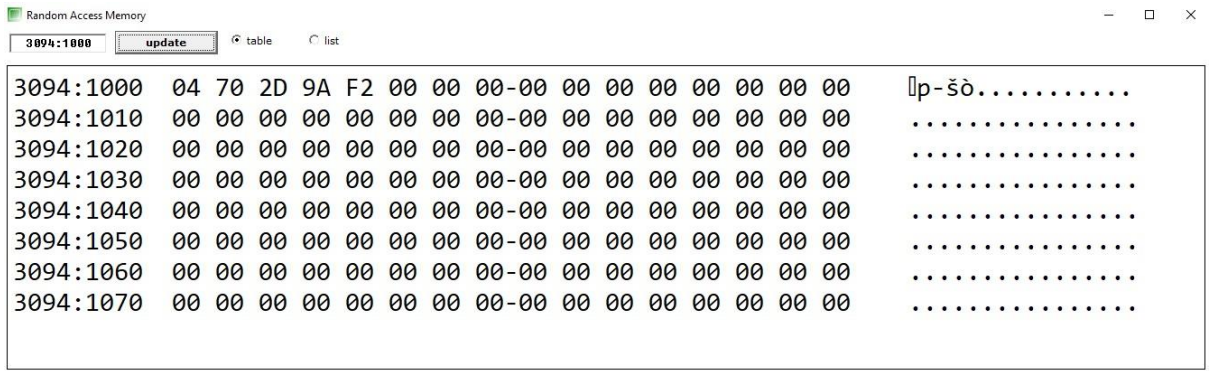
[31940] = 04H (Count)

[31941] = 70H

[31942] = 2DH

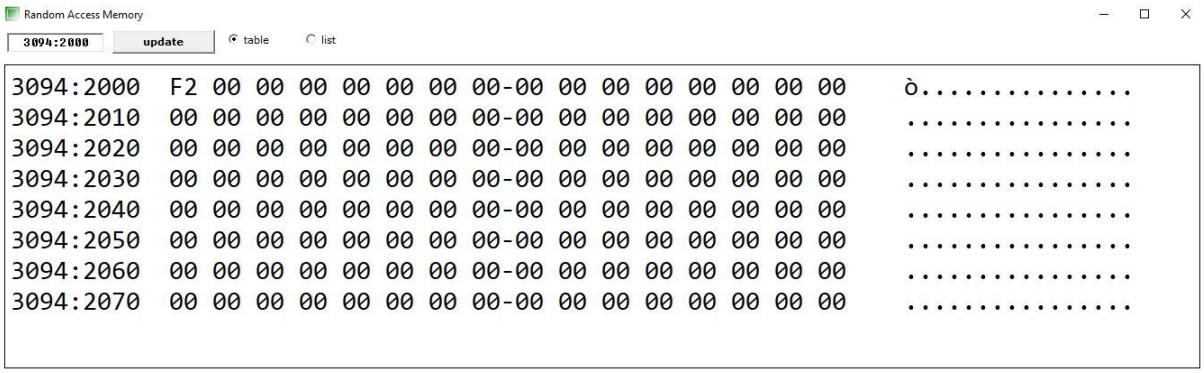
[31943] = 9AH

[31944] = F2H

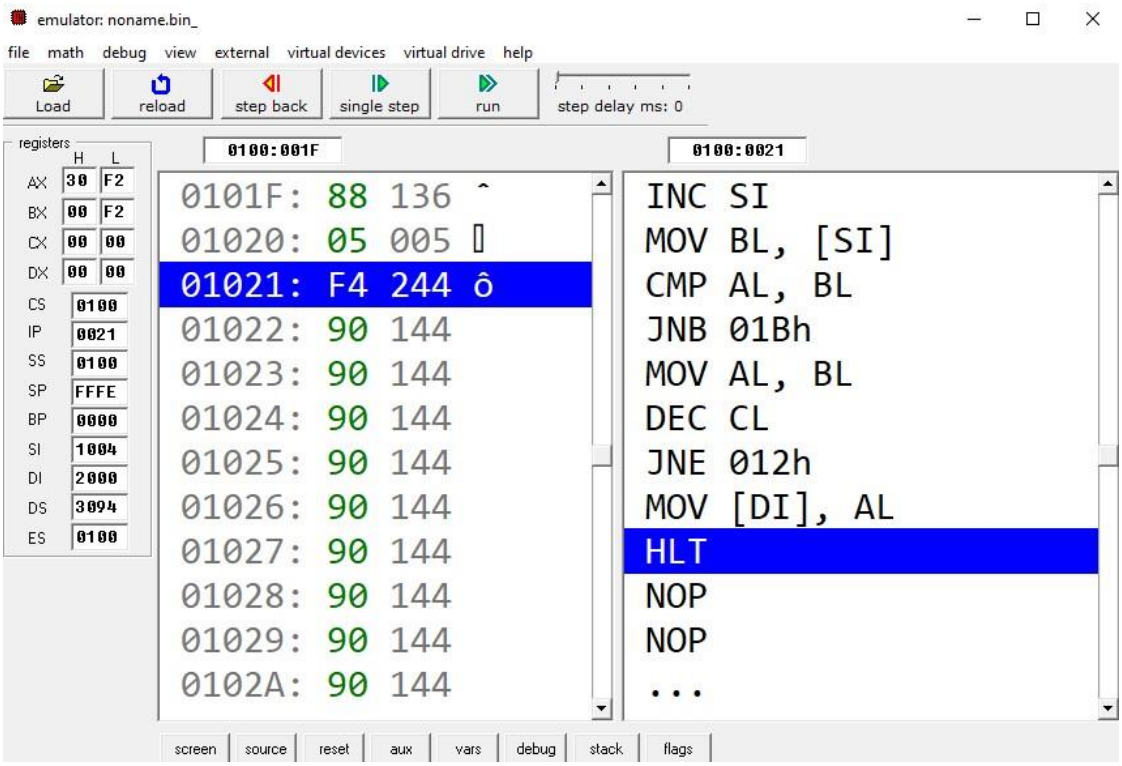


Values stored in memory location (Largest Value)

[32940] = F2H



Emulator window after execution



_____7

AIM: **To find the minimum of N given numbers**.

ARRANGING THE GIVEN DATA IN ASCENDING ORDER

CODE:

```
MOV AX , 3094H          ; Storing 3094H in AX
MOV DS , AX             ; Initialising DS with the help of AX
START: MOV SI , 1000H   ; Store 1000H in SI
MOV CL , [SI]           ; Store the content of SI to CL
DEC CL                  ; Decrement the value of CL
REPEAT: MOV SI , 1000H  ; Move 1000H to SI
MOV CH , [SI]           ; Move the content of SI to CH
DEC CH                  ; Decrement the value of CH
INC SI                  ; Increment the value of SI
REPCOM: MOV AL , [SI]   ; Get an element of Array in AL register
INC SI                  ; Increment the value of SI
CMP AL , [SI]           ; Compare the values of AL and the content of SI
```

Experiment

```
JC AHEAD                ; Jump if carry set to AHEAD
XCHG AL , [SI]          ; If AL is less than memory
XCHG AL , [SI-1]        ; Exchange the content of memory pointed by SI and its
                          previous location
AHEAD: DEC CH           ; Decrement the value of CH
JNZ REPCOM              ; Repeat comparison until count is 0
DEC CL                  ; Decrement CL
JNZ REPEAT              ; Repeat N-1 comparisons until cont is 0
HLT
```

OUTPUT:

Values stored in memory location (Values to be arranged in Ascending order)

[31940] = 06H (Count)

[31941] = 27H

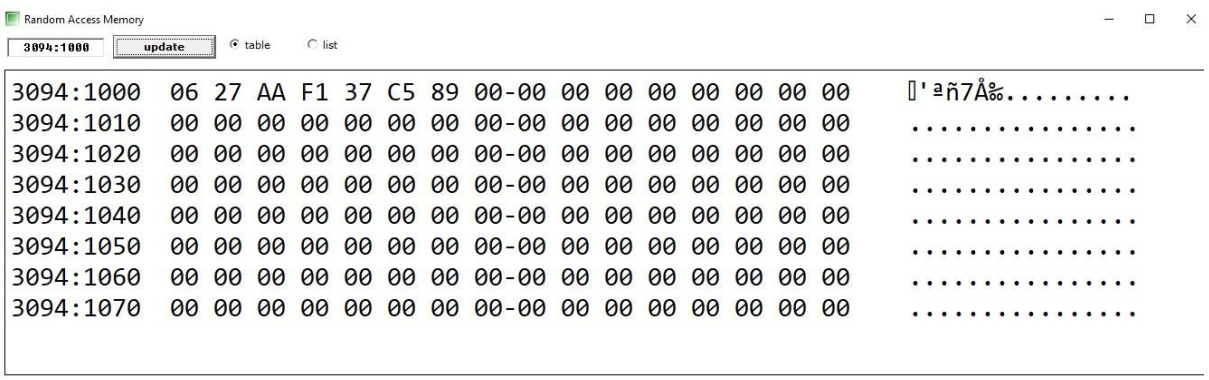
[31942] = AAH

[31943] = F1H

[31944] = 37H

[31945] = C5H

[31946] = 89H



Values stored in memory location (In ascending order)

[32940] = 06H (Count)

[32941] = 27H

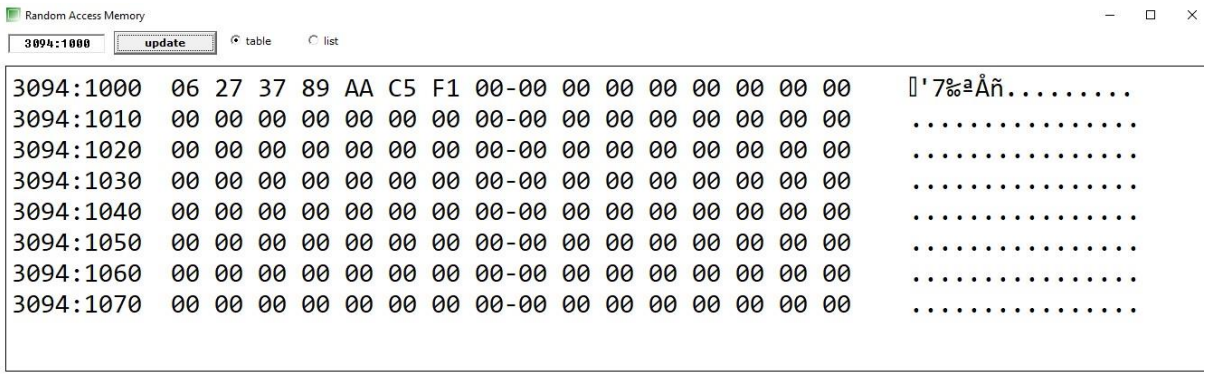
[32942] = 37H

[32943] = 89H

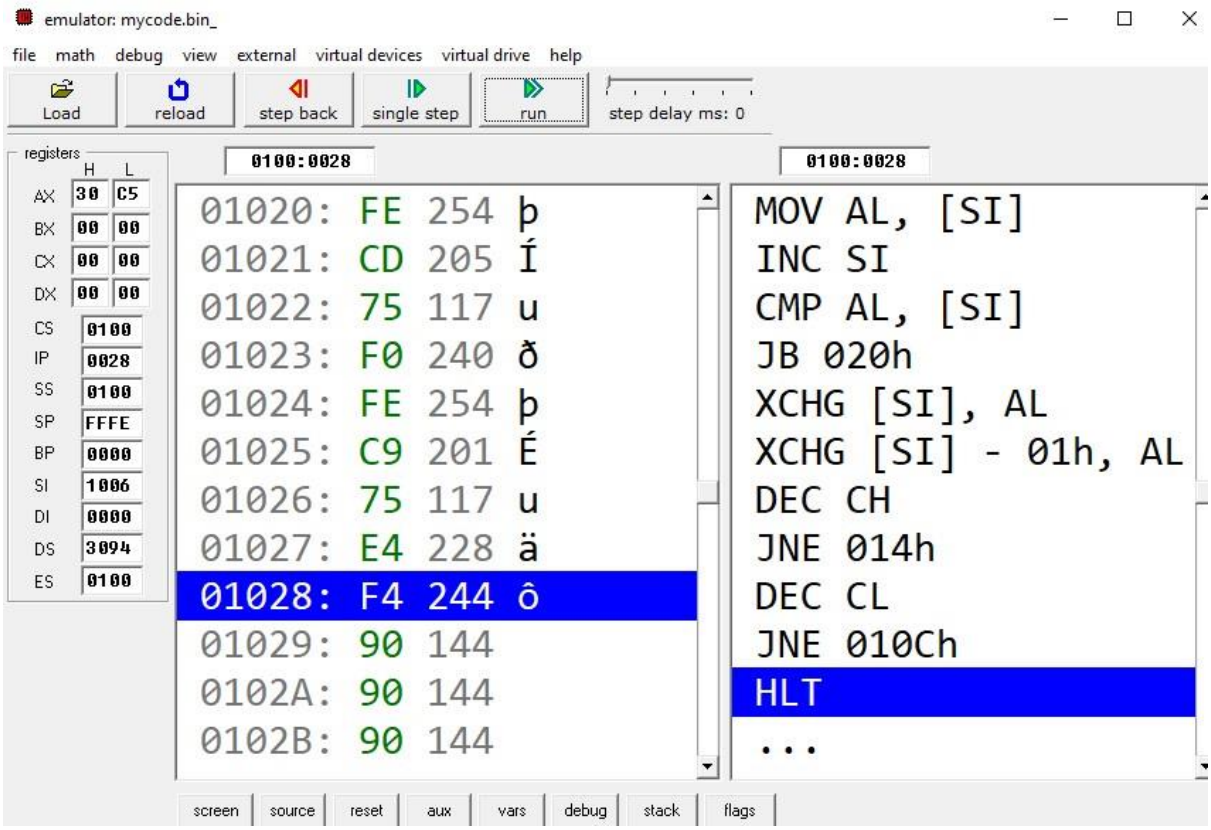
[32944] = AAH

[32945] = C5H

[32946] = F1H



Emulator window after execution


emulator: mycode.bin_
file math debug view external virtual devices virtual drive help
Load
reload
step back
single step
run
step delay ms: 0
registers
H L
AX 30 C5
BX 00 00
CX 00 00
DX 00 00
CS 0100
IP 0028
SS 0100
SP FFFE
BP 0000
SI 1006
DI 0000
DS 3094
ES 0100
0100:0028
01020: FE 254 þ
01021: CD 205 Í
01022: 75 117 u
01023: F0 240 ð
01024: FE 254 þ
01025: C9 201 É
01026: 75 117 u
01027: E4 228 ä
01028: F4 244 ô
01029: 90 144
0102A: 90 144
0102B: 90 144
0100:0028
MOV AL, [SI]
INC SI
CMP AL, [SI]
JB 020h
XCHG [SI], AL
XCHG [SI] - 01h, AL
DEC CH
JNE 014h
DEC CL
JNE 010Ch
HLT
...
screen source reset aux vars debug stack flags

Experiment 8

AIM: Program to arrange a given number in ascending order

ARRANGING THE GIVEN DATA IN DESCENDING ORDER

CODE:

```
MOV AX , 3094H          ; Storing 3094H in AX
MOV DS , AX             ; Initialising DS with the help of AX
START: MOV SI , 1000H   ; Store 1000H in SI
MOV CL , [SI]           ; Store the content of SI to CL
DEC CL                  ; Decrement the value of CL
REPEAT: MOV SI , 1000H  ; Move 1000H to SI
MOV CH , [SI]           ; Move the content of SI to CH
DEC CH                  ; Decrement the value of CH
INC SI                  ; Increment the value of SI
REPCOM: MOV AL , [SI]   ; Get an element of Array in AL register
INC SI                  ; Increment the value of SI
CMP AL , [SI]           ; Compare the values of AL and the content of SI
JNC AHEAD               ; Jump if carry not set to AHEAD
XCHG AL , [SI]          ; If AL is less than memory
XCHG AL , [SI-1]        ; Exchange the content of memory pointed by SI and its
                          previous location
AHEAD: DEC CH           ; Decrement the value of CH
JNZ REPCOM              ; Repeat comparison until count is 0
DEC CL                  ; Decrement CL
JNZ REPEAT              ; Repeat N-1 comparisons until cont is 0
HLT
```

OUTPUT:

Values stored in memory location (Values to be arranged in Descending order)

[31940] = 06H (Count)

[31941] = 83H

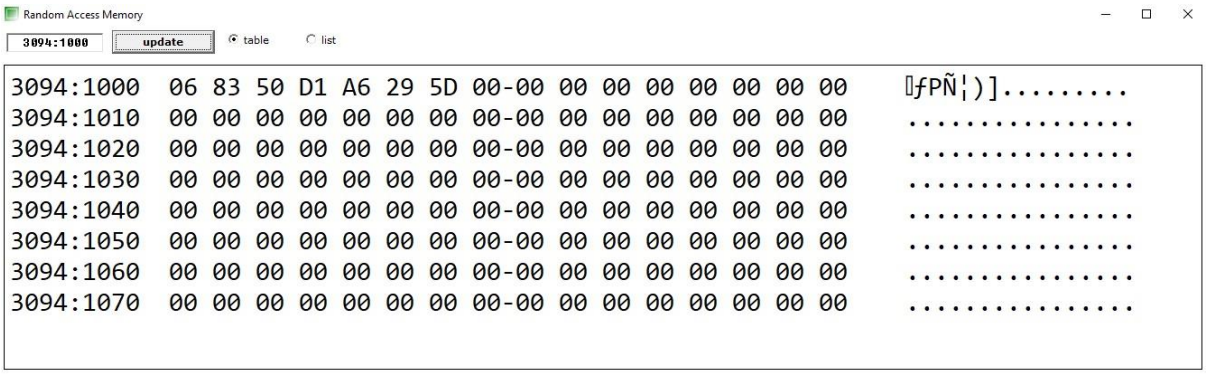
[31942] = 50H

[31943] = D1H

[31944] = A6H

[31945] = 29H

[31946] = 5DH



Values stored in memory location (In descending order)

[32940] = 06H (Count)

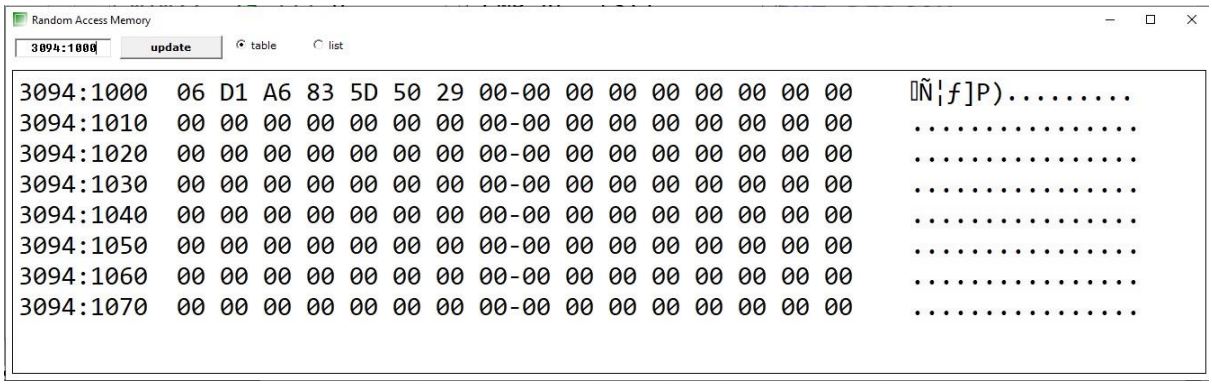
[32941] = D1H

[32942] = A6H

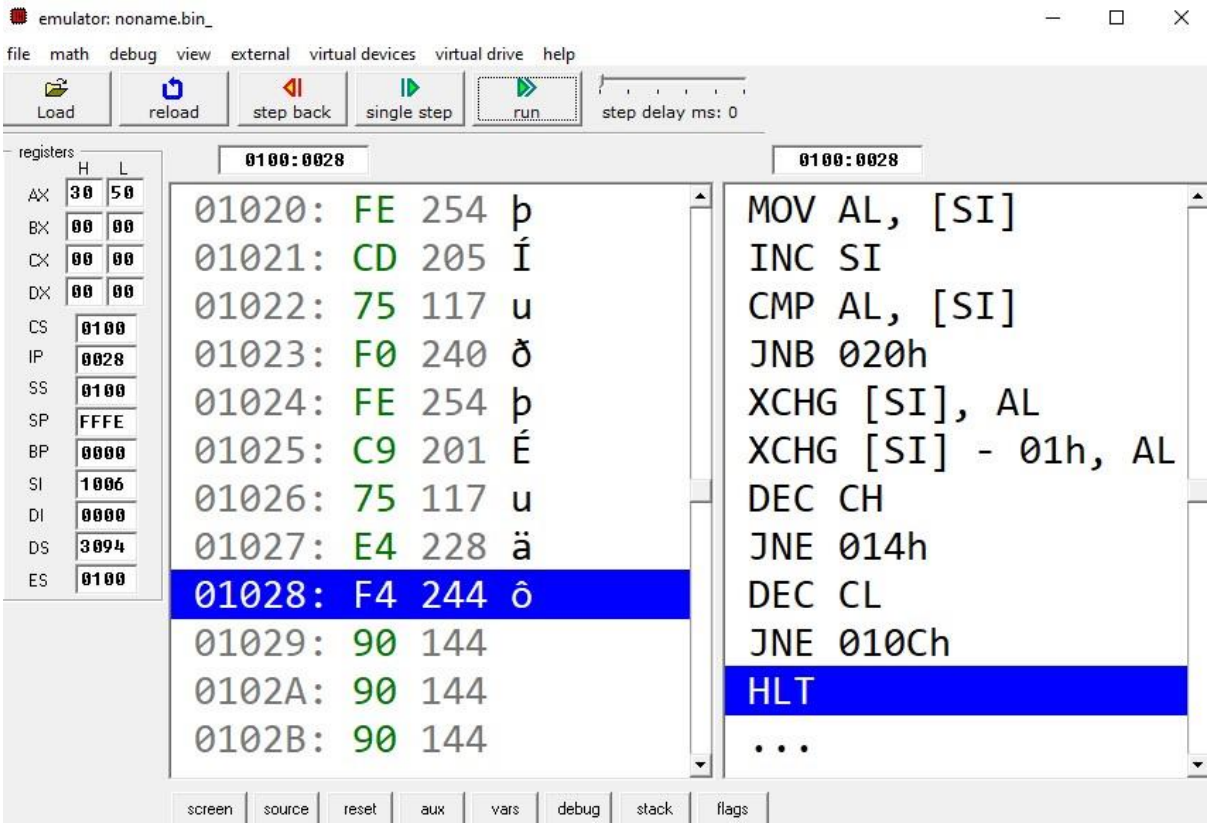
[32943] = 83H

[32944] = 5DH

[32945] = 50H [32946] = 29H



Emulator window after execution



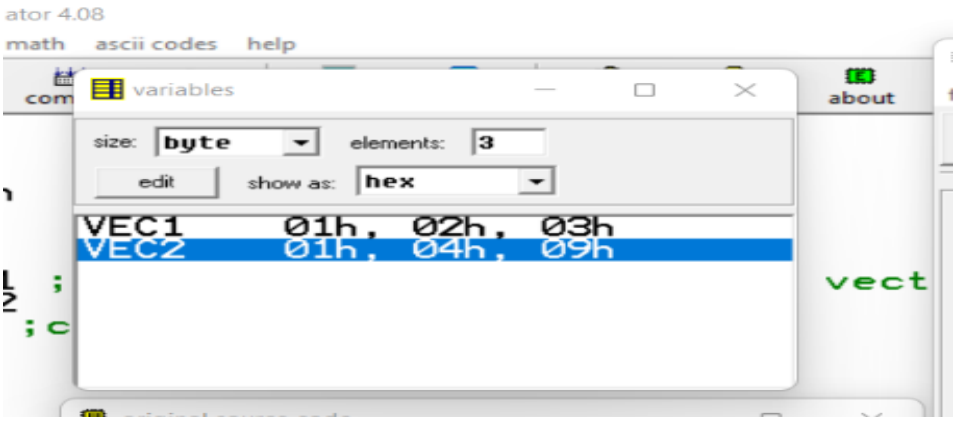
Experiment 9

AIM: Program to do square of the given series.

CODE:

```
ORG 100H

JNP START

VEC1 DB 1H,2H,3H

VEC2 DB ?,?,?


START:

   LEA SI, VEC1
```

```
    LEA DI, VEC2
    MOV CX, 3
AGAIN:
    MOV AX,[SI]
    MUL [SI]
    MOV [DI],AX
    INC SI
    INC DI
    DEC CX
    JNZ AGAIN
    HLT
RET
```



Experiment 10

AIM: Program to generate the Fibonacci series.

CODE:

```
MOV AL, 00H
MOV SI, 500
MOV [SI], Al
ADD SI, 1
ADD AL, 1
MOV [SI], AL
MOV CX, [0000H]
```

SUB CX, 02H

L1:

MOV AL, [SI-1]

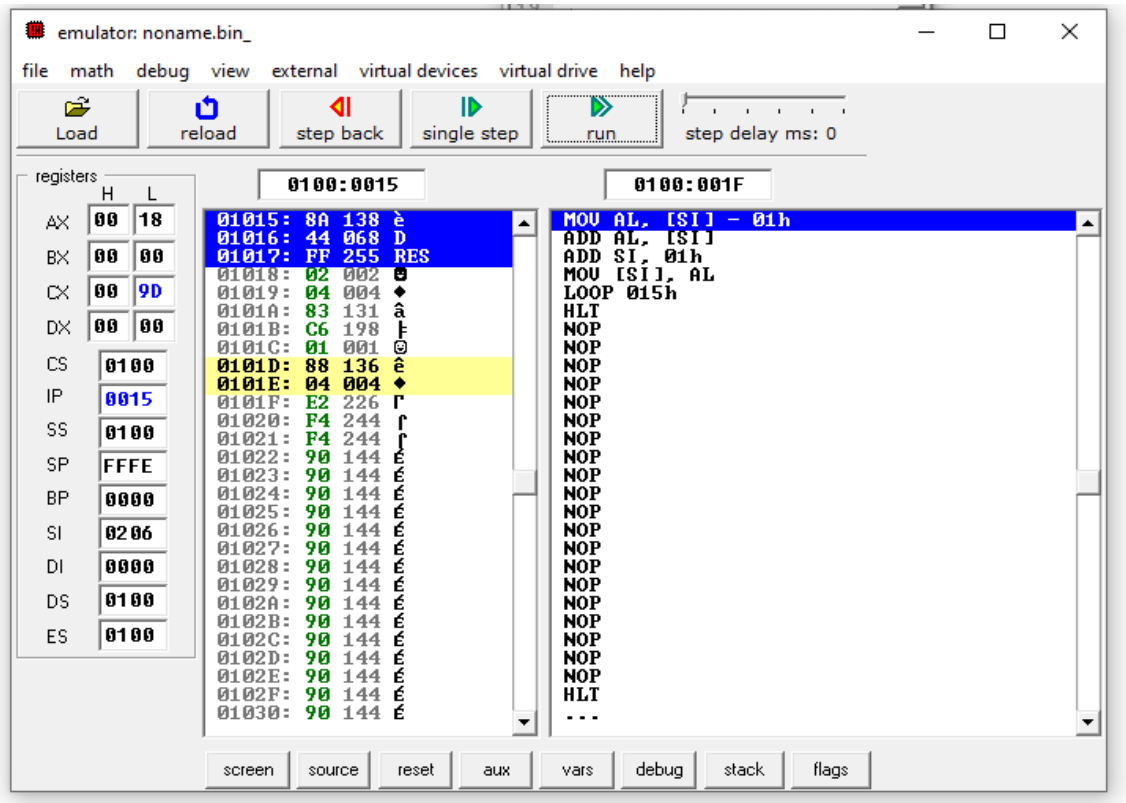
ADD AL, [SI]

ADD SI, 1

MOV [SI], AL

LOOP L1

HLT



Experiment 11

AIM: Program to find out EVEN and ODD numbers in a series.

ODD SUM:

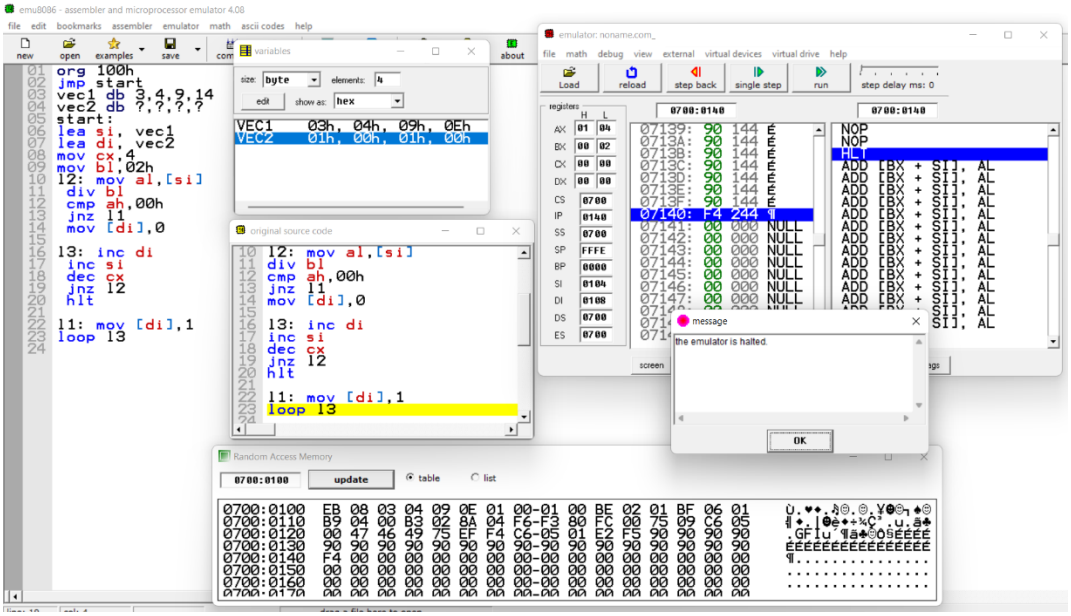
ORG 100H

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] ; AL=3
   DIV BL ; 3/2
   CMP AH,00H    ;R==0
   JNZ L1      ;
   MOV [DI],0 ; EVEN=0

L3: INC DI
   INC SI
   DEC CX  ;2  ;1
   JNZ L2
   HLT

L1: MOV [DI],1   ;ODD=1
   LOOP L3
```

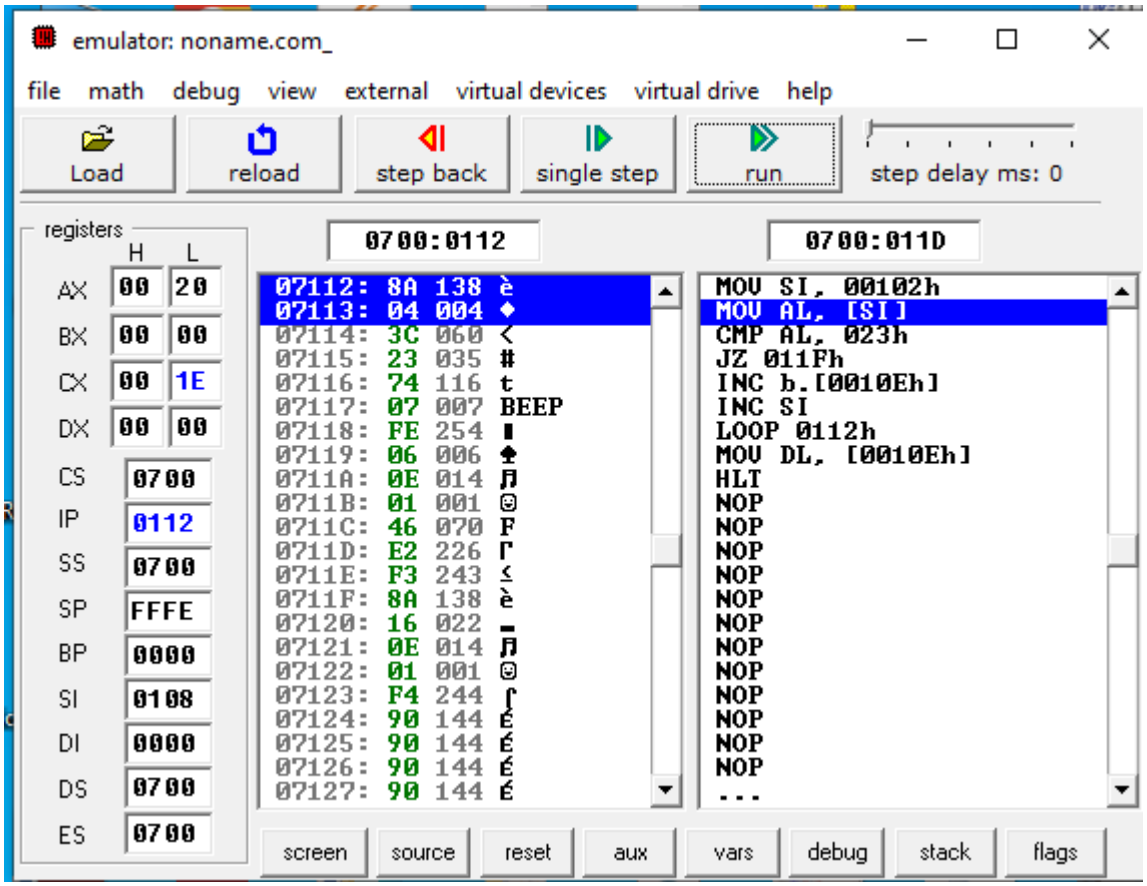


Experiment 12

AIM: Program to count the length of a string

```
org 100h

JMP START

 LEN DB 0
START:

 MOV SI , OFFSET STR1
 UP:  MOV AL , [SI]
```

```
    CMP AL , '#'
    JZ DN        ; IF DESTINATION == SOURCE THEN ZF = 1

    INC LEN
    INC SI
    LOOP UP

DN: MOV DL , LEN
HLT
```



Experiment 13

AIM: Program to display data on LED

CODE:

```
    Org 100h

    MOV AX, 00h

L3:OUT 199,AX

     CALL DELAY

     ADD AX,01H
```

```
    JMP L3
DELAY: MOV CL,01FH
       MOV CH,001H
L1:    DEC CH
       JNZ L1
L2:    DEC CL
       JNZ L2
       RET
```