**PRACTICAL – 1**

**AIM:**

|  |
| --- |
| **Introduction to 8086 Microprocessor & Assembly Language Programming.** |

**THEORY:**

**8086 Microprocessor**

* Intel 8086 microprocessor is the enhanced version of Intel 8085 microprocessor. It was designed by Intel in 1976.
* The 8086 microprocessor is a16-bit, N-channel, HMOS microprocessor. Where the HMOS is used for "**High-speed Metal Oxide Semiconductor**".
* Intel 8086 is built on a single semiconductor chip and packaged in a 40-pin IC package. The type of package is DIP (Dual Inline Package).
* Intel 8086 uses 20 address lines and 16 data- lines. It can directly address up to 220 = 1 Mbyte of memory.
* It consists of a powerful instruction set, which provides operation like division and multiplication very quickly.
* 8086 is designed to operate in two modes, i.e., Minimum and Maximum mode.

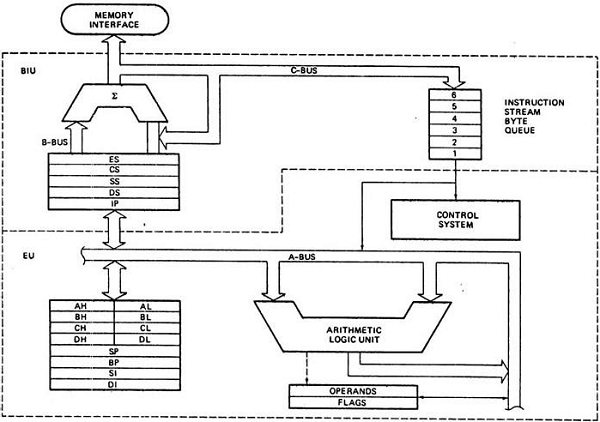
## Features of 8086

The most prominent features of a 8086 microprocessor are as follows −

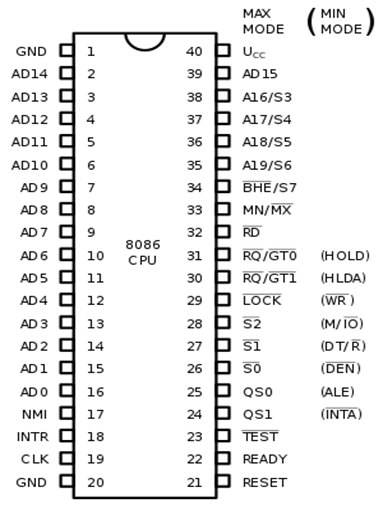
* It has an instruction queue, which is capable of storing six instruction bytes from the memory resulting in faster processing.
* It was the first 16-bit processor having 16-bit ALU, 16-bit registers, internal data bus, and 16-bit external data bus resulting in faster processing.
* It is available in 3 versions based on the frequency of operation −
  + 8086 → 5MHz
  + 8086-2 → 8MHz
  + (c)8086-1 → 10 MHz
* It uses two stages of pipelining, i.e. Fetch Stage and Execute Stage, which improves performance.
* Fetch stage can prefetch up to 6 bytes of instructions and stores them in the queue.
* Execute stage executes these instructions.
* It has 256 vectored interrupts.
* It consists of 29,000 transistors.

## Architecture of 8086

The following diagram depicts the architecture of a 8086 Microprocessor.



## 8086 pins configuration



**Assembly Language Programming**

* The assembly programming language is a low-level language which is developed by using mnemonics. The microcontroller or microprocessor can understand only the binary language like 0's or 1's therefore the assembler convert the assembly language to binary language and store it the memory to perform the tasks.

**CONCLUSION:** We learnt about 8086 Microprocessor & Assembly Language Programming.

**PRACTICAL – 2**

**AIM:**

|  |
| --- |
| **Store the data byte 32H into memory location 4000H.** |

**CODE:**

ORG 100H

MOV [4000H],32H

MOV AX,[4000H]

MOV [2000H],AX

RET

**OUTPUT:**

****

**CONCLUSION:** In this practical we learnt how to stored data byte in memory location.

**PRACTICAL – 3**

**AIM:**

|  |
| --- |
| **Exchange the contents of memory locations 2000H and 4000H** |

**CODE:**

ORG 100H

ORG 100H

MOV [4000H],32H

MOV [2000H],16H

MOV AX,[4000H]

MOV BX,[2000H]

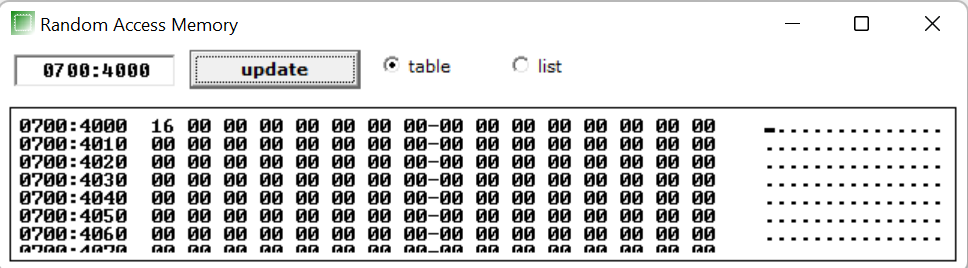
MOV [4000H],BX

MOV [2000H],AX

RET

**OUTPUT:**

****

****

**CONCLUSION:** In this practical we learnt to exchanged contents one memory location into another memory location.

**PRACTICAL – 4**

**AIM:**

|  |
| --- |
| **Convert the below given C Program into Assembly Language.**  **main()**  **{**  **Int l, n, o, p;**  **l = m + n + o + p;**  **}** |

**CODE:**

ORG 100H

MOV BL,05H

MOV CL,04H

MOV DL,02H

MOV AH,05H

ADD BL,CL

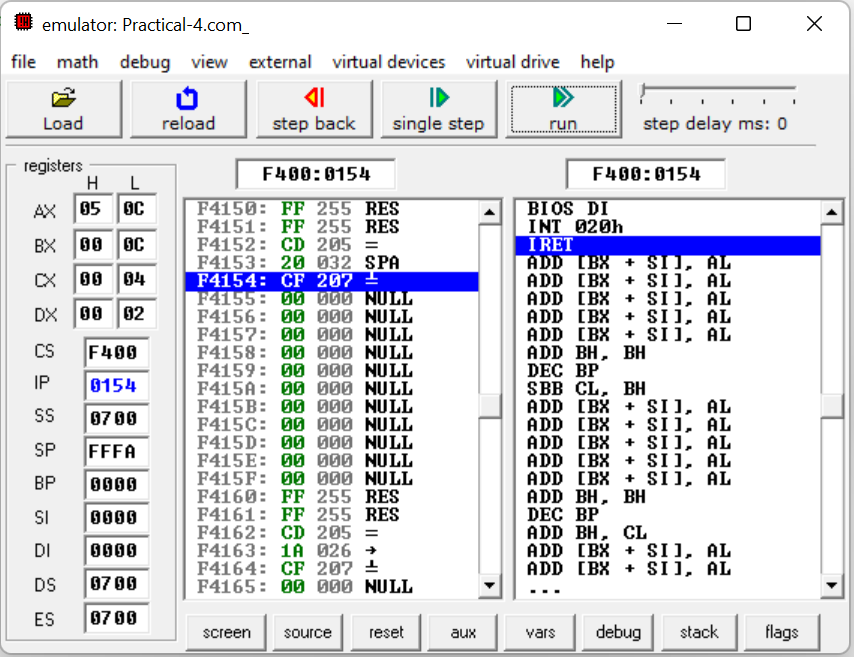
SUB BL,DL

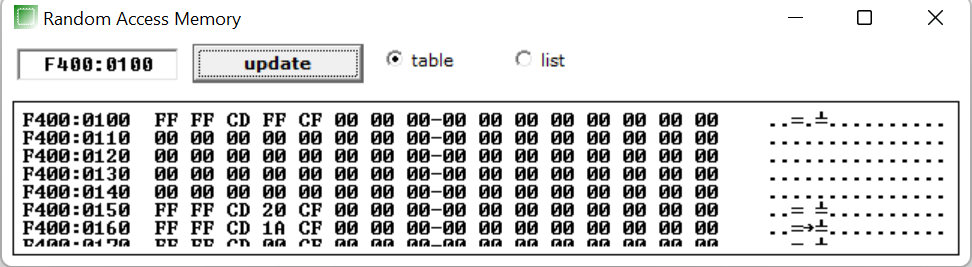
ADD BL,AH

MOV AL,BL

RET

**OUTPUT:**



****

**CONCLUSION:** In this practical we learnt ADD and SUB values of registors.

**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 [4000H],0505H

MOV [2000H],0202H

MOV AX,[4000H]

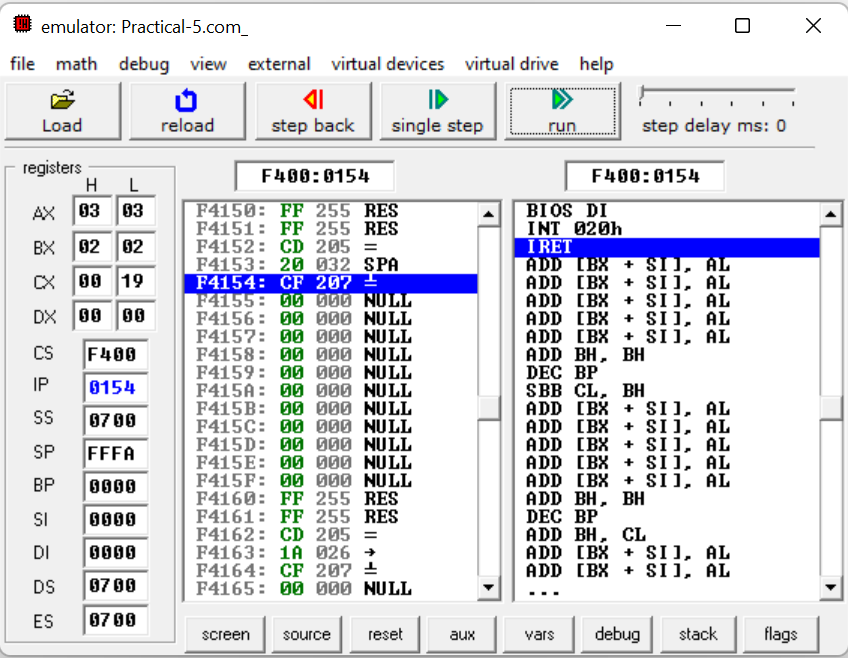
MOV BX,[2000H]

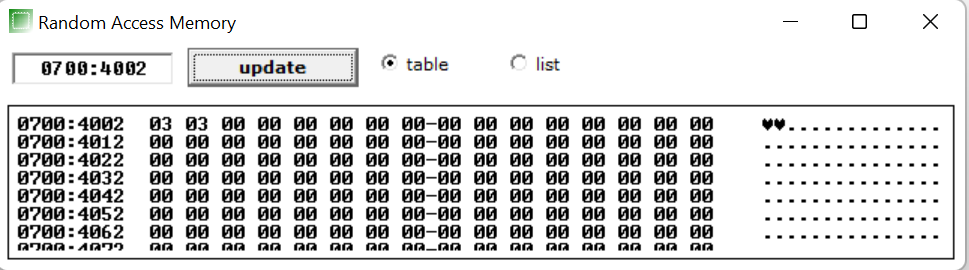
SUB AX,BX

MOV [4002H],AX

RET

**OUTPUT:**



****

**CONCLUSION:** In this practical we learnt to subtract the contents of memory location and stored result in another memory location.

**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 to be 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 [4000H],5050H

MOV [4002H],2020H

MOV AX,[4000H]

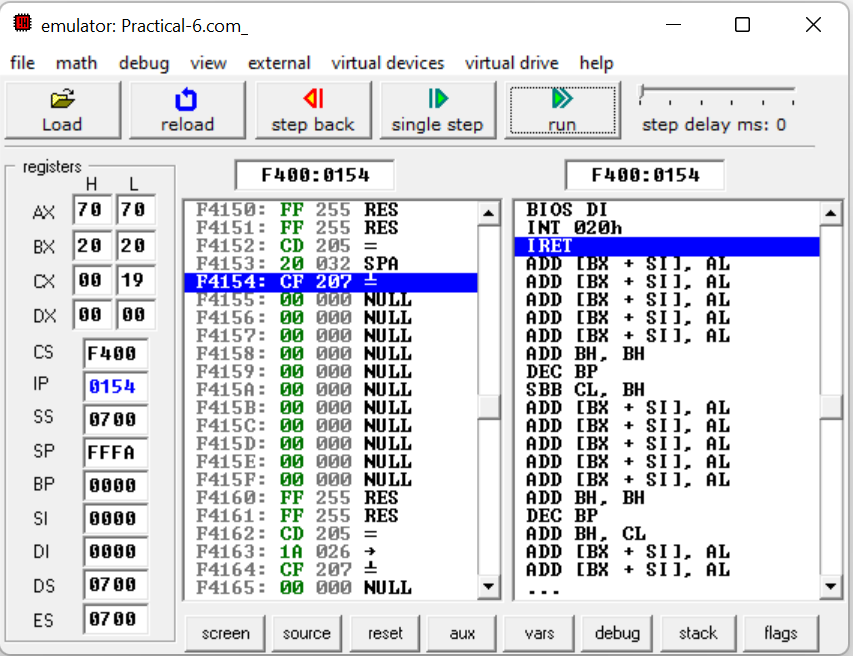
MOV BX,[4002H]

ADD AX,BX

MOV [4004H],AX

RET

**OUTPT:**



****

**CONCLUSION:**  In this practical we learnt how to add 16 bit number in memory location.

**Umber**

**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 [4002H],5050H

MOV [4000H],2020H

MOV AX,[4000H]

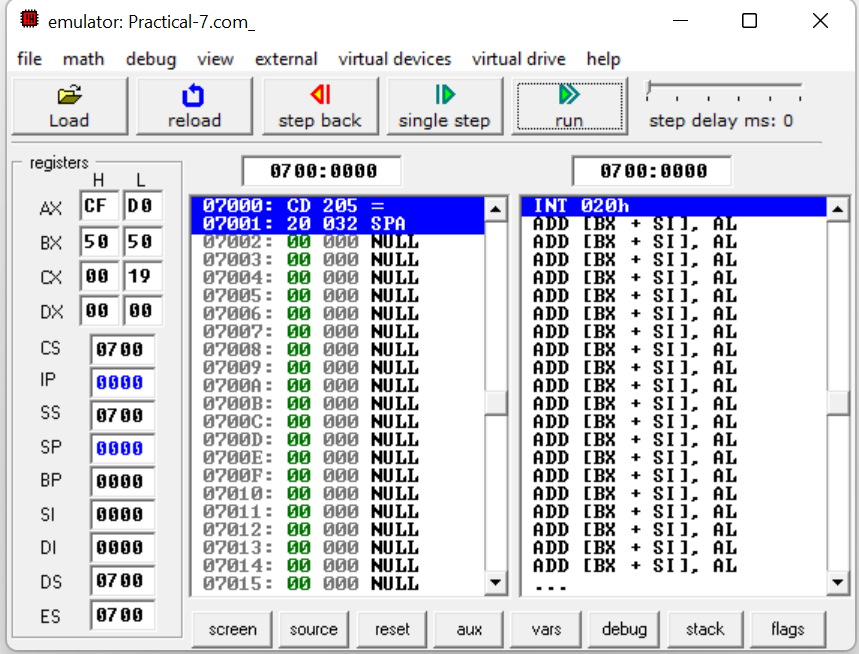
MOV BX,[4002H]

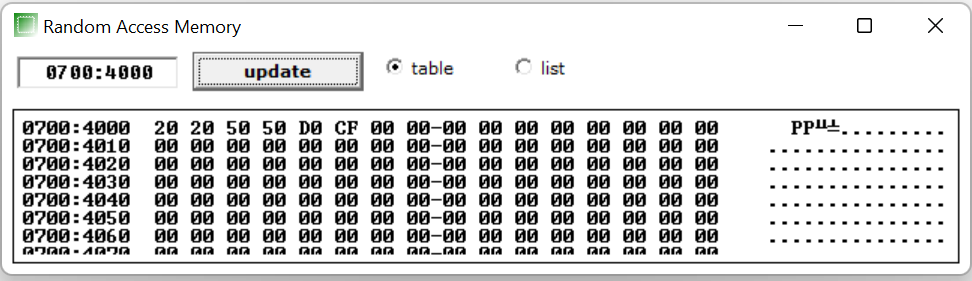
SUB AX,BX

MOV [4004H],AX

RET

**OUTPUT:**



****

**CONCLUSION:** In this practical we learnt how to subtract 16 bit number in memory location.

**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 [7000H], 7615H

MOV [7002H], 1495H

MOV [7004H], 1515H

MOV [7006H], 1313H

MOV AX, [7000H]

MOV BX, [7002H]

MOV CX, [7004H]

MOV DX, [7006H]

ADD AX,CX

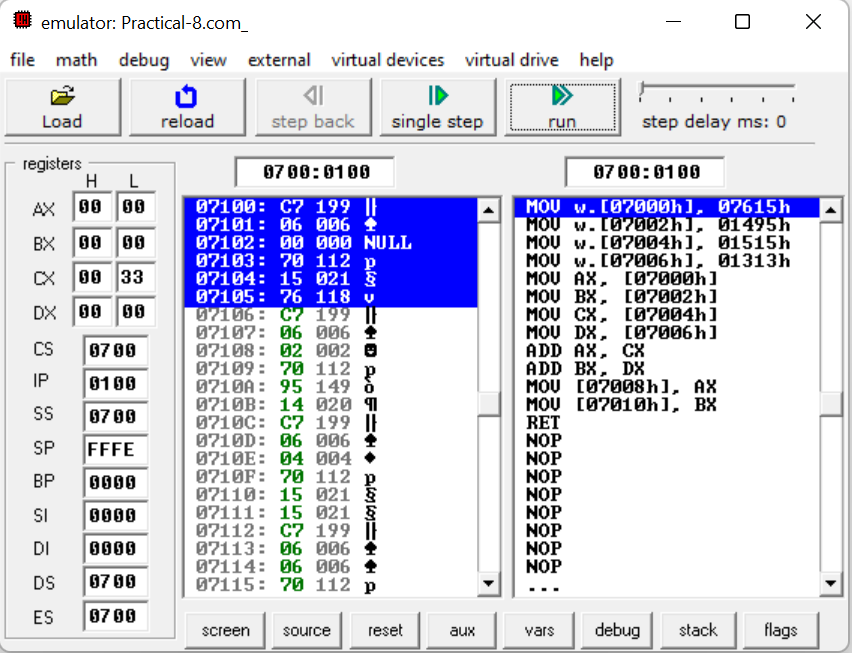
ADD BX,DX

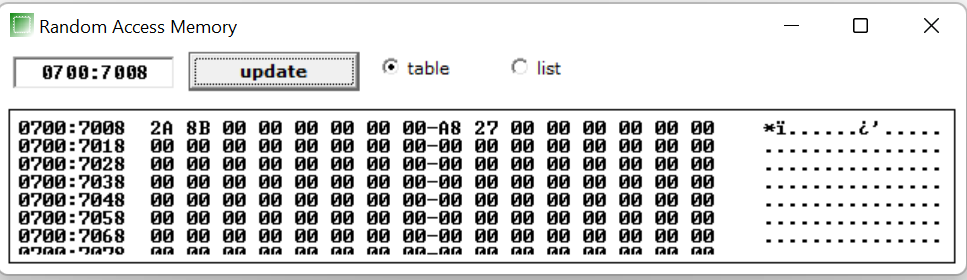
MOV [7008H], AX

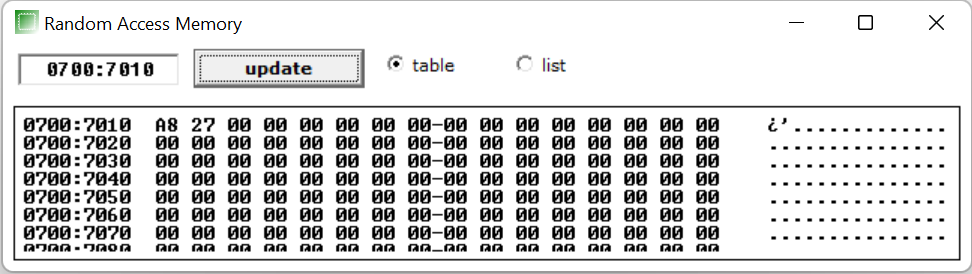
MOV [7010H], BX

RET

**OUTPUT:**

****

****

****

**CONCLUSION:** In this practical we learnt how to add two 32 bit number in consecutive memory location.

**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 [7000H], 7615H

MOV [7002H], 1495H

MOV [7004H], 1515H

MOV [7006H], 1313H

MOV AX, [7000H]

MOV BX, [7002H]

MOV CX, [7004H]

MOV DX, [7006H]

SUB AX,CX

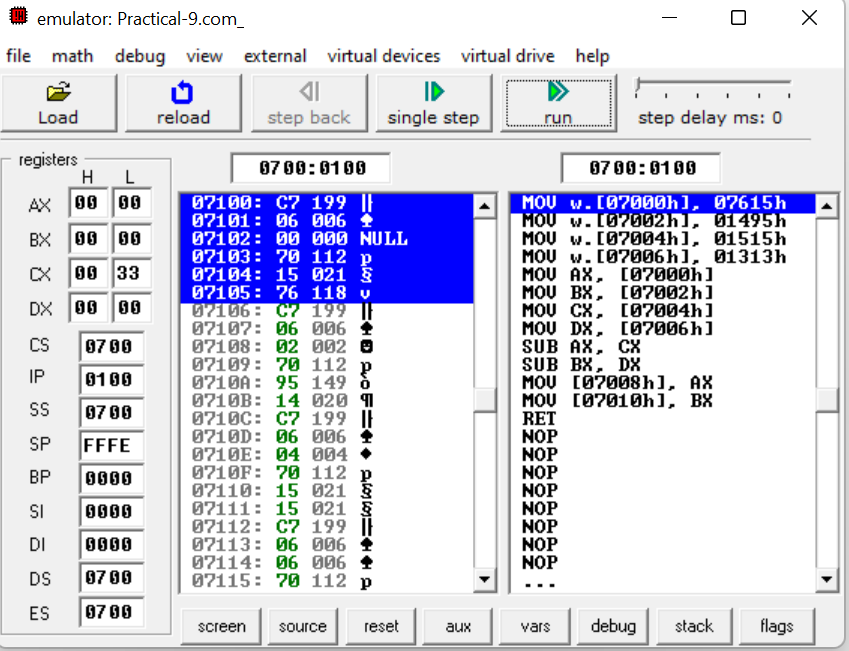
SUB BX,DX

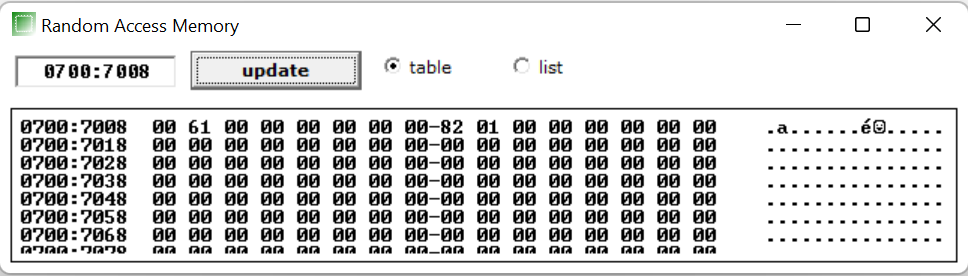
MOV [7008H], AX

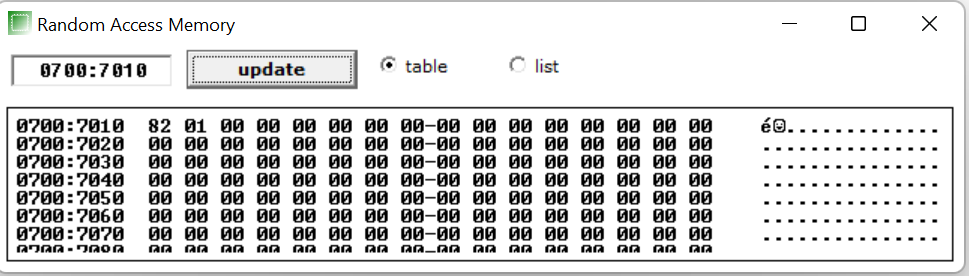
MOV [7010H], BX

RET

**OUTPUT:**

****

****

****

**CONCLUSION:** In this practical we learnt how to subtract two 32 bit number in consecutive memory 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, 98

SUB AL, 32

MOV BL, 05

MOV CL, 09

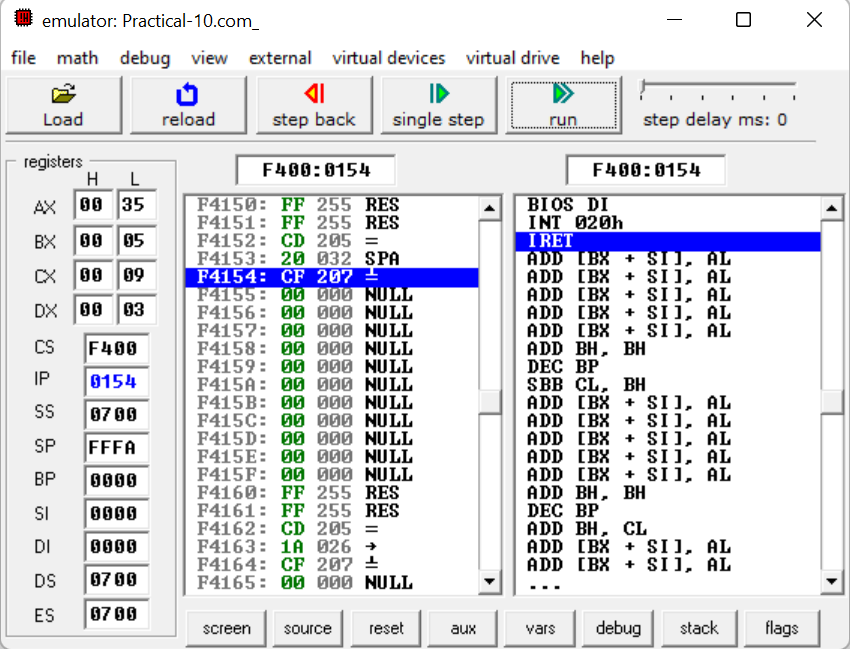
DIV CL

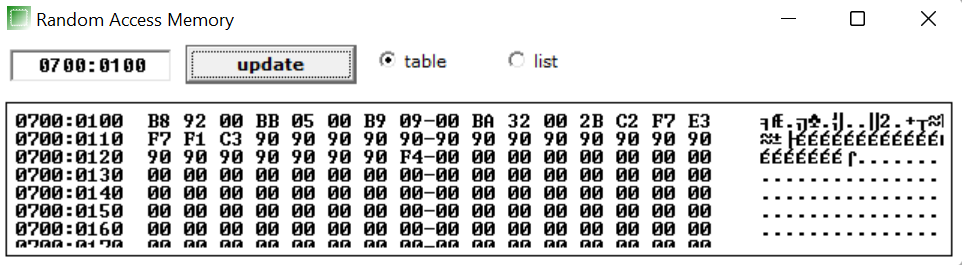
MUL AL

DIV CL

RET

**OUTPUT:**

****

****

**CONCLUSION:** We learnt how to calculate Celsius from given fahrenheit by giving specific memory locations.

**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], 10101011B

MOV [4002H], 11010101B

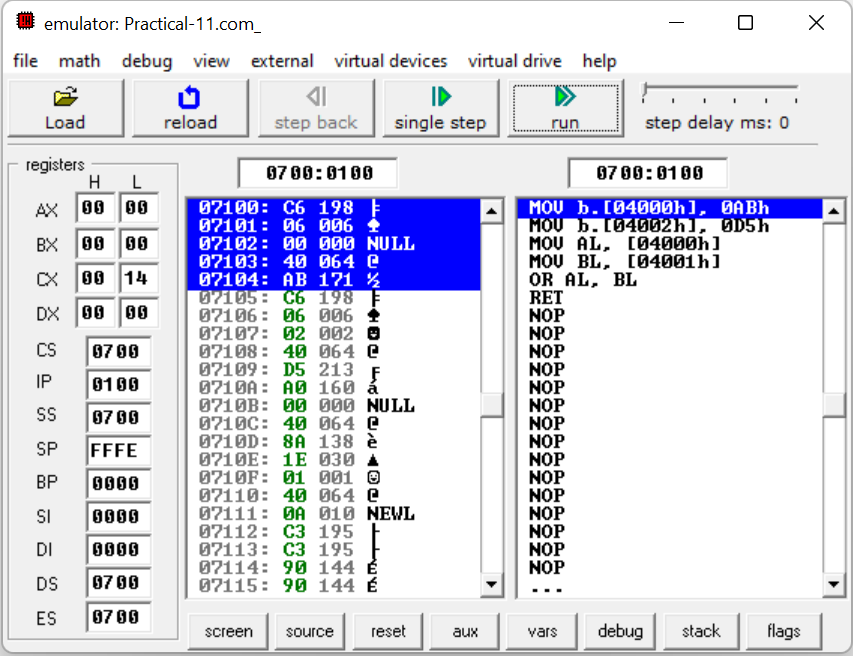
MOV AL,[4000H]

MOV BL,[4001H]

OR AL,BL

RET

**OUTPUT:**

****

**CONCLUSION:** We learnt how to selective operation on specific location.

**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], 10101011B

MOV [4002H], 11010101B

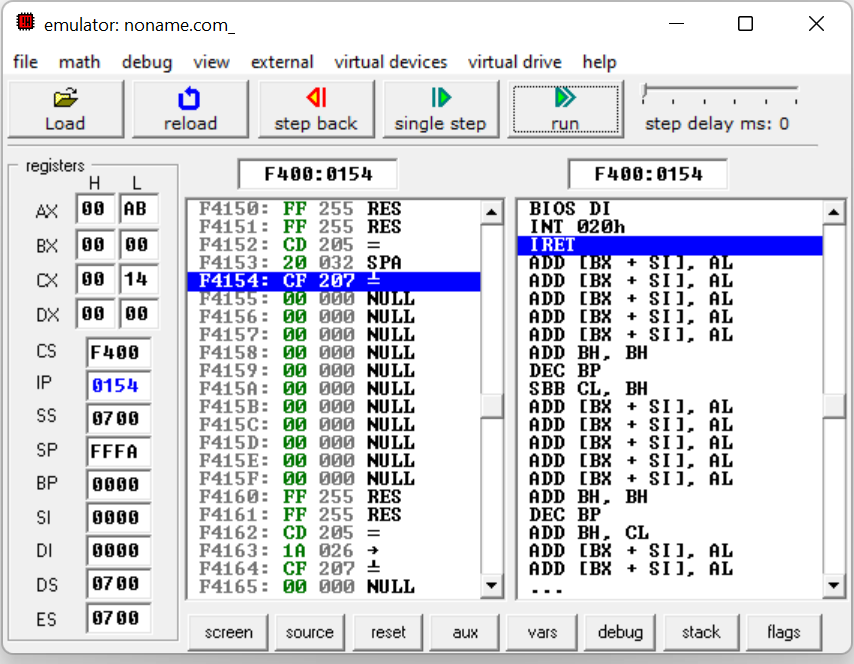
MOV AL,[4000H]

MOV BL,[4001H]

XOR AL,BL

RET

**OUTPUT:**

****

**CONCLUSION:** We learnt how to perform selective compliment operation. By XOR gate.

**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], 10101011B

MOV [4002H], 11010101B

MOV AL,[4000H]

MOV BL,[4001H]

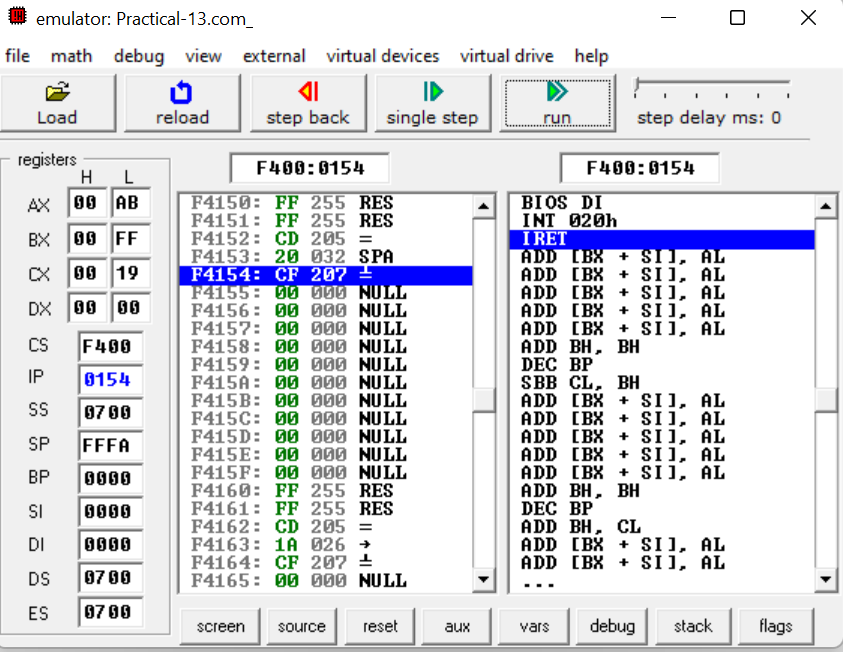
NOT BL

AND AL,BL

MOV [4001H],AL

RET

**OUTPUT:**

****

**CONCLUSION:** We learnt how to perform selective compliment operation. By AND gate.

**PRACTICAL – 11**

**AIM:**

**CODE:**

**OUTPUT:**

**PRACTICAL – 11**

**AIM:**

**CODE:**

**OUTPUT:**

**PRACTICAL – 11**

**AIM:**

**CODE:**

**OUTPUT:**

**PRACTICAL – 11**

**AIM:**

**CODE:**

**OUTPUT:**