|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FAKULTI TEKNOLOGI KEJURUTERAAN**  **ELEKTRIK DAN ELEKTRONIK**  **UNIVERSITI TEKNIKAL MALAYSIA MELAKA** | | | | | |
| **COMPUTER ORGANIZATION AND ARCHITECHTURE** | | | | | |
| BEEC 2373 | | | SEMESTER 2 | SESI 2019/2020 | |
| LAB 4: LOGICAL INSTRUCTION | | | | | |
| **NO.** | **STUDENTS' NAME** | | | | **MATRIC. NO.** |
| **1.** | **Ahmad Irfan Bin Harman** | | | | **B081910068** |
| **2.** | **CHE IZZATI AYUNI BINTI CHE ROHIM** | | | | **B081910021** |
|  |  | | | |  |
| **PROGRAMME** | | **1BEEC** | | | |
| **SECTION / GROUP** | | **S1/1** | | | |
| **DATE** | | **17/05/2000** | | | |
| **NAME OF INSTRUCTOR(S)** | | 1. **EN. NOOR MOHD ARIFF BIN BRAHIN** | | | |
|  | | | |
| **EXAMINER’S COMMENT(S)** | | | | **TOTAL MARKS** | |

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. No. | Date | Author(s) | Description |
| 1.0 | 12 FEB 2020 | 1. Noor Mohd Ariff 2. Ahmad Nizamudin | 1. Update to new UTeM logo 2. Update faculty's name 3. Change "course" to "programme" 4. Remove verification stamp |
|  |  |  |  |
|  |  |  |  |

## OBJECTIVES

* To construct and test programs using the following categories of 8086 Instruction Set:

• Data Movement

• Arithmetic Operations

* To understand 8086 microprocessor architecture.

## EQUIPMENT/COMPONENTS

1. Personal computer installed with 8086 Software.

## SYNOPSIS & THEORY

The table below summarizes the different categories of 8086 Instruction Set. In this experiment you will be creating and testing codes written in assembly language programs using 8086 Instruction Set. Observe how the 8086 internal registers are affected by the different instructions.

**TABLE 1: Categories of 8086 Instruction**

**Set**

**Type Description Operation Name**

**Data Transfer** Transfer data from one location to another

Move, Store, Load, Exchange, Clear, Set, Push, Pop

**Arithmetic** Perform arithmetic function in ALU

Add, Subtract, Multiply, Divide, Absolute, Negate, Increment, Decrement

**Logical** Perform logic function in ALU AND, OR, NOT, XOR, Test, Compare, Shift, Rotate

**Transfer of Control** Update program counter Jump, Jump Conditional, Jump to Subroutine, Return, Skip, Skip Conditional, Halt, Wait,No opertion

**Input/Output** Issue command to I/O

module

Input, Output, Start I/O, Test I/O

**Conversion** May involve special logic Translate, Convert

to perform conversion

1. **PROCEDURE**

## Logical Instructions

1. Run the emulator8086.
2. Perform this operation in single step mode and write the values of registers for every step.

ORG 100H

MOV AX, 102H

MOV BX, 5A2H

MOV CX, 54AH

OR AX, BX

XOR AX, CX

NOT AX

TEST CX, BX

AND CX, AX

RET

1. Given a logical operation:

A = 101010102 B = 100010002

C = 010101012

(A **AND** B) **OR** (C **XOR** B)

1. Develop your own codes in 8086 assembly language.
2. Make sure your registers are well defined.
3. Explain the output with the registers involved.

## Shift and Rotate command

1. Run these codes and find out the registers output.

MOV CL, 03H

MOV AX, 105BH

SHL AX, CL

HLT

MOV CL, 03H

MOV AX, 105BH

SAR AX, CL

HLT

* 1. MOV CL, 03H

MOV AX, 105BH

ROR AX, CL

HLT

* 1. MOV CL, 03H

MOV AX, 105BH

RCL AX, CL

HLT

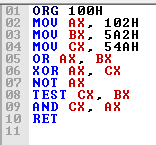
## RESULT

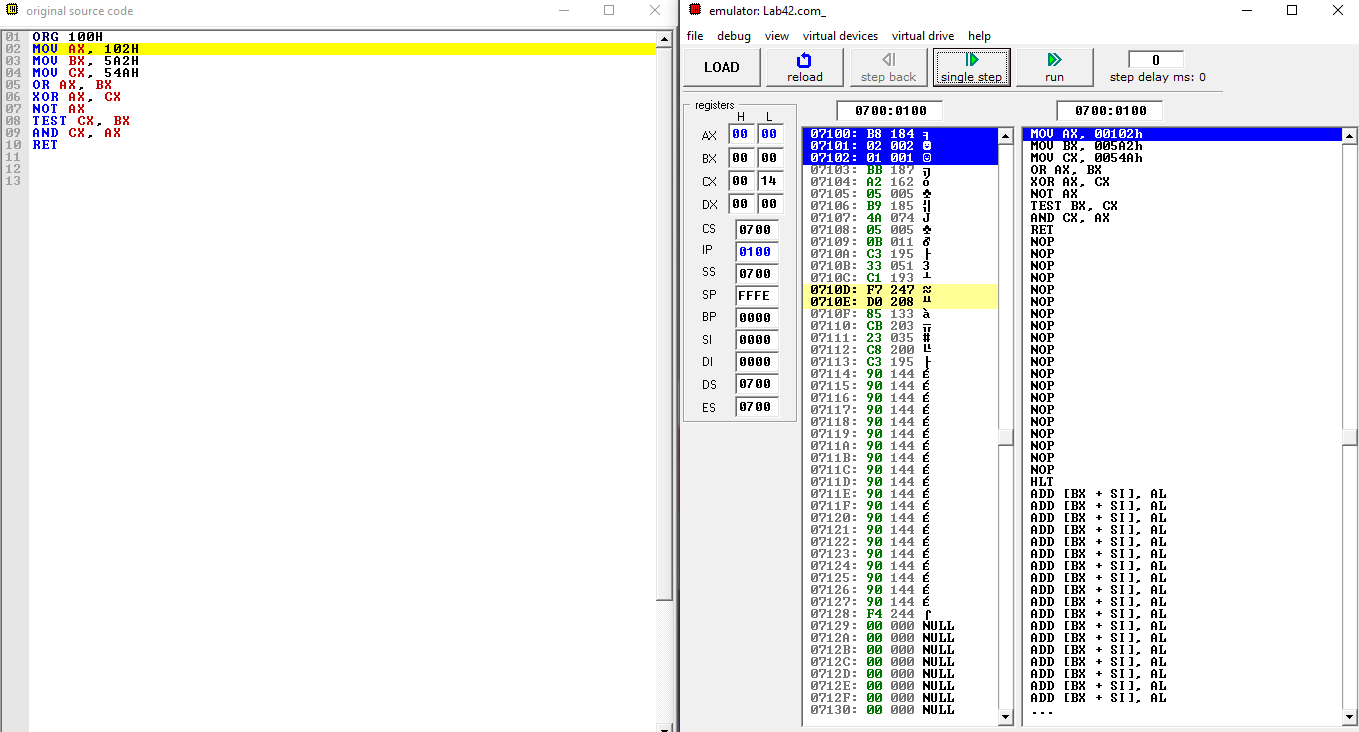
**5.1 Logical Instructions**

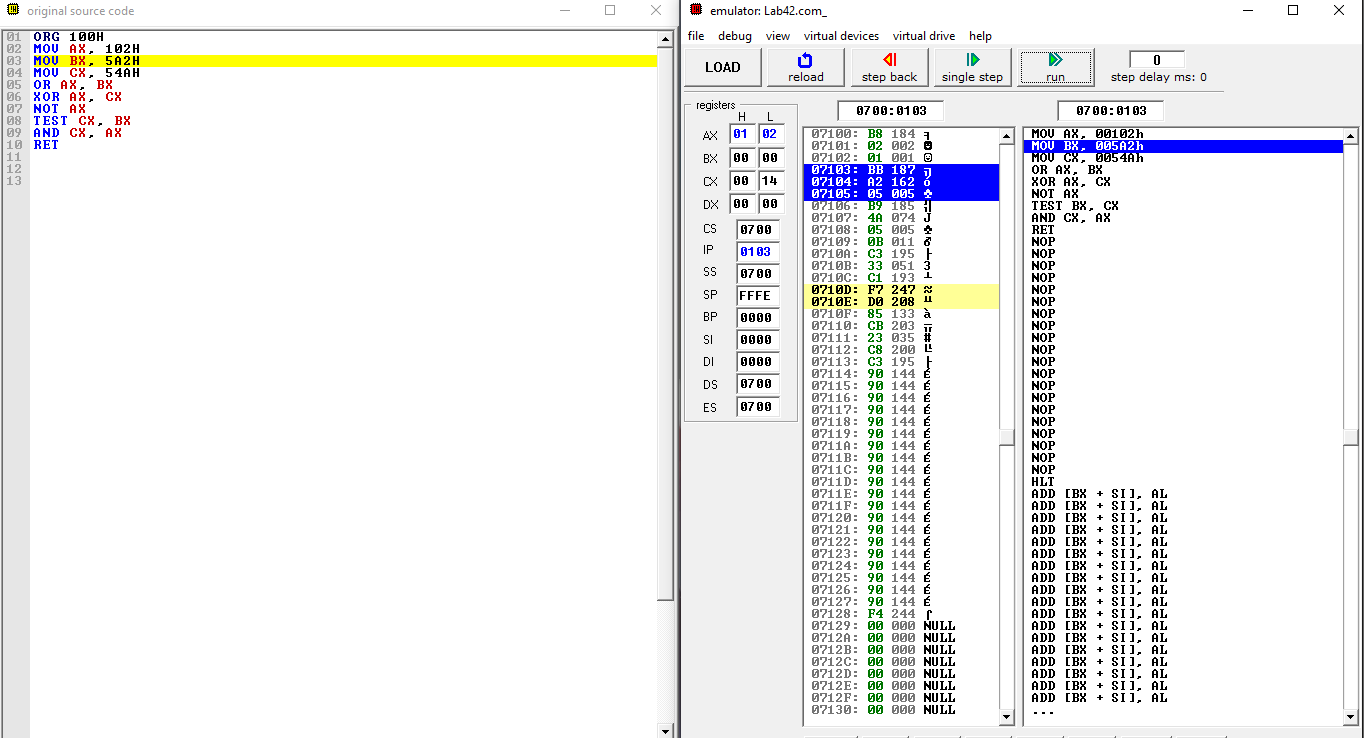
1. Observe the contents of the specified registers below after each instruction has been executed and record your result in Table 1.

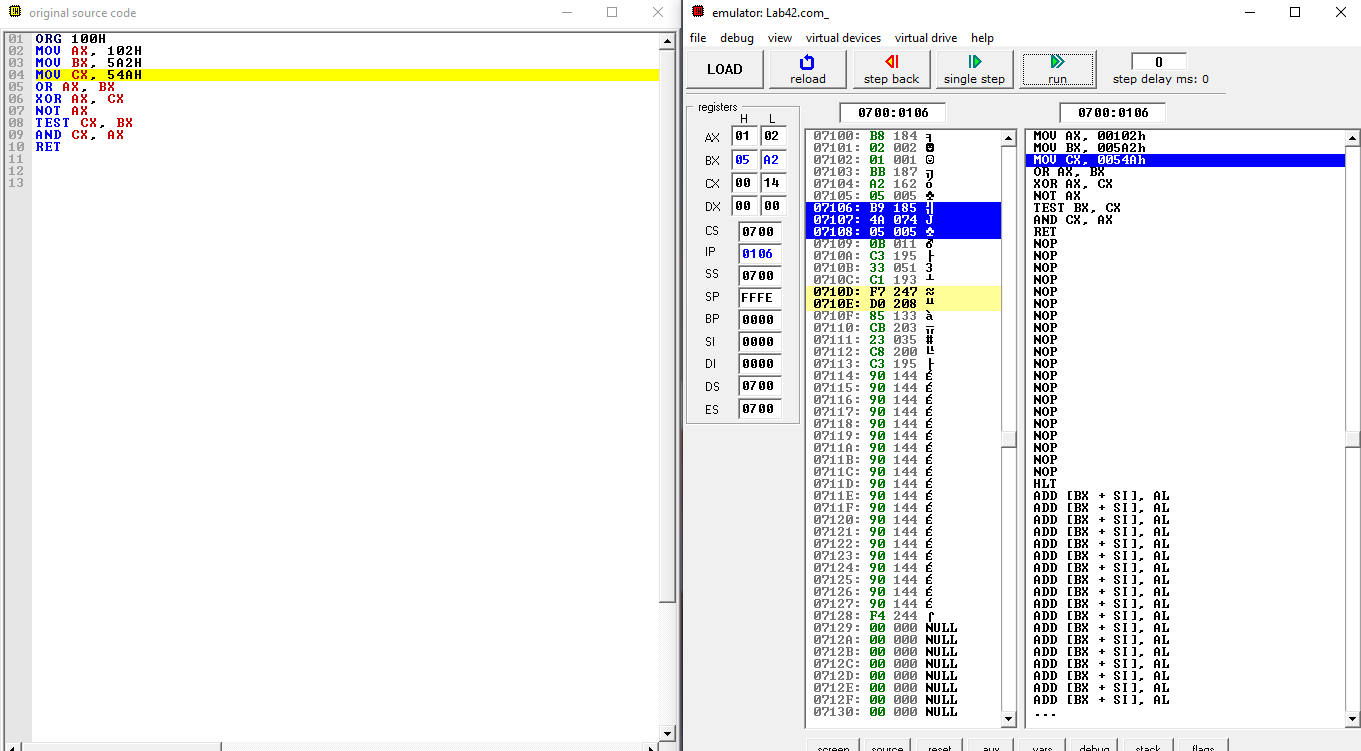
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Instruction | Register Content | | | | |
| AX | BX | CX | DX | IP |
| 1. ORG 100H | 0000 | 0000 | 0014 | 0000 | 0100 |
| 2. MOV AX, 102H | 0102 | 0000 | 0014 | 0000 | 0103 |
| 3. MOV BX, 5A2H | 0102 | 05A2 | 0014 | 0000 | 0106 |
| 4. MOV CX, 54AH | 0102 | 05A2 | 054A | 0000 | 0109 |
| 5. OR AX, BX | 05A2 | 05A2 | 054A | 0000 | 010B |
| 6. XOR AX, CX | 00E8 | 05A2 | 054A | 0000 | 010D |
| 7. NOT AX | FF17 | 05A2 | 054A | 0000 | 010F |
| 8. TEST CX, BX | FF17 | 05A2 | 054A | 0000 | 0111 |
| 9. AND CX, AX | FF17 | 05A2 | 0502 | 0000 | 0113 |
| 10. RET | FF17 | 05A2 | 0502 | 0000 | 0000 |

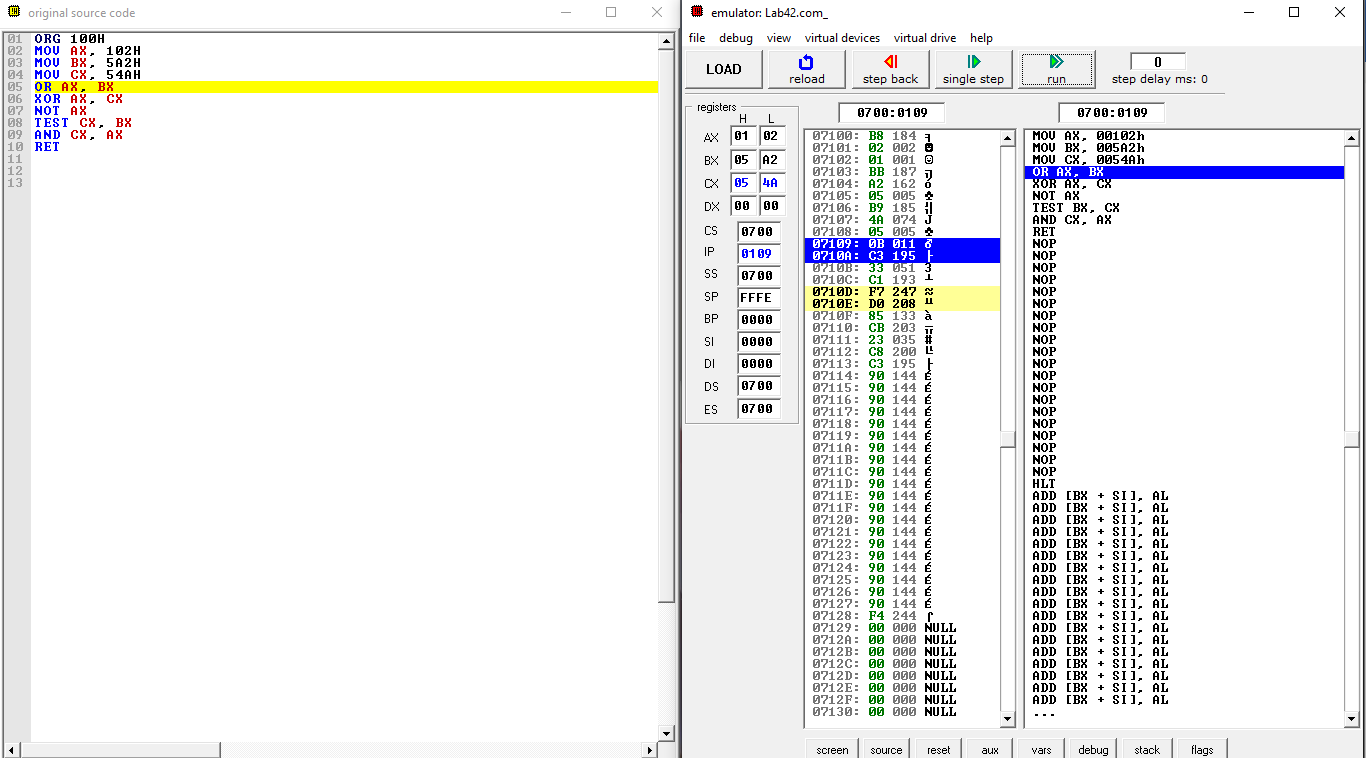
**Table 1**: Registers value

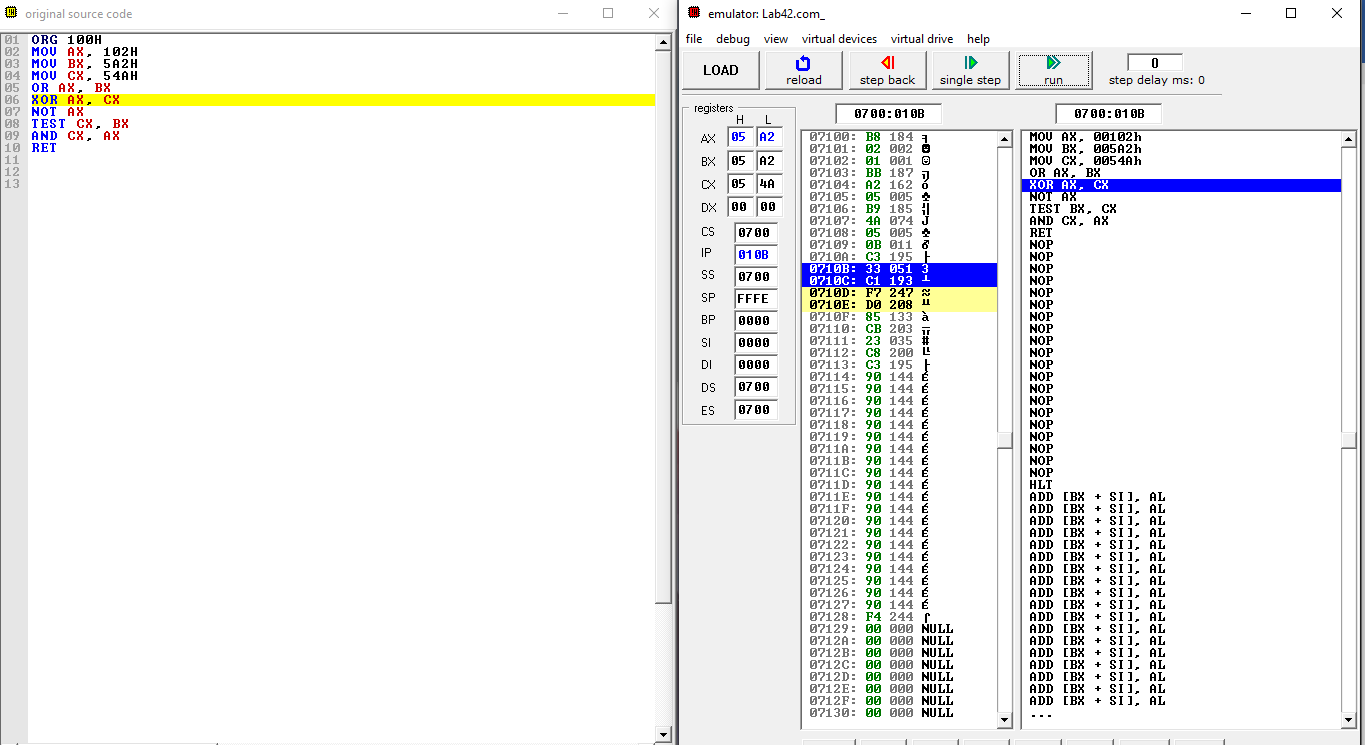


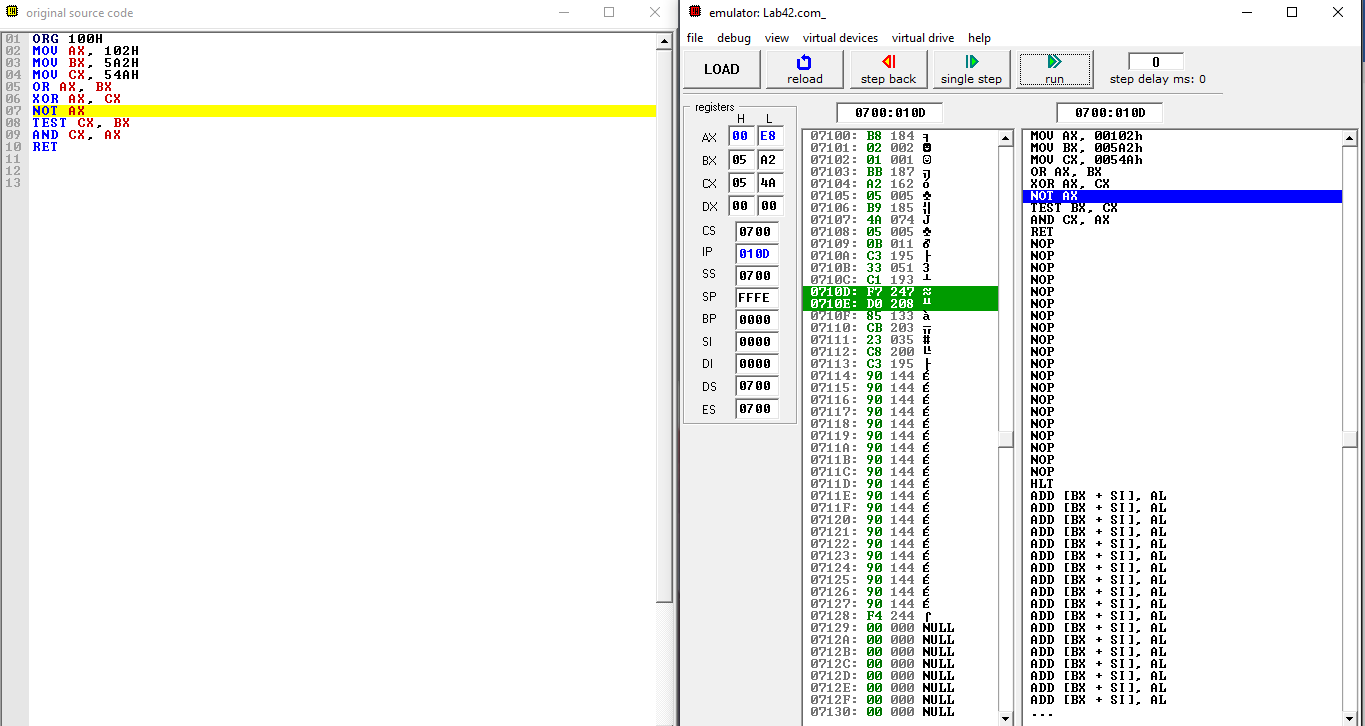


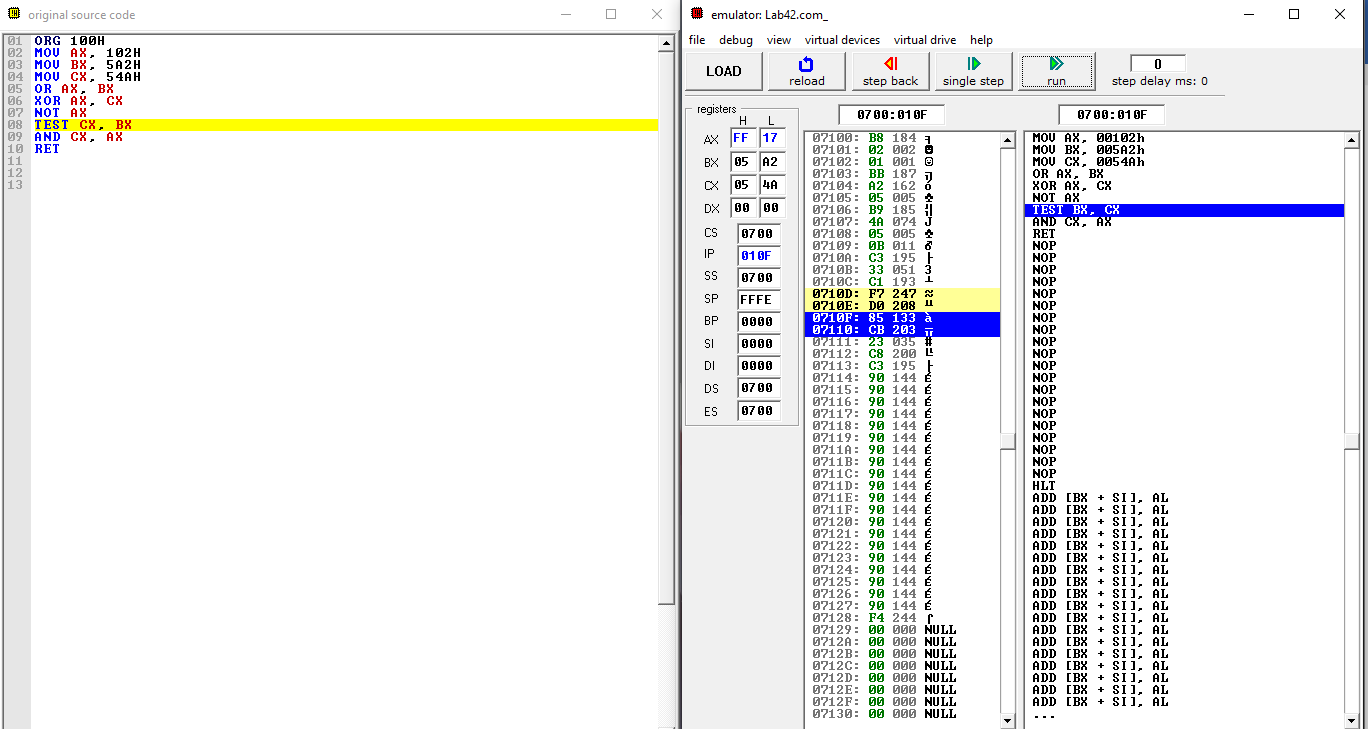


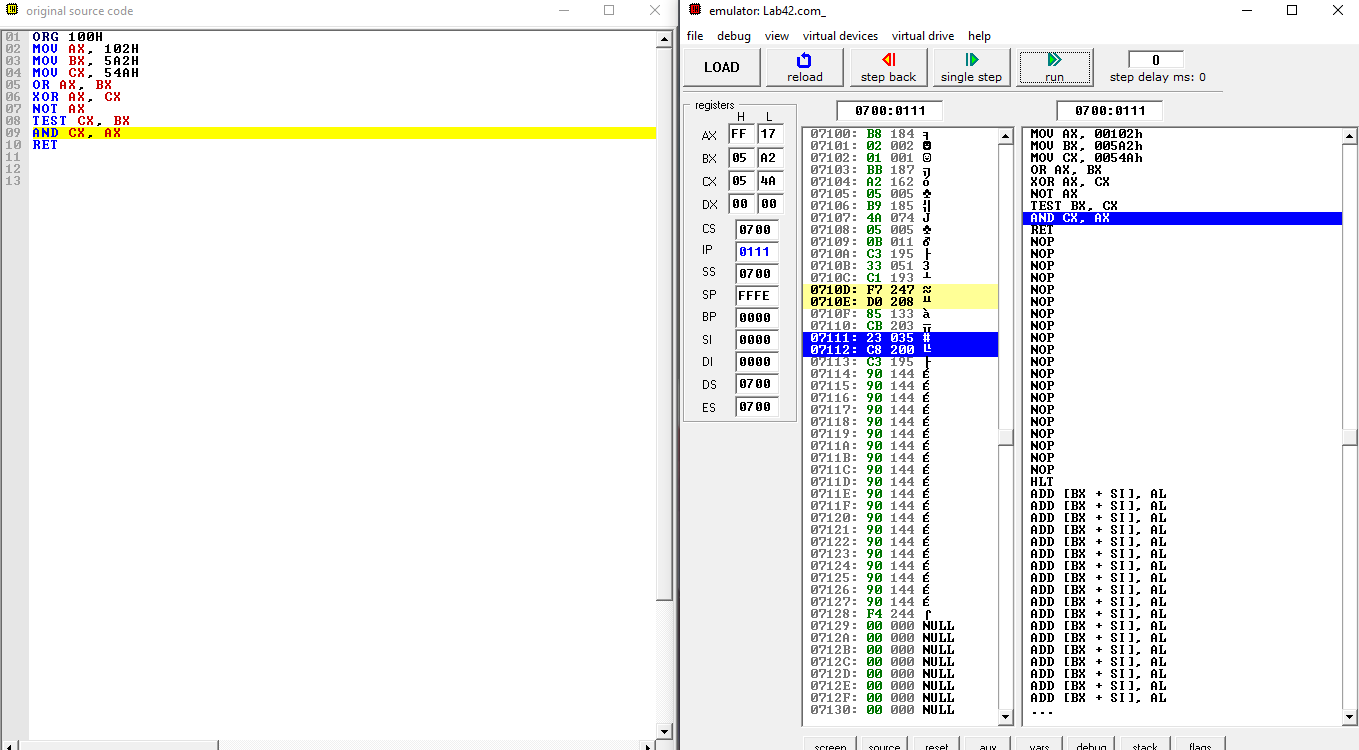


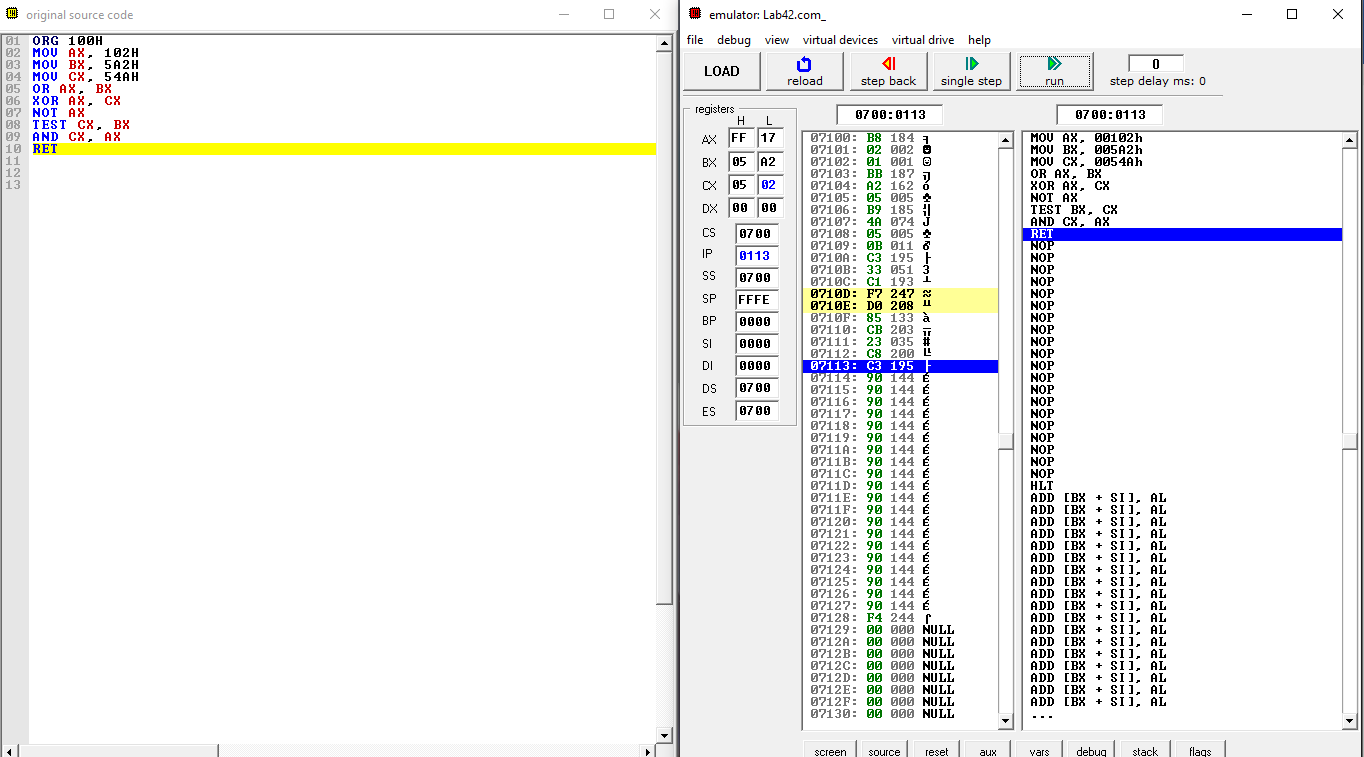








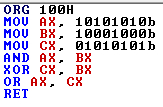


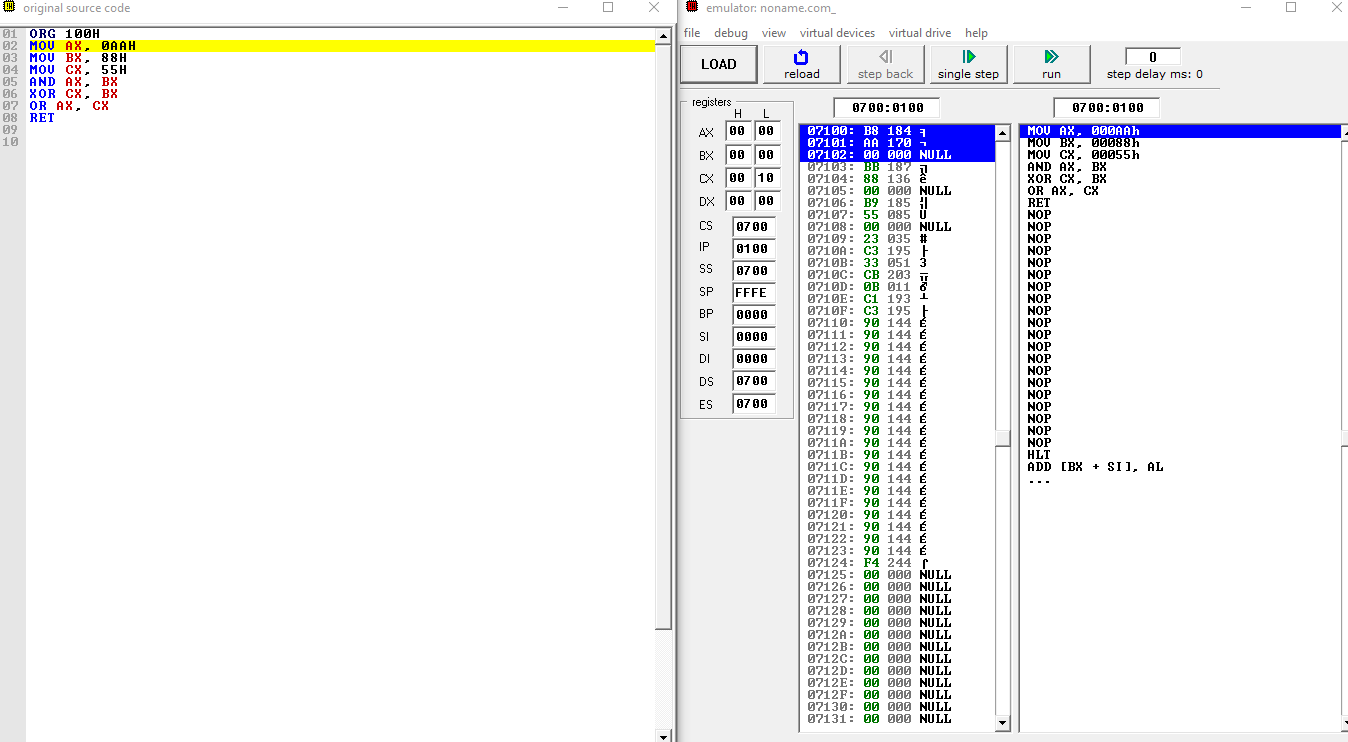


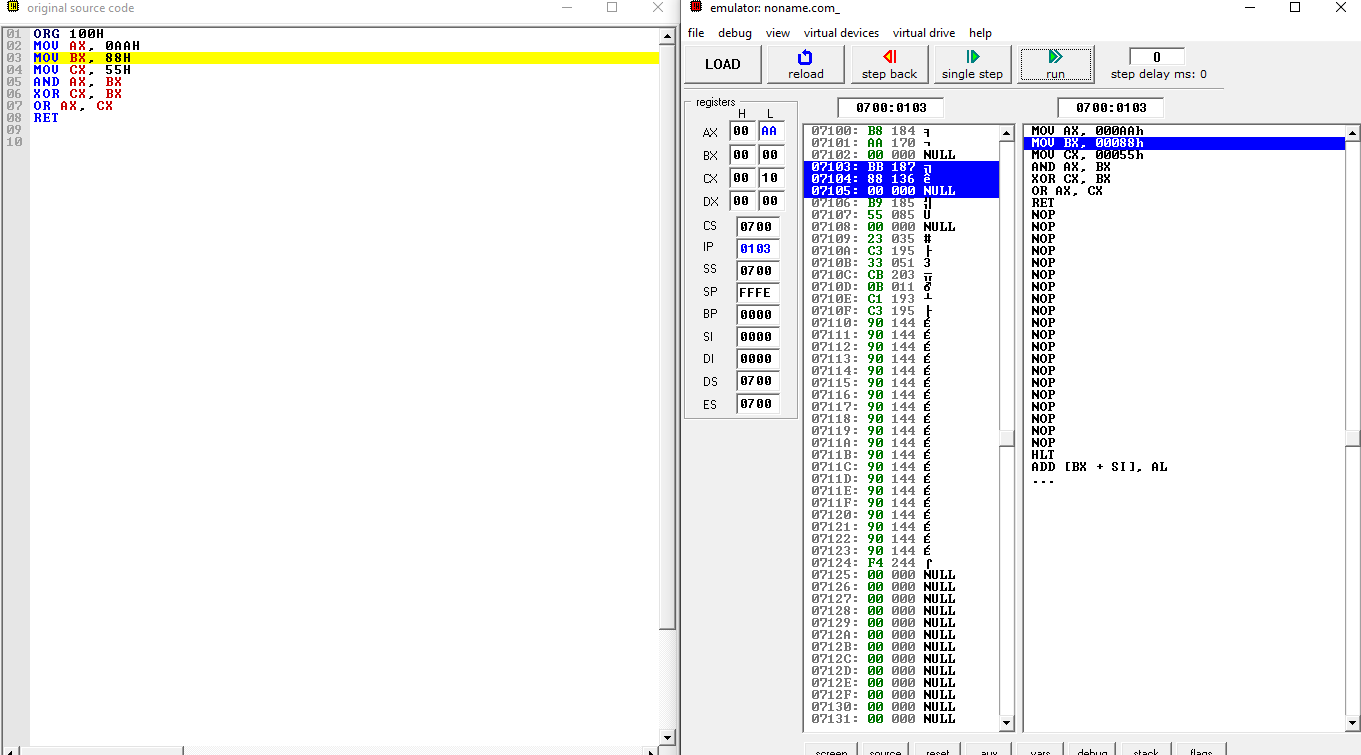
1. Write your own instruction based on the operation given in below table and record your result.

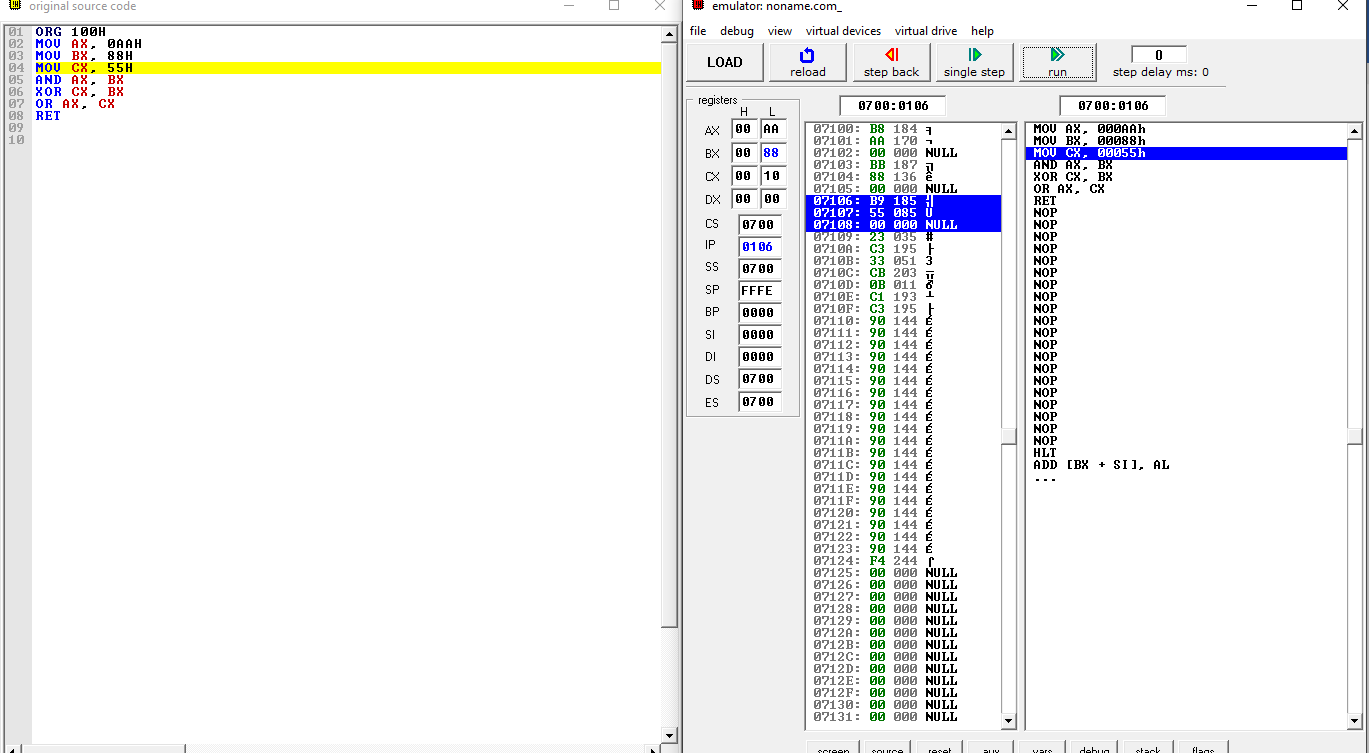
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Instruction | Register Content | | | | |
| AX | BX | CX | DX | IP |
| ORG 100H | 0000 | 0000 | 0010 | 0000 | 0100 |
| MOV AX,0AAH | 00AA | 0000 | 0010 | 0000 | 0103 |
| MOV BX,88H | 00AA | 0088 | 0010 | 0000 | 0106 |
| MOV CX,55H | 00AA | 0088 | 0055 | 0000 | 0109 |
| AND AX, BX | 0088 | 0088 | 0055 | 0000 | 010B |
| XOR CX, BX | 0088 | 0088 | 00DD | 0000 | 010D |
| OR AX, CX | 00DD | 0088 | 00DD | 0000 | 010F |
| RET | 00DD | 0088 | 00DD | 0000 | 0000 |

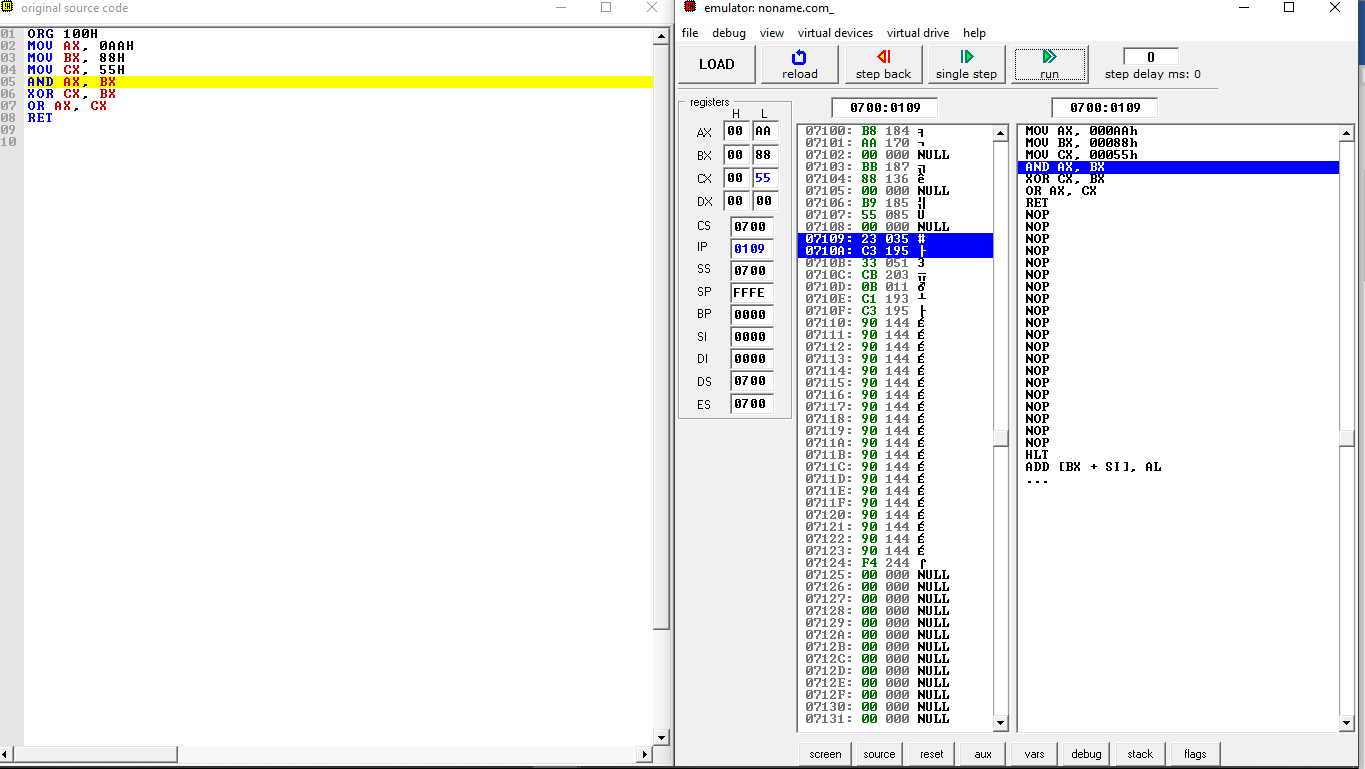
**Table 2**: Registers value

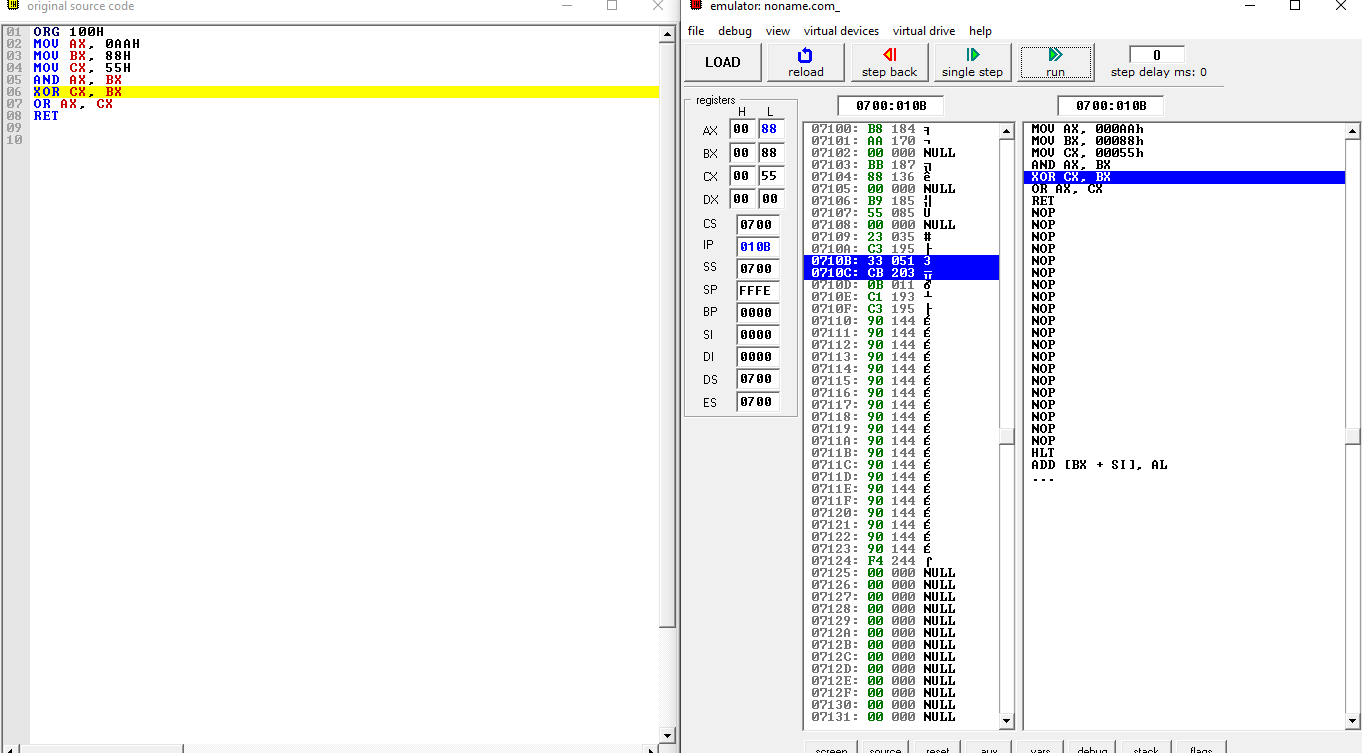


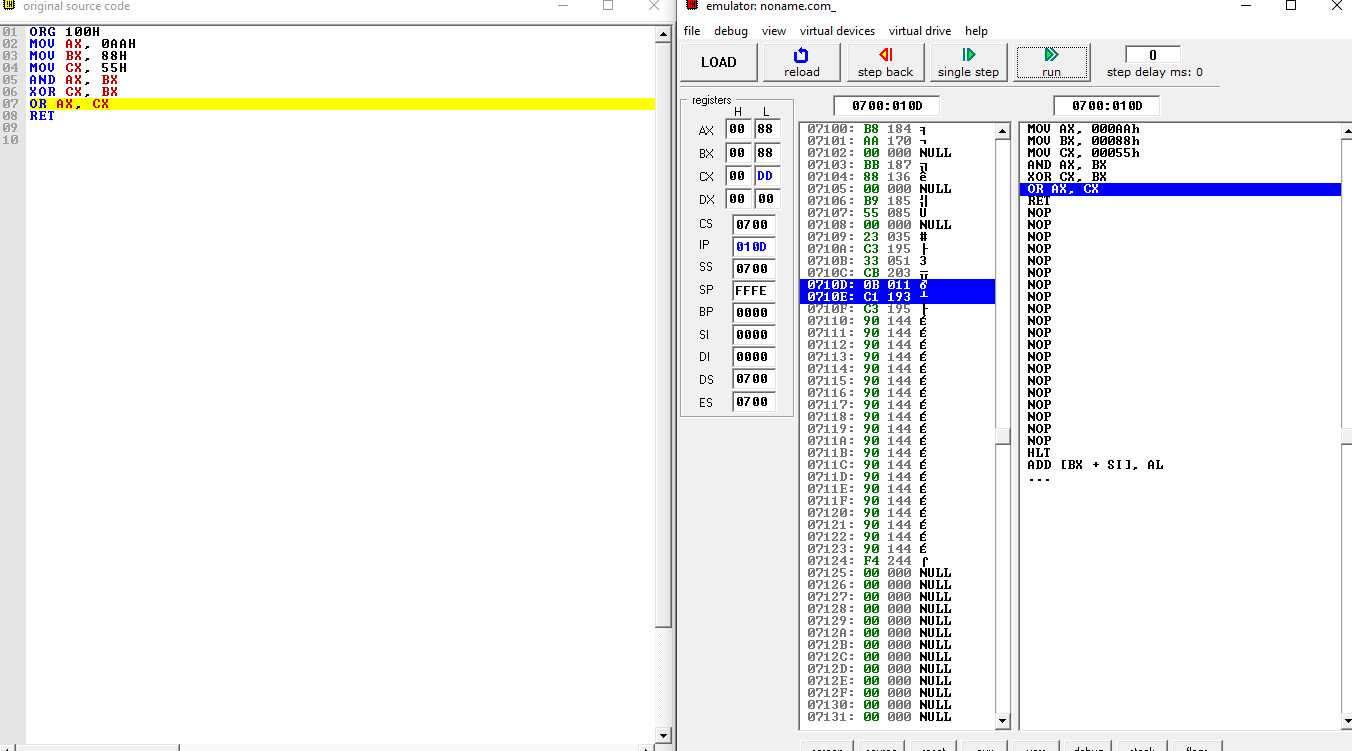


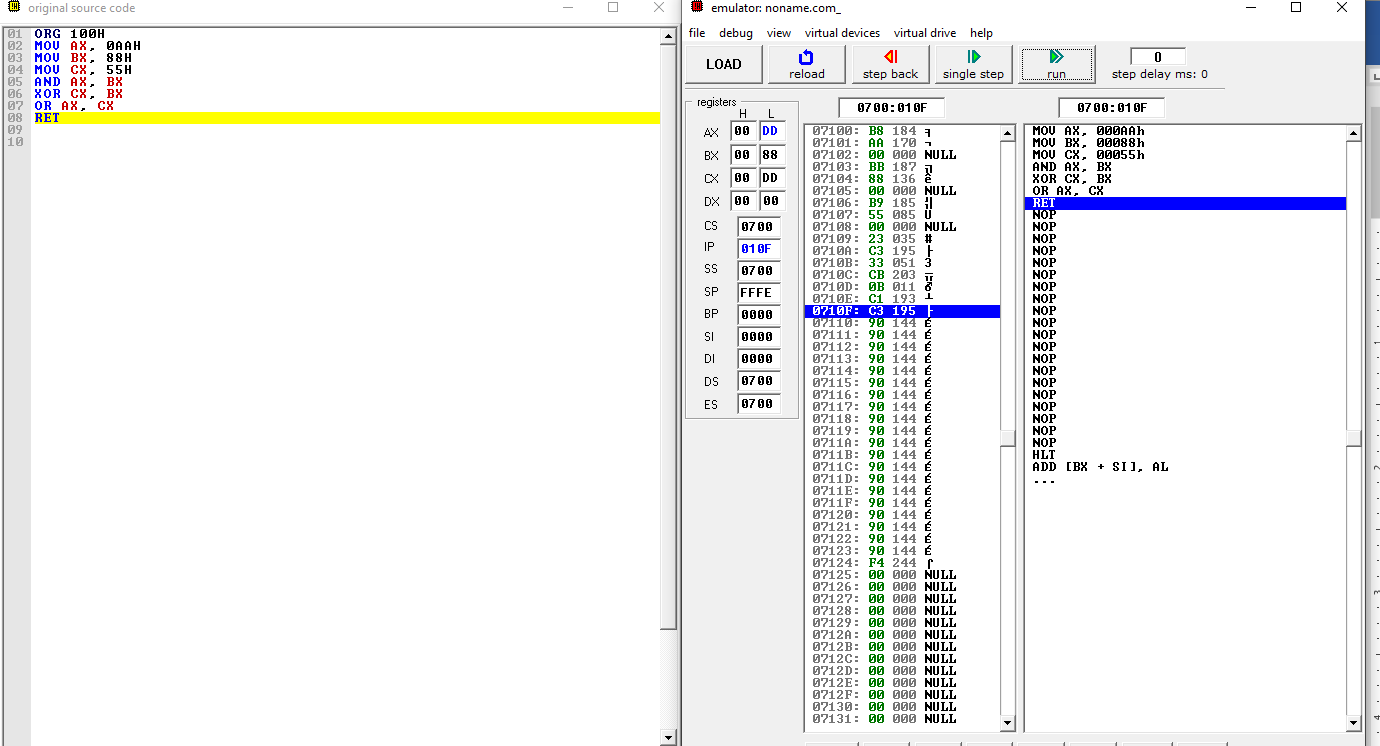












**5.2 Shift and Rotate command**

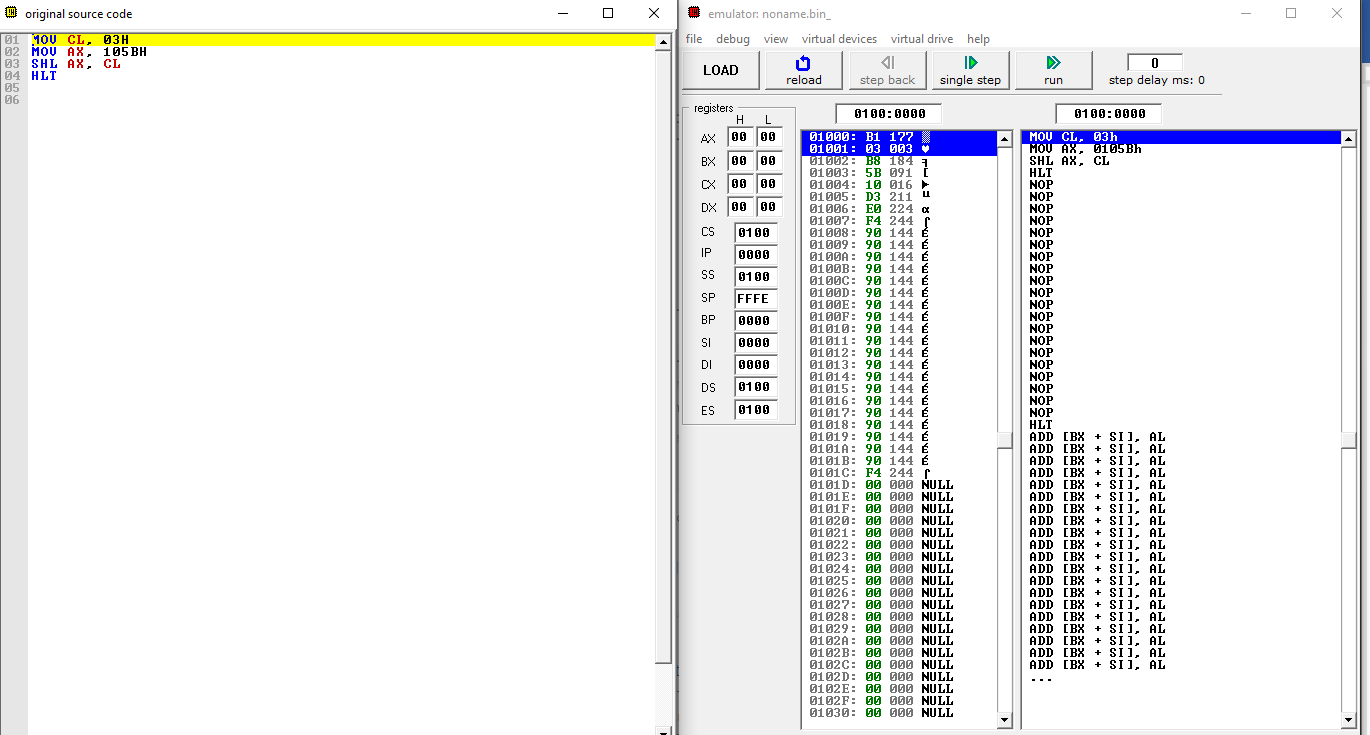
1. Obtain AX register value, write the previous value and present value in binary form. Write the type of operation for this instruction.

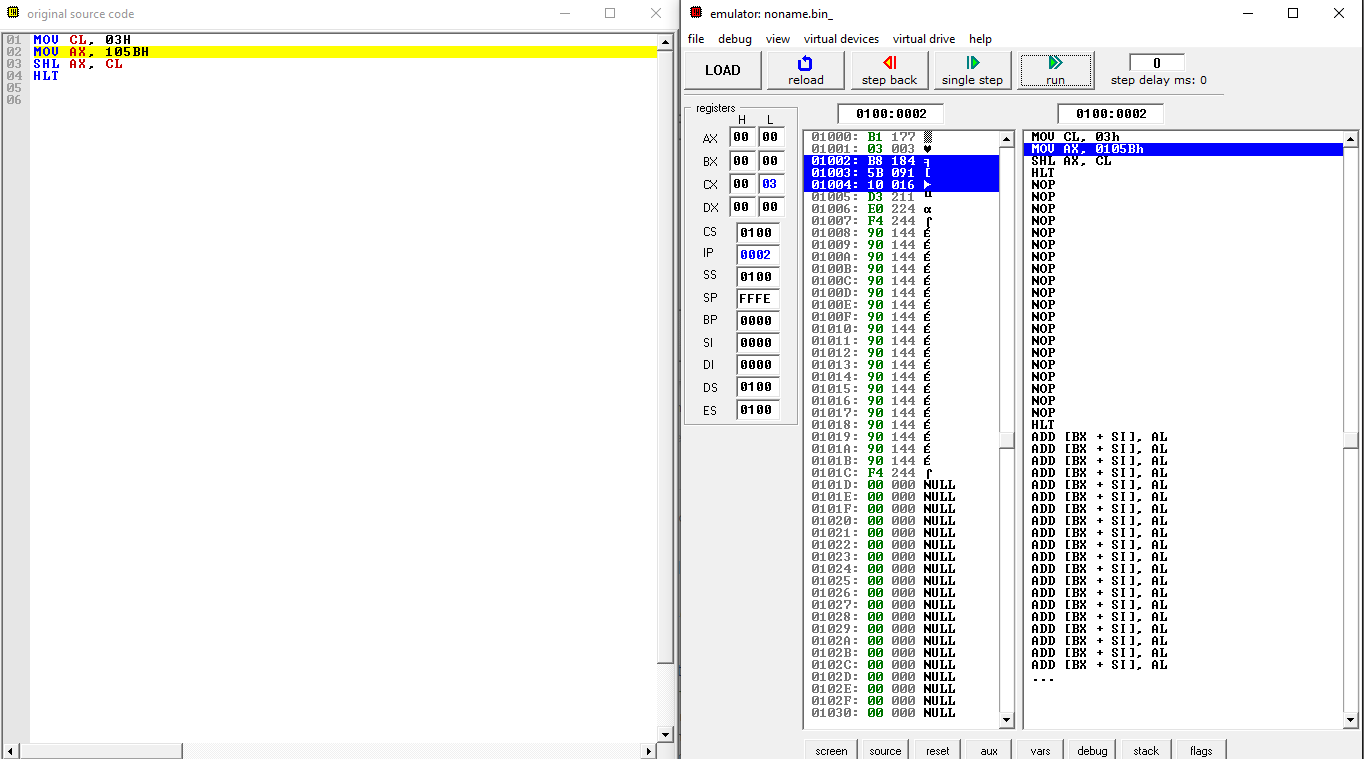
|  |  |  |  |
| --- | --- | --- | --- |
| Instruction | Register AX ( in binary ) | | Operation type |
| Previous value | Present value |
| 1. SHL AX,CL | 1000001011011b | 1000001011011000b | Logic(ShiftLeft) |
| 1. SAR AX,CL | 1000001011011b | 1000001011b | Shift Arithmetic Right |
| 1. ROR AX,CL | 1000001011011b | 110001000001011b | Rotate right |
| 1. RCL AX,CL | 1000001011011b | 1000001011011000b | Rotate Left with carry left |

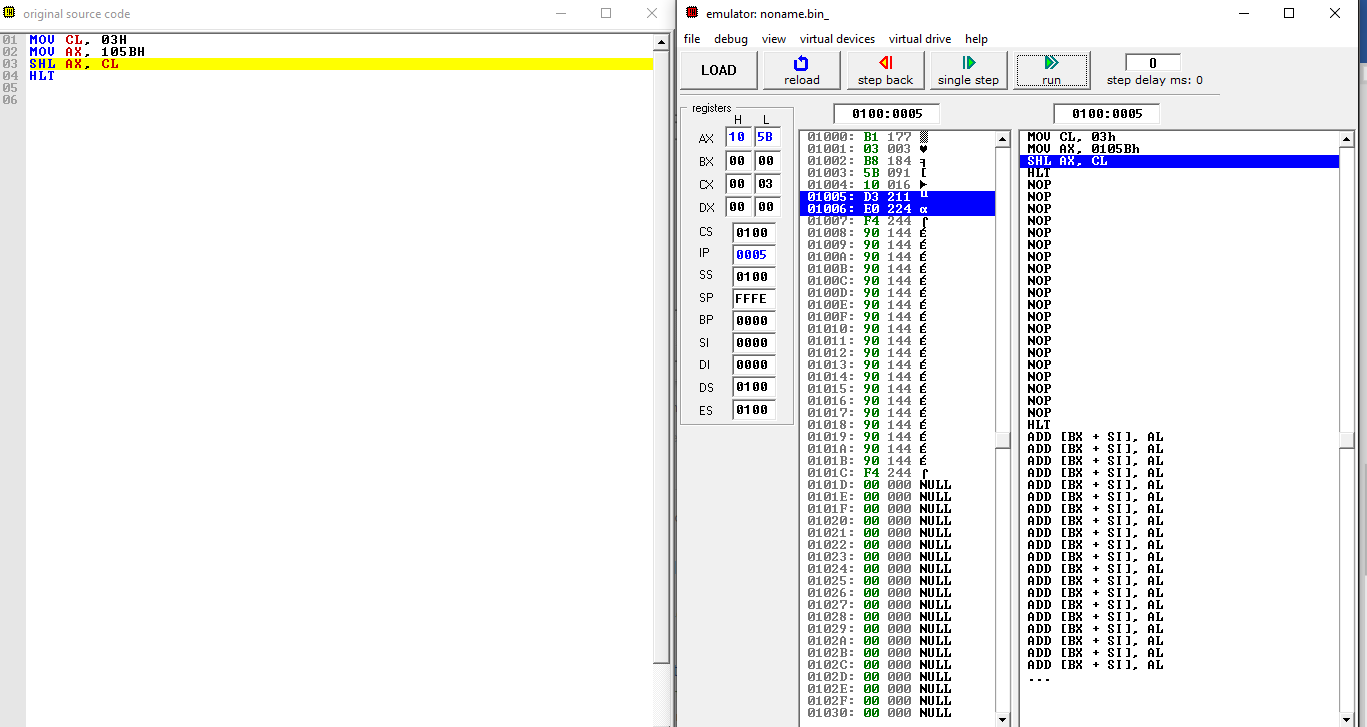
**Table 2**: Registers value

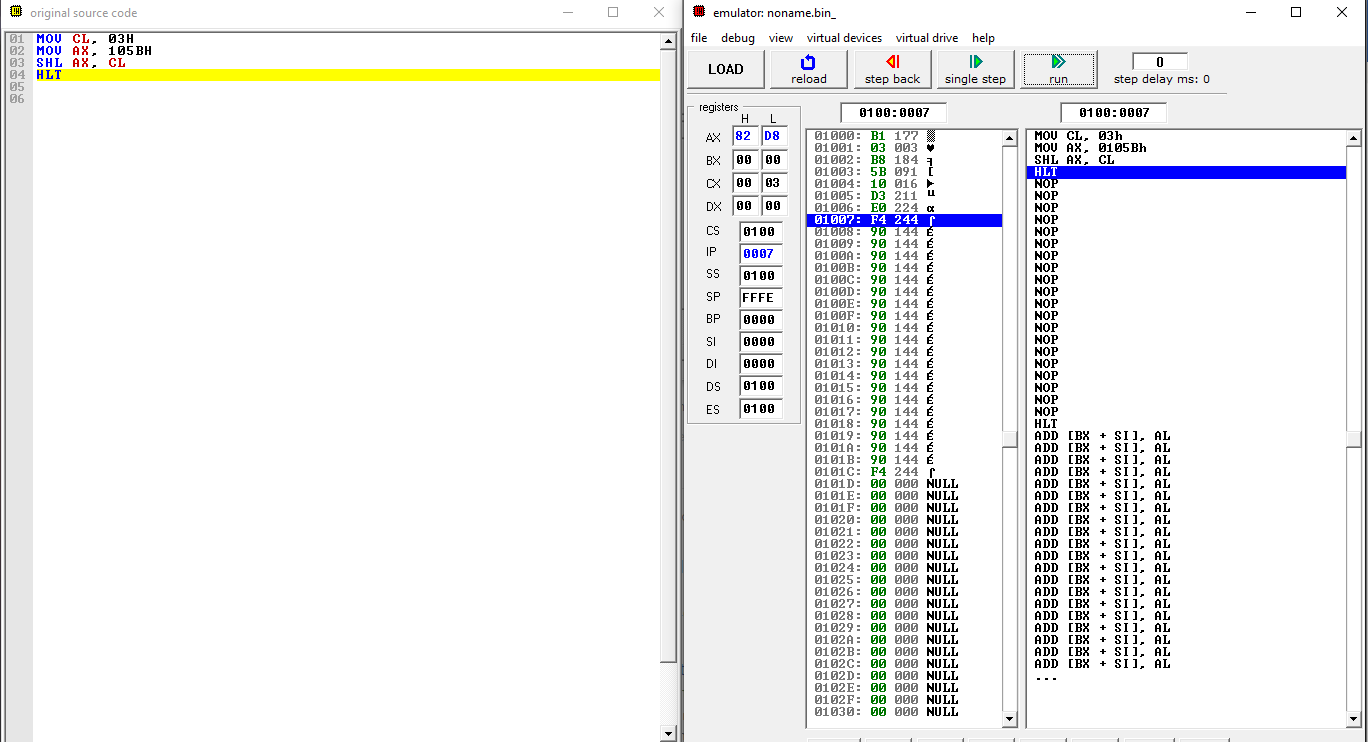
i)SHL



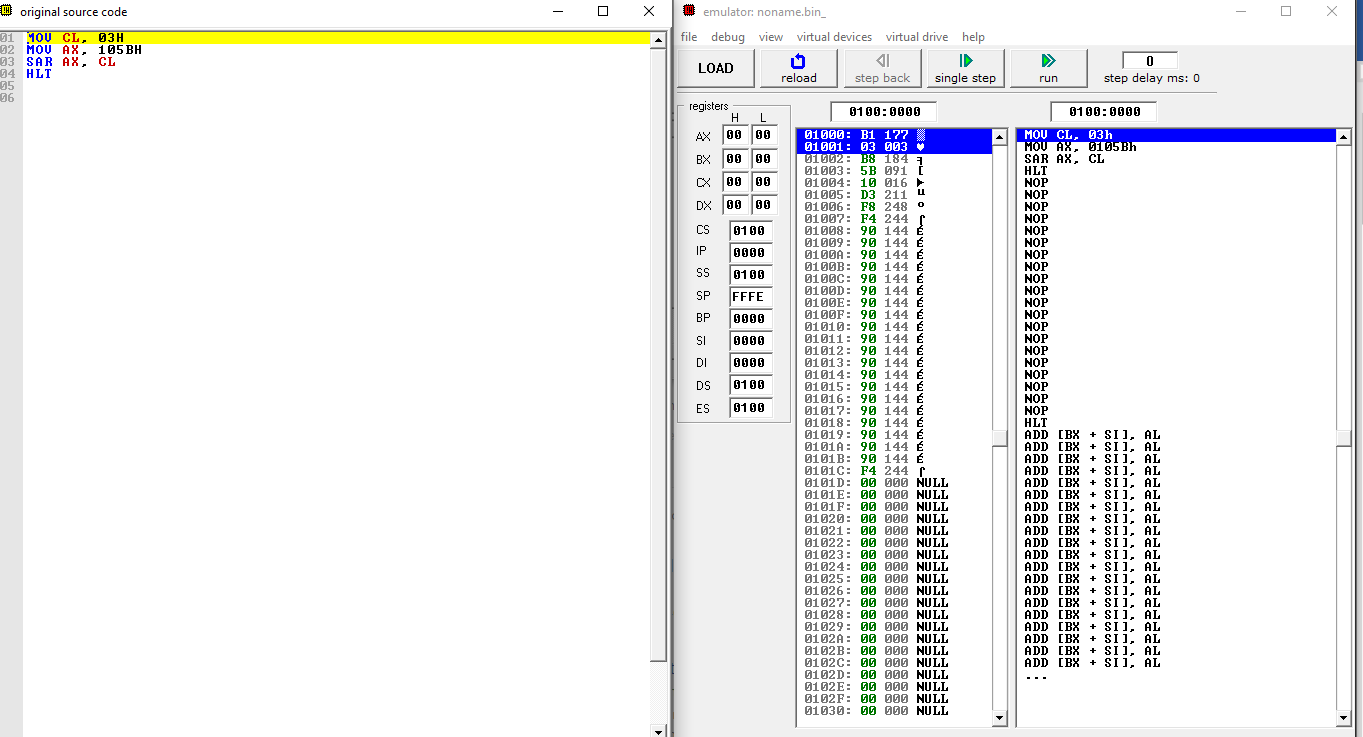


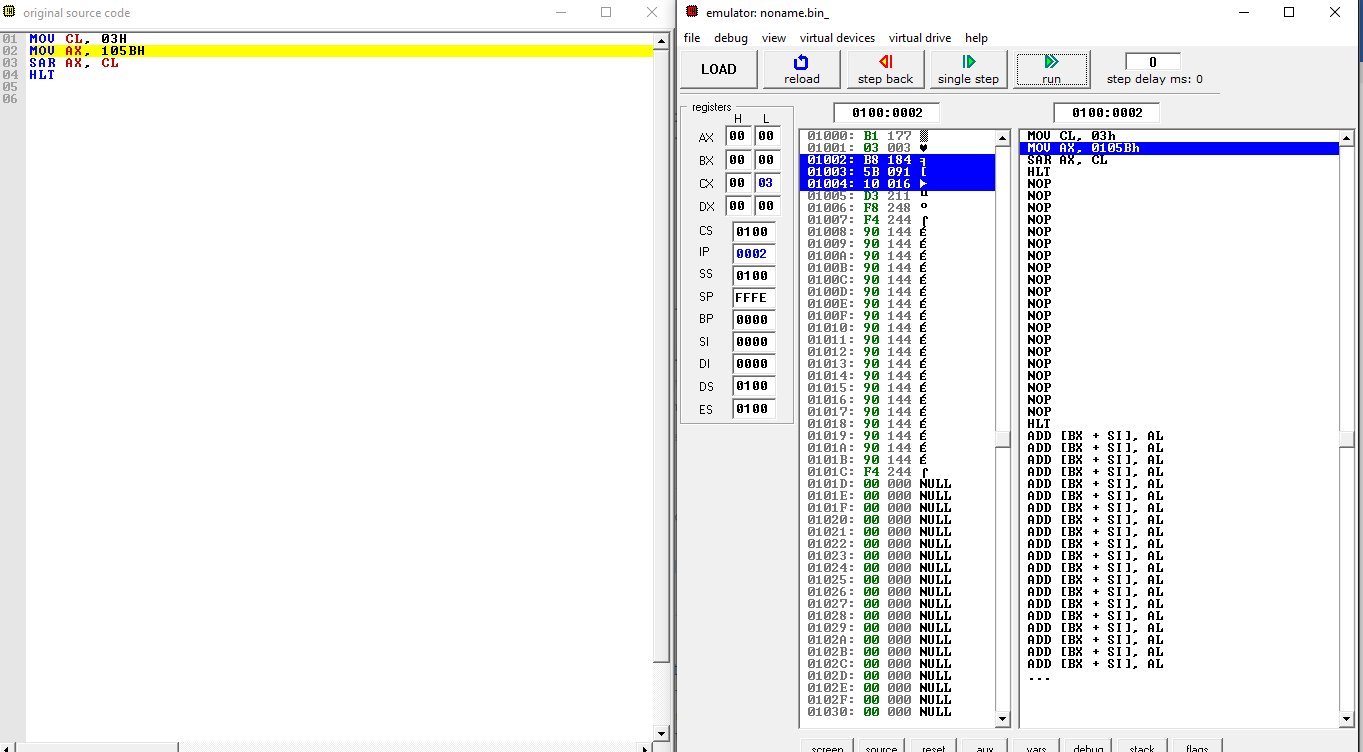


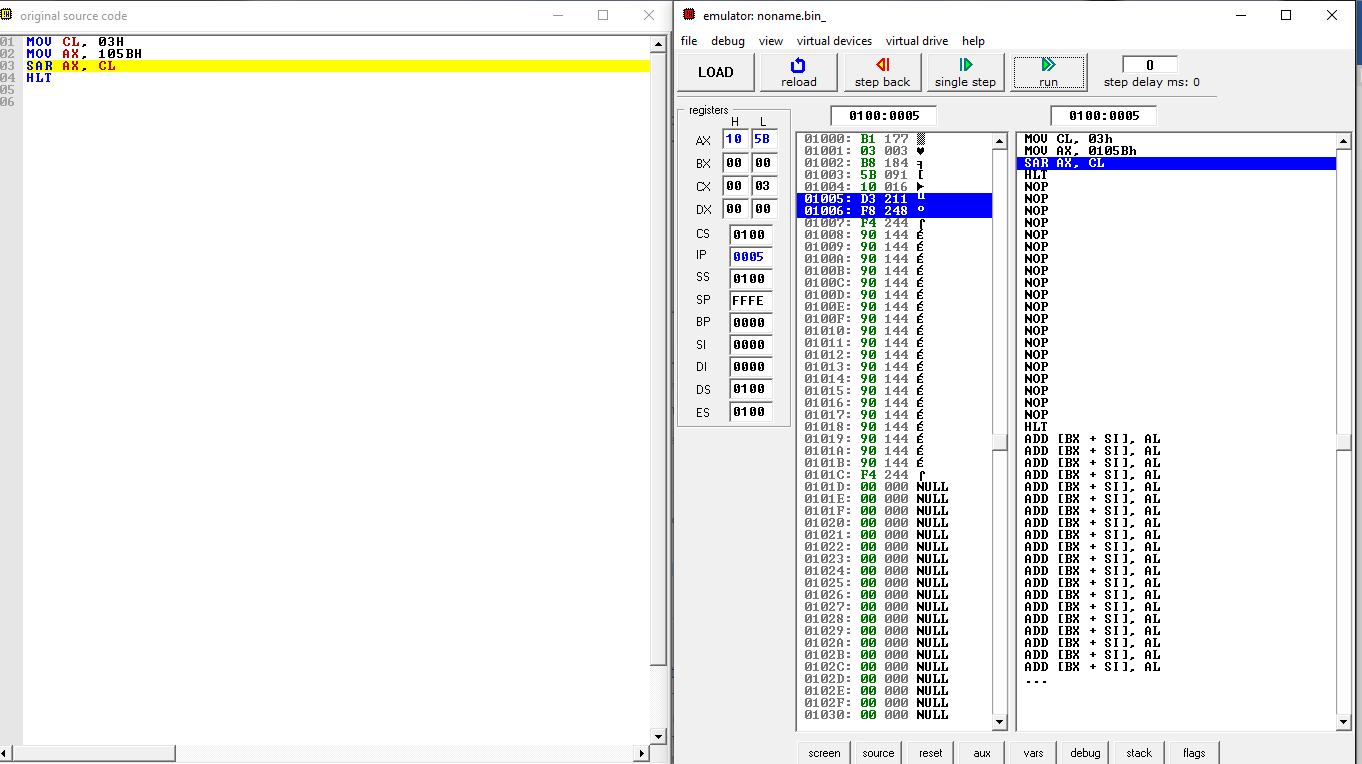


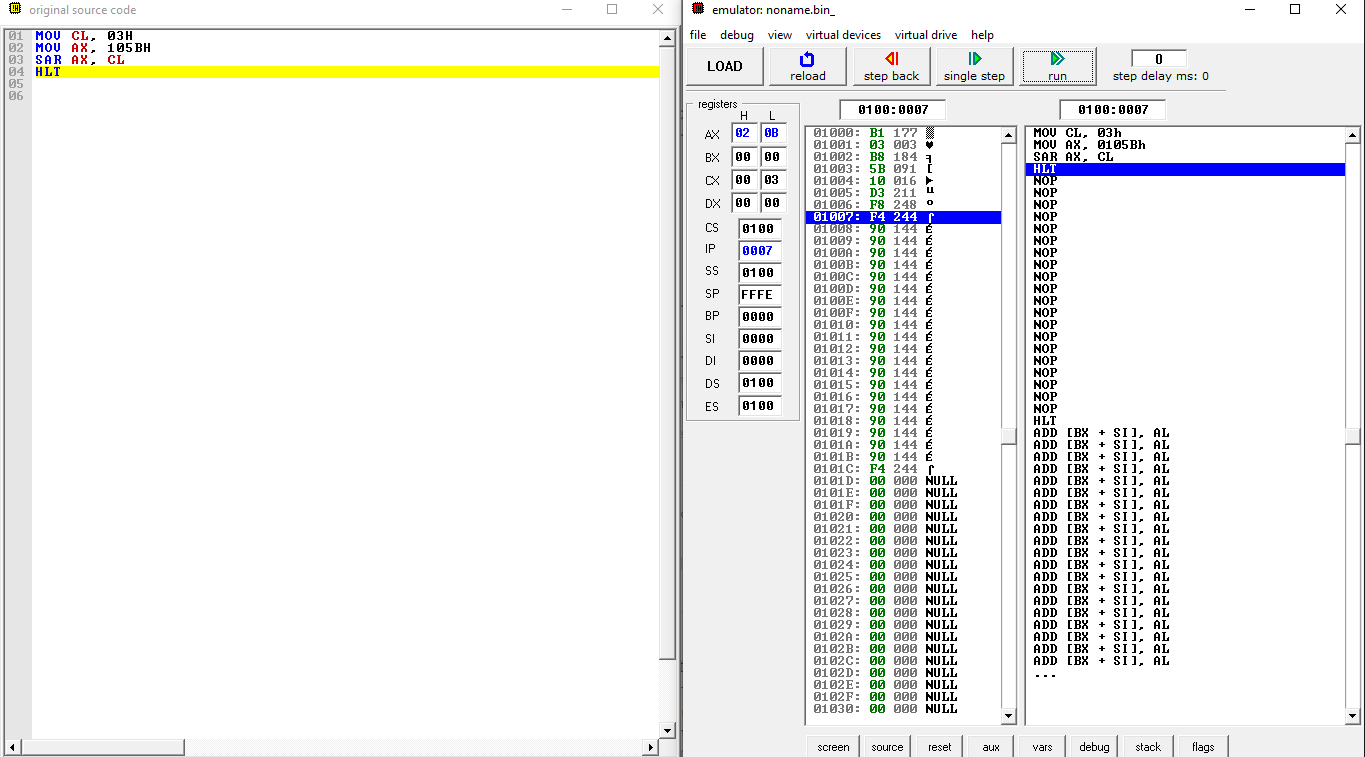


1. SAR

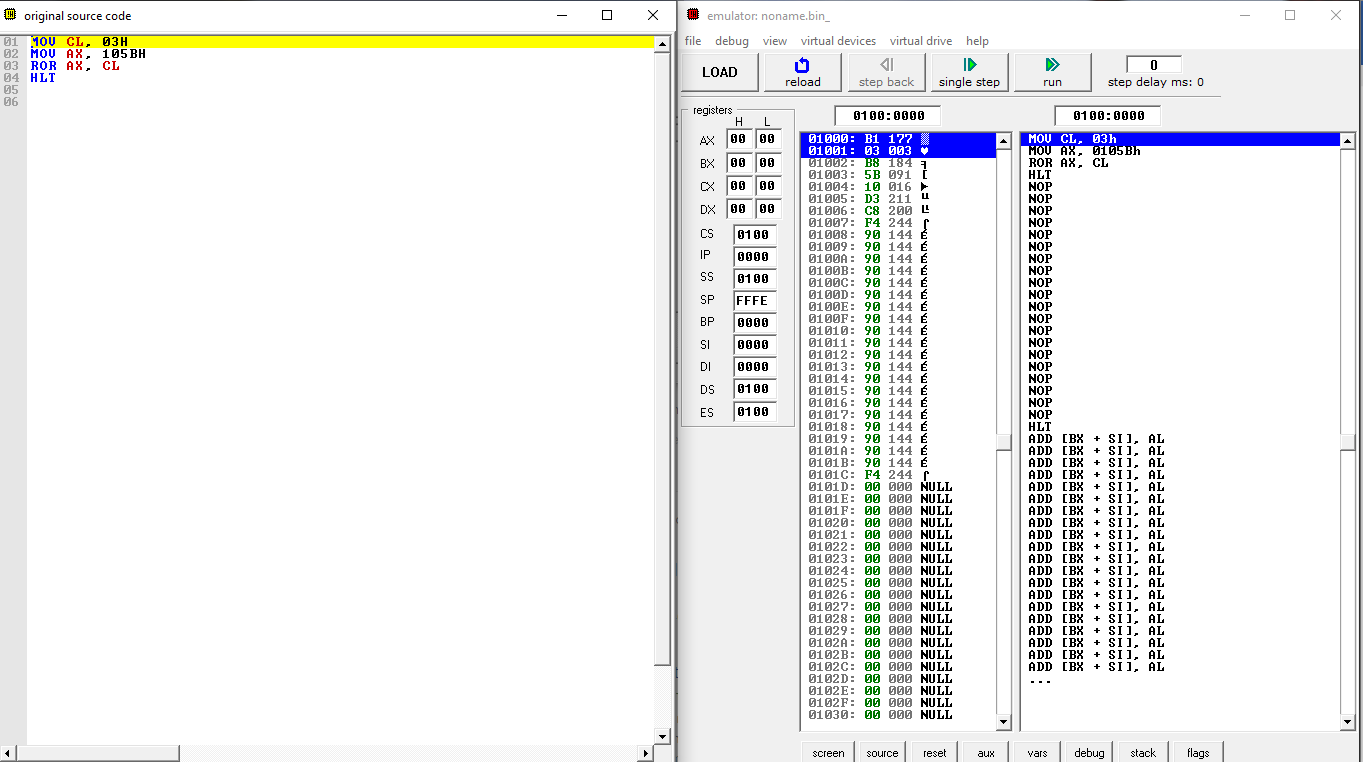


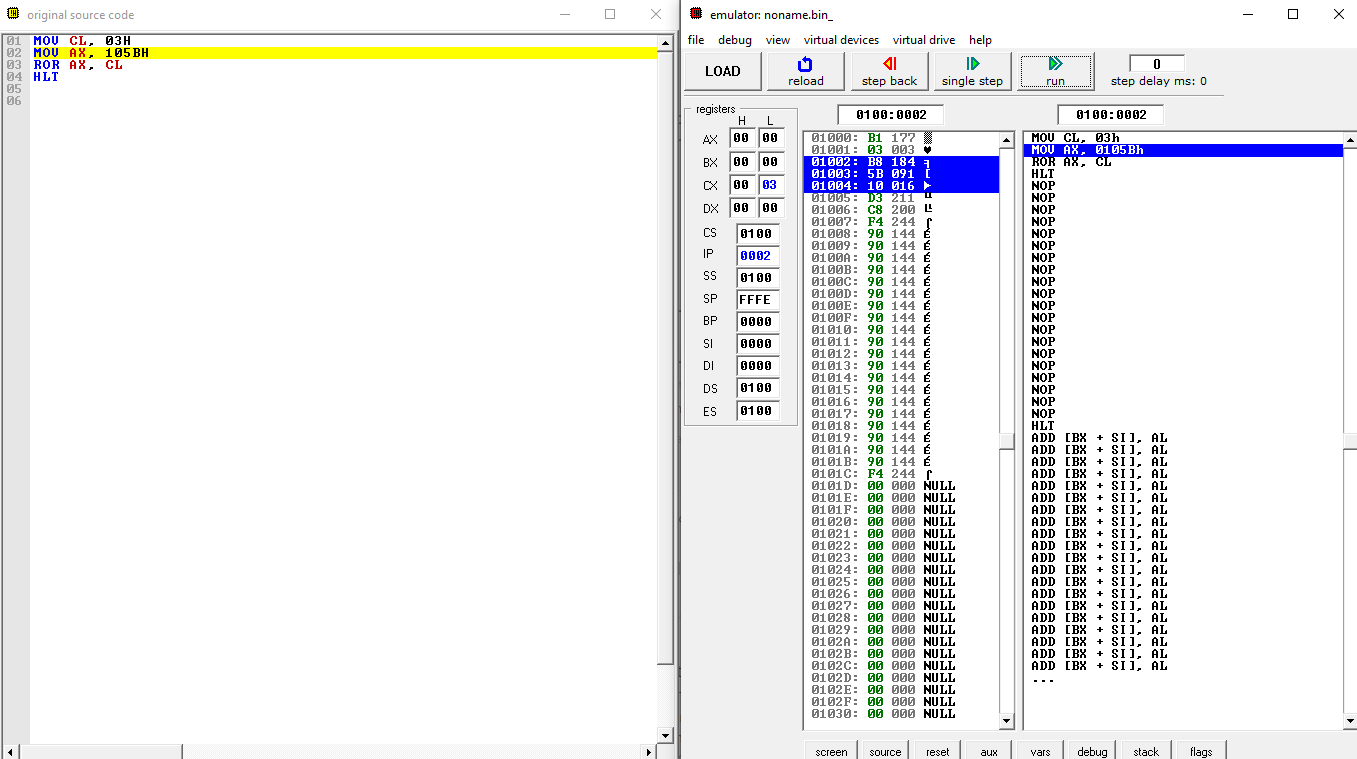


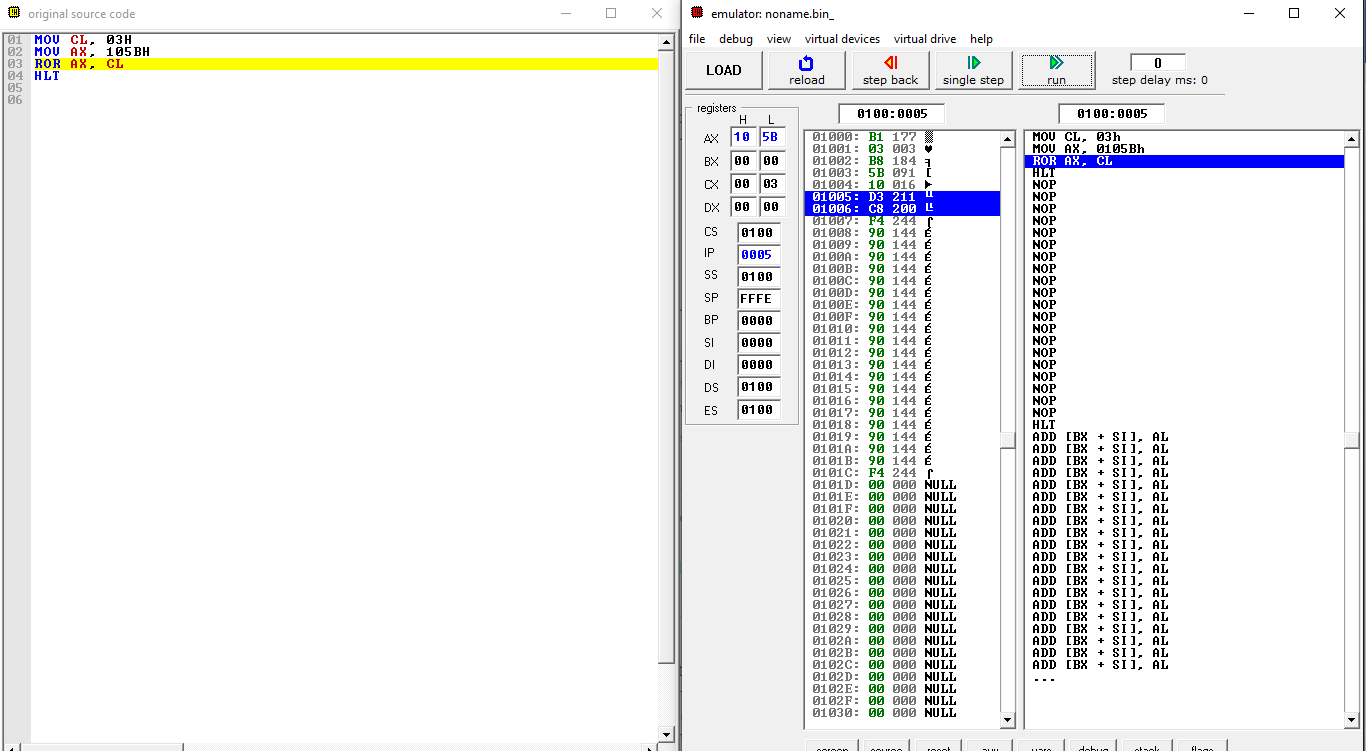


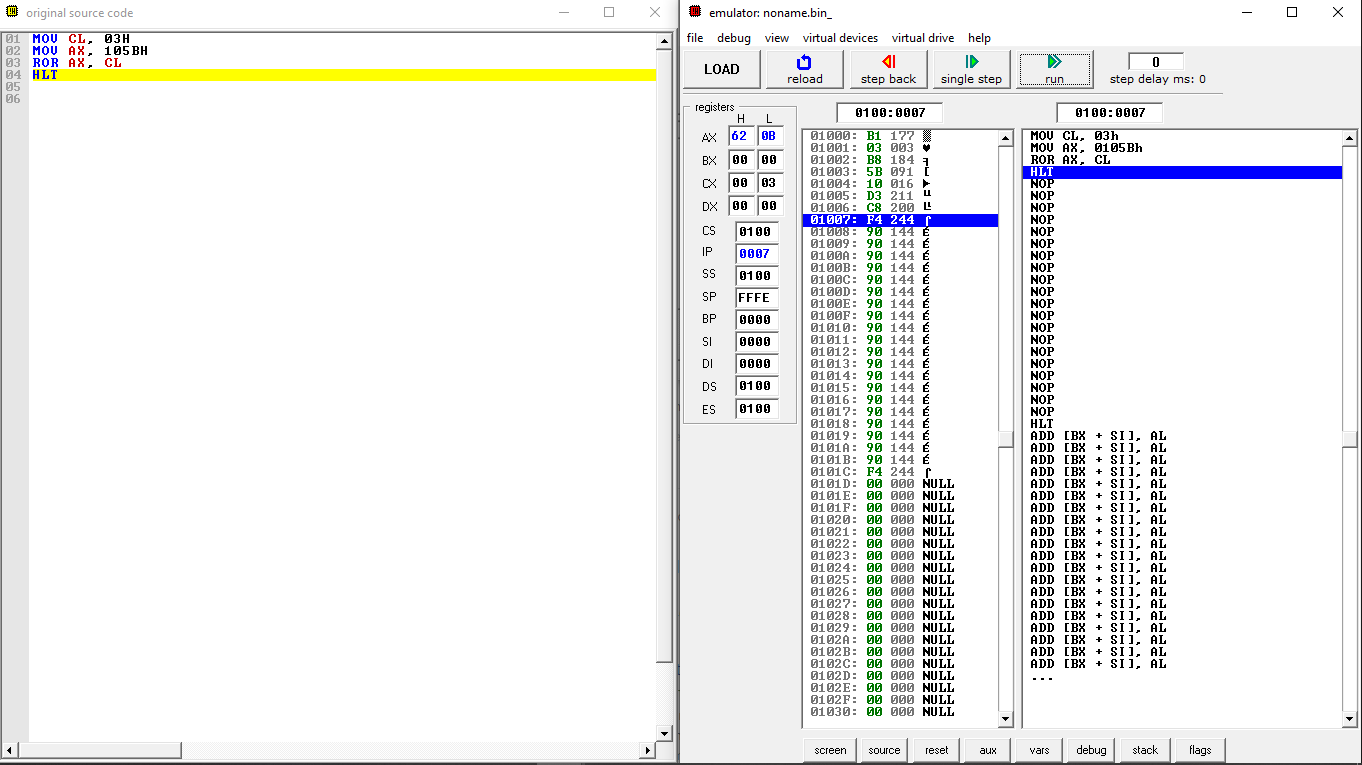


1. ROR

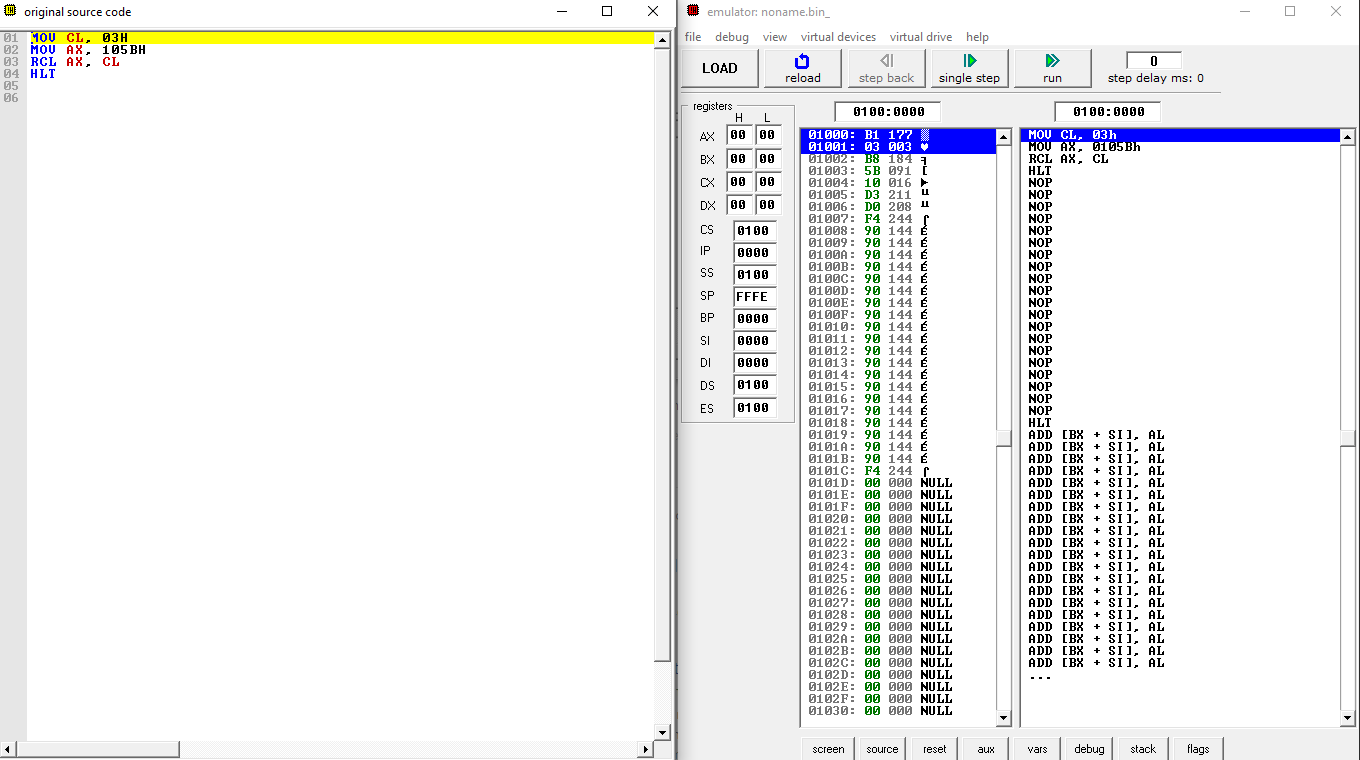


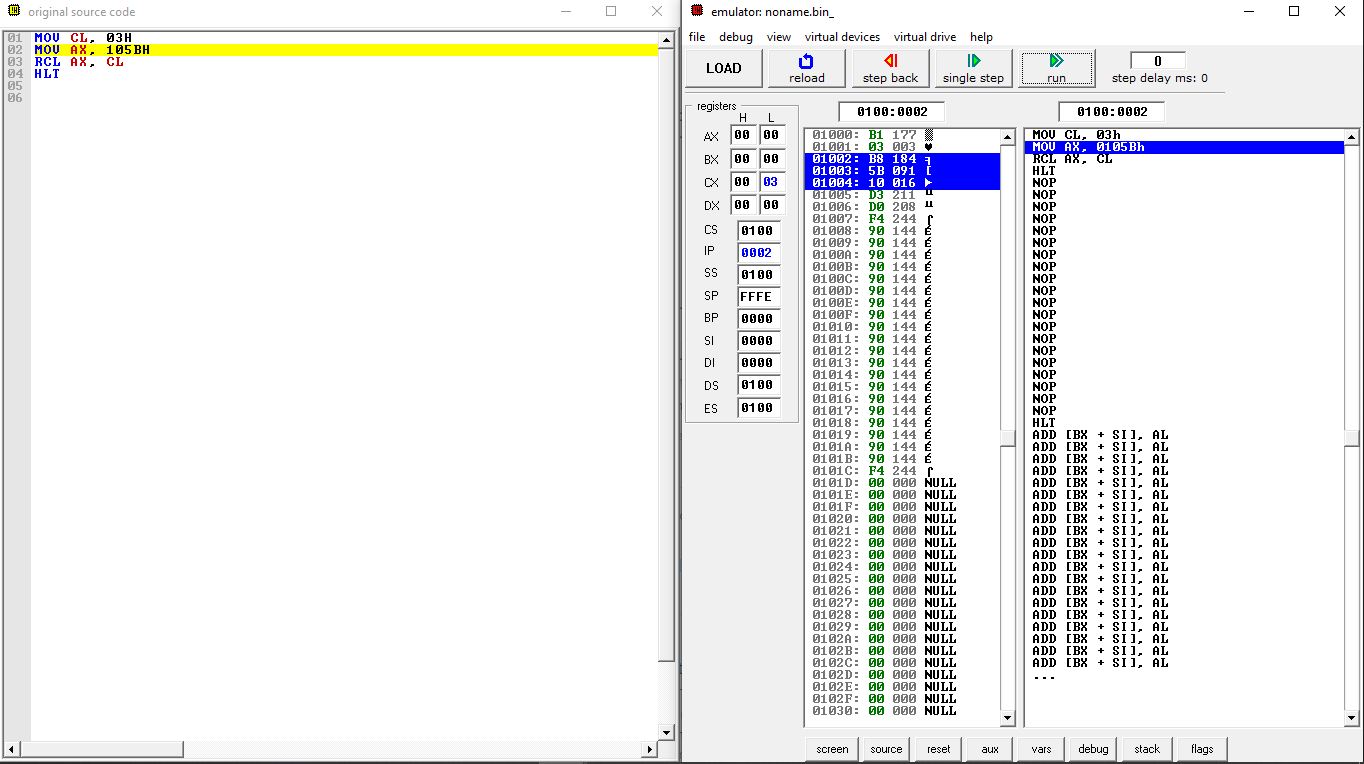


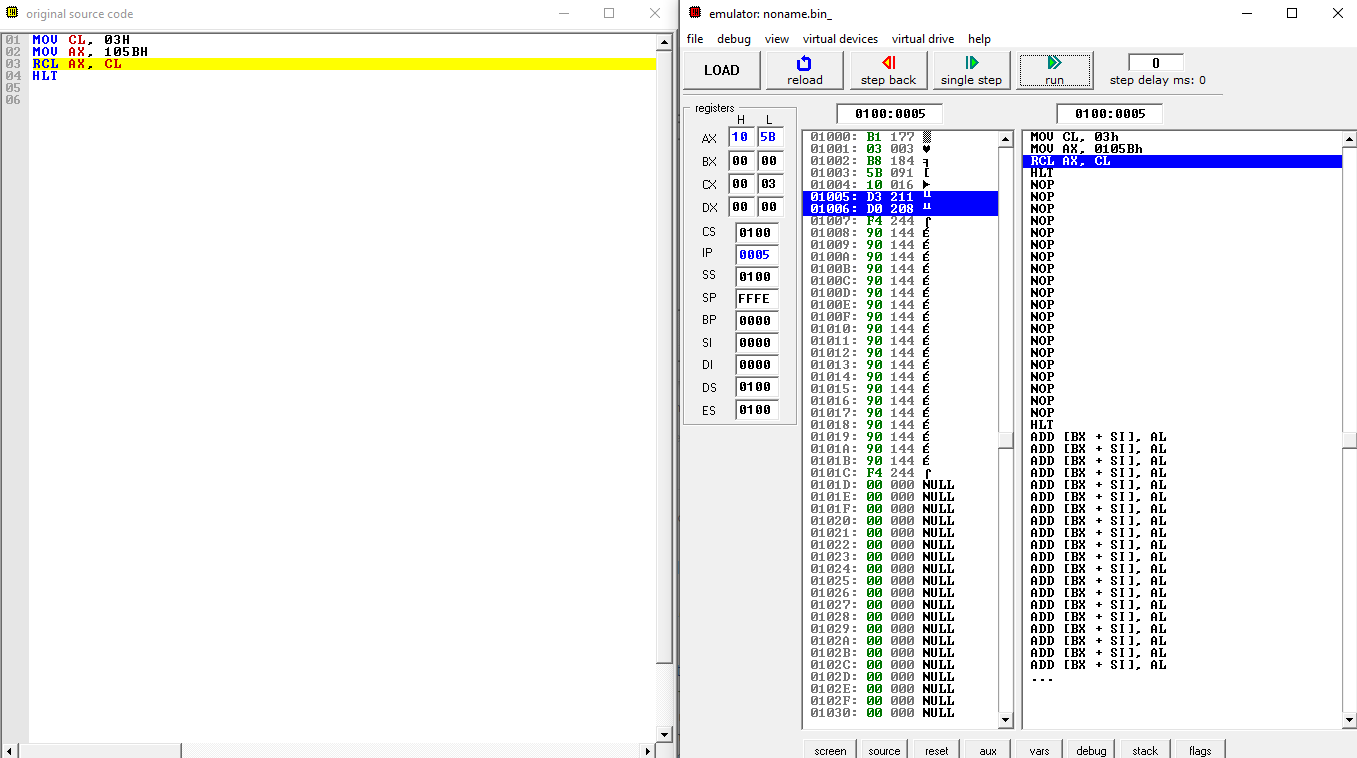


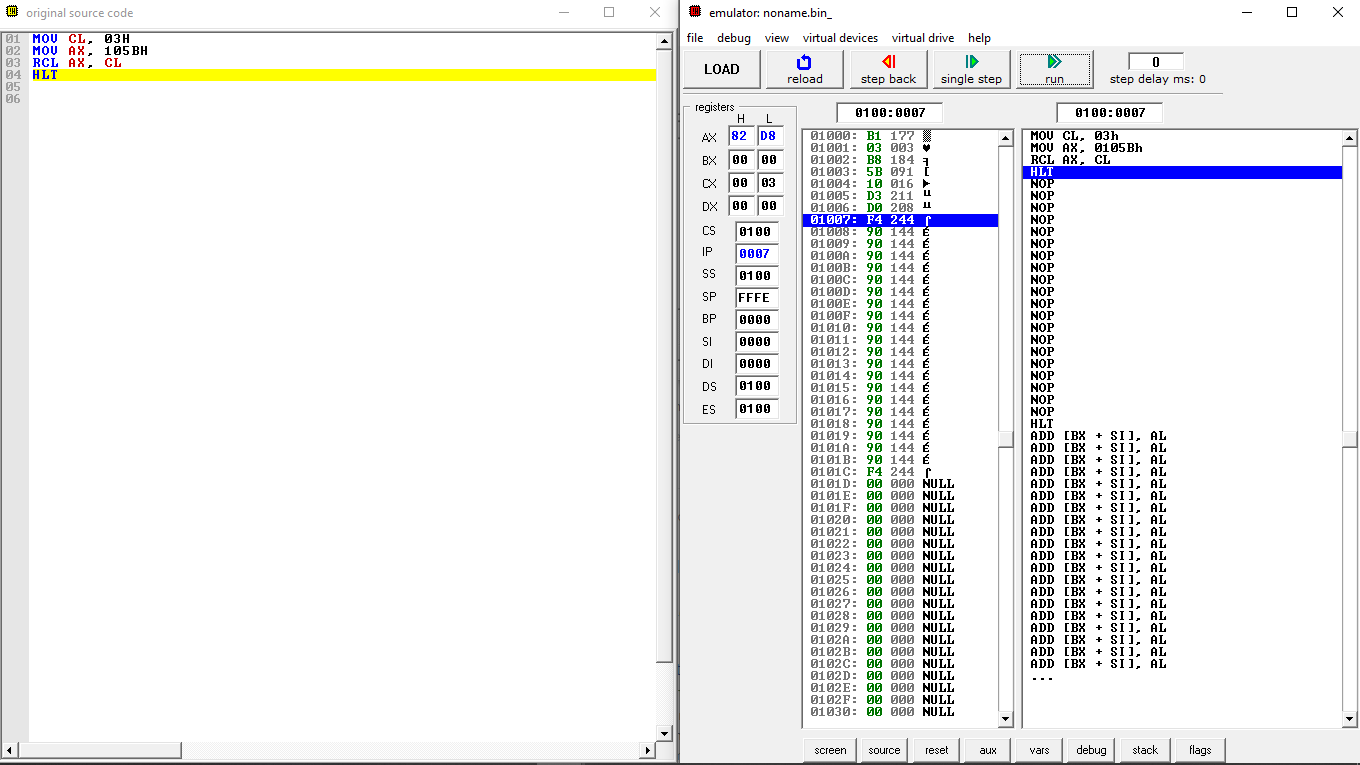


1. RCL









## DISCUSSION

* 1. **Logical Instructions**

Write the binary values and perform bit by bit operation **for every step with logical instructions (OR, XOR, NOT, TEST, AND)**. Explain how each operation works:

Example:

**(0102h) 0000 0001 0000 0010**

**OR (05A2h)0000 0101 1010 0010**

**= (05A2h)0000 0101 1010 0010**

**OR operator is a Boolean operator that returns a value of TRUE if either (or both) of its operands is TRUE.**

**1 OR 1 = 1**

**1 OR 0 = 1**

**0 OR 1 = 1**

**0 OR 0 = 0**

**OR GATE**

|  |  |  |
| --- | --- | --- |
| **INPUT** | | **OUTPUT** |
| A | B | A OR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**-**  The output of ‘or’ operation is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false." In other words, for the output to be 1, at least input one OR two must be 1.

**10101010**

**11011101 (or)**

**-------------**

11111111

**XOR GATE**

|  |  |  |
| --- | --- | --- |
| **INPUT** | | **OUTPUT** |
| A | B | A XOR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

The ‘xor’ operation make reverses the logic state. If the input is 1, then the output is 0. If the input is 0, then the output is 1.

**10101010**

**11011101 (XOR)**

**-------------**

1110111

**NOT GATE**

|  |  |
| --- | --- |
| Truth Table | |
| A | B |
| 0 | 1 |
| 1 | 0 |

The ‘not’ operation make reverses the logic state. If the input is 1, then the output is 0. If the input is 0, then the output is 1.

**10101010**

-----------(NOT)

**01010101**

**TEST INSTRUCTION**

**-** the **TEST** [instruction](https://en.wikipedia.org/wiki/Instruction_(computing)) performs a [bitwise AND](https://en.wikipedia.org/wiki/Bitwise_AND) on two [operands](https://en.wikipedia.org/wiki/Operand). The [flags](https://en.wikipedia.org/wiki/FLAGS_register) [SF](https://en.wikipedia.org/wiki/Sign_flag), [ZF](https://en.wikipedia.org/wiki/Zero_flag), [PF](https://en.wikipedia.org/wiki/Parity_flag) are modified while the result of the [AND](https://en.wikipedia.org/wiki/Bitwise_AND) is discarded.

**-** TEST computes the bit-wise logical AND of first operand (source 1 operand) and the second operand (source 2 operand) and sets the SF, ZF, and PF status flags according to the result. The result is then discarded.

**AND GATE**

The output of ‘and’ operation is "true" when both inputs are "true." Otherwise, the output is "false." In other words, the output is 1 only when both inputs one AND two are 1

|  |  |  |
| --- | --- | --- |
| **INPUT** | | **OUTPUT** |
| A | B | A AND B |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**10101010**

**11011101 (AND)**

**-------------**

10001000

**6.2 Shift and Rotate command**

Explain the Shift and Rotate command used in each questions in 4.2.

Shift Command

* Shifting means to move bits right and left inside an operand.

Rotate Command

* Rotate instructions are similar to shift instructions, ecept that rotate instructions are circular, with the bits shifted out one end returning on the other end. Rotates can be to the left or right. Rotates can also employ an extend bit for multi-precision rotates.

Explain the differences between **Shift Logical** and **Shift Arithmetic** as well as **Rotate** and **Rotate Through Carry**.

Differences between Shift Logical and Shift Arithmetic

* In arithmetic shift preserve sign bit, whereas logical shift can not preserve sign bit.
* Arithmetic shift perform multiplication and division operation, whereas Logical shift perform only multiplication operation.

Differences between Rotate and Rotate Through Carry

* In Rotate operation, sometimes called *rotate no carry*, the bits are "rotated" as if the left and right ends of the register were joined. The value that is shifted into the right during a left-shift is whatever value was shifted out on the left, and vice versa for a right-shift operation.
* *Rotate through carry* is a variant of the rotate operation, where the bit that is shifted in (on either end) is the old value of the carry flag, and the bit that is shifted out (on the other end) becomes the new value of the carry flag.

## CONCLUSION

At the end of this lab,we learnt to make a program for XOR,AND,OR,NOT operation and also TEST instruction.We also understand how the operation and instruction change the original value that assign to the AX register.Next, we learnt shift and rotate command that change the value of number in binary.The differences between Shift Logical and Shift Arithmetic as well as Rotate and Rotate Through Carry,make us more understand about the function of this command towards the program that we create.Finally,Arithmetic Logic Unit is important to manipulate data(eg.AND,OR,XOR) and also a basic of building block of many types of computing circuits