|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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.** | **AMNI NAJIHAH BIN ABD AZIZ** | | | | **B081910081** |
| **2.** | **AMIR FIRDAUS BIN MUHAMMAD FADZLI** | | | | **B081910238** |
| **3.** |  | | | |  |
| **PROGRAMME** | | **1 BEEC** | | | |
| **SECTION / GROUP** | | **1 / 2** | | | |
| **DATE** | | **12 / 05 / 2020** | | | |
| **NAME OF INSTRUCTOR(S)** | | 1. Noor Mohd Ariff | | | |
|  | | | |
| **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



MOV CL, 03H

MOV AX, 105BH

ROR AX, CL

HLT



MOV CL, 03H

MOV AX, 105BH

RCL AX, CL

HLT

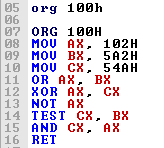
## RESULT

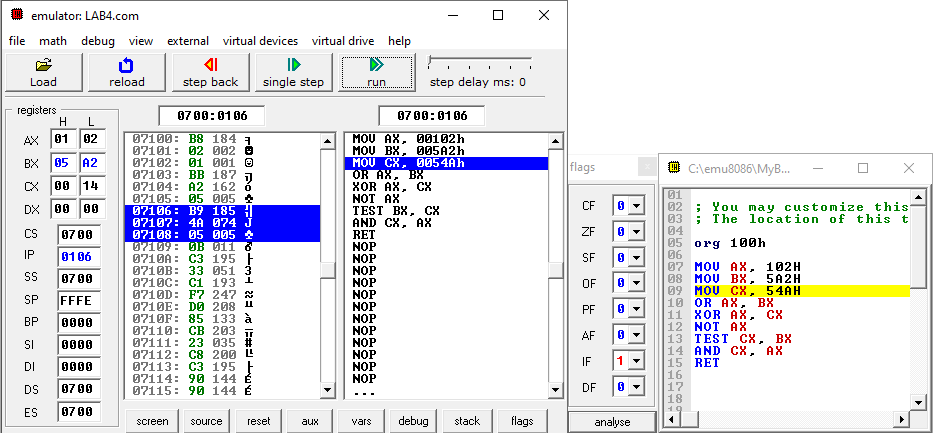
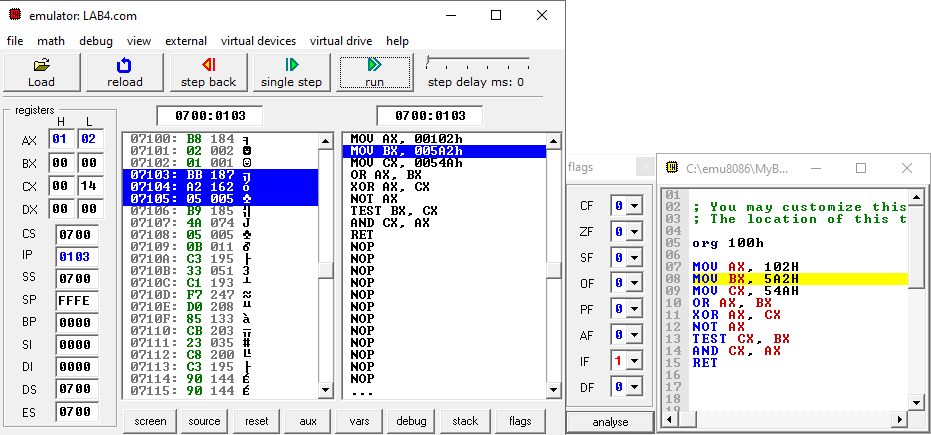
**5.1 Logical Instructions**

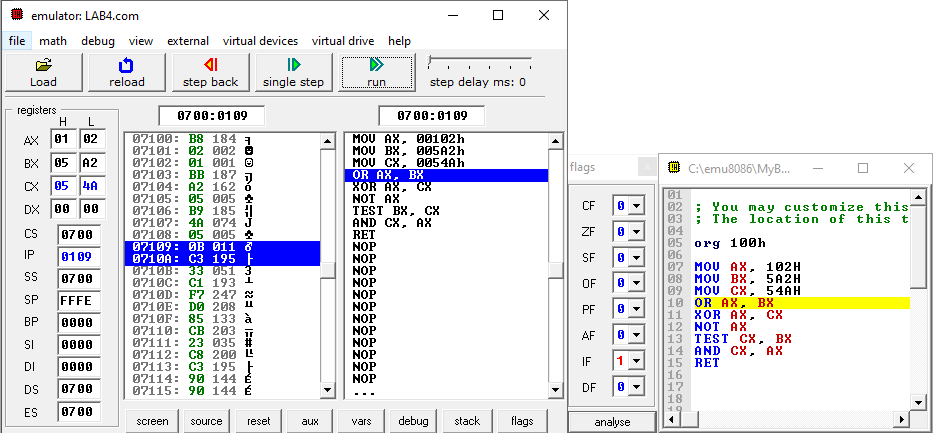
1. Observe the contents of the specified registers below after each instruction has been executed and record your result in Table 1.

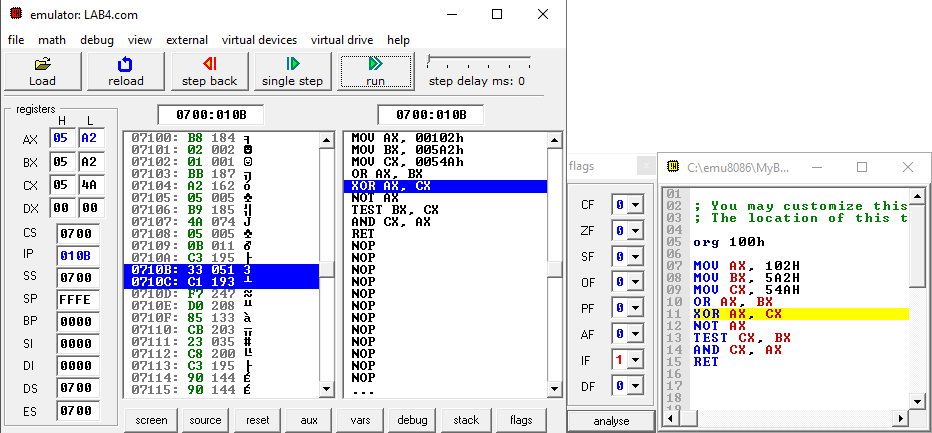
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Instruction | Register Content | | | | |
| AX | BX | CX | DX | IP |
| 1. ORG 100H | 0000 | 0000 | 0014 | 0000 | 0100 |
| 2. MOV AX, 102H | 0102 | 0000 | 0014 | 0000 | 0103 |
| 3. MOV BX, 5A2H | 0102 | 05A2 | 0014 | 0000 | 0106 |
| 4. MOV CX, 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 |

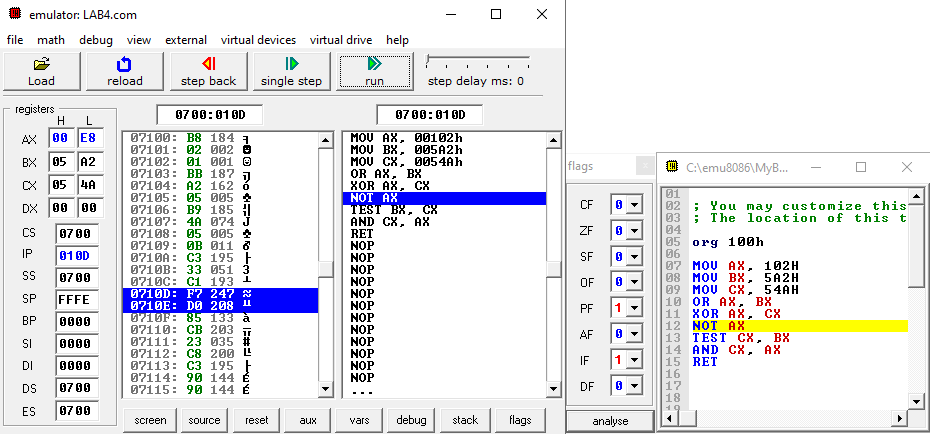
**Table 1**: Registers value

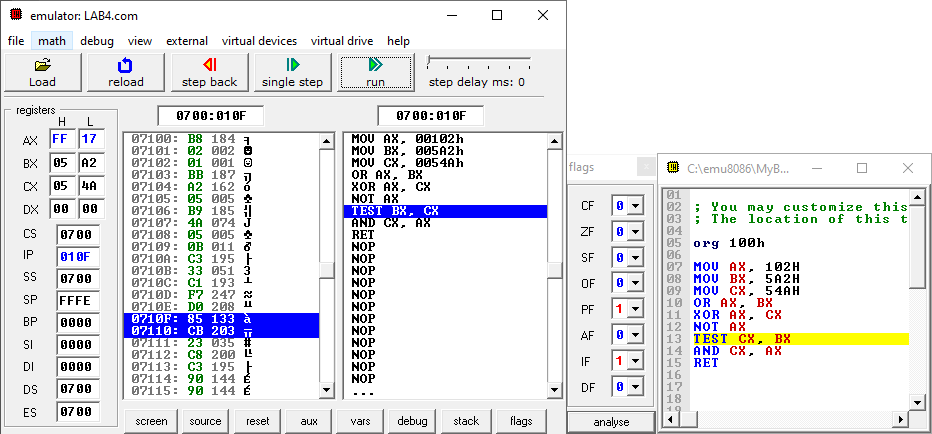


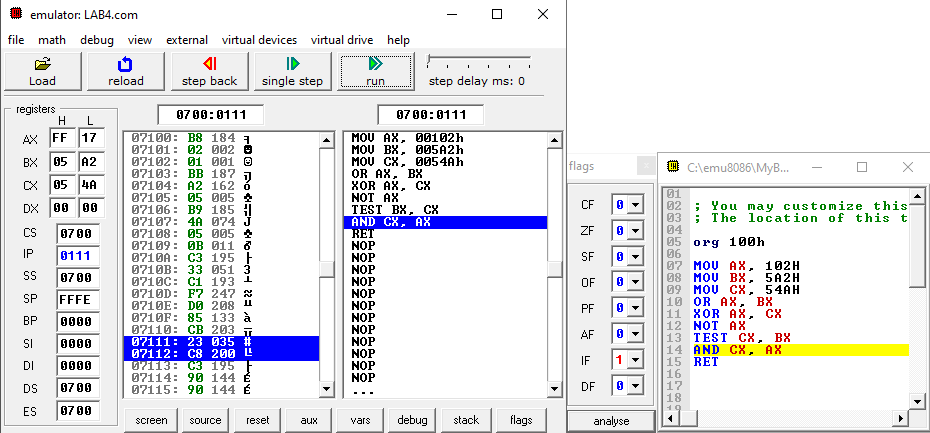


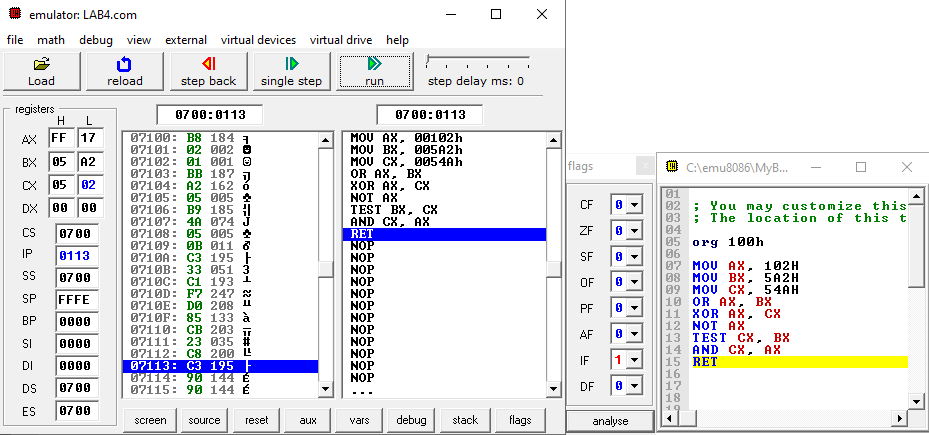








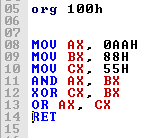


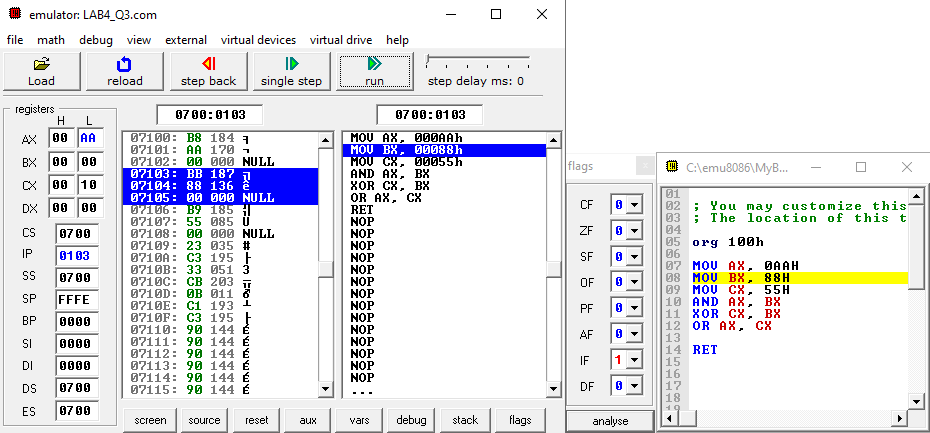


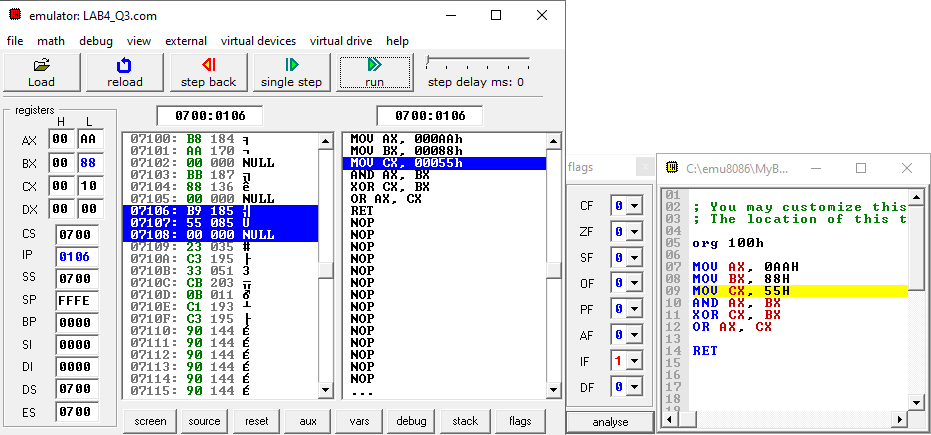
1. Write your own instruction based on the operation given in below table and record your result.

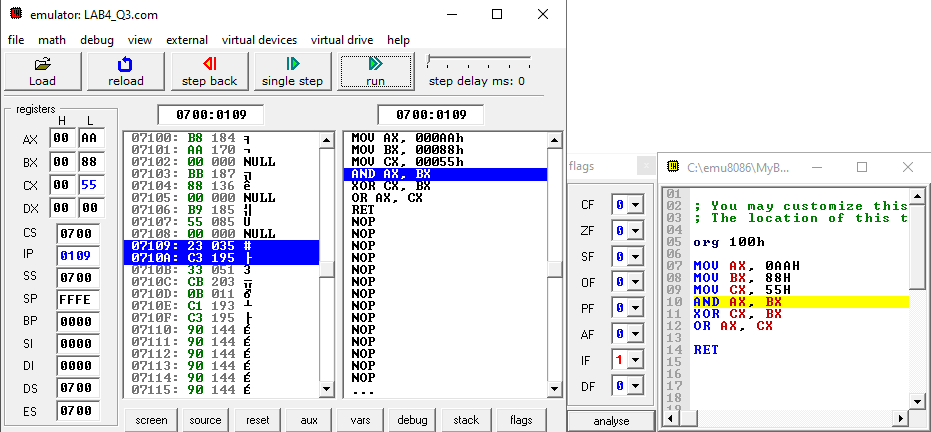
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Instruction | Register Content | | | | |
| AX | BX | CX | DX | IP |
| 1. ORG 100H | 0000 | 0000 | 0010 | 0000 | 0100 |
| 1. MOV AX, 0AAH | 00AA | 0000 | 0010 | 0000 | 0103 |
| 1. MOV BX, 88H | 00AA | 0088 | 0010 | 0000 | 0106 |
| 1. MOV CX, 55H | 00AA | 0088 | 0055 | 0000 | 0109 |
| 1. AND AX, BX | 0088 | 0088 | 0055 | 0000 | 010B |
| 1. XOR CX, BX | 0088 | 0088 | 00DD | 0000 | 010D |
| 1. OR AX, CX | 00DD | 0088 | 00DD | 0000 | 010F |
| 1. RET | 00DD | 0088 | 00DD | 0000 | 0000 |

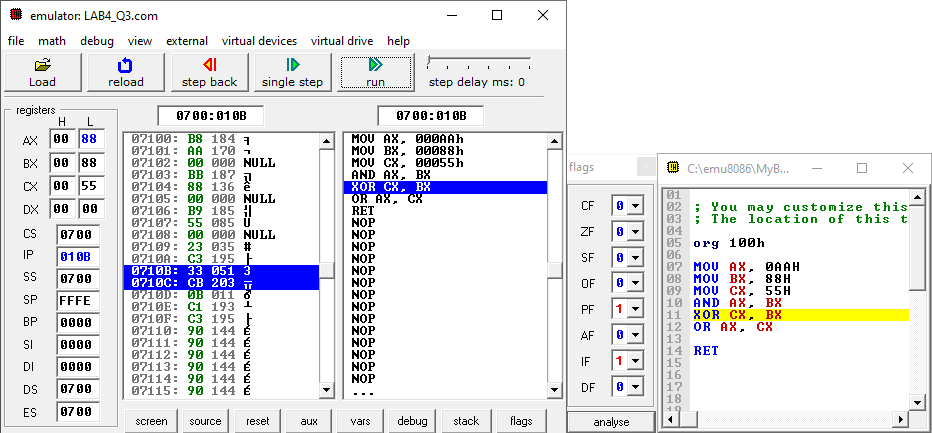
**Table 2**: Registers value

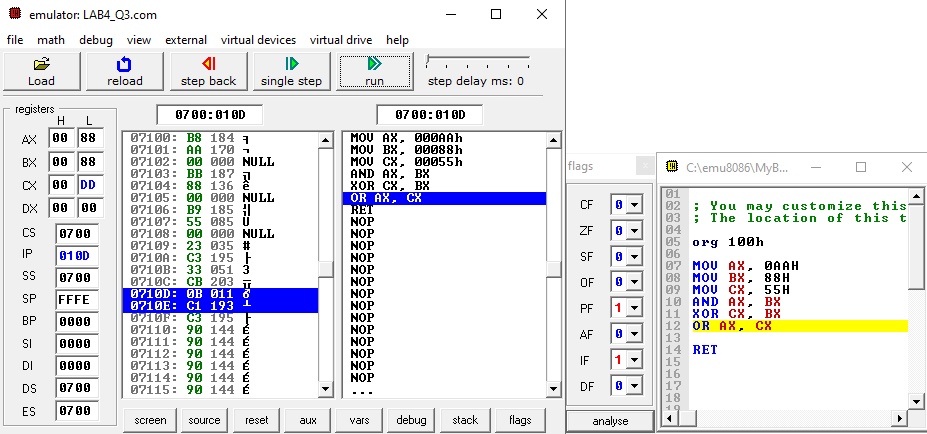


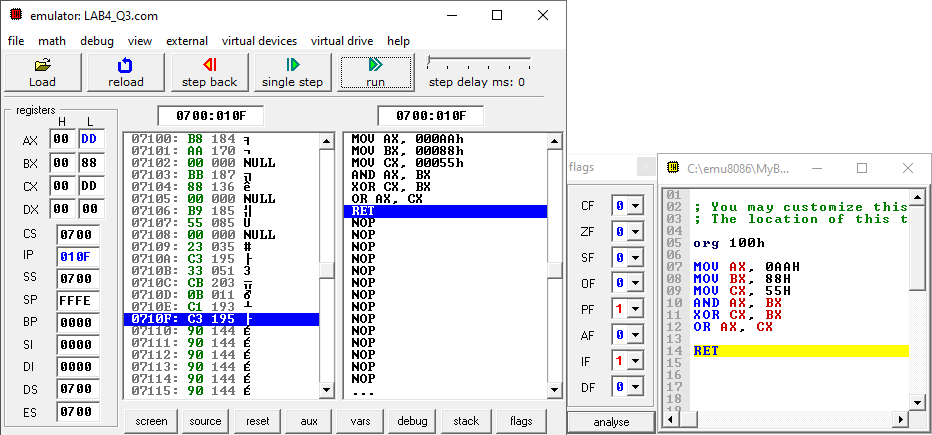












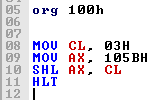
**5.2 Shift and Rotate command**

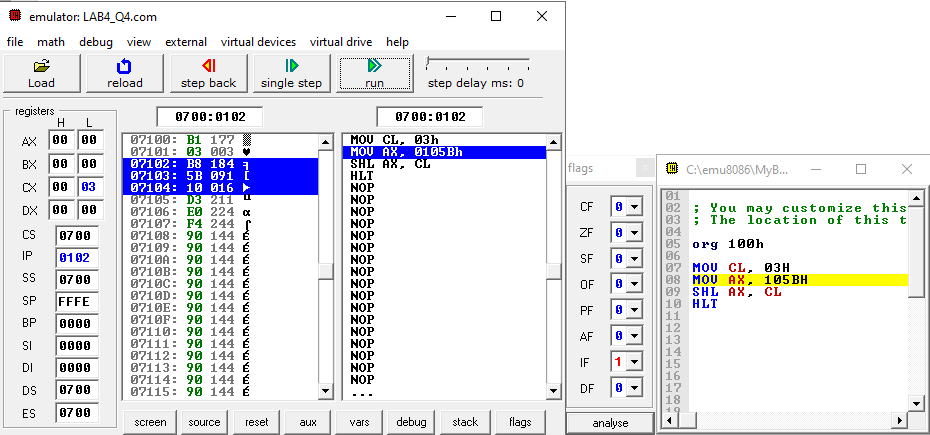
1. Obtain AX register value, write the previous value and present value in binary form. Write the type of operation for this instruction.

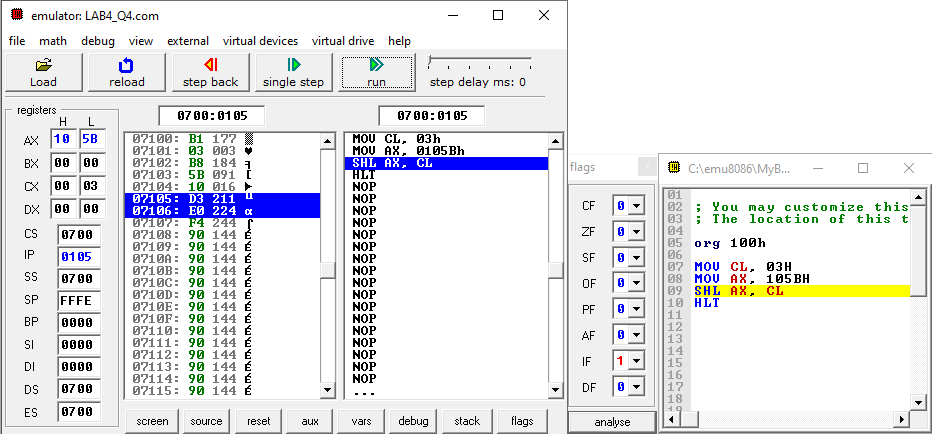
|  |  |  |  |
| --- | --- | --- | --- |
| Instruction | Register AX ( in binary ) | | Operation type |
| Previous value | Present value |
| 1. SHL AX,CL | 0001 0000 0101 1011 | 1000 0010 1101 1000 | SHIFT LEFT |
| 1. SAR AX,CL | 0001 0000 0101 1011 | 0000 0010 0000 1011 | SHIFT ARITHMETIC RIGHT |
| 1. ROR AX,CL | 0001 0000 0101 1011 | 0110 0010 0000 1011 | ROTATE RIGHT |
| 1. RCL AX,CL | 0001 0000 0101 1011 | 1000 0010 1101 1000 | ROTATE LEFT THROUGH CARRY |

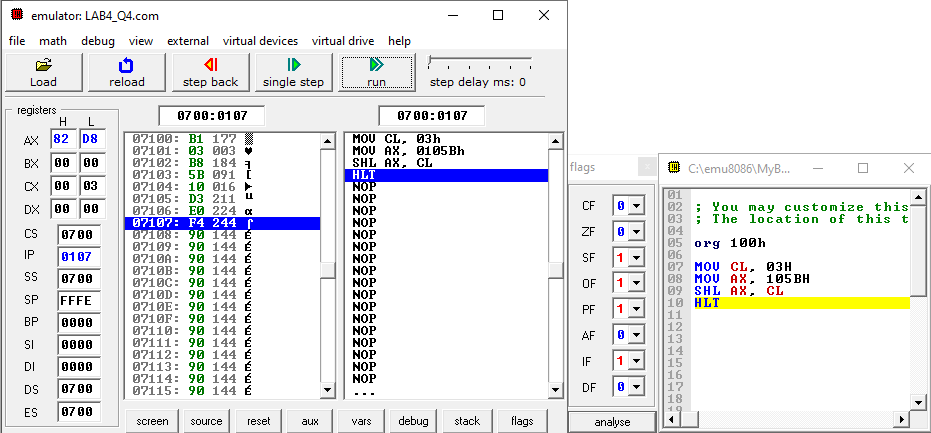
**Table 2**: Registers value

1. SHL AX, CL

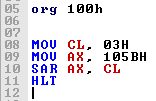


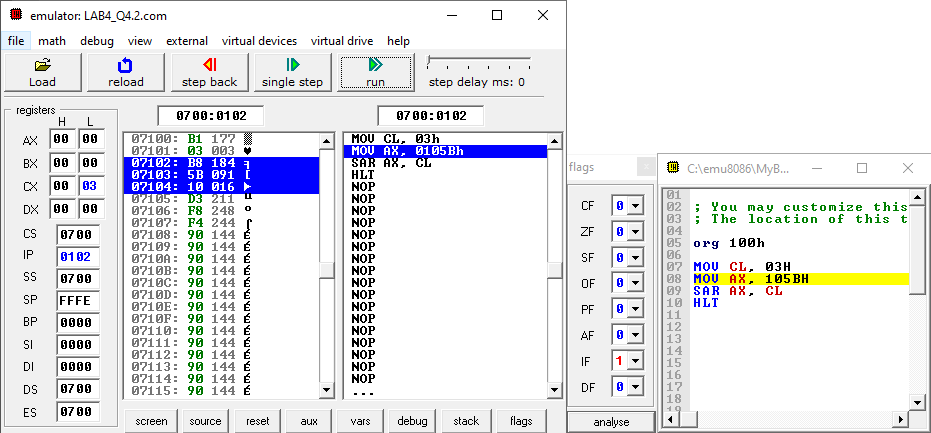


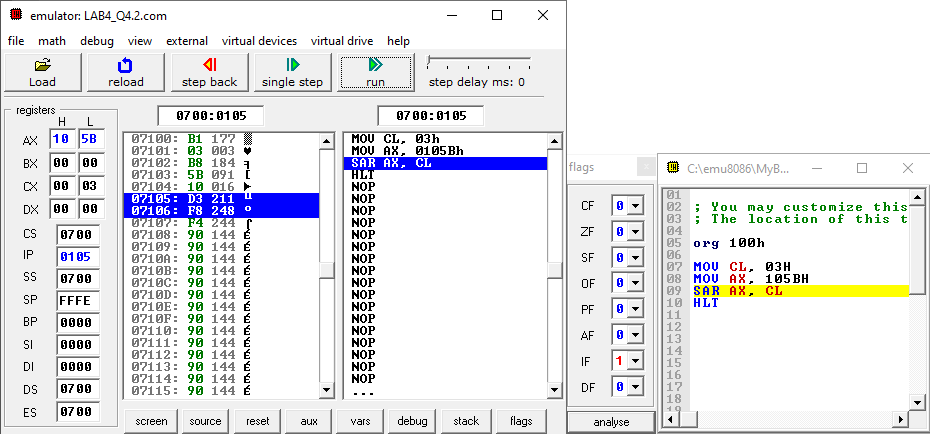


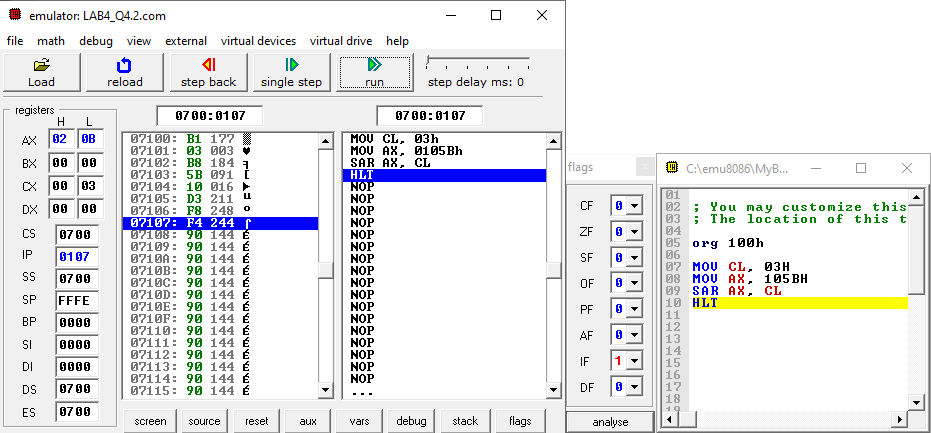


1. SAR AX,CL

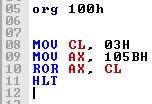


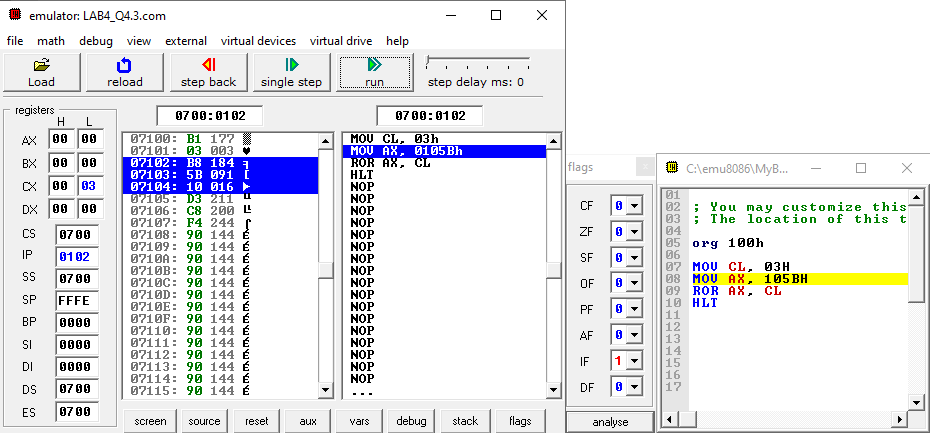


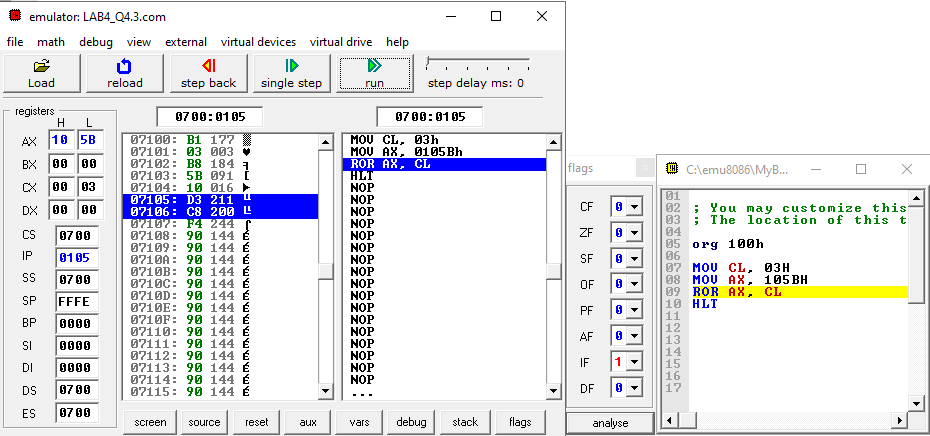


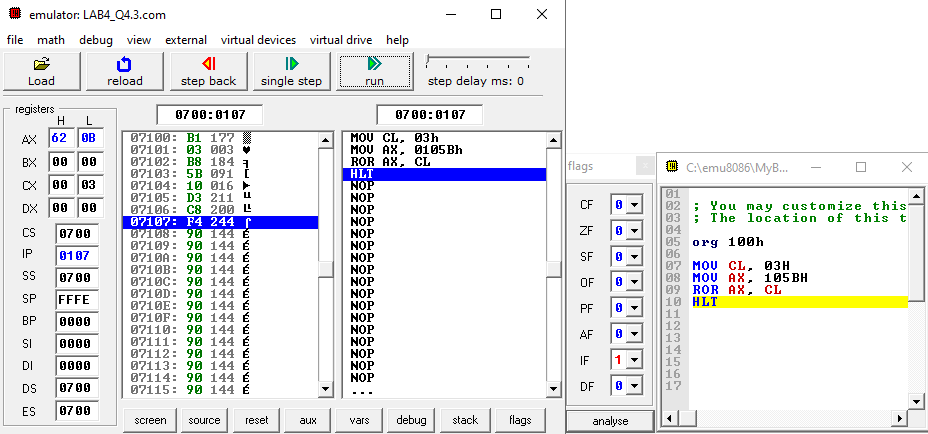


1. ROR AX,CL

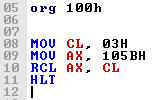


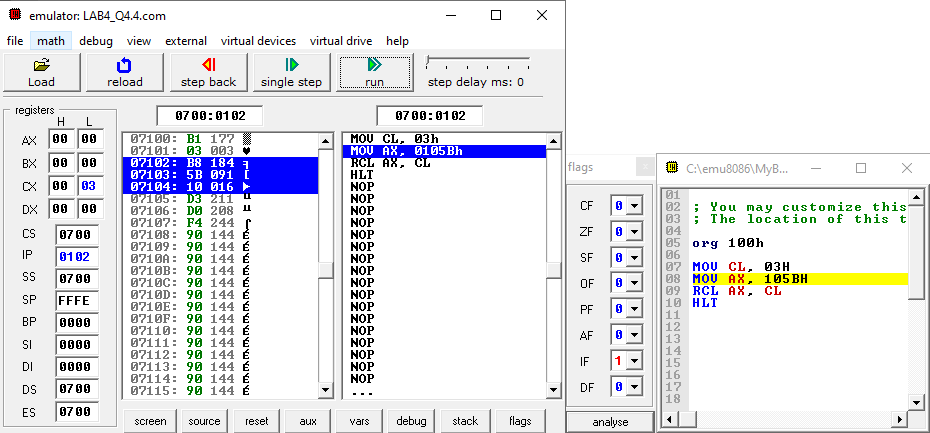


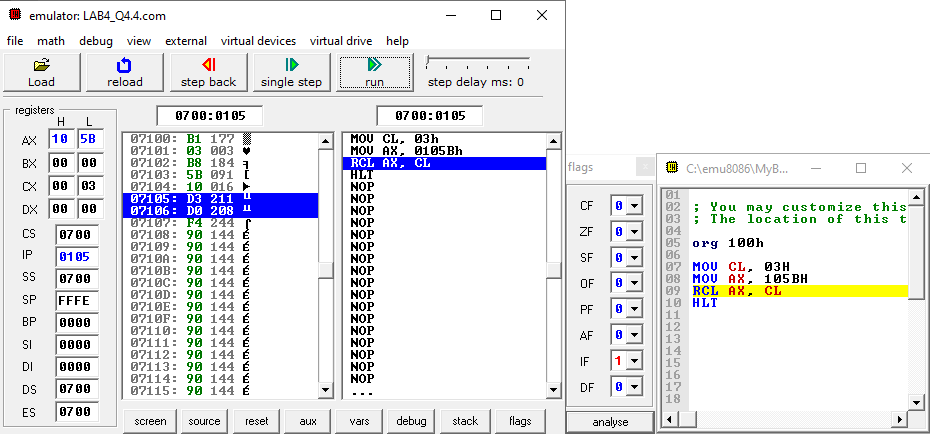


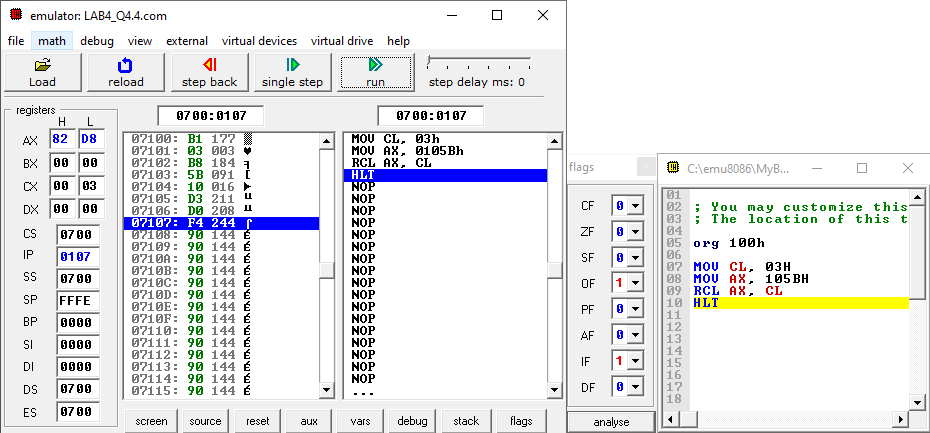


1. RCL AX,CL









## DISCUSSION

* 1. **Logical Instructions**

OR gate

* OR operation will return value 1 (TRUE) if the bits from either one or both operands are 1. It will returns 0 (FALSE) when both bits are 0.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | OR | 1 | = | 1 |
| 1 | OR | 0 | = | 1 |
| 0 | OR | 1 | = | 1 |
| 0 | OR | 0 | = | 0 |

Example:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | (0102H) | 0000 | 0001 | 0000 | 0010 |
| OR | (05A2H) | 0000 | 0101 | 1010 | 0010 |
|  | (05A2H) | 0000 | 0101 | 1010 | 0010 |

XOR gate

* XOR operation will return value of 1 (TRUE) if the bits from both operands are different. The value will returns 0 (FALSE) when bits between both operands are same.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | XOR | 1 | = | 0 |
| 1 | XOR | 0 | = | 1 |
| 0 | XOR | 1 | = | 1 |
| 0 | XOR | 0 | = | 0 |

Example:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | (05A2H) | 0000 | 0101 | 1010 | 0010 |
| XOR | (054AH) | 0000 | 0101 | 0100 | 1010 |
|  | (00E8H) | 0000 | 0000 | 1110 | 1000 |

NOT gate

* NOT operation will reverse the bits in operand. It means that if the bit is 1, return value will be 0 (FALSE). If bit is 0, return value will be 1 (TRUE).

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | NOT | = | 0 |
| 0 | NOT | = | 1 |

Example:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | (00E8H) | 0000 | 0000 | 1110 | 1000 |
| NOT |  |  |  |  |  |
|  | (FF17H) | 1111 | 1111 | 0001 | 0111 |

TEST instructions

* This instruction is basically works same as AND gate but it only updates flags without changing the destination or original number in register.
* Flags that involve in this instruction are Status Flag (SF), Zero Flag (ZF) and Parity Flag (PF).

AND gate

* AND operation will returns value 1 (TRUE) when the matching bits from both register are 1 only. Otherwise, it will returns 0 (FALSE).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | AND | 1 | = | 1 |
| 1 | AND | 0 | = | 0 |
| 0 | AND | 1 | = | 0 |
| 0 | AND | 0 | = | 0 |

Example:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | (054AH) | 0000 | 0101 | 0100 | 1010 |
| AND | (05A2H) | 0000 | 0101 | 1010 | 0010 |
|  | (0502H) | 0000 | 0101 | 0000 | 0010 |

* 1. **Shift and Rotate command**

Shift command

* Shift instructions move a bit string (or operand treated as a bit string) to the **right** or **left**, with excess bits discarded (although one or more bits might be preserved in flags).

Rotate command

* Rotate instructions are similar to shift instructions, except 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**.

* In **arithmetic shift left** or **logical shift left** zeros are shifted into the low-order bit. In **arithmetic shift right** the sign bit (most significant bit) is shifted into the high-order bit. In **logical shift right** zeros are shifted into the high-order bit.
* In **rotate through carry left/right**, all bits will shift to left/right and the bits that goes off will go to Carry Flag (CF) while value inside that flag is inserted to the right/left most position.

## CONCLUSION

In a conclusion, this lab taught us on how to use Logic Instructions to perform in Arithmetic Logic Unit (ALU). The commands that we use for this lab are AND gate, OR gate, NOT gate, XOR gate, TEST instruction, Rotate instruction and Shift instruction. Each of these commands have their own operation to produce an output. AND instruction only can returns 1 (TRUE) when both bits are 1. OR instruction will returns 1 (TRUE) if either or both bits are 1. NOT instruction will operate to reverse the all bits in register. XOR instruction will set the result as 1 (TRUE) when if both bits are different value. TEST commands can operates as AND gate but the result will not store in any register while the value of flags will change. Rotate instruction will move every bits in circular ways either left or right while Shift instruction will move the every bits in line to the left/right.