# LAB # 5

# To Explain and Show the Output of Logical Instructions using EMU8086 Software Tool

**Objectives**

* + To explain and show the output of AND, OR, NOT, and XOR Instruction using emu8086 software tool.
  + To explain and show the output of RCL, ROR, ROL, and RCR Instruction using emu8086 software tool.
  + To explain and show the output of SHL, SHR, and SAL Instruction using emu8086 software tool.

**Pre-Lab Exercise**

Read the details given below in order to comprehend the basic operation of logical instructions. It includes different operations such as **AND, OR, XOR, NOT**, Shifting and rotating of data. Study in detail and become familiar with the various ways and combinations in which these instructions can be used.

**AND Instruction**

AND instruction do logical AND on two 8-bit data or on two 16-bit data. For example, in order to logically AND contents of BH and DL, AND is used as:

**AND BH, DL**

This instruction logically ANDs the contents of BH and DL registers, leaving result in BH and DL remains unchanged. AND can be applied on 16-bit data stored in registers in the similar way. Any 8- bit register (**AH, AL, BH, BL, CH, CL, DH, DL**) can be used with AND. For 16-bit AND, instruction can be applied on **AX, BX, CX, DX, SI and DI**. You are encouraged to check the validity of AND instruction for another 16-bit registers yourself.

**OR Instruction**

OR instruction executes logical OR on two 8-bit data or on two 16-bit data. For example, in order to logically OR contents of BH and DL, OR is used as:

**OR BH, DL**

This instruction logically ORs the contents of BH and DL registers, leaving result in BH and DL remains unchanged. OR can be applied on 16-bit data stored in registers in the similar way. Any 8- bit register (**AH, AL, BH, BL, CH, CL, DH, DL**) can be used with OR. For 16-bit OR, instruction can be applied on **AX, BX, CX, DX, SI and DI**. You are encouraged to check the validity of OR instruction for another 16-bit registers yourself.

**XOR Instruction**

XOR instruction executes exclusive OR on two 8-bit data or on two 16-bit data. For example, in order to exclusive OR contents of BH and DL, XOR is used as:

**XOR BH, DL**

This instruction XORs the contents of BH and DL registers, leaving result in BH and DL remains unchanged. XOR can be applied on 16-bit data stored in registers in the similar way. Any 8-bit register (**AH, AL, BH, BL, CH, CL, DH, DL**) can be used with XOR. For 16-bit XOR, instruction can be applied on **AX, BX, CX, DX, SI and DI.** You are encouraged to check the validity of XOR instruction for another 16-bit registers yourself.

**NOT Instruction**

NOT instruction takes one complement of operand that can be 8-bit data or 16-bit data. For example, in order to apply NOT on AH register, following syntax is used.

**NOT AH**

After executing this instruction, AH is left with complement of its contents prior to execute this instruction.

Any 8-bit register (**AH, AL, BH, BL, CH, CL, DH, DL**) can be used with NOT. For 16-bit NOT, it can be applied on **AX, BX, CX, DX, SI and DI.** You are encouraged to check the validity of SUB instruction for another 16-bit registers yourself.

**RCL Instruction**

There are four different types of rotate instructions. RCL instruction stands for “Rotate Left through Carry” positions the bits in a register or in memory location according to following scenario.



*Figure 3-1 Rotate left through carry*

Previous value of CF goes to LSB of operand and MSB of operand goes into the CF. Bits in the operand are shifted left by one bit at a time.

Applying RCL on AH, for example, is:

**RCL BL, 6**

Above instruction accomplish RCL operation on BL for six times. If number of shift operations is a variable value, then it is to be placed in CL. Result is stored back in BL.

**ROL Instruction**

The rotate instruction ROL stands for “Rotate Left”. ROL positions the bits in a register or in memory location according to following scenario.



*Figure 3-2 Rotate Left operation*

Previous value of MSB of operand goes into the CF and the same also enters in to the LSB position, with each bit in operand shifting left by one bit at a time.

Applying ROL on SI, for example, is:

**RCL SI,14**

Above instruction executes ROL operation on SI for fourteen times. If number of shift operations is a variable value, then it is to be placed in CL. Result is stored back in SI.

**RCR Instruction**

RCR, “Rotate Right through Carry”, behaves in reverse of RCL. Function of RCR is represented in following diagram.



*Figure 3-3 Rotate right through carry*

Previous value of CF goes to MSB of operand and LSB of operand goes into the CF. Bits in the operand are shifted right by one bit at a time.

Consider following instruction.

**RCR AH, CL**

After execution, original content of AH register is rotated right times the value in CL. Result is stored back in AH.

**ROR Instruction**

The rotate instruction ROR stands for “Rotate Right”. ROR positions the bits in a register or in memory location according to following scenario.



*Figure 3-4 Rotate right instruction*

Previous value of LSB of operand goes into the CF and the same also enters in to the MSB position, with each bit in operand shifting right by one bit at a time.

Consider following instruction.

**RCR [23ABH], CL**

After execution, original content of memory location with offset address 23ABH are rotated right though carry times the value in CL. Result is stored back in memory location with offset address 23ABH.

**SHL/SAL Instructions**

Like rotate instructions, shift instructions are four in number. Instruction SHL, abbreviated from “Shift Left”, shifts the bits in operand left one bit at a time. Instruction SAL “Shift Arithmetic Left” behaves exactly identical to SHL as the MSB, that reflects the positivity or negativity of value is altered in shift left operation.

Bit shifted out from MSB of operand enters into the CF. 0 is entered in the LSB position for a single SHL or SAL operation. This operation is diagrammatically represented below.







*Figure 3-5 Shift left instruction*

Consider following instructions.

**SHL BX, 4**

**SAL DI, 6**

The instruction SHL BX, 4shifts the contents of BX four bits left, leaving result in BX. If number of shifts is a variable value, then it is to be stored in CL. SHL can be done on 8-bit register, 16-bit register or on a memory location.

The instruction SAL DI, 6 behaves exactly similar to above discussed instruction with difference that content of DI are shifted six bits left now.

**SHR Instruction**

Instruction SHR, abbreviated of “Shift Right”, shifts the bits in operand right one bit at a time. Bit shifted out from LSB of operand enters into the CF. 0 is entered in the MSB position for a single SHR operation. This operation is diagrammatically represented below.





*Figure 3-6 Shift right instruction*

Consider following instruction.

**SHR DX, CL**

After execution, original content of DX is shifted right times the value in CL. Result is stored back in DX register.

**SAR Instruction**

Instruction SAR, abbreviated of “Shift Arithmetic Right”, shifts the bits in operand right one bit at a time. Bit shifted out from LSB of operand enters into the CF with MSB remains unchanged. This retains the positivity or negativity of value in operand. This operation is diagrammatically represented below.



*Figure 3-7 Shift right arithmetic*

Consider following instruction.

**SAR DH, CL**

After execution, original content of DH is shifted right times the value in CL with MSB preserved. Result is stored back in DH register.

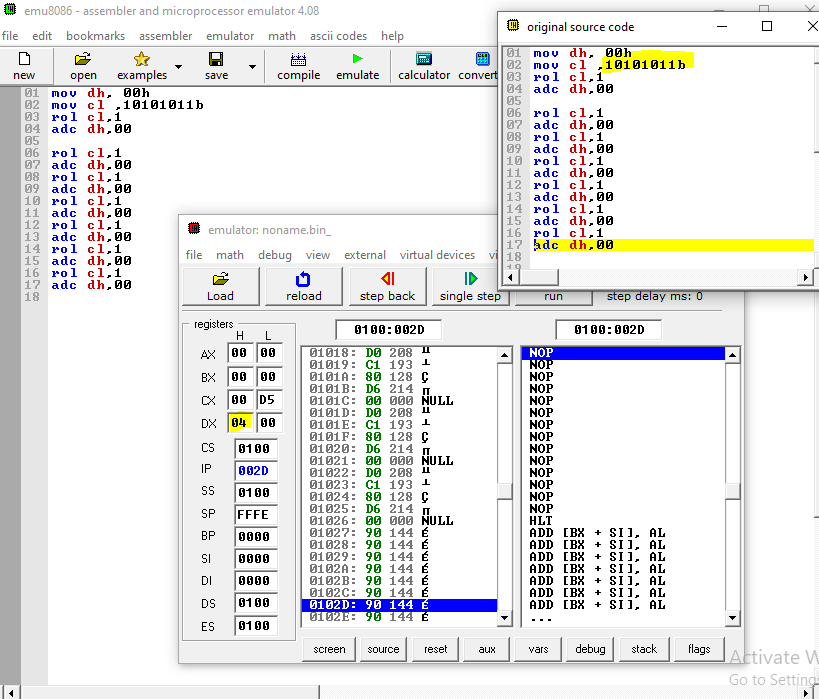
**In-Lab Exercise**

**Task 1: Write the following assembly program in editor and execute it**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **AH** | **AL** | **BH** | **BL** | **DH** | **DL** | **CL** | **CF** |
| **MOV AX,1456** | 05 | B0 | 00 | 00 | 00 | 00 | 00 | 0 |
| **MOV BX,0FF00H** | 05 | B0 | FF | 00 | 00 | 00 | 00 | 0 |
| **XOR AX, AX** | 00 | 00 | FF | 00 | 00 | 00 | 00 | 0 |
| **MOV BX,2300H** | 00 | 00 | 23 | 00 | 00 | 00 | 00 | 0 |
| **MOV DS, BX** | 00 | 00 | 23 | 00 | 00 | 00 | 00 | 0 |
| **MOV [1601H],25H** | 00 | 00 | 23 | 00 | 00 | 00 | 00 | 0 |
| **OR BH, [1601H]** | 00 | 00 | 27 | 00 | 00 | 00 | 00 | 0 |
| **MOV CL,3** | 00 | 00 | 27 | 00 | 00 | 00 | 03 | 0 |
| **MOV DX,23DCH** | 00 | 00 | 27 | 00 | 23 | DC | 03 | 0 |
| **SHL DX, CL** | 00 | 00 | 27 | 00 | 1E | E0 | 03 | 0 |
| **SHR DX,3** | 00 | 00 | 27 | 00 | 03 | DC | 03 | 0 |
| **SAL DX,1** | 00 | 00 | 27 | 00 | 07 | B8 | 03 | 0 |
| **MOV AH,250** | FA | 00 | 27 | 00 | 07 | B8 | 03 | 0 |
| **ADD AH,10** | 04 | 00 | 27 | 00 | 07 | B8 | 03 | 0 |
| **RCR AH,2** | 41 | 00 | 27 | 00 | 07 | B8 | 03 | 0 |

**Task 2: Write and assembly program to count the number of 1’s**

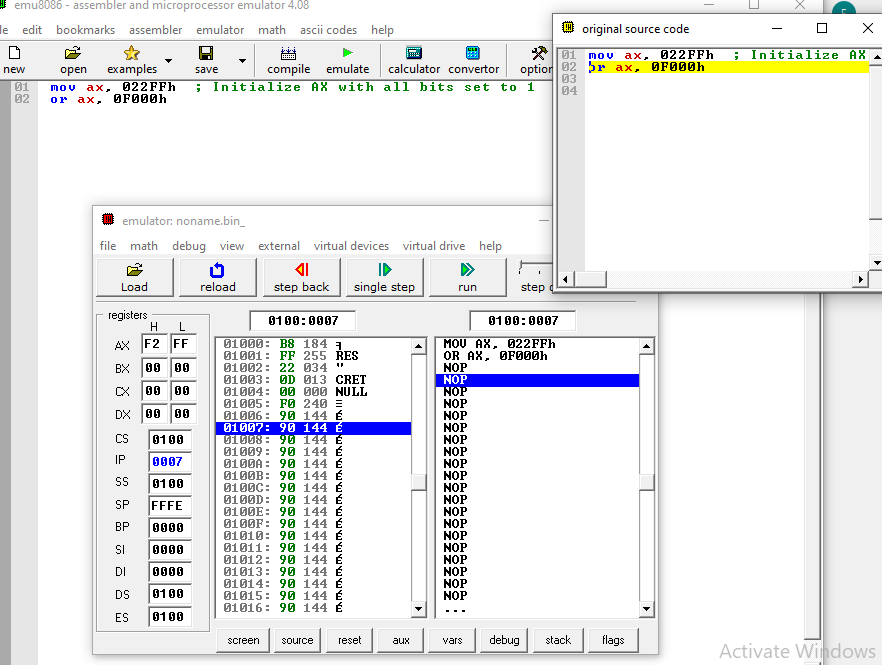
Write an assembly language program that counts the number of ‘1’s in a byte residing in CL register. Store the counted number in DH register.



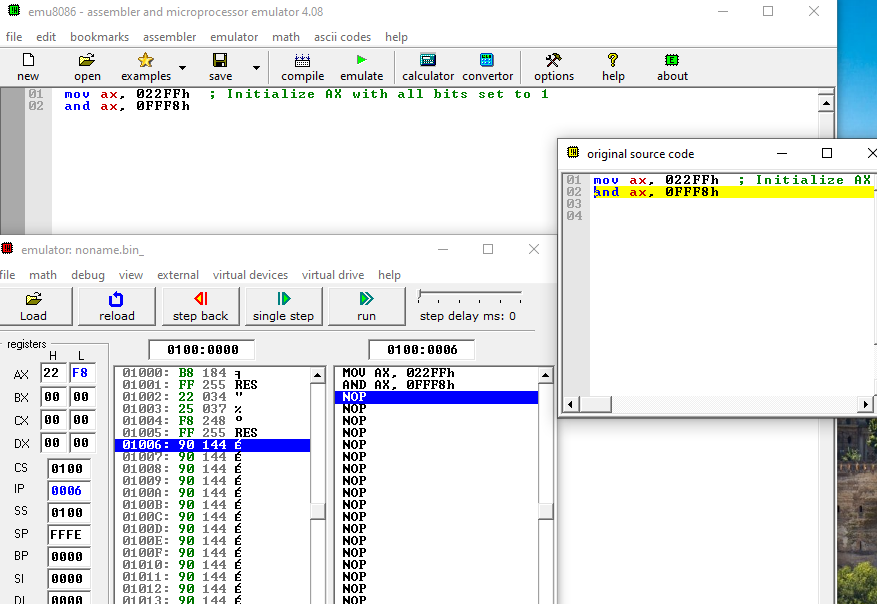
**Task 3: Bits’ manipulation**

Write an assembly language program to perform the following tasks:

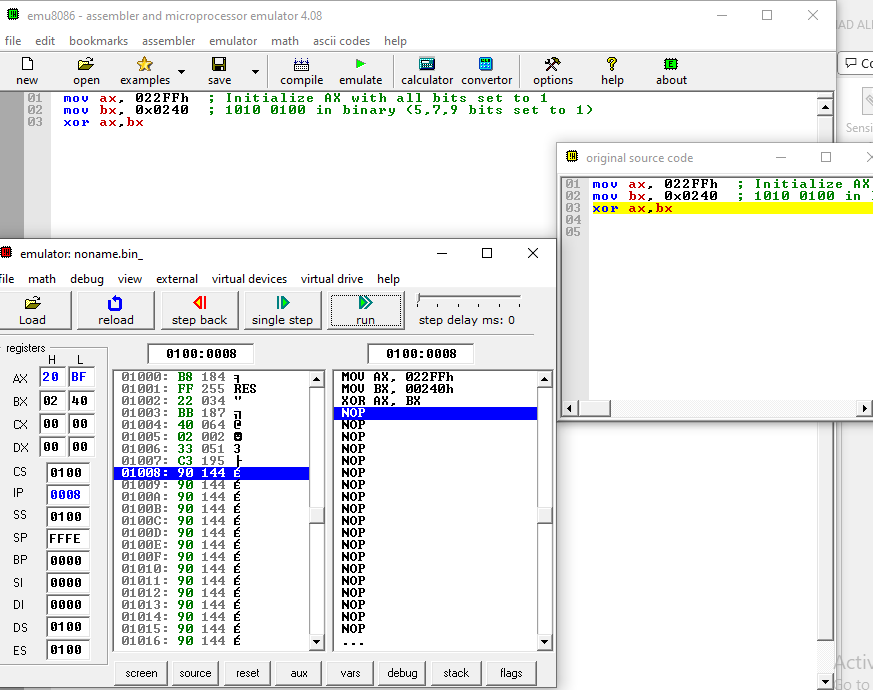
1. Set the leftmost 4 bits of AX



1. Clear the rightmost 3 bits of AX



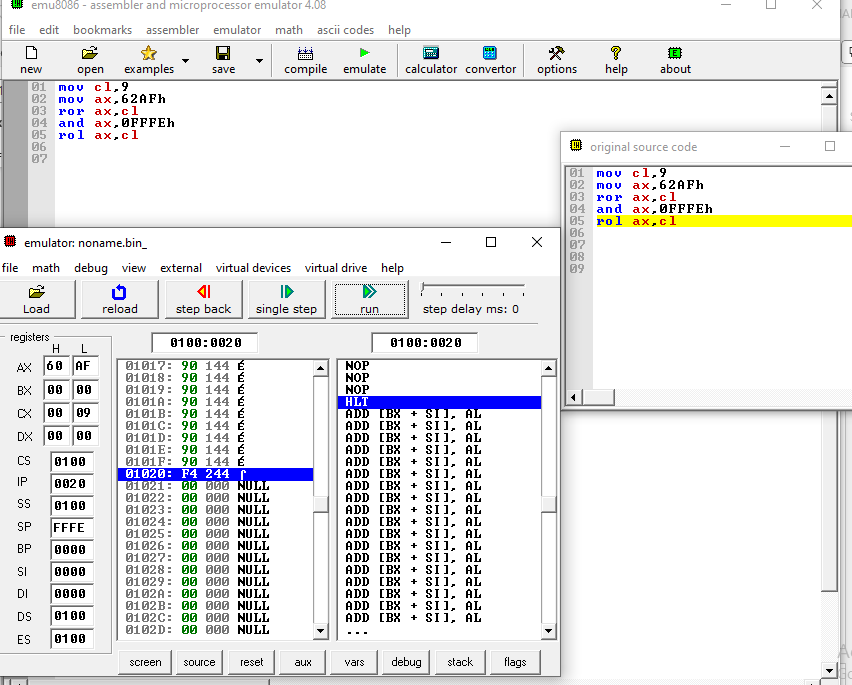
1. Invert the bits 5,7and 9 of AX.



**Task 4: Bits’ masking**

Write an assembly language program that clears any bit (from bit0 to bit15) in AX register, leaving other bits unchanged. Number of bits that is to be cleared is stored in CL register.

**Hint:** If the number 9 is stored in CL register, it means 9th bit of AX should be cleared



**Rubric for Lab Assessment**

|  |  |  |  |
| --- | --- | --- | --- |
| **The student performance for the assigned task during the lab session was:** | | | |
| Excellent | The student completed assigned tasks without any help from the instructor and showed the results appropriately. | 4 |  |
| Good | The student completed assigned tasks with minimal help from the instructor and showed the results appropriately. | 3 |  |
| Average | The student could not complete all assigned tasks and showed partial results. | 2 |  |
| Worst | The student did not complete assigned tasks. | 1 |  |

**Instructor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**