**Experiment No. 03**

* 1. **Experiment Name**

Logical operation and Loop using Assembly Language Programming

* 1. **Objectives**
* To understand the structure of Assembly Language programming
* To learn about the microprocessor emulator "Emu 8086" and its operation
* To get acquainted with the logical operation using ALU
* To learn how to implement program in “MDA 8086” Trainer Board and interconnect it with “Emu 8086”
  1. **Theory**

In assembly language programming, various logical operation is executed through exclusive instruction sets. In this experiment we will be using following operations and loops for study purpose.

**3.3.1 Shift Operation**: The shift operation simply changes the provided data bit in the register to the left or right. It comes in two varieties.

* **Shift Left (SHL):** The SHL instruction stands for “shift left”. This instruction moves the given register bits one at a time to the left for a positive integer.
* **Shift Right (SHR):** SHR is an abbreviation for "shift right." This instruction moves the given register bits one at a time to the right.
* **Shift Arithmetic Left (SAL):** The SAL is frequently used when numeric multiplication is required. In other words, shift binary one bit to the left while leaving the MSB untouched.
* **Shift Arithmetic Right (SAR):** SAR is quite similar to SAL with a slight change. Itshifts binary one bit to the right while leaving the MSB untouched.

**3.3.2 Rotate Operation:** This instruction rotates the bits. There are four rotation instructions:

* **Left Rotation (ROL):** The ROL instruction indicates to rotate the bits one by one to the left. The only distinction between ROL and SHL is, in SHL, the value of D0 is replaced by another value, whereas in ROL, the value of D0 is replaced by the bit from position D7.
* **Right Rotation (ROR):** The ROR instruction means to rotate the bits to the right one at a time. The sole difference between ROR and SHR is that in SHR, the bit of position D7 is replaced with another value.
* **Left Rotation with Carry (RCL):** The bits in the operand are rotated through the carry flag to the left using RCL. The most significant bit, together with the carry flag, is relocated to the least significant bit by the RCL instruction.
* **Right Rotation with Carry (RCR):** The bits in the operand are rotated through the carry flag using RCR. The least significant bit, together with the carry flag, is relocated to the most significant bit by the RCR instruction.

**3.3.3 AND**

Only if both of its inputs are true does the AND logic function return true. If one of the inputs is false, the output will be false as well.

**3.3.4 OR**

In OR operation, if one of the inputs is true, function return true. If one of the inputs is false, the output will be false as well.

**3.3.5 X- OR**

XOR compares two input bits and generates one output bit. The logic is simple. If the bits are the same, the result is true. If the bits are different, the result is false.

We will also implement looping operation in this experiment. Through this instruction, we will execute operation for the number of times defined before the loop

* 1. **Apparatus**
* Emu 8086 - Microprocessor Emulator
  1. **Emulator Code & output**
* **SHL**

**MOV CL, 2; COUNTER**

**MOV AX, 0B3H; MOVE 0B3H TO AX**

**SHL AX, CL; SHIFT LEFT**

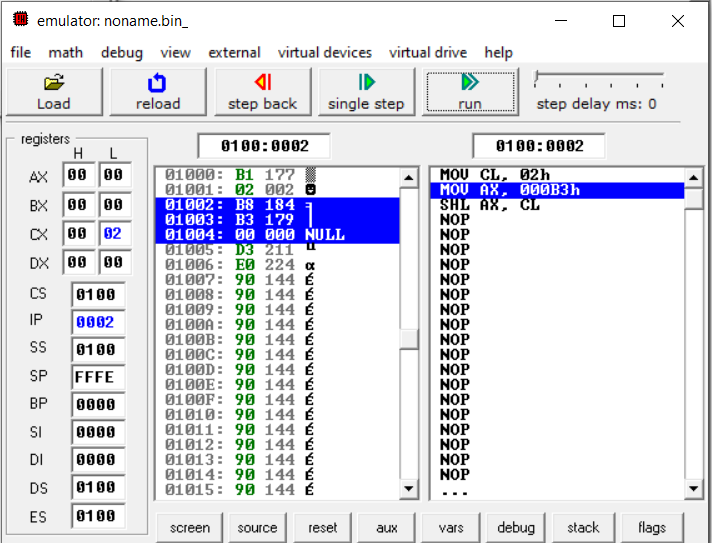
****

Fig 3.1: Emulator output for SHL (After Step 1)

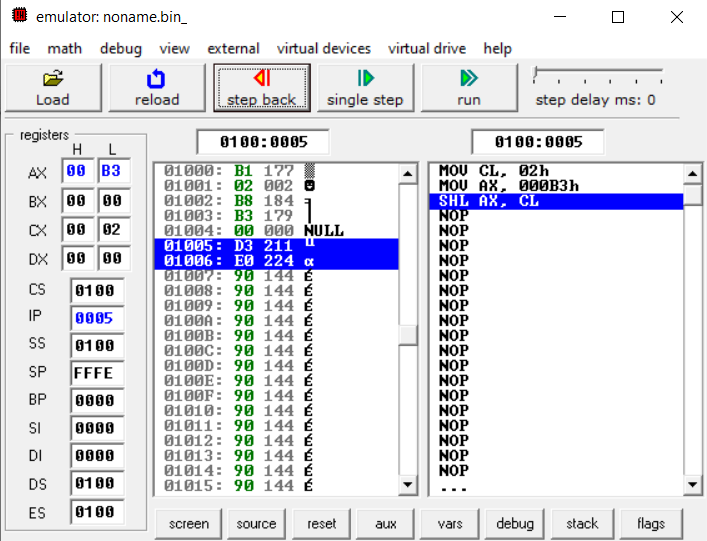
****

Fig 3.2: Emulator output for SHL (After Step 2)

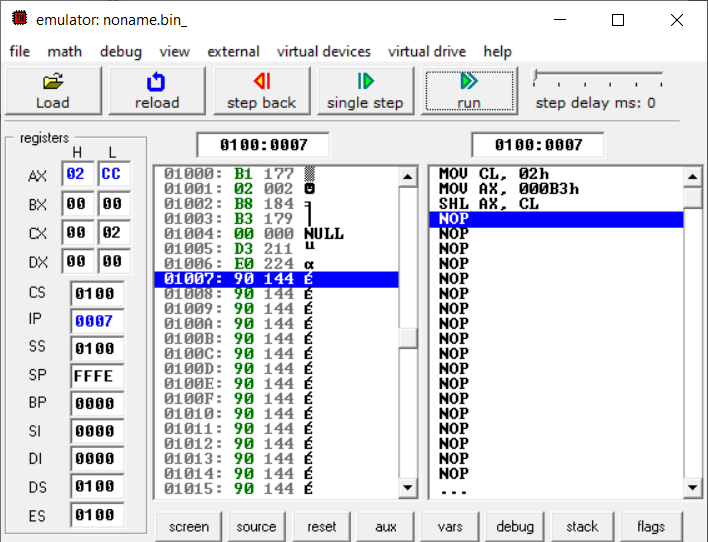
****

Fig 3.3: Emulator output for SHL (After Step 3)

* **SHR**

**MOV CL, 2; COUNTER**

**MOV AX, 0B3H; MOVE 0B3H TO AX**

**SHR AX, CL; SHIFT LEFT**

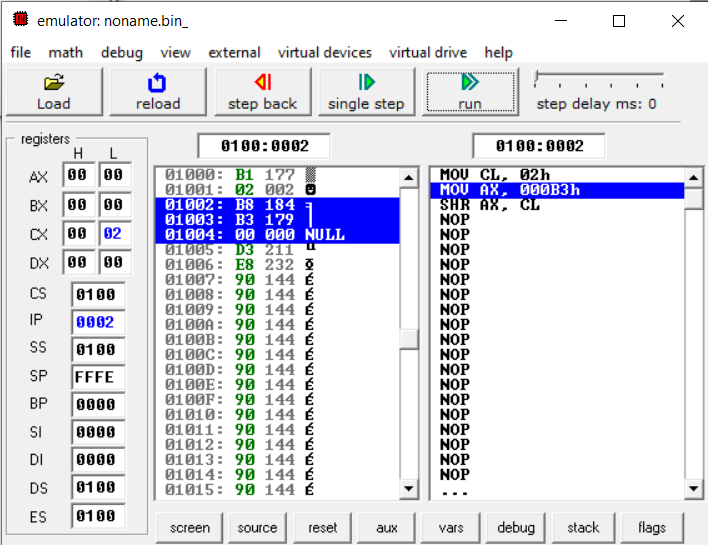
****

Fig 3.4: Emulator output for SHR (After Step 1)

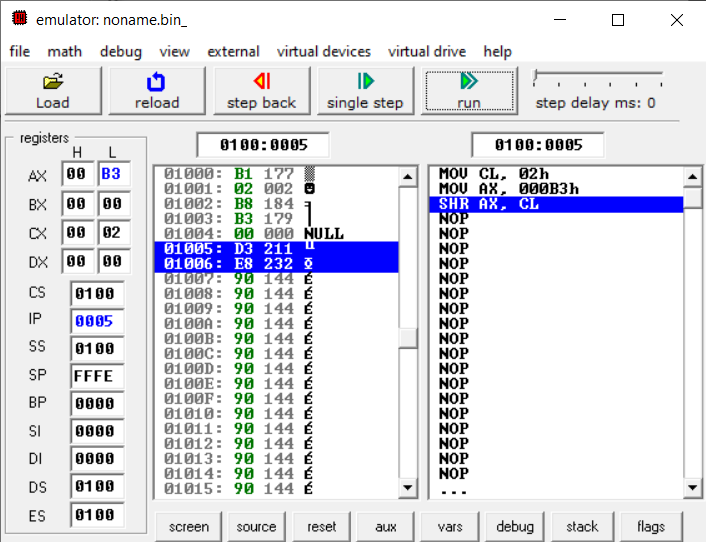
****

Fig 3.5: Emulator output for SHR (After Step 2)

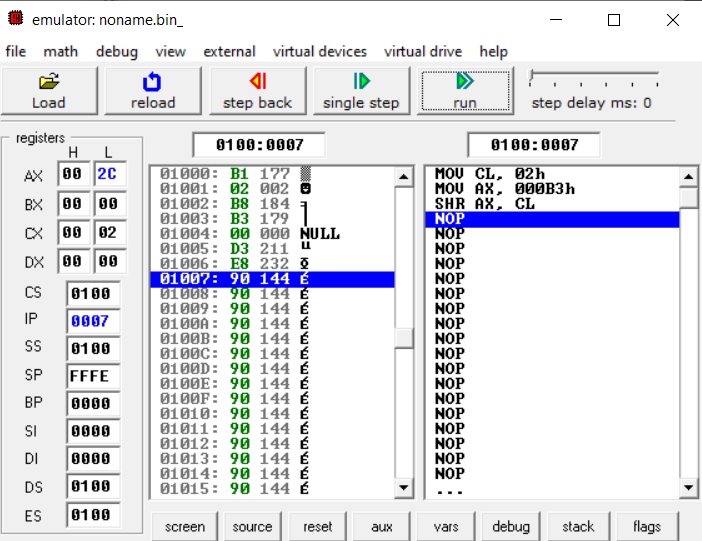
****

Fig 3.6: Emulator output for SHR (After Step 3)

* **SAL**

**MOV CL, 2; COUNTER**

**MOV AX, 0B3H; MOVE 0B3H TO AX**

**SAL AX, CL; SHIFT ARITHAMTIC LEFT**

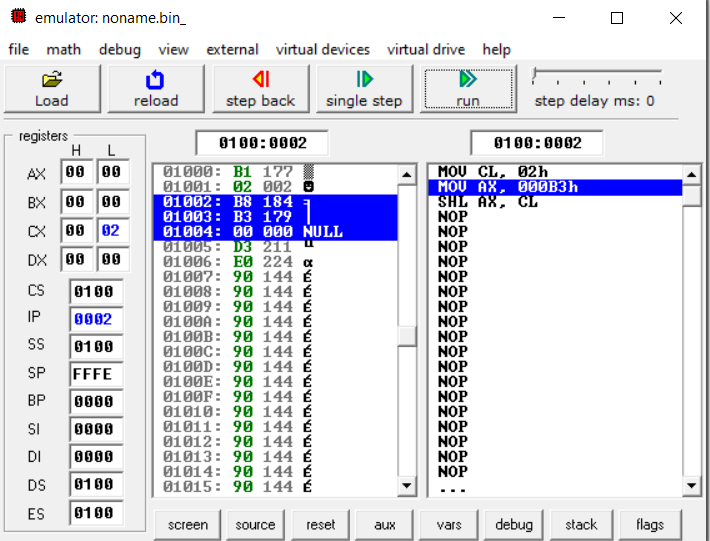
****

Fig 3.7: Emulator output for SAL (After Step 1)

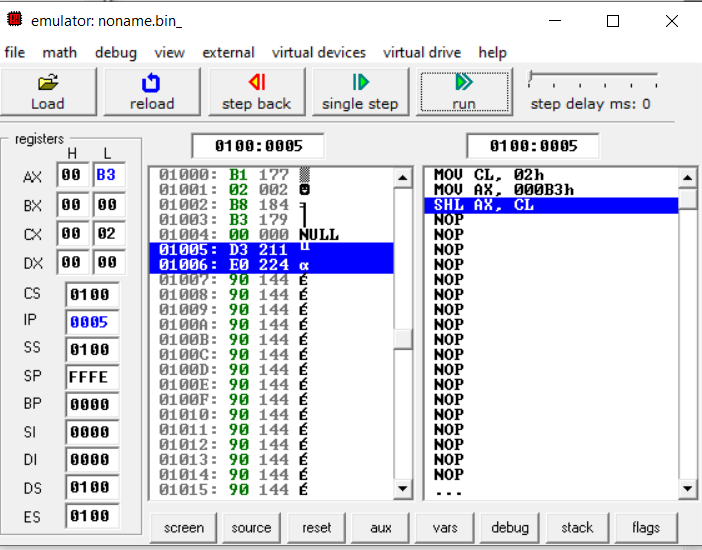
****

Fig 3.8: Emulator output for SAL (After Step 2)

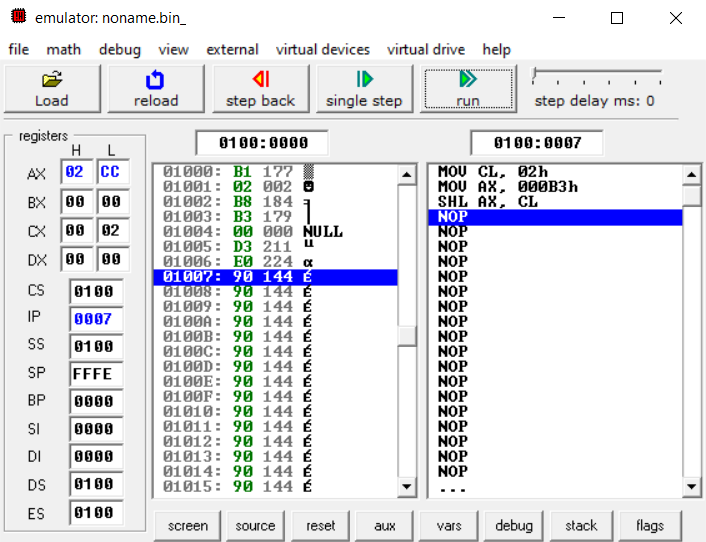
****

Fig 3.9: Emulator output for SAL (After Step 3)

* **SAR**

**MOV CL, 2; COUNTER**

**MOV AX, 0B3H; MOVE 0B3H TO AX**

**SAR AX, CL; SHIFT ARITHAMTIC LEFT**

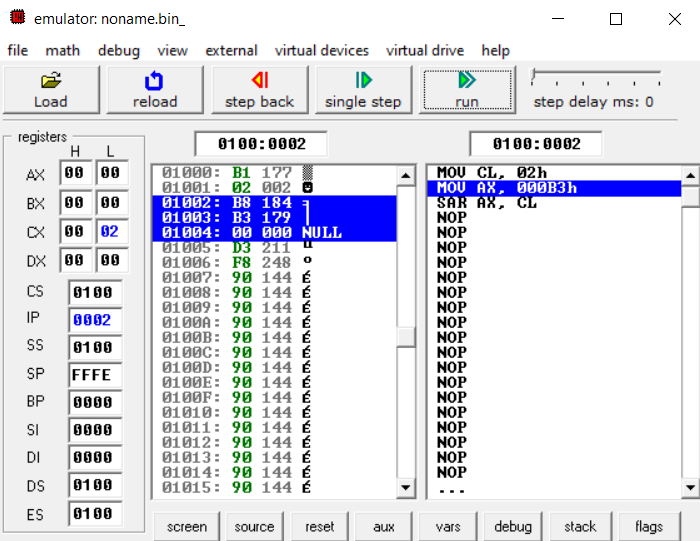
****

Fig 3.10: Emulator output for SAR (After Step 1)

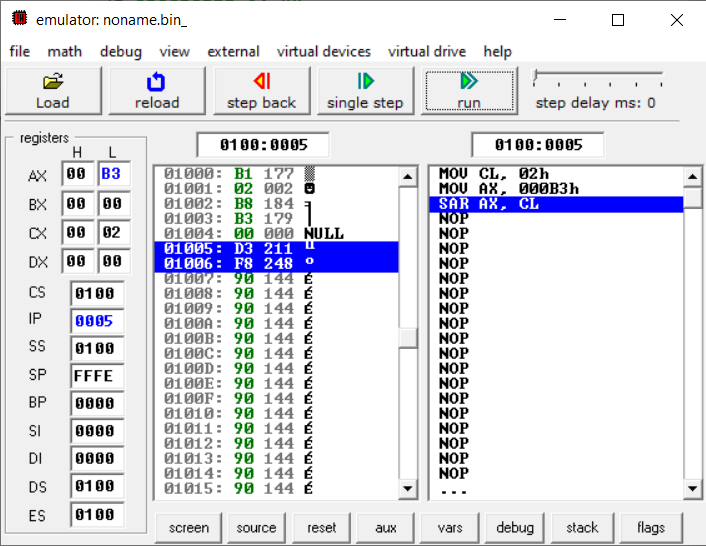
****

Fig 3.11: Emulator output for SAR (After Step 2)

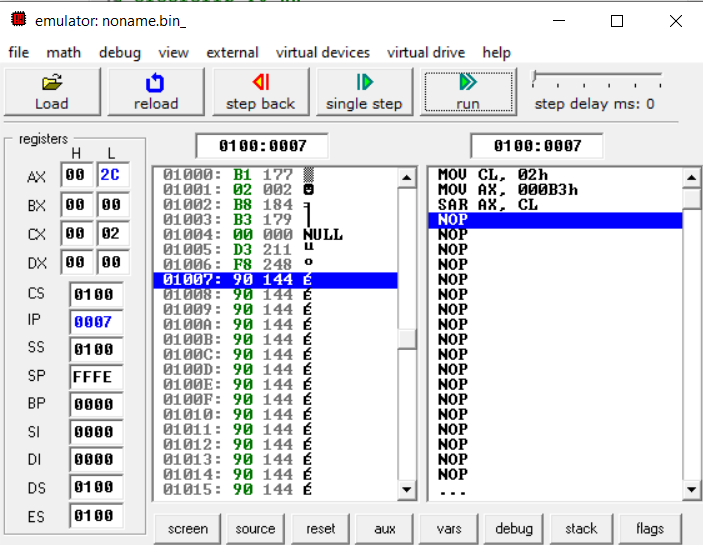
****

Fig 3.12: Emulator output for SAR (After Step 3)

* **ROL**

**MOV CL, 2; COUNTER**

**MOV AX, 0B3H; MOVE 0B3H TO AX**

**ROL AX, CL; ROTATE LEFT**

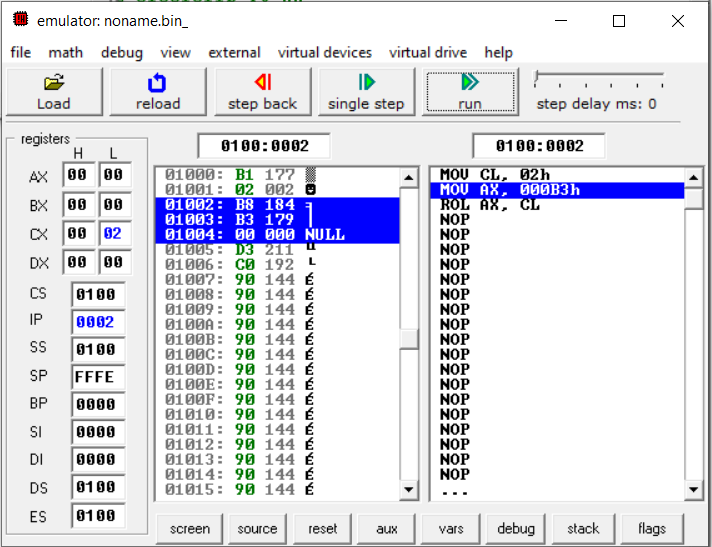
****

Fig 3.13: Emulator output for ROL (After Step 1)

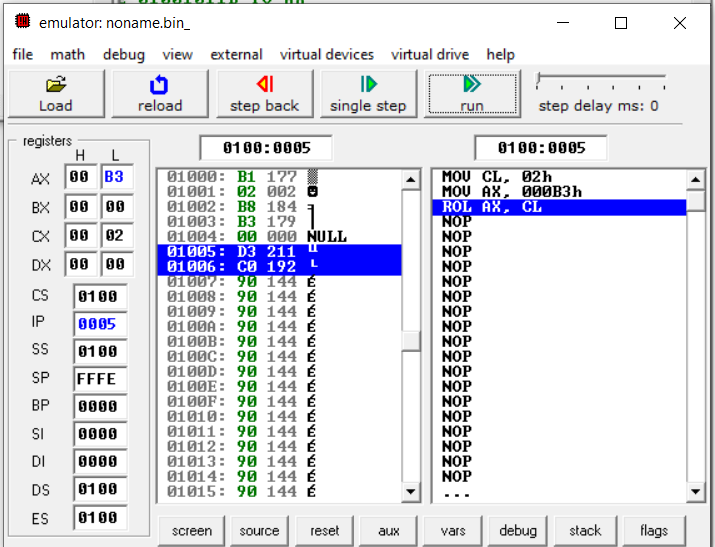
****

Fig 3.14: Emulator output for ROL (After Step 2)

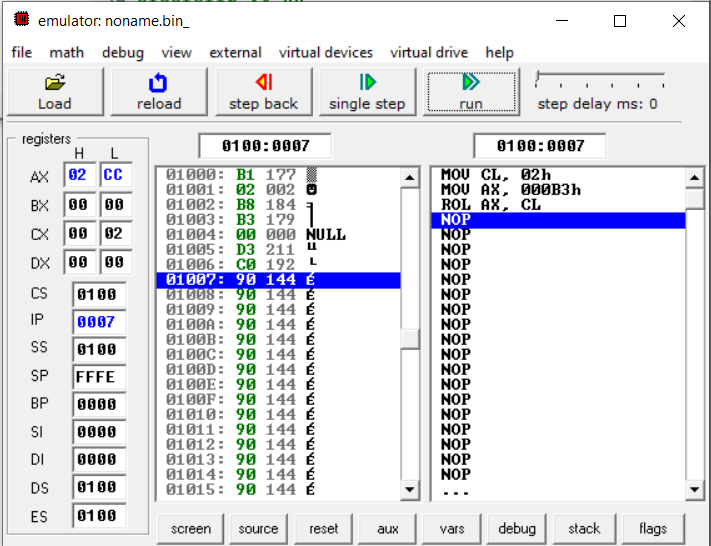
****

Fig 3.15: Emulator output for ROL (After Step 3)

* **ROR**

**MOV CL, 2; COUNTER**

**MOV AX, 0B3H; MOVE 0B3H TO AX**

**ROR AX, CL; ROTATE RIGHT**

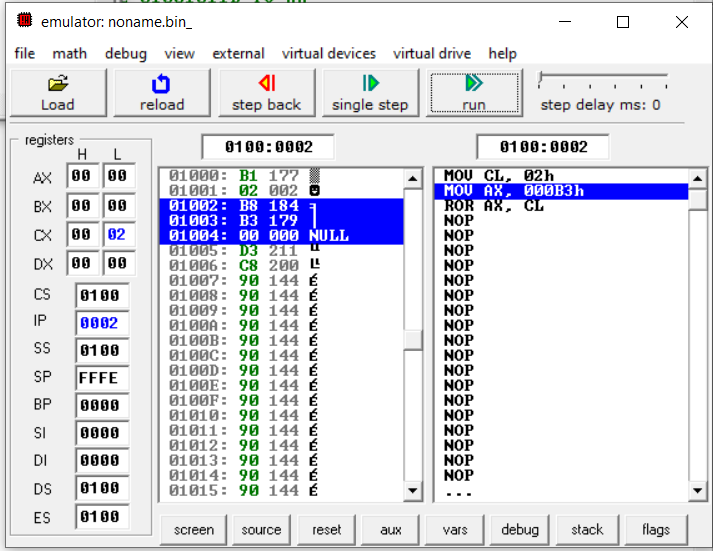
****

Fig 3.16: Emulator output for ROR (After Step 1)

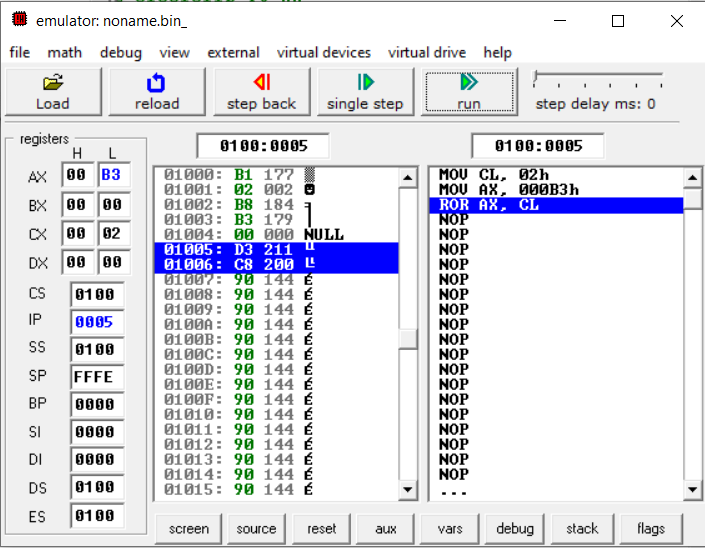
****

Fig 3.17: Emulator output for ROR (After Step 2)

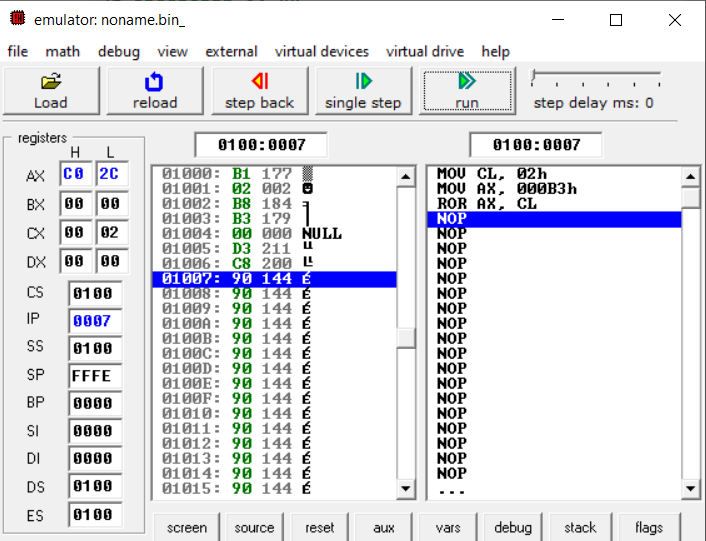
****

Fig 3.18: Emulator output for ROR (After Step 3)

* **RCL**

**MOV CL, 2; COUNTER**

**MOV AX, 0B3H; MOVE 0B3H TO AX**

**RCL AX, CL; ROTATE WITH CARRY TO LEFT**

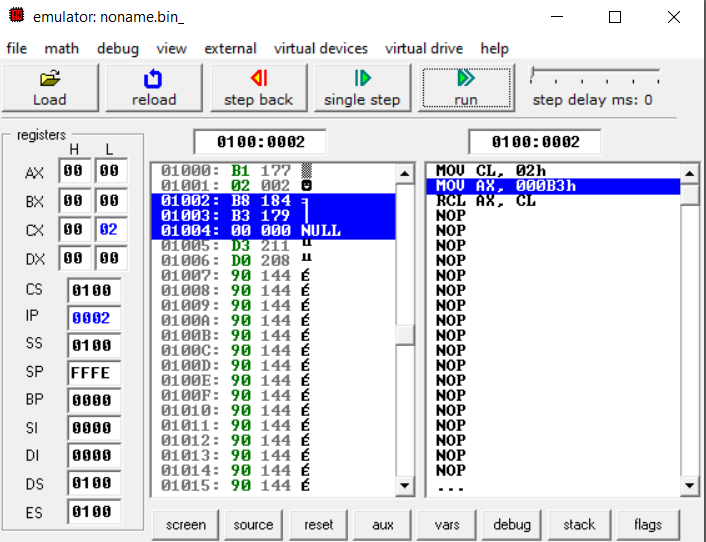
****

Fig 3.19: Emulator output for RCL (After Step 1)

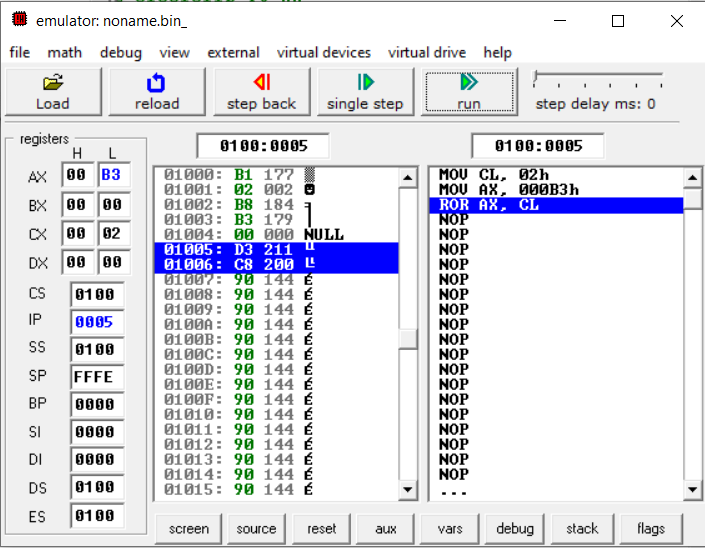
****

Fig 3.20: Emulator output for RCL (After Step 2)

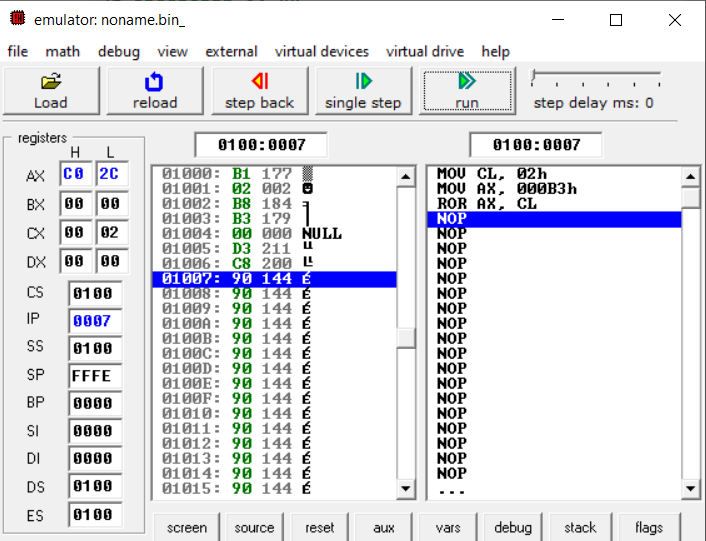
****

Fig 3.21: Emulator output for RCL (After Step 3)

* **RCR**

**MOV CL, 2; COUNTER**

**MOV AX, 0B3H; MOVE 0B3H TO AX**

**RCR AX, CL; ROTATE WITH CARRY TO RIGHT**

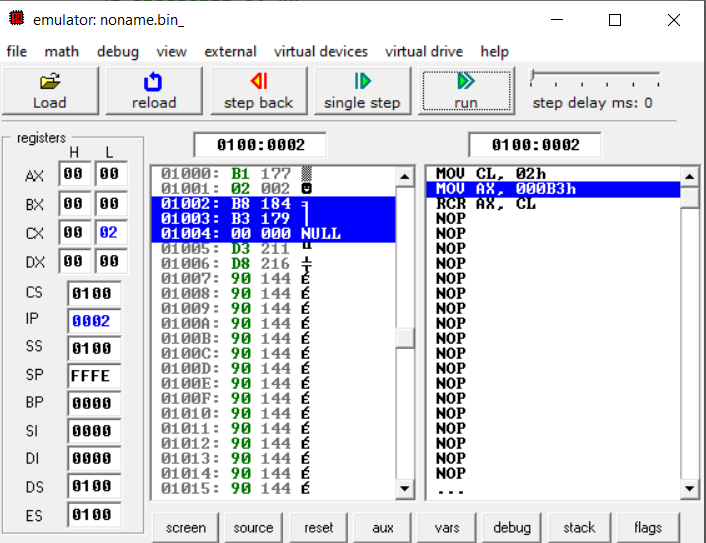


Fig 3.22: Emulator output for RCR (After Step 1)

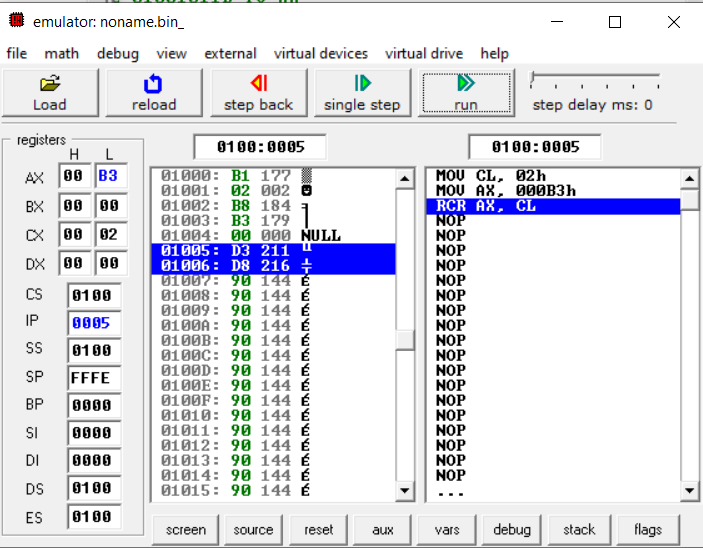


Fig 3.23: Emulator output for RCR (After Step 2)

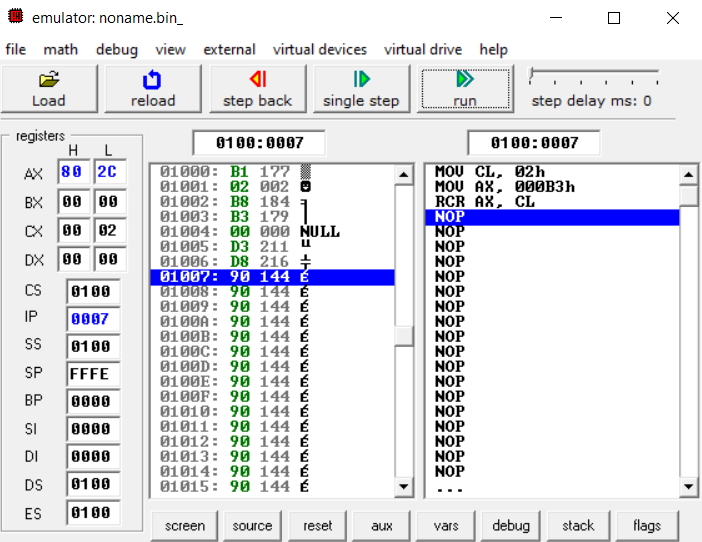
****

Fig 3.24: Emulator output for RCR (After Step 3)

* **AND**

**MOV AX, 5**

**MOV BX, 5**

**AND AX, BX**

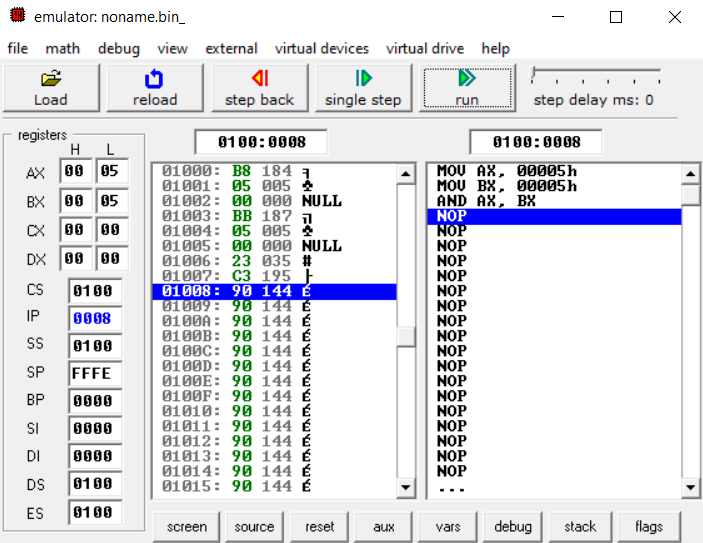
****

Fig 3.25: Emulator output for AND

* **OR**

**MOV AX, 5**

**MOV BX, 5**

**OR AX, BX**

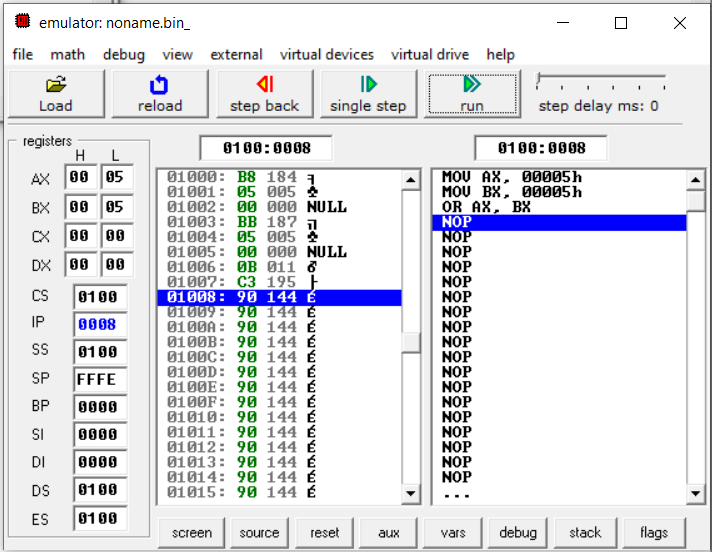
****

Fig 3.26: Emulator output for OR

* **XOR**

**MOV AX, 5**

**MOV BX, 5**

**XOR AX, BX**

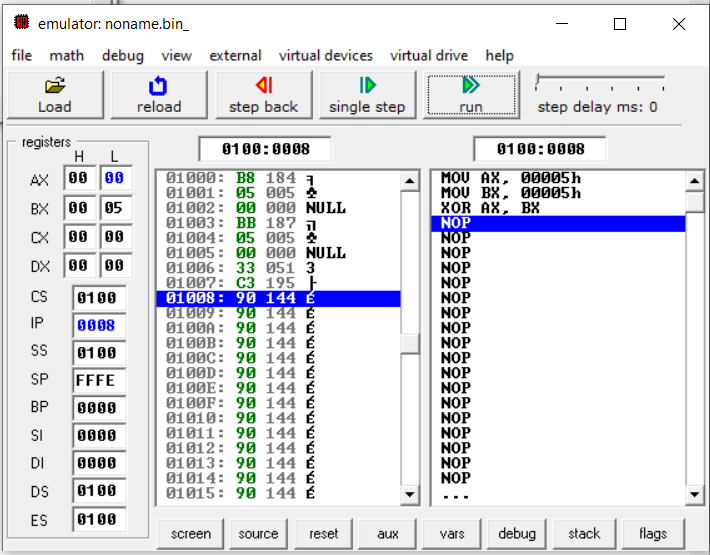
****

Fig 3.27: Emulator output for XOR

* **LOOP**

**MOV** **CL**, **5**

**MOV** **AL**, **10**

**TOP:**

**DEC** **AL**

**LOOP** **TOP**

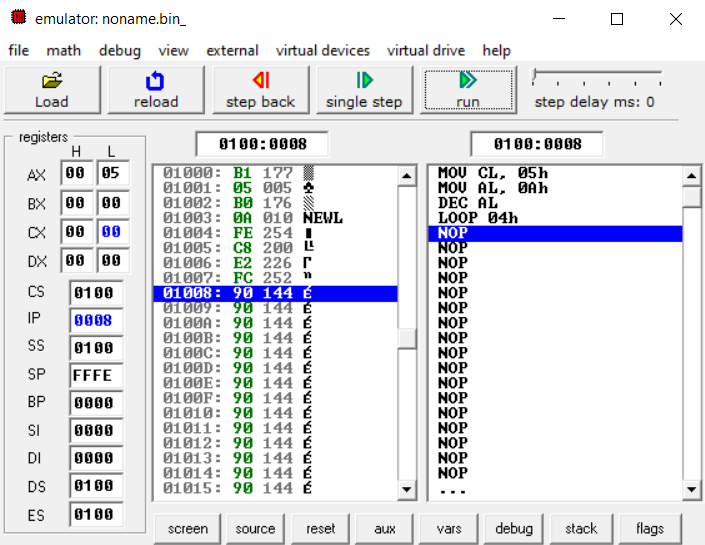


Fig 3.27: Emulator output for LOOP

* 1. **Discussion & Conclusion**

In this lab, we practiced assembly language programming on the emu8086 for 1-byte or 2-byte data. We also studied about arithmetic and shift-rotate programming languages.

The shift-left command moves the bit to the left, while the shift-right instruction moves it to the right. The only distinction between shift and rotate instructions is that rotate instructions cycle the bits. While shift rotates the parts out, around goes out one side and comes in the other. either side, leaving the region where the rotated bits were either unmodified or zeroes.

We also observed the process of AND, OR, and X-OR operation in the program. Finally, the objective of the experiment was achieved.