PRACTICAL 3

**2021**

USIU

Group 6

3/2/2021



**United States International University Africa  
Nairobi, Kenya  
APT2022-Introduction to Assembly Programming**

**PRACTICAL LAB 2: Emulator (EMU8086)**

**LECTURER: LINUS ALOO**

Group 6

* Clifford Ombiro 660904
* Timothy Diero 661405
* Moses Saruni 661821
* Aamir Abdirahman 661392
* Bill Musani 660884
* Charlton Charles 654039
* Alex Rugara 657317
* David Nelson 656429

**PRACTICAL LAB 3:**

**Title: 8086 Instruction Set Practice: Data Transfer Instructions**

**Objectives**

* To apply data transfer instructions in Assembly Language Programming
* To understand the use of emu 8086 emulator in assembly programming
* To show how the following data transfer instructions may be used with Emu 8086: XCHG, LEA, LDS, PUSH, POP, IN and OUT.
* To understand how the program instructions are executed

**Introduction**

In lab 2 exercise we are going to learn how to apply data instructions in assembly programming.

We are going to use assembler directives to understand the executions of the data transfer instructions

By studying the sample code provided we are going to test out different data transfer instructions to understand how they work

We will do this by looking at the results of the different registers by recording them and inferencing the results

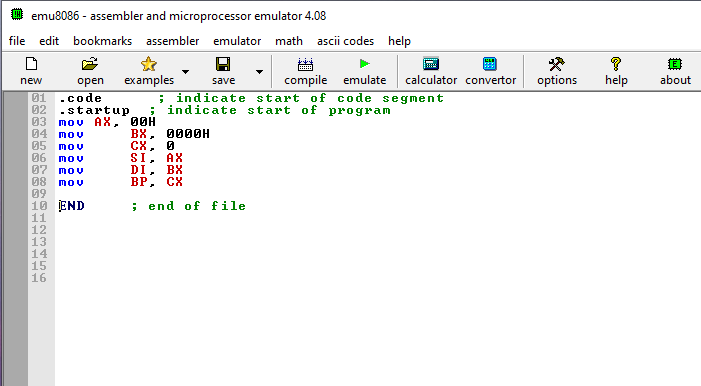
**Material under usage**

We will be using the following tools in our lab exercise

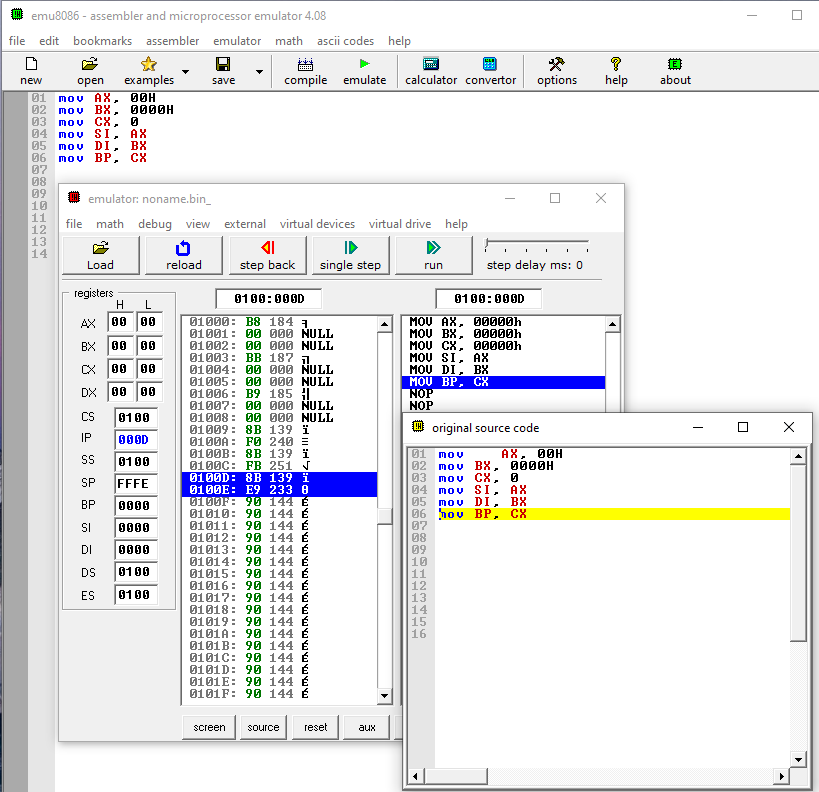
* A computer
* Emulator(EMU8086)
* Internet connection
* Microsoft word

**Methodology/Procedure**

Question 1

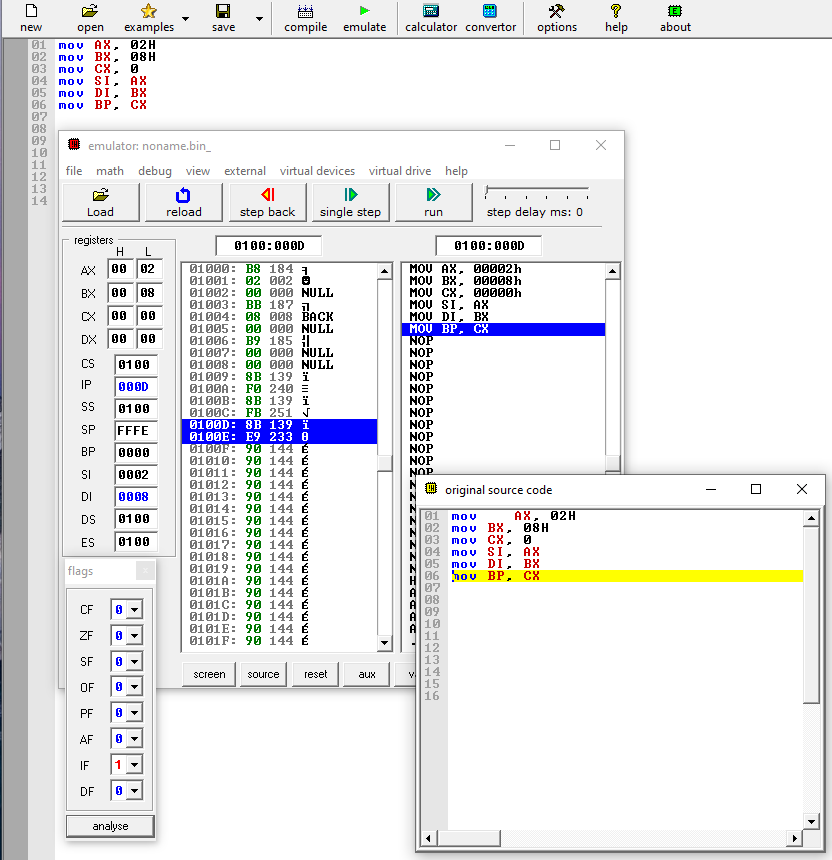


Question 2

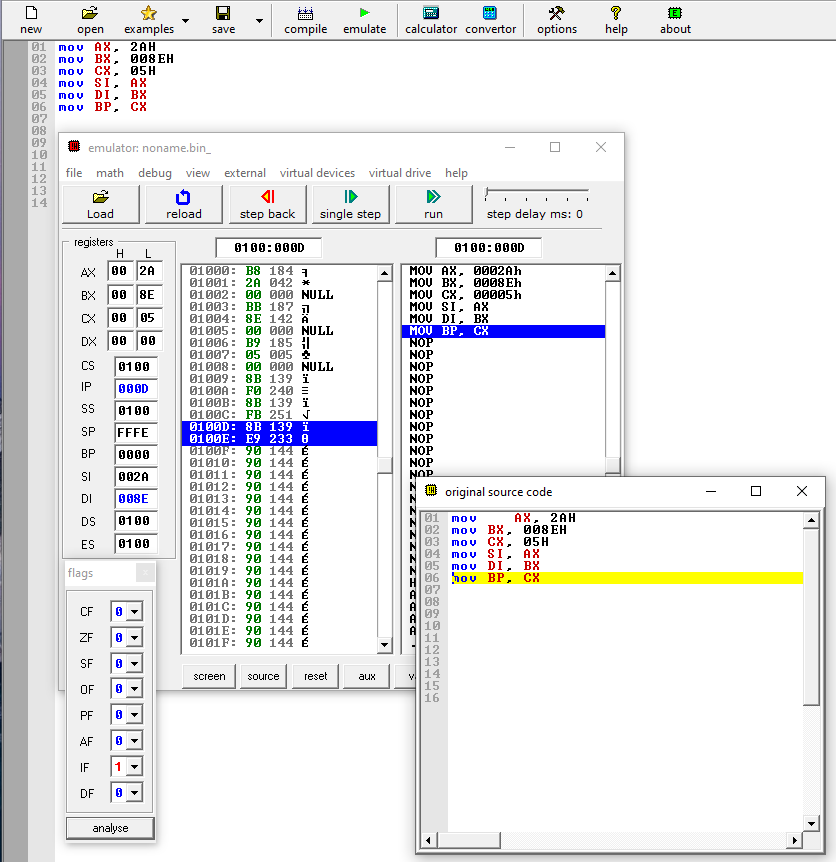


Activities

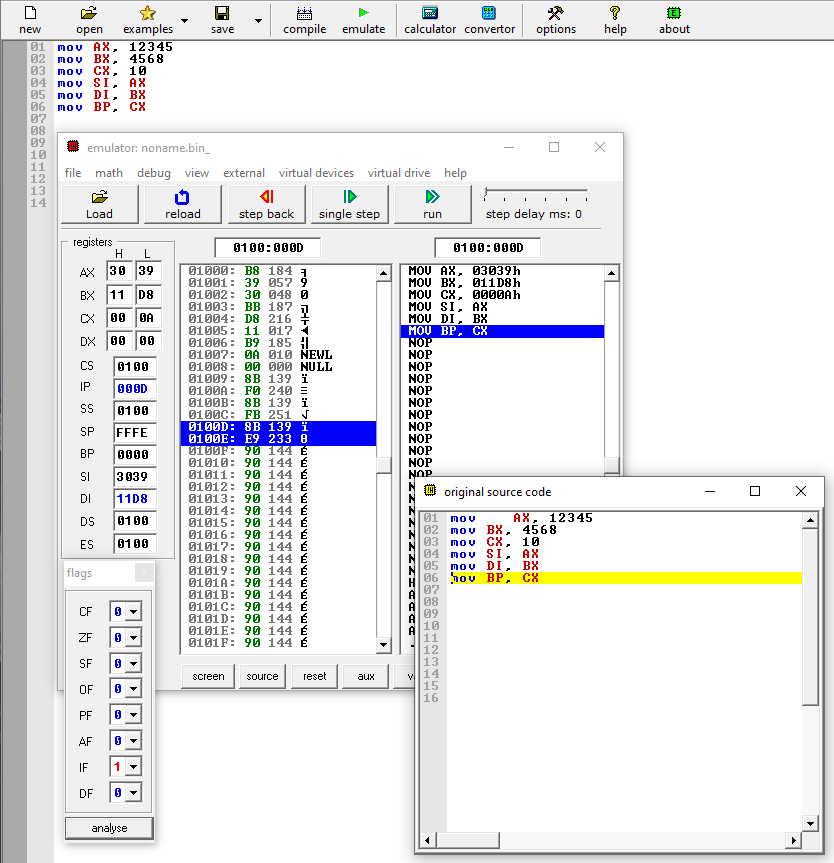
1(a)



B.

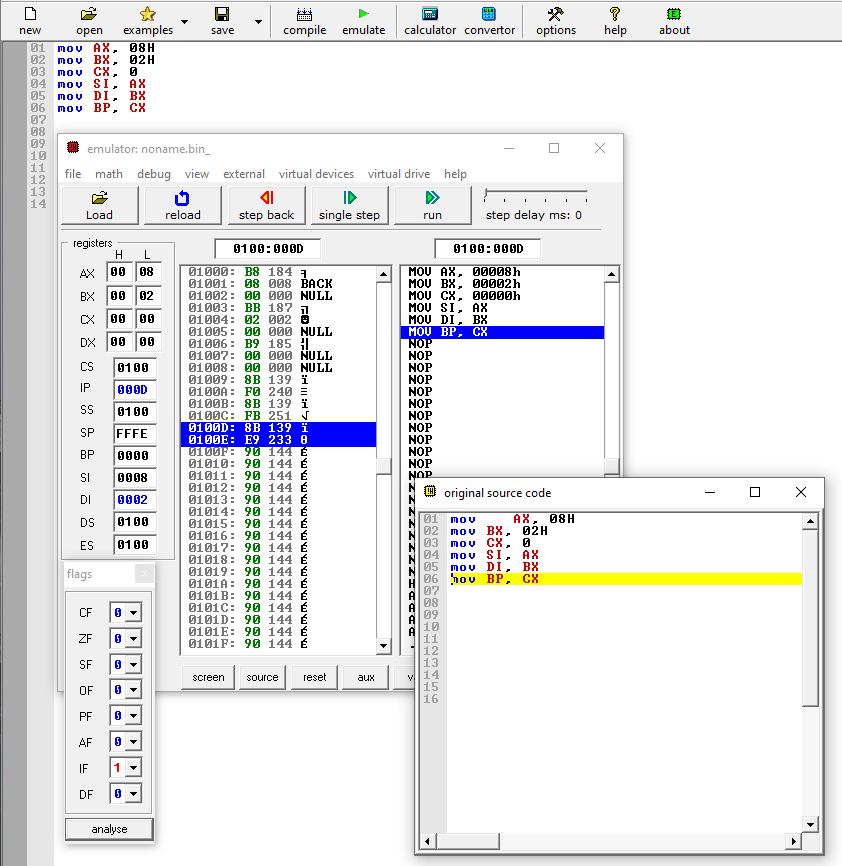


C.

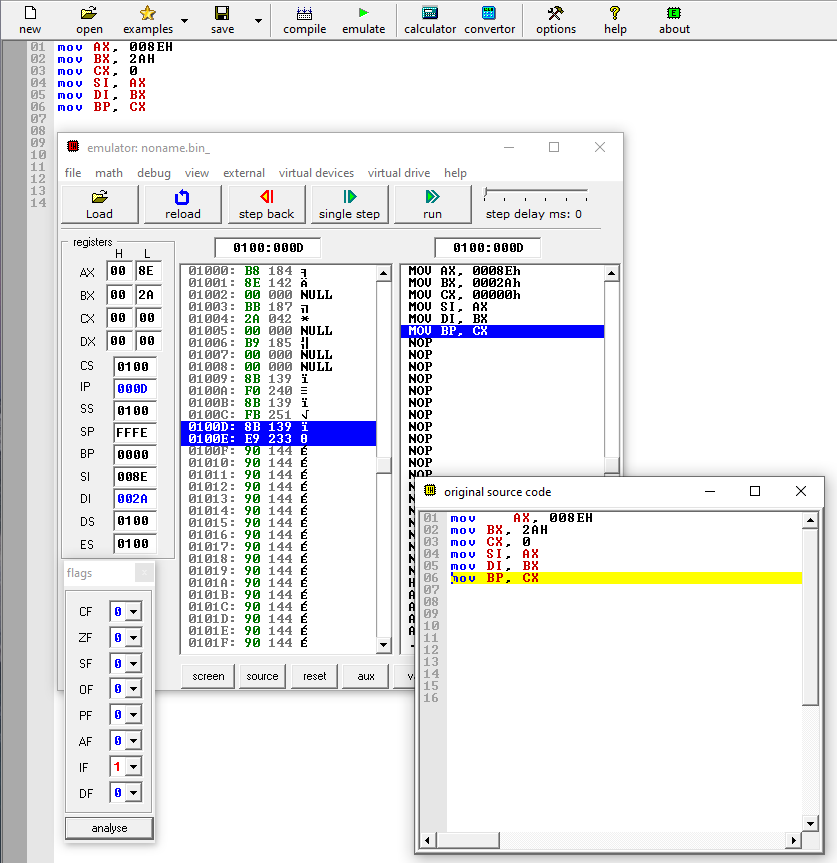


2.

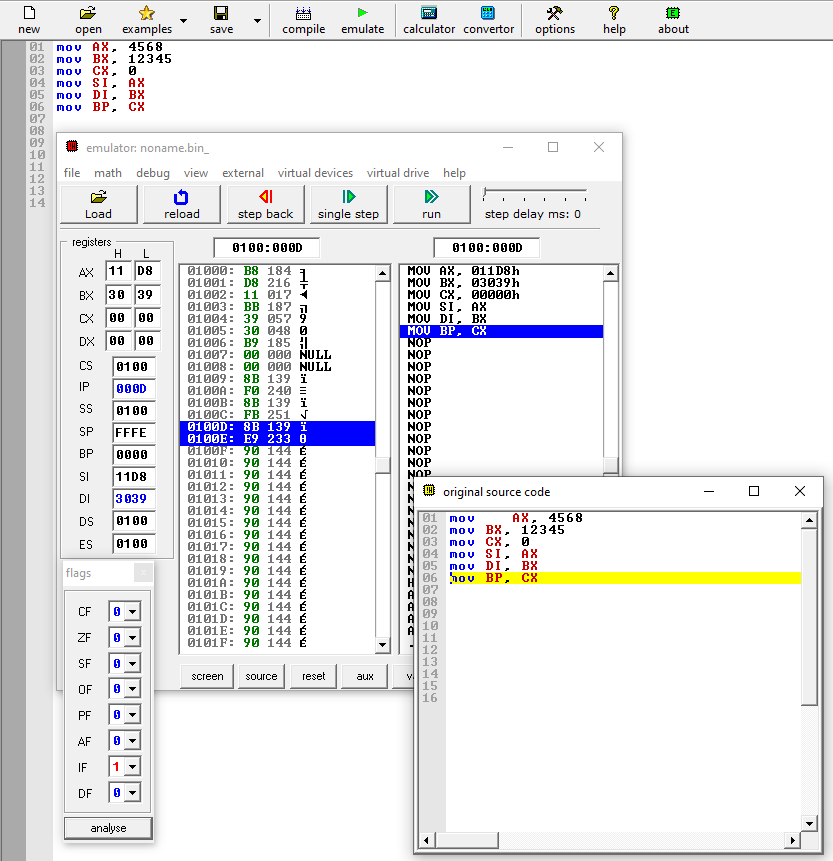
A.



B.



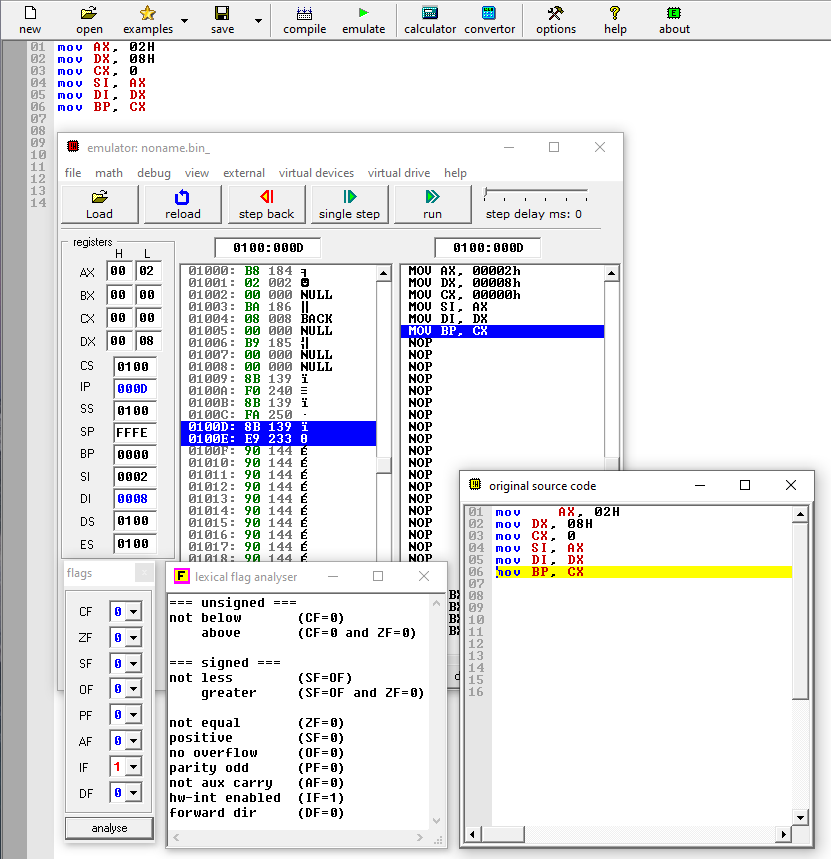
C.



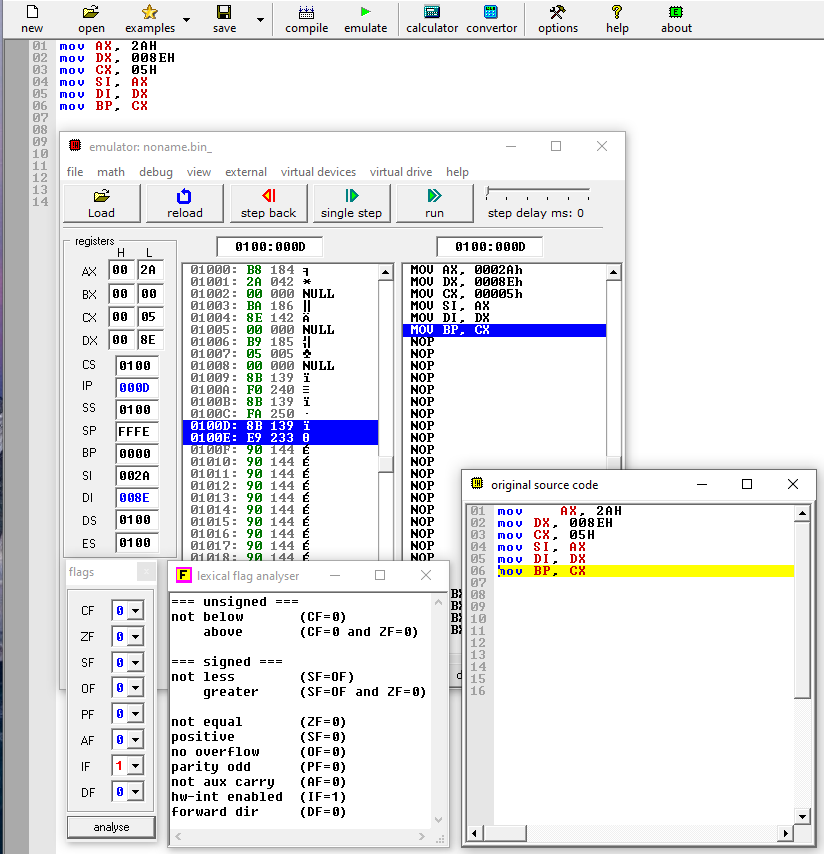
3.

Step 1

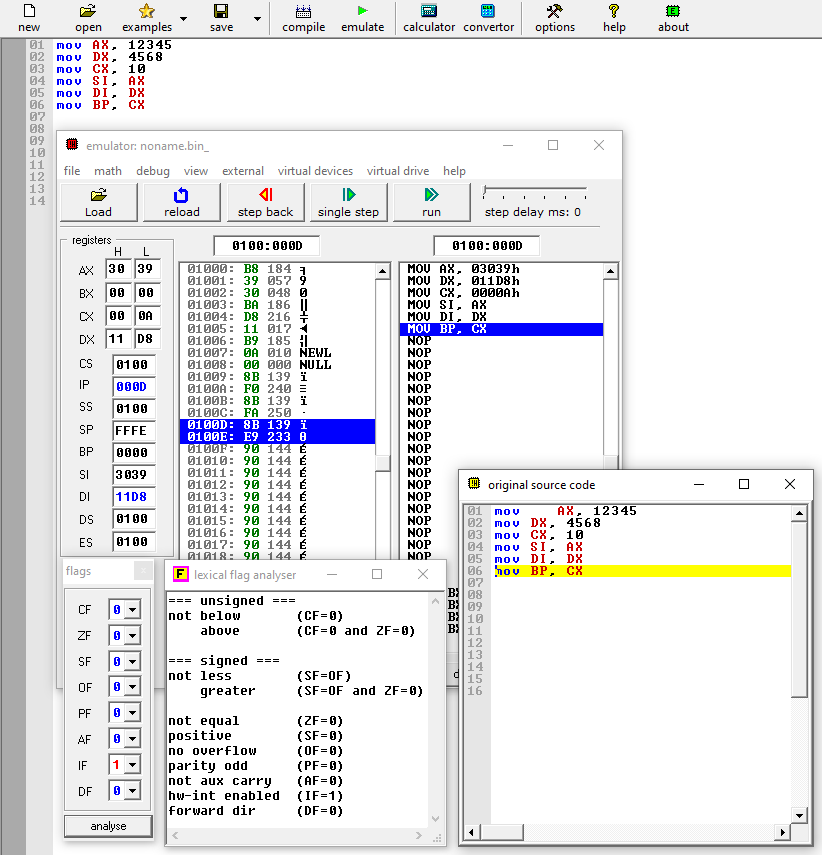
A.



B.

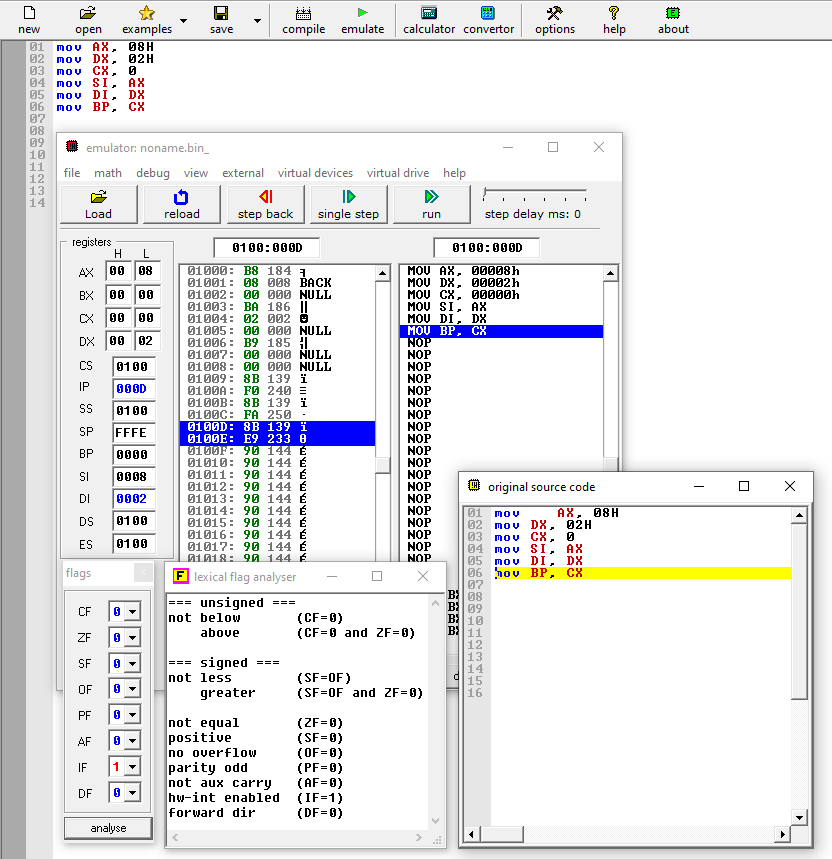


C.

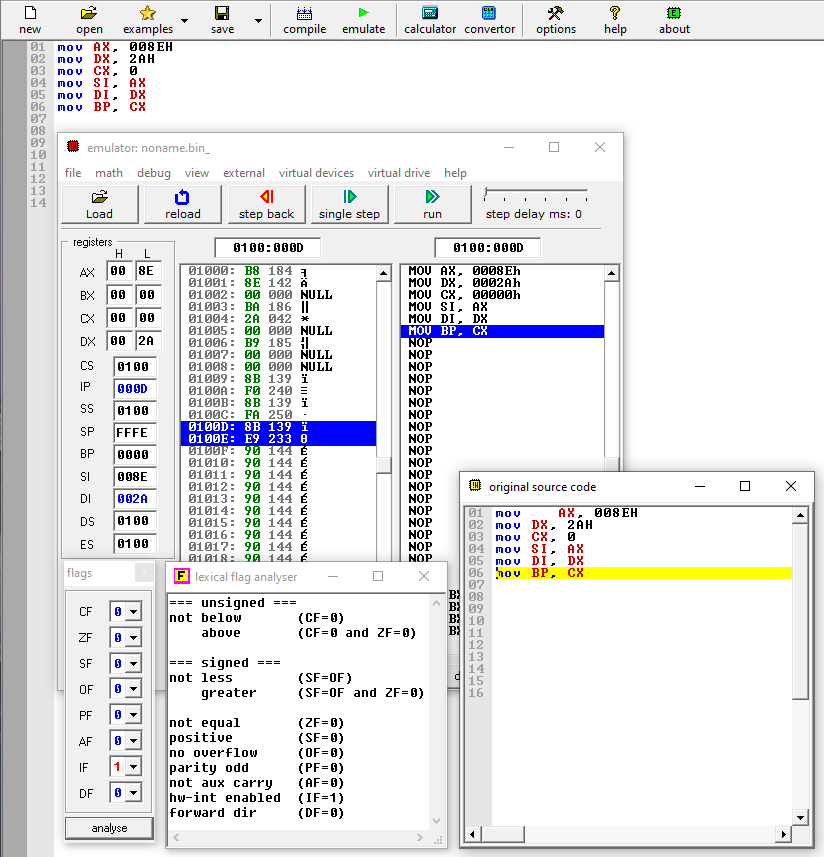


Step 2

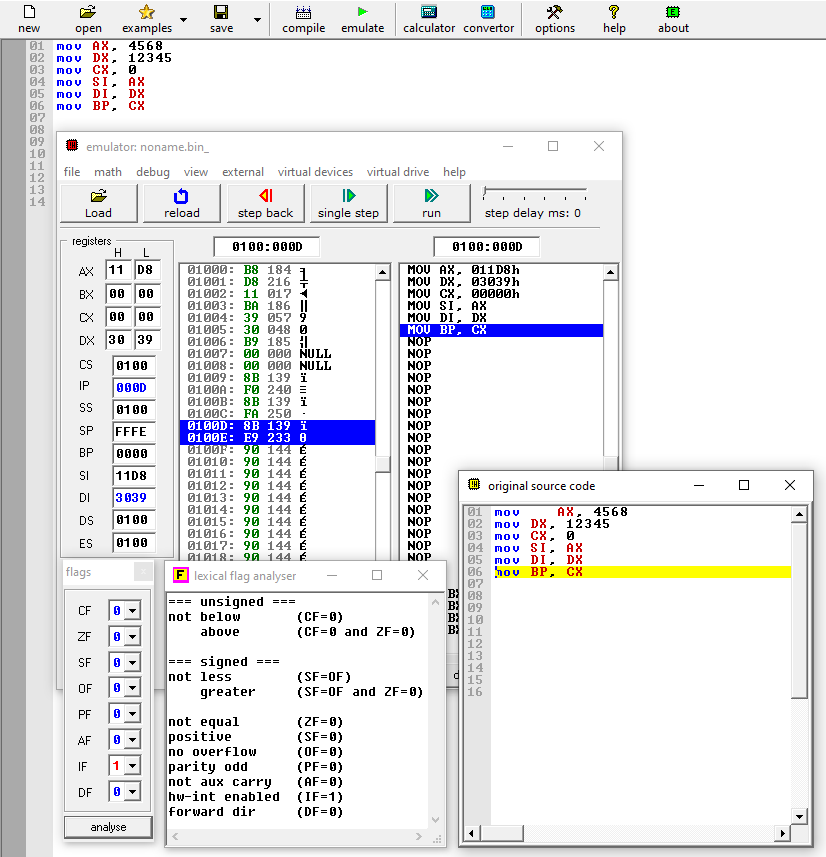
A.



B.



C.



**NO. 5**

**XCHG**

The XCHG instruction exchanges the contents of the source and destination. Both operands should be of the same type either word (16 bits) or a byte (8 bits). It does not support segment registers.  The direct exchange of data between memory locations is illegal. Both operands should be a general-purpose register. XCHG CL, 25[BX] exchanges bytes of CL with bytes stored in memory location DS: 25+BX. XCHG CS, AX is illegal.

EXAMPLE CODE

ORG 100h

MOV AX, a ; Set AX to 'a' variable

MOV CX, b ; Set CX to 'b' variable

XCHG AX, CX ; Exchange words of AX and CX

XCHG AH, CL ; Exchange bytes of AH and CL

RET ; stops the program

a DW 5C21h

b DW 3D05h

**PUSH**

It decrements the stack pointer by two and then stores the data from the source operand at the position of the stack pointer. The source operand can be a general-purpose register, segment register or a memory address but it should be a word.

The PUSH instruction decrements the SP by 2. You can see in the output the SP=FFFC which decrements by 2 becomes FFFA. The data of AX is pushed to memory location DS: FFFA which is 16FFA in this example.

ORG 100h

MOV AX, 1C2Bh ;Set AX to 1C2B

PUSH AX ;Set SP=SP-2

;Push AX data to stack location DS:SP

MOV BX, 302Ah ;Set BX to 302A

PUSH BX ;SP=SP-2

;Push BX data to stack location DS:SP

RET ;stops the program

**POP**

The POP instruction loads the word from the stack pointed by SP and then increments the SP by 2. The format for this instruction is: POP destination

The destination operand can be a general-purpose register, segment register, or memory address. The POP instruction does not support CS as a destination operation.

Example

ORG 100h

MOV AX, 1C2Bh ; Set AX to 1C2B

PUSH AX ; Set SP=SP-2

; Push AX data to stack location DS: SP

MOV BX, 302Ah ; Set BX to 302A

PUSH BX ;SP=SP-2

; Push BX data to stack location DS:SP

POP CX ; Load the last value stored at stack into CX which is BX

;incrementSPby 2 i.e., SP=SP+2

POP [12H] ;Load the value from location of SP to memory address DS:0012H

;increment SP by 2

RET ;stops the program

**IN**

The IN instruction takes the input from the port and transfers that data into the register. Thus, data transfer takes place between register and I/O device. The syntax of IN instruction is: IN reg, port address

If you want to access a port number over 255 then first load the port address into DX and then use IN instruction.

MOV DX, FA32H

IN AX, DX

**OUT**

The OUT instruction outputs the data of register on to a port specified in the instruction. The syntax of this instruction is:OUT Port Address, Register

Example

OUT 26H, AL

If you want to use port address over 255, then store this port address to DX and then execute OUT instruction

MOV DX, 456DH

OUT DX, AX

**LEA**

The LEA stands for load Effective address. As the name implies, it takes the data from the source and copies it to the destination operand. The destination is always a register whereas the source can be an offset address of a variable or a memory location. Both MOV and LEA instructions copy data from source to destination but the difference between them is LEA copies only offset address or a memory address to destination register. The syntax of LEA instruction is:LEA Reg, Source

Example Code

ORG 100h

MODEL SMALL

DATA

VAR DB 23h

code

MOV AX,@DATA ;Sets AX equal to the starting address of data segment

MOV DS,AX ;Copy AX to DS

LEA DX,VAR ;Loads the address of VAR

RET ;stops the program

**LDS**

The LDS instruction stores four consecutive memory locations into a specified destination register and a DS register. The format of LDS instruction is:

LDS Reg, Memory Address

The word from first two memory locations is loaded into a register and the word from the next two memory locations gets stored to DS register.

Example Assembly Code

ORG 100h

MODEL SMALL

DATA

VAR DB 23h

Code

LDS AX, VAR

RET

**RESULTS AND DISCUSSIONS**

The mov instruction is used to copy data between registers. We use registers because direct memory to memory moves are not possible. So we use registers to load and transfer data to the destination address.

We observed that it transfers contents between registers… this is no.4 in the activities as well

AS for #3 part 1 we observed that changing bx and dx had zero change in the other registers

From this we inference that the base register and data registers were used for the same function

The flow of the code … we observed that the mov function copies data from the source to the destination

CONCLUSIONS AND RECOMMENDATIONS

THE emulator 8086 is effective in learning assembly programming and we have learnt to appreciate programming with it learning assembler directives and how to code with 8086

REFERENCES

Lecturer Linus Aloo notes

The sample codes