PAMANTASAN NG LUNGSOD NG MAYNILA
(University of the City of Manila)
Intramuros, Manila

MICROPROCESSOR (LECTURE)

Activity No. 2
Assembly Language

Score

Submitted by:
Malabago, Antonio Emmanuel C.
S 1:00-7:00PM / CPE 0412-2

Date Submitted
21-10-2023

Submitted to:
Engr. Maria Rizette H. Sayo

1. Explain why each of the following MOV statements are invalid:

```
.data
bVal  BYTE    100
bVal2 BYTE    ?
wVal  WORD    2
dVal  DWORD   5
.code
    mov ds,45          ; a. Prohibited immediate move to DS register
    mov esi,wVal       ; b. Incompatible source & destination size
    mov eip,dval       ; c. EIP is inaccessible for destination
    mov 25,bval        ; d. Immediate value cannot be destination
    mov bVal2,bVal     ; e. Prohibited memory-to-memory move
```

Explanation:

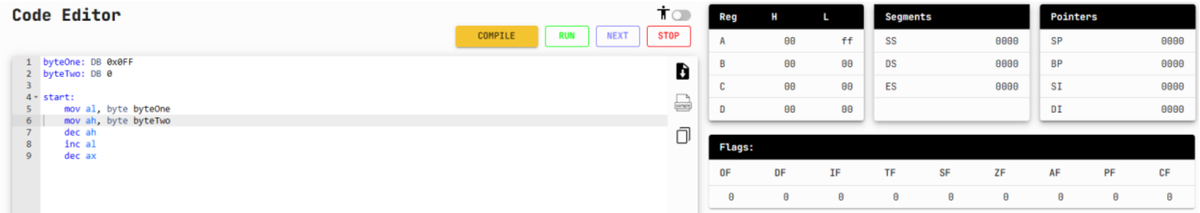
a. Prohibited immediate move to DS register.
   The DS register is one of the segment registers that can only be used to store segment addresses. Hence, attempting to move an immediate value of 45 to the DS register is unallowed and invalid.
b. Incompatible source & destination size.
   The MOV command always checks the compatibility of both the source and destination sizes. Since the variable wVal is a 16-bit type word while ESI is a 32-bit register, they are incompatible, and the move operation is invalid.
c. EIP is inaccessible for destination.
   The EIP register cannot be directly accessed for general-purpose operations since the microprocessor uses it to point to the next instruction. Hence, it is possible to move values into the EIP register.
d. Immediate value cannot be destination.
   Any immediate value cannot function as the destination register. Therefore, they can only be source values and cannot be used as an alternative to the memory location register.
e. Prohibited memory-to-memory move.
   In assembly language, data cannot be directly moved from one memory location to another. If required, load first the value in one register and then store it in another memory location.

2. Show the value of the destination operand after each of the following instructions executes:

```
.data
myByte BYTE 0FFh, 0
.code
    mov al,myByte        ; a. AL=FFh
    mov ah,[myByte+1]    ; b. AH=00h
    dec ah               ; c. AH=FFh
    inc al               ; d. AL=00h
    dec ax               ; e. AX=FEFFh
```

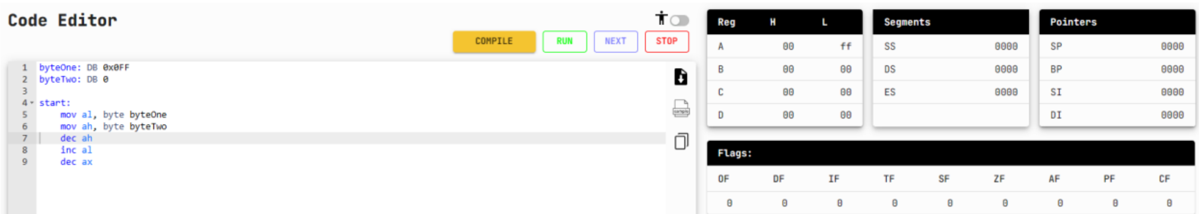
Explanation:

a. `mov al,myByte`



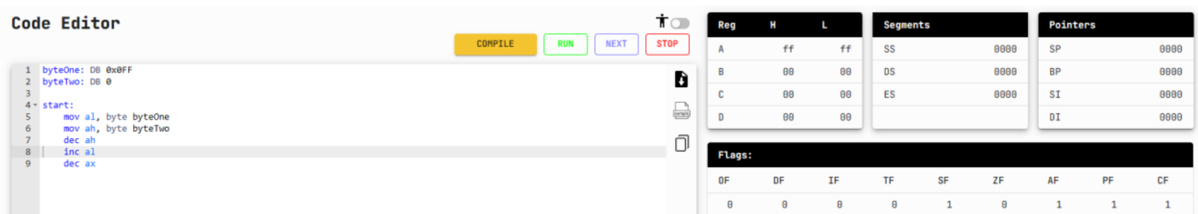
First, the myByte array was declared and initiated with two values: 0FFh and 0. Then, the first element of the myByte array was loaded to the AL register. Hence, the value of the AL register will be FFh.

b. `mov ah,[myByte+1]`



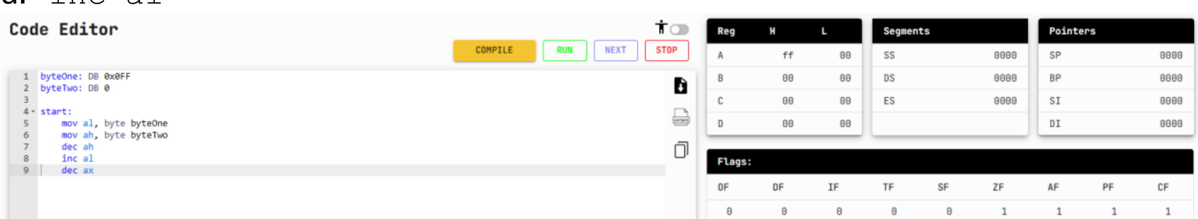
Meanwhile, the second element of the myByte array was loaded to the AX register. Afterwards, the value of the AX register will be 00h.

c. `dec ah`



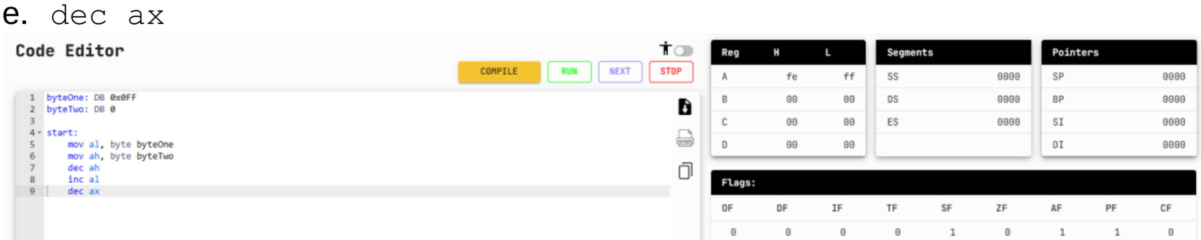
With the decrement command, the value of the AH register will be decreased by one. From 00h, its updated value will be FFh.

d. `inc al`



This increment instruction increases the value of the AL register by one. Hence, its value will be updated from FFh to 00h.

e. `dec ax`



Finally, the decrement command was used on the AX register. The recent values of AH and AL were FFh and 00h, respectively. Therefore, upon subtracting one, the value of the AX register will be FEFFh. It can be observed that the lower register borrows one bit from the higher register.

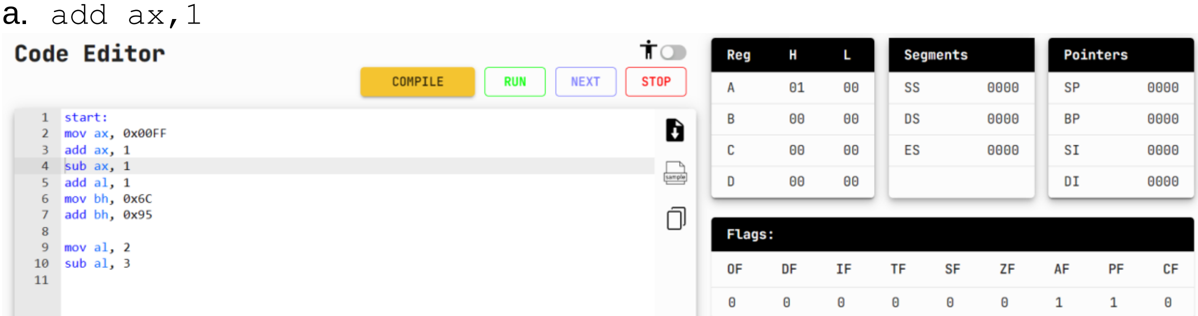
3. For each of the following marked entries, show the values of the destination operand and the Sign, Zero, and Carry flags:

```
mov ax,00FFh
add ax,1                ; a. AX=0100h SF=0 ZF=0 CF=0
sub ax,1                ; b. AX=00FFh SF=0 ZF=0 CF=0
add al,1                ; c. AL=00h   SF=0 ZF=1 CF=1
mov bh,6Ch
add bh,95h              ; d. BH=01h   SF=0 ZF=0 CF=1

mov al,2
sub al,3                ; e. AL=FFh   SF=1 ZF=0 CF=1
```

Explanation:

Flag registers are special registers with individual bit positions that indicate the recent status of the microprocessor. The resulting flags will primarily depend on the result of an arithmetic operation. In this item, the following flags are asked for each marked entry: sign, zero, and carry flags. Sign Flag (SF) is set as the resulting most significant bit (MSB) of the operation is negative. On the other hand, Zero Flag (ZF) is only activated when the corresponding operation results in zero. Meanwhile, Carry Flag (CF) is only set if the size of the resulting unsigned operation is too large for the destination.

a. `add ax,1`

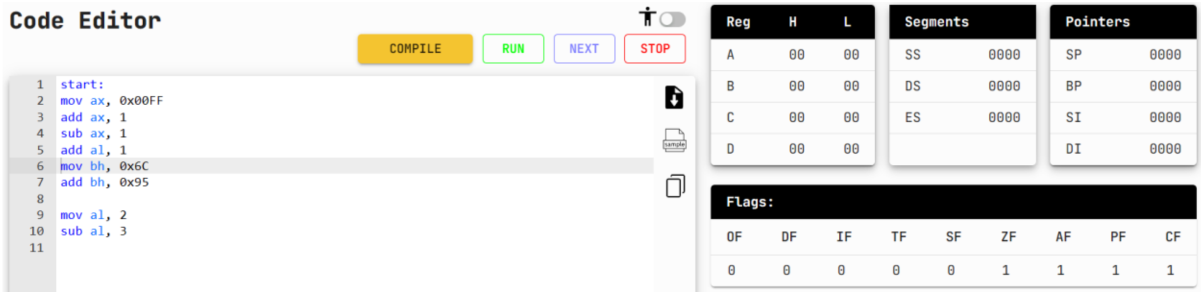


After moving the value 00FFh into the AX register, all flags remain unaffected. Then, one was added to AX, resulting in 0100h. The updated register flags are SF=0 (MSB is non-negative), ZF=0 (AX is non-zero), and CF=0 (no carry after addition).

b. `sub ax,1`

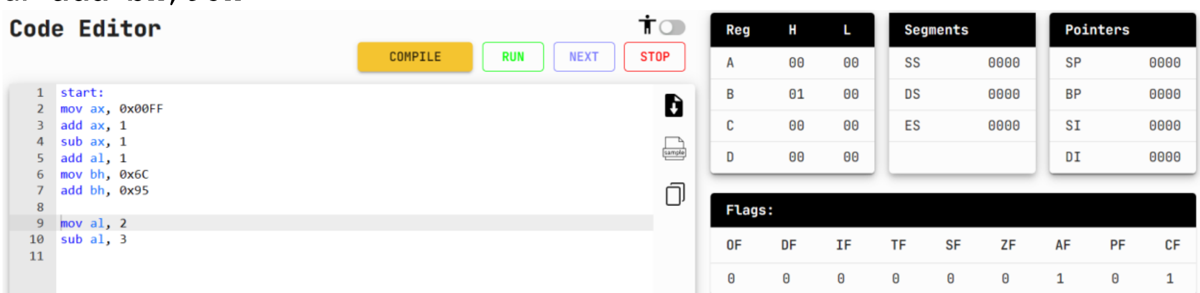
Subtracting one from AX resulted in 00FFh. The updated register flags are SF=0 (MSB is non-negative), ZF=0 (AX is non-zero), and CF=0 (no borrow upon subtraction).

c. `add al,1`



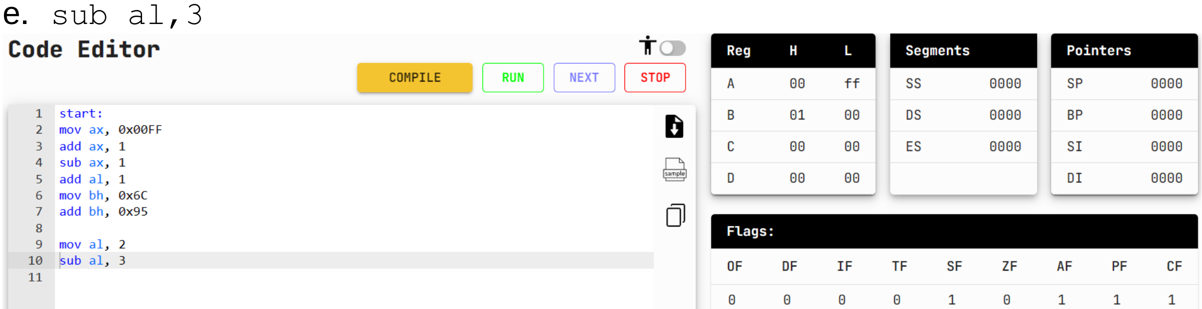
Adding 1 to the lower byte AL will result in 00h since its value is zero. The updated register flags are SF=0 (AL is non-negative), ZF=1 (AL is zero), and CF=1 (carry after addition).

d. `add bh,95h`



Initially, the BH register is assigned to 6Ch, and no flags are affected. Then, 95h was added to BH, resulting in 01h. The updated register flags are SF=0 (BH is non-negative), ZF=0 (BH is non-zero), and CF=1 (carry after addition).

e. `sub al,3`



Moving the value two into the AL register will overwrite its previous value. Afterwards, the values of 3 were subtracted, resulting in FFh. The updated register flags are SF=1 (AL is negative), ZF=0 (AL is non-zero), and CF=1 (borrow upon subtraction).

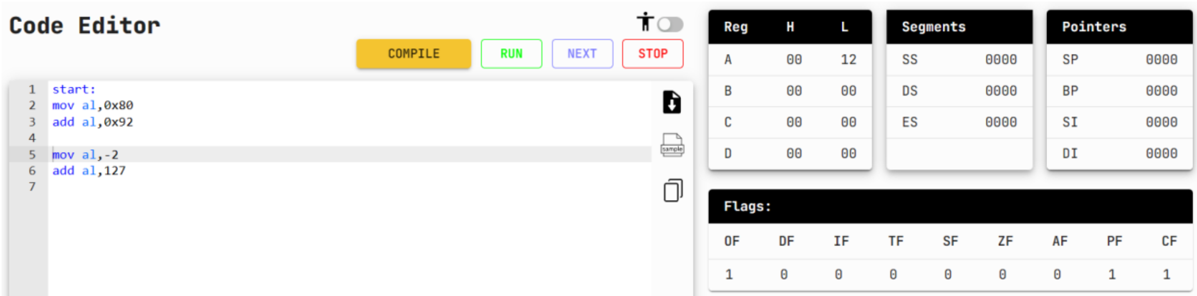
4. What will be the value of the Overflow flag?

```
mov al,80h
add al,92h                 ; a. OF=1

mov al,-2
add al,+127                ; b. OF=0
```

Explanation:

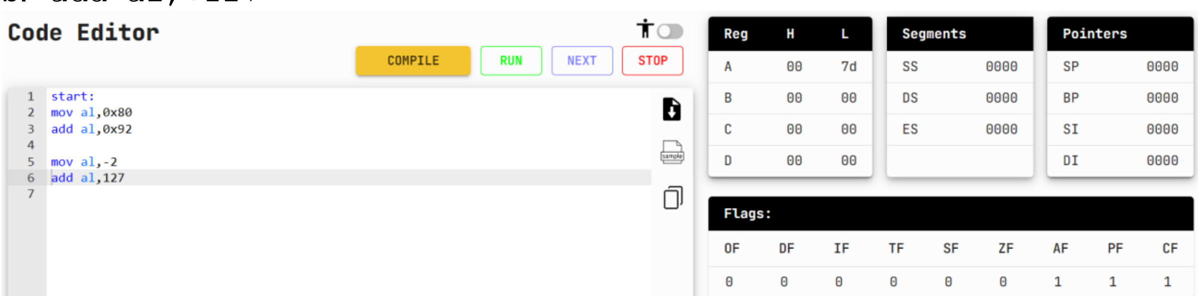
Another type of flag register is the Overflow Flag (OF). OF is only set when the resulting signed integer from an operation is either invalid or out of range. This is directly dependent on the capacity of the certain register to handle the resulting values from an operation.

a. `add al,92h`



The value of 80h was moved in the AL register. Afterwards, 92h was added, which resulted in an overflow. Consequently, 12h is the recorded value for AL instead of 172h. Due to this, the subsequent overflow flag (OF) is 1.

b. `add al,+127`



Initially, the AL register was assigned to -2. Then, 127 was added, which resulted in 7Dh. Since the output value does not exceed the 8-bit maximum limit, the subsequent overflow flag (OF) is 0.

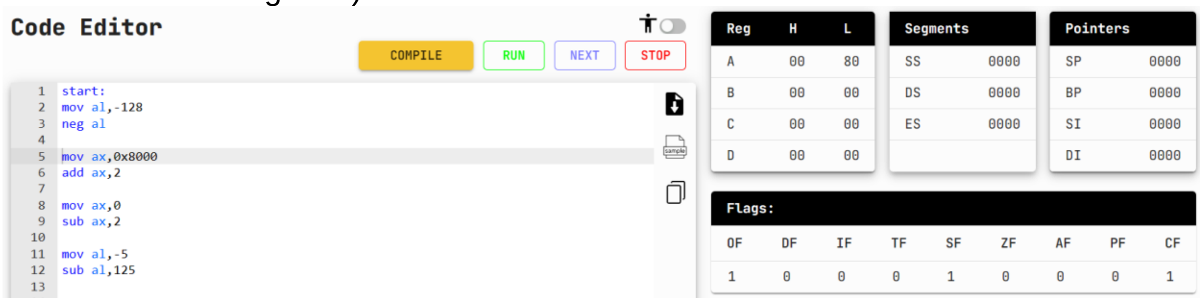
5. What will be the value of the Carry and Overflow flags after each operation?

```
mov al,-128
neg al                  ; a. CF=1 OF=1

mov ax,8000h
add ax,2                ; b. CF=0 OF=0

mov ax,0
sub ax,2                ; c. CF=1 OF=0

mov al,-5
sub al,+125             ; d. CF=0 OF=1
```

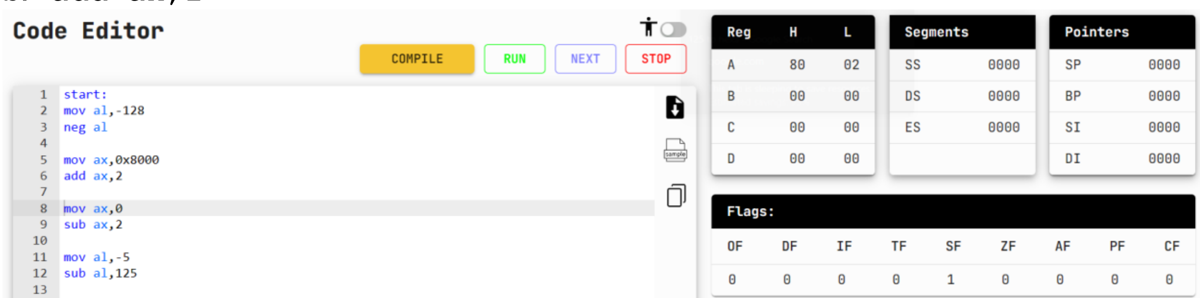
Explanation:

a. `neg al`

After moving the value -128 into the AL register, all flags remain unaffected. Afterwards, the value at AL was negated, resulting in 80h (two's complement of -128). The updated register flags are CF=1 (carry during negation) and OF=1 (signed overflow due to negation).

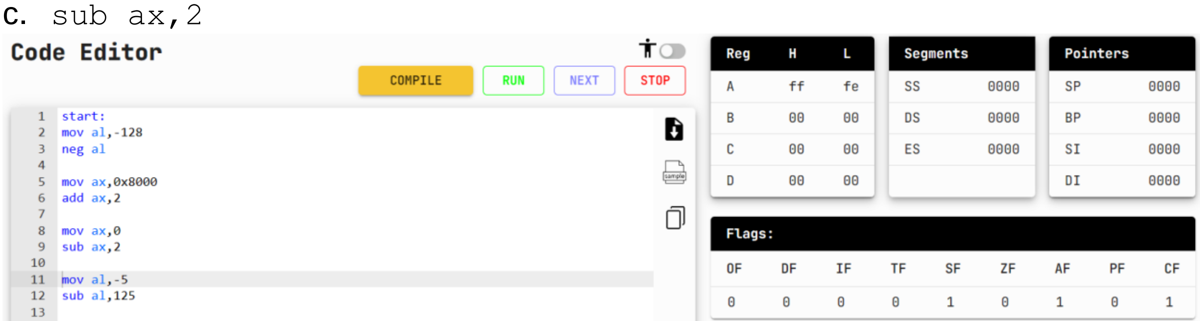


b. `add ax,2`



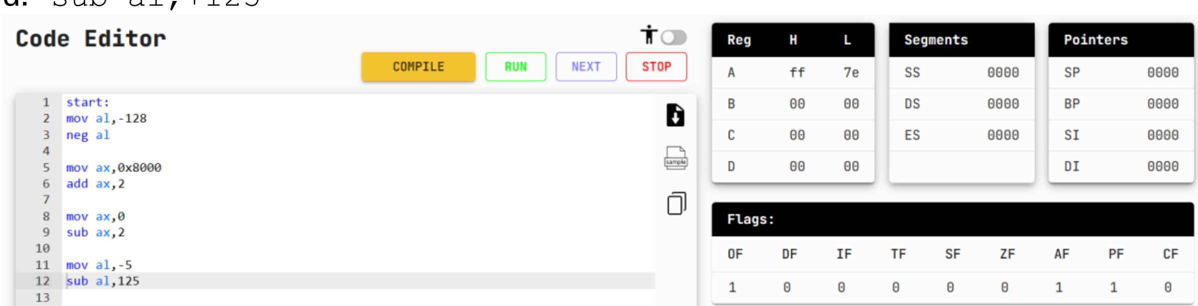
The value of 8000h is moved into the AX register. Then, the value at AL was increased by 2. The updated register flags are CF=0 (no carry after addition) and OF=0 (no signed overflow).

c. `sub ax,2`



Subsequently, the value of 0 is assigned to the AX register. Afterwards, two were subtracted, resulting in the value of FFFEh. The updated register flags are CF=1 (borrow during subtraction) and OF=0 (no signed overflow).

d. `sub al,+125`



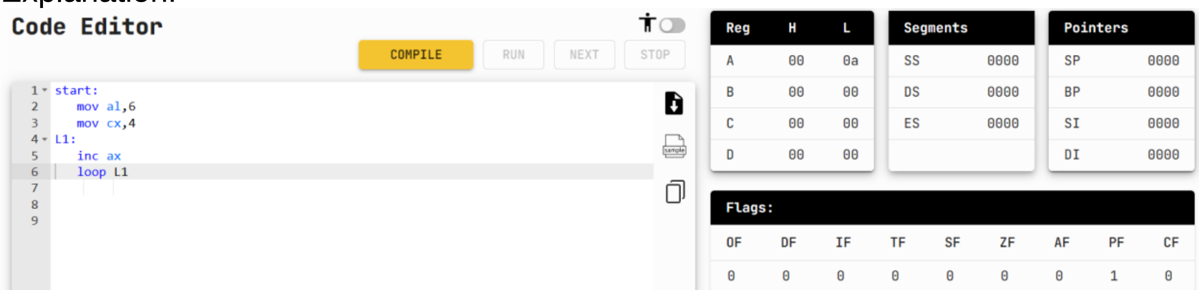
Meanwhile, -5 was loaded into the AL register. Then, 125 was subtracted from the same register, resulting in AL having the updated value of -130 or 7Eh. Hence, there will be an overflow since it is beyond the 8-bit range of the AL register. The final register flags are CF=0 (no borrow during subtraction) and OF=1 (signed overflow).

6. What will be the final value of AX?

```
    mov al,6
    mov ecx,4
L1:
    inc ax
    loop L1                    ; AX=10
```

Answer: 10

Explanation:



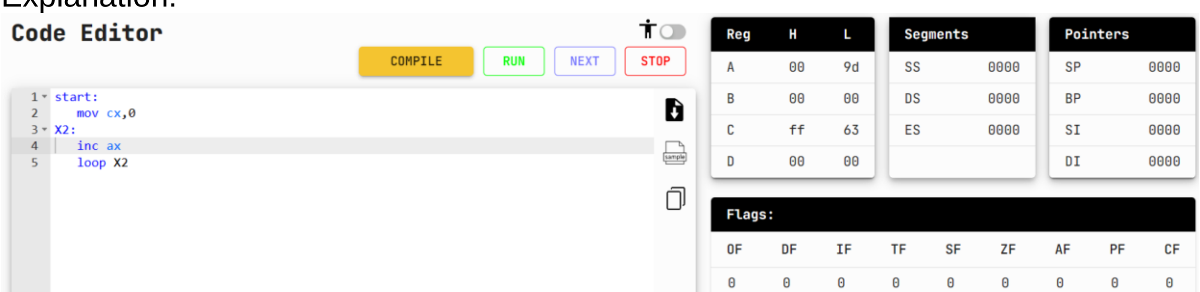
First, the value of 6 was assigned to the AL register. Then, the value of 4 was loaded in the CX register, often used as a loop counter. `L1` signifies the beginning of the loop. For every iteration, the AX value will be increased by 1. Meanwhile, the instruction `loop L1` decreases the value of CX by 1 for every loop and then jumps to the `L1` instruction. This loop will continue to execute as long as the CX register is not zero. At the first iteration, the value of AX was 7, while CX was 3. At the second iteration, the value of AX was 8, while CX was 2. At the third iteration, the value of AX was 9, while CX was 1. At the fourth iteration, the value of AX was 10, while CX was 0. Since CX is already 0, `loop L1` will prevent the program from iterating again.

How many times will the loop execute?

```
    mov ecx,0
X2:
    inc ax
    loop X2                    ; 4,294,967,296 times
```

Answer: 4,294,967,296

Explanation:



Initially, the value 0 was loaded to the CX register. Then, `X2` marks the beginning of the loop. For the first iteration, the AX value was increased by 1, while the CX register was decreased by 1. Hence, the updated values are AX=1 and CX=-1. However, `loop X2` will not terminate the program because, in the assembly language, the loop counter of -1 is understood as the unsigned integer 4,294,967,295 or 2^32 - 1 (two's complement of -1). Therefore, the loop will execute for 4,294,967,296 times.

References

[1] “Microprocessor - 8086 instruction sets.” Available: https://www.tutorialspoint.com/microprocessor/microprocessor_8086_instruction_sets.htm
[2] “Instruction Set of 8086 - javatpoint,” *www.javatpoint.com*. Available: https://www.javatpoint.com/instruction-set-of-8086
[3] “8086 instructions.” Available: https://www.eng.auburn.edu/~sylee/ee2220/8086_instruction_set.html
[4] “Instruction Set of 8085 - javatpoint,” *www.javatpoint.com*. Available: https://www.javatpoint.com/instruction-set-of-8085
[5] “Microprocessor - 8085 instruction sets.” Available: https://www.tutorialspoint.com/microprocessor/microprocessor_8085_instruction_sets.htm
[6] “8086 Compiler.” Available: https://yjdoc2.github.io/8086-emulator-web/