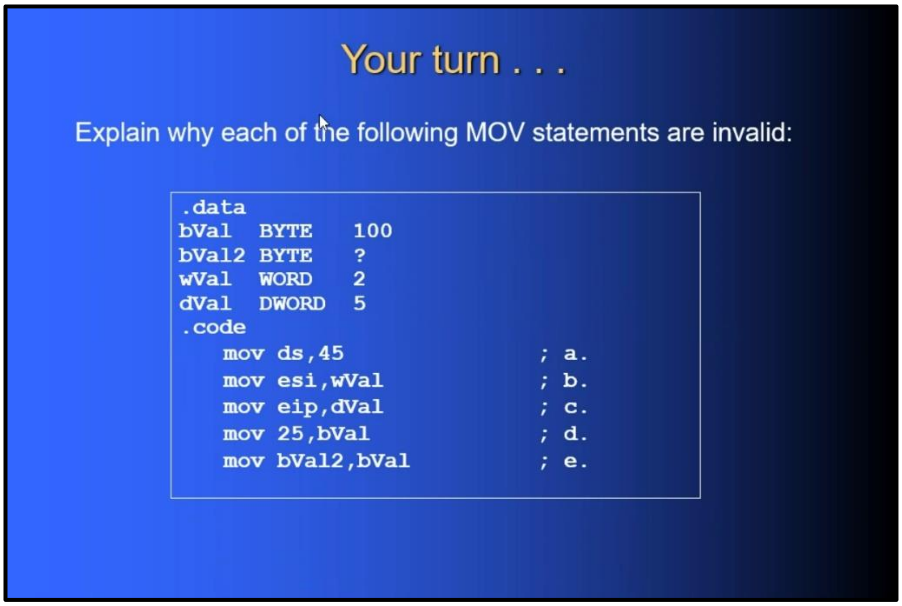
Manahan, Johnielle Mike T.

2020-12007

Microprocessors CPE 0412-2

October 21, 2023

Problem # 1



a. **mov ds,45** is invalid because segment registers cannot be loaded with immediate values.

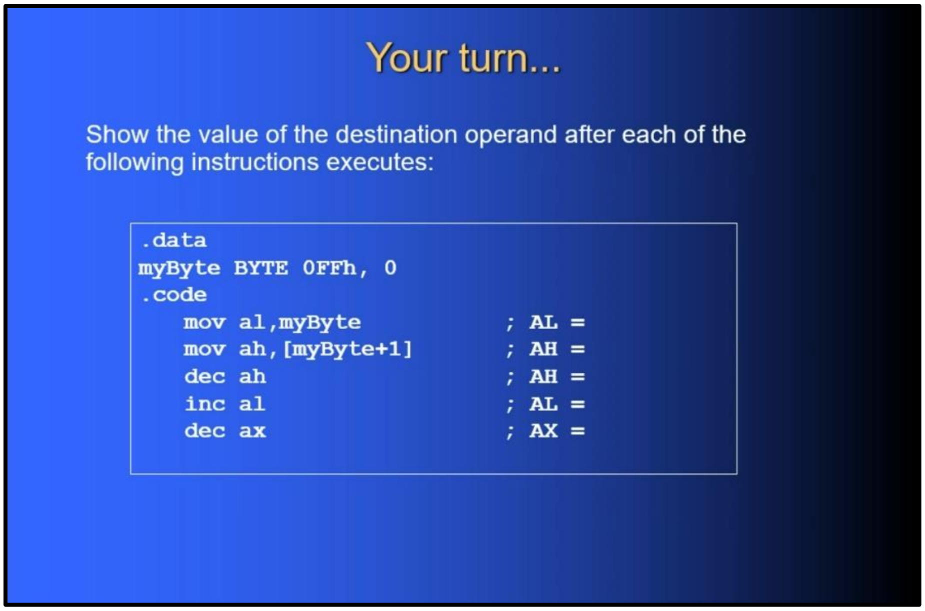
b. **mov esi,wVal** is invalid because the source and destination operands must have the same size. In this case, esi is a 32-bit register, but wVal is a 16-bit variable.

c. **mov eip,dVal** is invalid because the eip register cannot be modified directly. The eip register contains the address of the next instruction to be executed, and modifying it can cause the program to crash.

d. **mov 25, bVal** is invalid because the destination operand must be a register or memory location. In this case, 25 is an immediate value.

e. **mov bVal2, bVal** is invalid because the two operands cannot be the same memory location.

Problem # 2

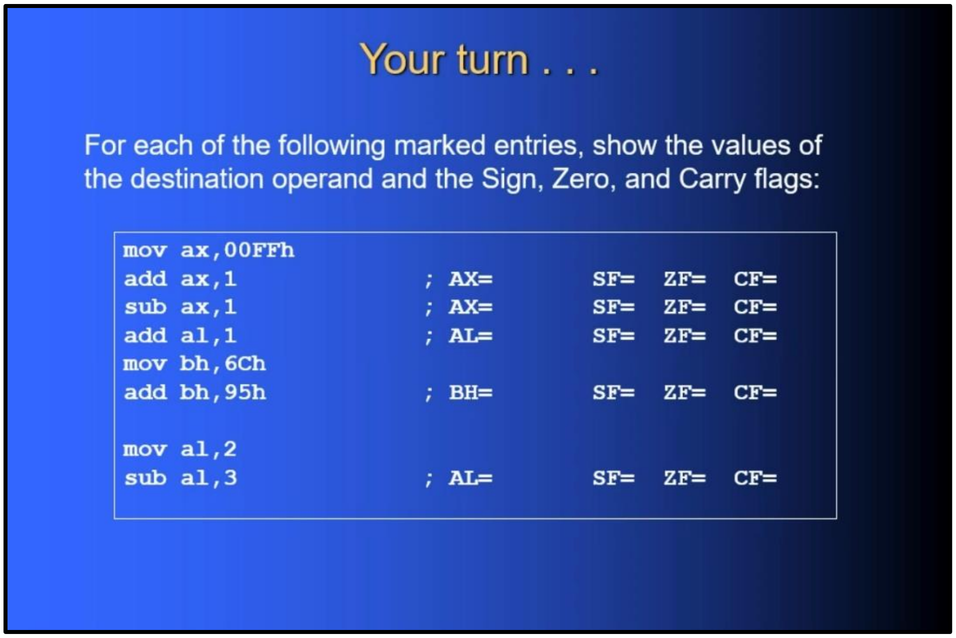


After each instruction executes, the value of the destination operand is as follows:

Instruction	Destination	Value
mov al, myByte	AL	0xFFh
mov ah, [myByte+1]	AH	0x00h
dec ah	AH	0xFFh
inc al	AL	0x00h
dec ax	AX	0xFEFFh

- The first instruction, mov al, myByte, copies the value of the first byte of the variable myByte (0xFFh) into the al register.
- The second instruction, mov ah, [myByte+1], copies the value of the second byte of the variable myByte (0x00h) into the ah register.
- The third instruction, dec ah, decrements the value of the ah register by one. This results in the value 0xFFh, since unsigned integer arithmetic wraps around.
- The fourth instruction, inc al, increments the value of the al register by one. This results in the value 0x00h, since unsigned integer arithmetic wraps around.
- The fifth instruction, dec ax, decrements the value of the ax register by one. This results in the value 0xFEFFh, since the ax register is a 16-bit register and the value 0x0000h is out of range.

Problem # 3



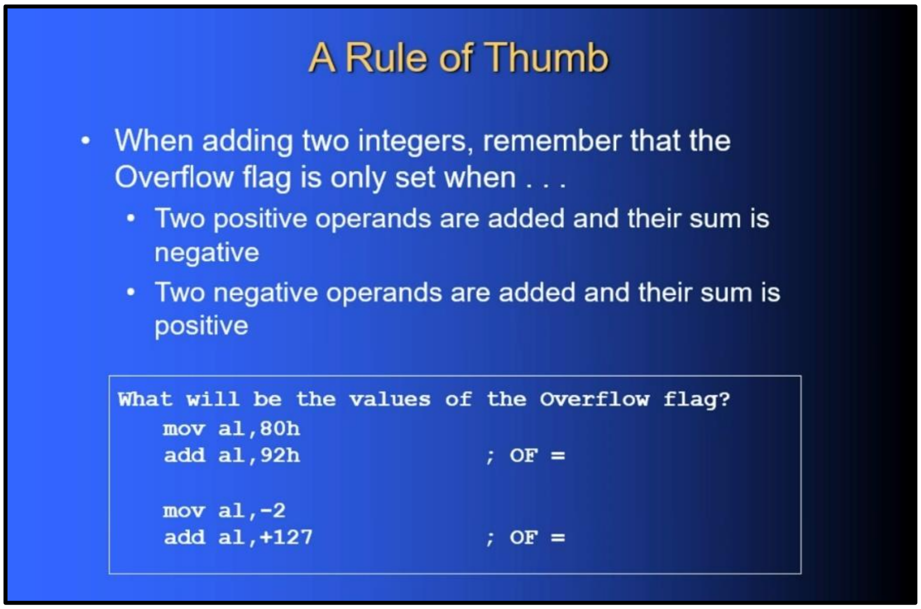
Here are the values of the destination operand and the Sign, Zero, and Carry flags for each of the following marked entries:

Instruction	Destination operand	Sign flag	Zero flag	Carry flag
add ax,1	AX = 0x100	0	0	0
sub ax,1	AX = 0xFF	1	0	0
add al,1	AL = 0x0	0	1	0
add bh,95h	BH = 0x01	0	1	1
sub al,3	AL = 0xFF	1	1	1

- add ax,1: This instruction adds the value 1 to the ax register. The result is 0x100, which is greater than zero, so the Sign flag is cleared. The Zero flag is cleared because the result is not zero. The Carry flag is cleared because there is no carry out of the most significant bit.
- sub ax,1: This instruction subtracts the value 1 from the ax register. The result is 0xFF, which is less than zero, so the Sign flag is set. The Zero flag is cleared because the result is not zero. The Carry flag is cleared because there is no carry out of the most significant bit.
- add al,1: This instruction adds the value 1 to the al register. The result is 0x0, which is equal to zero, so the Sign flag is cleared. The Zero flag is set because the result is zero. The Carry flag is cleared because there is no carry out of the most significant bit.
- add bh,95h: This instruction adds the value 95h to the bh register. The result is 0x01, which is greater than zero, so the Sign flag is cleared. The Zero flag is cleared because

the result is not zero. The Carry flag is set because there is a carry out of the most significant bit.
- sub al,3: This instruction subtracts the value 3 from the al register. The result is 0xFF, which is less than zero, so the Sign flag is set. The Zero flag is cleared because the result is not zero. The Carry flag is set because there is a carry out of the most significant bit.

Problem # 4



The value of the Overflow flag will be set after both of the following instructions execute:

mov al,80h

add al,92h

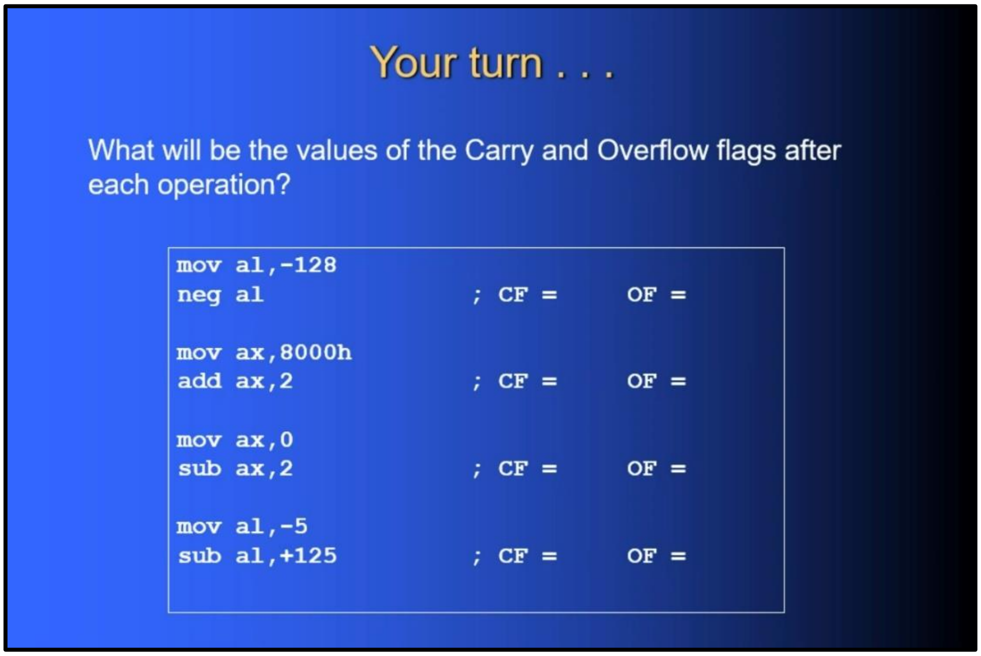
mov al,-2

add al,+127

- mov al,80h: This instruction loads the value 0x80h into the al register.
- add al,92h: This instruction adds the value 0x92h to the al register. The result, 0x112h, is out of range for the 8-bit al register, so the Overflow flag is set.
- mov al,-2: This instruction loads the signed value -2 into the al register.
- add al,+127: This instruction adds the signed value 127 to the al register. The result, 125, is out of range for the 8-bit al register, so the Overflow flag is set.

It is important to note that the Overflow flag is only set if the signed result of an operation is out of range for the destination operand. If the result is out of range but unsigned integer arithmetic wraps around, the Overflow flag will not be set.

Problem # 5

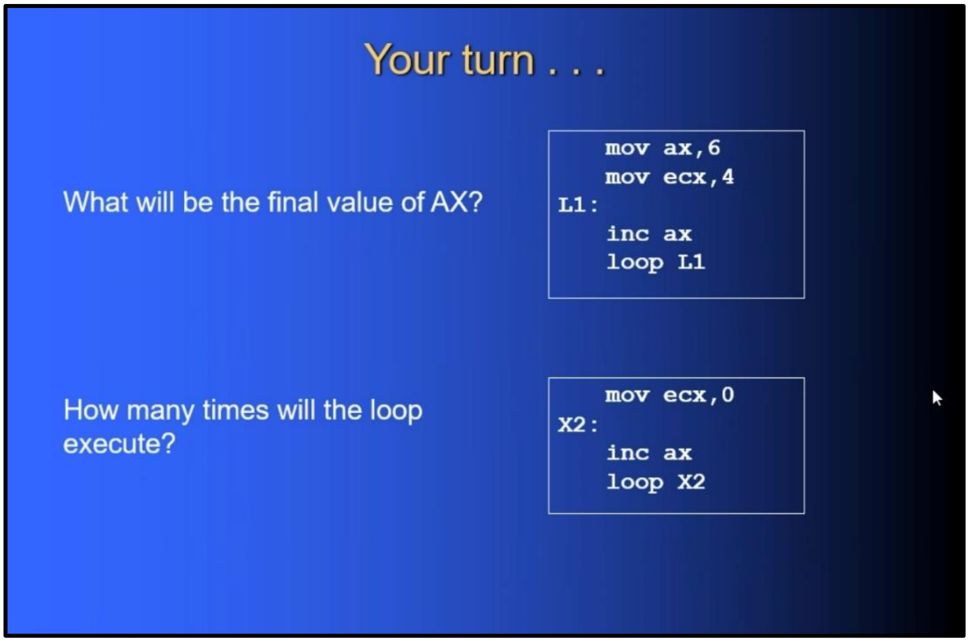


The values of the Carry and Overflow flags after each operation are as follows:

Operation	CF	OF
neg al	1	1
add ax,2	0	0
sub ax,2	1	0
sub al,+125	1	1

- neg al: This instruction negates the value of the al register. The result, 128, is out of range for the 8-bit al register, so the Carry and Overflow flags are set.
- add ax,2: This instruction adds the value 2 to the ax register. The result, 8002h, is within the range of the 16-bit ax register, so the Carry and Overflow flags are cleared.
- sub ax,2: This instruction subtracts the value 2 from the ax register. The result, 8000h, is equal to the most negative value that can be represented by a signed 16-bit integer. Subtracting from this value will result in a carry out of the most significant bit, so the Carry flag is set. The Overflow flag is not set because the result is still within the range of a signed 16-bit integer.
- sub al,+125: This instruction subtracts the value 125 from the al register. The result, -130, is out of range for the 8-bit al register, so the Carry and Overflow flags are set.

Problem # 6



a. The final value of AX will be 10.

   The loop instruction repeats the loop body as long as the CX register is not zero. The loop body in this case is the inc ax instruction, which increments the AX register.

   The loop will start with AX = 6 and CX = 4. The inc ax instruction will increment AX to 7. The loop instruction will then check the value of CX. Since CX is still not zero, the loop body will be repeated.

   The inc ax instruction will increment AX to 8. The loop instruction will then check the value of CX again. Since CX is still not zero, the loop body will be repeated again.

   The inc ax instruction will increment AX to 9. The loop instruction will then check the value of CX again. Since CX is still not zero, the loop body will be repeated one last time.

   The inc ax instruction will increment AX to 10. The loop instruction will then check the value of CX again. Since CX is now zero, the loop will terminate.

   Therefore, the final value of AX will be 10.

b. The loop will execute zero times.

   The loop instruction repeats the loop body as long as the CX register is not zero. In this case, the CX register is initialized to 0, so the loop body will never be executed.

   The inc ax instruction will be executed once, but it will have no effect because the loop body is never executed.

Therefore, the final value of AX will be 0, and the loop will have executed zero times.