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

Microprocessor

Score

Submitted by:

Gomez, Ericka Mae S.

<Saturday 1:00pm-7:00pm> / <CPE0412.1-2>

Date Submitted

21-10-2023

Submitted to:

Engr. Maria Rizette H. Sayo

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.
    mov esi,wVal     ;b.
    mov eip,dVal     ;c.
    mov 25,bVal      ;d.
    mov bVal2,bVal   ;e.
```

The provided instructions demonstrate several invalid uses of the "mov" instruction in assembly language. Firstly, trying to move a value into a segment register like "ds" is not allowed. Secondly, attempting to move a 16-bit variable, "wVal," into a 32-bit register like "esi" is not permitted. Moreover, directly modifying the instruction pointer "eip" is discouraged, as it can lead to unpredictable behavior. Additionally, "mov" cannot be used to assign a value directly to a memory location as in "mov 25, bVal." Lastly, trying to move a byte-sized variable, "bVal2," into an integer-sized variable, "bVal," is not allowed. In assembly language, the "mov" instruction is typically used to transfer values between registers and memory locations of the same size, and altering the instruction pointer directly should be avoided.

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

```
.data
myByte  BYTE 0FFh, 0
.code
    mov al,myByte           ; AL =
    mov ah, [myByte+1]      ; AH =
    dec ah                  ; AH =
    inc al                  ; AL =
    dec ax                  ; AX =
```

mov al,	myByte	; AL = 0FFh (signed -1 in two's complement)
mov ah,	[myByte+1]	; AH = 0 (since the second byte of myByte is 0)
dec ah		; AH = FFh (signed -1 in two's complement)
inc al		; AL = 00h (overflow from FFh results in 0)
dec ax		; AX = FFFFh (overflow from 00h results in FFFFh, -1 in two's complement)

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

```
1 mov ax, 00FFh
2 add ax, 1        ;AX =    SF= ZF CF=
3 sub ax, 1        ;AX =    SF= ZF CF=
4 add al, 1        ;AL =    SF= ZF CF=
5 mov bh, 6Ch
6 add bh, 95h      ;BH =    SF= ZF CF=
7
8 mov al, 2
9 sub al, 3        ;AL =    SF= ZF CF=
```

Instruction	Destination Operand	SF	ZF	Cf
mov ax, 00FFh	AX = 00FFh	0	0	0
add ax, 1	AX = 0100h	0	0	0
sub ax, 1	AX = 00FFh	0	0	0
add al, 1	AL = 00h	0	1	0
mov bh, 6Ch	BH = 6Ch	0	0	0
add bh, 95h	BH = 01h	0	1	0
mov al, 2	AL = 2	0	0	0
sub al, 3	AL = FFh	1	0	1

When adding two integers, remember that the Overflow flag is only set when…

a. **Two positive operands are added and their sum is negative**
b. **Two negative operands are added and their sum is positive**

```
1 mov al, 80h
2 add al, 92h          ;OF =
3
4 mov al, -2
5 add al, +127         ;OF =
```

a. mov al, 80h and add al, 92h: In this case, both operands are positive, but when you add them, the result is 112h, which is positive. This does not trigger an overflow, so OF = 0.

b. mov al, -2 and add al, +127: Here, the first operand is negative, and the second operand is positive. When you add them, the result is 7Dh, which is positive. This also does not trigger an overflow, so OF = 0.

In both cases, the result of the addition is within the valid range for signed 8-bit integers, so the Overflow flag is not set. The Overflow flag is typically set when the sign of the result doesn't match the expected sign for the operation.

What will be the values of the Carry and Overflow flags after each operation?

```
1  mov al, -128
2  neg al,              ; CF=   OF=
3
4  mov ax, 8000h
5  add ax, 2            ; CF=   OF=
6
7  mov ax, 0
8  sub ax, 2            ; CF=   OF=
9
10 mov al, -5
11 sub al, +!25         ; CF=   OF=
```

Operation	CF	OF
neg al	1	0
add ax, 2	0	0
sub ax, 2	1	0
sub al, +!25	1	1

- The neg instruction negates the value in the destination operand. The Carry Flag is set if the result of the negation is negative. In the first operation, the neg al instruction negates the value -128. The result is 128, which is a positive number. Therefore, the Carry Flag is not set.
- The add instruction adds two operands and stores the result in the destination operand. The Carry Flag is set if the result of the addition is too large to be stored in the destination operand. The Overflow Flag is set if the result of the addition is too large or too small to be stored in the destination operand. In the second operation, the add ax, 2 instruction adds 2 to the value in the ax register. The sum of 8000h and 2 is 8002h, which is within the range of values that can be stored in a 16-bit register. Therefore, the Carry Flag and the Overflow Flag are not set.
- The sub instruction subtracts two operands and stores the result in the destination operand. The Carry Flag is set if the result of the subtraction is negative. The Overflow Flag is set if the result of the subtraction is too small to be stored in the destination operand. In the third operation, the sub ax, 2 instruction subtracts 2 from the value in the ax register. The difference of 8000h and 2 is 7FFEh, which is within the range of values that can be stored in a 16-bit register. Therefore, the Carry Flag and the Overflow Flag are not set.
- The sub instruction subtracts two operands and stores the result in the destination operand. The Carry Flag is set if the result of the subtraction is negative. The Overflow Flag is set if the result of the subtraction is too small to be stored in the destination operand. In the fourth operation, the sub al, +!25 instruction subtracts 25 from the value in the al register. The difference of -5 and 25 is -30, which is less than the minimum value that can be stored in a byte register (0). Therefore, the Overflow Flag is set. The Carry Flag is also set because the result of the subtraction is negative.

What will be the final value of ax?

```
1 mov ax, 6
2 mov ecx, 4
3
4 L1:
5     inc ax
6     loop L1
```

- The ax register contain the value of 6+4=**10**

How many times will the loop execute?

```
1 mov ecx,0
2 X2:
3     inc ax
4     loop X2
```

- This code will cause the loop to execute **10 times**, because the CX register is initialized to 10