Overflow concept

At the level of the assembly language, 'overflow' is a situation / condition which expresses the fact that the result of the LPO (Last performed operation) didn't fit the reserved space for it (S or unsigned). The flags CF and OF will be set for specific cases of overflow.

Addition

- in the unsigned interpretation, the overflow is signaled by setting CF=1 whenever the result didn't fit the reserved. space for it (on a byte [0, 255]) otherwise CF=0.

```
Ex:  mov al, 100   ; 100 ∈ [0,255] , but 100+200=300 ∉ [0,255] =>
     mov bl, 200   ; 200 ∈ [0,255]   => CF=1 .
     add al, bl    ;
```

- in signed interpretation, the overflow is signaled by setting OF=1 whenever the base 2 addition reflects an incorrect mathematic result (in the signed interpretation). Otherwise OF=0. On a byte: [-128, 127].

Ex : ~~mov al~~ There are only 2 situations that can issue overflow for addition:

```
1.   0.... +            1.... +
     0.....      and    1....
     ------             -----
     1.....             0....
```

```
Ex:  mov al, 100   ; 100 ∈ [-128, 127]
     mov bl, 100   ; ~~to~~but 100+100=200 ∉ [-128, 127] =>
     add al, bl    ; => we bring the value in the domain by
                     subtracting (256) => 200-256 = -56 => OF=1
```

Subtraction :

- in the unsigned ~~representation~~ interpretation, the overflow is signaled by setting CF=1 whenever there exists a borrow from a non-existent position or in other words, the result didn't fit the reserved space for it. Otherwise CF=0.

```
Ex:  mov al, 100  |
     mov bl, 101  |  100 - 101 = -1 ∉ [0, 255] => CF=1.
     sub al, bl   |
```

- in the signed interpretation, the overflow is signaled by setting OF=1 whenever the base 2 subtraction reflects an incorrect result (in Signed interpretation). Otherwise OF=0

Ex: ~~mov~~ There are only 2 situations that can issue overflow for subtraction:

```
 1.... -            0.... -
 0......    and     1....
 -------            -------
 0......            1......
```

```
Ex: mov al, 100   |  100 ∈ [128, 127]
    mov bl, -100  |  -100 ∈ [-128, 127].
    sub al, bl    |  100 - (-100) = 200 ∈ [-128, 127) =>
```

=> 200 - 256 = -56

so + - (-) = - which is incorrect => OF = 1.

Multiplication :

- the multiplication operation does not produce overflow, the reserved space being enough for both interpretations. The decision was taken to set CF=OF=0 whenever the result is the same size of the operands. And CF = OF = 1 for the opposite - for the cases

~~Ex~~ byte * byte = byte
word * word = word
dword * dword = dword
=> OF = CF = 0 (no multiplication overflow).

And for the cases:

byte * byte = word
word * word = dword
dword * dword = qword
=> OF = CF = 1.

```
Ex: mov al, 200    ; 200 ∈ (0,255]
    mov bl, 200    ; 200 * 200 = 40000 ∈ (0,65536) =>
    mul bl         ; => on a word
```

Division

- in the case of division, if it happens, then it will result in the program crashing (Fatal Error) = division overflow. The values of CF & OF are irrelevant. the quotient didn't fit the reserved space

```
ex: mov ax, 4096
    mov bl, 10     ; 4096 : 10 = 409, r=6
    div bl           but 409 ∉ [0,256] =>
                   => FATAL ERROR / crash .
```

There are methods to deal with the overflow concept and the assembler gives us 2 specific instructions : ADC (add with carry) and SBB (subtraction with borrow).

Ex: we need to compute the value of Dx:Ax + Bx:Bx.

```
add ax, bx
adc dx, cx
```

→ to obtain a correct result, we make sure that the transport digit is not lost.

There is no "iadd" or "isub" because even if they existed, they would work exactly the same as "ADD" and "SUB". This is because in base 2 addition and subtraction are performed the same INDEPENDENTLY of the INTERPRETATION. There exists "IMUL" and "IDIV" because this rule does not apply to multiplication and division who work differently in both interpretations.

The programmer can avoid overflow situations by using larger data types (ex: word instead of byte), checking the input range or using instructions like: (jo - jump if OF=1, jno --||-- OF=0, jc --||-- CF=1; jnc --||-- CF=0).

Ex:

```
add al, bl
jnc avoid
; instructions
avoid:
```

– if the addition of the 2 registers results in an overflow by that fitting the reserved space, the jump will be performed.