Flag theory:

A flag is an indicator on 1 bit.

For x86 the EFLAGS register has 32 bits, from which only 9 are commonly used: CF, OF, SF, ZF, OF, TF, AF, IF, DF.

The flags are divided into 2 categories:

- the ones that report the effect of the LPO (CF, OF, SF, ZF, PF, AF)
- the ones with future effect after being set by the programmer (CF, IF, TF, DF)

Carry flag is a transport flag, signaling the overflow for unsigned representation. It is set to 1 if we have a carry digit outside the domain representation and 0 otherwise.

Examples: for add:

```
mov al, 100
mov ah, 200
add al, ah
```
⇒ CF=1 because 100+200=300 ∉[0,255]
⇒ we have a carry digit

```
mov al, 100
mov ah, 10
add al, ah
```
⇒ CF=0 because 100+10=110 ∈[0,255]
⇒ the result can be stored on the interval

for sub:

mov al, 100 ⇒ CF = 1 because 100-120 = -20 ∉ [0,99]

mov ah, 120

sub al, ah

mov al, 100 ⇒ CF=0 because 100-50 = 50 ∈ [0,255]

mov ah, 50

Sub al, ah

Overflow flag - it signals the overflow for signed representation, if the result of the LPO (considered in the signed interpretation) didn't fit the reserved space, then OF = 1 and OF = 0 otherwise

Overflow situations: for addition (signed representation)

I (+) + (+) = − or II (−) + (−) = +

```
0------+        1------+
0------         1------
-------        -------
1-----         0-------
```

for substraction:

I (−) − (+) = + or II (+) − (−) = −

```
1-------−      0--------−
0-------       1--------
--------       --------
0------        1-------
```

Examples: for aodd:

mov al, 100 ⇒ OF=1 because 100+100=200 $\notin [-128, 127]$

mov ah, 100

aodd al, ah

mov al, 10 ⇒ OF=0 because 10+10=20 $\in [-128, 127]$

mov ah, 10

aodd al, ah

for sub:

mov al, 100 ⇒ OF=1 because:

mov ah, 156 $100-(156-296)=100-(-100)=$

sub al, ah $=200 \notin [-128, 127]$

mov al, 100 ⇒ OF=0 because:

mov ah, 20 $100-20=80 \in [-128, 127]$

sub al, ah

The multiplication operation does not produce overflow, the reserved space being enough for both interpretations. The decision was taken to set both CF=OF=0 if the result is the same size as the operand (b*b=b, w*w=w, d*d=d) otherwise CF=OF=1 (b*b=w, w*w=dl, d*d=qw)

Examples:

```
mov al,-1      ⇒ CF=OF=0 because:
mov ah, 2        -1*2=-2 ∈ [-128,127]
imul ah          byte*byte=byte

mov al, 100    ⇒ CF=OF=1 because
mov ah, 10       100*10=1000
mul ah           byte*byte=word
```

The worst effect in case of overflow is for division, if the result does not fit the reserved space (byte for word/byte, word for dw/word, dw for qw/dw) then it will signal a run-time error and the OS will stop the program and will issue one of the messages: 'Divide overflow', 'Division by zero', 'Zero divide'.

in the case of correct division, CF and OF are undefined
if we have a division overflow, the program crashes
and the values of CF and OF don't matter.

Example: w/b = b

mov ax, 1002 ⇒ division overflow because
mov bh, 3 1002/3 = 334 ∉ [0,255]
div bh
AL = AX / BH

Why do we need CF, OF in EFLAGS simultaneously?
Because when performing an addition or substraction
in base 2, there are actually 2 operations performed
simultaneously in base 10, one in signed interp.
and one in unsigned interp.
As a result two different flags are needed to deal
with the 2 interpretations in base 10: CF for unsigned
and OF for signed
This happens because the operation of addition or
substraction in base 2 is performed identically, this
is also the reason there is no IADD or ISUB,
they would not work differently from ADD and SUB

Why do we need iMUL and iDiv?
Because signed and unsigned multiplication and division work differently in each interpretation.
In conclusion we need to specify before the operation how we want the operands to be interpreted (MUL & DIV for unsigned and iMUL & iDiv for signed)

zero flag - it signals if the result of the LPO was 0
It is not set for multiplication or division. It is set to 1 if the result of addition/substraction was zero and set to 0 otherwise

Examples:

```
mov al,10      => ZF=1 because
mov ah,-10          10+(-10)=0
add al,ah

mov al,10      => ZF=1 because
mov ah,10           10-10=0
Sub al,ah
```

Sign flag - signals the sign bit of the LPO

Examples:

mov al, 10 ⇒ SF=0 because
mov ah, -1 10+(-1) = 9 > 0
aadd al, ah

mov al, 20 ⇒ SF=1 because
mov ah, 30 20-30 = -10 ∈ [-128,127]
sub al, ah < 0

instructions used for setting the flags:

1. Carry flag:

CLC - clear carry flag ⇒ sets CF=0
STC - set carry flag ⇒ sets CF=1
CMC - complement carry flag ⇒ CF=0 if CF=1
CF=1 if CF=0
CF = ~CF

2. Direction flag:

CLD - clear direction flag ⇒ sets DF=0
STD - set direction flag ⇒ sets DF=1

3 interrupt flag - can be set only on 16 bits, on 32 bits this was removed

CLI ⇒ IF = 0

STI ⇒ IF = 1

interrupt flag: - if set to 1, interrupts are allowed, if set to 0 interrupts will not be handled

Trap flag - debugging flag. if it's set to 1 then the machine stops after every instruction

Auxiliary flag - stores the transport digit from bit 3 to bit 4 of the LPO

Direction flag: for operating string instructions. if set to 0, the parsing of the string will be performed in ascending order and if set to 1 in descending order

Parity flag - set to the least significant bit of the LPO, set to 1 if the result was odd, and set to 0 if even

Flag transfer instructions:

LAHF (Load register AH from flags):

- copies SF, ZF, AF, PF, CF from FLAGS register in the bits 7, 6, 4, 2, 0 of register AH
- bits 5, 3, 1 are undefined
- other flags are not affected

SAHF (Store register AH into flags)

- transfers the bits 7, 6, 4, 2, 0 of register AH in SF, ZF, AF, PF, CF, replacing the prev. values

PUSHF:

- transfers all the flags on top of the stack (the contents of EFLAGS register is transfered onto the stack)

POPF:

- extracts the word from the top of the stack and transfers its contents into the EFLAGS register