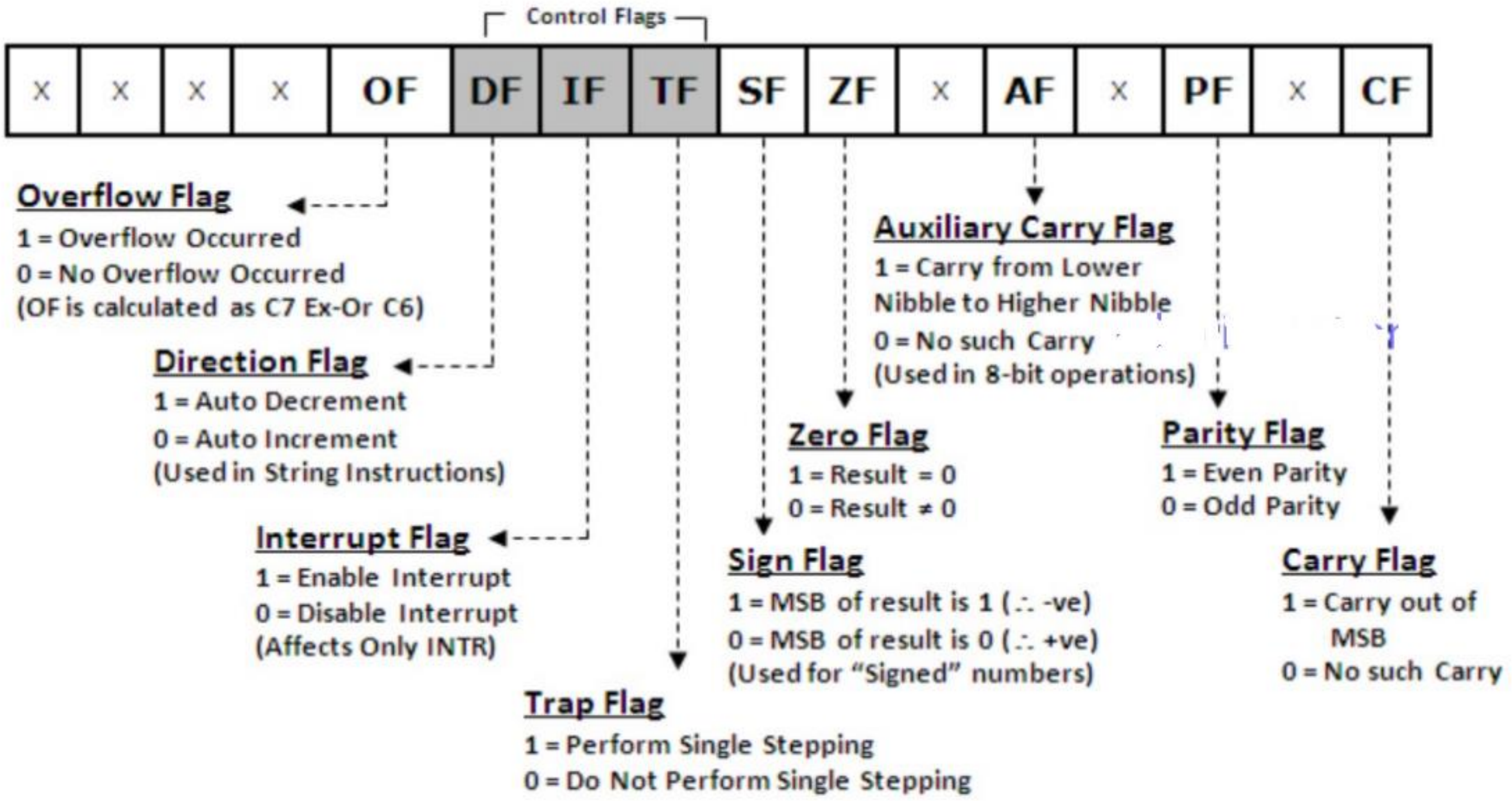
Microprocessor and Assembly Language Lab

Lab Material 5 for CSE 312 (M&AL Lab)

Dr. Shah Murtaza Rashid Al Masud
Associate Prof.
Dept. of CSE, UAP

The FLAGS Register



Overflow Flag
1 = Overflow Occurred
0 = No Overflow Occurred
(OF is calculated as C7 Ex-Or C6)

Direction Flag
1 = Auto Decrement
0 = Auto Increment
(Used in String Instructions)

Interrupt Flag
1 = Enable Interrupt
0 = Disable Interrupt
(Affects Only INTR)

Trap Flag
1 = Perform Single Stepping
0 = Do Not Perform Single Stepping

Auxiliary Carry Flag
1 = Carry from Lower Nibble to Higher Nibble
0 = No such Carry
(Used in 8-bit operations)

Zero Flag
1 = Result = 0
0 = Result ≠ 0

Sign Flag
1 = MSB of result is 1 (∴ -ve)
0 = MSB of result is 0 (∴ +ve)
(Used for "Signed" numbers)

Parity Flag
1 = Even Parity
0 = Odd Parity

Carry Flag
1 = Carry out of MSB
0 = No such Carry

Status Reg./FLAGS reg.

To indicate the status of the μP.
1 flag == 1 bit
Y/N
➔9 active bits – out of ?? Bits?
Q: Types of flags? Some names?

a. **Control flags – interrupt flag, trap flag, direction flag**
b. **Status flags – Zero flag, Carry flag, sign flag, parity, auxiliary flag, overflow flag**

- **The processor uses the status flags to reflect the result of an operation.**
- **Bits 0, 2, 4 ,6, 7, 11**

Status flags

Processor uses the status flags to reflect the result of an operation.
E.g.,

```
SUB     AX, AX
```

→Zero flag becomes 1
→It indicates that a zero result was produced

Carry flag – CF
1(MSB)000(LSB)
0001
1010

CF = 1

➔ if there is a carry out from the MSB on addition;

➔or, there is a borrow into the MSB on subtraction;

Else CF = 0

CF changes with SHIFT (SHL – shift left; SAL – shift arithmetic left; SHR – shift right; SAR – shift arithmetic right) &

ROTATE (ROL – rotate left; ROR – rotate right) instructions

Carry flag - CF

```
Carry Flag
----------

The rules for turning on the carry flag in binary/integer math are two:

1. The carry flag is set if the addition of two numbers causes a carry
   out of the most significant (leftmost) bits added.

   1111 + 0001 = 0000 (carry flag is turned on)

2. The carry (borrow) flag is also set if the subtraction of two numbers
   requires a borrow into the most significant (leftmost) bits subtracted.

   0000 - 0001 = 1111 (carry flag is turned on)

Otherwise, the carry flag is turned off (zero).
 * 0111 + 0001 = 1000 (carry flag is turned off [zero])
 * 1000 - 0001 = 0111 (carry flag is turned off [zero])
```

Parity Flag (PF)

- PF = 1 if the low byte of a result has an even number of one bits (even parity).
- PF = 0 if the low byte has odd parity.
- If the result of a word addition is FFFEh, then the low byte contains 7 one bits, so PF = 0.
- FFFEh

1111 1111 1111 1110 b

Parity flag - PF

PF = 1

→If the low byte of a result has an even number of one bits [even parity]

PF = 0

→If the low byte has odd parity

<u>Q.:</u> If after ADD – the result is FFF<u>E</u>h – then the low byte [F<u>E</u>] contains 7 one bits [F<u>E</u> = 1111 1110], PF = ?

Ans.: PF = 1

➔ Nooooooo… ➔ PF = 0

Auxiliary Carry Flag (AF)

- AF = 1 if there is a carry out from bit 3 on addition, or a borrow into bit 3 on subtraction.

```
1110 (hn) 0010 (ln)  (addition)
1001 (hn) 1010 (ln)
               (c)1111 1100
CF=1
```

Auxiliary Carry Flag - AF

AF = 1
→If there is a carry out from bit 3 on addition,
→Or a borrow into bit 3 on subtraction

AF is used in BCD operations [see ch. 18]

BCD – Binary-Coded Decimal [BCD uses 4 bits to code each decimal digit, from 0000 to 1001.
Combination of 1010 to 1111 are illegal in BCD.

Zero Flag (ZF)

- ZF = 1 for a zero result.
- ZF = 0 for a nonzero result.

- Sub 7, 7
- Result=0

- Sub 7, 5
- Result=2

Sign Flag (SF)

- SF = 1 if the **MSB** of a **result** is 1; it means the result is negative if you are giving a signed interpretation.
- SF = 0 if the MSB is 0.

Overflow Flag (OF)

```
Overflow Flag
-------------

The rules for turning on the overflow flag in binary/integer math are two:

1. If the sum of two numbers with the sign bits off yields a result number
   with the sign bit on, the "overflow" flag is turned on.

   0100 + 0100 = 1000 (overflow flag is turned on)

2. If the sum of two numbers with the sign bits on yields a result number
   with the sign bit off, the "overflow" flag is turned on.

   1000 + 1000 = 0000 (overflow flag is turned on)

Otherwise, the overflow flag is turned off.
 * 0100 + 0001 = 0101 (overflow flag is turned off)
 * 0110 + 1001 = 1111 (overflow flag is turned off)
 * 1000 + 0001 = 1001 (overflow flag is turned off)
 * 1100 + 1100 = 1000 (overflow flag is turned off)

Note that you only need to look at the sign bits (leftmost) of the three
numbers to decide if the overflow flag is turned on or off.
```

How the Processor Determines that Signed Overflow Occurred

- OF = 1
  - There is a carry in to the MSB but no carry out.
  - There is a carry out but no carry in.
- Addition
  - The sum has a different sign.
- Subtraction
  - The result has a different sign than expected.
  - A – (–B) = A + B
  - –A – B = –A + –B
- Addition of Numbers with Different Signs
  - Overflow is impossible.
  - A + (– B) = A – B

How the Processor Determines that Signed Overflow Occurred

```
Calculating Overflow Flag: Method 1
-----------------------------------

Overflow can only happen when adding two numbers of the same sign and
getting a different sign.  So, to detect overflow we don't care about
any bits except the sign bits.  Ignore the other bits.

With two operands and one result, we have three sign bits (each 1 or
0) to consider, so we have exactly 2**3=8 possible combinations of the
three bits.  Only two of those 8 possible cases are considered overflow.
Below are just the sign bits of the two addition operands and result:

       ADDITION SIGN BITS
    num1sign num2sign sumsign
   ---------------------------
        0 0 0
 *OVER* 0 0 1 (adding two positives should be positive)
        0 1 0
        0 1 1
        1 0 0
        1 0 1
 *OVER* 1 1 0 (adding two negatives should be negative)
        1 1 1
```

How the Processor Determines that Signed Overflow Occurred

```
We can repeat the same table for subtraction.  Note that subtracting
a positive number is the same as adding a negative, so the conditions that
trigger the overflow flag are:

      SUBTRACTION SIGN BITS
    num1sign num2sign sumsign
   ---------------------------
         0 0 0
         0 0 1
         0 1 0
 *OVER* 0 1 1 (subtracting a negative is the same as adding a positive)
 *OVER* 1 0 0 (subtracting a positive is the same as adding a negative)
         1 0 1
         1 1 0
         1 1 1
```

How the Processor Determines that Signed Overflow Occurred

```
Calculating Overflow Flag: Method 2
-----------------------------------

When adding two binary values, consider the binary carry coming into
the leftmost place (into the sign bit) and the binary carry going out
of that leftmost place.  (Carry going out of the leftmost [sign] bit
becomes the CARRY flag in the ALU.)

Overflow in two's complement may occur, not when a bit is carried out
of the left column, but when one is carried into it and no matching
carry out occurs. That is, overflow happens when there is a carry into
the sign bit but no carry out of the sign bit.

The OVERFLOW flag is the XOR of the carry coming into the sign bit (if
any) with the carry going out of the sign bit (if any).  Overflow happens
if the carry in does not equal the carry out.
```

How the Processor Determines that Signed Overflow Occurred

```
Examples (2-bit signed 2's complement binary numbers):

    11
   +01
   ===
    00

   - carry in is 1
   - carry out is 1
   - 1 XOR 1 = NO OVERFLOW


    01
   +01
   ===
    10

   - carry in is 1
   - carry out is 0
   - 1 XOR 0 = OVERFLOW!
```

How the Processor Determines that Signed Overflow Occurred

```
 11
+10
===
 01

- carry in is 0
- carry out is 1
- 0 XOR 1 = OVERFLOW!


 10
+01
===
 11

- carry in is 0
- carry out is 0
- 0 XOR 0 = NO OVERFLOW
```

How the Processor Determines that Signed Overflow Occurred

```
Note that this XOR method only works with the *binary* carry that goes
into the sign *bit*.  If you are working with hexadecimal numbers, or
decimal numbers, or octal numbers, you also have carry; but, the carry
doesn't go into the sign *bit* and you can't XOR that non-binary carry
with the outgoing carry.

Hexadecimal addition example (showing that XOR doesn't work for hex carry):

    8Ah
   +8Ah
   ====
    14h

   The hexadecimal carry of 1 resulting from A+A does not affect the
   sign bit.  If you do the math in binary, you'll see that there is
   *no* carry *into* the sign bit; but, there is carry out of the sign
   bit.  Therefore, the above example sets OVERFLOW on.  (The example
   adds two negative numbers and gets a positive number.)
```

Examples

Given AL = 80h, BL = 81h. Determine the status of all flags the following cases:

a. ADD AL, BL

b. SUB AL, BL

c. SUB BL, AL

d. NEG AX

Given AL = 7Fh, BL = 01h. Determine the status of all flags in the following cases:

a. ADD AL, BL

b. SUB AL, BL

c. SUB BL, AL

Given AX = 0FFFFh, BX = 0001h. Determine the status of all flags in the following cases:

a. NEG AX

b. ADD AX, BX

c. SUB BX, AX

d. SUB AX, BX

e. INC AX

How Instructions Affect the Flags

Instruction	Affects Flags
MOV/XCHG	none
ADD/SUB	all
INC/DEC	all except CF
NEG	all (CF = 1 unless result is 0, OF = 1 if word operand is 8000h, or byte operand is 80h)

ADD AX, BX where AX contains FFFFh and BX contains FFFFh.

$$
\begin{array}{ll}
\begin{array}{rl}
 & \text{FFFFh} \\
+ & \text{FFFFh} \\
\hline
\cancel{1} & \text{FFFEh}
\end{array}
\qquad\qquad
\begin{array}{rl}
 & 1111\ 1111\ 1111\ 1111 \\
+ & 1111\ 1111\ 1111\ 1111 \\
\hline
\cancel{1} & 1111\ 1111\ 1111\ 1110
\end{array}
\qquad
\begin{array}{l}
\\
\\
\text{AX = FFFEh}
\end{array}
\end{array}
$$

SF = 1 because the msb is 1.

PF = 0 because there are 7 (odd number) of 1 bits in the low byte of the result.

ZF = 0 because the result is nonzero.

CF = 1 because there is a carry out of the msb on addition.

OF = 0 because the sign of the stored result is the same as that of the numbers being added (as a binary addition, there is a carry in to the msb and also a carry out).

ADD AL, BL where AL contains 80h and BL contains 80h.

```
    80h             1000 0000
+   80h           + 1000 0000
~~1~~   00h           ~~1~~  0000 0000                AL = 00h
```

SF = 0 because the msb is 0.

PF = 1 because all the bits in the result are 0.

ZF = 1 because the result is 0.

CF = 1 because there is a carry out of the msb on addition.

OF = 1 because the numbers being added are both negative, but the result is 0 (as a binary addition, there is no carry in to the msb but there is a carry out).

SUB AX, BX where AX contains 8000h and BX contains 0001h.

```
    8000h              1000 0000 0000 0000
–   0001h           –  0000 0000 0000 0001
    7FFFh              0111 1111 1111 1111 AX = 7FFFh
```

SF = 0	because the msb is 0.
PF = 1	because there are 8 (even number) one bits in the low byte of the result.
ZF = 0	because the result is nonzero.
CF = 0	because a smaller unsigned number is being subtracted from a larger one.
OF = 1	because in a signed sense we are subtracting a positive number from a negative one, which is like adding two negatives but the result is positive (the wrong sign).

INC AL where AL contains FFh.

FFh	1111 1111	
+ 1h	+ 0000 0001	
~~1~~ 00h	~~1~~ 0000 0000	AL = 00h

SF = 0, PF = 1, ZF = 1.

CF is unaffected by INC.

If CF = 0 before the execution of the instruction, CF will still be 0 afterward.

OF = 0because numbers of unlike sign are being added (there is a carry into the msb and also a carry out).

MOV AX, -5

AX = FFFBh

None of the flags are affected by MOV.

NEG AX where AX contains 8000h.

8000h	= 1000 0000 0000 0000	
one’s complement	= 0111 1111 1111 1111	
	+1	
	= 1000 0000 0000 0000	= 8000h

SF = 1, PF = 1, ZF = 0.

CF = 1 because for NEG CF is always 1 unless the result is 0.

OF = 1 because the result is 8000h; when a number is negated, we would expect a sign change, but because 8000h is its own two’s complement, there is no sign change.

Q.:

ADD AX, BX

AX = FFFFh

BX = FFFFh

So, after ADD, AX = 1 FFFEh

So inside …

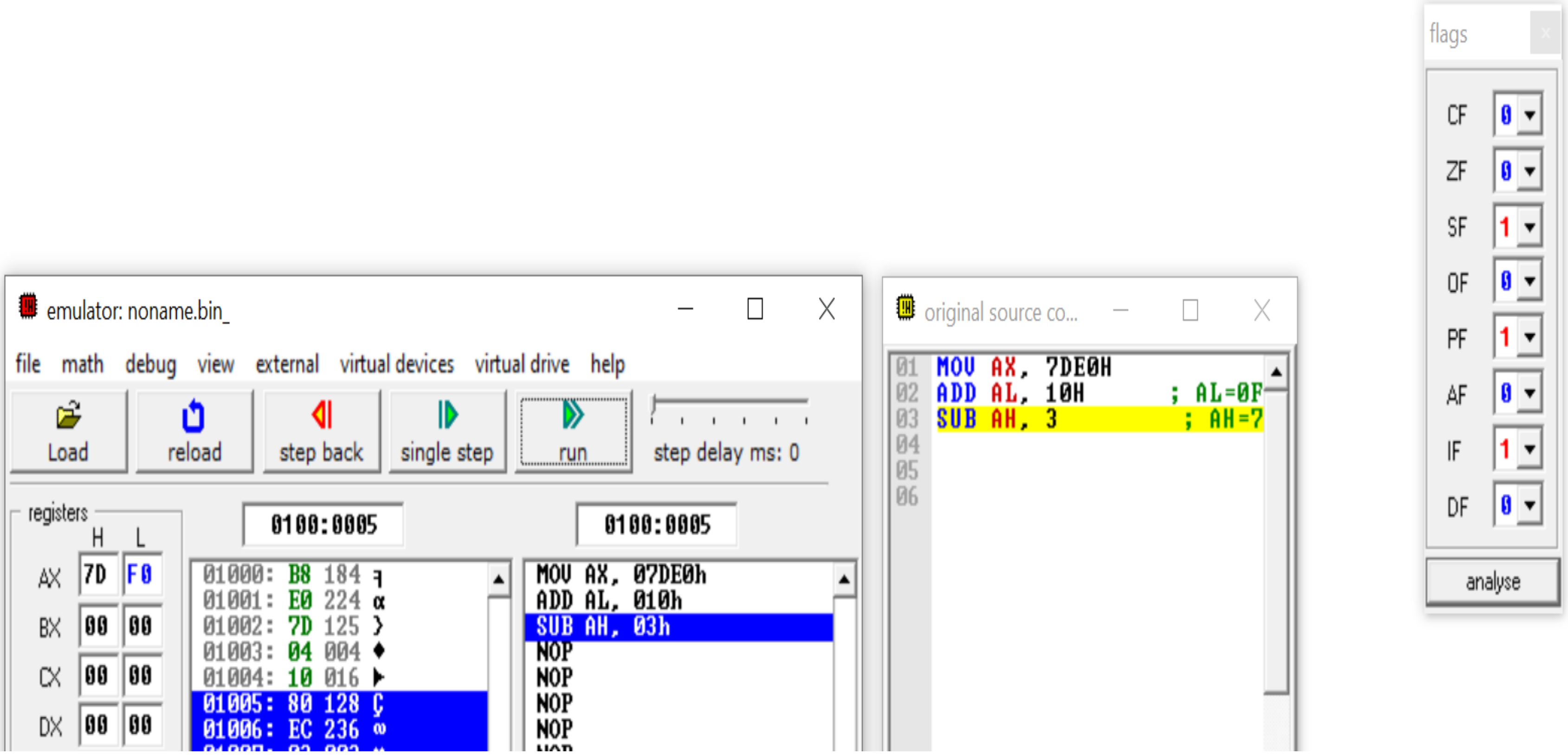
Q.:

<u>Flags affected by Arithmetic Instructions:</u>

```
            MOV AX, 7DE0H
   ADD AL, 10H        ; AL=0F0H, CF=0, SF=1,   ZF=0,   OF=0
   SUB AH, 3                      ; AH=7AH,  CF=1, SF=0,   ZF=0,   OF=0
4
Calculation: AL=E0H=11100000B
                   +10H= 00010000B
-----------------------------------------
                 11110000B=0F0H→AL
            CF=0, SF=1, ZF=0, OF=0

            03H=00000011B
                  11111100B
                         +1B
------------------------------------------------------------------
--------------------------------------------------
           -03H=11111101B

           AH=7DH=01111101B
                 -03H=11111101B
--------------------------------------------
                        01111010B=7AH→AH
           CF=1, SF=0, ZF=0, OF=0
```

Q.:

```
MOV AX, 7DE0H
    ADD AL, 10H      ; AL=0F0H, CF=0, SF=1,   ZF=0,   OF=0
    SUB AH, 3        ; AH=7AH,  CF=1, SF=0,   ZF=0,   OF=0       4
```



```
     H   L
AX   7D  F0
BX   00  00
CX   00  00
DX   00  00
```

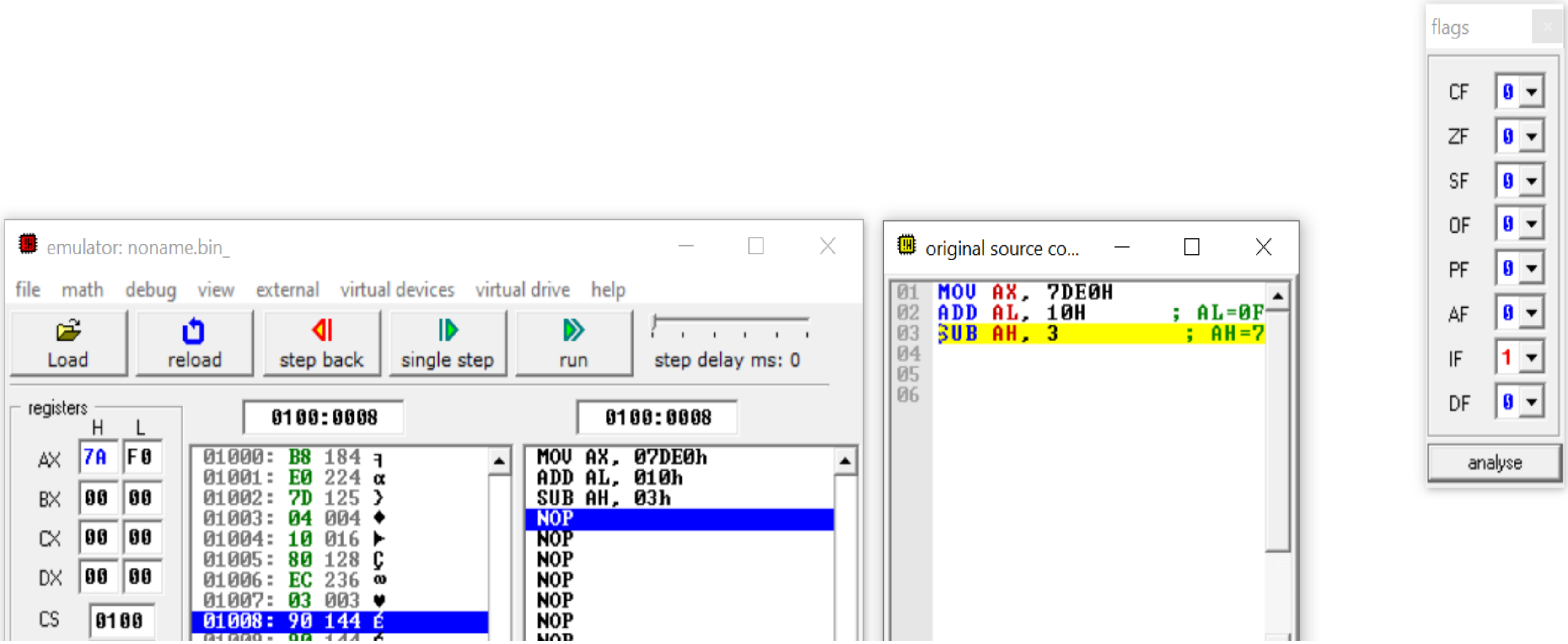
```
0100:0005                    0100:0005

01000: B8 184 ╕              MOV AX, 07DE0h
01001: E0 224 α              ADD AL, 010h
01002: 7D 125 }              SUB AH, 03h
01003: 04 004 ♦              NOP
01004: 10 016 ►              NOP
01005: 80 128 Ç              NOP
01006: EC 236 ∞              NOP
```

```
MOV AX, 7DE0H
ADD AL, 10H           ; AL=0F
SUB AH, 3             ; AH=7
```

```
CF 0
ZF 0
SF 1
OF 0
PF 1
AF 0
IF 1
DF 0
```

Q.:

```
MOV AX, 7DE0H
    ADD AL, 10H       ; AL=0F0H, CF=0, SF=1,   ZF=0,   OF=0
    SUB AH, 3         ; AH=7AH,  CF=1, SF=0,   ZF=0,   OF=0       4
```



```
0100:0008                    0100:0008

    H  L
AX  7A F0     01000: B8 184 ╕    MOV AX, 07DE0h
BX  00 00     01001: E0 224 α    ADD AL, 010h
CX  00 00     01002: 7D 125 }    SUB AH, 03h
DX  00 00     01003: 04 004 ♦    NOP
CS  0100      01004: 10 016 ►    NOP
              01005: 80 128 Ç    NOP
              01006: EC 236 ∞    NOP
              01007: 03 003 ♥    NOP
              01008: 90 144 É    NOP
```

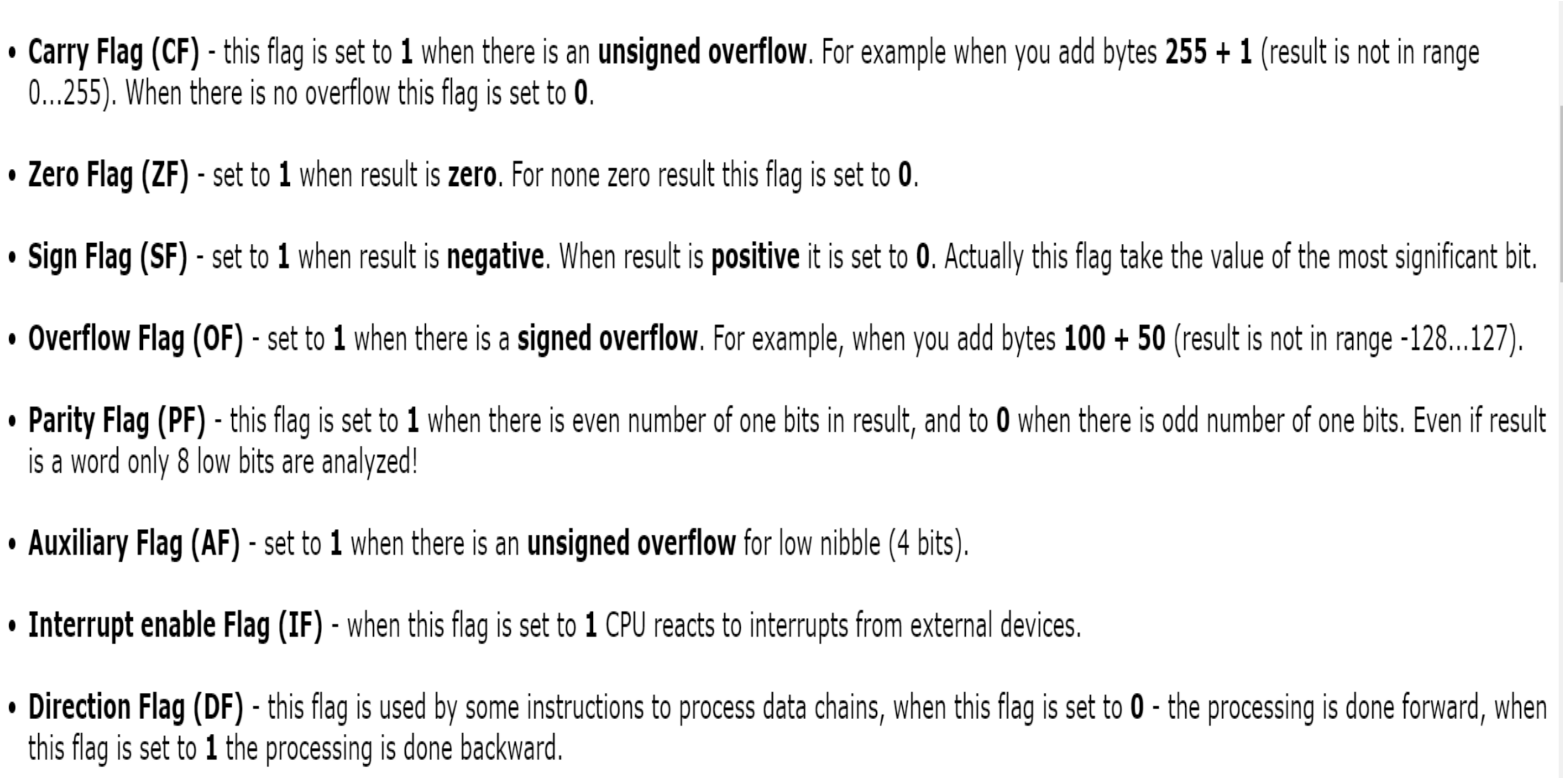
original source co...

```
MOV AX, 7DE0H
ADD AL, 10H          ; AL=0F
SUB AH, 3            ; AH=7
```

flags

CF	0
ZF	0
SF	0
OF	0
PF	0
AF	0
IF	1
DF	0

Q.:



- **Carry Flag (CF)** - this flag is set to **1** when there is an **unsigned overflow**. For example when you add bytes **255 + 1** (result is not in range 0...255). When there is no overflow this flag is set to **0**.

- **Zero Flag (ZF)** - set to **1** when result is **zero**. For none zero result this flag is set to **0**.

- **Sign Flag (SF)** - set to **1** when result is **negative**. When result is **positive** it is set to **0**. Actually this flag take the value of the most significant bit.

- **Overflow Flag (OF)** - set to **1** when there is a **signed overflow**. For example, when you add bytes **100 + 50** (result is not in range -128...127).

- **Parity Flag (PF)** - this flag is set to **1** when there is even number of one bits in result, and to **0** when there is odd number of one bits. Even if result is a word only 8 low bits are analyzed!

- **Auxiliary Flag (AF)** - set to **1** when there is an **unsigned overflow** for low nibble (4 bits).

- **Interrupt enable Flag (IF)** - when this flag is set to **1** CPU reacts to interrupts from external devices.

- **Direction Flag (DF)** - this flag is used by some instructions to process data chains, when this flag is set to **0** - the processing is done forward, when this flag is set to **1** the processing is done backward.

DEBUG

- **R** display registers
- **T** trace an instruction
- **G** execute a program
- **Q** quit

DEBUG Flag Symbols

Status Flag	**Set (1) Symbol**	**Clear (0) Symbol**
CF	CY (carry)	NC (no carry)
PF	PE (even parity)	PO (odd parity)
AF	AC (auxiliary carry)	NA (no auxiliary carry)
ZF	ZR (zero)	NZ (nonzero)
SF	NG (negative)	PL (plus)
OF	OV (overflow)	NV (no overflow)
Control Flag		
DF	DN (down)	UP (up)
IF	EI (enable interrupts)	DI (disable interrupt)

Control Flag

Control Flags – The control flags enable or disable certain operations of the microprocessor. There are 3 control flags in 8086 microprocessor and these are:

Directional Flag (D) – This flag is specifically used in string instructions.
If directional flag is set (1), then access the string data from higher memory location towards lower memory location.
If directional flag is reset (0), then access the string data from lower memory location towards higher memory location.
Interrupt Flag (I) – This flag is for interrupts.
If interrupt flag is set (1), the microprocessor will recognize interrupt requests from the peripherals.
If interrupt flag is reset (0), the microprocessor will not recognize any interrupt requests and will ignore them.
Trap Flag (T) – This flag is used for on-chip debugging. Setting trap flag puts the microprocessor into single step mode for debugging. In single stepping, the microprocessor executes a instruction and enters into single step ISR.
If trap flag is set (1), the CPU automatically generates an internal interrupt after each instruction, allowing a program to be inspected as it executes instruction by instruction.
If trap flag is reset (0), no function is performed