Esercizio 1 (già fatto in classe).

Scrivere un programma in Assembly che esegua la somma tra due variabili di tipo byte e che memorizzi il risultato nel registro AL e poi nella variabile **ris.**
Modificare i valori delle due variabili come indicato sotto e vedere quali flags vengono modificati.

a) 127 e 1 [MODIFICATI: o=1 perché: , s=1, a=1]
b) 128 e 1 [MODIFICATI: o=0 (perché?), s=1, p=1]
c) 128 e −1 [MODIFICATI: o=1(perché?), c=1]

Esercizio 2. Scrivere un programma in assembly che esegua la **differenza** fra due numeri (su 16 bit) che si trovano nelle locazioni di memoria (variabili) SOT1(=0A1h) e SOT2 (=42h) e ponga il risultato nella locazione RIS.

Modificare i valori delle due variabili e vedere quali flags vengono modificati.

Esercizio3. Sommare due numeri, uno contenuto nella variabile num1=150d e l'altro un valore immediato pari a 200d. Poi, i**ncrementare e decrementare di 1 i**l risultato.

Esercizio 4. Scrivere un programma che calcoli l'espressione w=x-2y +3z (usare solo somme e differenze: 2y= y+y …). Si supponga che x,y,z e w siano numeri su 16 bit .

Ad esempio: x=100 (64h), y=45 (2Dh), z=15 (Fh). Risultato w=55 (37h)

L' ISTRUZIONE NEG

Il linguaggio Assembly mette a disposizione un' istruzione particolare di nome **NEG.** La si usa per calcolare il negativo di un numero, il quale viene espresso in **COMPLEMENTO A 2.**

Sintassi: NEG <op1>

<op1> può essere:

- Un registro;
- Un indirizzo di memoria.

OPERAZIONE DI MOLTIPLICAZIONE: Istruzione MUL e IMUL

L'istruzione **MUL** esegue la moltiplicazione tra due numeri senza segno.

Per eseguire moltiplicazioni tra numeri con segno si utilizza l'istruzione **IMUL**, che prevede le stesse caratteristiche dell'istruzione **MUL.**

Sintassi:

MUL <operando> **IMUL <operando>**

Con la sintassi dell'istruzione viene specificato solo un fattore della moltiplicazione.
Il secondo fattore è memorizzato in AL o in AX a seconda della dimensione dell'operando.

Più precisamente,

- Se l'operando è di tipo byte, l'altro operando deve essere posto in **AL** e il risultato viene posto in **AX**.

Quindi **MUL <operando> // IMUL <operando>** corrisponde all'istruzione:

AX = AL * <operando>

- Se l'operando è di tipo word, l'altro operando deve essere posto in **AX** e il risultato viene posto nei due registri **DX e AX**.

Quindi **MUL <operando> // IMUL <operando>** corrisponde all'istruzione:

DX e AX.= AX * <operando>

Da notare è che l'operando non può essere un valore immediato.

L'istruzione **MUL e IMUL** possono modificare il

- CF (**Carry Flag)** per indicare se la parte alta del risultato è usata o no: il flag è a 1 se viene usata anche la parte alta;
- il flag di overflow **OF**.

Esercizio 5.

Scrivere il seguente programma in assembly che esegue moltiplicazioni fra due numeri (su 8 bit) contenuti in due variabili. Con il debug (TD) cercare di capire la differenza tra MUL e IMUL, trovare il risultato delle operazioni.

```
; programma assembly che MOLTIPLICA due numeri della memoria centrale contenuti in due variabili

.model small
.stack
.data
        A Db 20
        M1 Db 5
        M2 DB -5
        F dw ?
.code
 inizio:
    ; prodotto tra due numeri byte senza segno
    mov ax,@data
    mov ds,ax

        mov al, A
        mov bl, M1
        mul bl
        mov F,ax

        ; prodotto tra due numeri con segno --> mul M2 è CONSIDERATO POSITIVO
        mov al, A
        mov bl, M2
        mul bl
        mov F,ax

        ; prodotto tra due numeri con segno --> imul M2 è CONSIDERATO NEGATIVO
        mov al, A
        mov bl, M2
        imul bl
       mov F,ax

        mov ah,4ch
    int 21h
 end inizio
```

Esercizio 6. Rifare l'es.3 usando l'operatore mul.

Esercizio 7. (moltiplicazione a 16 bit) Scrivere un programma in Assembly che esegua la moltiplicazione tra 65535 e 2. Con il debug verificare il risultato 1FFFE (DX=0001 e AX = FFFE)

```
.model small
.stack
.data

.code
 inizio:

   mov bx,65535
   mov ax,2            ; FFFF+FFFF=1FFFE
   mul bx              ; in debug verificare il risultato in DX:AX = 0001FFFE

   mov ah,4ch
   int 21h
 end inizio
```

OPERAZIONE DI DIVISIONE:
Istruzione DIV e IDIV

L'istruzione **DIV** esegue la divisione intera tra due numeri senza segno. La sua sintassi è:

DIV <operando>

operando è il divisore mentre il dividendo è posto in AX o in DX e AX a seconda della dimensione di operando

- se l'operando è un byte :

DIV <operando> corrisponde all'istruzione:

AL = AX/<operando> e **AH =resto**

- se l'operando è un word:

DIV <operando> corrisponde all'istruzione:

AX = (DX e AX)/<operando> e **DX=Resto**

Da notare è che il divisore non può essere un valore immediato. L'istruzione **DIV** può causare una condizione di overflow se il risultato è troppo grande per essere contenuto nel registro relativo. Si deve porre particolare attenzione alla divisione per 0 e per 1.

Per eseguire divisioni tra numeri con segno si utilizza l'istruzione **IDIV**, che prevede le stesse caratteristiche dell'istruzione **DIV**.

Esercizio 8. La media tra due numeri.

Per semplicità decidiamo di operare su variabili numeriche intere. Assumiamo inoltre che i valori dei due numeri siano definiti e non acquisiti da tastiera.

L'algoritmo, molto semplice, consiste nel salvare i due numeri all'interno di un registro, sommarli e dividerli per 2, ignorando il resto della divisione. Salvare infine il risultato in una variabile.

L'esempio di seguito riportato fa riferimento ai numeri 10 e 20.

```
MOV AH, 10      ;Iniziamo inserendo nel registro AH il valore 10
MOV AL, 20      ;ed in AL il valore 20

ADD AL, AH      ;Sommiamo il contenuto di al con il contenuto di ah
MOV AH, 0       ;Ed azzeriamo AH, ora in AL sara' presente il valore 30

MOV BX, 2       ;Copiamo in BX il 2
DIV BL          ;E dividiamo il contenuto di AX per il contenuto di BL

MOV Ris, AL     ;Ed infine copiamo il risultato della divisione nella variabile dichiarata sotto.
                ;L'eventuale resto della divisione viene conservato nel registro AH in quanto si
                ;opera con dati di tipo Byte.
```

Esercizio 9. Scrivere il seguente programma in Assembly che esegue la moltiplicazione e la divisione tra due variabili num1= 50 e num2= 16. Memorizzare i risultati nelle variabili prodotto, quoziente e resto.

Osservazioni:

- ✓ Quoto della divisione è posto in AL e il resto della divisione è posto in AH
- ✓ L'istruzione mul **num2** viene memorizzato nel segmento CD con mul **byte ptr [0005]**

OPERATORE **PTR**
sintassi: tipo PTR nome

forza l'assemblatore a modificare il tipo di dato avente come identificatore il nome della variabile. In caso di ambiguità (mul può essere un'operazione tra byte o word) bisogna specificare se il dato da manipolare è di tipo WORD (16 bit) o di tipo BYTE (8 bit), usando l'istruzione "PTR". Nel nostro caso **ptr** dice all'assemblatore che il valore della variabile num2 memorizzata nella locazione 0005 è un byte e quindi mul è un'operazione tra byte e non tra word.

```
; programma assembler che effettua le seguenti operazioni:
; moltiplicazione, divisione e resto

.model small
.stack
.data
    num1 db 50
    num2 db 16
    prodotto dw ?
    quoziente db ?
    resto db ?
.code
 inizio:
    mov ax,@data
    mov ds,ax
    mov al,num1
    mul num2
    mov prodotto,ax

   mov al,num1
   mov ah,0
   div num2
   mov quoziente,al
   mov resto,ah

  mov ah,4ch
  int 21h

 end inizio
```

Esercizio 10. (divisione a 16 bit) Scrivere un programma in Assembly che esegua la divisione tra 0003 FFFE e 2 = ….. Con il debug verificare il risultato: resto DX=00h e quoto in AX = FFFFh

```
.model small
.stack

.data

.code
 inizio:

   mov dx,0003h     ;dividendo in DX:AX
   mov ax,0FFFEh  ; se la cifra dell'esadecimale è una lettera farla precedere con 0
   mov bx,2         ;divisore
   div bx           ; in debug verificare il risultato: resto DX=00h e quoto in AX = FFFFh

   mov ah,4ch
   int 21h
 end inizio
```