Esercizi Assembly 1

M. Rebaudengo – R. Ferrero

Politecnico di Torino
Dipartimento di Automatica e Informatica

Esercizio 1

- Siano date tre variabili di tipo byte corrispondenti a tre caratteri alfabetici minuscoli (ASCII)
  - Var1 = ‘a’
  - Var2 = ‘s’
  - Var3 = ‘m’
- Si scriva un programma che stampi a video i tre caratteri convertiti in maiuscolo.

Codice

```
            .model small
            .stack
            .data
Var1        db 'a'
Var2        db 's'
Var3        db 'm'
            .code
            .startup
            mov ah, 2
            mov dl, Var1
            add dl, 'A'-'a'
            INT 21h
            mov dl, Var2
            add dl, 'A'-'a'
            INT 21h
            mov dl, Var3
            add dl, 'A'-'a'
            INT 21h
            .exit
            end
```

Esercizio 2

- Siano date le seguenti variabili di tipo byte già inizializzate in memoria:
  - `n1 db 10`
  - `n2 db 10h`
  - `n3 db 10b`
- Si calcoli la seguente espressione, il cui risultato dovrà essere salvato nella variabile *byte* `res`, e si verifichi il risultato:
  - n1 + n2 – n3.

Codice

```
        .model small
        .stack
        .data
n1      db 10
n2      db 10h
n3      db 10b
res     db ?
        .code
        .startup
        mov al, n1
        add al, n2
        sub al, n3
        mov res, al
        .exit
        end
```

Esercizio 3

- Siano date le seguenti variabili di tipo *word* (*con segno*) già inizializzate in memoria:
  - OPA = – 459
  - OPB = 470
  - OPC = 32756
  - OPD = 1
- Si scriva un programma per l'esecuzione dell'espressione OPA+OPB+OPC+OPD utilizzando il registro AX
- Si osservino in modalità passo-passo il risultato parziale e il comportamento delle flag (*sign*, *overflow* e *carry*), spiegando quanto visto.

Codice

```
            .model small
            .stack
            .data
opa         dw -459
opb         dw 470
opc         dw 32756
opd         dw 1
            .code
            .startup
            mov ax, opa
            add ax, opb   ← SF = 0; CF = 1; OF=0; risultato corretto

            add ax, opc   ← SF = 0; CF = 0; OF=0; risultato corretto

            add ax, opd   ← SF = 1; CF = 0; OF = 1; risultato errato
            .exit
            end
```

Esercizio 4

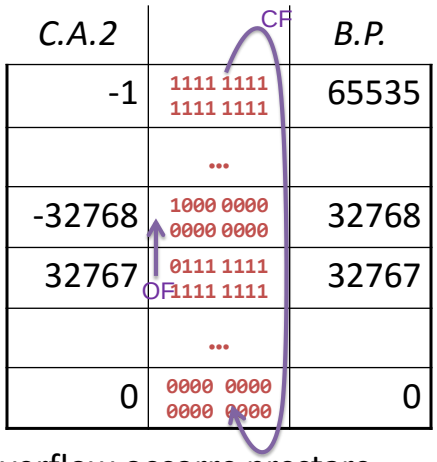
- Siano date le seguenti variabili di tipo *word* (*unsigned*) già inizializzate in memoria:
  - OPA = 32767
  - OPB = 1
- Si scriva un programma per l'esecuzione dell'espressione OPA+OPB+OPA+OPB utilizzando il registro AX
- Si osservino in modalità passo-passo il risultato parziale e il comportamento delle flag (*sign*, *overflow* e *carry*), spiegando quanto visto.

Codice

```
            .model small
            .stack
            .data
opa         dw 32767
opb         dw 1
            .code
            .startup
            mov ax, opa
            add ax, opb   ← CF = 0; OF = 1; SF = 1; risultato corretto
            add ax, opa   ← CF = 0; OF = 0; SF = 1; risultato corretto
            add ax, opb   ← CF = 1; OF = 0; SF = 0; risultato errato
            .exit
            end
```

8086: Rappresentazione dei numeri

- Il processore 8086 permette di svolgere operazioni su numeri in *complemento a 2* (con segno) o in *binario puro*
- Analizziamo l'istruzione add:
  - Carry Flag è settata nel passare tra FFFF e 0000
  - Overflow Flag è settata nel passare tra 7FFF e 8000



- Per valutare condizioni di overflow occorre prestare attenzione al comportamento delle flag tenendo conto del tipo di rappresentazione che si intende utilizzare (il comportamento del processore non cambia).

Esercizio 5

- Dato un vettore di DIM word in memoria, rimpiazzarlo con il vettore inverso (senza usare un altro vettore di appoggio).

prima
423
3191
23
11
-412
3
9

dopo
9
3
-412
11
23
3191
423

Implementazione

- Non è possibile effettuare operazioni tra due indirizzi di memoria
- Utilizziamo:
  - il registro CX come contatore
  - il registro SI per l'indirizzo "basso"
  - il registro DI per l'indirizzo "alto"
  - il registro AX per il dato temporaneo.

Codice

```
DIM EQU 7

.model small
.stack
.data

vettore dw 423, 3191, 23, 11, -412, 3, 9

.code
.startup

mov cx, DIM/2        ; calcolato @compile-time; funziona con DIM pari o dispari
xor si, si           ; azzeramento indice elemento origine
mov di, (DIM-1)*2    ; calcolo indice dell'ultimo elemnento (@compile-time)
```

Codice [cont.]

```
ciclo: mov ax, vettore[si]
       xchg vettore[di], ax
       mov vettore[si], ax
       add si, 2
       sub di, 2
       dec cx
       jnz ciclo

.exit
end
```

Implementazione alternativa

- Utilizzo lo *stack*:
  - Prima leggo il vettore ed eseguo DIM operazioni di *push*
  - Dopo eseguo DIM operazioni di *pop* e scrivo il vettore.

Codice

```
DIM EQU 7

.model small
.stack
.data

vettore dw 423, 3191, 23, 11, -412, 3, 9

.code
.startup

mov cx, DIM
xor si, si
```

Codice [cont.]

```
ciclo1:  push vettore[si]
         add si, 2
         dec cx
         jnz ciclo1

mov cx, DIM
xor si, si
ciclo2:  pop vettore[si]
         add si, 2
         dec cx
         jnz ciclo2

.exit
end
```