Esercizi Assembly 3

M. Rebaudengo – R. Ferrero

Politecnico di Torino
Dipartimento di Automatica e Informatica

Esercizio 1

- Si scriva un programma in Assembly 8086 che, presi due vettori di 4 *word* ciascuno come matrici riga e colonna, ne calcoli il prodotto.
- Si ricorda che

  Se $x = (x_1, x_2, \ldots, x_n)$ e $y = (y_1, y_2, \ldots, y_n)$ sono due vettori a n componenti, il prodotto fra il vettore colonna x e il vettore riga y coincide con la matrice di ordine $n \cdot n$ in cui l'elemento di indice ij è dato dal prodotto tra la i-esima componente di x e la j-esima componente di y. In formule:

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} (y_1 \quad y_2 \quad \cdots \quad y_n) = \begin{pmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_ny_1 & x_ny_2 & \cdots & x_ny_n \end{pmatrix}$$

Implementazione

- Per il prodotto serve una matrice: utilizziamo il *base indexed addressing*
  - Esempi:

spiazzamento[BX][DI]	*spiazzamento[BX][SI]*
spiazzamento[BP][DI]	*spiazzamento[BP][SI]*
[BX][DI]	*[BX][SI]*
[BP][DI]	*[BP][SI]*

Implementazione [cont.]

- Memorizziamo la matrice per righe, su WORD
- matrice[x][y] = matrice[BX][SI], con
  - x $\in$ [1, NUM_RIG] y $\in$ [1, NUM_COL]
  - BX = (x-1)*2*NUM_COL SI = (y-1)*2

Per passare da una riga alla successiva: sommare NUM_BYTE*NUM_COL (NUM_BYTE = 2 se si lavora con word)

Per passare da una colonna alla successiva: sommare NUM_BYTE

Codice

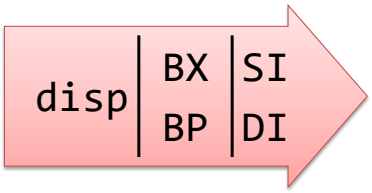
```
DIM EQU 4

.model small
.stack
.data

vetrig dw 12, 56, 1, -5
vetcol dw -51, 11, 0, 4

matrice dw DIM*DIM DUP (?)

.code
.startup

mov cx, DIM                ; contatore ciclo righe
mov bx, 0                  ; indice righe matrice
mov si, 0                  ; indice vettore colonna
```

Codice [cont.]

```
ciclorig: push cx
          mov cx, DIM              ; contatore ciclo colonne
          mov di, 0                ; indice colonne matrice
ciclocol: mov ax, vetcol[si]
          imul vetrig[di]
          jc overflow
          mov matrice[bx][di], ax
          add di, 2                ; incremento indice colonna (word)
          loop ciclocol
          pop cx
          add bx, 2*DIM            ; incremento indice riga
          add si, 2
          loop ciclorig
          jmp fine
overflow:
          ; istruzioni per gestione overflow…
fine:     .exit
          end
```

Indirizzamento 8086

- La seguente rappresentazione riassume tutti i 17 possibili modi di indirizzamento dell'8086:



- Esempi:

  [BX][SI], [DI], disp, disp[DI], disp[BX][DI]…

- N.B.: quando viene utilizzato [BP], il processore fa accesso allo *stack segment* (*data segment* in tutti gli altri casi).

Esercizio 2

- Si scriva un programma in grado di generare una tavola pitagorica (10x10) e memorizzarla.

Codice

```
DIM     EQU 10
        .model small
        .stack
        .data
Tavola  DB DIM*DIM DUP (?)  ; i numeri sono <101 => uso byte
        .code
        .startup
        mov ch, 1
        mov bx, 0
```

Codice [cont.]

```
ciclo1: mov cl, 1
        mov di, 0

ciclo2: mov al, ch
        mul cl
        mov tavola[bx][di], al
        inc cl
        inc di
        cmp cl, DIM
        jle ciclo2

        inc ch
        add bx, DIM
        cmp ch, DIM
        jle ciclo1

        .exit
        end
```

Esercizio 3

- Sia data la seguente tabella di *word*:

154	123	109	86	4	?
412	-23	-231	9	50	?
123	-24	12	55	-45	?
?	?	?	?	?	?

- Implementare in Assembly 8086 il programma che scriva la somma di ciascuna riga e colonna rispettivamente nell'ultima colonna e riga.

Codice

```
NUMCOL EQU 6
NUMRIG EQU 4

.model small
.stack
.data

tabella dw 154, 123, 109, 86, 4, ?
   dw 412, -23, -231, 9, 50, ?
   dw 123, -24, 12, 55, -45, ?
   dw NUMCOL dup(?)
```

Codice [cont.]

```
.code
.startup


mov cx, NUMRIG-1
mov bx, 0
ciclo1: push cx
        xor ax, ax                  ; azzeramento ax
        mov si, 0
        mov cx, NUMCOL-1
  ciclo11:  add ax, tabella[bx][si]
            add si, 2
            loop ciclo11
        Pop cx
        mov tabella[bx][si], ax     ; scrittura risultato
        add bx, 2*(NUMCOL)
        loop ciclo1
```

Codice [cont.]

```
mov cx, NUMCOL
mov si, 0
ciclo2: xor ax, ax
        mov bx, 0
        mov dx, NUMRIG-1
  ciclo22:  add ax, tabella[bx][si]
            add bx, NUMCOL*2
            dec dx
            jnz ciclo22
        mov tabella[bx][si], ax
        add si, 2
        loop ciclo2

.exit
end
```

Esercizio 4

- Scrivere un programma in Assembly che sommi i seguenti numeri rappresentati in un vettore di *byte*: -5, -45, -96, -128
- Sommare ancora al risultato il valore di addendo, variabile di tipo *doubleword* con valore 69000
- La somma deve essere salvata nella variabile `risultato` di tipo *doubleword*
  - In fase di debug, porre particolare attenzione al modo in cui sono memorizzate le doubleword.

Implementazione

- Le variabili di vettore sono *byte* in CA2
  - Per effettuare correttamente la somma è necessario estendere la rappresentazione su word: CBW (NB: solo per CA2)
- Il risultato parziale su *word* deve essere esteso a *doubleword*: CWD (NB: solo per CA2)
- Attenzione alla somma del *carry* quando necessario: istruzione ADC
- Le variabili di tipo *doubleword* sono memorizzate a partire dal byte meno significativo.

Codice

```
.model small
.stack
.data

vettore db -5, -45, -96, -128
risultato dd ?
addendo dd 69000

.code
.startup
mov cx, 4
mov si, 0
xor dx, dx                      ; azzeramento dx
ciclo: mov al, vettore[si]      ; somma elementi vettore in dx
       cbw                      ; estensione del segno: al -> ax
       add dx, ax
       inc si
       loop ciclo
```

Codice [cont.]

```
mov ax, dx
cwd                   ; estensione del segno: ax -> dx, ax

add ax, word ptr addendo        ; notare little-endianness
adc dx, word ptr addendo+2

mov word ptr risultato, ax
mov word ptr risultato+2, dx

.exit
end
```