

Esercizi Assembly 3

M. Sonza Reorda – M. Grosso

Politecnico di Torino
Dipartimento di Automatica e Informatica

Esercizio 1

- Si scriva un programma in linguaggio Assembly 8086 che scriva in un vettore definito di 20 elementi di tipo *word* i primi 20 valori della serie di Fibonacci.
- Serie di Fibonacci
 - $\text{vet}[i] = \text{vet}[i-1] + \text{vet}[i-2] \Rightarrow \text{vet} = 1, 1, 2, 3, 5, 8, \dots$

Codice

```

LUNG      EQU 20
          .model small
          .stack
          .data
vet        dw LUNG dup (?)
          .code
          .startup
          mov vet[0], 1
          mov vet[2], 1
          lea si, vet
          mov cx, LUNG-2
ciclo:    mov ax, [si]
          add si, 2
          add ax, [si]
          mov [si+2], ax
          loop ciclo

          .exit
          end

```

Esercizio 2

- 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 $x = (x_1, x_2, \dots, x_n)$ e $y = (y_1, y_2, \dots, y_n)$ 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:

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} (y_1 \quad y_2 \quad \cdots \quad y_n) = \begin{pmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_n \\ x_2 y_1 & x_2 y_2 & \cdots & x_2 y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_n y_1 & x_n y_2 & \cdots & x_n y_n \end{pmatrix}$$

Implementazione

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

– Esempi:

spiazzamento[BX][DI]

spiazzamento[BP][DI]

[BX][DI]

[BP][DI]

spiazzamento[BX][SI]

spiazzamento[BP][SI]

[BX][SI]

[BP][SI]

Implementazione [cont.]

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

Per passare da una riga alla successiva: sommare $\text{NUM_BYTE} * \text{NUM_COL}$ ($\text{NUM_BYTE} = 2$ 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 16 DUP (?)

.code
.startup

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

```

Codice [cont.]

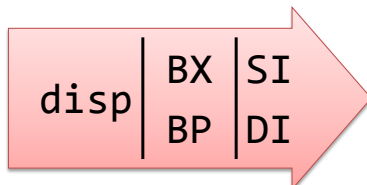
```

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 3

- 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 4

- 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 5

- Un modo per calcolare la radice quadrata approssimata di un numero intero consiste nel contare la quantità di numeri dispari che possono essere sottratti dal numero di partenza. La soluzione proposta in linguaggio C è la seguente:

```
main()
{
    int num, sqr=0, disp=1;
    ...
    num--;
    while (num >= 0)
    {
        sqr++;
        disp += 2;
        num -= disp;
    }
    ...
}
```

- Realizzare un programma in Assembly 8086 che calcoli la radice quadrata approssimata di un intero positivo (16 bit).

Codice

```
.model small
.stack
.data

numero    dw 48
risultato dw ?

.code
.startup
mov cx, -1
mov dx, -1
mov ax, numero
ciclo: inc cx
      add dx, 2
      sub ax, dx
      jnc ciclo          ; salta se carry flag != 1

mov risultato, cx
.exit
end
```