

Esercizi Assembly 4

M. Rebaudengo – R. Ferrero

Politecnico di Torino
Dipartimento di Automatica e Informatica

Esercizio 1

- Si scriva un programma in linguaggio Assembly 8086 che dica se un'equazione di secondo grado nella forma $ax^2+bx+c=0$ ha o meno soluzioni reali. I coefficienti a , b e c siano variabili di tipo *word*.
 - Si ricorda che la soluzione di un'equazione di secondo grado ha la forma:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Codice

```
CR    EQU 13
NL    EQU 10
LUNG_MSG EQU 30
      .model small
      .stack
      .data
aa    dw 2
bb    dw 4
cc    dw 2
sol_msg    db "Esistono due soluzioni reali", CR, NL
no_sol_msg db "Non esistono soluzioni reali", CR, NL
sol_coinc  db "Due soluzioni coincidenti!!!", CR, NL
```

Codice [cont.]

```
.code
.startup
mov ax, bb
imul bb
jc overflow ; ho deciso di lavorare al più con numeri rappresentabili su word
push ax
mov ax, aa
imul cc
jc overflow
mov bx, 4
imul bx
jc overflow
pop bx
sub bx, ax
jo overflow
js mess2
jz mess3
lea si, sol_msg
jmp next
mess2: lea si, no_sol_msg
jmp next
mess3: lea si, sol_coinc
```

Codice [cont.]

```
next: mov bx, LUNG_MSG
      mov ah, 2
ciclo: mov dl, [si]
      INT 21h
      inc si
      dec bx
      jnz ciclo
      jmp fine

overflow: nop

fine: .exit
      end
```

Esercizio 2

- Si scriva un programma in grado di calcolare il valore di un insieme di monete di diverso importo (espresso in centesimi di Euro). Siano dati i seguenti vettori:
 - *valore*, vettore di *word* indicante il valore di ciascun tipo di moneta
 - *monete*, vettore di *byte* indicante il numero di monete di ciascun tipo.
- Ad esempio, con
 - *valore* dw 1, 2, 5, 10, 20, 50, 100, 200
 - *monete* db 100, 23, 17, 0, 79, 48, 170, 211si hanno 100 monete da 1 centesimo, 23 monete da 2 centesimi, e così via.
- Il programma deve fornire il risultato aggiornando due variabili precedentemente dichiarate, di tipo *word*, denominate *euro* e *cent*, e rappresentanti rispettivamente l'importo in euro e in centesimi. Nell'esempio, il valore risultante è pari a 63411 centesimi, quindi alla fine del programma le due variabili *euro* e *cent* varranno rispettivamente 634 e 11.

Codice

```
LUNG EQU 8
.MODEL small
.STACK

.DATA
valore DW 1, 2, 5, 10, 20, 50, 100, 200
monete DB 100, 23, 17, 0, 79, 48, 170, 211
euro DW ?
cent DW ?

.CODE
.STARTUP

XOR SI, SI ; indice per il vettore valore
XOR DI, DI ; indice per il vettore monete
XOR BX, BX ; parte bassa dell'accumulatore
XOR CX, CX ; parte alta dell'accumulatore
```

Codice [cont.]

```
ciclo:
MOV AL, monete[DI]
XOR AH, AH
MUL valore[SI]
ADD BX, AX
ADC CX, DX
JC overflow
ADD SI, 2
INC DI
CMP DI, LUNG
JL ciclo

MOV AX, BX
MOV DX, CX
MOV CX, 100
DIV CX
MOV euro, AX
MOV cent, DX
JMP fine

overflow:
;gestione overflow

fine:

.EXIT
END
```

Esercizio 3

- Sia data una matrice quadrata di *word* memorizzata per righe (numero di righe pari a DIM, con DIM dichiarato come costante).
- Si scriva un programma che sia in grado di valutare se la matrice quadrata è simmetrica o diagonale. Il programma dovrà stampare a video un valore pari a:
 - 2 se la matrice è diagonale
 - 1 se la matrice è simmetrica
 - 0 se la matrice non è simmetrica.

Esercizio 3 [cont.]

- Si ricorda che in una matrice diagonale solamente i valori della diagonale principale possono essere diversi da 0, mentre una matrice simmetrica ha la proprietà di essere la trasposta di se stessa

- Esempio di matrice diagonale:
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$
- Esempio di matrice simmetrica:
$$\begin{bmatrix} 1 & 4 & 5 & 6 & 7 \\ 4 & 2 & 8 & 6 & 4 \\ 5 & 8 & 3 & 2 & 9 \\ 6 & 6 & 2 & 4 & 4 \\ 7 & 4 & 9 & 4 & 5 \end{bmatrix}$$

Codice

```
DIM EQU 5
        .model small
        .stack
        .data

matrix  dw 1, 4, 5, 6, 7
        dw 4, 2, 8, 6, 4
        dw 5, 8, 3, 2, 9
        dw 6, 6, 2, 4, 4
        dw 7, 4, 9, 4, 5

        .code
        .startup
        MOV DX, 2          ; ipotesi iniziale: e' diagonale
        MOV CX, DIM*2
        XOR BX, BX
```

Codice [cont.]

```
ciclo1: MOV SI, 2
        MOV DI, DIM*2
ciclo2: MOV AX, matrix[BX][SI]
        CMP AX, 0
        JZ next
        MOV DX, 1          ; non e' diagonale
next:   CMP AX, matrix[BX][DI]
        JNE nosimm
        ADD SI, 2
        ADD DI, DIM*2
        CMP SI, CX
        JNE ciclo2
        ADD BX, DIM*2+2
        SUB CX, 2
        CMP CX, 2
        JNZ ciclo1

        JMP fine
```

Codice [cont.]

```
nosimm: MOV DX, 0 ; non e' simmetrica

fine:   ADD DL, '0'
        MOV AH, 2
        INT 21H

.exit
end
```

Esercizio 4

- Sia data una matrice quadrata di *byte* di dimensione 8x8 preinizializzata. La matrice contiene valori *unsigned*.
- Per ogni elemento della matrice si calcoli la somma dei 4 elementi limitrofi (nelle posizioni N, E, S, O; per gli elementi lungo i bordi si consideri solo il sottoinsieme di elementi esistenti). Quindi, si trovi l'elemento per cui tale somma è massima e ne si forniscano le coordinate di riga e colonna.
- In caso di occorrenze multiple, si operi una scelta opportuna.

Esercizio 4 [cont.]

- Esempio:

0	4	0	0	0	0	0	60
0	5	0	0	11	0	0	0
0	5	7	0	0	10	0	0
0	0	0	9	0	0	49	0
0	0	10	0	0	0	0	0
0	10	3	9	0	0	12	0
0	0	58	0	0	17	0	0
0	1	0	0	3	0	0	0

- Risultato: $x = 3$, $y = 6$

Codice

```
DIM      EQU 8
          .model small
          .stack
          .data
matrice db 0, 4, 0, 0, 0, 0, 0, 60
        db 0, 5, 0, 0, 11, 0, 0, 0
        db 0, 5, 7, 0, 0, 10, 0, 0
        db 0, 0, 0, 9, 0, 0, 49, 0
        db 0, 0, 10, 0, 0, 0, 0, 0
        db 0, 10, 3, 9, 0, 0, 12, 0
        db 0, 0, 58, 0, 0, 17, 0, 0
        db 0, 1, 0, 0, 3, 0, 0, 0
coordx dw ?
coordy dw ?
          .code
          .startup
MOV CX, DIM
XOR DX, DX      ; azzeramento valore massimo
XOR BX, BX
```


Codice [cont.]

```
cicloR: PUSH CX
        MOV CX, DIM
        XOR SI, SI

cicloC: XOR AX, AX      ; azzeramento accumulatore
        CMP SI, 0      ; verifica elemento 0
        JE next1
        ADD AL, matrice[BX][SI][-1]
        ADC AH, 0
next1:  CMP BX, 0      ; verifica elemento N
        JE next2
        ADD AL, matrice[BX][SI][-DIM]
        ADC AH, 0
```

Codice [cont.]

```
next2:  CMP SI, DIM-1 ; verifica elemento E
        JE next3
        ADD AL, matrice[BX][SI][1]
        ADC AH, 0
next3:  CMP BX, DIM*(DIM-1) ; verifica elemento S
        JE next4
        ADD AL, matrice[BX][SI][DIM]
        ADC AH, 0
next4:  CMP AX, DX
        JB next5      ; salvo ultimo massimo trovato
        MOV DX, AX
        MOV coordy, BX
        MOV coordx, SI
```

Codice [cont.]

```
next5: INC SI
      LOOP cicloC
      POP CX

      ADD BX, DIM
      LOOP cicloR

      MOV AX, coordy
      MOV BL, DIM
      DIV BL
      INC AX
      MOV coordy, AX
      INC coordx

      .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
```