

Esercizi Assembly 6

M. Rebaudengo – R. Ferrero

Politecnico di Torino
Dipartimento di Automatica e Informatica

Esercizio 1

- Si scriva un programma in linguaggio Assembly 8086 per la conversione di una parola di caratteri minuscoli in caratteri maiuscoli, attraverso un'opportuna procedura.
- Si passi alla procedura il codice ASCII di un carattere alla volta come parametro *by value* utilizzando il registro AX; lo stesso registro deve contenere il carattere convertito.

Codice

```
Lung      EQU 6

          .model small
          .stack
          .data
stringa DB "parola"
          .code
          .startup
          MOV CX, 6
          LEA SI, stringa
          XOR AX, AX

          ciclo: MOV AL, [SI]
                  CALL converti
                  MOV [SI], AL
                  INC SI
                  LOOP ciclo
                  .exit

          converti PROC
                  ADD AL, 'A'
                  SUB AL, 'a'
                  RET
          converti ENDP

          END
```

Esercizio 2

- Si scriva un programma in linguaggio Assembly 8086 che stampi a video una stringa tramite una procedura.
- Tale procedura riceve come parametro l'indirizzo iniziale della stringa (passaggio *by reference*) e la sua lunghezza (*by value*), rispettivamente nei registri AX e BX.

Codice

```
LUNG EQU 44
CR EQU 13
NL EQU 10

.model small
.stack
.data
stringa DB "Meglio un uovo oggi che una gallina domani", CR, NL

.code
.startup
LEA AX, stringa
MOV BX, LUNG
CALL stampa
.exit
```

Codice [cont.]

```
stampa PROC
    PUSH AX ; salvataggio valori dei registri utilizzati dalla procedura
    PUSH BX
    PUSH DX
    PUSH SI
    MOV SI, AX
    MOV AH, 2
ciclo: MOV DL, [SI]
    INT 21h
    INC SI
    DEC BX
    JNZ ciclo
    POP SI ; ripristino dei valori iniziali
    POP DX
    POP BX
    POP AX
    RET
stampa ENDP

END
```

Esercizio 3

- Si scriva un programma in linguaggio Assembly 8086 che esegua la media tra gli elementi corrispondenti di due vettori definiti di tipo *word*, utilizzando una procedura con passaggio di parametri tramite variabili globali.
 - Si assuma che la somma di ciascuna coppia di elementi nei due vettori non produca overflow.
 - La procedura riceve i due valori su cui calcolare la media attraverso il meccanismo di passaggio dei parametri specificato, e restituisce il risultato analogamente.

Codice

```
lung      EQU 4

.model small
.stack
.data
a         DW 9, 1, 3, 5
b         DW 19, 2, 4, 6
res       DW lung DUP (0)
n1        DW ?
n2        DW ?
med       DW ?

.code
.startup
MOV CX, lung
XOR SI, SI
```

Codice [cont.]

```
ciclo: MOV AX, a[SI]
        MOV n1, AX
        MOV AX, b[SI]
        MOV n2, AX
        CALL media
        MOV AX, med
        MOV res[SI], AX
        ADD SI, 2
        LOOP ciclo
        .exit

media    PROC                                ; riceve dati da n1 e n2 - risultato in med
        PUSH AX
        MOV AX, n1
        ADD AX, n2
        SHR AX, 1
        MOV med, AX
        POP AX
        RET
media    ENDP
        END
```

Esercizio 4

- Si scriva un programma in linguaggio Assembly 8086 che esegua la media tra gli elementi corrispondenti di due vettori definiti di tipo *word*, utilizzando una procedura con passaggio di parametri tramite registri.

Codice

```
lung      EQU 4

.model small
.stack
.data
a  DW 9, 1, 3, 5
b  DW 19, 2, 4, 6
res DW lung dup (0)

.code
.startup
MOV CX, lung
XOR SI, SI
```

Codice [cont.]

```
ciclo: MOV AX, a[SI]
      MOV BX, b[SI]
      CALL media
      MOV res[SI], AX
      ADD SI, 2
      LOOP ciclo
      .exit

media  PROC                                ; riceve dati in AX e BX - risultato in AX
      ADD AX, BX
      SHR AX, 1
      RET
media  ENDP
      END
```

Esercizio 5

- Si scriva un programma in linguaggio Assembly 8086 che esegua la media tra gli elementi corrispondenti di due vettori definiti di tipo *word*, utilizzando una procedura con passaggio di parametri tramite stack.

Codice

```
lung      EQU 4

.model small
.stack
.data
a        DW 9, 1, 3, 5
b        DW 19, 2, 4, 6
res      DW lung dup (0)

.code
.startup
MOV CX, lung
XOR SI, SI
```

Codice [cont.]

```
ciclo: SUB SP, 2          ; riserva spazio per valore di ritorno
        PUSH a[SI]
        PUSH b[SI]
        CALL media
        ADD SP, 4
        POP res[SI]
        ADD SI, 2
        LOOP ciclo
        .exit

media    PROC            ; riceve dati via stack - risultato in stack
        MOV BP, SP
        PUSH AX
        MOV AX, [BP+4]
        ADD AX, [BP+2]
        SHR AX, 1
        MOV [BP+6], AX
        POP AX
        RET
media    ENDP
        END
```

Esercizio 6

- Scrivere un programma che calcoli il determinante di una matrice 3x3 i cui elementi sono interi con segno (*word*), usando una procedura che calcola il determinante di una matrice 2x2
 - Sia lecito supporre che non si verifichi mai *overflow* nelle somme e nelle moltiplicazioni.

Codice

passaggio di parametri con variabile globale

```
.model small
.stack
.data
m_3x3    DW 3, 0, -3
          DW 1, 4, 2
          DW 2, -1, 3
m_2x2    DW 2*2 dup (?)
res       DW 0
.code
.startup      ; metodo di Laplace: scelgo la prima riga
MOV AX, m_3x3[6][2]
MOV m_2x2[0][0], AX
MOV AX, m_3x3[6][4]
MOV m_2x2[0][2], AX
MOV AX, m_3x3[12][2]
MOV m_2x2[4][0], AX
MOV AX, m_3x3[12][4]
MOV m_2x2[4][2], AX ; parametri by value con var. globale m_2x2
CALL calc_2x2      ; salva il risultato in AX
IMUL m_3x3[0][0]
```

Codice [cont.]

```
ADD RES, AX

MOV AX, m_3x3[6][0]
MOV m_2x2[0][0], AX
MOV AX, m_3x3[6][4]
MOV m_2x2[0][2], AX
MOV AX, m_3x3[12][0]
MOV m_2x2[4][0], AX
MOV AX, m_3x3[12][4]
MOV m_2x2[4][2], AX
CALL calc_2x2
; salva risultato in AX
IMUL m_3x3[0][2]
SUB res, AX

MOV AX, m_3x3[6][0]
MOV m_2x2[0][0], AX
MOV AX, m_3x3[6][2]
MOV m_2x2[0][2], AX
MOV AX, m_3x3[12][0]
MOV m_2x2[4][0], AX
MOV AX, m_3x3[12][2]

MOV m_2x2[4][2], AX
CALL calc_2x2
END
```

```
MOV m_2x2[4][2], AX
CALL calc_2x2
; salva risultato in AX
IMUL m_3x3[0][4]
ADD res, AX

.exit

calc_2x2 PROC
PUSH BX ; salvataggio dei registri
PUSH DX
MOV AX, m_2x2[0][2]
IMUL m_2x2[4][0]
MOV BX, AX
MOV AX, m_2x2[0][0]
IMUL m_2x2[4][2]
SUB AX, BX
POP DX
POP BX
RET
calc_2x2 ENDP
```

Codice

passaggio di parametri tramite stack

```
.model small
.stack
.data
m_3x3 DW 3, 0, -3
        DW 1, 4, 2
        DW 2, -1, 3
res DW 0
.code
.startup ; metodo di Laplace: scelgo la prima riga

SUB SP, 2 ; riservo spazio per il valore di ritorno
; salvo nello stack i quattro elementi della matrice 2x2
; ottenuta eliminando la prima riga e la prima colonna
PUSH m_3x3[6][2]
PUSH m_3x3[6][4]
PUSH m_3x3[12][2]
PUSH m_3x3[12][4]
CALL calc_2x2
ADD SP, 8
POP AX
IMUL m_3x3[0][0]
ADD RES, AX
```

Codice [cont.]

```
SUB SP, 2 ; riservo spazio per il valore di ritorno
; salvo nello stack i quattro elementi della matrice 2x2
; ottenuta eliminando la prima riga e la seconda colonna
PUSH m_3x3[6][0]
PUSH m_3x3[6][4]
PUSH m_3x3[12][0]
PUSH m_3x3[12][4]
CALL calc_2x2
ADD SP, 8
POP AX
IMUL m_3x3[0][2]
SUB res, AX

SUB SP, 2 ; riservo spazio per il valore di ritorno
; salvo nello stack i quattro elementi della matrice 2x2
; ottenuta eliminando la prima riga e la terza colonna
PUSH m_3x3[6][0]
PUSH m_3x3[6][2]
PUSH m_3x3[12][0]
PUSH m_3x3[12][2]
CALL calc_2x2
ADD SP, 8
```

Codice [cont.]

```
POP AX
IMUL m_3x3[0][4]
ADD res, AX
.EXIT

calc_2x2 PROC
MOV BP, SP
PUSH AX
PUSH BX
MOV AX, [BP + 6]
IMUL [BP + 4]
MOV BX, AX
MOV AX, [BP + 8]
IMUL [BP + 2]
SUB AX, BX
MOV [BP + 10], AX

POP BX
POP AX
RET
calc_2x2 ENDP
END
```