# Esercizi Assembly 8086

M. Rebaudengo – R. Ferrero

Politecnico di Torino
Dipartimento di Automatica e Informatica

Esercizio 1)

Dati in memoria i seguenti due vettori di 50 word ciascuno:

- PREZZI rappresentante i prezzi di 50 articoli venduti in un negozio
- SCONTATI inizialmente di contenuto indeterminato,

si scriva una procedura in linguaggio Assembly 8086 in grado di calcolare il prezzo scontato di ciascun articolo e salvarlo nel corrispondente elemento del vettore SCONTATI. La procedura deve leggere da una variabile intera di tipo word denominata SCONTO l'ammontare dello sconto percentuale da applicare. Si esegua un arrotondamento alla cifra superiore se la parte decimale del prezzo risultante è maggiore o uguale a 0,5.

Inoltre, la procedura deve salvare in una variabile di tipo word TOTSCONTO l'ammontare totale delle riduzioni effettuate.

Esempio:

PREZZI: 39, 1880, 2394, 1000, 1590

SCONTO: 30

SCONTATI: 27, 1316, 1676, 700, 1113

TOTSCONTO: 2071

Di seguito un esempio di programma chiamante:

```
[…]
DIM EQU 5
.DATA
prezzi DW 39, 1880, 2394, 1000, 1590
scontati DW DIM DUP (?)
sconto DW 30
totsconto DW ?

.CODE
.STARTUP
CALL calcola_sconto
[…]
```

Esercizio 2)

Si abbia un vettore contenente alcuni interi rappresentanti anni passati (0 ÷ 2008). Si scriva una **procedura in linguaggio Assembly 8086** che sia in grado di determinare se tali anni sono bisestili. Si ricorda che un anno è bisestile se il suo numero è divisibile per 4, con l'eccezione che gli anni secolari (quelli divisibili per 100) sono bisestili solo se divisibili anche per 400. In altre parole,

```
IF (anno divisibile per 100)
  { IF (anno divisibile per 400)
       Anno_bisestile = TRUE
    ELSE Anno_bisestile = FALSE
  }
ELSE
  {  IF (anno divisibile per 4)
       Anno_bisestile = TRUE
     ELSE Anno_bisestile = FALSE
  }
```

La procedura deve ricevere come input:

- *tramite il registro SI*, l'offset di un vettore di *word* contenente gli anni da valutare
- *tramite il registro DI*, l'offset di un vettore di *byte* della stessa lunghezza, che dovrà contenere, al termine dell'esecuzione della procedura, nelle posizioni corrispondenti agli anni espressi nell'altro vettore, il valore 1 se l'anno è bisestile oppure 0 nel caso opposto
- *tramite il registro BX*, la lunghezza di tali vettori.

Esempio:

anni: 1945, 2008, 1800, 2006, 1748, 1600

risultato: 0, 1, 0, 0, 1, 1

lunghezza: 6

Di seguito un esempio di programma chiamante:

```
[…]
LUNG      equ 6
          .data
anni      dw 1945, 2008, 1800, 2006, 1748, 1600
ris       db LUNG DUP (?)
          .code
          .startup
          lea si, anni
          lea di, ris
          mov bx, lung
          call bisestile
[…]
```

Esercizio 3)

Si scriva una procedura “converti” in linguaggio Assembly 8086 in grado di rimuovere tutte le occorrenze di caratteri ripetuti consecutivamente in una stringa.

Ad esempio, la stringa “notte rossa” (dimensione 11) deve essere trasformata nella stringa “note rosa” (dimensione 9).

La procedura deve ricevere come input tramite *stack*:

- l’indirizzo della stringa origine (tale stringa dovrà essere sovrascritta dalla nuova stringa elaborata)
- la dimensione in byte della stringa origine.

Sempre tramite *stack*, la procedura deve fornire come output la dimensione della stringa trasformata. Non è ammesso l’uso di altre variabili in memoria.

Si supponga dunque che il programma chiamante contenga il seguente codice:

```
[…]
.code
lea ax, stringa
push ax
mov ax, DIMENSIONE
push ax
sub sp, 2
call converti
pop ax
mov DIMENSIONE_AGGIORNATA, ax

add sp, 4
[…]
```

Esercizio 4)

Si scriva una **procedura `potenza`** in linguaggio Assembly 8086 in grado di calcolare l'elevamento a potenza tra interi positivi.

La procedura riceve base ed esponente come *word unsigned* mediante lo stack e restitisce il risultato come *doubleword unsigned*, sempre mediante lo stack.

Di seguito un esempio di programma chiamante:

```
[…]
        .data
risult  DD ?
        .code
        .startup
        PUSH 3       ; base
        PUSH 12      ; esponente
        SUB  SP, 4
        CALL potenza
        POP AX
        POP DX
        ADD SP, 4
        mov risult, AX
        mov risult+2, DX
[…]
```

È inoltre richiesto di verificare la presenza di eventuali condizioni di *overflow*, che devono essere segnalate restituendo il valore esadecimale 0FFFFFFFFh.

Soluzione Esercizio 1)

```
DIM EQU 5
.model small
.stack
.data
prezzi dw 39, 1880, 2394, 1000, 1590
scontati dw DIM DUP (?)
sconto dw 30
totsconto dw ?

.code
.startup
call calcola_sconto
.exit

calcola_sconto proc
    push AX
    push BX
    push CX
    push DX
    push SI
    mov totsconto, 0
    mov CX, DIM  ; contatore elementi
    mov SI, 0    ; indice prezzi
    mov BX, 100
ciclo: mov AX, prezzi[SI]
       sub BX, sconto ; calcolo frazione prezzo
       mul BX         ; calcolo percentuale
       mov BX, 100
       div BX
       cmp DX, 50     ; arrotondamento
       jb next
       add AX, 1
next:  mov scontati[SI], AX
       mov DX, prezzi[SI]
       sub DX, AX
       add totsconto, DX
       add SI, 2
       loop ciclo
    pop SI
    pop DX
    pop CX
    pop BX
    pop AX
       ret
calcola_sconto endp
end
```

Soluzione Esercizio 2)

```
LUNG      equ 6
          .model small
          .stack
          .data
anni      dw 1945, 2008, 1800, 2006, 1748, 1600
ris       db LUNG DUP (?)

          .code
          .startup
          lea si, anni
          lea di, ris
          mov bx, lung
          call bisestile
          .exit

bisestile proc
          push ax
          push bx
          push cx
          push dx
          push si
          push di
ciclo:    mov [di], 0
          mov dx, [si]
          mov ax, dx
          mov cl, 100
          div cl
          cmp ah, 0
          jnz non_sec
          mov ax, dx
          mov dx, 0
          mov cx, 400
          div cx
          cmp dx, 0
          jnz next
          mov [di], 1
          jmp next
non_sec:  mov ax, dx
          test ax, 3
          jnz next
          mov [di], 1
next:     add si, 2
          inc di
          dec bx
          jnz ciclo
          pop di
```

```
          pop si
          pop dx
          pop cx
          pop bx
          pop ax
          ret
bisestile endp

          end
```

Soluzione Esercizio 3)

```
DIM       EQU 11
          .model small
          .stack
          .data

stringa   db "notte rossa"
newdim    dw ?

          .code
          .startup

          lea ax, stringa
          push ax
          mov ax, DIM
          push ax
          sub sp, 2
          call converti
          pop newdim
          add sp, 4
          .exit

converti  proc
          mov BP, SP
          push AX
          push BX
          push CX
          push DI
          push SI

          mov CX, [BP+4]
          mov SI, [BP+6]

          mov DI, SI
          inc DI
          dec CX
          mov BX, 1

ciclo:    mov AL, [DI]
          cmp AL, [SI]
          je  next
```

```
          inc si
          mov [si], al
          inc bx

next:     inc di
          loop ciclo

          mov [bp+2], bx
          pop SI
          pop DI
          pop CX
          pop BX
          pop AX

          ret
converti  endp

          end
```

Soluzione Esercizio 4)

```
        .model small
        .stack
        .data
risult  DD ?
        .code
        .startup
        PUSH 3
        PUSH 12
        SUB  SP, 4
        CALL potenza
        POP AX
        POP DX
        ADD  SP, 4
        mov risult, AX
        mov risult+2, DX
        .exit

potenza proc
        MOV BP, SP
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
        MOV AX, 1           ; accumulatore doubleword AX, DX
        XOR DX, DX          ; inizialmente a 1
        MOV CX, [BP+6]      ; contatore numero di iterazioni
        CMP CX, 0
        JE uno

ciclo:  MOV BX, DX
        MUL WORD PTR [BP+8] ; moltiplicazione "in colonna"
                            ; cominciando da parte meno significativa
        PUSH AX
```

```
        PUSH DX
        MOV AX, BX
        MUL WORD PTR [BP+8] ; moltiplicazione parte piu' significativa
        JC  ovf2            ; overflow se eccedo 16 bit su parte piu' sign.
        POP DX
        ADD DX, AX
        JC ovf1             ; overflow se eccedo 16 bit su parte piu' sign.
        POP AX
        LOOP ciclo

        MOV [BP+2], AX
        MOV [BP+4], DX
        JMP fine

uno:    MOV [BP + 2], 1
        MOV [BP + 4], 0
        JMP fine
ovf2:   ADD SP, 2
ovf1:   ADD SP, 2
        MOV [BP+2], 0FFFFh
        MOV [BP+4], 0FFFFh
fine:   POP DX
        POP CX
        POP BX
        POP AX
        RET

potenza endp
END
```