Esercizi Assembly 2

M. Rebaudengo – R. Ferrero

Politecnico di Torino
Dipartimento di Automatica e Informatica

Esercizio 1

- Si scriva un programma che calcoli la media (intera) tra i DIM valori di un vettore di *byte*, e ne salvi il risultato nella variabile `risultato`
- Verificare l'arrotondamento effettuato.

Codice

```
DIM EQU 10
.model small
.stack
.data

vettore db 2, 4, 5, -4, 5, -31, -90, 9, 10, 4
risultato db ?

.code
.startup

lea si, vettore
mov cx, DIM
mov bx, 0
```

Codice [cont.]

```
ciclo:   mov al, [si]
    cbw                 ; estensione a 16 bit: DA USARE SOLO CON CA2
    add bx, ax
    jo errore
    inc si
    loop ciclo

mov cl, dim
mov ax, bx
idiv cl
mov risultato, al
jmp fine

errore: [...]
fine:
.exit
end
```

Esercizio 2

- Si scriva un programma che stampi a video il valore decimale di un intero nell'intervallo [0, 2^{16}-1] memorizzato in un'opportuna variabile.

Implementazione

- Si utilizza un algoritmo in due passi:
  - Scomposizione del numero nelle sue cifre tramite divisioni successive per 10, salvando i resti e ripetendo l'operazione sul quoziente sino a che questo è diverso da zero
  - Visualizzazione delle cifre così ottenute in ordine inverso a quello di generazione, utilizzando lo stack
  - N.B.: le cifre devono essere convertite in caratteri ASCII prima della stampa.

Codice

```
          .model small
          .stack
          .data

datoin    dw 3721     ; dato da stampare

          .code
          .startup

           mov cx, 10        ; divisore
           mov dx, 0         ; word più significativo del dividendo
           mov ax, datoin    ; word meno significativo del dividendo
           mov bx, 0         ; contatore cifre
```

Codice [cont.]

```
conv:   div cx             ; ax / 10 = ax + dx / 10
        add dx, '0'        ; conversione ASCII
        push dx            ; inserimento in stack di cifra
        mov dx, 0          ; azzeramento dx per divisione successiva
        inc bx             ; incremento contatore cifre
        cmp ax, 0          ; verifica esistenza altre cifre da convertire
        jnz conv

        mov ah, 02h        ; codice system call per stampa carattere
stampa: pop dx             ; prelevamento di cifra da stack (in ordine inverso)
        int 21h            ; system call
        dec bx             ; decremento contatore cifre
        jnz stampa

       .exit
        end
```

Esercizio 3

- Si scriva un programma che richieda all'utente un intero positivo (eventualmente composto da più cifre, e concluso con ENTER) e lo salvi in una variabile di tipo *word*. L'inserimento di valori troppo grandi deve segnalare un errore.
  - Approfondimento: Acquisire 5 interi positivi separati da ENTER e memorizzarli in un vettore di *word*.

Implementazione

- Si utilizza un algoritmo in due passi:
  - nel primo si acquisiscono i caratteri ASCII;
  - nel secondo passo si convertono in intero, valutando la presenza eventuale di overflow
- I due passi possono essere svolti nello stesso ciclo.

Codice

```
DIM EQU 5
LF  EQU 10
CR  EQU 13

.model small
.stack
.data

message db 'Introdurre 5 interi positivi separati da ENTER: ', CR, LF
errore1 db 'ERRORE:   Caratteri non numerici introdotti!!!!!', CR, LF
errore2 db 'ERRORE:   Intero introdotto troppo grande!!!!!!!', CR, LF

vettris dw DIM DUP (?)
fattore dw 10
```

NB: La soluzione proposta include anche la stampa di alcuni messaggi a video allo scopo di rendere più completo il programma, ma che non sono necessari per soddisfare la richiesta.

Codice [cont.]

```
.code
.startup

mov ah, 02h         ; questo blocco stampa un messaggio a video
lea si, message
mov cx, 50          ; lunghezza message
stampa:  mov dl, [si]
    int 21h
    inc si
    loop stampa

mov cx, DIM        ; inizio ciclo acquisizione

lea di, vettris

ciclo:   mov ax, 0
    mov [di], ax
```

Codice [cont.]

```
leggi:   mov ah, 01h  ; codice system call per acquisizione caratteri
   int 21h
   cmp al, CR   ; verifico digitazione “ENTER” (separatore numeri)
   jz next
   cmp al, '0'  ; verifico che il carattere acquisito sia una cifra
   jl err1
   cmp al, '9'
   jg err1
   sub al, '0'  ; conversione cifra ASCII -> binario
   mov ah, 0
   mov bx, ax
   mov ax, [di]
   mul fattore
   jc err2
   add ax, bx
   jc err2
   mov [di], ax
   jmp leggi
next:    add di, 2
   loop ciclo
    jmp fine
```

Codice [cont.]

```
err1:    mov ah, 02h  ; stampa messaggio di errore
lea si, errore1
mov cx, 50          ; lunghezza messaggio
stampa1: mov dl, [si]
   int 21h
   inc si
   loop stampa1
   jmp fine

err2:    mov ah, 02h  ; stampa messaggio di errore
lea si, errore2
mov cx, 50          ; lunghezza messaggio
stampa2: mov dl, [si]
   int 21h
   inc si
   loop stampa2

fine:
.exit
end
```

Esercizio 4

- Siano date tre variabili di tipo *byte* in memoria, che rappresentino rispettivamente il numero di giorni, ore e minuti passati da un certo istante T_0. Si calcoli il numero totale di minuti passati da T_0, e tale valore sia salvato nella variabile di tipo *word* `risultato`.
  - Per estendere l'intervallo di numeri rappresentabili, si richiede di lavorare con una rappresentazione in binario puro
  - In caso di *overflow* della rappresentazione scrivere in `risultato` il valore FFFFh.

Implementazione

- È necessario eseguire una somma pesata delle variabili di ingresso:
  - Conversione dei giorni in ore
  - Somma delle ore
  - Conversione delle ore in minuti
  - Somma dei minuti
- Operazioni intermedie eseguite su *word*
- Valutare quando è possibile ottenere *overflow* nelle operazioni di somma/moltiplicazione!

Codice

```
.model small
.stack
.data

giorni db 45
ore    db 12
minuti db 40

risultato dw ?

.code
.startup

mov al, 24          ; conversione giorni -> ore
mul giorni          ; al * giorni => ah, al
```

Codice [cont.]

```
mov bl, ore
mov bh, 0           ; conversione di unsigned a 16 bit su BX
add ax, bx          ; somma ore: non può dare luogo a overflow anche con massimo
                    ; numero di giorni e ore (255*24+255)
mov bx, 60          ; conversione ore -> minuti
mul bx
jc ovf              ; verifico che risultato sia contenuto in ax
mov bl, minuti      ; bh vale 0
add ax, bx          ; somma minuti
jc ovf              ; verifica overflow su somma tra unsigned
mov risultato, ax
jmp fine

ovf:  mov risultato, 0FFFFh
fine:
.exit
end
```

- Non sottovalutate i problemi di rappresentazione dei numeri!
  - Si veda per esempio http://www.ima.umn.edu/~arnold/disasters/ariane.html

Esercizio 5

- 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
  - vet[i] = vet[i-1] + vet[i-2] => vet = 1, 1, 2, 3, 5, 8, …

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
```