

Architetture dei Sistemi a Elaborazione – a.a. 2010/11

Esercitazione di Laboratorio 1

1. Si scriva un programma in linguaggio Assembler 8086 che esegua le seguenti operazioni:
 - a. Definisca un vettore VETT di 6 elementi su 8 bit, non inizializzato
 - b. Riempia il VETT con la parola “Torino”
 - c. Memorizzi in AX il numero di bit su cui è espresso ogni elemento di VETT
 - d. Memorizzi in BX il numero di bit complessivamente occupati in memoria da VETT
 - e. Memorizzi in CX il numero di elementi di VETT.

Possibile soluzione:

```

.MODEL small
.STACK
.DATA
VETT DB 6 DUP (?)
.CODE
.STARTUP
LEA DI, VETT
MOV VETT, 'T'
MOV VETT[1], 'o'
MOV BYTE PTR [DI+2], 'r'
; questa linea (commentata) genera errore causa
; indeterminazione delle dimensioni (occorre cast con PTR)
;
MOV VETT+3, 'i'
MOV SI, 4
MOV VETT[SI], 'n'
MOV VETT[SI]+1, 'o'
MOV AX, TYPE VETT*8 ;Questa istruzione e la seguente danno errori di
MOV BX, SIZE VETT*8 ;Assemblaggio in emu8086
MOV CX, LENGTH VETT
.EXIT
.END

```

Nota bene: per chi utilizza EMU8086, se MASM installato, è possibile aggiungere #masm per assemblare con MASM e debuggare con EMU8086.

2. Si scriva un programma in linguaggio Assembler 8086 che esegua le seguenti operazioni:
 - a. Definiscano 2 vettori VETT1 e VETT2, ciascuno composto di 2 elementi su 16 bit, inizializzati a piacimento a valori positivi
 - b. Esegua la seguente operazione descritta di seguito in modo matematico:

$$(VETT1(0)/(VETT2(1)-1))^2 - ((-VETT1(1))/(VETT2(0)+1))^2$$
 - c. Si memorizzi il risultato in una variabile di dimensione opportuna.

Si modifichi il programma in modo che possa funzionare anche in caso di numeri con segno.

Possibile soluzione:

NUMERI SENZA SEGNO				NUMERI CON SEGNO (C2)			
<pre> .MODEL small .STACK .DATA VETT1 DW 10, 20 VETT2 DW 3, 4 RIS DD ? .CODE .STARTUP </pre>				<pre> .MODEL small .STACK .DATA VETT1 DW 10, 20 VETT2 DW 3, 4 RIS DD ? .CODE .STARTUP </pre>			

<pre> LEA SI, VETT1 LEA DI, VETT2 MOV BX, [DI]+2 DEC BX MOV AX, [SI] MOV DX, 0 DIV BX MUL AX MOV WORD PTR RIS, AX MOV WORD PTR RIS+2, DX MOV BX, [DI] INC BX MOV AX, [SI]+2 MOV DX, 0 DIV BX MUL AX SUB WORD PTR RIS, AX SBB WORD PTR RIS+2, DX .EXIT END </pre>	<pre> LEA SI, VETT1 LEA DI, VETT2 MOV BX, [DI]+2 DEC BX MOV AX, [SI] CWD IDIV BX IMUL AX MOV WORD PTR RIS, AX MOV WORD PTR RIS+2, DX MOV BX, [DI] INC BX MOV AX, [SI]+2 NEG AX CWD IDIV BX IMUL AX SUB WORD PTR RIS, AX SBB WORD PTR RIS+2, DX .EXIT END </pre>
--	---

3. Si scriva un programma in linguaggio Assembler 8086 che esegua le seguenti operazioni:

- Definisca un vettore VETT di 3 elementi ciascuno su 32 bit, inizializzato a piacimento a numeri positivi e negativi, senza utilizzare la direttiva DUP
- Esegua la somma di tutti numeri positivi contenuti in VETT e depositi il risultato in una variabile VARP su 32 bit
- Esegua la somma di tutti numeri negativi contenuti in VETT e depositi il risultato in una variabile VARN su 32 bit
- In entrambi i punti b. e c., si tenga conto dell'overflow: in caso di overflow la variabile risultato deve essere posta al valore 0.*

Si definiscano i valori di VETT più volte, cercando di osservare (tramite i flag) le possibili situazioni di carry e overflow.

Possibile soluzione:

```

.MODEL small
.STACK
.DATA
VETT DD 2,4,-3,6,5,800000FFH,0FFFFFFFH,-7
; non da overflow

;VETT DD 2,4,-3,6,5,80000000H,0FFFFFFFH,-7
; da overflow sui numeri negativi

;VETT DD 2,4,-3,6,7FFFFFFFH,800000FFH,0FFFFFFFH,-7
; da overflow sui numeri positivi

;VETT DD 2,4,-3,6,7FFFFFFFH,80000000H,0FFFFFFFH,-7
; da overflow su entrambi

VARP DD 0
VARN DD 0

.CODE
.STARTUP
LEA SI, VETT
MOV DL, 0
; uso DL come flag
; DL = 01H se overflow positivo
; DL = 02H se overflow negativo
; DL = 03H se overflow positivo e negativo
MOV CX, 8
NEW: MOV BX, WORD PTR [SI]
MOV AX, WORD PTR [SI]+2
CMP AX, 0
JL NEGAT

; il numero è positivo
TEST DL, 01H
JNZ NEXT
; test se overflow in somme precedenti

```

```

        ADD WORD PTR VARP, BX
        ADC WORD PTR VARP+2, AX
        JNO NEXT
        OR DL, 01H
        JMP NEXT

NEGAT:  ; il numero è negativo
        TEST DL, 02H
        JNZ NEXT
        ; test se overflow in somme precedenti
        ADD WORD PTR VARN, BX
        ADC WORD PTR VARN+2, AX
        JNO NEXT
        OR DL, 02H

NEXT:   ADD SI, 4
        LOOP NEW

        TEST DL, 01H
        JZ N_O_PO
        MOV WORD PTR VARP, 0
        MOV WORD PTR VARP+2, 0
N_O_PO: TEST DL, 02H
        JZ N_O_NE
        MOV WORD PTR VARN, 0
        MOV WORD PTR VARN+2, 0
N_O_NE: NOP

        .EXIT
        END

```

4. Si scriva un programma in linguaggio Assembler 8086 che esegua le seguenti operazioni:
- Definisca una matrice 5*5 chiamata MATR i cui elementi siano memorizzati su 16 bit, inizializzati a piacimento.
 - Senza perdere precisione, si memorizzi in una locazione di memoria scelta la somma di tutti gli elementi della riga 2
 - Senza perdere precisione, si memorizzi in una locazione di memoria scelta la somma di tutti gli elementi della colonna 3
 - Esegua la differenza dei due risultati ottenuti al punto b. e c. e la si memorizzi in una locazione di memoria scelta.

Possibile soluzione:

```

ROW EQU 10
COL EQU 2
.MODEL small
.STACK
.DATA
MATR DW 0,1,2,3,4
      DW 5,6,7,8,9
      DW 10,11,12,13,14
      DW 15,16,17,18,19
      DW 20,21,22,23,24
SUMR DD 0
SUMC DD 0
DIFF DD ?
.CODE
.STARTUP
LEA BX, MATR
ADD BX, 2*ROW
MOV SI, 0
MOV DX, 0
MOV AX, [BX][SI]
MOV DX, 0
ADD AX, [BX][SI+2]
ADC DX, 0
ADD AX, [BX][SI+4]
ADC DX, 0
ADD AX, [BX][SI+6]
ADC DX, 0
ADD AX, [BX][SI+8]
ADC DX, 0
MOV WORD PTR SUMR, AX
MOV WORD PTR SUMR+2, DX

LEA BX, MATR
MOV SI, 3*COL
MOV DX, 0

```

```
MOV     AX, [BX][SI]
MOV     DX, 0
ADD     BX, ROW
ADD     AX, [BX][SI]
ADC     DX, 0
ADD     BX, ROW
ADD     AX, [BX][SI]
ADC     DX, 0
ADD     BX, ROW
ADD     AX, [BX][SI]
ADC     DX, 0
ADD     BX, ROW
ADD     AX, [BX][SI]
ADC     DX, 0
MOV     WORD PTR SUMR, AX
MOV     WORD PTR SUMR+2, DX

MOV     AX, WORD PTR SUMC
SUB     WORD PTR SUMR, AX
MOV     AX, WORD PTR SUMC+2
SBB     WORD PTR SUMR+2, AX
.EXIT
END
```

;Si scriva un programma che esegue le seguenti operazioni:
; a. Definisca un vettore VETT di 6 elementi su 8 bit, non inizializzato
; b. Riempia il VETT con la parola "Torino"
; c. Memorizzi in AX il numero di bit su cui è espresso ogni elemento di VETT
; d. Memorizzi in BX il numero di bit complessivamente occupati in memoria da VETT
; e. Memorizzi in CX il numero di elementi di VETT

LEN equ 6

.model small
.data

VETT db LEN DUP (?)

.stack
.code
.startup

```
    lea si, VETT                ; mov si, offset VETT
    mov VETT[0], 'T'
    mov VETT[1], 'o'           ; mov [si+1], 'o'           ;ERRORE
    mov byte ptr [si+2], 'r'
    mov byte ptr [si+3], 'i'
    mov byte ptr [si+4], 'n'
    mov byte ptr [si+5], 'o'
    mov ax, TYPE VETT
    mov bx, SIZE VETT
    mov cx, LENGTH VETT
```

.exit
end

```
; Si scriva un programma che esegua le seguenti operazioni:
; a. Definiscano 2 vettori VETT1 e VETT2, ciascuno composto di 2 elementi
;    su 16 bit, inizializzati a piacimento a valori positivi
; b. Esegua la seguente operazione descritta di seguito in modo matematico:
;    (VETT1[0]/(VETT2[1]-1))^2 - ((-VETT1[1])/(VETT2[0]+1))^2
; c. Si memorizzi il risultato in una variabile di dimensione opportuna
; Si modifichi il programma in modo che possa funzionare anche in caso di
; numeri con segno
```

```
.model small
.data
```

```
VETT1 dw 2, 5
VETT2 dw 3, 7
RES    dd 0
```

```
.stack
.code
.startup
```

```
; (VETT1[0]/(VETT2[1]-1))^2 - ((-VETT1[1])/(VETT2[0]+1))^2
lea si, VETT1
mov bx, VETT2[2]      ; bx = VETT2[1]
dec bx                ; bx = bx - 1
mov ax, VETT1[0]      ; ax = VETT1[0]
mov dx, 0
div bx                ; ax = [dx,ax]/bx, dx = [dx,ax]%bx (ignorato?)
mul ax                ; [dx,ax] = ax * ax
mov word ptr RES, ax
mov word ptr RES+2, dx

mov bx, VETT2[0]      ; bx = VETT2[0]
inc bx                ; bx = bx + 1
mov ax, VETT1[2]      ; ax = VETT1[1]
neg ax                ; ax = -ax
div bx                ; ax = [dx,ax]/bx
dx = [dx,ax]%bx (ignorato?)
mul ax                ; [dx,ax] = ax * ax

; adesso facciamo la sottrazione finale
sub word ptr RES, ax
sbb word ptr RES+2, dx
```

```
.exit
end
```

```
; Si scriva un programma che esegua le seguenti operazioni:
; a. Definiscano 2 vettori VETT1 e VETT2, ciascuno composto di 2 elementi
;    su 16 bit, inizializzati a piacimento a valori positivi
; b. Esegua la seguente operazione descritta di seguito in modo matematico:
;    (VETT1[0]/(VETT2[1]-1))^2 - (VETT1[1]/(VETT2[0]+1))^2
; c. Si memorizzi il risultato in una variabile di dimensione opportuna
; Si modifichi il programma in modo che possa funzionare anche in caso di
; numeri con segno
```

```
.model small
.data
```

```
VETT1 dw 2, 5
VETT2 dw 3, 7
RES dd 0
```

```
.stack
.code
.startup
```

```
    lea si, VETT1
    mov bx, VETT2[2]      ; bx = VETT2[1]
    dec bx                ; bx = bx - 1
    mov ax, VETT1[0]      ; ax = VETT1[0]
    mov dx, 0
    div bx                ; ax = [dx,ax]/bx, dx = [dx,ax]%bx (ignorato?)
    mul ax                ; [dx,ax] = ax * ax
    mov word ptr RES, ax
    mov word ptr RES+2, dx
```

```
    mov bx, VETT2[0]      ; bx = VETT2[0]
    inc bx                ; bx = bx + 1
    mov ax, VETT1[2]      ; ax = VETT1[1]
    div bx                ; ax = [dx,ax]/bx
                        ; dx = [dx,ax]%bx (ignorato?)
    mul ax                ; [dx,ax] = ax * ax
```

```
; adesso facciamo la sottrazione finale
```

```
sub word ptr RES, ax
sbb word ptr RES+2, dx
```

```
.exit
end
```

```
; Si scriva un programma che esegua le seguenti operazioni:
; a. Definiscano 2 vettori VETT1 e VETT2, ciascuno composto di 2 elementi
; su 16 bit, inizializzati a piacimento a valori positivi
; b. Esegua la seguente operazione descritta di seguito in modo matematico:
;  $(VETT1[0]/(VETT2[1]-1))^2 - ((-VETT1[1])/(VETT2[0]+1))^2$ 
; c. Si memorizzi il risultato in una variabile di dimensione opportuna
; Si modifichi il programma in modo che possa funzionare anche in caso di
; numeri con segno
```

```
.model small
.data
```

```
VETT1 dw -2, 5
VETT2 dw 3, -7
RES dd 0
```

```
.stack
.code
.startup
```

```
lea si, VETT1
mov bx, VETT2[2] ; bx = VETT2[1]
dec bx ; bx = bx - 1
mov ax, VETT1[0] ; ax = VETT1[0]
cwd
idiv bx ; ax = [dx,ax]/bx
imul ax ; [dx,ax] = ax * ax
mov word ptr RES, ax
mov word ptr RES+2, dx
```

```
mov bx, VETT2[0] ; bx = VETT2[0]
inc bx ; bx = bx + 1
mov ax, VETT1[2] ; ax = VETT1[1]
neg ax
cwd
idiv bx ; ax = [dx,ax]/bx
imul ax ; [dx,ax] = ax * ax
```

```
; adesso facciamo la sottrazione finale
```

```
sub word ptr RES, ax
sbb word ptr RES+2, dx
```

```
.exit
end
```



```
; Si scriva un programma che esegua le seguenti operazioni:
; a. Definisca un vettore VETT di 3 elementi ciascuno su 32 bit, inizializzato a
;    piacimento a numeri positivi e negativi senza utilizzare la direttiva DUP.
; b. Esegua la somma di tutti i numeri positivi contenuti in VETT e
;    depositi il risultato in una variabile VARP su 32 bit.
; c. Esegua la somma di tutti i numeri negativi contenuti in VETT e
;    depositi il risultato in una variabile VARN su 32 bit.
; Si definiscano i valori di VETT piu' volte, cercando di osservare
; (tramite i flag) le possibili situazioni di carry e overflow
```

```
    LEN    equ 8
    OVF_P   equ 0001H
    OVF_N   equ 0002H
.model small
.data

; non da overflow
;VETT dd 2, 4, -3, 6, 5, 800000FFh, 0FFFFFFFh, -7
; overflow sui numeri negativi
;VETT dd 2, 4, -3, 6, 5, 80000000h, 0FFFFFFFh, -7
; overflow sui numeri positivi
;VETT dd 2, 4, -3, 6, 7FFFFFFFh, 800000FFh, 0FFFFFFFh, -7
; overflow sui numeri positivi e negativi
VETT dd 2, 4, -3, 6, 7FFFFFFFh, 80000000h, 0FFFFFFFh, -7

    VARP dd 0
    VARN dd 0

.stack
.code
.startup

    mov     cx, LEN
    lea     si, VETT
    mov     dx, 0                ; lo uso come flag per vedere gli overflow
ciclo:    mov     bx, word ptr [si]
    mov     ax, word ptr [si]+2
    test    ax, 08000h
    jnz     is_neg

    ;il numero e' positivo, se ci sono ovf positivi non faccio la somma
    test    dl, OVF_P
    jnz     next
    add     word ptr VARP,    bx
    adc     word ptr VARP+2, ax
    jno     next
    or      dx, OVF_P
    jmp     next

is_neg:    ;il numero e' negativo, se ci sono ovf negativi non faccio la somma
    test    dl, OVF_N
    jnz     next
    add     word ptr VARN,    bx
    adc     word ptr VARN+2, ax
    jno     next
    or      dx, OVF_N

next:      add     si, 4
    dec     cx
    jnz     ciclo

    ;se ci sono ovf positivi, metto a 0 VARP
    test    dx, OVF_P
    jz      no_ovf_p
    mov     word ptr VARP, 0
    mov     word ptr VARP+2, 0

    ;se ci sono ovf negativi, metto a 0 VARN
no_ovf_p:  test    dx, OVF_N
    jz      fine
    mov     word ptr VARN, 0
    mov     word ptr VARN+2, 0

fine:      nop
    .exit
end
```

```
;Si Scriva un programma che esegua le seguenti operazioni:
; a. Definisca una matrisce 5*5 chiamata MATR i cui elementi siano
; memorizzati su 16 bit, inizializzati a piacimento
; b. Senza perdere precisione, si memorizzi in una locazione di memoria
; scelta la somma di tutti gli elementi della riga 2
; c. Senza perdere precisione, si memorizzi in una locazione di memoria
; scelta la somma di tutti gli elementi della colonna 3
; d. Esegua la differenza dei due risultati ottenuti ai punti b. e c. e
; la si memorizzi in una locazione di memoria scelta
```

NROW equ 5

```
.model small
.stack
.data
    MATR dw 0, 1, 2, 3, 4
          dw 5, 6, 7, 8, 9
          dw 10, 11, 12, 13, 14
          dw 15, 16, 17, 18, 19
          dw 20, 21, 22, 23, 24

    SUMR dd 0
    SUMC dd 0
    DIFF dd 0

.code
.startup
    mov cx, 5
    lea si, MATR+20

ciclo1:    mov ax, [si]
            add word ptr SUMR, ax
            adc word ptr SUMR+2, 0
            add si, 2
            dec cx
            jnz ciclo1

            mov cx, 5
            lea si, MATR+6
ciclo2:    mov ax, [si]
            add word ptr SUMC, ax
            adc word ptr SUMC+2, 0
            add si, 10
            dec cx
            jnz ciclo2

            mov ax, word ptr SUMR
            mov dx, word ptr SUMR+2
            sub ax, word ptr SUMC
            sbb dx, word ptr SUMC+2
            mov word ptr DIFF, ax
            mov word ptr DIFF+2, dx

    .exit
end
```