

# Calcolatori Elettronici

## Esercitazione 4

M. Sonza Reorda – M. Monetti

M. Rebaudengo – R. Ferrero

L. Sterpone – M. Grosso

Politecnico di Torino

Dipartimento di Automatica e Informatica

# Obiettivi

- Salti e controllo del flusso del programma
- Array (vettori e matrici)

# Esercizio 1

- Si scriva un programma in linguaggio Assembly MIPS che scriva in un vettore definito di 20 elementi di tipo *word* i primi 20 valori della serie di Fibonacci.
- Serie di Fibonacci
  - $\text{vet}[i] = \text{vet}[i-1] + \text{vet}[i-2] \Rightarrow \text{vet} = 1, 1, 2, 3, 5, 8, \dots$

# Soluzione

```
                .data
NUM_ELEM = 20
DIM = 4 * NUM_ELEM
wVet:          .space DIM

                .text
                .globl main
                .ent main

main:
                li $t0, 0      # contatore
                # Caricamento primi 2 valori
                li $t1, 1
                sw $t1, wVet($t0)
                addi $t0, $t0, 4
                li $t2, 1
                sw $t2, wVet($t0)
                addi $t0, $t0, 4
```

# Soluzione [cont.]

```
ciclo:    add $t3, $t1, $t2
          sw $t3, wVet($t0)
          move $t1, $t2
          move $t2, $t3
          addi $t0, $t0, 4
          blt $t0, DIM, ciclo
          li $v0, 10
          syscall
          .end main
```

## Esercizio 2

- Scrivere un programma che, dati due operandi ***opa*** e ***opb*** di tipo *word* in memoria, del valore rispettivo di 2043 e 5, esegua un'operazione tra interi scelta dall'utente e salvi il risultato nella variabile *word* ***res***
- A seconda dell'intero digitato dall'utente, il programma deve eseguire:
  - $0 \rightarrow \text{res} = a+b$
  - $1 \rightarrow \text{res} = a-b$
  - $2 \rightarrow \text{res} = a*b$
  - $3 \rightarrow \text{res} = a/b$  (divisione intera).

# Implementazione

- Occorre implementare un costrutto *switch*:

```
switch (espressione)
{
    case val1: sequenza1;
               break;
    case val2: sequenza2;
               break;
    ...
    default:   sequenza_def;
}
```

- Si possono utilizzare:
  - operazioni di *compare* e salti condizionati a blocchi di istruzioni
  - una tabella di *jump* e un'unica istruzione di salto incondizionato

```
                .data
tab:            .word somma, sottrazione, moltiplic, divisione
                ...
                .code
                ...
                lw $t2, tab($t0)
                jr $t2
somma:          ...
sottrazione:    ...
```

# Soluzione

```
.data
opa:    .word 2043
opb:    .word 5
res:    .space 4
tab:    .word somma, sottrazione, moltiplic, divisione

.text

.globl main
.ent main
main:   lw $t0, opa
        lw $t1, opb
        li $v0, 5
        syscall
        blt $v0, 0, errore
        bgt $v0, 3, errore
        sll $t2, $v0, 2
        lw $t2, tab($t2)
        jr $t2
```



# Soluzione [cont.]

```
somma:      add $t0, $t0, $t1
            b fine
sottrazione: sub $t0, $t0, $t1
            b fine
moltiplic:   mul $t0, $t0, $t1
            b fine
divisione:   div $t0, $t0, $t1
            b fine

errore:      # gestione errore
            b fine2

fine:        sw $t0, res
fine2:       li $v0, 10
            syscall

.end main
```

# Esercizio 3

- Si scriva un programma MIPS che, dati due vettori di 4 *word* ciascuno come matrici riga e colonna, ne calcoli il prodotto.

- Si ricorda che

Se  $x = (x_1, x_2, \dots, x_n)$  e  $y = (y_1, y_2, \dots, y_n)$  sono due vettori a  $n$  componenti, il prodotto fra il vettore colonna  $x$  e il vettore riga  $y$  coincide con la matrice di ordine  $n \cdot n$  in cui l'elemento di indice  $ij$  è dato dal prodotto tra la  $i$ -esima componente di  $x$  e la  $j$ -esima componente di  $y$ . In formule:

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} (y_1 \quad y_2 \quad \cdots \quad y_n) = \begin{pmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_n \\ x_2 y_1 & x_2 y_2 & \cdots & x_2 y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_n y_1 & x_n y_2 & \cdots & x_n y_n \end{pmatrix}$$

# Soluzione

```
                .data
NUM_ELEM = 4
DIM = 4 * NUM_ELEM
vetrig:        .word 12, 56, 1, -5
vetcol:        .word -51, 11, 0, 4
matrice:       .space DIM * NUM_ELEM

                .text
                .globl main
                .ent main
main:          li $t0, 0      # offset matrice
               li $t1, 0      # contatore righe

ciclorig:      lw $t3, vetrig($t1)
               li $t2, 0      # contatore colonne
```

# Soluzione [cont.]

```
ciclocol:    lw $t4, vetcol($t2)
             mult $t3, $t4
             mfhi $t4
             beq $t4, $0, noOverflow
             bne $t4, 0xFFFFFFFF, overflow
noOverflow:  mflo $t4
             sw $t4, matrice($t0)
             addi $t0, $t0, 4
             addi $t2, $t2, 4
             blt $t2, DIM, ciclocol
             addi $t1, $t1, 4
             blt $t1, DIM, ciclorig
             j fine
overflow:    # istruzioni per gestione overflow
fine:       li $v0, 10
             syscall
             .end main
```

## Esercizio 4

- Si scriva un programma in grado di generare una tavola pitagorica (10x10) e memorizzarla.

# Soluzione

```

DIM = 10
Pitagora:
    .data
    .space 100
    .text
    .globl main
    .ent main
main:
    la $t0, Pitagora
    li $t1, 1      # contatore righe
ciclorig:
    li $t2, 1      # contatore colonne
ciclocol:
    multu $t1, $t2
    mflo $t3
    sb $t3, ($t0)
    addi $t0, $t0, 1
    addi $t2, $t2, 1
    ble $t2, DIM, ciclocol
    addi $t1, $t1, 1
    ble $t1, DIM, ciclorig
    li $v0, 10
    syscall
    .end main
```

# Esercizio 5

- Sia data la seguente tabella di *word*:

154	123	109	86	4	?
412	-23	-231	9	50	?
123	-24	12	55	-45	?
?	?	?	?	?	?

- Implementare in Assembly MIPS il programma che scriva la somma di ciascuna riga e colonna rispettivamente nell'ultima colonna e riga.

# Soluzione

```
.data
NUMCOL = 6
NUMRIG = 4
DIMRIG = 4 * NUMCOL
wMat:    .word    154, 123, 109, 86, 4, 0, 412, -23, -231, 9, 50, 0, 123, -24,
12, 55, -45, 0, 0, 0, 0, 0, 0
        .text
        .globl main
        .ent main
main:    la $t0, wMat
        li $t1, 1      # contatore righe
ciclorig1: li $t2, 1      # contatore colonne
        li $t3, 0      # accumulatore
ciclocol1: lw $t4, ($t0)
        add $t3, $t3, $t4
        addi $t0, $t0, 4
        addi $t2, $t2, 1
        blt $t2, NUMCOL, ciclocol1
        sw $t3, ($t0)
        addi $t0, $t0, 4
```



# Soluzione [cont.]

```
    addi $t1, $t1, 1
    blt $t1, NUMRIG, ciclorig1
    li $t1, 0      # contatore colonne
ciclocol2:    la $t0, wMat
    sll $t3, $t1, 2
    add $t0, $t0, $t3      # indirizzo del primo elemento della colonna
    li $t2, 1      # contatore righe
    li $t3, 0      # accumulatore
ciclorig2:    lw $t4, ($t0)
    add $t3, $t3, $t4
    addi $t0, $t0, DIMRIG
    addi $t2, $t2, 1
    blt $t2, NUMRIG, ciclorig2
    sw $t3, ($t0)
    addi $t0, $t0, 4
    addi $t1, $t1, 1
    blt $t1, NUMCOL, ciclocol2
    li $v0, 10
    syscall
    .end main
```