

Calcolatori Elettronici

Esercitazione 9

M. Sonza Reorda – M. Monetti

M. Rebaudengo – R. Ferrero

L. Sterpone – E. Vacca

Politecnico di Torino

Dipartimento di Automatica e Informatica

Esercizio 1

Sono date due matrici quadrate contenenti numeri con segno, memorizzate per righe, di DIMxDIM elementi. Si scriva una procedura **Variazione** in linguaggio MIPS in grado di calcolare la variazione percentuale (troncata all'intero) tra gli elementi di indice corrispondente della *riga* l della prima matrice ($[l, 0]$, $[l, 1]$, $[l, 2]$...) e della *colonna* l della seconda ($[0, l]$, $[1, l]$, $[2, l]$...). Ad esempio, nel caso di due matrici 3x3 e con $l = 2$:

$$\begin{bmatrix} 4 & -45 & 15565 \\ 6458 & 4531 & 124 \\ -548 & 2124 & 31000 \end{bmatrix} \quad \begin{bmatrix} 6 & -5421 & -547 \\ -99 & 4531 & 1456 \\ 4592 & 118 & 31999 \end{bmatrix}$$

il risultato è 0, -31, 3

Esercizio 1 [cont.]

- La variazione percentuale è calcolata come segue:

$$\textit{Variazione} = (Val2 - Val1) \cdot 100 / Val1$$

- La procedura riceve i seguenti parametri:
 - L'indirizzo della prima matrice mediante \$a0
 - L'indirizzo della seconda matrice mediante \$a1
 - L'indirizzo del vettore risultato mediante \$a2
 - La dimensione DIM tramite \$a3
 - L'indice i per mezzo dello stack.

Esercizio 1 [cont.]

- Di seguito un esempio di programma chiamante:

```
DIM = 3
DIM_RIGA = DIM * 4

        .data
mat1:    .word 4, -45, 15565,    6458, 4531, 124,    -548, 2124, 31000
mat2:    .word 6, -5421, -547,    -99, 4531, 1456,    4592, 118, 31999
indice:  .word 2
vet_out: .space DIM_RIGA

        .text
        .globl main
        .ent main
main:    [...]
        la $a0, mat1
        la $a1, mat2
        la $a2, vet_out
        li $a3, DIM
        subu $sp, $sp, 4
        lw $t0, indice
        sw $t0, ($sp)
        jal ProceduraVariazione
        addu $sp, $sp, 4
        [...]
        .end main
```

[1]-Soluzione

```
DIM = 3
DIM_RIGA = DIM * 4

.data
mat1:    .word 4, -45, 15565, 6458, 4531, 124, -548, 2124, 31000
mat2:    .word 6, -5421, -547, -99, 4531, 1456, 4592, 118, 31999
indice:  .word 2
vet_out: .space DIM_RIGA

.text
.globl main
.ent main
main:    subu $sp, $sp, 4
        sw $ra, ($sp)
        la $a0, mat1
        la $a1, mat2
        la $a2, vet_out
        li $a3, DIM
        subu $sp, $sp, 4
        lw $t0, indice
        sw $t0, ($sp)
        jal ProceduraVariazione
        addu $sp, $sp, 4
        lw $ra, ($sp)
        addu $sp, $sp, 4
        jr $ra
        .end main
```

[1]-Soluzione [cont.]

.ent ProceduraVariazione

ProceduraVariazione:

```
sub $sp, $sp, 4
sw $fp, ($sp)          # salvo il valore corrente di $fp
move $fp, $sp          # uso $fp per avere un riferimento costante ai parametri ricevuti
subu $sp, $sp, 12
sw $ra, ($sp)          # salvo $ra perche' la procedura non e' leaf
sw $s0, 4($sp)
sw $s1, 8($sp)
```

```
lw $t0, 4($fp)          # recupero il parametro «indice» passato via stack
sll $s0, $a3, 2          # in $a3 ho il valore DIM, lo moltiplico x 4
mul $t1, $t0, $s0        # moltiplico l'indice per $s0
addu $a0, $a0, $t1       # indirizzo primo elemento della matrice 1 (riga)
```

```
mul $t1, $t0, 4
addu $a1, $a1, $t1       # indirizzo primo elemento della matrice 2 (colonna)
```

```
li $s1, 0                # contatore
```

```
ciclo1: lw $t2, ($a0)
lw $t3, ($a1)
subu $sp, $sp, 16
sw $a0, ($sp)            # salvo i registri $a0-$a3 nello stack
sw $a1, 4($sp)           # perche' CalcoloVariazione potrebbe modificarli
```

[1]-Soluzione [cont.]

```
sw $a2, 8($sp)
sw $a3, 12($sp)
move $a0, $t2      # ELEMENTO RIGA - VAL1
move $a1, $t3      # ELEMENTO COLONNA - VAL2
jal CalcoloVariazione

lw $a0, ($sp)
lw $a1, 4($sp)
lw $a2, 8($sp)
lw $a3, 12($sp)
addiu $sp, $sp, 16
sw $v0, ($a2)
addiu $a0, $a0, 4   # $a0, riga, per passare all'elemento successivo + 4
addu $a1, $a1, $s0  # $a1, colonna, per passare all'elemento successivo + Offset (DIM x 4)
addiu $a2, $a2, 4   # $a2 vettore uscita
addiu $s1, $s1, 1
bne $s1, $a3, ciclo1

lw $ra, ($sp)
lw $s0, 4($sp)
lw $s1, 8($sp)
lw $fp, 12($sp)
addu $sp, $sp, 16
jr $ra             # return
.end ProceduraVariazione
```

[1]-Soluzione [cont.]

```
.ent CalcoloVariazione
CalcoloVariazione:      # lavoro nell'ipotesi di non avere overflow
    sub $t0, $a1, $a0
    mul $t0, $t0, 100
    div $v0, $t0, $a0
    jr $ra              # return
.end CalcoloVariazione
```


Esercizio 2

- Si scriva una procedura **sostituisci** in grado di espandere una stringa precedentemente inizializzata sostituendo tutte le occorrenze del carattere % con un'altra stringa data. Siano date quindi le seguenti tre stringhe in memoria:
 - `str_orig`, corrispondente al testo compresso da espandere
 - `str_sost`, contenente il testo da sostituire in `str_orig` al posto di %
 - `str_new`, che conterrà la stringa espansa (si supponga che abbia dimensione sufficiente a contenerla).
- Di seguito un esempio di funzionamento:
 - Stringa originale: "% nella citta' dolente, % nell'eterno dolore, % tra la perduta gente"
 - Stringa da sostituire: "per me si va"
 - Risultato: "per me si va nella citta' dolente, per me si va nell'eterno dolore, per me si va tra la perduta gente"

Esercizio 2 [cont.]

- La procedura riceve gli indirizzi delle 3 stringhe attraverso i registri \$a0, \$a1 e \$a2, e restituisce la lunghezza della stringa finale attraverso \$v0.
- Le stringhe sono terminate dal valore ASCII 0x00.
- Di seguito un esempio di programma chiamante:

```
str_orig:      .data
perduta gente" .ascii " % nella citta' dolente, % nell'eterno dolore, % tra la
str_sost:      .asciiz "per me si va"
str_new:       .space 200

               .text
               .globl main
               .ent main
main:          [...]
               la $a0, str_orig
               la $a1, str_sost
               la $a2, str_new
               jal sostituisci
               [...]
               .end main
```

[2]-Soluzione

```

                                .data
str_orig:                      .asciiiz "% nella citta' dolente, % nell'eterno dolore, % tra la perduta gente"
str_sost:                      .asciiiz "per me si va"
str_new:                       .space 200

                                .text
                                .globl main
                                .ent main
main:                          subu $sp, $sp, 4
                               sw $ra, ($sp)

                               la $a0, str_orig
                               la $a1, str_sost
                               la $a2, str_new
                               jal sostituisci

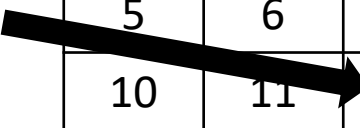
                               la $a0, str_new
                               li $v0, 4
                               syscall
                               lw $ra, ($sp)
                               addiu $sp, $sp, 4
                               jr $ra
                               .end main
```

[2]-Soluzione [cont.]

```
.ent sostituisci
sostituisci:    subu $sp, $sp, 4
               sw $a2, ($sp)          # salvataggio indirizzo str_new (per calcolo lunghezza)
ciclo1:        lbu $t0, ($a0)
               beqz $t0, fine          # controllo fine stringa
               bne $t0, '%', copia    # controllo carattere da sostituire
               move $t1, $a1          # $a1 indirizzo stringa sostituzione
ciclo2:        lbu $t2, ($t1)
               beqz $t2, next
               sb $t2, ($a2)
               addiu $t1, 1
               addiu $a2, 1
               j ciclo2
copia:         sb $t0, ($a2)          # copia caratteri stringa
               addiu $a2, 1
next:          addiu $a0, 1
               j ciclo1
fine:          sb, $0, ($a2)
               lw $t0, ($sp)          # calcolo lunghezza della nuova stringa
               addiu $sp, $sp, 4
               subu $v0, $a2, $t0
               jr $ra
.end sostituisci
```

Esercizio 3

- Sia data una matrice di byte, contenente numeri senza segno.
- Si scriva una procedura **contaVicini** in grado di calcolare (e restituire come valore di ritorno) la somma dei valori contenuti nelle celle adiacenti ad una determinata cella.
- La procedura **contaVicini** riceve i seguenti parametri:
 - indirizzo della matrice
 - numero progressivo della cella X, così come indicato nell'esempio a fianco
 - numero di righe della matrice
 - numero di colonne della matrice.
- La procedura deve essere conforme allo standard per quanto riguarda passaggio di parametri, valore di ritorno e registri da preservare.



0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

Esercizio 3 [cont.]

- Di seguito un esempio di programma chiamante:

RIGHE = 4

COLONNE = 5

.data

matrice: .byte 0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23,
9, 24, 8, 25, 43, 62

.text

.globl main

.ent main

main: [...]

la \$a0, matrice

li \$a1, 12 ~~#si parte da 0~~

li \$a2, RIGHE

li \$a3, COLONNE

jal contaVicini

[...]

.end main

0	1	3	6	2
7	13	20	12	21
11	22	10	23	9
24	8	25	43	62

il valore restituito è 166, pari a
 $13 + 20 + 12 + 22 + 23 + 8 + 25 + 43$

[3]-Soluzione

RIGHE = 4

COLONNE = 5

.data

matrice: .byte 0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9, 24, 8, 25, 43, 62

.text

.globl main

.ent main

main: subu \$sp, \$sp, 4

sw \$ra, (\$sp)

la \$a0, matrice

li \$a1, 12

li \$a2, RIGHE

li \$a3, COLONNE

jal contaVicini

lw \$ra, (\$sp)

addu \$sp, \$sp, 4

jr \$ra

.end main

[3]-Soluzione [cont.]

```

contaVicini:      .ent contaVicini
                  divu $a1, $a3          # $a1 la casella, $a3 numero Colonne
                  mflo $t0, # indice riga      # 2, secondo numeri esempio
                  mfhi $t1, # indice colonna   # 2, secondo numeri esempio
                  move $v0, $0              # somma delle celle vicine
                  # in $t2=RigaSopra,in $t3=RigaSotto,in $t4=ColonnaSx,in $t5=ColonnaDx
                  addi $t2, $t0, -1          # indice riga sopra
                  bne $t2, -1, indiceRigaSotto
                  move $t2, $0
indiceRigaSotto:   addi $t3, $t0, 1
                  bne $t3, $a2, indiceColonnaASinistra
                  sub $t3, $a2, 1

indiceColonnaASinistra: addi $t4, $t1, -1
                  bne $t4, -1, indiceColonnaADestra
                  move $t4, $0
indiceColonnaADestra: addi $t5, $t1, 1
                  bne $t5, $a3, indiciCelle
                  sub $t5, $a3, 1
```


[3]-Soluzione [cont.]

```
indiciCelle:      mul $t1, $t2, $a3
                  add $t0, $t1, $t4      # indice dell'elemento a sinistra nella riga sopra
                  add $t1, $t1, $t5      # indice dell'elemento a destra nella riga sopra
                  mul $t2, $t3, $a3
                  add $t2, $t2, $t4      # indice dell'elemento a sinistra nella riga sotto
                  add $t0, $t0, $a0      # somma l'indirizzo iniziale della matrice
                  add $t1, $t1, $a0
                  add $t2, $t2, $a0
                  add $a1, $a1, $a0

cicloEsterno:     move $t3, $t0
cicloInterno:     beq $t3, $a1, saltaElemento
                  lb $t4, ($t3)
                  add $v0, $v0, $t4
saltaElemento:    add $t3, $t3, 1
                  bleu $t3, $t1, cicloInterno
                  add $t0, $t0, $a3
                  add $t1, $t1, $a3
                  bleu $t0, $t2, cicloEsterno
                  jr $ra
                  .end contaVicini
```

Esercizio 4

- Il gioco della vita sviluppato dal matematico John Conway si svolge su una matrice bidimensionale.
- Le celle della matrice possono essere vive o morte.
- I vicini di una cella sono le celle ad essa adiacenti.
- La matrice evolve secondo le seguenti regole:
 - una cella con meno di due vicini vivi muore (isolamento)
 - una cella con due o tre vicini vivi sopravvive alla generazione successiva
 - una cella con più di tre vicini vivi muore (sovrappopolazione)
 - una cella morta con tre vicini vivi diventa viva (riproduzione).
- L'evoluzione avviene contemporaneamente per tutte le celle.

Esercizio 4 [cont.]

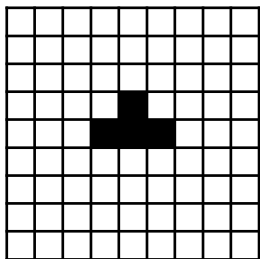
- Si scriva un programma in MIPS in grado di giocare al gioco della vita.
- Il programma principale esegue un ciclo di N iterazioni; ad ogni iterazione chiama la procedura **evoluzione** che determina il nuovo stato delle celle nella matrice.
- La procedura **evoluzione** riceve i seguenti parametri:
 - indirizzo di una matrice di byte, le cui celle hanno solo due valori: vivo (1) e morto (0)
 - indirizzo di una seconda matrice di byte non inizializzata di pari dimensioni
 - numero di righe delle due matrici
 - numero di colonne delle due matrici.

Esercizio 4 [cont.]

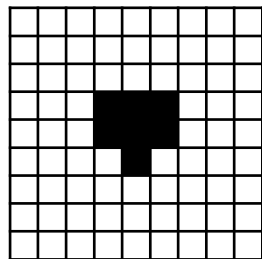
- La procedura **evoluzione** effettua un ciclo su tutte le celle della prima matrice:
 - per ogni cella, chiama la procedura **contaVicini**, implementata nell'esercizio precedente, per contare il numero di vicini
 - in base allo stato della cella e al suo numero di vicini, setta lo stato futuro della corrispondente cella nella seconda matrice.
- Al termine del ciclo, la procedura **evoluzione** chiama la procedura **stampaMatrice** che visualizza a video la seconda matrice, passando i seguenti parametri:
 - indirizzo della matrice
 - numero di righe della matrice
 - numero di colonne della matrice.
- Tutte le procedure devono essere conformi allo standard.

Esempio

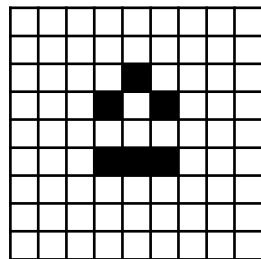
generazione 0



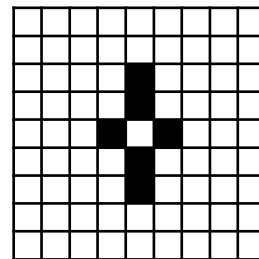
generazione 1



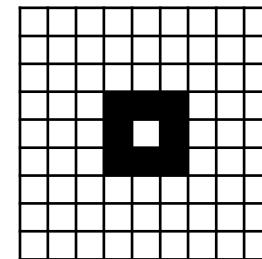
generazione 2



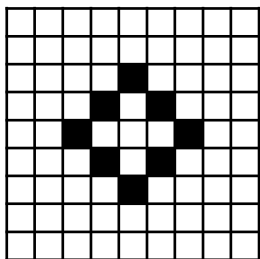
generazione 3



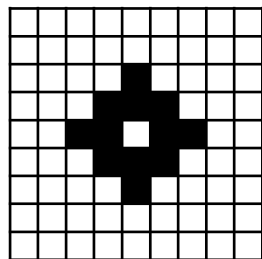
generazione 4



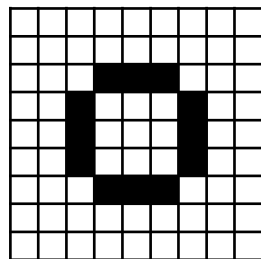
generazione 5



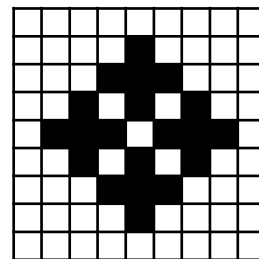
generazione 6



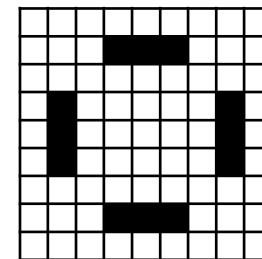
generazione 7



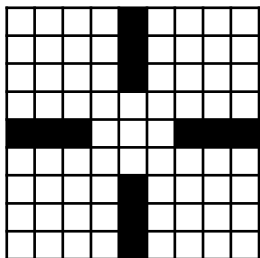
generazione 8



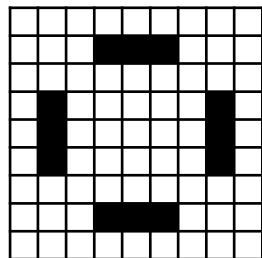
generazione 9



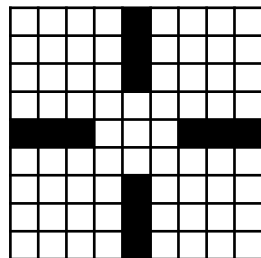
generazione 10



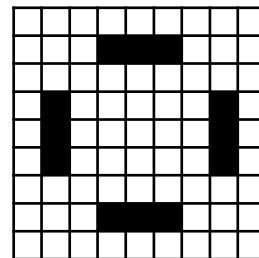
generazione 11



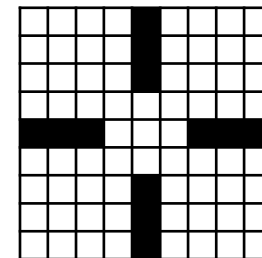
generazione 12



generazione 13



generazione 14



Per testare altre configurazioni: <https://playgameoflife.com/>

[4]-Soluzione

```
.data
RIGHE = 9
COLONNE = 9
DIM = RIGHE * COLONNE
ITERAZIONI = 14
matrice1: .byte 0, 0, 0, 0, 0, 0, 0, 0, 0
          .byte 0, 0, 0, 0, 0, 0, 0, 0, 0
          .byte 0, 0, 0, 0, 0, 0, 0, 0, 0
          .byte 0, 0, 0, 0, 1, 0, 0, 0, 0
          .byte 0, 0, 0, 1, 1, 1, 0, 0, 0
          .byte 0, 0, 0, 0, 0, 0, 0, 0, 0
          .byte 0, 0, 0, 0, 0, 0, 0, 0, 0
          .byte 0, 0, 0, 0, 0, 0, 0, 0, 0
          .byte 0, 0, 0, 0, 0, 0, 0, 0, 0
matrice2: .space DIM
```

[4]-Soluzione [cont.]

```
.text
.globl main
.ent main
main:    subu $sp, $sp, 4
        sw $ra, ($sp)
        move $s0, $0
cicloMain: and $t0, $s0, 1      # passaggio parametri
        beqz $t0, pari        # nelle iterazioni dispari, la matrice iniziale e' matrice2
        la $a0, matrice2
        la $a1, matrice1
        b altriParametri
pari:    la $a0, matrice1      # nelle iterazioni pari, la matrice iniziale e' matrice1
        la $a1, matrice2
altriParametri: li $a2, RIGHE
        li $a3, COLONNE
        jal evoluzione
        addi $s0, $s0, 1
        bne $s0, ITERAZIONI, cicloMain
        lw $ra, ($sp)
        addu $sp, $sp, 4
        jr $ra
.end main
```

[4]-Soluzione [cont.]

```
.ent evoluzione
evoluzione: subu $sp, $sp, 36
            sw $ra, ($sp)
            sw $s0, 4($sp)
            sw $s1, 8($sp)
            sw $s2, 12($sp)
            sw $s3, 16($sp)
            sw $s4, 20($sp)
            sw $s5, 24($sp)
            sw $s6, 28($sp)
            sw $s7, 32($sp)
            move $s0, $a0    # salvo gli argomenti perche' le procedure leaf potrebbero cambiarli
            move $s1, $a1
            move $s2, $a2
            move $s3, $a3
            move $s4, $0
            mul $s5, $a2, $a3    # numero di elementi nella matrice
            move $s6, $s0        # elemento corrente nella matrice corrente
            move $s7, $s1        # elemento corrente nella matrice futura
```


[4]-Soluzione [cont.]

```
ciclo:      move $a0, $s0
            move $a1, $s4
            move $a2, $s2
            move $a3, $s3
            jal contaVicini
            lb $t0, ($s6)
            beqz $t0, cellaMorta
            beq $v0, 2, cellaFuturaViva
            beq $v0, 3, cellaFuturaViva
cellaFuturaMorta: li $t0, 0
                b next
cellaMorta:     bne $v0, 3, cellaFuturaMorta
cellaFuturaViva: li $t0, 1
next:          sb $t0, ($s7)
                addi $s4, $s4, 1
                addi $s6, $s6, 1
                addi $s7, $s7, 1
                bne $s4, $s5, ciclo
                move $a0, $s1
                move $a1, $s2
                move $a2, $s3
                jal stampaMatrice
```

[4]-Soluzione [cont.]

```
lw $ra, ($sp)
lw $s0, 4($sp)
lw $s1, 8($sp)
lw $s2, 12($sp)
lw $s3, 16($sp)
lw $s4, 20($sp)
lw $s5, 24($sp)
lw $s6, 28($sp)
lw $s7, 32($sp)
addu $sp, $sp, 36
jr $ra
.end Evoluzione
```

Soluzione [cont.]

```

                                .ent stampaMatrice
stampaMatrice:                 li $v0, 11
                                move $t0, $a0
                                move $t1, $0          # indice riga
cicloRighe:                    move $t2, $0          # indice colonna
cicloColonne:                  lb $t3, ($t0)
                                li $a0, ' '
                                beqz $t3, stampaCarattere
                                li $a0, '*'
stampaCarattere:               syscall
                                addi $t0, $t0, 1
                                addi $t2, $t2, 1
                                bne $t2, $a2, cicloColonne
                                li $a0, '\n'
                                syscall
                                addi $t1, $t1, 1
                                bne $t1, $a1, cicloRighe
                                syscall    # stampa un altro new line
                                jr $ra
                                .end stampaMatrice
```