

# Calcolatori Elettronici

## Esercitazione 8

M. Sonza Reorda – M. Monetti

M. Rebaudengo – R. Ferrero

L. Sterpone – M. Grosso

Politecnico di Torino

Dipartimento di Automatica e Informatica

# Esercizio 1

- Il costo di un parcheggio è pari a X Euro per ogni periodo di Y minuti. Per eventuali minuti di un periodo non completo sono addebitati comunque X Euro.
- Esempio:
  - X: 1 Euro
  - Y: 40 minuti
  - Orario di ingresso: 12.47
  - Orario di uscita: 18.14
  - Il tempo di permanenza corrisponde a 8 periodi interi più 7 minuti. Il costo del parcheggio è 9 Euro.
- Si scriva una procedura **costoParcheggio** in linguaggio assembly MIPS32 in grado di calcolare il costo per il parcheggio.
- Gli orari di ingresso e di uscita sono memorizzati ciascuno in un vettore di 2 byte: il primo indica l'ora e il secondo i minuti. La procedura costoParcheggio riceve l'indirizzo dei due vettori tramite i registri \$a0 e \$a1, X e Y mediante \$a2 e \$a3, e restituisce il costo del parcheggio attraverso \$v0.
- Si assuma che gli orari siano sempre consecutivi e appartenenti alla stessa giornata.

# Esercizio 1 [cont.]

- Di seguito un esempio di programma chiamante:

```

                                .data
ora_in:                        .byte 12, 47
ora_out:                       .byte 18, 14
X:                             .byte 1
Y:                             .byte 40

                                .text
                                .globl main
main:                          [...]
                                la $a0, ora_in      # indirizzo di ora_in
                                la $a1, ora_out      # indirizzo di ora_out
                                lbu $a2, X
                                lbu $a3, Y
                                jal costoParcheggio
                                [...]

```

# Soluzione

```
.data
ora_in:      .byte 12, 47
ora_out:     .byte 18, 14
X:           .byte 1
Y:           .byte 40

.text
.globl main
main:        sub $sp, 4
             sw $ra, ($sp)
             la $a0, ora_in      # indirizzo di ora_in
             la $a1, ora_out     # indirizzo di ora_out
             lbu $a2, X
             lbu $a3, Y
             jal costoParcheggio
             lw $ra, ($sp)
             jr $ra
```

# Soluzione [cont.]

```
.ent costoParcheggio
costoParcheggio: lbu $t0, 0($a1)      # data la dimensione dei dati non puo'
                                     # verificarsi overflow

lbu $t1, 0($a0)
subu $t0, $t0, $t1
li $t1, 60
multu $t0, $t1
lbu $t0, 1($a1)
lbu $t1, 1($a0)
subu $t0, $t0, $t1
mflo $t1
addu $t0, $t0, $t1

divu $t0, $a3
mflo $t0
mfhi $t1
beqz $t1, next
addiu $t0, $t0, 1
next: multu $t0, $a2
mflo $v0
jr $ra          # return
.end costoParcheggio
```

# Esercizio 2

- Si abbia un vettore contenente alcuni interi rappresentanti anni passati ( $0 \div 2018$ ). Si scriva una procedura **bisestile** 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
}
```

## Esercizio 2 [cont.]

- La procedura deve ricevere come input:
  - *tramite il registro \$a0*, l'indirizzo di un vettore di *word* contenente gli anni da valutare
  - *tramite il registro \$a1*, l'indirizzo 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 \$a2*, la lunghezza di tali vettori.
- Esempio:
  - anni: 1945, 2008, 1800, 2006, 1748, 1600
  - risultato: 0, 1, 0, 0, 1, 1
  - lunghezza: 6

# Soluzione

LUNG = 6

```
anni:      .data
           .word 1945, 2008, 1800, 2006, 1748, 1600
ris:       .space LUNG
```

```
           .text
           .globl main
main:      sub $sp, 4
           sw $ra, ($sp)
```

```
           la $a0, anni
           la $a1, ris
           li $a2, LUNG
           jal bisestile
```

```
           li $t1, LUNG
           la $t2, ris
ciclo_stampa: li $v0, 1
           lbu $a0, ($t2)
           syscall
           addiu $t2, $t2, 1
           subu $t1, $t1, 1
           bnez $t1, ciclo_stampa

           lw $ra, ($sp)
           jr $ra
```



# Soluzione [cont.]

```
.ent bisestile

bisestile:
ciclo:      sb $0, ($a1)          # ipotesi iniziale: non bisestile
            lw $t0, ($a0)
            li $t1, 100
            divu $t0, $t1
            mfhi $t1
            bnez $t1, no_100
            li $t1, 400
            divu $t0, $t1
            mfhi $t1
            bnez $t1, next
            li $t1, 1
            sb $t1, ($a1)
            b next
no_100:      li $t1, 4
            divu $t0, $t1
            mfhi $t1
            bnez $t1, next
            li $t1, 1
            sb $t1, ($a1)
next:       addiu $a0, 4
            addiu $a1, 1
            subu $a2, 1
            bnez $a2, ciclo
            jr $ra                # return
.end bisestile
```

# Esercizio 3

- Si scriva una procedura `calcola_sconto` in MIPS in grado di calcolare il prezzo scontato degli articoli venduti in un negozio e salvarlo nel corrispondente elemento del vettore scontati. La procedura riceve i seguenti parametri:
  1. indirizzo del vettore prezzi, contenente i prezzi di ciascun articoli venduti in un negozio
  2. indirizzo del vettore scontati, inizialmente di contenuto indeterminato
  3. numero di elementi contenuti nei due vettori
  4. ammontare dello sconto percentuale da applicare
  5. eventuale arrotondamento. Se il valore del parametro è 1, si deve eseguire un arrotondamento alla cifra superiore qualora la parte decimale del prezzo scontato sia maggiore o uguale a 0,5. Se il valore del parametro è 0, il prezzo scontato è sempre arrotondato alla cifra inferiore.

## Esercizio 3 [cont.]

- La procedura restituisce l'ammontare totale delle riduzioni.
- La procedura deve essere conforme allo standard per quanto riguarda il passaggio dei parametri in input, del valore di ritorno e dei registri da preservare.
- Esempio: prezzi = {39, 1880, 2394, 1000, 1590}, sconto = 30.
- Se arrotondamento = 1, scontati = {27, 1316, 1676, 700, 1113} e il valore restituito dalla procedura è 2071.
- Se arrotondamento = 0, scontati = {27, 1316, 1675, 700, 1113} e il valore restituito dalla procedura è 2072.
- La slide successiva riporta un esempio di programma chiamante.

## Esercizio 3 [cont.]

```
NUM = 5
DIM = NUM * 4
SCONTO = 30
ARROTONDA = 1
    .data
prezzi: .word 39, 1880, 2394, 1000, 1590
scontati: .space DIM
    .text
    .globl main
    .ent main
main:    [...]
        la $a0, prezzi
        la $a1, scontati
        li $a2, NUM
        li $a3, SCONTO
        li $t0, ARROTONDA
        subu $sp, 4
        sw $t0, ($sp)
        jal calcola_sconto
        [...]
    .end main
```

# Soluzione

```
NUM = 5
DIM = NUM * 4
SCONTO = 30
ARROTONDA = 1

        .data
prezzi: .word 39, 1880, 2394, 1000, 1590
scontati: .space DIM
totSconto: .space 4
        .text
        .globl main
        .ent main
main:    subu $sp, $sp, 4
        sw $ra, ($sp)
        la $a0, prezzi
        la $a1, scontati
        li $a2, NUM
        li $a3, SCONTO
        li $t0, ARROTONDA
        subu $sp, 4
        sw $t0, ($sp)
        jal calcola_sconto
        sw $v0, totSconto
        lw $ra, 4($sp)
        addu $sp, $sp, 8
        jr $ra
        .end main
```

# Soluzione [cont.]

```
.ent calcola_sconto
calcola_sconto: subu $fp, $sp, 4
                 move $t0, $a0                # prezzi
                 move $t1, $a1                # scontati
                 move $t2, $0                 # indice ciclo
                 li $t5, 100                  # costante
                 sub $t3, $t5, $a3            # percentuale del prezzo dopo lo sconto
                 lw $t4, 4($fp)               # arrotondamento
                 move $v0, $0                 # sconto totale
ciclo:           lw $t6, ($t0)
                 mul $t7, $t6, $t3
                 div $t7, $t5
                 mflo $t7
                 beqz $t4, noArrotondamento
                 # arrotonda il prezzo
                 mfhi $t8
                 blt $t8, 50, noArrotondamento
                 addiu $t7, $t7, 1
noArrotondamento: sw $t7, ($t1)
                 subu $t8, $t6, $t7
                 addu $v0, $v0, $t8
                 addiu $t0, $t0, 4
                 addiu $t1, $t1, 4
                 addiu $t2, $t2, 1
                 bne $t2, $a2, ciclo
                 jr $ra
                 .end calcola_sconto
```

# Esercizio 4

La distanza di Hamming tra due stringhe di ugual lunghezza è pari al numero di posizioni nelle quali i simboli corrispondenti sono diversi. In altri termini, la distanza di Hamming misura il numero di sostituzioni necessarie per convertire una stringa nell'altra, o il numero di modifiche necessarie per trasformare una stringa nell'altra. Ad esempio, si consideri la distanza di Hamming binaria tra i seguenti due interi:

11011101    11001001

Il risultato in questo caso è 2.

Si scriva una procedura `CalcolaDistanzaH` in linguaggio assembly MIPS32 che calcoli la distanza di Hamming binaria tra gli elementi di indice corrispondente di due vettori di *word* di lunghezza DIM (dichiarato come costante).

Esempio (valori in decimale e binario):

vet1	vet2	risultato
56 (0000 0000 0011 1000)	1 (0000 0000 0000 0001)	4
12 (0000 0000 0000 1100)	0 (0000 0000 0000 0000)	2
98 (0000 0000 0110 0010)	245 (0000 0000 1111 0101)	5
129 (0000 0000 1000 0001)	129 (0000 0000 1000 0001)	0
58 (0000 0000 0011 1010)	12 (0000 0000 0000 1100)	4

# Esercizio 4 [cont.]

- La procedura riceve tramite registri l'indirizzo dei due vettori di dati, del vettore risultato e la loro dimensione. Di seguito un esempio di programma chiamante.

DIM = 5

```
        .data
vet1:    .word    56, 12, 98, 129, 58
vet2:    .word    1, 0, 245, 129, 12
risultato: .space DIM
        .text
        .globl main
        .ent main
main:    [...]
        la $a0, vet1
        la $a1, vet2
        la $a2, risultato
        li $a3, DIM
        jal CalcolaDistanzaH
        [...]
        .end main
```



# Soluzione

DIM = 5

```
.data
vet1:      .word    56, 12, 98, 129, 58
vet2:      .word    1, 0, 245, 129, 12
risultato: .word    space 20

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

           la $a0, vet1
           la $a1, vet2
           la $a2, risultato
           li $a3, DIM

           jal  calcola_distanzaH

           lw $ra, ($sp)

           addiu $sp, $sp, 4
           jr $ra
           .end main
```

# Soluzione [cont.]

```
.ent calcola_distanzaH
calcola_distanzaH:  # $a0= ind. vet1  $a1= ind. vet2    $a2= ind. risultato  $a3=DIM
                   li  $t0, 0          # $t0 contatore Cicli
ciclo:              beq  $t0, $a3, fine_ciclo

calcoloH:           lw  $t7, ($a0)
                   lw  $t8, ($a1)
                   xor  $t2, $t7, $t8
                   and  $t3, $0, $0    # azzeramento risultato
                   and  $t4, $0, $0    # azzeramento indice
                   li  $t5, 1          # mask per lettura bit a 1
cicloH:             and  $t6, $t2, $t5
                   beq  $t6, 0, nextH
                   addiu $t3, $t3, 1
nextH:              sll  $t5, $t5, 1
                   addiu $t4, $t4, 1
                   bne  $t4, 32, cicloH

                   # in $t3 il risultato
                   sb   $t3, ($a2)
                   addiu $t0, $t0, 1
                   addiu $a0, $a0, 4
                   addiu $a1, $a1, 4
                   addiu $a2, $a2, 4
                   j  ciclo
fine_ciclo:         jr  $ra
                   .end calcola_distanzaH
```

# Esercizio 5

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 5: implementazione

- 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.

# 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
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
```

# Soluzione [cont.]

Variazione:

```
.ent variazione
```

```
subu $sp, $sp, 4      # si lavora nell'ipotesi di non avere overflow
sw $ra, ($sp)
move $fp, $sp
```

```
lw $t0, 4($fp)
mul $t1, $t0, DIM_RIGA
addu $a0, $a0, $t1     # indirizzo primo elemento RIGA della prima matrice
mul $t1, $t0, 4
addu $a1, $a1, $t1     # indirizzo primo elemento COLONNA della seconda matrice
li $t1, 0              # contatore
```

```
ciclo1: lw $t2, ($a0)
lw $t3, ($a1)
subu $sp, $sp, 8
sw $a0, ($sp)          # Salvataggio $a0 e $a1
sw $a1, 4($sp)
```

```
move $a0, $t2          # ELEMENTO RIGA - VAL1
move $a1, $t3          # ELEMENTO COLONNA - VAL2
```

```
jal CalcoloVariazione  # a titolo di esempio si usa una seconda procedura per il calcolo
                        # ma non e' strettamente necessario
```

```
lw $a0, ($sp)
lw $a1, 4($sp)
addiu $sp, 8
```

# Soluzione [cont.]

```
sw $v0, ($a2)
addiu $a0, $a0, 4          # RIGA ELEMETO + 1 , offeset = 4
addiu $a1, $a1, DIM_RIGA  # COLONNA ELEMETO + 1 , offeset = DIM_RIGA

addiu $a2, $a2, 4
addiu $t1, $t1, 1

bne $t1, $a3, ciclo1

lw $ra, ($sp)
addu $sp, $sp, 4
jr $ra                    # return
.end variazione

.ent CalcoloVariazione
CalcoloVariazione:
    sub $t0, $a1, $a0
    mul $t0, $t0, 100
    div $v0, $t0, $a0
    jr $ra                # return

.end CalcoloVariazione
```

# Esercizio 6

- 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 la 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 6 [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:

```

                                .data
str_orig:                      .ascii " % nella citta' dolente, % nell'eterno dolore, % tra la
perduta gente %"
str_sost:                      .ascii "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
                                [...]
```

# Soluzione

```

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

                                .text
                                .globl main
                                .ent main
main:        subu $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, 4
            jr $ra
            .end main
```

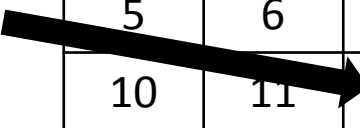
# Soluzione [cont.]

```
.ent sostituisci
sostituisci:  subu $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          # sostituzione
ciclo2:      lbu $t2, ($t1)
              beqz $t2, next          # fine sostituzione
              sb $t2, ($a2)
              addiu $t1, 1
              addiu $a2, 1
              j ciclo2
copia:       sb $t0, ($a2)            # copia (ASIS) 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, 4
              subu $v0, $a2, $t0
              jr $ra
              .end sostituisci
```

# Esercizio 7

- 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 7 [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  
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  
 $13 + 20 + 12 + 22 + 23 + 8 + 25 + 43$

# 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, 19
    li $a2, RIGHE
    li $a3, COLONNE
    jal contaVicini
    lw $ra, ($sp)
    addu $sp, $sp, 4
    jr $ra
.end main

.ent contaVicini
contaVicini:
    divu $a1, $a3
    mflo $t0, # indice riga
    mfhi $t1, # indice colonna
    move $v0, $0 # somma delle celle vicine
    # $t2=RigaSopra, $t3=RigaSotto, $t4=ColonnaSX, $t50=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

```

# Soluzione [cont.]

```
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
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 8

- 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 8 [cont.]

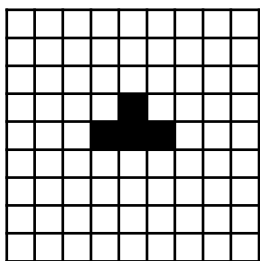
- Si scriva un programma in MIPS in grado di giocare al gioco della vita.
- Il programma principale esegue un ciclo di N iterazioni; ad ogni iterazione chiama la procedura **evoluzione** che determina il nuovo stato delle celle nella matrice.
- La procedura **evoluzione** riceve i seguenti parametri:
  - 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 8 [cont.]

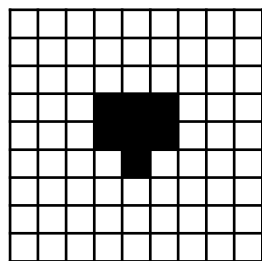
- La procedura **evoluzione** effettua un ciclo su tutte le celle della prima matrice:
  - per ogni cella, chiama la procedura **contaVicini**, implementata nell'esercizio precedente, per contare il numero di vicini
  - in base allo stato della cella e al suo numero di vicini, setta lo stato futuro della 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

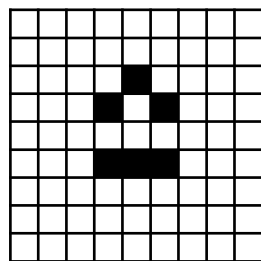
generazione 0



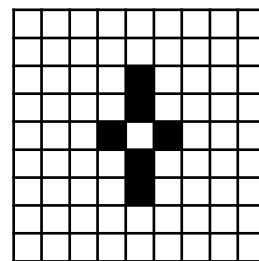
generazione 1



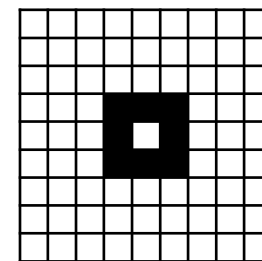
generazione 2



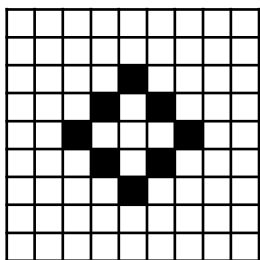
generazione 3



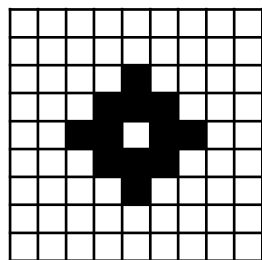
generazione 4



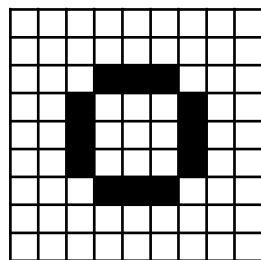
generazione 5



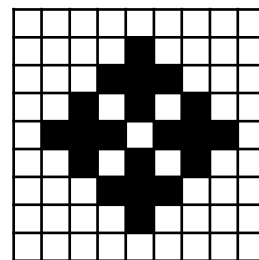
generazione 6



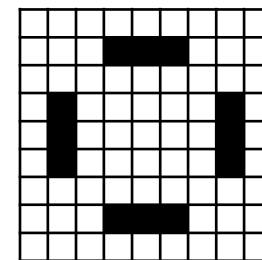
generazione 7



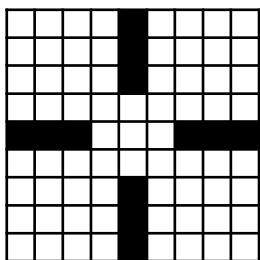
generazione 8



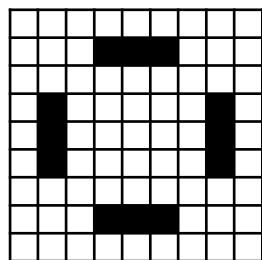
generazione 9



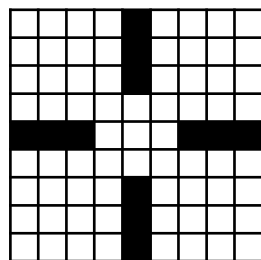
generazione 10



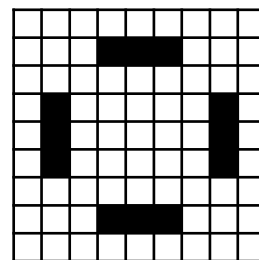
generazione 11



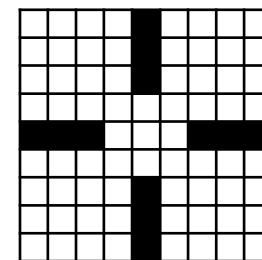
generazione 12



generazione 13



generazione 14



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

# 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
```

# 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
```

# 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
ciclo:      move $a0, $s0
            move $a1, $s4
            move $a2, $s2
            move $a3, $s3
            jal contaVicini
```

# Soluzione [cont.]

```

                                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
                                lw $ra, ($sp)
                                lw $s0, 4($sp)
                                lw $s1, 8($sp)
                                lw $s2, 12($sp)
                                lw $s3, 16($sp)
                                lw $s4, 20($sp)
```

# Soluzione [cont.]

```
lw $s5, 24($sp)
lw $s6, 28($sp)
lw $s7, 32($sp)
addu $sp, $sp, 36
jr $ra
.end evoluzione

.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
```