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.]

```
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
```

Esercizio 2

- Si abbia un vettore contenente alcuni interi rappresentanti anni passati (0 ÷ 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
                .data
anni:           .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.]

```
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
```

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.]

```
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
```

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.]

```
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, 1
                      j ciclo
fine_ciclo:           jr $ra
```