

Calcolatori Elettronici

Esercitazione 5

M. Sonza Reorda – M. Monetti

M. Rebaudengo – R. Ferrero

L. Sterpone – M. Grosso

Politecnico di Torino

Dipartimento di Automatica e Informatica

Obiettivi

- Stack
- Algoritmi

Esercizio 1

- La *system call* 1 scrive in output un numero intero con segno, compreso fra -2^{31} e $2^{31} - 1$.
- Volendo stampare un intero *unsigned* su 32 bit, non è possibile utilizzare tale system call
 - Che valore è visualizzato se il numero è un intero senza segno compreso fra 2^{31} e $2^{32} - 1$?
- Data una variabile di tipo *word* in memoria inizializzata a 3141592653, si realizzi un programma che ne stampi il valore in output.
- Il programma deve scrivere le singole cifre tramite la system call 11.

Implementazione

- Si utilizza un algoritmo in due passi:
 1. Scomposizione del numero nelle sue cifre tramite divisioni successive per 10, salvando i resti e ripetendo l'operazione sul quoziente sino a che questo è diverso da zero
 2. Visualizzazione dei resti in ordine inverso a quello di generazione, utilizzando lo *stack*
- N.B.: le cifre devono essere convertite in caratteri ASCII prima della stampa.

Stack

- Lo *stack* è una struttura di memoria con accesso ai dati di tipo LIFO
- Per inserire un dato si effettua una *push*:
 subu \$sp, \$sp, 4
 sw \$t0, (\$sp)
- Per prelevare un dato si effettua una *pop*:
 lw \$t0, (\$sp)
 addu \$sp, \$sp, 4
- N.B.: Si raccomanda di effettuare sempre operazioni *push/pop* di intere *word*. Inoltre, il valore dello *stack pointer* alla fine del programma o della procedura deve essere lo stesso che all'inizio.

Soluzione

```
        .data
hugeNumber: .word 3141592653
        .text
        .globl main
        .ent main
main:    lw $a0, hugeNumber
        li $v0, 1
        syscall      #stampa -1153374643
        li $t0, 0    # numero di cifre da stampare
        li $t1, 10   # costante
        lw $t2, hugeNumber
ciclo1:  divu $t2, $t1
        mfhi $t2
        addu $t0, $t0, 1
        subu $sp, $sp, 4    #push
        sw $t2, ($sp)
```

Soluzione [cont.]

```
mflo $t2
bnez $t2, ciclo1
fine:  li $v0, 11
      li $a0, '\n'
      syscall
ciclo2: lw $a0, ($sp)          #pop
      addu $sp, $sp, 4
      addu $a0, $a0, '0'
      syscall
      subu $t0, $t0, 1
      bne $t0, $zero, ciclo2
      li $v0, 10
      syscall
      .end main
```

Esercizio 2

- Si scriva un programma che verifichi se la stringa introdotta dall'utente è palindroma.
- La lettura dell'input avviene un carattere alla volta tramite la system call 12 e termina quando l'utente introduce '\n'.
- Siccome il numero di caratteri introdotto dall'utente non è noto a priori, si utilizzi lo *stack* per memorizzarli invece di allocare una quantità di memoria costante.

Soluzione

```
.data
input: .ascii "Introduci una stringa: "
outputVuoto: .ascii "non hai inserito nessun carattere"
outputNoPalindromo: .ascii "La stringa non e' palindroma"
outputPalindromo: .ascii "La stringa e' palindroma"

.text
.globl main
.ent main

main:
    move $t0, $sp #posizione iniziale dello stack
    move $s0, $sp  #posizione iniziale dello stack
    li $t1, 0      #numero di caratteri introdotti dall'utente
    la $a0, input
    li $v0, 4
    syscall
```

Soluzione [cont.]

```
cicloLettura:  li $v0, 12
                 syscall
                 beq $v0, '\n', fineLettura
                 addi $t1, $t1, 1
                 subu $sp, $sp, 4
                 sw $v0, ($sp)
                 b cicloLettura
fineLettura:   beq $t1, 0, noInput
cicloControllo: subu $t0, $t0, 4
                 lw $t2, ($t0)
                 lw $t3, ($sp)
                 addu $sp, $sp, 4
                 bne $t2, $t3, noPalindromo
                 addi $t1, $t1, -2
                 bgt $t1, 0, cicloControllo
                 la $a0, outputPalindromo
                 b stampa
```

Soluzione [cont.]

noPalindromo:

la \$a0, outputNoPalindromo

b stampa

noInput:

la \$a0, outputVuoto

stampa:

li \$v0, 4

syscall

move \$sp, \$s0 # ripristino lo stack pointer

li \$v0, 10

syscall

.end main

Esercizio 3

- Si scriva un programma in linguaggio MIPS che dica se un'equazione di secondo grado nella forma

$$ax^2 + bx + c = 0$$

ha o meno soluzioni reali.

- a , b e c sono interi con segno introdotti dall'utente.
- Per i salti condizionati, si utilizzino soltanto le istruzioni `slt`, `beq` e `bne`.
- Sia lecito assumere che i calcoli non diano *overflow*.

Soluzione

```
.data
msgInput: .ascii "Inserisci i valori di A, b e C (separati da invio): "
msg_due_sol: .ascii "Esistono due soluzioni reali"
msg_no_sol: .ascii "Non esistono soluzioni reali"
msg_sol_coinc: .ascii "Due soluzioni coincidenti"

.text
.globl main
.ent main
main:  la $a0, msgInput
      li $v0, 4
      syscall
      li $v0, 5# legge A e lo salva in $t0
      syscall
      move $t0, $v0
      li $v0, 5# legge B e lo salva in $t1
      syscall
      move $t1, $v0
      li $v0, 5# legge C e lo salva in $t2
      syscall
```

Soluzione [cont.]

```
move $t2, $v0
mul $t3, $t1, $t1 # $t3 = B^2
mul $t4, $t0, $t2 # $t4 = AC
sll $t4, $t4, 2      # $t4 = 4AC
sub $t3, $t3, $t4    # $t3 = DISCRIMINANTE
beq $t3, 0, sol_coinc
slt $t3, $t3, 0
bne $t3, 0, no_sol
la $a0, msg_due_sol
b print
sol_coinc: la $a0, msg_sol_coinc
b print
no_sol: la $a0, msg_no_sol
print:  li $v0, 4
        syscall
        li $v0, 10
        syscall
        .end main
```

Esercizio 4

- Sia data una matrice quadrata di *word* memorizzata per righe (numero di righe pari a DIM, con DIM dichiarato come costante).
- Si scriva un programma che sia in grado di valutare se la matrice quadrata è simmetrica o diagonale. Il programma dovrà stampare a video un valore pari a:
 - 2 se la matrice è diagonale
 - 1 se la matrice è simmetrica
 - 0 se la matrice non è simmetrica.

Esercizio 4 [cont.]

- Si ricorda che in una matrice diagonale solamente i valori della diagonale principale possono essere diversi da 0, mentre una matrice simmetrica ha la proprietà di essere la trasposta di se stessa

- Esempio di matrice diagonale:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

- Esempio di matrice simmetrica:

$$\begin{bmatrix} 1 & 4 & 5 & 6 & 7 \\ 4 & 2 & 8 & 6 & 4 \\ 5 & 8 & 3 & 2 & 9 \\ 6 & 6 & 2 & 4 & 4 \\ 7 & 4 & 9 & 4 & 5 \end{bmatrix}$$

Soluzione

```
DIM = 5
```

```
    NEXT_COL = 4
```

```
    NEXT_ROW = 4*DIM
```

```
    NEXT_DIAG = 4*(DIM+1)
```

```
    .data
```

```
matrix: .word  1, 0, 0, 0, 0
```

```
        .word  0, 2, 0, 1, 0
```

```
        .word  0, 0, 3, 0, 0
```

```
        .word  0, 1, 0, 4, 0
```

```
        .word  0, 0, 0, 0, 5
```

```
    .text
```

```
    .globl main
```

```
    .ent main
```

```
main:  la $t0, matrix      # $t0 puntatore a elemento su diagonale
```

```
      li $t1, DIM-1      # $t1 contatore ciclo esterno
```

```
      li $a0, 2          # $a0 risultato (ipotesi iniziale: diagonale)
```

Soluzione [cont.]

```
ciclo1:  move $t2, $t1          # $t2 contatore ciclo interno
         move $t3, $t0          # $t3 puntatore a elementi su riga
         move $t4, $t0          # $t4 puntatore a elementi su colonna
ciclo2:  addiu $t3, $t3, NEXT_COL
         addiu $t4, $t4, NEXT_ROW
         lw $t6, ($t3)
         beq $t6, 0, next
         li $a0, 1              # non e' diagonale
next:    lw $t7, ($t4)
         bne $t6, $t7, no_simm  # se non e' simmetrica (ne' diagonale),
                                # esco dal ciclo
         sub $t2, $t2, 1
         bne $t2, 0, ciclo2

         addiu $t0, $t0, NEXT_DIAG
         sub $t1, $t1, 1
         bne $t1, 0, ciclo1
```

Soluzione [cont.]

```
b fine
```

```
no_simm: li $a0, 0
```

```
fine:    li $v0, 1  
        syscall
```

```
        li $v0, 10  
        syscall  
.end main
```