

Calcolatori Elettronici

Esercitazione 7

M. Sonza Reorda – M. Monetti

M. Rebaudengo – R. Ferrero

L. Sterpone – M. Grosso

Politecnico di Torino

Dipartimento di Automatica e Informatica

Obiettivi

- Chiamata a procedura
- Passaggio parametri tramite stack
- Salvataggio e ripristino del valore dei registri
- Procedure *leaf* e non *leaf*
- Ritorno dal main con `jr $ra`

Esercizio 1

- Si scriva una procedura *somma* che, dopo aver ricevuto tramite *stack* due valori interi *word*, azzeri i registri da $\$t0$ a $\$t4$ e da $\$s0$ a $\$s4$, salvi i valori ricevuti in $\$t2$ e $\$s2$ e ne restituisca la somma tramite $\$v0$.
- Quindi, si scriva un programma MIPS che
 - Inizializzi i registri da $\$t0$ a $\$t3$ con valori interi crescenti da 15 a 18, e quelli da $\$s0$ a $\$s3$ con i valori interi crescenti da 223 a 226
 - Richiami la procedura *somma* passando come parametri i valori contenuti in $\$t0$ e $\$s0$, e quindi ne salvi il risultato in $\$t4$
- Alla fine del programma, i valori nei registri da $\$t0$ a $\$t3$ e da $\$s0$ a $\$s4$ devono essere quelli iniziali: si usi opportunamente lo *stack* per salvarli durante la chiamata a procedura.

Esercizio 1: implementazione

- Si utilizzi lo *stack* per salvare provvisoriamente il valore dei registri quando necessario.
- Si ricorda che i registri di tipo $\$tx$ sono *caller-save*, ossia devono essere salvati e poi ripristinati dalla procedura chiamante, mentre i registri $\$sx$ sono *callee-save*, e devono essere salvati e poi ripristinati dalla procedura chiamata.
- Si disegni l'occupazione dello stack durante l'esecuzione della procedura prima di scrivere il codice.

Soluzione

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

       li $t0, 15
       addiu $t1, $t0, 1
       addiu $t2, $t1, 1
       addiu $t3, $t2, 1
       li $s0, 223
       addiu $s1, $s0, 1
       addiu $s2, $s1, 1
       addiu $s3, $s2, 1

       subu $sp, 16      # salvataggio registri $tx
       sw $t0, ($sp)
       sw $t1, 4($sp)
       sw $t2, 8($sp)
       sw $t3, 12($sp)

       subu $sp, 8       # passaggio parametri
       sw $t0, ($sp)
       sw $s0, 4($sp)
```

Soluzione [cont.]

```
jal somma  
move $t4, $v0  
addiu $sp, 8
```

```
lw $t0, ($sp)      # ripristino registri $tx  
lw $t1, 4($sp)  
lw $t2, 8($sp)  
lw $t3, 12($sp)  
addiu $sp, 16
```

```
lw $ra, ($sp)  
addiu $sp, 4  
jr $ra  
.end main
```

Soluzione [cont.]

somma:

```
.ent somma
sub $sp, $sp, 4
sw $fp, ($sp)
move $fp, $sp
```

```
subu $sp, 16
sw $s0, ($sp)
sw $s1, 4($sp)
sw $s2, 8($sp)
sw $s3, 12($sp)
```

```
move $t0, $0
move $t1, $0
move $t2, $0
move $t3, $0
move $s0, $0
move $s1, $0
move $s2, $0
move $s3, $0
lw $t2, 4($fp)
lw $s2, 8($fp)
addu $v0, $t2, $s2
```

```
lw $s0, ($sp)
lw $s1, 4($sp)
lw $s2, 8($sp)
lw $s3, 12($sp)
lw $fp, 16($sp)
addiu $sp, 20
jr $ra
.end somma
```

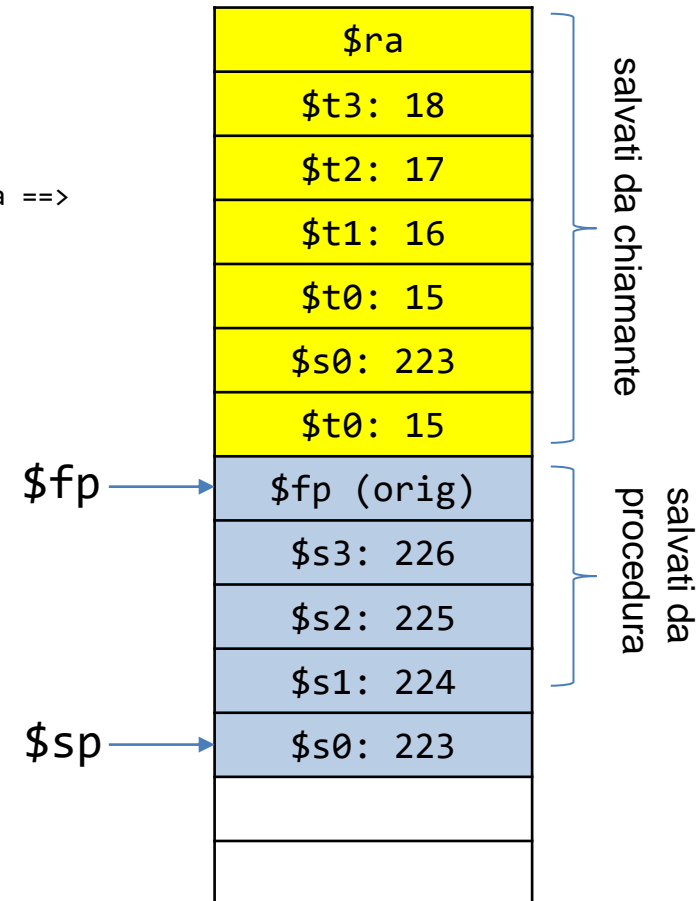
usare \$fp permette di avere un riferimento
costante ai parametri ricevuti dal main

salvataggio registri \$sx

situazione descritta dallo schema ==>

prelevamento dati da stack

ripristino registri \$sx



Esercizio 2

- Si consideri una sequenza di numeri naturali in cui, scelto il primo numero della sequenza c_0 , gli elementi successivi sono così ottenuti:

$$c_{i+1} = \begin{cases} \frac{c_i}{2} & \text{se } c_i \text{ è pari} \\ 3 * c_i + 1 & \text{se } c_i \text{ è dispari} \end{cases}$$

- Si scriva una procedura `calcolaSuccessivo` che riceva tramite `$a0` un numero naturale e calcoli l'elemento successivo della sequenza. Tale numero è stampato a video e restituito attraverso `$v0`.

Soluzione

```
input:    .data
          .ascii "Introduci un numero: "
          .text
          .globl main
          .ent main
main:     la $a0, input
          li $v0, 4
          syscall
          li $v0, 5
          syscall
          move $a0, $v0
          jal calcolaSuccessivo
          li $v0, 10
          syscall
          .end main
```

Soluzione [cont.]

```
        .ent calcolaSuccessivo
calcolaSuccessivo:
        and $t0, $a0, 1
        beqz $t0, pari
        mulou $t0, $a0, 3      # il numero e' dispari
        addi $t0, $t0, 1
        b fine
pari:    sra $t0, $a0, 1
fine:    # stampa il numero seguito da un new line
        move $a0, $t0
        li $v0, 1
        syscall
        li $a0, '\n'
        li $v0, 11
        syscall
        move $v0, $t0
        jr $ra
        .end calcolaSuccessivo
```

Esercizio 3

- La congettura di Collatz afferma che, per qualunque valore iniziale c_0 , la sequenza definita nell'esercizio precedente raggiunge sempre il valore 1 passando attraverso un numero finito di elementi.
- Esempio: se $c_0 = 19$, la sequenza è: 19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1. La sequenza contiene 21 elementi.
- La congettura di Collatz non è mai stata dimostrata, però è stata verificata sperimentalmente per tutti i numeri naturali fino a $87 * 2^{60} \approx 10^{21}$.

Esercizio 3 [cont.]

- Si scriva una procedura `sequenzaDiCollatz` che riceva tramite `$a0` un numero naturale e restituisca attraverso `$v0` il numero di elementi necessari per arrivare a 1.
- La procedura è costituita da un ciclo che a ogni iterazione calcola l'elemento successivo della sequenza, richiamando la procedura `calcolaSuccessivo` implementata nell'esercizio precedente.
- Nota: si ricordi di salvare il valore di `$ra` quando necessario.

Soluzione

```
input:    .data
          .ascii "Introduci un numero: "
output:   .asciiz "Numero di elementi nella sequenza: "
          .text
          .globl main
          .ent main
main:     la $a0, input
          li $v0, 4
          syscall
          li $v0, 5
          syscall
          move $a0, $v0
          jal sequenzaDiCollatz
          move $t0, $v0
          la $a0, output
          li $v0, 4
          syscall
          move $a0, $t0
          li $v0, 1
```

Soluzione [cont.]

```
    syscall
    li $v0, 10
    syscall
    .end main
    .ent sequenzaDiCollatz
sequenzaDiCollatz:
    addi $sp, $sp, -8
    sw $ra, 4($sp)
    sw $s0, ($sp)
    li $s0, 1      # numero di elementi nella successione
ciclo:    beq $a0, 1, fineCiclo
          jal calcolaSuccessivo
          move $a0, $v0
          addi $s0, $s0, 1
          b ciclo
fineCiclo: move $v0, $s0
          lw $s0, ($sp)
          lw $ra, 4($sp)
          addi $sp, $sp, 8
          jr $ra
    .end sequenzaDiCollatz
```

Chiamata del main in QtSpim

- L'assemblatore di QtSpim aggiunge alcune righe di codice prima e dopo la chiamata del main

<code>lw \$4, 0(\$29)</code>	<code>; 183: lw \$a0 0(\$sp) # argc</code>
<code>addiu \$5, \$29, 4</code>	<code>; 184: addiu \$a1 \$sp 4 # argv</code>
<code>addiu \$6, \$5, 4</code>	<code>; 185: addiu \$a2 \$a1 4 # envp</code>
<code>sll \$2, \$4, 2</code>	<code>; 186: sll \$v0 \$a0 2</code>
<code>addu \$6, \$6, \$2</code>	<code>; 187: addu \$a2 \$a2 \$v0</code>
<code>jal 0x00400024 [main]</code>	<code>; 188: jal main</code>
<code>nop</code>	<code>; 189: nop</code>
<code>ori \$2, \$0, 10</code>	<code>; 191: li \$v0 10</code>
<code>syscall</code>	<code>; 192: syscall 10 (exit)</code>

Chiamata del main in QtSpim

- Se il main è *leaf*, può essere terminato con `jr $ra` invece di chiamare la system call 10. Così si evita di avere una syscall ridondante.

- Se il main non è *leaf*, le istruzioni diventano:

```
subu $sp, $sp, 4      # salva $ra nello stack
sw $ra, ($sp)
...                   # istruzioni nel main
lw $ra, ($sp)         # ripristina $ra
addu $sp, 4           # ripristina $sp
jr $ra
```


Esercizio 4

- Si scriva una procedura `determinante2x2` che calcoli il valore del determinante di una matrice quadrata 2x2, ricevendo i 4 elementi tramite i registri `$a0`, `$a1`, `$a2` e `$a3` (matrice memorizzata per righe) e salvi il risultato in `$v0`

$$\det = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} = a_1 b_2 - a_2 b_1$$

- Per validare la procedura, si scriva un anche un programma chiamante che legga 4 valori salvati in memoria e lanci la procedura. Si termini il programma chiamante con `jr $ra`.
- Si assuma di non avere *overflow* nei calcoli.

Soluzione

```
.data
matrice:    .word 10, 6, 7, 4
msg_output: .asciiz "Valore determinante: "
.text
.globl main
.ent main
main:       subu $sp, $sp, 4           # salvataggio di $ra nello stack
            sw $ra, ($sp)
            la $t0, matrice
            lw $a0, ($t0)
            lw $a1, 4($t0)
            lw $a2, 8($t0)
            lw $a3, 12($t0)
            jal determinante2x2
            move $t0, $v0
```

Soluzione [cont.]

```
la $a0, msg_output      # argomento: stringa
li $v0, 4                # syscall 4 (print_str)
syscall
move $a0, $t0 # intero da stampare
li $v0, 1
syscall
lw $ra, ($sp)
addu $sp, 4
jr $ra
.end main
```

```
.ent determinante2x2
```

determinante2x2:

```
mul $t0, $a0, $a3
mul $t1, $a1, $a2
sub $v0, $t0, $t1
jr $ra
.end determinante2x2
```

Esercizio 5

- Si scriva una procedura determinante3x3 in grado di calcolare il valore del determinante di una matrice quadrata 3x3.

$$\det = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}$$

$$\det = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} = a_1 \begin{vmatrix} b_2 & c_2 \\ b_3 & c_3 \end{vmatrix} - b_1 \begin{vmatrix} a_2 & c_2 \\ a_3 & c_3 \end{vmatrix} + c_1 \begin{vmatrix} a_2 & b_2 \\ a_3 & b_3 \end{vmatrix}$$

Esercizio 5 [cont.]

- La procedura `determinante3x3` riceve in input i 9 elementi della matrice. I primi 4 elementi sono passati attraverso i registri `$a0-$a3`, gli altri 5 attraverso lo stack.
- La procedura `determinante3x3` chiama 3 volte la procedura `determinante2x2` implementata nell'esercizio 4.
- Per validare la procedura, si scriva un anche un programma chiamante che legga 9 valori salvati in memoria e lanci la procedura. Si termini il programma chiamante con `jr $ra`.
- Si assuma di non avere *overflow* nei calcoli.

Soluzione

```
        .data
matrice: .word 1, 41, 42, 13, 56, 23, 73, 9, 50
msg_output: .ascii "Valore determinante: "
        .text
        .globl main
        .ent main
main:    subu $sp, $sp, 4 # salvataggio di $ra nello stack
        sw $ra, ($sp)
        la $t0, matrice
        lw $a0, ($t0)
        lw $a1, 4($t0)
        lw $a2, 8($t0)
        lw $a3, 12($t0)
        move $t1, $0      #           indice del ciclo
ciclo:  lw $t2, 16($t0)
        subu $sp, $sp, 4
        sw $t2, ($sp)
        addiu $t0, $t0, 4
        addiu $t1, $t1, 1
        bne $t1, 5, ciclo
        jal determinante3x3
        move $t0, $v0
```

Soluzione [cont.]

```
    la $a0, msg_output          # argomento: stringa
    li $v0, 4                   # syscall 4 (print_str)
    syscall
    move $a0, $t0              # intero da stampare
    li $v0, 1
    syscall
    lw $ra, 20($sp)
    addu $sp, 24
    jr $ra
.end main

    .ent determinante3x3
determinante3x3:
    subu $sp, $sp, 4
    sw $fp, $sp
    move $fp, $sp
    subu $sp, 20                # salva ra e s0-s3
    sw $s0, ($sp)
    sw $s1, 4($sp)
    sw $s2, 8($sp)
    sw $s3, 12($sp)
    sw $ra, 16($sp)
    move $s0, $a0
    move $s1, $a1
    move $s2, $a2
    move $s3, $a3
```

Soluzione [cont.]

```
lw $a0, 20($fp)
lw $a1, 16($fp)
lw $a2, 8($fp)
lw $a3, 4($fp)
jal determinante2x2
mul $s0, $s0, $v0

move $a0, $s3
lw $a1, 16($fp)
lw $a2, 12($fp)
lw $a3, 4($fp)
jal determinante2x2
mul $s1, $s1, $v0

move $a0, $s3
lw $a1, 20($fp)
lw $a2, 12($fp)
lw $a3, 8($fp)
jal determinante2x2
mul $s2, $s2, $v0

add $v0, $s0, $s2
sub $v0, $v0, $s1

lw $s0, ($sp) # ripristina ra e s0-s3
lw $s1, 4($sp)
lw $s2, 8($sp)
lw $s3, 12($sp)
lw $ra, 16($sp)
addu $sp, 20

lw $fp, ($sp)
addu $sp, 4
jr $ra

.end determinante3x3
```