

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 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 dalla procedura chiamante, mentre i registri $\$sx$ sono *callee-save*, e devono essere salvati dalla procedura chiamata.
- Si disegni l'occupazione dello stack durante l'esecuzione della procedura prima di scrivere il codice.

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`.

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 a_0 un numero naturale e restituisca attraverso v_0 il numero di elementi necessari per arrivare a 1.
- La procedura è costituita da un ciclo che ad ogni iterazione calcola l'elemento successivo della sequenza, richiamando la procedura calcolaSuccessivo implementata nell'esercizio precedente.
- Nota: si ricordi di salvare il valore di r_a quando necessario.

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.

Esercizio 5

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

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

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

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.