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 `polinomio` in grado di calcolare il valore di un polinomio p(x) di terzo grado senza usare moltiplicazioni, tramite il metodo delle differenze finite.
- Nel main, inizializzare i registri $t0, $t1, $t2 e $t3 con i coefficienti del polinomio.
  Esempio: $p(x) = 4x^3 + 2x^2 – 5x + 3$
  $t0 = 4, $t1 = 2, $t2 = -5, $t3 = 3
- Nel main, inizializzare alcuni registri con valori utili:
  $s0 = 2^3 = 8; $s1 = 2^2 = 4; $s2 = 3^3 = 27; $s3 = 3^2 = 9; $s4 = 4^3 = 64; $s5 = 4^2 = 16
- Il main richiama la procedura `polinomio` passando come argomenti p(1), p(2), p(3), p(4) e il valore N, per ottenere p(N).

Esercizio 1: passaggio di parametri

- I primi 4 parametri sono passati attraverso \$a0-\$a3:
  - \$a0 = p(1) = \$t0 + \$t1 + \$t2 + \$t3
  - \$a1 = p(2) = \$t0 * \$s0 + \$t1 * \$s1 + \$t2 * 2 + \$t3
  - \$a2 = p(3) = \$t0 * \$s2 + \$t1 * \$s3 + \$t2 * 3 + \$t3
  - \$a3 = p(4) = \$t0 * \$s4 + \$t1 * \$s5 + \$t2 * 4 + \$t3
- Dal quinto parametro in poi, si deve usare lo stack. Nell'esercizio, il valore N è passato attraverso lo stack.

Esercizio 1: procedura `polinomio`

- La procedura `polinomio` effettua le seguenti inizializzazioni:

$t0 = $a1 - $a0, $t1 = $a2 - $a1, $t2 = $a3 - $a2,

$s0 = $t1 - $t0, $s1 = $t2 - $t1, $s2 = $s1 - $s0, $v0 = $a3

- I valori dei seguenti registri sono aggiornati in un ciclo:
  - $s1 = $s1 + $s2
  - $t2 = $t2 + $s1
  - $v0 = $v0 + $t2
- Il ciclo al punto precedente è ripetuto N – 4 volte. Es: N = 7
  - valore iniziale di $v0 = $a3 = p(4)
  - prima iterazione: $v0 = p(5)
  - seconda iterazione: $v0 = p(6)
  - terza iterazione: $v0 = p(7)

Esercizio 1: salvataggio dei registri

- Quando la procedura `polinomio` restituisce il valore p(N) al programma chiamante, i valori nei registri da `$t0` - `$t3` e `$s0` - `$5` devono essere quelli iniziali (rispettivamente i coefficienti del polinomio e i valori delle potenze).
- Si utilizzi lo *stack* per salvare provvisoriamente il valore dei registri quando necessario.
- Si ricorda che i registri di tipo `$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.
- Per informazioni sul metodo utilizzato: https://it.wikipedia.org/wiki/Macchina_differenziale

Chiamata del main in QtSpim

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

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

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 2

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

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

- 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 `$a0` un numero naturale e restituisca attraverso `$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.

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 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 determinante di una matrice quadrata 3x3.

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

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

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.