Tutorato Architettura degli Elaboratori 07

Alberto Paparella[1]

22 Maggio 2025

[1]Dipartimento di Matematica e Informatica, Università degli studi di Ferrara

Esercizio sulla ricorsione in MIPS

La ricorsione in MIPS

Vogliamo implementare la **funzione di Fibonacci** in MIPS dato il seguente codice C:

```c
int fib(int n) {
    if (n <= 1)
        return n;
    else
        return fib(n - 1) + fib(n - 2);
}
```

Listing 1: Codice C per il calcolo della funzione di Fibonacci.

Attenzione! Si noti che questo codice contiene due chiamate ricorsive. Serve quindi salvare il risultato della prima chiamata a `fib` prima di chiamarla ancora.

La ricorsione in MIPS

1. Assegnare i nomi dei registri alle variabili e determinare qual è il caso base e qual è il caso ricorsivo.
   - Un unico input, *n*, è passato nel registro $a0
   - Il *caso base* è rappresentato dalla clausola **then**
   - Il *caso ricorsivo* dalla clausola **else**
2. Convertire il codice per il *caso base*

```
1 fib:
2     bgt     $a0, 1, recurse
3     move    $v0, $a0
4     jr      $ra
```

Listing 2: Codice assembly MIPS per il caso base.

La ricorsione in MIPS

3. Salvare i **registri salvati** del chiamato e del chiamante sullo *stack*

```
recurse:
    sub $sp, $sp, 12    # Dobbiamo memorizzare 3 registri sullo stack
    sw  $ra, 0($sp)     # $ra e' il primo registro
    sw  $a0, 4($sp)     # $a0 e' il secondo registro, non possiamo assumere che i registri $a non saranno sovrascritti dal chiamato
```

Listing 3: Codice assembly MIPS per la gestione dello stack.

4. Chiamare `fib` ricorsivamente

```
    addi    $a0, $a0, -1    # N-1
    jal     fib
    sw      $v0, 8($sp)     # Memorizzare $v0, il terzo registro da memorizzare sullo stack in modo che non venga sovrascritto dal chiamato
```

Listing 4: Codice assembly MIPS per la prima chiamata ricorsiva.

La ricorsione in MIPS

5. Chiamare `fib` ricorsivamente, **ancora**

```
1    lw      $a0, 4($sp)      # Recuperare il valore
     originale di N
2    addi    $a0, $a0, -2     # N-2
3    jal     fib
```

Listing 5: Codice assembly MIPS per la seconda chiamata ricorsiva.

- La tentazione potrebbe essere di calcolare $N - 2$ sottraendo 1 dal valore in `$a0`, invece di ricaricare N e sottrarre 2.
- Anche se questo è tecnicamente corretto (assumendo che si recuperi il valore originale di `$a0` prima di ritornare dalla procedura), questo è prone a errori (**error prone**) ed un esempio di cattiva pratica di programmazione (**bad coding practice**).
- La convenzione del MIPS indica di non fare **nessuna assunzione** su cosa verrà ritornato in qualunque registro oltre a `$s0-7`, `$sp`, `$gp` e `$ra`, i quali avranno i loro valori preservati.

La ricorsione in MIPS

6. Pulire lo stack e ritornare il risultato

```
1     lw      $t0, 8($sp) # Recuperare il primo risultato
      della funzione
2     add     $v0, $v0, $t0
3     lw      $ra, 0($sp) # Recuperare il return address
4     addi    $sp, $sp, 12
5     jr      $ra
```

Listing 6: Codice assembly MIPS per la pulizia dello stack.

Dal MIPS al C

Nel seguente codice assembly MIPS, il valore nel registro $a0 è un input e il valore nel registro $v0 è l'output.

1. Ritradurre la seguente funzione MIPS al codice C equivalente

```
.
1  func:
2      addi    $t0, $zero, 1   # i = 1
3      addi    $v0, $zero, 1   # v = 1
4  Loop:
5      sle     $t1, $t0, $a0       # Settare $t1 a 1 se (1 <= arg)
6      beq     $t1, $zero, Exit    # Uscire dal loop se (i > arg)
7      mul     $v0, $v0, $t0       # v *= i
8      addi    $t0, $t0, 1         # i++
9      j       Loop                # Ciclo
10 Exit:
11     jr      $ra
```

Listing 7: Codice assembly MIPS

2. Quale funzione matematica esegue questo codice?

Soluzione

1. Ritradurre la seguente funzione MIPS al codice C equivalente

```c
int func(int arg) {
    int v = 1, i;
    for (i = 1; i <= arg; i+) {
        v = v * i;
    }
    return v;
}
```

Listing 8: Codice C della soluzione.

2. Il codice esegue il **fattoriale per argomenti non-negativi**

Esercizi di laboratorio

Istruzioni

Ogni esercizio è descritto tramite un programma in linguaggio C.

Se la realizzazione di un I/O non è richiesta, le variabili possono essere inizializzate con istruzioni assembler o con direttive (nel caso degli array). Si raccomanda di seguire le convenzioni del linguaggio assembler specie per ciò che riguarda le funzioni.

Si raccomanda di utilizzare i commenti per indicare la corrispondenza fra variabili del C e registri.

I commenti sono anche utili per descrivere cosa volete fare nel codice.

Esercizio 1

Programma che calcola il massimo fra 3 interi. L'esercizio deve essere risolto senza utilizzare macro (bge ...).

```c
int main() {
    int a, b, c;
    int x;  /* massimo */

    a = 4; b = 10; c = 8;
    /* acquisisce a, b, c (non importa farlo) */
    x = c;
    if ((a > b) && (a > c))
        x = a;
    else
        if (b > c)
            x = b;
    /* stampa x (non importa farlo) */
}
```

Listing 9: Codice C per il calcolo della del massimo fra 3 interi.

Esercizio 2

Calcolo del massimo di un vettore. Anche in questo caso non si devono utilizzare macro.

```c
int main() {
    int array[8] = {0,1,4,2,7,8,4,6};
    int i, x;   /* x = massimo del vettore */

    /* acquisisce array (da non fare) */
    i = 1;
    x = array[0];
    while (i < 8) {
        if (array[i] > x)
            x = array[i];
        i = i + 1;
    }
    /* stampa x (da non fare) */
}
```

Listing 10: Codice C per il calcolo della del massimo di un vettore.

Esercizio 3

Funzione che calcola la distanza fra due interi.

```c
int main() {
    int x, y;
    int v;

    x = 7; y = 4;
    /* acquisisce x e y (da non fare) */
    v = dist(x, y);
}

int dist(int x, int y) {
    int result;
    if (a > b)
        result = a - b;
    else
        result = b - a;
    return result;
}
```

Listing 11: Codice C per il calcolo della distanza fra due interi.

Esercizio 4

Programma che calcola il numero di uni presenti in un intero a 32 bit.

```c
int main() {
    int i, n, x, y;

    n = i = 0;
    x = 18; /* intero di cui si calcola il no. di uni */

    while (1 < 32) {
        y = x & 1;
        n = n + y;
        x = x >> 1;
        i = i + 1;
    }
}
```

Listing 12: Codice C per il calcolo del numero di uni in un intero a 32 bit.

Soluzioni

Esercizio 1 (1)

```
1  .text
2      # a, b, c, x mappati su $s0, $s1, $s2, $s3
3      addi    $s0, $zero, 4
4      addi    $s1, $zero, 10
5      addi    $s2, $zero, 8
6
7      addi    $s3, $s2, 0
8
9      slt     $t0, $s1, $s0   # b < a
10     slt     $t1, $s2, $s0   # c < a
11     and     $t2, $t0, $t1   # &&
12
13     bne     $t2, $zero, label0
14     slt     $t3, $s2, $s1   # c < b
15     beq     $t3, $zero, end
16     addi    $s3, $s1, 0
17     j       end
```

Listing 13: Codice assembly MIPS per il calcolo del massimo fra 3 interi (1).

Esercizio 1 (2)

```
1  label_0:     # (a > b) && (a > c)
2      addi     $s3, $s0, 0
3  end:
4
5      # non richiesto
6      addi     $v0, $zero, 1
7      addi     $a0, $s3, 0
8      syscall
9
10 exit:
11     addi $v0, $zero, 10
12     syscall
```

Listing 14: Codice assembly MIPS per il calcolo del massimo fra 3 interi (2).

Esercizio 2 (1)

```
1  .data
2      array0: .word   0,1,4,2,7,8,4,6
3      # i due vettori hanno la stessa dimensione
4
5  .text
6      # i in $s0, x in $s1
7
8      # indice i
9      addi     $s0, $zero, 4
10     # inizializzo x
11     lw       $s1, array0($zero)
12
13     # dimensione dell'array in byte
14     addi     $t0, $zero, 32
```

Listing 15: Codice assembly MIPS per il calcolo del massimo di un vettore (1).

Esercizio 2 (2)

```
1  loop:
2      beq      $s0, $t0, exit
3      # carico array[i] in $t2
4      lw       $t2, array0($s0)
5      slt      $t1, $s1, $t2
6      beq      $t1, $zero, label    # x >= array[i]
7      addi     $s1, $t2, 0
8
9  label:
10     addi     $s0, $s0, 4
11     j loop
12
13 exit:
14     addi $v0, $zero, 10
15     syscall
```

Listing 16: Codice assembly MIPS per il calcolo del massimo di un vettore (2).

Esercizio 3 (1)

```
1  .text
2  main:
3      # $s0 = y
4      addi    $s0, $zero, 7   # x
5      addi    $s1, $zero, 4   # y
6
7      # operazioni su $s0 e $s1
8      addi    $a0, $s0, 0 # argomento 1
9      addi    $a1, $s1, 0 # argomento 2
10     jal     dist        # chiamata a funzione
11     addi    $s2, $v0, 0 # valore di ritorno
12
13 exit:
14     addi $v0, $zero, 10
15     syscall
```

Listing 17: Codice assembly MIPS per il calcolo della distanza fra due interi (1).

Esercizio 3 (2)

```
1  dist:
2      addi    $sp, $sp, -8    # fare spazio sullo stack per
       memorizzare due registri
3      sw      $t0, 0($sp)     # salva $t0 sullo stack
4      sw      $t1, 4($sp)     # salva $t1 sullo stack
5                              # non serve per i registri t
6      slt     $t0, $a1, $a0   # x > y
7      beq     $t0, $zero, label
8      sub     $t1, $a0, $a1   # x - y
9      j       join
10 label:
11     sub     $t1, $a1, $a0
12 join:
13     addi    $v0, $t1, 0
14     lw      $t0, 0($sp) # recupera $t0 dallo stack
15     lw      $t1, 4($sp) # recupera $t0 dallo stack
16     addi    $sp, $sp, 8 # dealloca spazio dallo stack
17     jr      $ra         # ritorna al chiamante
```

Listing 18: Codice assembly MIPS per il calcolo della distanza fra due interi (2).

Esercizio 3 (3)

Attenzione ad indicare una corretta terminazione del programma:

```
exit:
    addi $v0, $zero, 10
    syscall
```

alla fine del codice relativo al **main**.

Avendo una funzione, il rischio è che, dopo aver terminato l'esecuzione del main, il programma riesegua il codice della funzione, entrando in un loop infinito!

Esercizio 4

```
1  .text
2      addi    $s0, $zero, -68 # x
3      addi    $s1, $zero, 0   # y
4      addi    $s2, $zero, 0   # n
5      addi    $s3, $zero, 0   # i
6      addi    $t0, $zero, 32
7
8  loop:
9      beq     $s3, $t0, exit
10     andi    $t1, $s0, 1     # y = x & 1
11     add     $s2, $s2, $t1   # n = n + y
12     srl     $s0, $s0, 1     # x = x >> 1
13     addi    $s3, $s3, 1
14     j       loop
15
16 exit:
17     addi $v0, $zero, 10
18     syscall
```

Listing 19: Codice assembly MIPS per il numero di uni in un intero a 32 bit.