Università degli Studi di Roma
"La Sapienza"

Architettura degli elaboratori II

Funzioni

(tratto da slides di R. Solmi)

Indice degli argomenti

- **Programmazione assembly**
  - Definizione e controllo di sottoprogrammi
  - Definizione di funzione
  - Chiamate a funzioni
  - Convenzioni riguardanti l'uso dei registri
  - Funzioni ricorsive
    - Lo Stack

Funzioni

- **Una funzione (o routine) è una sequenza di istruzioni che esegue un compito definito e che viene usata *come se* fosse una unità di codice.**
  - **Una funzione è una astrazione presente in (quasi) tutti i linguaggi ad alto livello.**
  - **Assegna un *nome* ad una operazione complessa (fatta da più istruzioni)**
  - **Maggiore leggibilità del codice.**
  - **Riutilizzazione del codice**
  - **L'intero programma assembly viene visto come un insieme di funzioni**

  - ***Non è una supportata dall'assembly; quest'ultimo fornisce solo le istruzioni e i registri che consentono di realizzare delle funzioni. È un insieme di convenzioni.***

Esempio Funzione

Funzione Chiamante

```
main:   la $a0,str
        jal length
        sw $v0,result
        jr $ra
```

Argomento: str

Chiamata a funzione

Funzione Chiamata

```
length:   move $t0,$a0
          li $t1,0
nextChar: lbu $t2,($t0)
          beqz $t2,end
          addu $t0,$t0,1
          addu $t1,$t1,1
          j nextChar
end:      move $v0,$t1
          jr $ra
```

Parametro: $a0

Valore di ritorno: $v0

Funzione

- **Una funzione può essere:**
- ***Foglia:*** Se non chiama altre funzioni al suo interno
- ***Non-foglia:*** Se è sia chiamante che chiamata

```
main:   lw $a0,x
        lw $a1,y
        jal adder
        sw $v0,z
```

FUNZIONE NON FOGLIA

```
adder:
        add $v0,$a0,$a1
        jr $ra
```

FUNZIONE FOGLIA

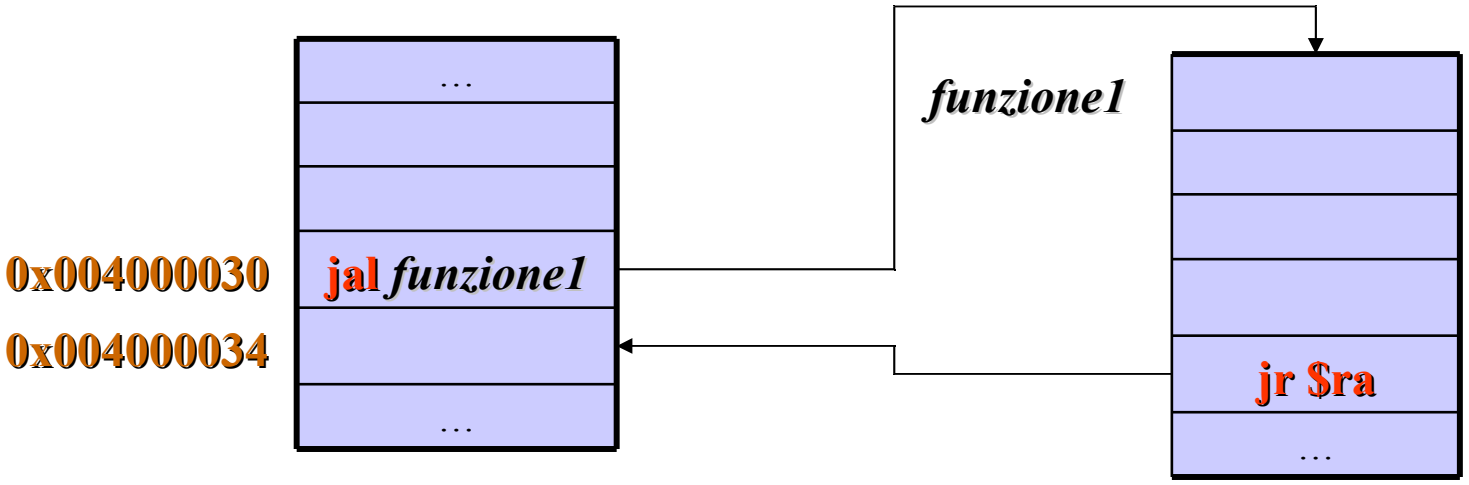
Chiamata a funzione e ritorno

- **Chiamata a funzione:**
  - Per chiamare una funzione si usa l'istruzione

    jal function-name ***(jal = jump and link)***
  - Memorizza l'indirizzo dell'istruzione che *segue* il jal nel registro ***$ra*** e salta all'etichetta ***function-name***
- **Ritorno da funzione:**
  - Per tornare da una funzione si usa l'istruzione

    jr $ra ***(jr = jump register)***
  - Salta all'indirizzo contenuto nel registro ***$ra***
  - *All'occorrenza si può anche usare un altro registro*

Chiamata a funzione e ritorno
-ESECUZIONE -



1. Vengono eseguite le istruzioni della funzione chiamante f che **precedono** la chiamata (jal) a *funzione1*
2. **Chiamata**: l'istruzione jal salva l'indirizzo della prossima istruzione (da eseguire al ritorno) in $ra e salta all'inizio della funzione chiamata
3. Vengono eseguite tutte le istruzioni di funzione1 fino alla jr conclusiva
4. **Ritorno**: l'istruzione jr salta alla prima istruzione che segue la chiamata
5. Vengono eseguite le rimanenti istruzioni della funzione chiamante

Chiamata a funzione e ritorno
-PROBLEMI GENERALI -

1. Il registro $ra non è sufficiente se la funzione chiamata è a sua volta un chiamante (non foglia)
2. Desidero passare dei parametri alla funzione chiamata e ottenere un risultato
3. Mi fa comodo definire una convenzione per stabilire quali registri vengono preservati dopo una chiamata e quali no
4. La funzione può avere bisogno di variabili locali se non bastano i registri stabiliti.

Convenzioni: REGISTRI

Registri usati per passare parametri e risultati:

$a0-$a3 argomenti di una funzione

E se voglio passare più di 4 parametri?

$v0-$v1 risultato di una funzione (32/64 bit)

Registri preservati tra le chiamate

$t0-$t9 e $ra non vengono preservati tra le chiamate

E se ho bisogno di preservarli?

$s0-$s7 vengono preservati tra le chiamate

Come si fanno a preservare se li uso?

E se non mi bastano i registri che ci sono per scrivere una funzione particolarmente complessa?

ACTIVATION FRAME

- **Ogni funzione necessita di uno spazio di memoria per**
  - Memorizzare i valori passati alla funzione come argomenti
  - Salvare i registri che una funzione può modificare, ma che il chiamante si aspetta che vengano preservati
  - Fornire spazio per le variabili locali della procedura (se non mi ho abbastanza registri da utilizzare)
- ***Questo spazio viene chiamato***

  frame di attivazione della funzione

Frame di attivazione statico

```
f:  sw $ra,fFrame
    li $a0,5
    #li $a1-$a3 ...
    li $t0,12
    sw $t0,pFrame+8
    jal p
    lw $ra,fFrame
    jr $ra
```

```
p:  sw $ra,pFrame
    lw $t0,pFrame+8
    addi $t0,$t0,1
    sw $t0,pFrame+4
    jal q
    lw $t0,pFrame+4
    addi $v0,$t0,3
    lw $ra,pFrame
    jr $ra
```

```
q:  sw $s0,qFrame
    sw $s1,qFrame+4
    li $s0,1
    move $s1,$s0
    li $t0,5
    lw $s0,qFrame
    lw $s1,qFrame+4
    jr $ra
```

```
.data
fFrame: .space 4 # spazio per $ra
pFrame: .space 12 # spazio per $ra, $t0 e il V°parametro
qFrame: .space 8 # spazio per $s0 e $s1
```

Frame di attivazione e funz. ricorsive

```
fact:   sw $ra,factFrame
        ble $a0,1,base
        sw $a0,factFrame+4      # salva n
        subu $a0,$a0,1
        jal fact                # fact(n-1)
        lw $t0,factFrame+4      # carica n
        mul $v0,$t0,$v0         # n*(n-1)!
        j end
base:   li $v0,1                # caso base
end:    lw $ra,factFrame
        jr $ra

        .data
        factFrame: .space 8     # spazio per $ra e n
```

- Un singolo frame per funzione *non è sufficiente* nel caso di funzioni *ricorsive*
- Da qui il nome: "*frame di attivazione*" perché è necessario un frame per ogni *attivazione* (chiamata) di funzione

Come faccio ad allocare del nuovo spazio in memoria ogni volta che chiamo una funzione?

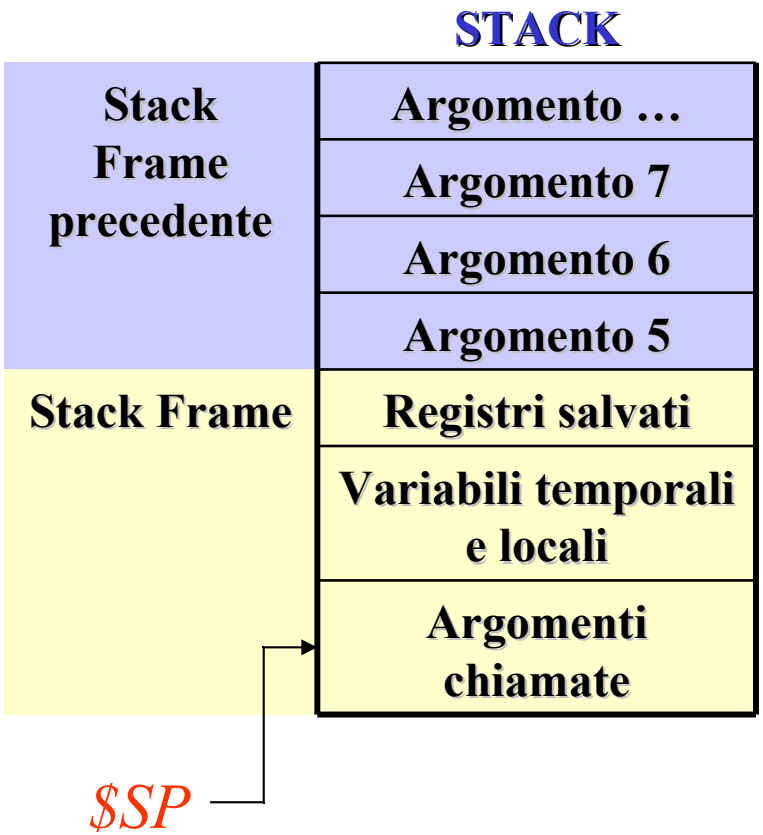
uso lo STACK

STACK

- **Last-in, first-out (LIFO)**
  - I frame di attivazione di funzione vengono creati e distrutti seguendo un modello last- in, first-out
  - È quindi possibile memorizzarli in uno stack
- **Operazioni sullo stack:**
  - ***push***: inserisce un frame nello stack, inserendolo in ultima posizione
  - ***pop***: rimuove l'ultimo frame dallo stack
  - ***read/write:*** nel frame corrente (ultimo elemento) dello stack

STACK FRAME

- **Uno stack frame contiene:**
  - Registri salvati
  - Variabili locali e temporanee
  - Argomenti per le funzioni chiamate dalla funzione (massimo numero richiesto; si conta dal quinto)
- **È accessibile da:**
  - Stack pointer *$sp*
  - Lo stack cresce verso gli indirizzi più bassi della memoria (Per allocare spazio devo sottrarre)
- ***NB La dimensione del frame deve essere un multiplo di 8 byte***



Convenzioni: Funzioni non foglia

```
f:  subu $sp,$sp,24
    sw $s0,20($sp)
    sw $s1,16($sp)
    sw $ra,12($sp)
    ...
    sw $t1,0($sp)
    sw $t0,8($sp)
    jal p
    lw $t0,8($sp)
    ...

    lw $ra,12($sp)
    lw $s1,16($sp)
    lw $s0,20($sp)
    addu $sp,$sp,24
    jr $ra
```

1. Alloco nello stack uno spazio sufficiente a contenere tutti i registri che devo salvare, le variabili locali e il *numero max* di argomenti.

2. Salvo i registri $s0-$s7 che intendo usare e $ra

a. Se la funzione chiamata richiede più di 4 argomenti li metto nello stack.

b. Se voglio preservare i registri $t0-$t9,$a0-$a3,$v0-$v1 devo salvarli prima della chiamata a funzione e ripristinarli dopo.

3. Ripristino i registri salvati $ra, $s0-$s7.

4. Libero lo spazio sullo stack allocato all'inizio

Convenzioni: Funzioni foglia

```
p: subu $sp,$sp,16
   sw $s0,12($sp)
   sw $s1,8($sp)
   sw $s2,4($sp)
   ...

   lw $s0,12($sp)
   lw $s1,8($sp)
   lw $s2,4($sp)
   addu $sp,$sp,16
   jr $ra

q: ...
   jr $ra
```

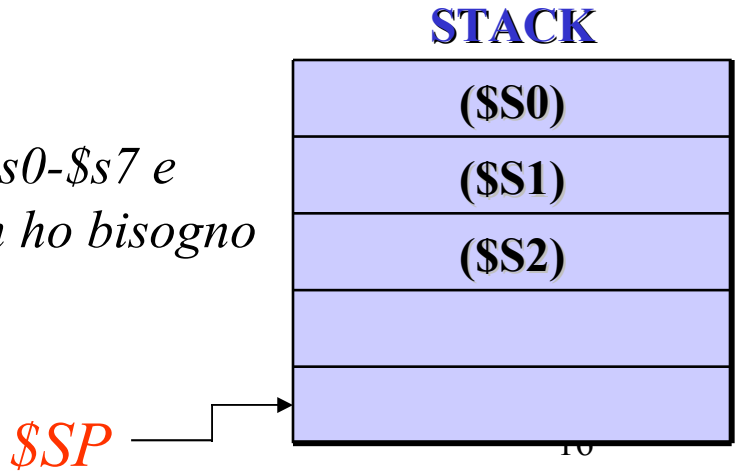
1. Alloco nello stack uno spazio sufficiente a contenere tutti i registri che devo salvare e le eventuali variabili locali

2. Salvo i registri $s0-$s7 che intendo usare

3. Ripristino i registri salvati $s0-$s7.

4. Libero lo spazio sullo stack allocato all'inizio

NB. Se non uso i registri $s0-$s7 e non ho variabili locali non ho bisogno di fare nulla!



Fattoriale (Ricorsivo)

```
main:
        subu $sp,$sp,8
sw $ra,4($sp)
        lw $a0,n
        jal fact #fact(n)
        sw $v0,result
        lw $ra,4($sp)
        addu $sp,$sp,8
jr $ra
```

```
fact: subu $sp,$sp,8
        sw $ra,4($sp)
        ble $a0,1,base
        sw $a0,0($sp) #salva n
        subu $a0,$a0,1
        jal fact #fact(n-1)
        lw $t0,0($sp) #carica n
        mul $v0,$t0,$v0 #n*(n-1)!
        j end
base:li $v0,1 #caso base

end: lw $ra,4($sp)

addu $sp,$sp,8
        jr $ra
```

Fattoriale (Iterativo)

- Non bisogna usare lo stack
- Il risultato è molto più efficiente sia in spazio che in tempo

NB I compilatori dei linguaggi ad alto livello sono in genere abbastanza intelligenti da fare traduzioni iterative anche di funzioni ricorsive.

```
fact: ble $a0,1,base      #$a0 n
      move $v0,$a0        #$v0 risultato
loop: subu $a0,$a0,1
      mul $v0,$v0,$a0     #n*(n-1)*(n-2)…
      bgt $a0,2,loop
      jr $ra
base: li $v0,1                #caso base
      jr $ra
```

Fattoriale (Precalcolato)

- Il risultato occupa più spazio ma calcola in tempo costante i risultati più richiesti

NB I compilatori dei linguaggi ad alto livello non fanno questo genere di ottimizzazioni perché non sanno se ne vale la pena. Le macchine virtuali si.

```
fact: ble $a0,10,base    # $a0 n
      move $v0,$a0       # $v0 risultato
loop: subu $a0,$a0,1
      mul $v0,$v0,$a0    # n*(n-1)*(n-2)…
      bgt $a0,2,loop
      jr $ra
base: sll $a0,$a0,2      # caso precalcolato
      lw $v0,table($a0)
      jr $ra

.data
table: .word 1,1,2,6,24,120,720,5040,40320,362880,3628800
```