Università degli Studi di Milano
Dipartimento di Informatica “Giovanni Degli Antoni”
Corso di Laurea Triennale in Informatica

Architettura degli Elaboratori II Laboratorio

Procedure 2/2: Procedure annidate e ricorsive

Procedure «foglia»

- Scenario più semplice: `main` chiama la procedura `funct` che, senza chiamare a sua volta altre procedure, termina e restituisce il controllo al `main`

main

```
f = f + 1;

if (f == g)
        res = funct(f,g);

else
        f = f -1;

print(res)
```

funct

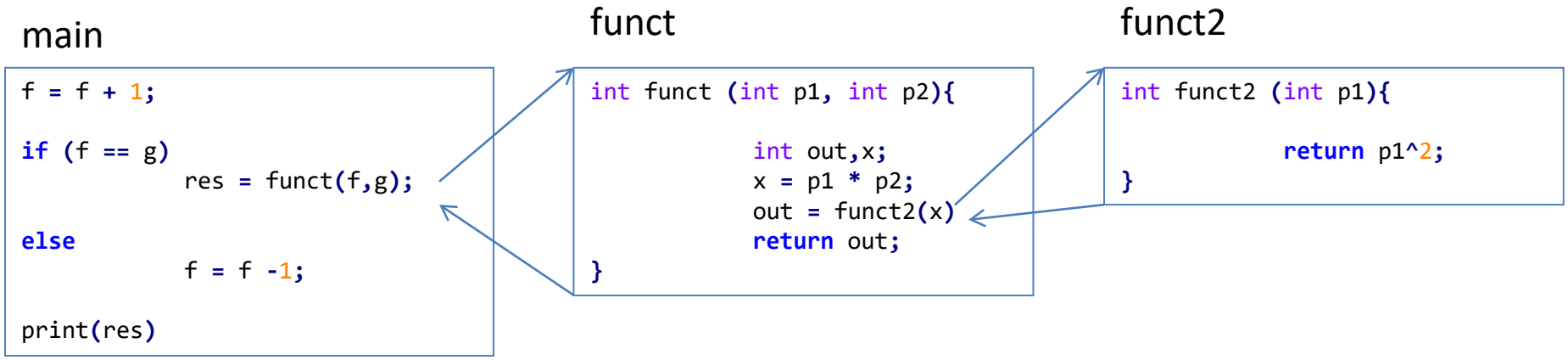
```
int funct (int p1, int p2){

        int out;
        out = p1 * p2;
        return out;
}
```

- Una procedura che non ne chiama un'altra al suo interno è detta procedura *foglia*

  Perché? Rappresentiamo le nostre procedure con un albero: le procedure diventano nodi e un arco tra due nodi x e y indica che x contiene almeno una chiamata a y

Procedure non «foglia»

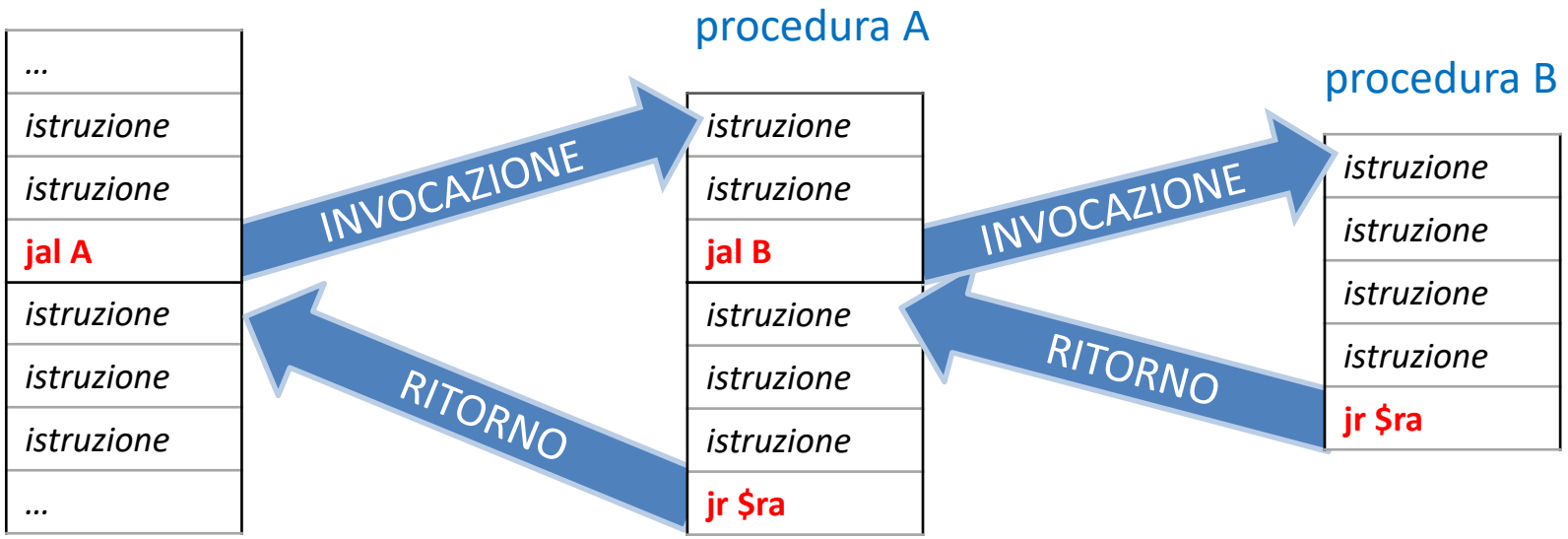
- Una procedura che può invocarne un'altra durante la sua esecuzione non è una procedura foglia, ha annidata al suo interno un'altra procedura:



- Se una procedura contiene una chiamata ad un'altra procedura dovrà effettuare delle operazioni per (1) garantire la non-alterazione dei registri opportuni (2) consentire una restituzione del controllo consistente con l'annidamento delle chiamate.

- *Ricordiamo: in assembly la modularizzazione in procedure è un'assunzione concettuale sulla struttura e sul significato del codice. Nella pratica, ogni «blocco» di istruzioni condivide lo stesso register file e aree di memoria*

Invocazione di procedura annidate



Convenzione sull'uso dei registri da parte delle procedure

Registro:	$0	$1	$2	$3	$4	$5	$6	$7
Sinonimo:	$r0	$at	$v0	$v1	$a0	$a1	$a2	$a3
Registro:	$8	$9	$10	$11	$12	$13	$14	$15
Sinonimo:	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7
Registro:	$16	$17	$18	$19	$20	$21	$22	$23
Sinonimo:	$s0	$s1	$s2	$s3	$s4	$s5	$s6	$s7
Registro:	$24	$25	$26	$27	$28	$29	$30	$31
Sinonimo:	$t8	$t9	$k0	$k1	$fp	$sp	$s8	$ra

deve rimanere invariato

può essere modificato dalla procedura

Local variables (in Go)

```
func pippo() {
    var a , b , c int
    a = 10
    b = 20
    c = a + b
    if a%2 == 0 {
        var d , e , f int
        ...
        for j := 7; j <= 9; j++ {
            k := j+3
            fmt.Println(k)
        }
    } else {
        pippo := 6
        …
    }
    …
}
```

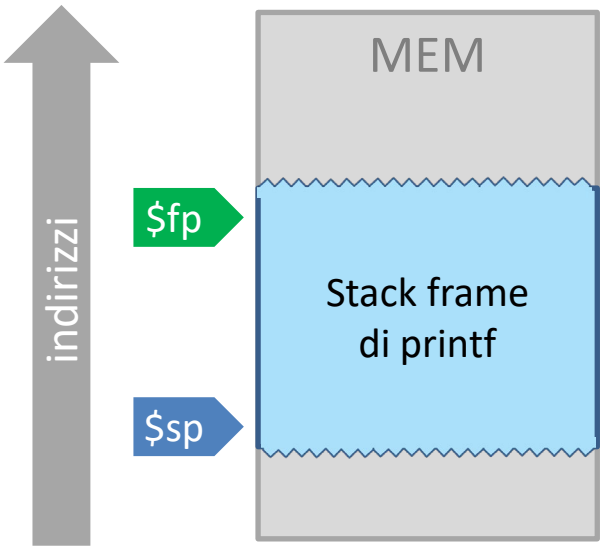
Nuove variabili locali sono aggiunte in vari punti dell'esecuzione di una funzione (compreso il main)

Record di attivazione – Stack frame

- Una procedura ha bisogno di usare la memoria
  - Per memorizzare le sue variabili locali
  - Per memorizzare la copia dei registri da preservare

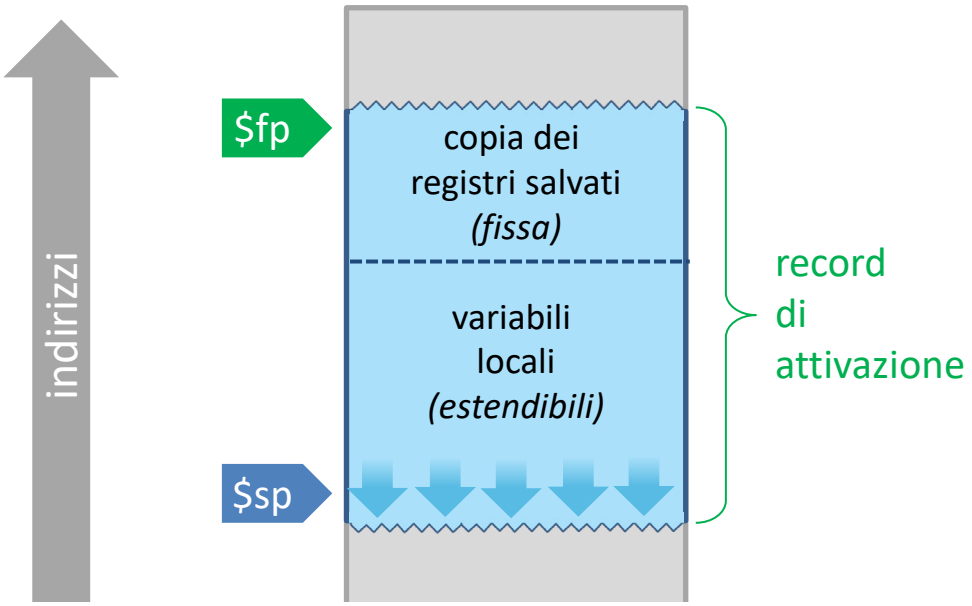
- Dedichiamo ad ogni procedura in esecuzione una sua area di memoria *sullo stack*, detta record di attivazione o stack frame

- MIPS riserva due registri per indirizzare lo stack frame della procedura attualmente in esecuzione:
  da `$sp` (stack pointer)
  a `$fp` (frame pointer)
  compresi!



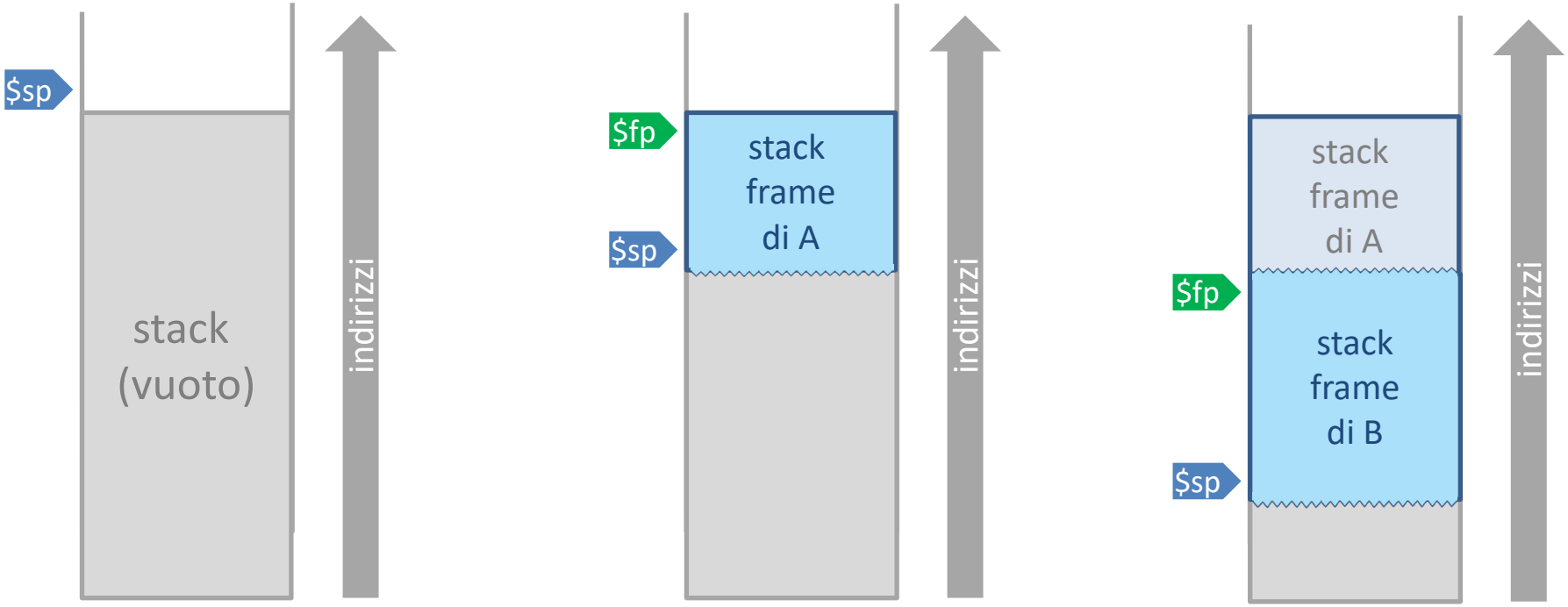
Il record di attivazione di una funzione

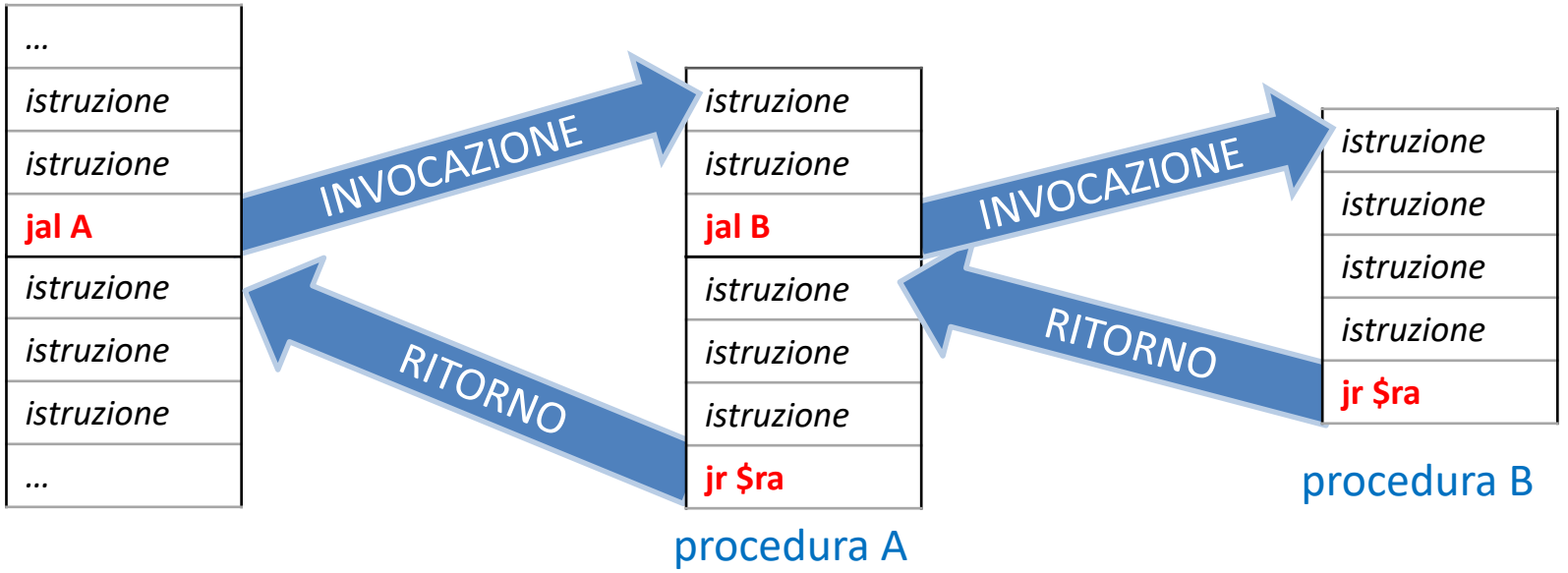
- Il record di attivazione di una procedura memorizza
  - La copia dei registri da preservare per il chiamante
  - Le variabili locali (attaverso push e pop, come normale)



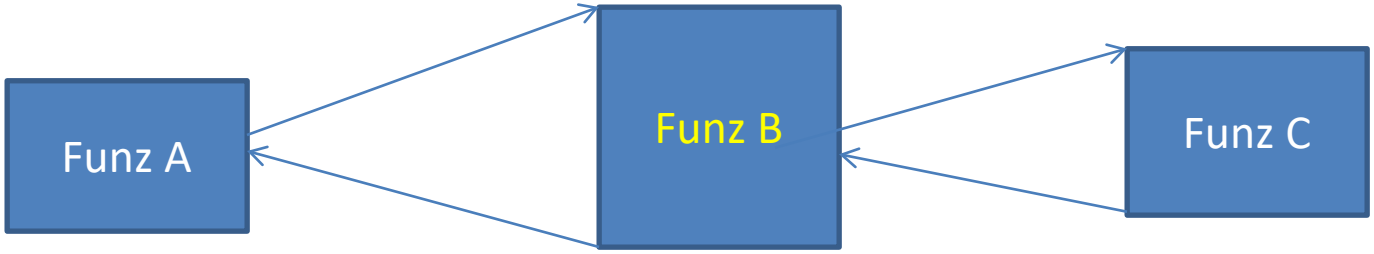
Allocazione e deallocazione degli stack frame

- I record di attivazione si impilano (LIFO) in memoria sullo stack
- quando una procedura viene invocata,
  un nuovo record di attivazione viene impilato nello stack
  - sotto al precedente
  - modificando i registri $sp e $fp
- quando una procedura termina, il suo record di attivazione (che è sempre quello inferiore) viene rimosso
  - modificando i registri $sp e $fp
  - nota: non è necessario «cancellare la memoria» semplicemente, l'area dello stack verrà riutilizzata dalle prossime procedure o variabili locali


$sp
stack
(vuoto)
indirizzi
$fp
stack
frame
di A
$sp
indirizzi
stack
frame
di A
$fp
stack
frame
di B
$sp
indirizzi


...
istruzione
istruzione
jal A
istruzione
istruzione
istruzione
...
INVOCAZIONE
RITORNO
istruzione
istruzione
jal B
istruzione
istruzione
istruzione
jr $ra
procedura A
INVOCAZIONE
RITORNO
istruzione
istruzione
istruzione
istruzione
jr $ra
procedura B

Problemi per le procedure non «foglia»



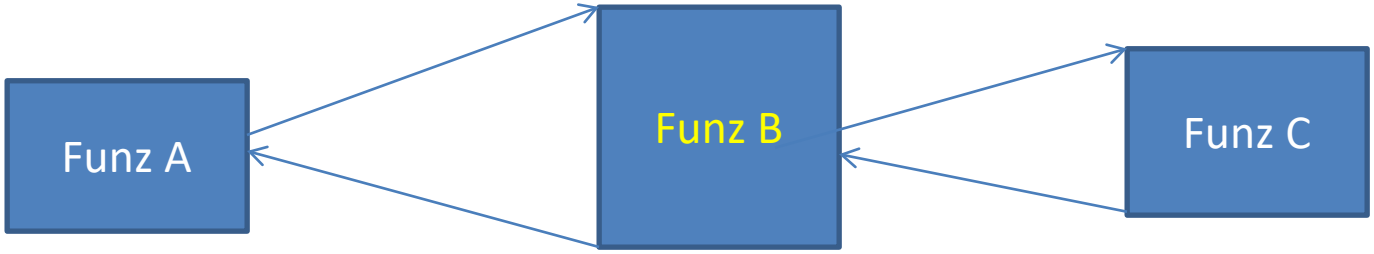
La funzione B ha alcuni problemi da risolvere…
Problema 1:

- Se B usa registri $t questi vengono (potenzialmente) distrutti da C, lecitamente ☹
- Se B usa registri $s (in scrittura), contravviene al «contratto» con A ☹
- Quali registri deve usare B?

Problema 2:

- Quando B usa la JAL per invocare C, sovrascrive il $ra
- Al momento di tornare ad A, non ha più l'indirizzo di ritorno!

Problemi per le procedure non «foglia»



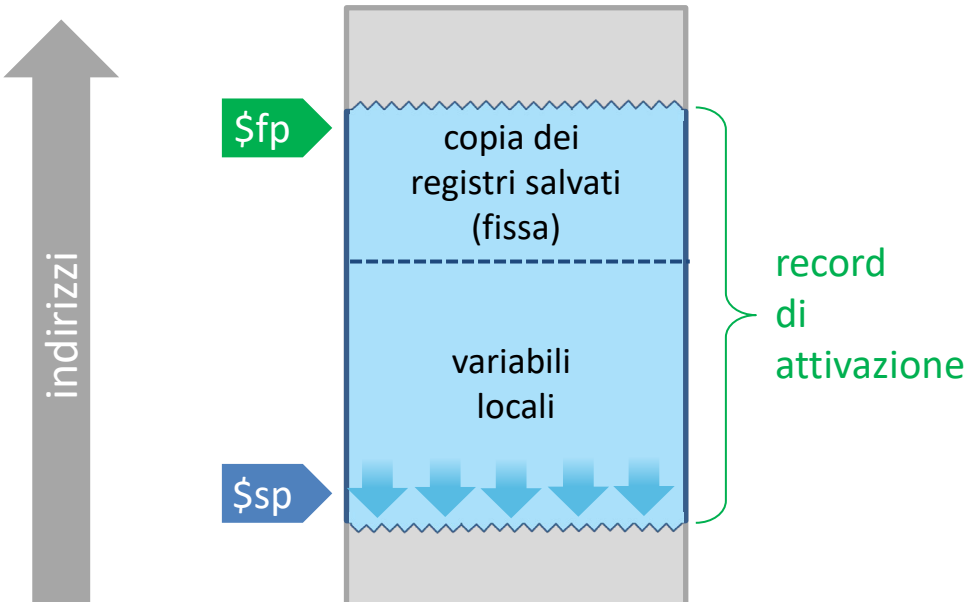
Soluzione ad entrambi i problemi:

Prima di usare i registri $s (ma anche $ra, e gli altri)
Funz B li memorizza nel proprio RECORD DI ATTIVAZIONE

La stessa strategia vale anche per $fa e $sp

$fp e $sp

- $sp può variare nel corso della procedura: viene decrementato quando una si allocano nuove variabili locali (compreso, da parte di sottofunzioni)
- $fp invece *non cambia* durante l'esecuzione della procedura
- $fp è utile a tener traccia di dove sono stati salvati i registri



Manuale per scrivere una procedura (versione completa)

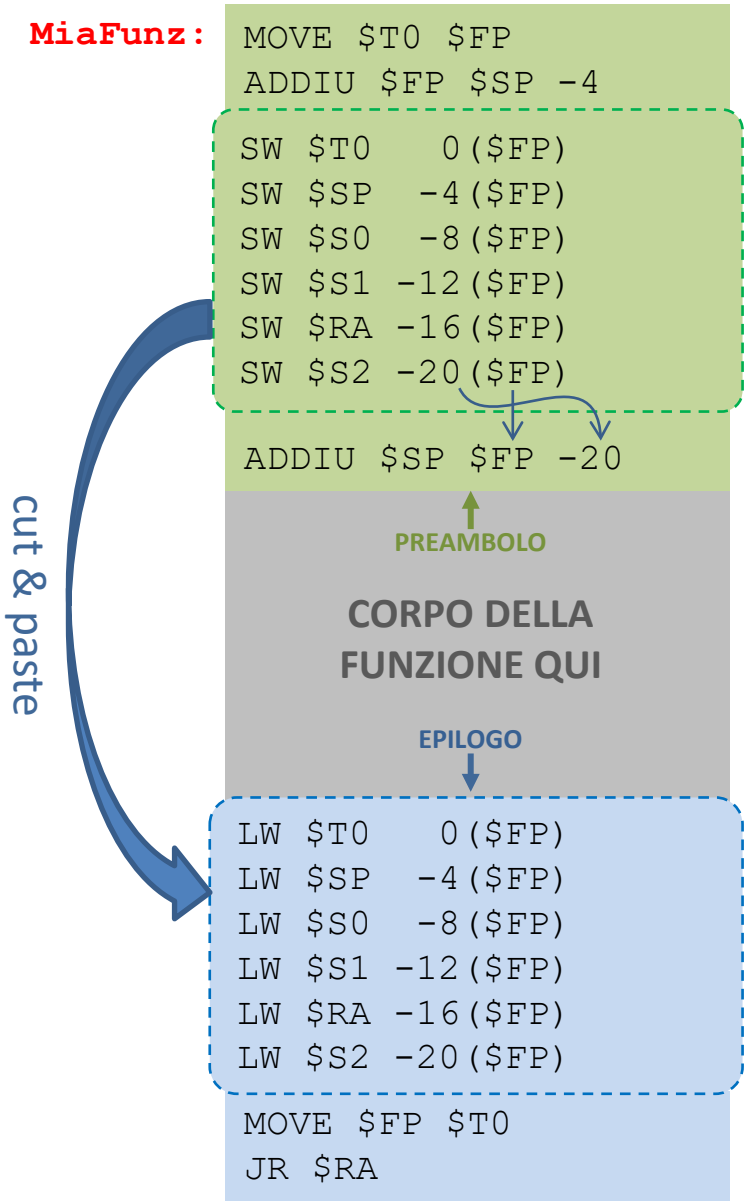
1. salvare una copia di $fp nel frame stack
   - con una store word ad indirizzo $sp – 4
2. aggiornare il valore di $fp a $sp -4
3. salvare una copia di $sp nello stack frame all'indirizzo $fp -4
4. salvare una copia dei registri $s nel frame stack
   - con un store word agli indirizzo, decre $fp – 12, $fp – 16, etc
   - solo quelli che si intende usare
5. salvare una copia di $ra nel frame stack
   - necessario solo se si invoca una funzione
6. aggiornare il valore di $sp
   - [dimensione record di attivazione] da $sp

Preambolo

7. implementare la procedura! (codice)
   - leggere gli eventuali input da $a0 .. $a3
   - scrivere l'eventuale output in $v0 (e/o $v1)
   - se servono nuove variabili locali, ingrandire il record di attivazione (decrementando $sp)
8. ripristinare tutti i registri salvati nel frame stack nei passi 1-5
   - con altrettante load word agli stessi indirizzi
   - $sp verrà ripristinato automaticamente
9. restituire il controllo al chiamante
   - jr $ra

Epilogo

Guida pratica per funzioni non-foglia



Copia temporanea del Frame Pointer *iniziale* (in T0).
Il *nuovo* record di attivazione comincia subito dopo il vecchio.

I valori dei registri *iniziali* sono i salvati (in qualsiasi ordine) nel (nuovo) record di attivazione.
Compreso lo stack pointer SP, il Return Address RA, e anche FP stesso (sotto forma di T0)

Aggiornamento dello SP
(punta sempre all'ultimo elemento occupato dello stack)

La funzione può

- usare i registri S solo se sono stati salvati (qui: s0, s1, s2).
- invocare altre funzioni (quindi usando RA),
- allocare variabili nello stack (quindi usano SP).
- usare i registri T, sapendo che non vengono mantenuti dopo l'invocazione di funzione

Recupero valore iniziale di tutti i registri salvati, compreso lo SP (flush dello stack)

… e compreso il FP
Ritorno al chiamante

Esercizi preliminari

1) Scrivere una procedura che converta in maiuscolo una stringa in input. Suggerimento: usare SB e LB (StoreByte e LoadByte) per gestire dati non allineati. Cosa succede per la stringa “Hello World?”

2) Adattare le funzione per convertire solo le lettere minuscole tramite un ulteriore procedura “MaiuscoloLettera” che agisca su una sola lettera alla volta data in input.

Università degli Studi di Milano
Dipartimento di Informatica “Giovanni Degli Antoni”
Corso di Laurea Triennale in Informatica

Architettura degli Elaboratori II Laboratorio