Architettura degli Elaboratori

MIPS:

Uso di procedure

Barbara Masucci
UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

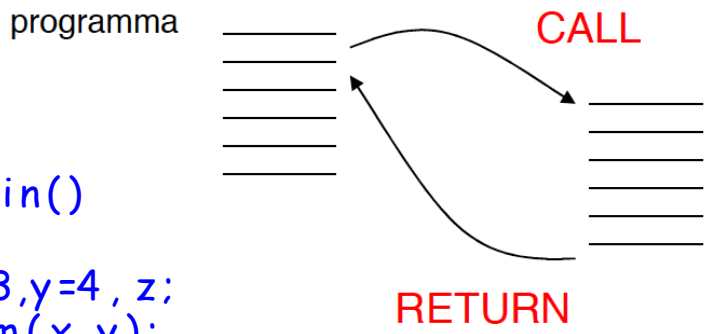
Punto della situazione

- Stiamo studiando l'Instruction Set del MIPS
- Abbiamo visto
  - Le istruzioni aritmetiche, logiche, di trasferimento dati
  - Le istruzioni per prendere decisioni e realizzare cicli
  - L'organizzazione dei registri e della memoria
  - La codifica delle istruzioni in codice macchina

- Obiettivo di oggi
  - Chiamata di procedure in assembly MIPS

Procedure

- Le procedure (anche dette funzioni o sottoprogrammi) sono utilizzate quando si vuole evitare di riscrivere più volte lo stesso codice
  - Permettono di organizzare il programma in moduli, che possono essere richiamati più volte quando serve




```
void main()
{
   int x=3,y=4, z;
   z=sum(x,y);
}
```

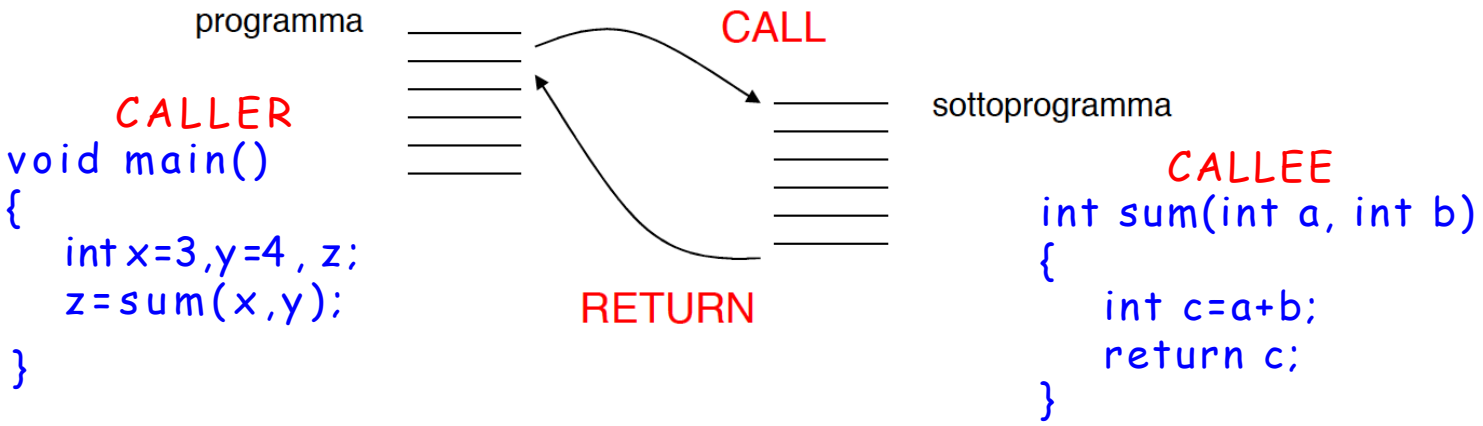
sottoprogramma

```
int sum(int a, int b)
{
   int c=a+b;
   return c;
}
```

Procedure

Abbiamo quindi due entità:

- Il programma chiamante (caller), che può essere a sua volta una procedura
- La procedura chiamata (callee)



- Una procedura che non chiama altre procedure viene detta procedura foglia

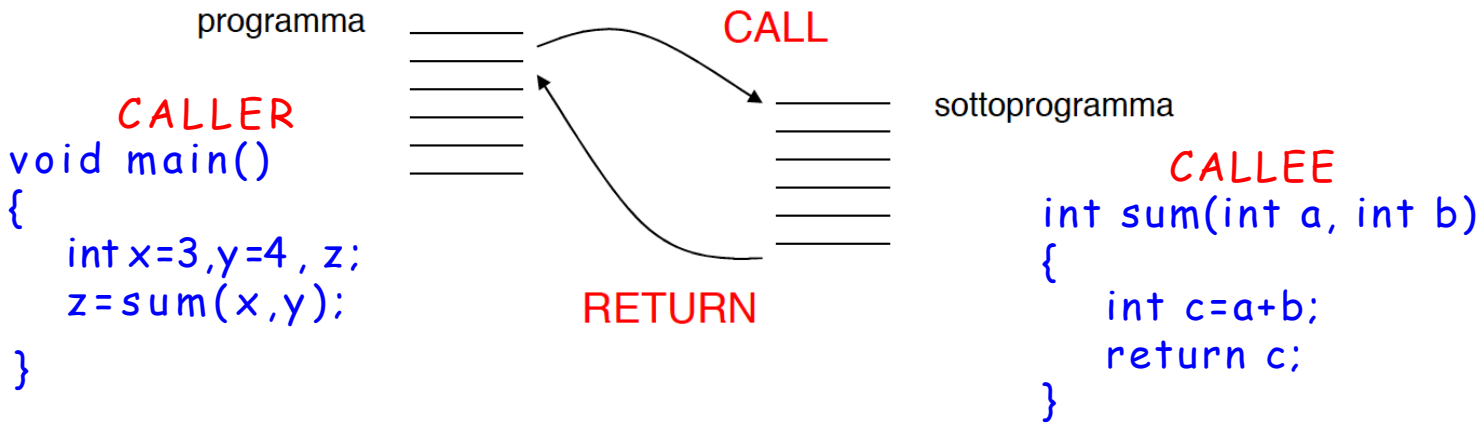
Caller e callee

- La procedura chiamante (caller) deve eseguire i passi seguenti:
  1. Mettere i parametri di ingresso in un luogo accessibile alla procedura chiamata
  2. Trasferire il controllo alla procedura chiamata

- La procedura chiamata (callee) deve
  1. Allocare lo spazio di memoria necessario alla sua esecuzione (record di attivazione)
  2. Eseguire il compito richiesto
  3. Memorizzare il risultato in un luogo accessibile alla procedura chiamante
  4. Restituire il controllo alla procedura chiamante

Chiamata di procedure

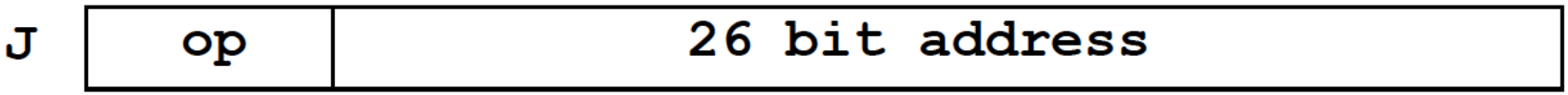
- E' necessaria un'istruzione per
  - Effettuare il salto alla procedura, trasferendole il controllo
  - Salvare l'indirizzo di ritorno (istruzione successiva alla chiamata a procedura) per consentire al caller di proseguire il suo lavoro dopo aver ricevuto il risultato dal callee



Istruzione jal

- L'istruzione fornita dal MIPS è Jump And Link (jal)
  - jal Label_Procedura salta all'indirizzo (della memoria istruzioni) con etichetta Label_Procedura e salva l'indirizzo di ritorno in un apposito registro: $ra

- L'istruzione jal ha formato J e codice operativo 3



Istruzione jr

- Quando la procedura chiamata ha terminato il suo compito, restituisce il controllo alla procedura chiamante tramite l'istruzione Jump Register:

  jr $ra

- L'istruzione jr ha formato R
  - codice operativo 0
  - rs=indirizzo del registro a cui saltare, nel caso dell'istruzione jr $ra, allora rs=31
  - rt=rd=shamt=0
  - funct=8

op	rs	rt	rd	shamt	funct

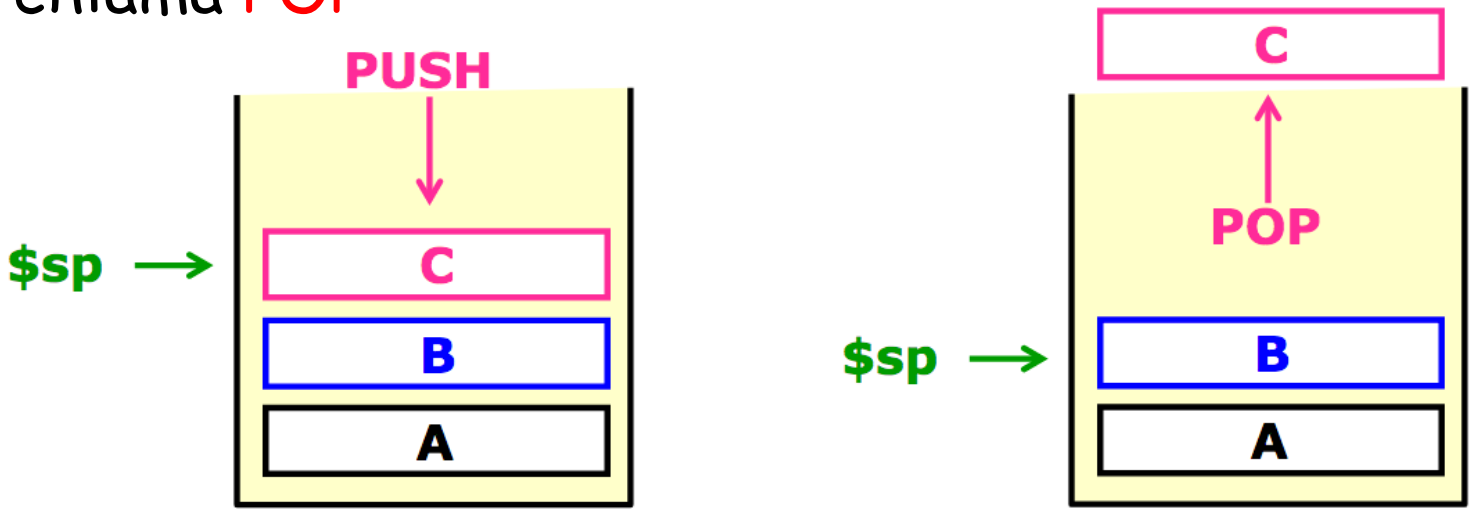
Procedure e registri

- Che succede quando una procedura termina il suo lavoro?
  - Il contenuto dei registri usati dal programma chiamante deve essere ripristinato con il valore precedente la chiamata
  - Pertanto è necessario copiare il contenuto dei registri in memoria prima che la procedura li modifichi

- La struttura dati più adatta per tale compito è lo stack (pila), una coda del tipo Last-In-First-Out
  - E' dotato di un puntatore (stack pointer) all'ultimo dato inserito, a cui è riservato un registro del MIPS: $sp

Stack

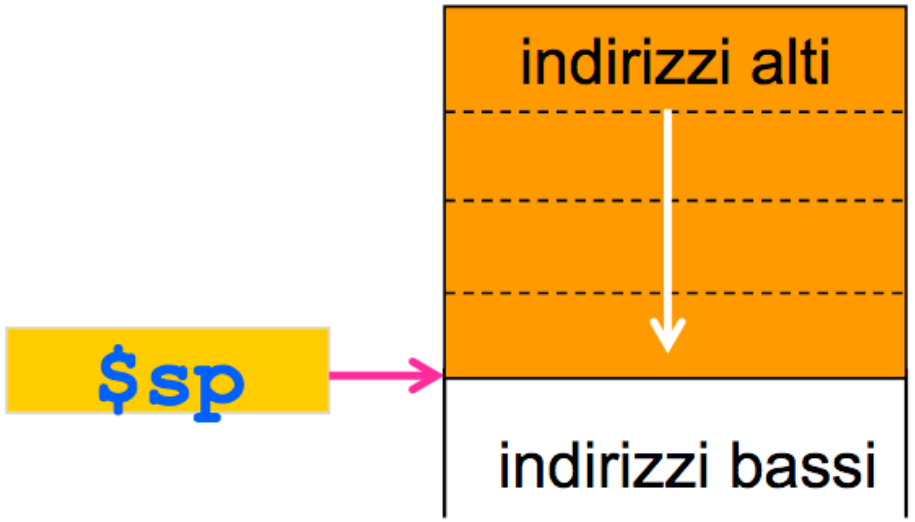
- L'operazione di inserimento di un elemento nello stack si chiama PUSH
- L'operazione di prelievo di un elemento dallo stack si chiama POP



- Il valore di $sp deve essere aggiornato dopo ogni PUSH e ogni POP

Gestione dello Stack

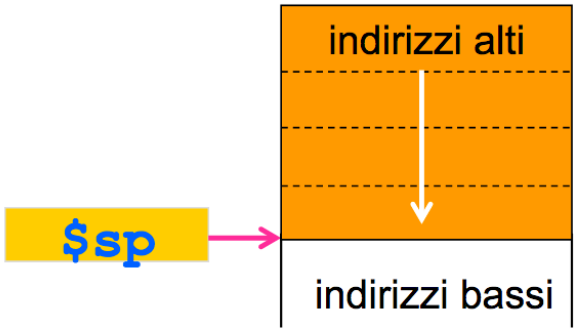
- Lo stack si trova in memoria e cresce da indirizzi alti verso indirizzi bassi



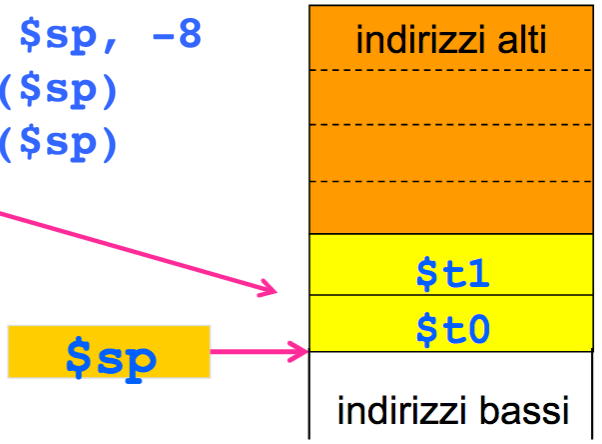
Gestione dello Stack

- Inserimento di un dato nello stack: PUSH
  - Si decrementa di 4 il valore di $sp per allocare spazio al nuovo dato (un dato occupa 32 bit)
  - Si esegue una istruzione sw per inserire il dato nello stack

- Ad esempio, per salvare il valore di due registri $t0 e $t1 nello stack:



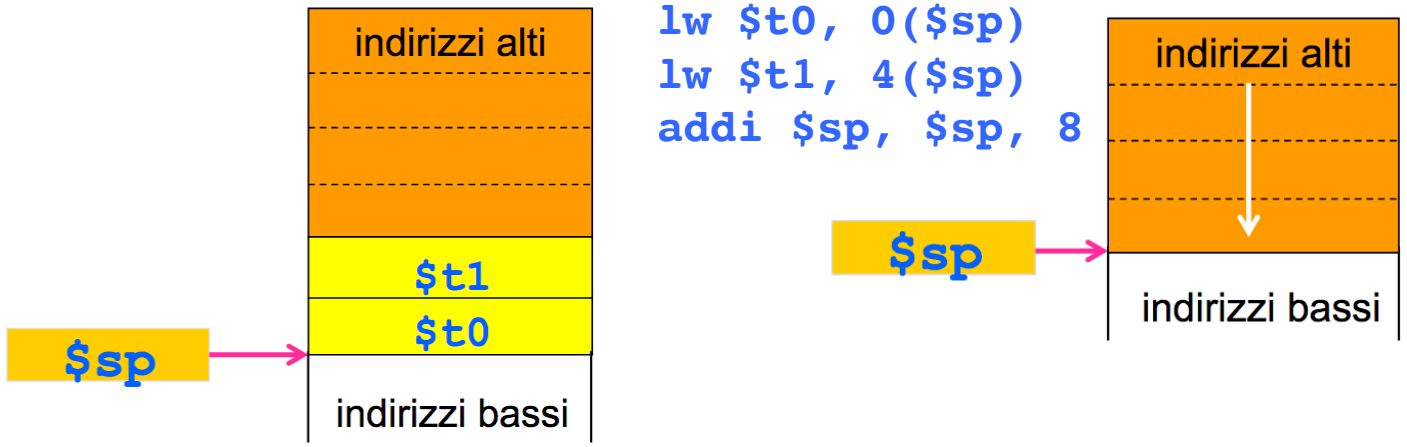

```
addi $sp, $sp, -8
sw $t1, 4($sp)
sw $t0, 0($sp)
```



Gestione dello Stack

- Prelievo di un dato nello stack: POP
  - Si esegue una istruzione lw per prelevare il dato dallo stack
  - Si incrementa di 4 il valore di $sp per eliminare il dato, riducendo la dimensione dello stack

- Ad esempio, per prelevare il valore dei registri $t0 e $t1 dallo stack:



Procedure e registri

- Per le chiamate alle procedure, il MIPS utilizza i suoi 32 registri secondo queste convenzioni
  - $ra: registro per memorizzare l'indirizzo della prima istruzione del chiamante da eseguire al termine della procedura
  - $a0,…,$a3: quattro registri argomento per il passaggio dei parametri
  - $v0, $v1: due registri valore per la restituzione dei valori calcolati

Procedure e registri

- La procedura chiamante (caller)
  - Inserisce i dati da passare alla procedura chiamata nei registri argomento $a0,…,$a3
  - Salta alla procedura chiamata mediante l'istruzione jal Label_Procedura e inserisce il valore di PC+4 (corrispondente all'indirizzo dell'istruzione successiva alla chiamata al callee nella procedura chiamante) nel registro $ra
- La procedura chiamata (callee)
  - Esegue il compito richiesto
  - Memorizza il risultato nei registri $v0, $v1
  - Restituisce il controllo alla procedura chiamante tramite l'istruzione jr $ra

Stack: Esempio di utilizzo

- Vediamo il codice assembly MIPS corrispondente alla seguente procedura in C:

```
int somma (int g, int h, int i, int j)
{
        int f;
        f = (g+h) - (i+j);
        return f
}
```

- Nota: si tratta di una procedura foglia, perchè non chiama altre procedure

Stack: Esempio di utilizzo

- La procedura inizia specificando l'etichetta SOMMA:
- Viene creato il record di attivazione, facendo spazio nello stack per memorizzare i registri che la procedura può modificare
  - La procedura utilizza tre registri temporanei $s0 (conterrà f), $s1 (conterrà g+h) ed $s2 (conterrà i+j)
  - Quindi, bisogna far posto per 3 word

```
addi, $sp, $sp, -12#aggiorna lo stack pointer
                   #per far posto a 3 elementi (word)
```

- Viene fatto il push dei tre registri sullo stack

```
sw $s0, 8($sp) #salva nello stack il registro $s0
sw $s1, 4($sp) #salva nello stack il registro $s1
sw $s2, 0($sp) #salva nello stack il registro $s2
```

Stack: Esempio di utilizzo

- Poi ci sono le istruzioni corrispondenti al corpo della procedura
  - Per i parametri della procedura (g, h, i, j) vengono utilizzati i registri argomento $a0, $a1, $a2, $a3

```
add $s1, $a0, $a1 #il registro $s1 contiene g+h
add $s2, $a2, $a3 #il registro $s2 contiene i+j
sub $s0, $s1, $s2 #f assume il valore (g+h) – (i+j)
```

- Per restituire il valore di f occorre copiarlo in un registro di ritorno

```
add $v0, $s0, $zero #restituzione di f
```

Stack: Esempio di utilizzo

- Prima del ritorno al programma chiamante, il vecchio valore dei registri deve essere rispristinato mediante un'operazione di pop dallo stack (deallocazione memoria)

```
lw $s2, 0($sp) #ripristina il registro $s2
lw $s1, 4($sp) #ripristina il registro $s1
lw $s0, 8($sp) #ripristina in registro $s0
```

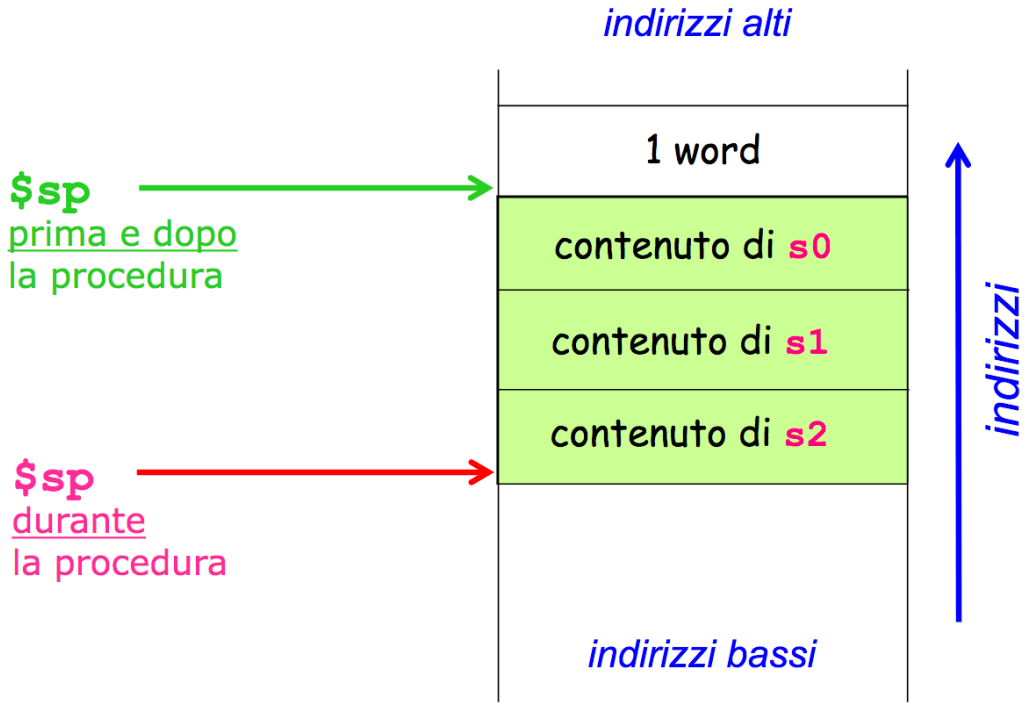
- Lo stack pointer deve essere aggiornato

```
addi $sp, $sp, 12 #aggiornamento sp dopo l'eliminazione
                  #di 3 elementi dallo stack
```

- La procedura termina con un'istruzione di salto al programma chiamante

```
jr $ra #ritorno al programma chiamante
```

Stack:
Esempio di utilizzo



Situazione dello stack prima, durante e dopo
la chiamata della procedura

Struttura di una procedura

Ogni procedura è composta da tre sezioni consecutive

- Prologo
  - Acquisizione delle risorse necessarie per memorizzare i dati interni alla procedura e il salvataggio dei registri
  - Salvataggio dei registri
- Corpo
  - Esecuzione della procedura vera e propria
- Epilogo
  - Salvataggio del risultato in un luogo accessibile al chiamante
  - Ripristino dei registri
  - Liberazione delle risorse usate per la procedura
  - Restituzione del controllo alla procedura chiamante

Stack:
Esempio di utilizzo

SOMMA:

prologo

```
addi, $sp, $sp, -12
sw $s0, 8($sp)
sw $s1, 4($sp)
sw $s2, 0($sp)
```

corpo

```
add $s1, $a0, $a1
add $s2, $a2, $a3
sub $s0, $s1, $s2
```

epilogo

```
add $v0, $s0, $zero
lw $s2, 0($sp)
lw $s1, 4($sp)
lw $s0, 8($sp)
addi $sp, $sp, 12
jr $ra
```

Stack: Esempio di utilizzo

- Nell'esempio sono stati usati dei registri temporanei $s0, $s1, $s2 il cui valore (precedente alla modifica da parte della procedura) è stato salvato nello stack e poi ripristinato

- Per evitare di salvare e ripristinare registri il cui valore non sarò più utilizzato, il MIPS utilizza questa suddivisione
  - $t0, …,$t9: registri temporanei il cui contenuto non viene salvato in caso di chiamata a procedure
  - $s0,…,$s7: registri il cui contenuto deve essere salvato in caso di chiamata a procedure

Stack: Esempio di utilizzo

- Per eliminare il salvataggio dei registri $s1 ed $s2, possiamo usare $t1 al posto di $s1 e $t2 al posto di $s2

```
add $s1, $a0, $a1          add $t1, $a0, $a1
add $s2, $a2, $a3          add $t2, $a2, $a3
```

- Inoltre possiamo eliminare l'uso del registro $s0 usando direttamente $v0 per memorizzare il risultato da restituire al chiamante

```
sub $s0, $s1, $s2          sub $v0, $t1, $t2
```

Stack: Esempio di utilizzo

- Il codice della procedura senza il salvataggio dei registri è più semplice

```
SOMMA:
        add $t1, $a0, $a1
        add $t2, $a2, $a3
        sub $v0, $t1, $t2
        jr $ra
```

- Il programma chiamante potrebbe essere questo:

```
MAIN:
        ...
        addi $a0, $zero, 5
        addi $a1, $zero, 6
        addi $a2, $zero, 4
        addi $a3, $zero, 3
        jal SOMMA
        ...
```

Riepilogo e riferimenti

- Cenni sull'uso di procedure in MIPS
  - [PH] par. 2.8
    (esclusa la parte da "Procedure annidate" in poi)