


Il Linguaggio Assembly: Le procedure

Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
borghese@dsi.unimi.it

Università degli Studi di Milano

Riferimento sul Patterson: 2.6, 2.7


Sommario

Gli attori

Ci sono due ***attori:***
• Procedura chiamante.
• Procedura chiamata.

I due problemi delle procedure**:**
• Passaggio dei dati.
• Trasferimento del controllo.

```
f = f + 1;
if (f == g)
 res = funct(f,g)
else f = f -1;
……
```

```
int funct (int p1, int p2)
{ int out
   out = p1 * p2;
   return out;
}
```

I due moduli si parlano solamente attraverso i parametri:
-Parametri di input (argomenti della funzione).
-Parametri di output (valori restituiti dalla funzione).

Struttura di una procedura

- Ogni procedura ha:
  - **un prologo**
    - Acquisire le risorse necessarie per memorizzare i dati interni alla procedura ed il salvataggio dei registri.
    - Salvataggio dei registri di interesse.
  - **un corpo**
    - Esecuzione della procedura vera e propria
  - **un epilogo**
    - Mettere il risultato in un luogo accessibile al programma chiamante.
    - Ripristino dei registri di interesse.
    - Liberare le risorse utilizzate dalla procedura
    - Restituzione del controllo alla procedura chiamante.

I compiti

- La procedura **chiamante** deve eseguire le seguenti operazioni:
  - Predisporre i parametri di ingresso della procedura in un posto accessibile alla procedura
  - Trasferire il controllo alla procedura
- La procedura **chiamata** deve eseguire le seguenti operazioni:
  - Allocare lo spazio di memoria necessario alla memorizzazione dei dati e alla sua esecuzione (record di attivazione)
  - Eseguire il compito richiesto
  - Memorizzare il risultato in un luogo accessibile al chiamante
  - Restituire il controllo al chiamante

MIPS: Software conventions for Registers

0	zero	constant 0
1	at	reserved for assembler
2	v0	expression evaluation &
3	v1	function results
4	a0	arguments
5	a1	
6	a2	
7	a3	
8	t0	temporary: caller saves
. . .		(callee can clobber)
15	t7	
16	s0	callee saves
. . .		(caller can clobber)
23	s7	
24	t8	temporary (cont'd)
25	t9	
26	k0	reserved for OS kernel
27	k1	
28	gp	Pointer to global area
29	sp	Stack pointer
30	fp	frame pointer (s8)
31	ra	Return Address (HW)

Caller = chiamante - callee = chiamato

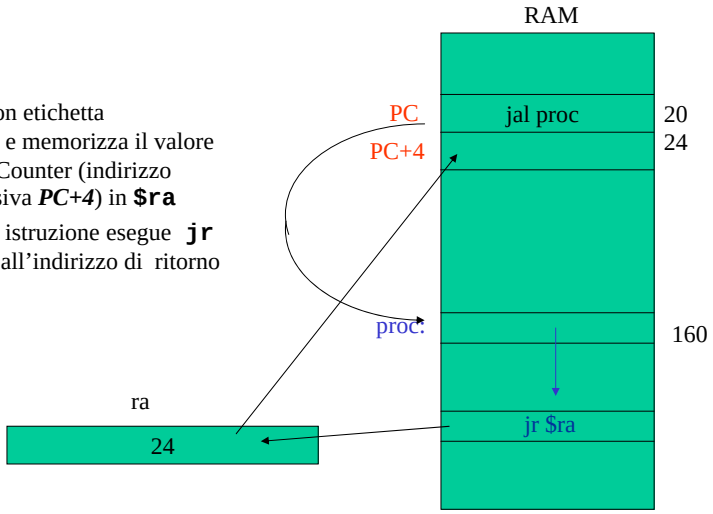
Meccanismo di chiamata::trasferimento del controllo

- Necessaria un'istruzione apposita che cambia il flusso di esecuzione (salta alla procedura) e salva l'indirizzo di ritorno (istruzione successiva alla chiamata di procedura): **`jal`** **(jump and link).**

•**jal Indirizzo_Procedura**

➢Salta all'indirizzo con etichetta **Indirizzo_Procedura** e memorizza il valore corrente del Program Counter (indirizzo dell'istruzione successiva ***PC+4***) in **`$ra`**

•La procedura come ultima istruzione esegue **`jr $ra`** per effettuare il salto all'indirizzo di ritorno della procedura.



La chiamata a procedura::trasferimento dati

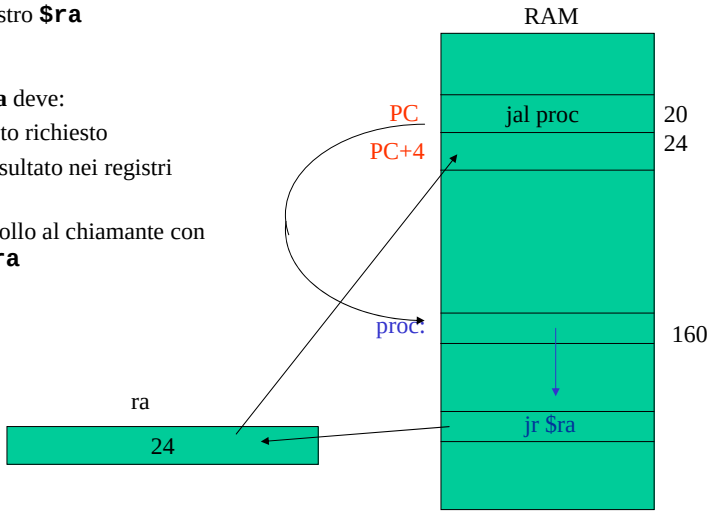
- Il programma **chiamante** deve:
  - Mettere i valori dei parametri da passare alla procedura nei registri **`$a0-$a3`**
  - Utilizzare l'istruzione **`jal address`** per saltare alla procedura e salvare il valore di **`(PC+4)`** nel registro **`$ra`**

•La procedura **chiamata** deve:

–Eseguire il compito richiesto

–Memorizzare il risultato nei registri **`$v0,$v1`**

–Restituire il controllo al chiamante con l'istruzione **`jr $ra`**



Esempio

Ci sono due ***attori:***
- Procedura chiamante.
- Procedura chiamata.

I due problemi delle procedure*:*
- Passaggio dei dati.
- Trasferimento del controllo.

f,g → $s0, $s1

```
        addi $s0, $s0, 1
        bne $s0, $s1, Else
        mov $a0,$s0
        mov $a1,$s1
        jal Funct
        j End
Else:   addi $s0, $s0, -1
End:    ……..
```

```
Funct:  mul $v0, $a0, $a1
        jr $ra
```


Problemi

- Una procedura può avere bisogno di più registri rispetto ai 4 a disposizione per i parametri e ai 2 per la restituzione dei valori.
- Salvare i registri che una procedura potrebbe modificare, ma che il programma chiamante ha bisogno di mantenere inalterati.
- Fornire lo spazio necessario per le variabili locali alla procedura.
- Gestione di procedure annidate (procedure che richiamano al loro interno altre procedure) e procedure ricorsive (procedure che invocano dei ‘cloni’ di se stesse).

utilizzo dello stack

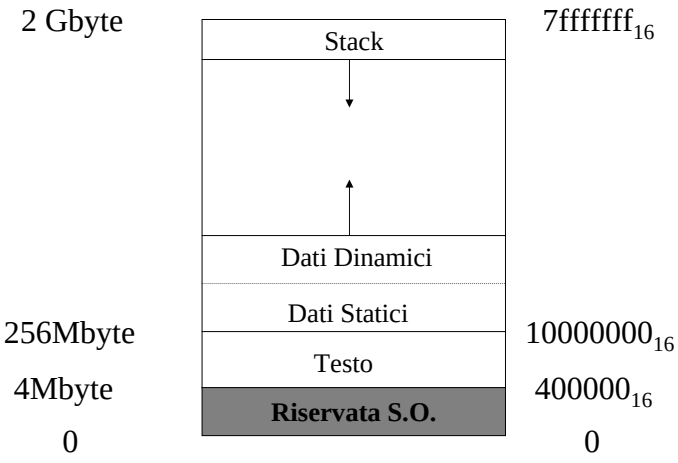
Sommario

Le procedure

Lo stack

La chiamata a procedura

Organizzazione logica della memoria

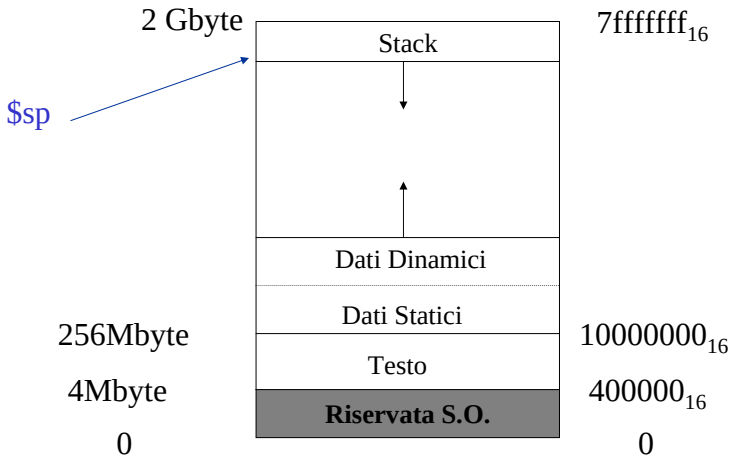


Uso dei registri: registro $sp

Nome	Numero	Utilizzo
$zero	0	costante zero
$at	1	riservato per l'assemblatore
$v0-$v1	2-3	valori di ritorno di una procedura
$a0-$a3	4-7	argomenti di una procedura
$t0-$t7	8-15	registri temporanei (non salvati)
$s0-$s7	16-23	registri salvati
$t8-$t9	24-25	registri temporanei (non salvati)
$k0-$k1	26-27	gestione delle eccezioni
$gp	28	puntatore alla global area (dati)
$sp	29	stack pointer
$s8	30	registro salvato (fp)
$ra	31	indirizzo di ritorno

$sp indica l'ultimo indirizzo dell'area di stack.

Organizzazione logica della memoria



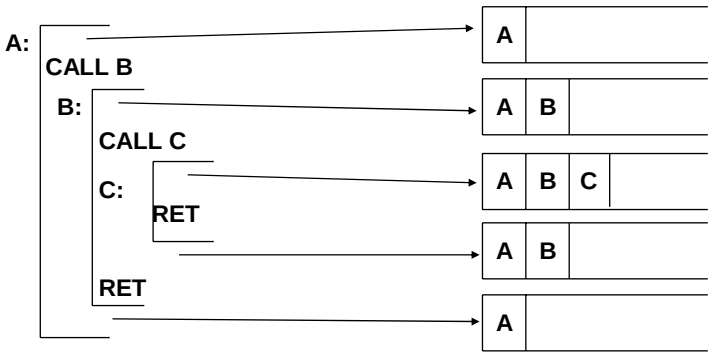
Uso dei registri: registro $fp

Nome	Numero	Utilizzo
$zero	0	costante zero
$at	1	riservato per l'assemblatore
$v0-$v1	2-3	valori di ritorno di una procedura
$a0-$a3	4-7	argomenti di una procedura
$t0-$t7	8-15	registri temporanei (non salvati)
$s0-$s7	16-23	registri salvati
$t8-$t9	24-25	registri temporanei (non salvati)
$k0-$k1	26-27	gestione delle eccezioni
$gp	28	puntatore alla global area (dati)
$sp	29	stack pointer
$s8	30	registro salvato (**fp**)
$ra	31	indirizzo di ritorno

$fp indica l'indirizzo più elevato dello stack in cui sono memorizzati i dati relativi ad una procedura.

Calls: Why Are Stacks So Great?

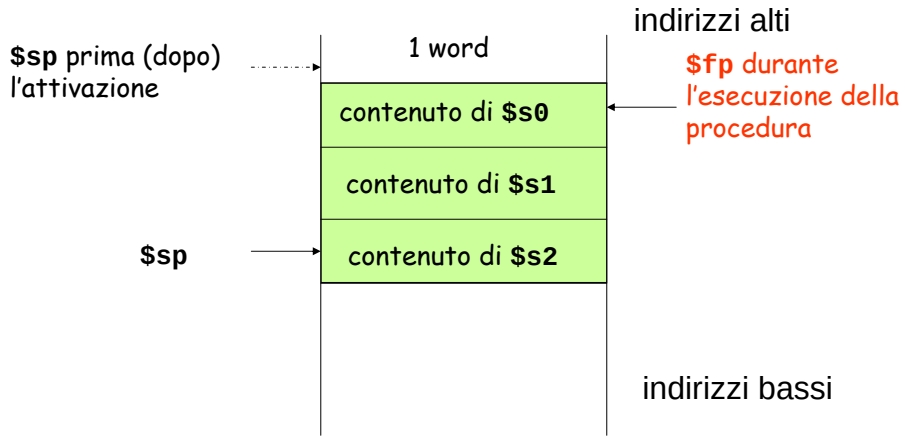
Stacking of Subroutine Calls & Returns and Environments:



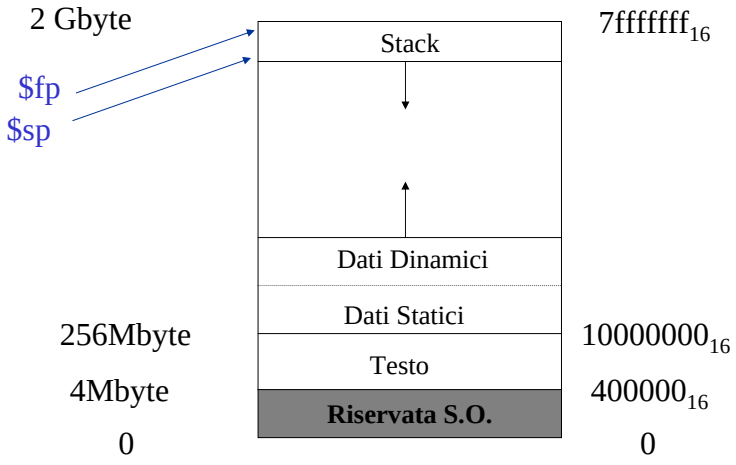
Some machines provide a memory stack as part of the architecture (e.g., VAX)

Sometimes stacks are implemented via software convention (e.g., MIPS)

Il frame pointer



Organizzazione logica della memoria



Gestione dello stack nel MIPS

- Lo stack (pila) è una struttura dati costituita da una coda LIFO (last-in-first-out)
- Lo stack cresce **da indirizzi di memoria alti verso indirizzi bassi**
- Il registro **`$sp`** contiene l'indirizzo dell'ultima locazione utilizzata in cima allo stack.
- L'inserimento di un dato nello stack **(operazione di push)** avviene **decrementando `$sp`** per allocare lo spazio ed eseguendo una sw per inserire il dato.
- Il prelevamento di un dato dallo stack **(operazione di pop)** avviene eseguendo una lw ed **incrementando `$sp`** (per eliminare il dato), riducendo quindi la dimensione dello stack.

- Tutto lo spazio in stack di cui ha bisogno una procedura **(record di attivazione)** viene *esplicitamente* allocato dal programmatore in una sola volta, all'inizio della procedura.

- Lo spazio riservato ad una procedura si trova tra i registri $sp e $fp.

Gestione dello stack nel MIPS

- Alla chiamata di procedura, lo spazio nello stack viene allocato **sottraendo** a **`$sp`** il numero di byte necessari:
  - Es:

```
addi $sp,$sp,-24   #alloca 24 byte nello stack
```

- Al rientro da una procedura il record di attivazione viene rimosso dalla procedura (deallocato) incrementando **`$sp`** della stessa quantità di cui lo si era decrementato alla chiamata
  - Es:

```
addi $sp, $sp,24   #dealloca 24 byte nello stack
```

- È necessario liberare lo spazio allocato per evitare di riempire tutta la memoria

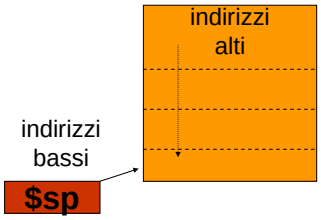
Esempio di gestione dello stack nel MIPS

- Per inserire elementi nello stack

```
sw $t0, offset($sp) # salvataggio di $t0
```

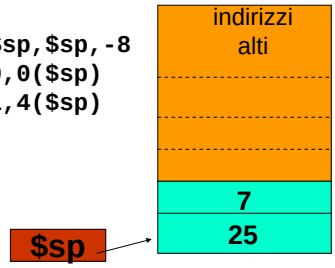
- Per recuperare elementi dallo stack

```
lw $t0, offset($sp) # ripristino di $t0
```




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

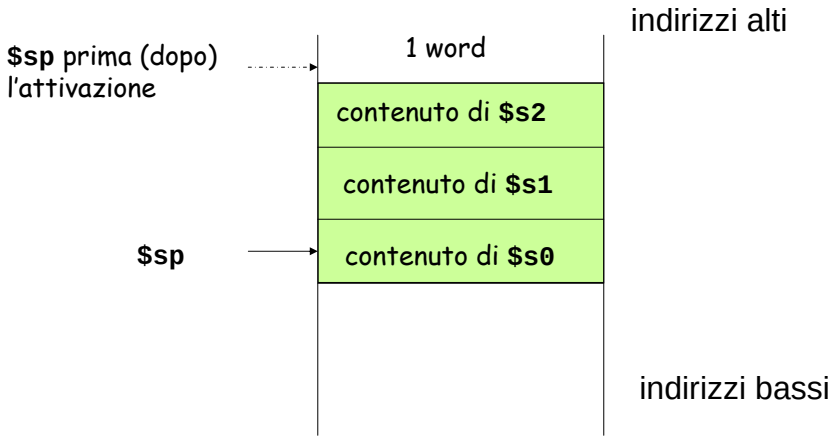
$t0 <- 25
$t1 <- 7



Esempio: somma

```
Somma_algebrica:
  addi $sp,$sp,-12          # alloca nello stack lo spazio per i 3 registri
  sw $s0, 8($sp)            # salvataggio di $s0
  sw $s1, 4($sp)            # salvataggio di $s1
  sw $s2, 0($sp)            # salvataggio di $s2

  add $s0, $a0, $a1         # $t0 ← g + h
  add $s1, $a2, $a3         # $t1 ← i + j
  sub $s2, $t0, $t1         # f ← $t0 - $t1

  add $v0, $s2, $zero               # restituisce f copiandolo nel reg. di ritorno $v0

  # ripristino del vecchio contenuto dei registri estraendolo dallo stack
  lw $s2, 0($sp)            # ripristino di $s0
  lw $s1, 4($sp)            # ripristino di $t0
  lw $s0, 8($sp)            # ripristino di $t1

  addi $sp, $sp, 12         # deallocazione dello stack per eliminare 3 registri
  jr $ra                            #ritorno al prog. chiamante
```

I registri nello stack



Sommario

Le procedure

Lo stack

La chiamata a procedura

Record di attivazione

- Una procedura è eseguita in uno spazio *privato* detto **record di attivazione**
  - area di memoria dove vengono allocate le variabili locali della procedura e i parametri
- Il programmatore assembly deve provvedere esplicitamente ad allocare/cedere lo spazio necessario (*frame di chiamata a procedura*) per:
  - Mantenere i valori passati come parametri alla procedura;
  - Salvare i registri che una procedura potrebbe modificare ma che al chiamante servono inalterati.
  - Fornire spazio per le variabili locali alla procedura.
- Quando sono permesse chiamate di procedura annidate, i record di attivazione sono allocati e rimossi come gli elementi di uno stack

MIPS: Software conventions for Registers

0	zero	constant 0
1	at	reserved for assembler
2	v0	expression evaluation &
3	v1	function results
4	a0	arguments
5	a1	
6	a2	
7	a3	
8	t0	temporary: caller saves
. . .		(callee can clobber)
15	t7	
16	s0	callee saves
. . .		(caller can clobber)
23	s7	
24	t8	temporary (cont'd)
25	t9	
26	k0	reserved for OS kernel
27	k1	
28	gp	Pointer to global area
29	sp	Stack pointer
30	fp	frame pointer (s8)
31	ra	Return Address (HW)

Caller = chiamante - callee = chiamato

Procedure foglia

- Procedura **foglia** è una procedura che ***non*** ha annidate al suo interno chiamate ad altre procedure
  - **non serve che salvi `$ra` (perché nessun altro lo modifica)**

- Nel caso di procedure foglia, il **chiamante** salva nello stack:
  - I registri temporanei di cui vuole salvare il contenuto di cui ha bisogno dopo la chiamata (**$t0-$t9,...).**
- Nel caso di procedure foglia, il **chiamato** alloca nello stack:
  - I registri non temporanei che vuole utilizzare **`($s0-$s8)`**
  - Strutture dati locali (es: array, matrici) e variabili locali della procedura che non stanno nei registri.

Lo stack pointer **`$sp`** è aggiornato per tener conto del numero di registri memorizzati nello stack; alla fine i registri vengono ripristinati e lo stack pointer riaggiornato.

Le stesse operazioni vengono eventualmente eseguite sul $fp.

Procedure intemedie

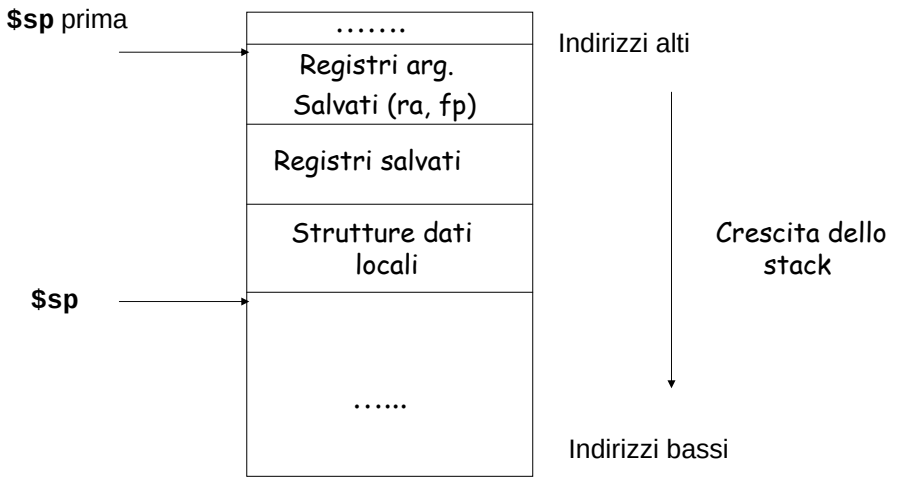
- Procedura **intermedia** è una procedura che contiene al suo interno (annidata) la chiamata ad almeno un'altra procedura
  deve salvare l'indirizzo di ritorno da `$ra` (perché verrà modificato dalla procedura da lei chiamata)

- Nel caso di procedure intermedie, il **chiamante** salva nello stack:
  - I registri temporanei di cui vuole salvare il contenuto di cui ha bisogno dopo la chiamata (**$t0-$t9,...).**
- Nel caso di procedure intermedie, il **chiamato** alloca nello stack:
  - I registri non temporanei che vuole utilizzare **`($s0-$s8)`**
  - Strutture dati locali (es: array, matrici) e variabili locali della procedura che non stanno nei registri.
  - Il registro di ritorno **$ra** ed eventualmente il registro $fp

Lo stack pointer **`$sp`** è aggiornato per tener conto del numero di registri memorizzati nello stack; alla fine i registri vengono ripristinati e lo stack pointer riaggiornato.

Le stesse operazioni vengono eventualmente eseguite sul $fp.

Record di attivazione.



Storia della chiamata

Chiamante (main):

- Fare spazio nello stack per memorizzare i registri di eventuale interesse nello stack (registri $t, registri $a) e memorizzarne il contenuto in stack.
- Mettere i parametri della procedura nei registri $a0, $a1, $a2, $a3 ed eventualmente in stack i parametri in eccesso.
- Trasferire il controllo alla procedura: definizione di un nome-etichetta per la procedura, es: **proc_name:**, ed esecuzione dell'istruzione *jal proc_name*.

Chiamato (procedura):

- Fare spazio nello stack per memorizzare i dati locali.
- Salvataggio dei registri di interesse nello stack (registri $ra, $fp, se si verificano delle chiamate ad altre procedure).
- Salvataggio dei registri varibile: $s, se utilizzati all'interno della procedura.

Prologo – record di attivazione

- Determinazione della dimensione del record di attivazione
- Per determinare la dimensione del record di attivazione si deve stimare lo spazio per:
  - registri degli argomenti (ra, fp)
  - registri di variabile da salvare (s)
  - registri per variabili locali.

NB I registri $t e $a vengono salvati in stack dal chiamante, non fanno parte del record di attivazione.

1) Allocazione dello spazio sullo stack => aggiornare il valore di **$sp:**
(lo stack pointer viene decrementato della dimensione prevista per la procedura)

addi $sp,$sp,-dim_record_attivaz

2) Salvataggio dei registri per i quali è stato allocato spazio nello stack:

sw reg,[dim_record_attivaz-N]($sp)

N (N ≥ 4) viene incrementato di 4 ad ogni salvataggio

Esempio di salvataggio dei registri

- Record di attivazione: 20 byte

addi $sp,$sp,-24

sw $s0, 20($sp) dim_record_attivazione - 4
sw $s1, 16($sp) dim_record_attivazione - 8
sw $s2, 12($sp) dim_record_attivazione – 12

sw $ra, 8($sp) dim_record_attivazione – 16

sw $t0, 4($sp) dim_record_attivazione - 20
sw $t1, 0($sp) dim_record_attivazione - 24

Corpo della procedura

- Stesura delle istruzioni per l'esecuzione delle funzionalità previste dalla procedura

```
add $s0, $a0, $a1        # $t0 ← g + h
add $s1, $a2, $a3        # $t1 ← i + j
sub $s2, $t0, $t1        # f ← $t0 - $t1

add $v0, $s2, $zero      # restituisce f copiandolo nel reg. di ritorno $v0
```


Epilogo

- Ripristino dei registri di interesse dallo stack (i registri $s e $f; $ra e $fp, se vengono utilizzati dalla procedura internamente).
- Restituzione dei parametri della procedura (dai registri $v0, $v1 e dallo stack).
- Eliminare lo spazio dello stack in cui sono stati memorizzati i dati locali.
- Trasferire il controllo al programma chiamante.

- Ripristino dei registri salvati:

```
lw reg, dim_record_attivaz - N($sp)
```

- Rimozione dello spazio allocato sullo stack:

```
addi $sp,$sp,dim_record_attivaz.
```

- Restituzione del controllo al chiamante:

```
jr $ra
```

Il flusso di esecuzione riprende dall'istruzione successiva a quella che ha chiamato la procedura.


Procedura chiamante

- Salva i registri temporanei: $t, $a, di cui vuole preservare il contenuto.
- Copia eventuali argomenti in numero superiore a quattro nello stack (oltre a quelli contenuti nei registri **`$a0-$a3`**)
- Esegue:

```
jal  proc_name
```



Esempio

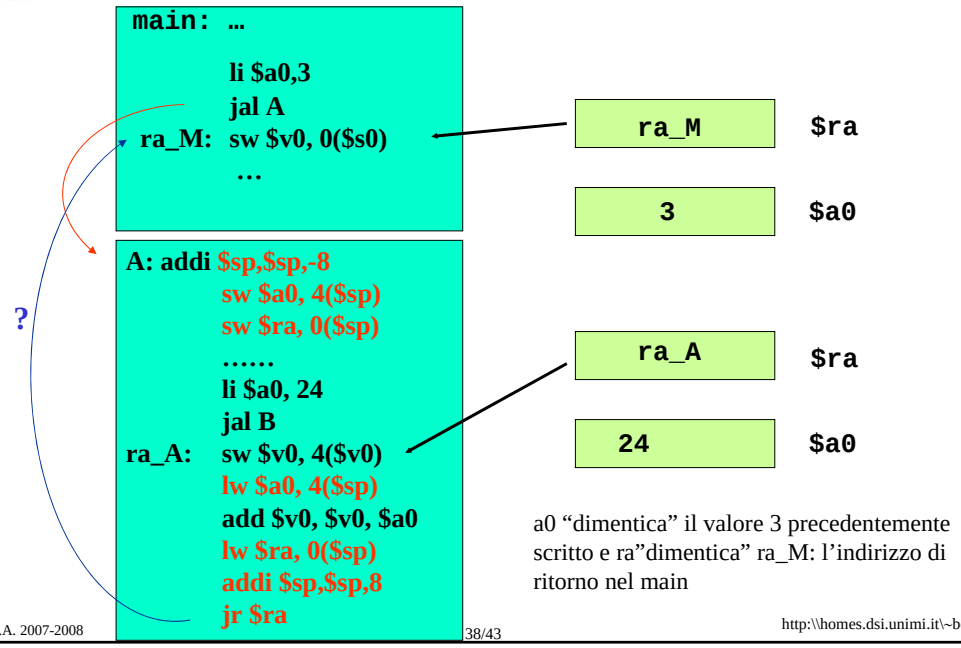
```
# Programma che stampa una stringa mediante procedura print
# Voglio utilizzare $s0 all'interno della procedura chiamata.
        .data
str:    .asciiz "benvenuti in xSPIM\n "
        .text
        .globl main
main:   la $s1, str             # $a0 ← ind. stringa da stampare
        li $s0, 16              # $v0 ← valore 16 da preservare
        move $a1, $s1
        jal stampa
        add $s0, $s0, $v0       # Devo utilizzare $v0 (il valore 16)
        li $v0, 10              # $v0 ← codice della exit
        syscall                 # esce dal programma

stampa: addi $sp, $sp,-4        # allocazione dello stack
        sw $s0, 0($sp)          # salvo $s0 che vado a
                                # modificare nella print
        li $s0, 4
        move $v0, $s0           # $v0 ← codice di print_string
        syscall                 # stampa della stringa
        lw $s0, 0($sp)          # ripristina il reg. $s0
        addi $sp,$sp,4          # deallocazione dello stack
        jr $ra
```

Procedure intermedie

- Sono procedure che richiamano al loro interno altre procedure (non sono procedure foglia)
- Devono salvare nello stack un ambiente più ampio

- Rispetto alle procedure foglia, in una procedura intermedia devono essere salvati anche:
  - i parametri di input della procedura ($a0, $a1, $a2, $a3) se vengono riutilizzati all'interno della procedura intermedia.
  - l'indirizzo di ritorno ($ra)
    - la procedura chiamata all'interno di un'altra riscrive il contenuto di **$ra.**

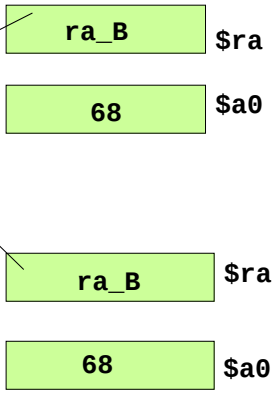
Procedure annidate



a0 "dimentica" il valore 3 precedentemente scritto e ra"dimentica" ra_M: l'indirizzo di ritorno nel main

Procedure annidate

```
B:      addi $sp,$sp,-4
        sw $ra, 0($sp)
        li $a0, 68
        jal C
ra_B:   sw $v0,0($v0)
        lw $ra, 0($sp)
        addi $sp, $sp, 4
        jr $ra

C:      …
        …
        move $t0, $a0
        add $v0, $zero, $t0
        …
        jr $ra
```



Procedure intermedie

- Procedura **intermedia** è una procedura che ***ha*** annidate al suo interno chiamate ad altre procedure.

- Nel caso di procedure intermedia, il **chiamante** salva nello stack:
  - I registri temporanei di cui vuole salvare il contenuto di cui ha bisogno dopo la chiamata (**$t0-$t9,...)**
  - I registri argomento (**$a0-$a3,...)** nel caso in cui il loro contenuto debba essere preservato (sono considerati registri temporanei).
  - Eventuali argomenti aggiuntivi oltre a quelli che possono essere contenuti nei registri **$a0-$a3.**
  - Il contenuto dei registri $v0, $v1 nel caso in cui il contenuto debba servire.
- Nel caso di procedure foglia, il **chiamato** alloca nello stack:
  - I registri non temporanei che vuole utilizzare **($s0-$s8)**
  - Strutture dati locali (es: array, matrici) e variabili locali della procedura che non stanno nei registri.
  - I registri della procedura ($ra, $fp).

Lo stack pointer **$sp** è aggiornato per tener conto del numero di registri memorizzati nello stack; alla fine i registri vengono ripristinati e lo stack pointer riaggiornato.

Esempio - C

Eseguo s = (a+b) – (c+d) utilizzando due procedure annidate:

```
main
{
,…
res = opera(a,b,c,d)
….
}

int opera(int a, int b, int c, int d)
{
t0 = a + b;
t1 = c + d;
res = diff(t0, t1);
return(ret);
}

int diff (t0, t1)
{   ret = t0 – t1;
   return(ret);
}
```


Esempio - Assembly

Suppongo a, b, c, d contenuti nei registri $s0, …. $s3; il risultato sarà inserito in $s5

```
mov $a0, $s0
mov $a1, $s1
mov $a2, $s2
mov $a3, $s3,
jal opera
mov $s5, $v0

opera:
addi $sp, $sp, -4
sw $ra, 0($sp)
add $a0, $a0, $a1
add $a2, $a2, $a3
jal diff
lw $ra, 0($sp)
jr $ra

diff:
sub $v0, $a0, $a1
jr $ra
```

Sommario

Le procedure

Lo stack

La chiamata a procedura