Lezione 14

Assembly IV
Le procedure

Proff. A. Borghese, F. Pedersini

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

(Patterson-Hennessy: cap. 2)

Chiamata a procedura: esempio

Procedura chiamante

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

Procedura chiamata

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

- Cosa succede?
  - Cambio di flusso
  - Passaggio informazioni
- *Come si fa in Assembly?*

Chiamata a procedura

Due ***attori:*** **chiamante** e **chiamata**

- La procedura **chiamante** (**caller**) deve:
  1. Predisporre i parametri di ingresso in un posto accessibile alla procedura chiamata
  2. Trasferire il controllo alla procedura chiamata

- La procedura **chiamata** (**callee**) deve:
  1. Allocare lo spazio di memoria necessario all'esecuzione ed alla memorizzazione dei dati: → **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 a procedura: `jal`

- Necessaria un'istruzione apposita che:
  1. **cambia il flusso** di esecuzione (salta alla procedura) e
  2. salva **l'indirizzo di ritorno**
     (istruzione successiva alla chiamata a procedura)

`jal` (jump and link)

Sintassi: `jal Indirizzo_Procedura`

1. Salta all'indirizzo con etichetta `Indirizzo_Procedura`
2. Memorizza il **valore corrente del Program Counter in** `$ra`
   (indirizzo dell'istruzione successiva: **$(Program Counter) + 4**)

- **La procedura chiamata**, come **ultima istruzione**, esegue: `jr $ra`
  per effettuare il salto all'indirizzo di ritorno della procedura

Procedure foglia

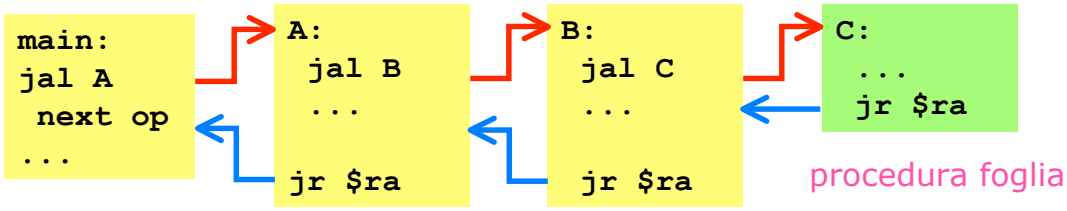
Procedura foglia: è una procedura che non ha annidate al suo interno chiamate ad altre procedure

Non è necessario salvare **`$ra`**, perché nessun altro lo modifica

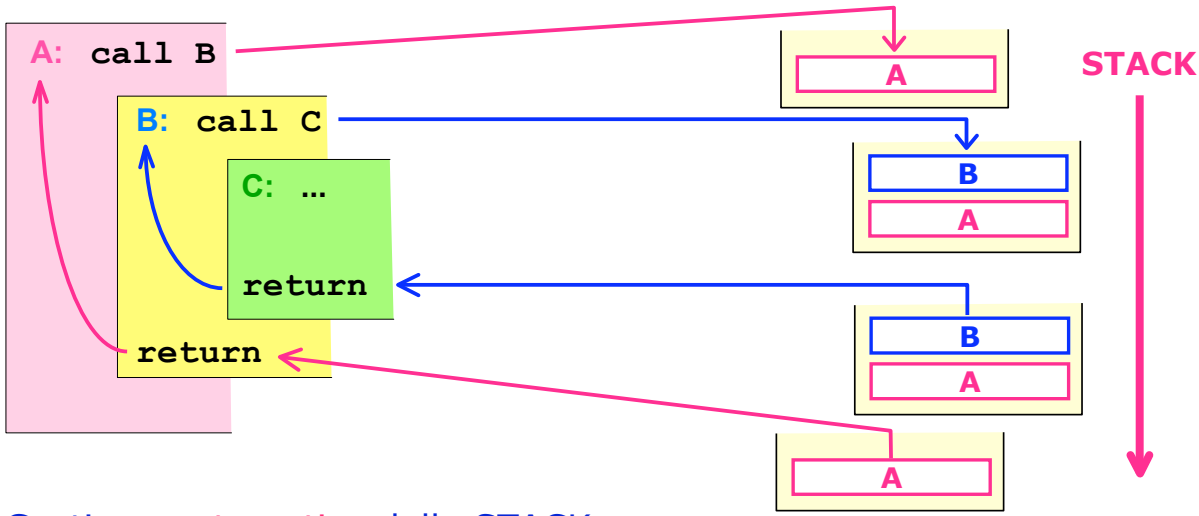
NON sono procedure foglia:

- Procedure annidate, Procedure recursive
- Procedure che chiamano altre sotto-procedure

E' necessario salvare **`$ra`** nello stack, perchè la procedura chiamata lo modifica a sua volta!



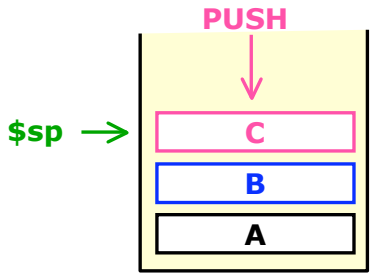
Gestione dello stack per ritorno da procedure



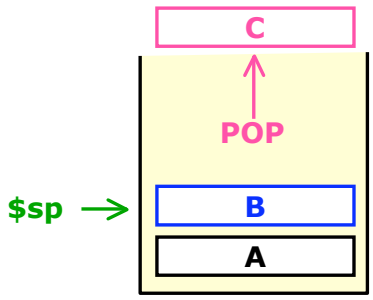
- Gestione automatica dello STACK:
  - Stack gestito automaticamente dall'architettura, mediante circuiti dedicati nella CPU (es. VAX)
- **MIPS**: gestione manuale dello STACK:
  - Implementata secondo convenzioni software

Lo stack

- Struttura dati: **coda LIFO** (*last-in-first-out*)
- Inserimento dati nello stack: ***PUSH***
- Prelievo dati dallo stack: ***POP***



- Necessario un puntatore alla cima dello stack per salvare i dati
  - Il registro **`$sp`** (**stack pointer** – puntatore allo stack) contiene l'indirizzo della cima dello stack (Top of Stack, TOS)
  - **`$sp`** deve essere aggiornato ogni volta che viene inserito o estratto un dato dallo stack



Gestione dello *stack* in *MIPS*

Stack MIPS:

- Lo stack cresce da indirizzi di memoria alti verso indirizzi bassi
- Il registro **`$sp`** contiene l'indirizzo della prima locazione libera in **cima allo stack**

- Inserimento di un dato nello stack – **PUSH:**
  - Si decrementa **`$sp`** per allocare lo spazio e
  - Si esegue **`sw`** per inserire il dato nello stack

- Prelevamento di un dato dallo stack – **POP:**
  - Si esegue **`lw`** per recuperare il dato dallo stack
  - Si incrementa **`$sp`** (per eliminare il dato), riducendo la dimensione dello stack.

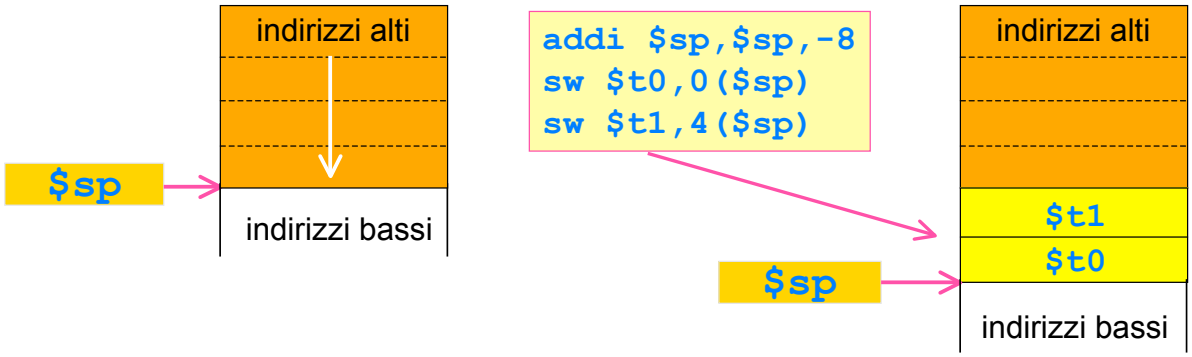
Gestione dello stack MIPS

- Per inserire elementi nello stack (push)

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

- Per recuperare elementi dallo stack (pop)

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



Allocazione dei registri

- Convenzioni per l'allocazione dei registri:
- **$ra** **return address**
  - registro per memorizzare l'indirizzo della prima istruzione del chiamante, da eseguire al termine della procedura
- **$a0÷$a3 ($f12÷$f15)** registri **argomento**
  - usati dal chiamante per il passaggio dei parametri
  - Se i parametri sono più di 4, si usa la memoria (stack)
- **$v0,$v1 ($f0÷$f3)** registri **valore**
  - usati dalla procedura per memorizzare i valori di ritorno

Convenzioni chiamate a procedura

- Il programma **chiamante**:
  - Inserisce i parametri da passare alla procedura nei **registri argomento**: `$a0-$a3`
  - Salta alla procedura mediante l'istruzione `jal address` e salvare il valore di `(PC+4)` nel registro `$ra`

- La procedura **chiamata**:
  - Esegue il compito richiesto
  - Memorizza il risultato nei registri `$v0,$v1`
  - Restituisce il controllo al chiamante con l'istruzione `jr $ra`

Record di attivazione

La procedura chiamata usa gli stessi registri usati dal chiamante.
Al ritorno, la chiamante trova registri modificati!!

- **Record di attivazione (MIPS):**
  tutto lo **spazio nello stack** di cui ha bisogno una procedura.
  - ASSEMBLY: lo spazio per memorizzare il record di attivazione viene *esplicitamente* allocato dal programmatore

- *Quando si chiama una procedura **i registri utilizzati dal chiamato** vanno:*
  - salvati nello stack
  - ripristinati alla fine dell'esecuzione della procedura

MIPS: gestione Record di Attivazione

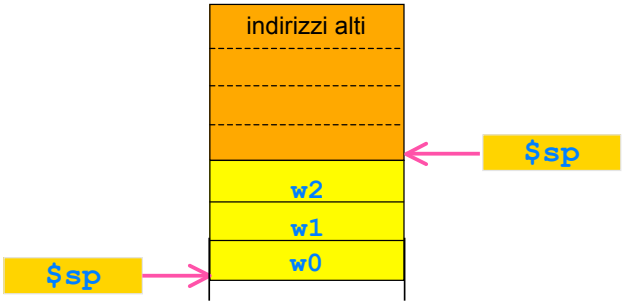
- **All'inizio della procedura**, il record di attivazione viene **allocato** decrementando **$sp (PUSH)**

```
addi $sp, $sp,-12     # alloca 12 byte
                      # nello stack
```

- **Al termine dalla procedura**, il record di attivazione viene **rimosso** incrementando **$sp** della stessa quantità **(POP)**

```
addi $sp, $sp, 12     # dealloca 12 byte
                      # dallo stack
```

- Tale spazio esiste soltanto **durante l'esecuzione** della procedura.



Esempio

C:

```
int somma_algebrica (int g, int h, int i, int j)
{
        int f;

        f = (g + h) - (i + j);
        return f;
}
```

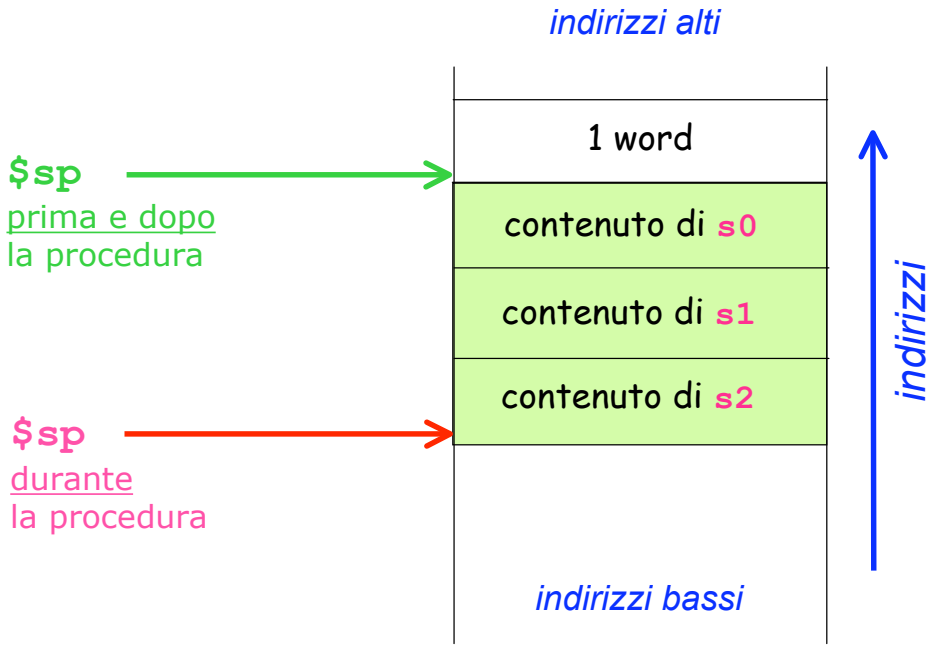
Assembly:

```
# g,h,i,j associati a $a0 … $a3, f associata ad $s0
# Il calcolo di f richiede 3 registri: $s0, $s1, $s2
# E' necessario salvare i 3 registri nello stack

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
```

Esempio (cont.)

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

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

# ripristino del vecchio contenuto dei registri
# estraendoli dallo stack

lw $s2, 0($sp)             # ripristino di $s2
lw $s1, 4($sp)             # ripristino di $s1
lw $s0, 8($sp)             # ripristino di $s0

addiu $sp, $sp, 12         # deallocazione dello stack
                           # per eliminare 3 registri

jr $ra                     #ritorno al prog. chiamante
```


Esempio (cont.)



MIPS: convenzioni di salvataggio registri

- Per evitare di salvare inutilmente i registri, sono divisi in due classi:
  - **Registri temporanei**:

    `$t0, …, $t9 ($f4...$f11, $f16...$f19)`
  - il cui contenuto **non è salvato dal chiamato** nello stack (viene salvato dal chiamante se serve);
  - **Registri non-temporanei**:

    `$s0, …, $s8 ($f20, …, $f31)`
  - il cui contenuto è **salvato dal chiamato** nello stack **se utilizzato**

Si può eliminare il salvataggio dei registri `$s0` e `$s1`, utilizzando al loro posto i registri `$t`, ed eliminare l'utilizzo del registro `$s2`:

```
sub $s2, $s0, $s1    # f ← $t0 - $t1
```

può diventare:

```
sub $v0, $t0, $t1
```


Gestione delle procedure foglia

- **Tutte** le procedure → 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.
- Procedure **foglia** → il **chiamato** salva 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.
- Procedure **non-foglia** → il **chiamato** salva nello stack anche:
  - I registri argomento della procedura,
  - `$ra ($fp)`

Sintesi:

- **Record di attivazione:**
  spazio privato della procedura eseguita
  - 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 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 dallo stack.

Struttura di una procedura

- Ogni procedura è composta **3 SEZIONI** consecutive:
- **Prologo**
  - Acquisire le risorse necessarie per memorizzare i dati interni alla procedura ed il salvataggio dei registri.
  - Salvataggio dei registri di interesse.
- **Corpo**
  - Esecuzione della procedura vera e propria
- **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.

Prologo

- Determinazione della dimensione del record di attivazione
  - Stimare lo spazio per: registri da salvare, argomenti, variabili locali.
- Allocazione dello spazio in stack: aggiornamento valore di `$sp:`

```
addi $sp, $sp, -dim_record_attivaz
# lo stack pointer viene decrementato
# della dimensione del record di attivazione
# prevista per la procedura
```

- Salvataggio dei registri per i quali è stato allocato spazio in stack:

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

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

Esempio: record di attivazione: 3 registri → **12 byte**

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


Epilogo

- Restituzione dei parametri della procedura
  - dai registri `$v0,$v1` (e/o stack)
- Ripristino dei registri dallo stack
  - `$s` e `$f` ; `$ra` e `$fp` per procedure non–foglia

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

- Liberare lo spazio nello stack in cui sono stati memorizzati i dati locali.

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

- Trasferire il controllo al programma chiamante:

```
jr $ra
```