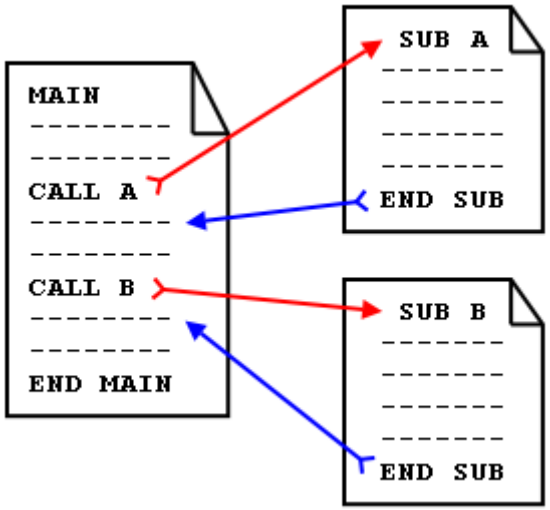
ARCHITETTURA DEL SET DI ISTRUZIONI

Chiamata a Procedure – I

Michele Favalli

Procedure

I programmatori di alto livello ricorrono spesso a procedure o funzioni per la miglior comprensibilità del programma, e per facilitare il suo riutilizzo.



Esempio di statements in C

```
Main()
{
int a;
a=calcola_somma(2,3);
…
}
```

Un principio importante: *non deve rimanere traccia dell'esecuzione di una procedura nello stato del sistema, che deve tornare quello precedente l'invocazione della procedura!*

Unica perturbazione: i valori di ritorno della procedura sono «magicamente» disponibili in locazioni pre-fissate!

In assembly si ha un overhead per il processore che nei linguaggi ad alto livello non viene visto dal programmatore

Overhead per il microprocessore

Programma

Mettere i parametri in un posto dove la procedura li possa leggere

Procedura

Programma

Procedura

Trasferire il controllo alla procedura

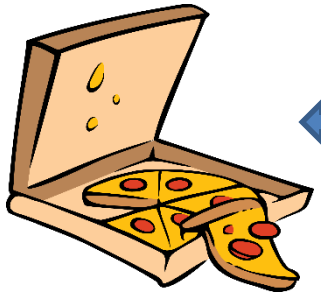
Procedura

Acquisire le informazioni memorizzate

Svolgere il lavoro

Procedura

Overhead per il microprocessore



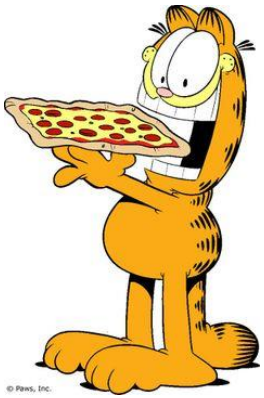
Procedura

Deposita le informazioni di ritorno in un posto da cui il programma chiamante possa recuperarle

Programma

Procedura

Ritorno del controllo al chiamante, **nello stesso punto in cui è stata effettuata la chiamata**



ATTENZIONE!
Una procedura può essere chiamata da diverse parti di un programma!

In pratica…

4 registri per passaggio argomenti **($a0-$a3**)

1 registro che punta all'indirizzo della prossima istruzione del chiamante **$ra**

2 registri per i valori di ritorno **($v0-$v1)**

Programma

```
#include <stdio.h>
int add(int a,int b);
int main(){
     ..............
    sum=add(num1, num2);
     ..............
 }
```

Procedura

```
int add(int a, int b)
{
 int add;
 ..............
return add;
}
```

La chiamata a procedura **add(num1,num2)** viene eseguita dall'istruzione speciale **jump-and-link**: **jal** *ProcedureAddress* che salta all'indirizzo della procedura e salva la prossima istruzione del chiamante (**PC**+4) in **$ra**

In pratica…

4 registri per passaggio argomenti **($a0-$a3**)

1 registro che punta all'indirizzo della prossima istruzione del chiamante
$ra

2 registri per i valori di ritorno
($v0-$v1)

Programma

……..
Metti i parametri in $a0-$a3
jal add
……..

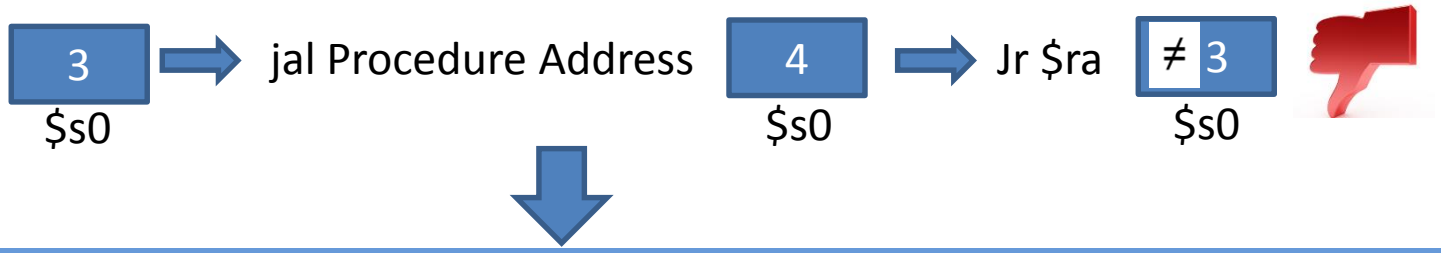
Procedura

Leggi parametri da $a0-$a3
…….
Metti i valori di ritorno in $v0-$v1
jr $ra

L'istruzione **jr (jump register)** esegue un salto incondizionato all'indirizzo nel registro **$ra** che **viene chiamato «return address register»**

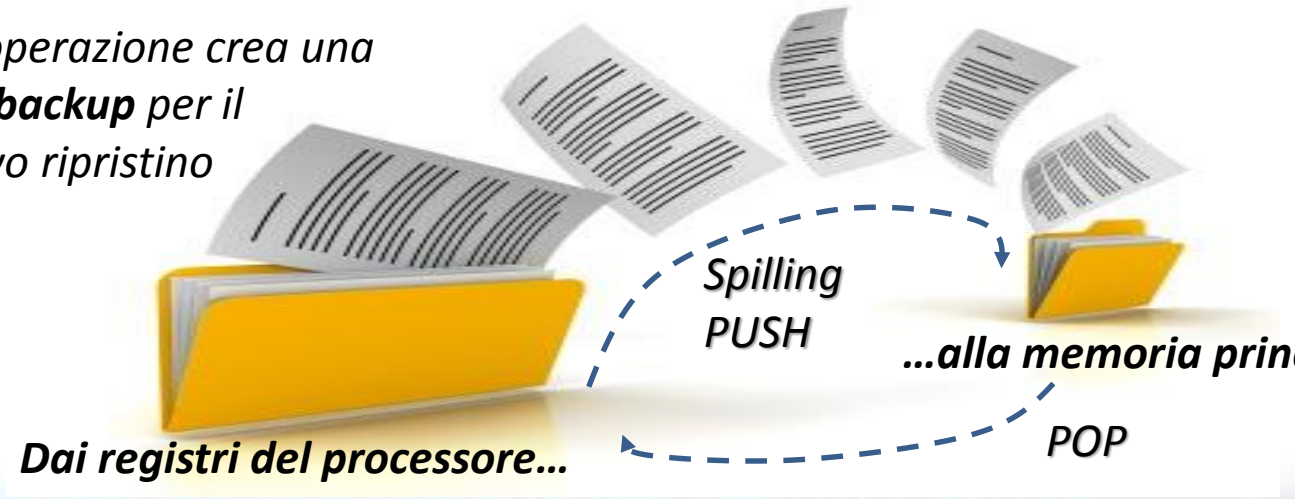
Obiezioni

La procedura a sua volta usa registri, magari già in uso nel programma principale



E' necessario fare lo «spilling» in memoria principale dei registri utilizzati nell'invocazione ed esecuzione della procedura.

*Questa operazione crea una copia di **backup** per il successivo ripristino*

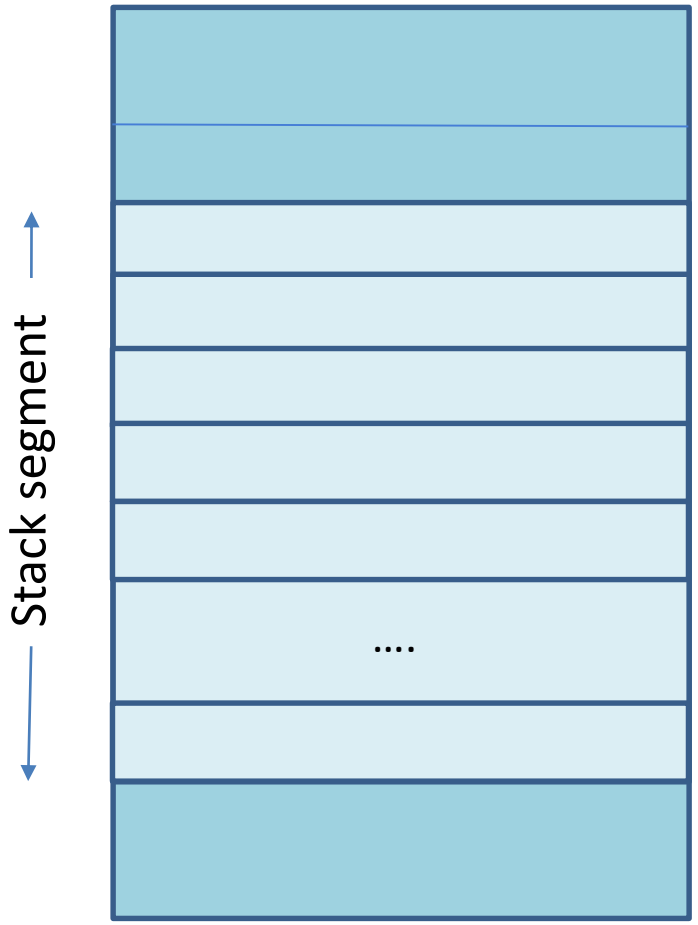


Per facilitare PUSH e POP, una porzione della memoria viene identificata e «gestita» in modo particolare (STACK)

Lo stack

- Una coda Last-In First-Out è la struttura di spilling ideale, chiamata **stack**
  - **push:** inserimento di un dato
  - **pop:** estrazione di un dato
- Dal punto di vista dei linguaggi ad alto livello lo stack può essere gestito in vari modi (lista con inserimento ed eliminazione in testa, vettore), ma a noi serve una gestione più semplice possibile
- **Nel nostro caso, lo stack risiede in un segmento (una porzione di locazioni contigue) della memoria principale.**
- **L'architettura MIPS alloca un registro appositamente per lo stack pointer: $sp**
- **Tale registro contiene l'indirizzo del top dello stack**

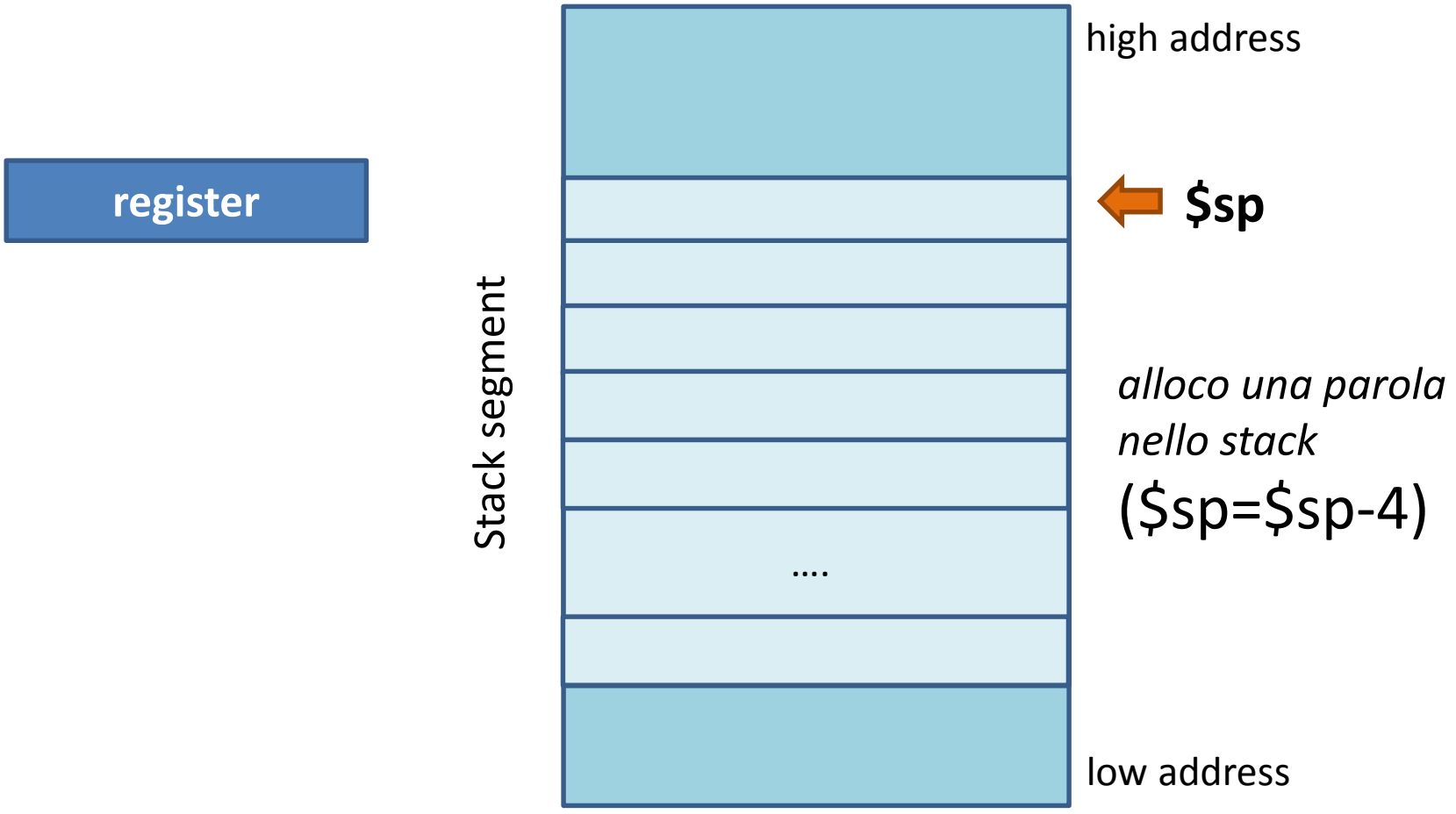
Lo stack dell'architettura MIPS



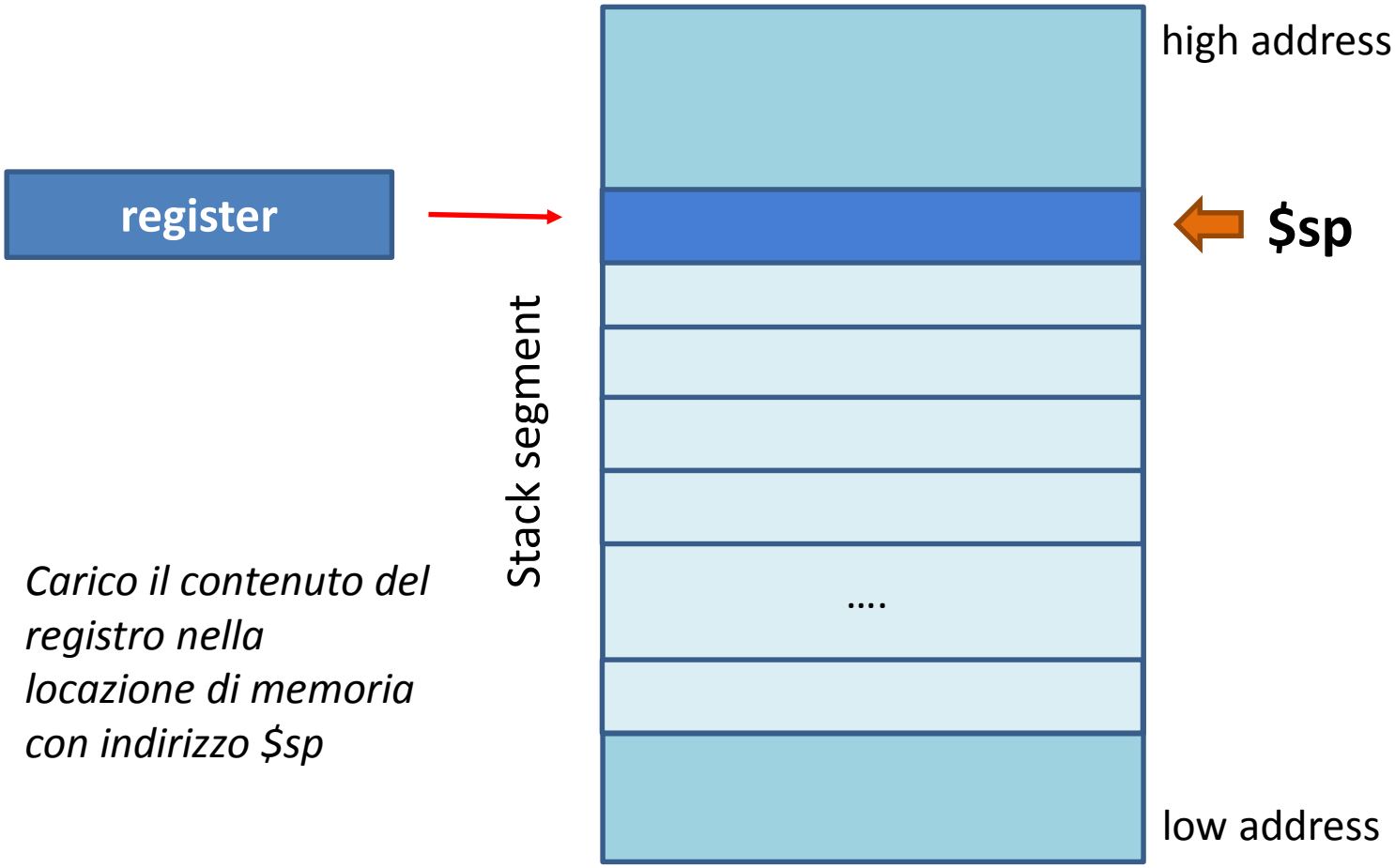
Come convenzione, lo stack viene riempito a partire da un indirizzo alto

La situazione è quella in cui lo stack è vuoto

Lo stack dell'architettura MIPS: push



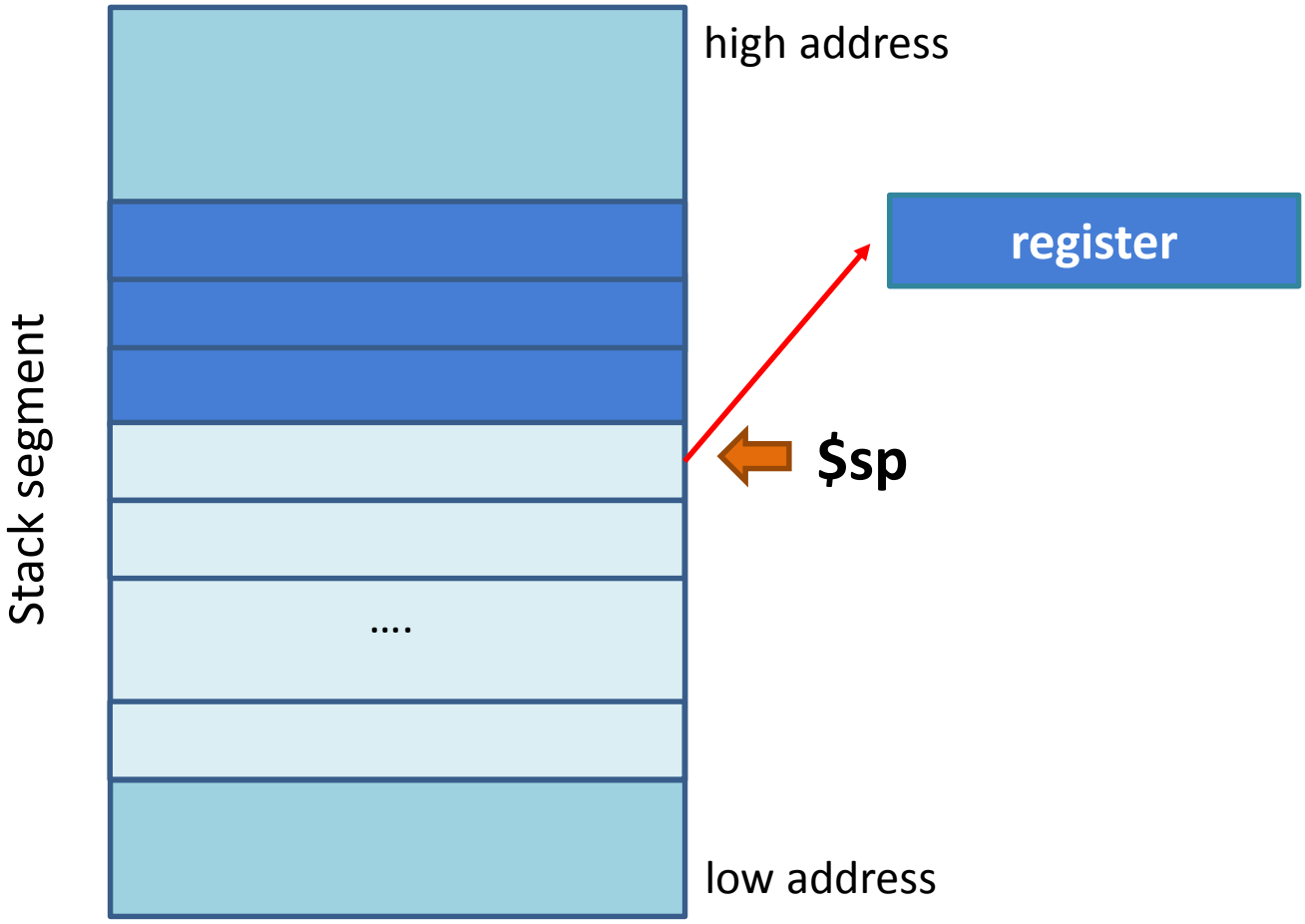
Lo stack dell'architettura MIPS: push



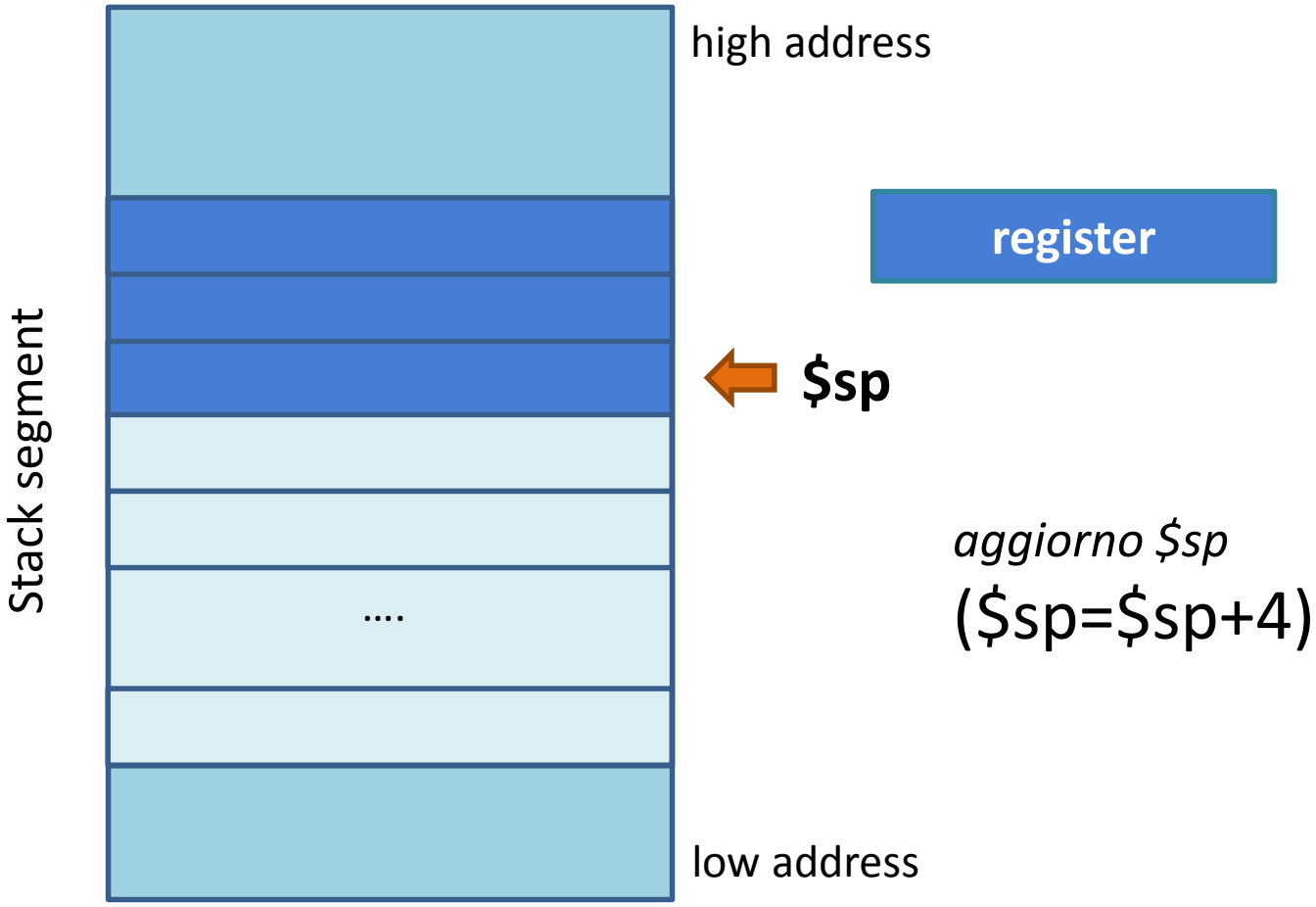
Carico il contenuto del registro nella locazione di memoria con indirizzo $sp

Lo stack dell'architettura MIPS: pop

Lo stack dell'architettura MIPS: pop



Lo stack dell'architettura MIPS: pop



Es.: compilazione di una procedura

- ***I parametri g, h, i, j saranno disponibili attraverso i registri $a0, $a1, $a2, $a3***
- ***La variabile f potrebbe essere associata dal compilatore al registro $s0.***

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

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

Es.: compilazione di una procedura

La procedura utilizzerà 3 registri, il cui contenuto deve pertanto essere salvato nello stack:

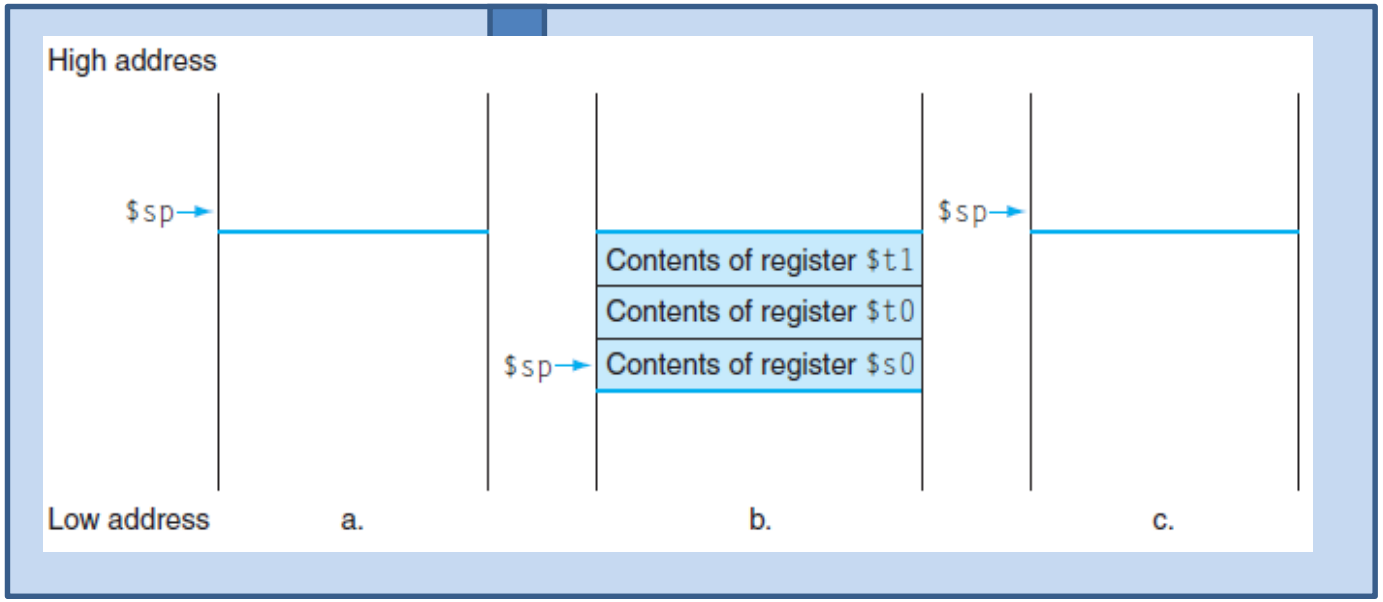
- Il ***registro $t0***, che sarà utilizzato per memorizzare il termine intermedio *«g+h»*
- Il ***registro $t1***, che sarà utilizzato per memorizzare il termine intermedio *«i+j»*
- Il ***registro $s0***, che sarà utilizzato per memorizzare la variabile locale *«f»*

Si fa l'ipotesi che tali registri siano utilizzati dal programma principale e che quindi il loro contenuto debba essere conservato

Es.: compilazione di una procedura

Chi si occupa di salvare tali registri?
La procedura che sa quali registri utilizza

La sezione di stack allocata da una procedura prende il nome di FRAME



Es.: compilazione di una procedura

Label della procedura

```
leaf_example:
```

Salvataggio dei registri nello stack

```
addi $sp,$sp,-12 # adjust stack to make room for 3 items
sw  $t1, 8($sp)  # save register $t1 for use afterwards
sw  $t0, 4($sp)  # save register $t0 for use afterwards
sw  $s0, 0($sp)  # save register $s0 for use afterwards
```

Esecuzione del corpo della procedura

```
add $t0,$a0,$a1 # register $t0 contains g + h
add $t1,$a2,$a3 # register $t1 contains i + j
sub $s0,$t0,$t1 # f = $t0 - $t1, which is (g + h)-(i + j)
```

Salva il valore di ritorno nel registro apposito $v0

```
add $v0,$s0,$zero # returns f ($v0 = $s0 + 0)
```

Ripristino del valore dei registri.

```
lw  $s0, 0($sp) # restore register $s0 for caller
lw  $t0, 4($sp) # restore register $t0 for caller
lw  $t1, 8($sp) # restore register $t1 for caller
addi $sp,$sp,12 # adjust stack to delete 3 items
```

Salto incondizionato al valore di ritorno

```
jr  $ra    # jump back to calling routine
```

Es.: compilazione di una procedura

Label della procedura

```
leaf_example:
```

Salvataggio dei registri nello stack

```
addi $sp,$sp,-12 # adjust stack to make room for 3 items
sw  $t1, 8($sp)  # save register $t1 for use afterwards
sw  $t0, 4($sp)  # save register $t0 for use afterwards
sw  $s0, 0($sp)  # save register $s0 for use afterwards
```

Esecuzione del corpo della procedura

```
add $t0,$a0,$a1 # register $t0 contains g + h
add $t1,$a2,$a3 # register $t1 contains i + j
sub $s0,$t0,$t1 # f = $t0 - $t1, which is (g + h)-(i + j)
```

Salva il valore di ritorno nel registro apposito $v0

```
add $v0,$s0,$zero # returns f ($v0 = $s0 + 0)
```

Ripristino del valore dei registri.

```
lw  $s0, 0($sp) # restore register $s0 for caller
lw  $t0, 4($sp) # restore register $t0 for caller
lw  $t1, 8($sp) # restore register $t1 for caller
addi $sp,$sp,12 # adjust stack to delete 3 items
```

Salto incondizionato al valore di ritorno

```
jr  $ra    # jump back to calling routine
```

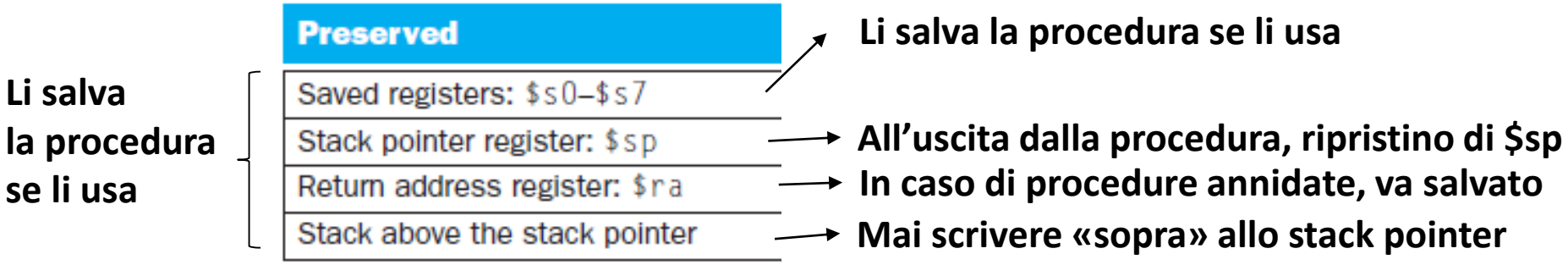
La procedura non lascia tracce del suo «passaggio» se non per $v0

Ottimizzazione

Come minimizzare gli spilling di registri in memoria?

Convenzione:

- I **registri $t0-$t9** sono registri temporanei che NON devono necessariamente essere preservati dalla procedura chiamata.
- I **registri $s0-$s7** sono registri che devono invece ESSERE PRESERVATI attraverso una chiamata a procedura.
- La procedura preserva **$sp** aggiungendovi ogni volta la stessa quantità che vi sottrae.



Assunzioni sull'integrità dei registri attraverso una chiamata a procedura, che LA PROCEDURA CHIAMATA deve rispettare

Ottimizzazione

Come minimizzare gli spilling di registri in memoria?

Convenzione:

- I **registri $t0-$t9** sono registri temporanei che NON devono essere preservati dalla procedura chiamata.
- I **registri $s0-$s7** sono registri che devono invece ESSERE PRESERVATI attraverso una chiamata a procedura.
- La procedura preserva **$sp** aggiungendovi ogni volta la stessa quantità che vi sottrae.

Li salva sullo stack Il CHIAMANTE se necessario, prima della «jal»

Not preserved
Temporary registers: $t0–$t9
Argument registers: $a0–$a3
Return value registers: $v0–$v1
Stack below the stack pointer

Se si vuol preservare il contenuto di un registro non preservato, occorre che il CHIAMANTE faccia la push esplicita sullo stack (prima di «jal»).

Complicazioni

Il Frame di una Procedura sullo Stack non serve solamente per salvare il valore dei registri da preservare, ma tipicamente anche per:

- ❑ allocazione di variabili locali alla procedura
- ❑ garantire la consistenza dell'esecuzione in caso di chiamate ricorsive a subroutines *(nested procedures)*
- ❑ quanto visto finora vale per procedure (leaf) che non chiamano altre procedure
- ❑ passare un numero di argomenti > 4