05 – Laboratorio 04

Procedure
Stack
Chiamata a procedure

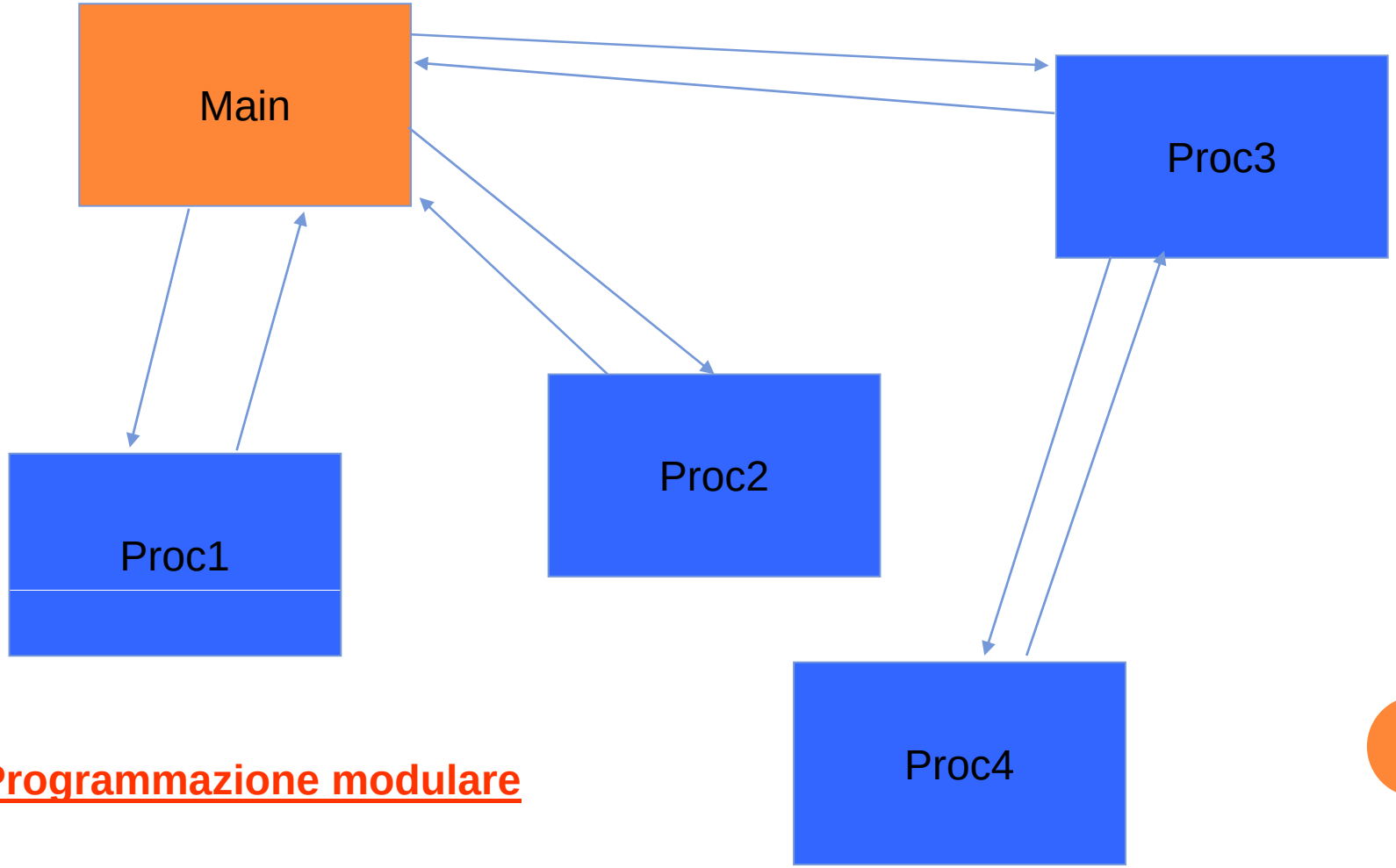
I. Frosio

15 April 2011

SOMMARIO

- Procedure
- Stack
- Chiamata a procedure

PERCHÈ LE PROCEDURE?



Programmazione modulare

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).

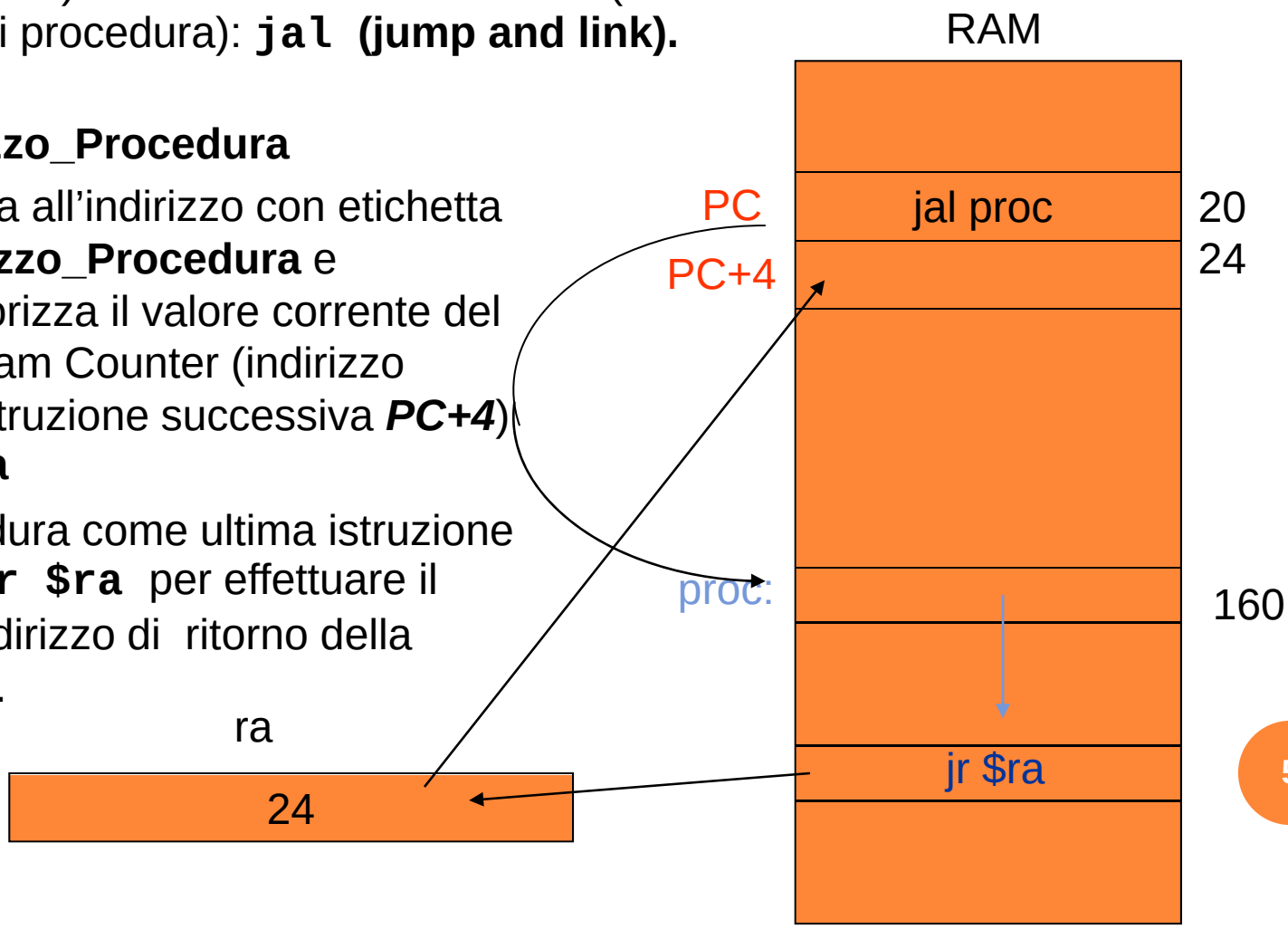
MECCANISMO DI CHIAMATA: TRASFERIMENTO DEL CONTROLLO

- 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.



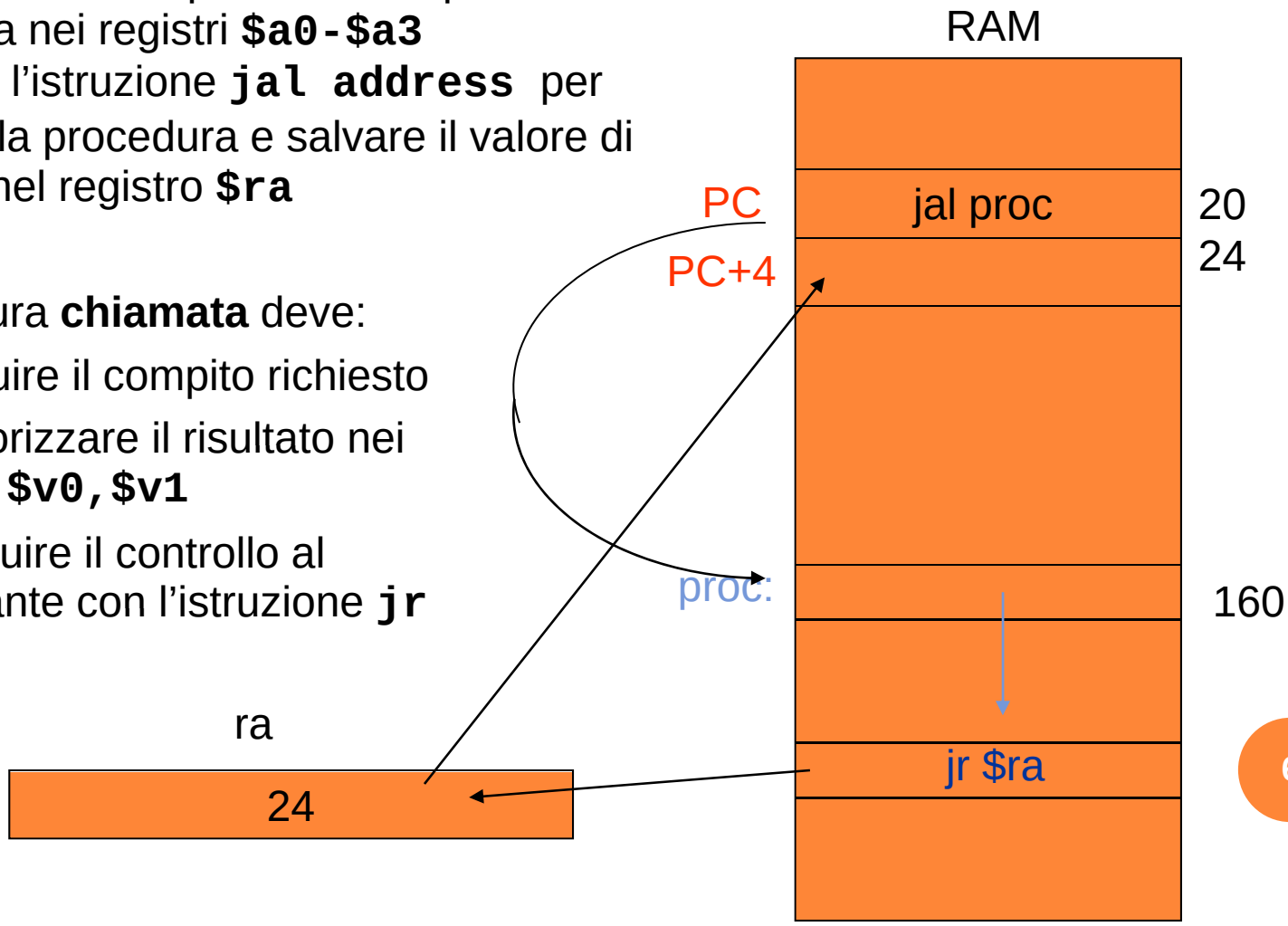
MECCANISMO DI CHIAMATA: TRASFERIMENTO DEI DATI

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`**



USO DEI REGISTRI NELLE CHIAMATE A PROCEDURE

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

ESEMPIO

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

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

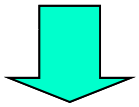
f,g → $s0, $s1

```
            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).

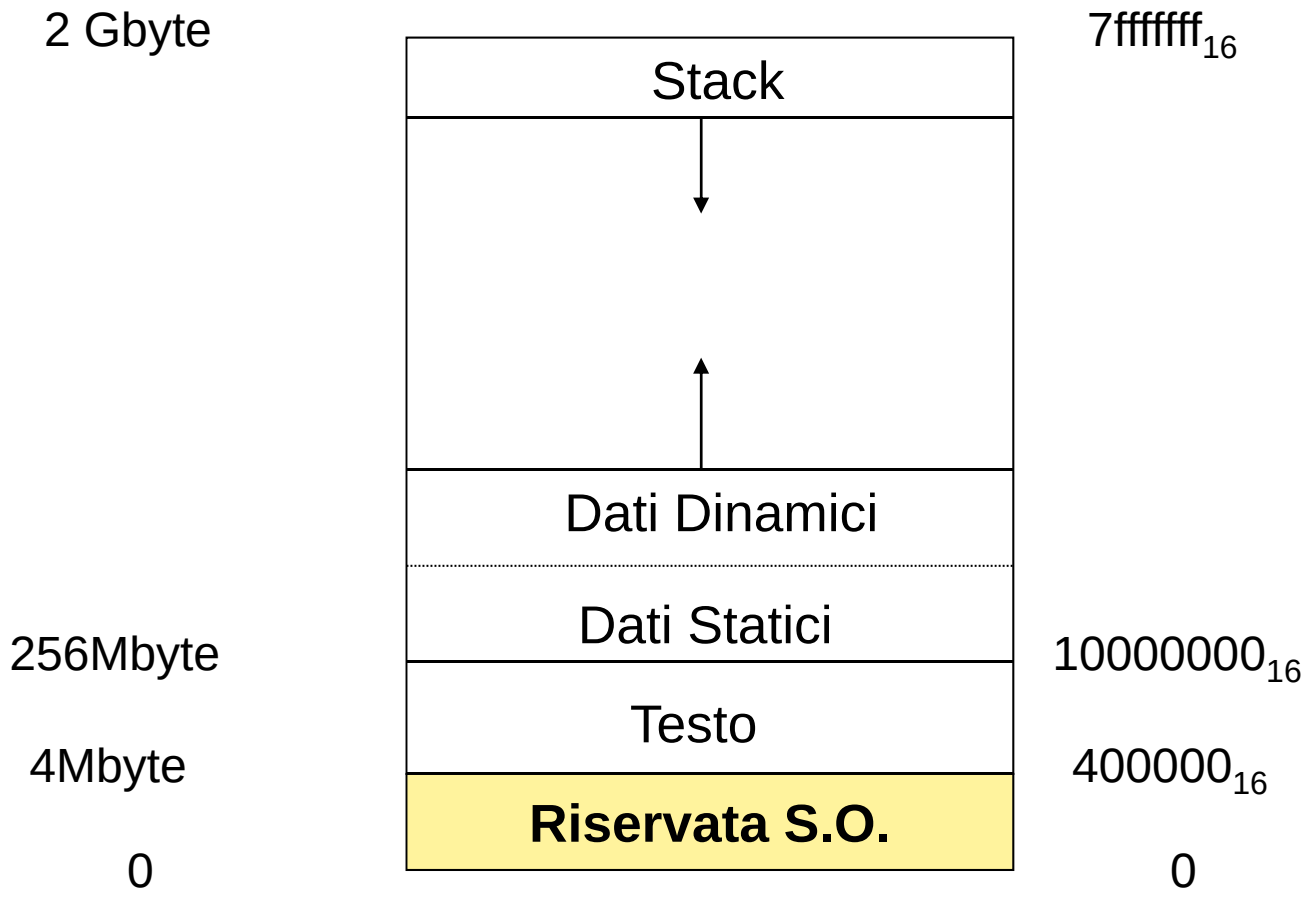


utilizzo dello stack

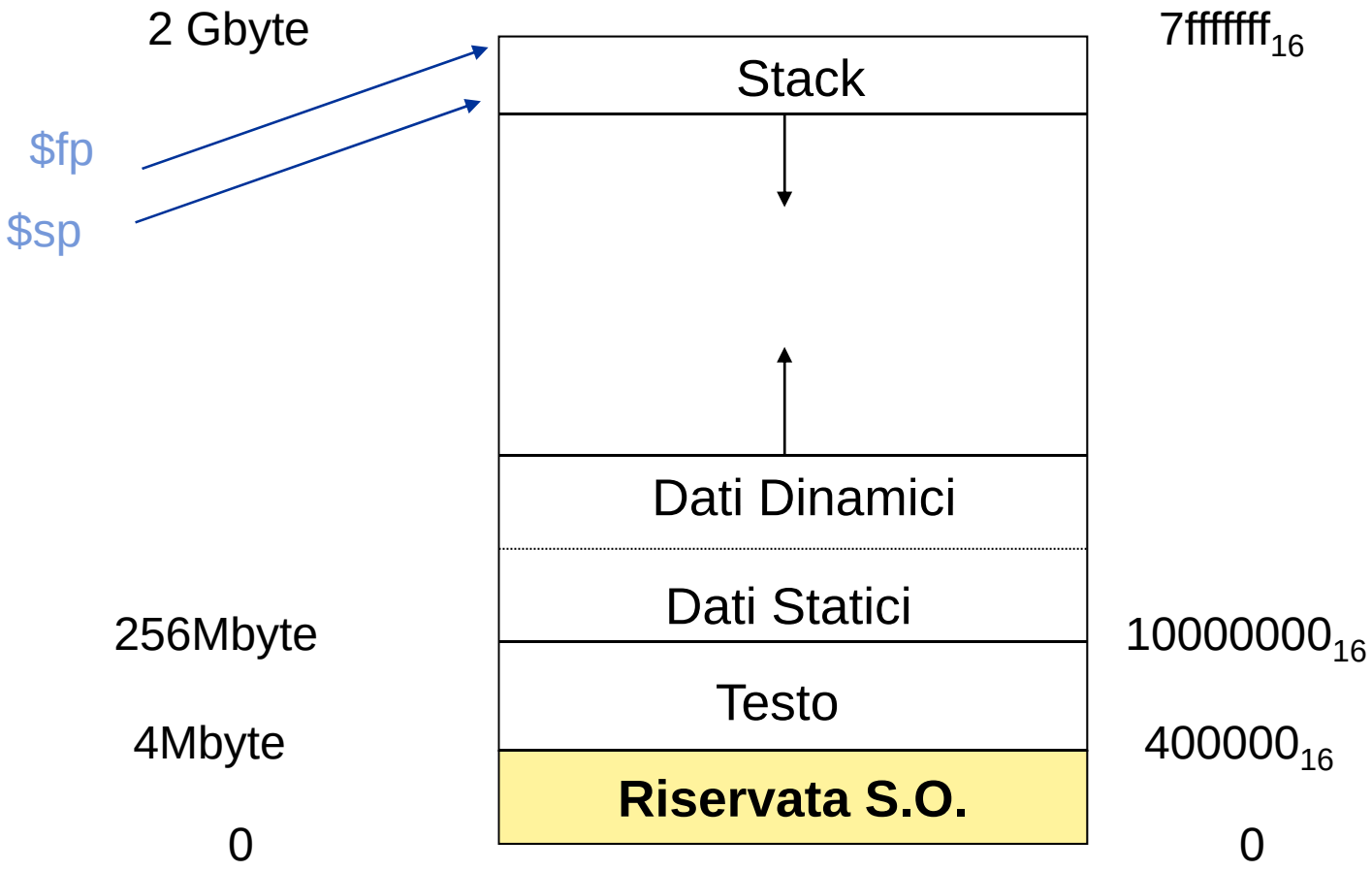
SOMMARIO

- Procedure
- Stack
- Chiamata a procedure

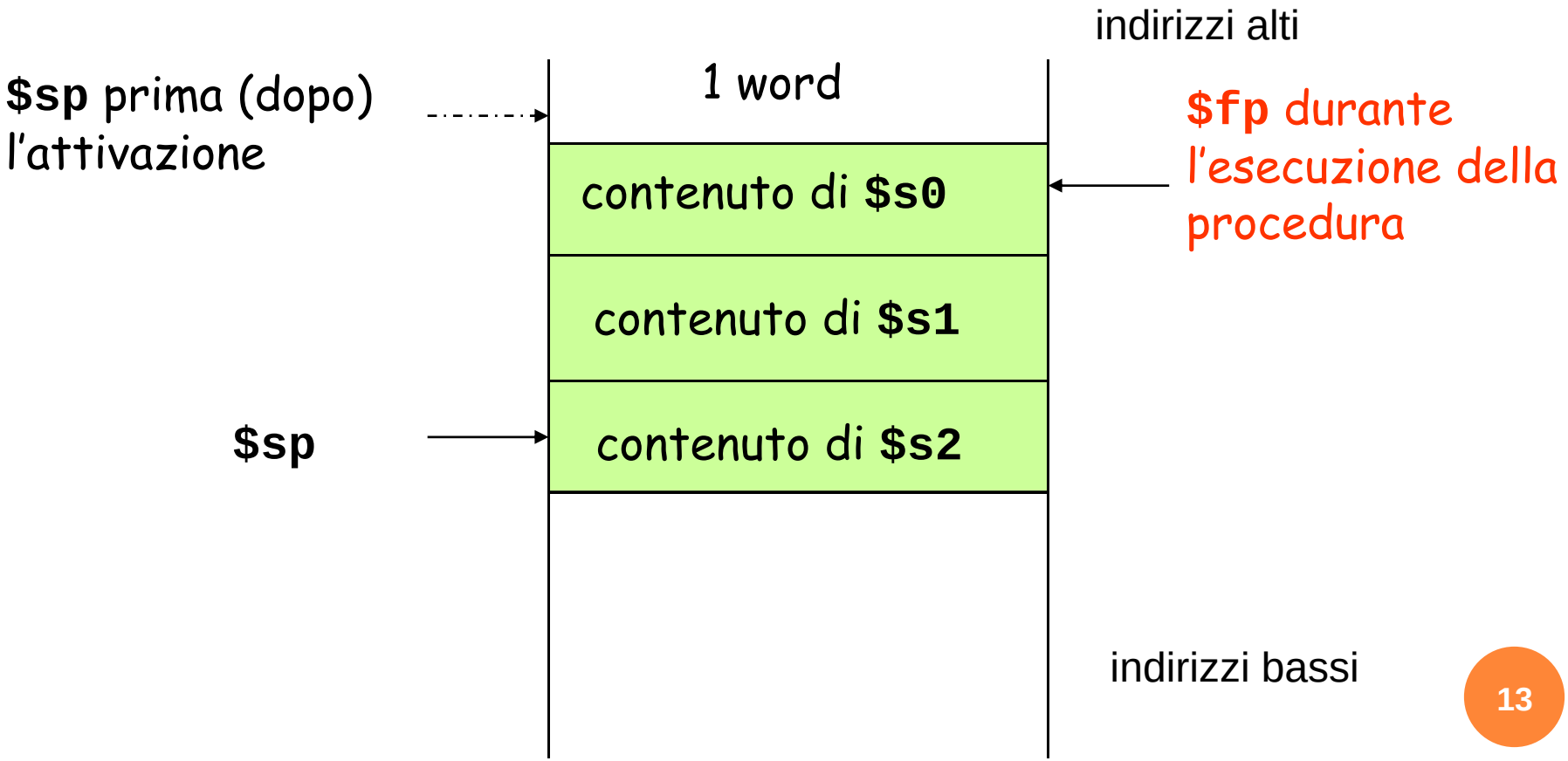
LO STACK


2 Gbyte
7fffffff16
Stack
Dati Dinamici
Dati Statici
256Mbyte
10000000₁₆
Testo
4Mbyte
400000₁₆
Riservata S.O.
0
0

LO STACK


2 Gbyte
$fp
$sp
Stack
Dati Dinamici
Dati Statici
Testo
Riservata S.O.
7fffffff16
10000000₁₆
400000₁₆
0
256Mbyte
4Mbyte
0

FRAME POINTER ($FP)



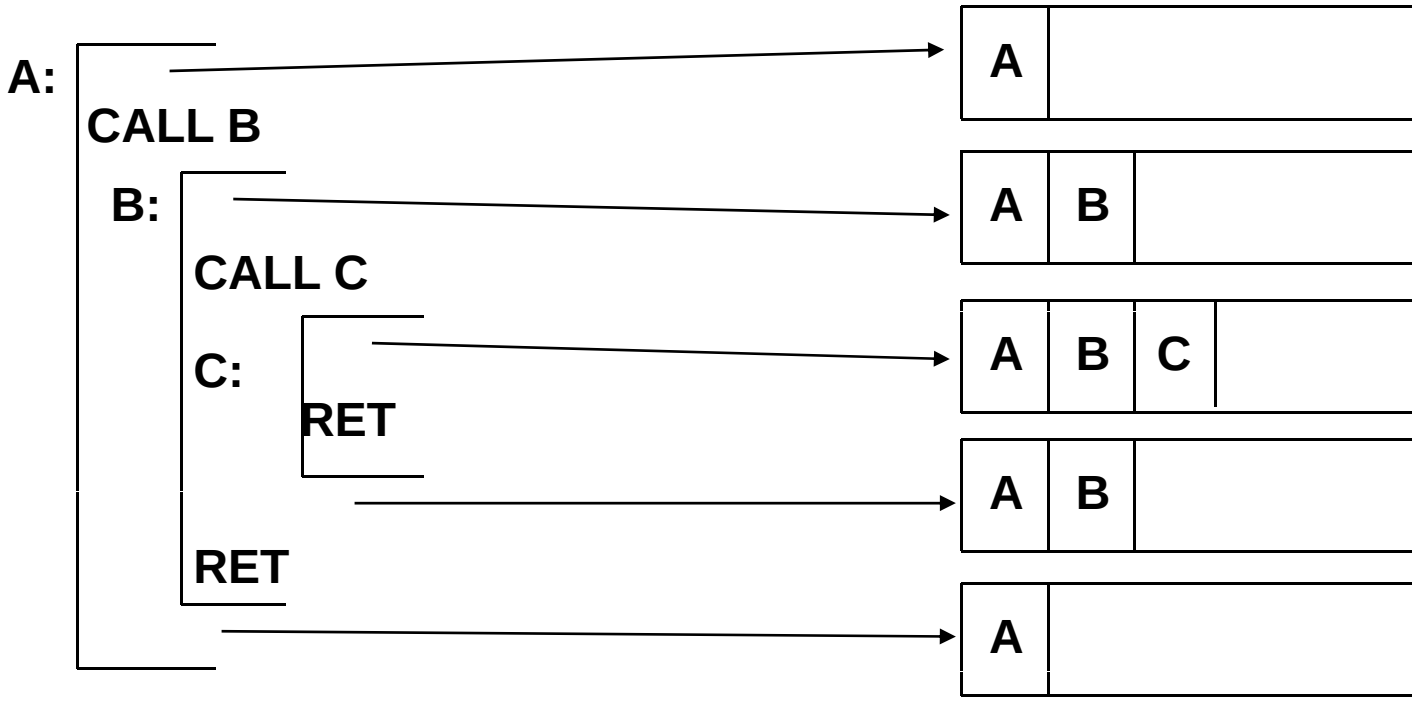
USO DEI REGISTRI: $SP

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.

QUANTO DEVE ESSERE GRANDE LO STACK?

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)

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.

- 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`**

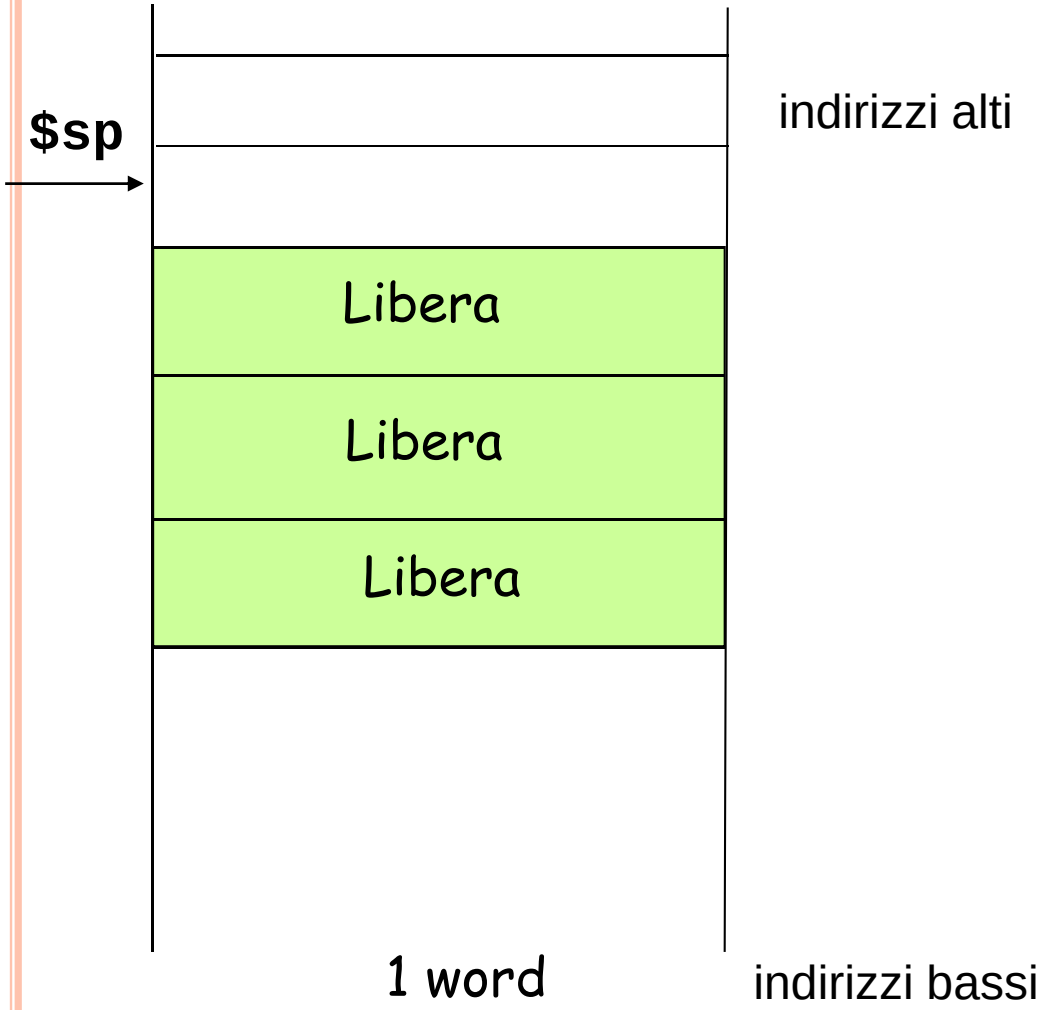
- È necessario liberare lo spazio allocato per evitare di riempire tutta la memoria

ESEMPIO: PROCEDURA SOMMA

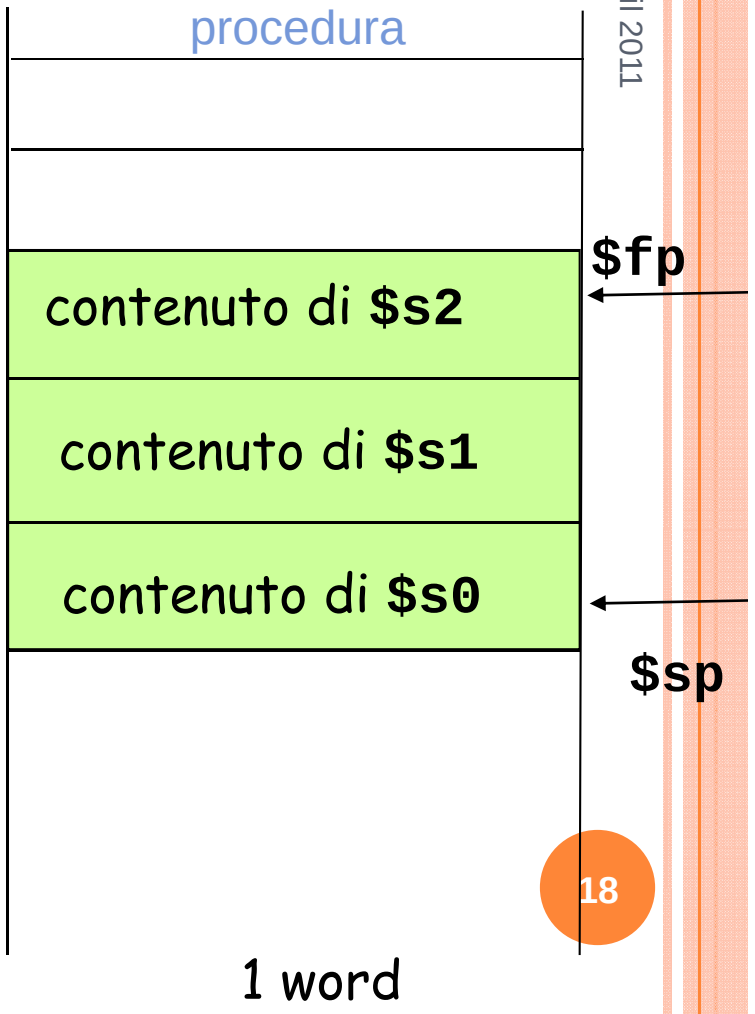
```
Somma_algebrica:
  addi $sp,$sp,-12          # alloca nello stack lo spazio per i 3 registri
  sw $s0, 8($sp)            # salvataggio di $s0
  sw $s1, 4($sp)            # salvataggio di $s1
  sw $s2, 0($sp)            # salvataggio di $s2

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

Prima della chiamata e dopo il ritorno



Dopo la chiamata e durante l'esecuzione della procedura



SOMMARIO

- Procedure
- Stack
- Chiamata a procedure

SALVATAGGIO DEI REGISTRI

- Main e procedure lavorano sullo stesso Register File.
- Gli stessi registri possono essere utilizzati da Main e da altre procedure...

Occorre garantire che le variabili memorizzate nei registri dal main, non vengano toccate dalla procedura.

Potrei salvare l'intero register file in stack…. Oppure la parte che serve. Vengono definite delle convenzioni.

CHIAMATA A PROCEDURA: CONVENZIONI

- Parametri di ingresso: registri $a0... $a3;
- Parametri di output: registri $v0, $v1;
- Registri saved ($s0...$s7): nel caso in cui la procedura li utilizzi, al termine della procedura deve lasciare inalterato il contenuto di questi registri (=> utilizzo dello stack per il deposito del valore dei registri);
- Registri temporary ($t0... $t7): possono essere modificati dalla procedura (Att.ne, non utilizzare i registri $t... per depositare dati che possono essere modificati dalla procedura).

- NB: E' necessario salvare tutti i registri "saved" ad ogni chiamata di procedura?

Es. 4.1

- Si scriva una procedura assembly, chiamata Elaboratore, che esegue la somma, differenza, moltiplicazione e divisione tra due numeri interi.
- Gli input della funzione siano i due numeri su cui operare e una terzo parametro per la selezione dell'operazione.
- Gli output della funzione siano costituiti dal risultato e, nel caso della divisione, da risultato e dal resto.

- Si scriva poi il main nel quale viene chiesto all'utente di inserire il codice per la selezione dell'operazione ed i due numeri su cui operare.
- Lo stesso main mostri a schermo il risultato.

ES.4.1 - SOLUZIONE

Es. 4.2 – Somma di un array

- Si scriva un programma che permetta all'utente di inserire un array di interi di dimensione arbitraria. Si scriva quindi una procedura che calcoli la somma di tutti gli elementi presenti nell'array.
- Il contenuto dell'array dovrà essere salvato nello stack prima dell'avvio della procedura. La procedurà preleverà i dati dell'array dallo stack, liberando poi lo spazio in memoria.

Es. 4.2 – Soluzione e osservazioni

Es. 4.3 – Somma di un array (rivista)

- Si scriva un programma che permetta all'utente di inserire un array di interi di dimensione arbitraria. Si scriva quindi una procedura che calcoli la somma di tutti gli elementi presenti nell'array.
- L'array verrà allocato staticamente in memoria. Alla procedura dovrà essere passato come parametro l'indirizzo in memoria dell'array.

ES. 4.3 – SOLUZIONE E OSSERVAZIONI

- Soluzione esercizio -> download dal sito web.