MIPS & SPIM

Modulo del Corso di Architettura degli Elaboratori

Nicola Paoletti

Università di Camerino
Scuola di Scienze e Tecnologie

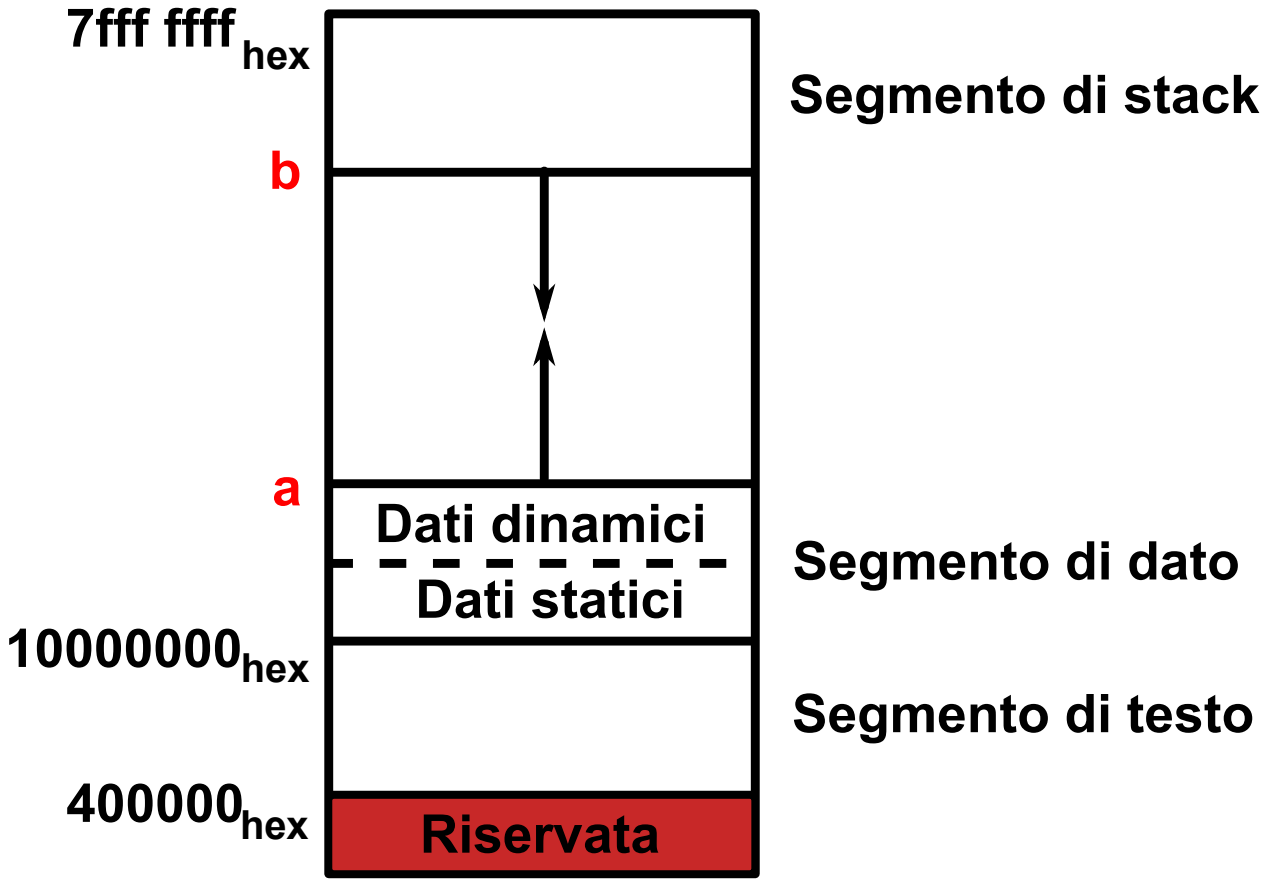
8 Maggio 2013
AA 2012/2013

Lezioni precedenti

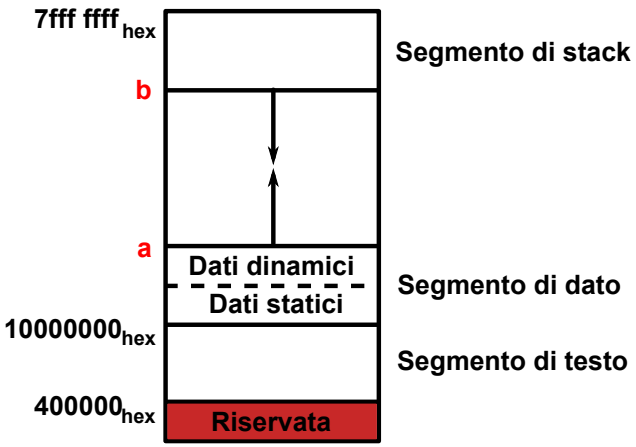
- **Lezione 1:** linguaggio macchina/assembly/alto livello; RISC vs CISC.
- **Lezione 2:** Introduzione al MIPS Instruction Set; registri; istruzioni aritmetiche e di data transfer.
- **Lezione 3:** Codifica delle istruzioni (R-type, I-type, J-type).

Riepilogo

Suddivisione della memoria (1/3)

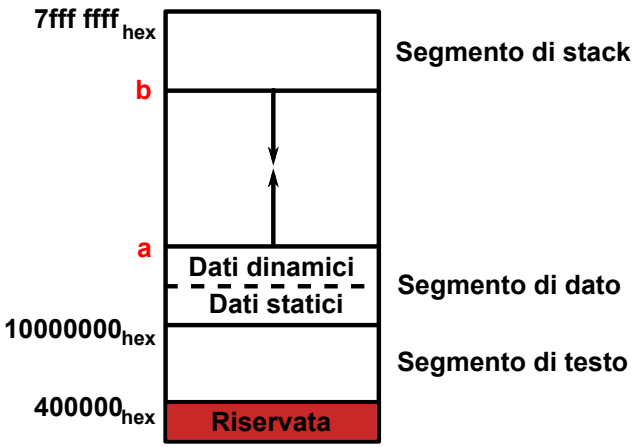


Suddivisione della memoria (2/3)



- **Segmento di testo (`0x400000-0x10000000`):** memorizza il codice (le istruzioni) del programma
- **Segmento di dato (`0x10000000-a`):** suddiviso in
  - *dati statici:* contengono gli oggetti di dimensione nota al compilatore, la cui vita è l'intera durata di esecuzione del programma (es. variabili globali)
  - *dati dinamici:* sono i dati allocati a run-time; tale segmento può espandersi verso l'alto per soddisfare le richieste del programma

Suddivisione della memoria (3/3)



- **Segmento di testo** (`0x400000-0x10000000`)
- **Segmento di dato** (`0x10000000-a`)
- **Segmento di stack** (`b-0x7fffffff`)**:** di dimensione variabile; si espande verso il basso.

Naturalmente deve valere: $a \leq b$

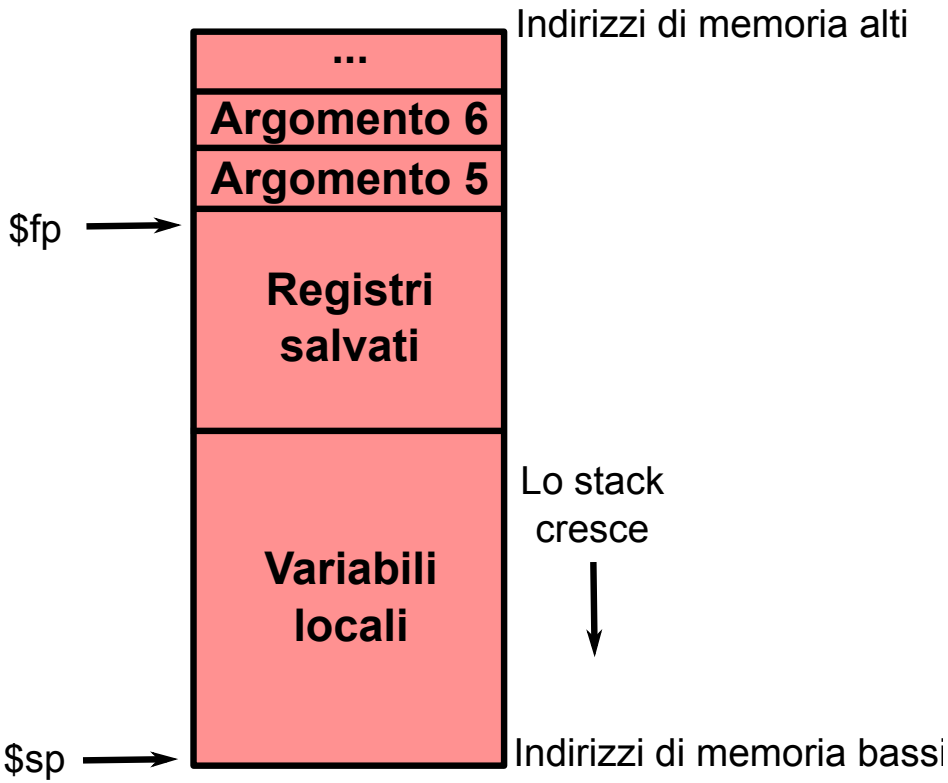
Riepilogo

Chiamate a procedura

- La chiamata a una procedura (subroutine, funzione, ...) richiede l'allocazione dinamica di memoria, mentre il Il ritorno da una procedura dealloca la memoria.
- Questo segue una politica LIFO (Last In First Out), ovvero viene deallocata la porzione di memoria che corrisponde all'ultima procedura chiamata.
- L'implementazione si avvale quindi di una struttura a stack, il *segmento di stack*.

Stack frame

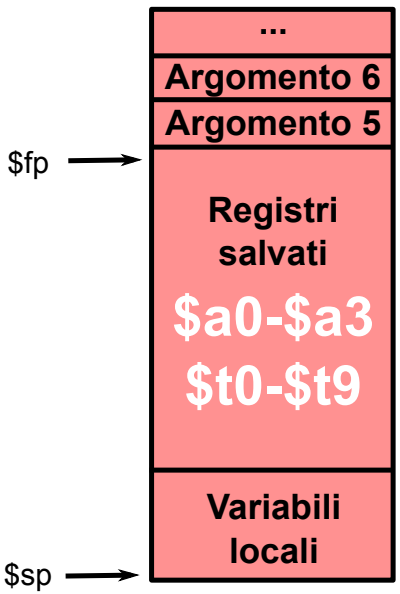
Lo *stack frame* è la porzione di memoria allocata per una procedura



- `$fp` (frame pointer) punta alla prima parola del frame della procedura correntemente in esecuzione; `$sp` (stack pointer) punta all'ultima parola del frame
- Lo stack si espande verso il basso, quindi l'indirizzo puntato da `$sp` è minore di quello puntato da `$fp`

Stack frame - Chiamate a procedura (1/3)

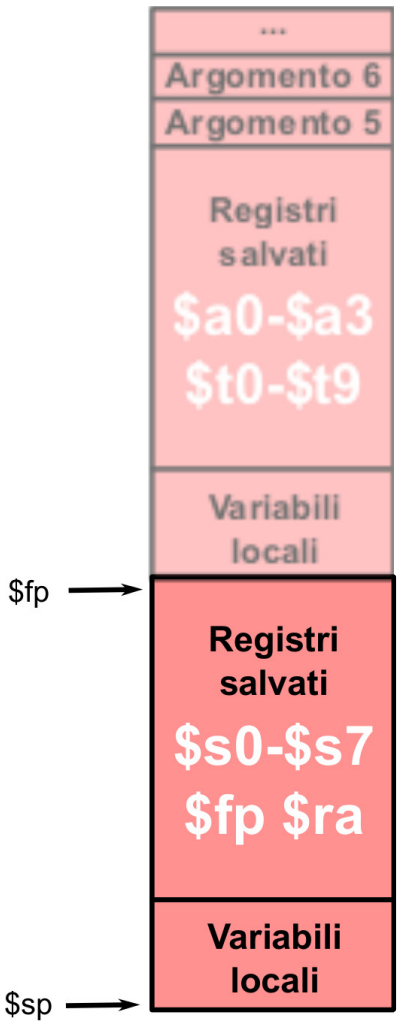
Convenzione utilizzata nella maggior parte delle macchine MIPS



Fase 1. Eseguita dal programma chiamante, immediatamente prima della chiamata

1. Passa gli argomenti. I primi 4 sono memorizzati in `$a0-$a3`, i restanti all'inizio dello stack
2. Salva i registri di competenza del chiamante (`$a0-$a3` e `$t0-$t9`). Siccome la procedura chiamata pu utilizzare questi registri, il chiamante deve salvarli se ne vuole riutilizzare i valori.
3. Esegue un'istruzione `$jal` (jump and link), che salta alla prima istruzione del programma chiamato e salva l'indirizzo di ritorno in `$ra` (`$31`)

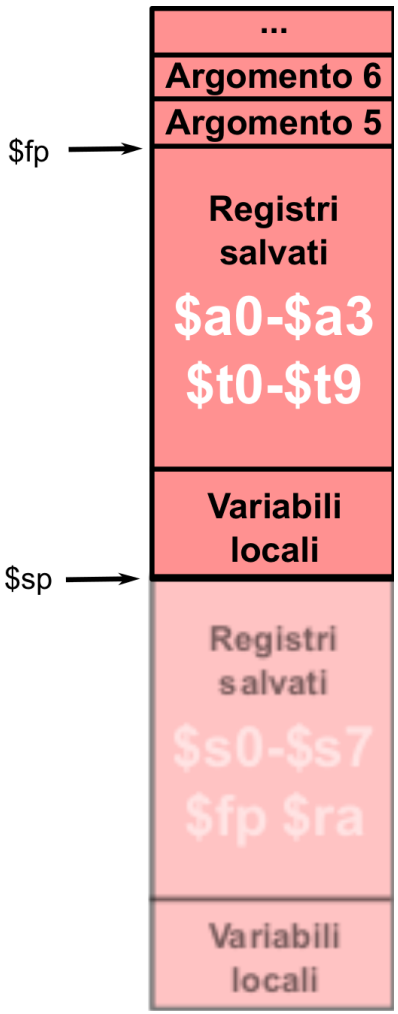
Stack frame - Chiamate a procedura (2/3)



Fase 2. Il programma chiamato inizializza il suo stack frame

1. Alloca lo stack frame: `$sp = $sp - dimensione_frame`
2. Salva i registri di competenza del chiamato. In particolare salva `$s0 - $s7`, `$fp` e `$ra` prima di modificarli, dato che il chiamante si aspetta di trovarli inalterati. `$ra` va salvato solo se il chiamato chiama a sua volta una procedura. `$fp` solo se il chiamato alloca uno stack frame.
3. Inizializza il frame pointer: `$fp = $sp + dimensione_frame - 4`

Stack frame - Chiamate a procedura (3/3)



Fase 3. Il programma chiamato restituisce il controllo al chiamante

1. Salva eventuali valori di ritorno in `$v0`
2. Ripristina i registri salvati
3. Libera (pop) lo stack frame: `$sp = $sp + dimensione_frame`
4. Restituisce il controllo, facendo un salto all'indirizzo in `$ra` (jump to register, `jr $ra`)

Esempio di chiamata a procedura (1/4)

```
main()
{
  printf("Il fattoriale di 10 e' %d\n", fact(10));
}

int fact(int n)
{
  if(n < 1)
    return 1;
  else
    return n*fact(n - 1);
}
```

Listing 1: Fattoriale in C

Esempio di chiamata a procedura (2/4)

```
   .text
   .globl main
main:
   subu $sp,$sp,32    #Lo Stack frame e' grande 32 bytes
   #32 bytes sono + del necessario, possiamo vederli
   #come una dimensione minima per lo stack frame
   sw $ra,20($sp)     #Salva il vecchio indirizzo di ritorno
   sw $fp,16($sp)     #Salva il vecchio frame pointer
   addiu $fp,$sp,28   #Inizializza il frame pointer

   li $a0,10          #Mette l'argomento (10) in $a0
   jal fact           #Chiama la funzione fattoriale
   la $a0,$LC         #Mette in $a0 la variabile stringa $LC
   move $a1,$v0       #Sposta il risultato di fact in $a1
   jal printf         #Chiama la funzione printf

   lw $ra,20($sp)     #Ripristina l'indirizzo di ritorno
   lw $fp,16($sp)     #Ripristina il frame pointer
   addiu $sp,$sp,32   #Libera lo stack frame
   jr $ra             #Ritorna il controllo al chiamante

   .data
$LC:
   .ascii "Il fattoriale di 10 e' %d\n"
```

Listing 2: Fattoriale in MIPS - Procedura main

Esempio di chiamata a procedura (3/4)

```
  .text
fact:
  subu $sp,$sp,32    #Lo Stack frame e' grande 32 bytes
  sw $ra,20($sp)     #Salva l'indirizzo di ritorno
  sw $fp,16($sp)     #Salva il frame pointer
  addiu $fp,$sp,28   #Inizializza il frame pointer
  sw $a0,0($fp)      #Salva l'argomento (n)

  lw $v0,0($fp)    #Carica n
  bgtz $v0,$L2     #Se n>0 va all'istruzione $L2
  li $v0,1         #altrimenti ritorna 1
  jr $L1           #passando per l'istruzione $L1
```

Listing 3: Fattoriale in MIPS - Procedura fact (1/2)

Esempio di chiamata a procedura (4/4)

```
$L2:
  lw $v1,0($fp)       #Carica n
  subu $v0,$v1,1      #Calcola n - 1
  move $a0,$v0        #Sposta il risultato in $a0
  jal fact            #Chiama fact
  lw $v1,0($fp)       #Carica n
  mul $v0,$v0,$v1     #Calcola fact(n-1) * n

$L1:                      #Il risultato e' in $v0
  lw $ra, 20($sp)         #Ripristina $ra
  lw $fp, 16($sp)         #Ripristina il frame pointer
  addiu $sp, $sp, 32      #Libera lo stack frame
  jr $ra                  #Ritorna il controllo al caller
```

Listing 4: Fattoriale in MIPS - Procedura fact (2/2)

Esempio di chiamata a procedura - Stack