Università degli Studi di Milano
Corso di Laurea in Informatica, A.A. 2017-2018

Convenzioni per la chiamata a procedure


Homepage del corso

Turno A
Nicola Basilico
Dipartimento di Informatica
Via Comelico 39/41 - 20135 Milano (MI)
Ufficio S242
nicola.basilico@unimi.it
+39 02.503.16294

Turno B
Jacopo Essenziale
Dipartimento di Informatica
Via Celoria 20 - 20133 Milano (MI)
AISLab
jacopo.essenziale@unimi.it
+39 02.503.14010

Chiamata a procedura

```
f = f + 1;

if (f == g)
        res = funct(f,g);

else
        f = f - 1;
```

Procedura chiamante (**caller**)

```
int funct (int p1, int p2){

        int out;
        out = p1 * p2;
        return out;
}
```

Procedura chiamata (**callee**)

Caller e callee comunicano attraverso:

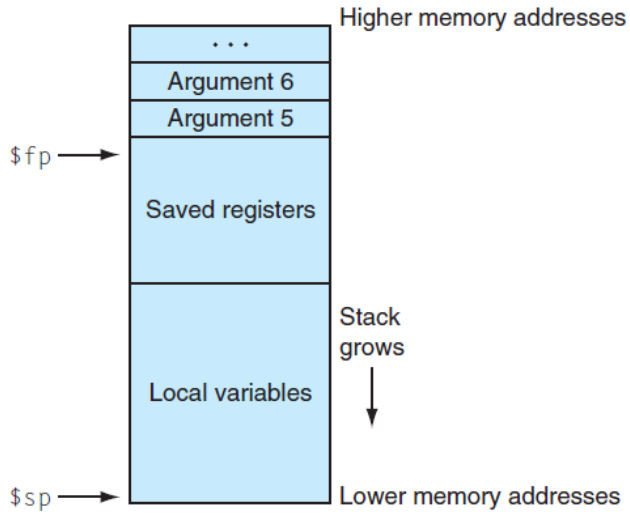
- passaggio di parametri di input (dal caller al callee)
- ritorno di parametri di output (dal callee al caller)

Chiamata a procedura

- Le informazioni di servizio (registri callee-saved e variabili locali) di una procedura sono contenute nello **stack frame**: un segmento di stack associato ad ogni procedura, anche detto *record di attivazione*

- Le chiamate e i ritorni da procedura seguono un ordine **LIFO**: l'ultima procedura invocata sarà la prima a restituire il controllo al suo chiamante

- Gli stack frame sono allocati sullo stack seguendo la sequenza delle chiamate (lo stack è gestito con una **convenzione** LIFO)



Registri caller-saved e registri callee-saved

Caller-saved

«salvati dal chiamante»
*sono i registri rispetto a cui **non** vige una convenzione di preservazione attraverso chiamate a procedura*

`$t0 … $t9, $a0 … $a3, $v0, $v1`

Un callee è libero di sovrascrivere questi registri: se un chiamante vuole essere sicuro di non perderne il valore deve salvarli sullo stack prima della chiamata a procedura

Esempio: `main` ha un dato importante nel registro `$t0`, prima di invocare `f` salva `$t0` sullo stack, una volta riacquisito il controllo lo ripristina.

Callee-saved

«salvati dal chiamato»
sono i registri rispetto cui la convenzione esige che vengano preservati attraverso chiamate a procedura

`$s0 … $s9, $ra, $fp`

un callee non può sovrascrivere questi registri: il chiamante si aspetta che restino invariati dopo la chiamata a procedura. Se il callee li vuole usare, deve prima salvarli sullo stack per poi ripristinarli una volta terminato

Esempio: `main` ha un dato importante nel registro `$s1` e invoca `f`; `f` salva `$s1` sullo stack prima di utilizzarlo, una volta terminato lo ripristina.

Chiamata a procedura

- Le chiamate a procedura avvengono secondo questa convenzione:

A cura del chiamante

1. **Salvataggio dei registri caller-saved**: `$t0 … $t9, $a0 … $a3, $v0, $v1` e **passaggio parametri**: il chiamante alloca i parametri di input in registri a cui la procedura accederà per leggerli.
2. **Deviazione del flusso di controllo** al blocco di istruzioni della procedura (chiamata a procedura).

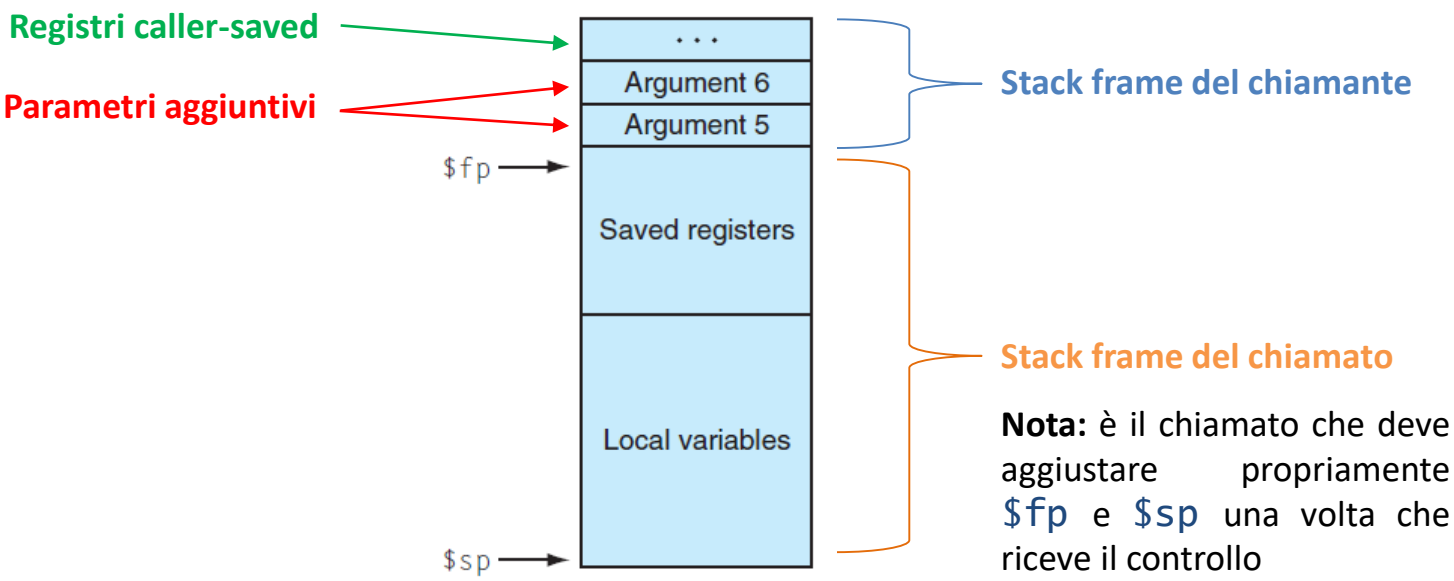
A cura del chiamato

3. **Allocazione dello stack frame del chiamato e salvataggio dei registri callee-saved**: `$s0 … $s7, $fp, $ra`.
4. **Esecuzione della procedura**.
5. **Ritorno dei valori di output (se presenti), ripristino dei registri callee-saved e restituzione del controllo al chiamante** (che riprenderà la sua esecuzione dall'istruzione successiva alla chiamata a procedura).

Nota: «chiamante» e «chiamato» scaturiscono da una distinzione concettuale di due segmenti di codice assembly funzionalmente indipendenti (potremmo anche chiamarli «moduli»). Un chiamante e un chiamato non fanno/richiedono assunzioni sulla loro interazione che vadano oltre la convenzione indicata sopra (punti 1-5).

Chiamante: Passaggio dei parametri (1)

- Dopo aver salvato sullo stack i registri caller-saved, il chiamante alloca i valori da passare nei registri `$a0, $a1, $a2, $a3`
- Nel caso si debbano passare più di 4 parametri? Per il quinto parametro e i successivi bisogna usare lo stack, **come?**
- Ci sono diverse convenzioni, ad esempio MIPS specifica un meccanismo abbastanza efficiente ma intricato, mentre *gcc* ne utilizza uno più semplice. Noi ci ispiriamo a quello di *gcc*.



Chiamante: Deviazione del flusso di controllo (2)

- Utilizzare l'istruzione **Jump and Link** per passare alla procedura (jump) e salvare l'indirizzo di ritorno (link)

```
jal label_procedura
```

- Salto incondizionato all'istruzione che sta alla label `label_procedura`, salva l'indirizzo della prossima istruzione (PC+4) in `$ra`

Chiamato: Stack frame e registri callee-saved (3)

- Creazione dello **stack frame**: sottrarre allo stack pointer corrente la dimensione del frame (che dipende da ciò che la procedura necessita)

```
$sp = $sp – DIM_FRAME
```

- Salvare sullo stack i registri callee-saved (se necessario):
  - `$s0 … $s7` se il chiamato necessita di sovrascriverli (il chiamante si aspetta che restino inalterati);
  - `$fp` se il chiamato crea uno stack frame (sempre);
  - `$ra` se il chiamato chiamerà a sua volta un'altra procedura

- Settaggio del **frame pointer**: sommare allo stack pointer la dimensione del frame – 4 (ricordiamo che `$fp` punta alla prima parola del frame)

```
$fp = $sp + DIM_FRAME - 4
```

A questo punto la procedura può eseguire le proprie istruzioni (4)

Chiamato: Nota sullo Stack Frame (record di attivazione)

- L'ultimo frame allocato sullo stack è associato alla procedura **P** correntemente in fase di esecuzione
- Fintanto che **P** non ha terminato, `$fp` contiene l'indirizzo della prima parola allocata nel frame di **P**
- Supponiamo che P chiami un'altra procedura **Q**: **Q** dovrà salvare sullo stack il registro `$fp` prima di rilocarlo sul proprio frame e, prima di restituire il controllo a **P**, ripristinarlo. Questo per garantire a **P** la non alterazione del suo frame pointer (è lo stesso principio che vale per ogni calle-saved register).
- A cosa serve il frame pointer? Può essere comodo per indirizzare sullo stack variabili locali della procedura usandolo come base address del frame; la stessa cosa non vale per lo stack pointer che varia ad ogni operazione di POP o PUSH

Chiamato: operazioni di ritorno (5)

- Se il chiamato è una *funzione* avrà un valore di ritorno da restituire al chiamante: viene lasciato nel registro `$v0` (e `$v1`)

- Ripristino dei registri callee-saved che erano stati salvati precedentemente, assegnandoli il loro valore iniziale presente sullo stack (**nota: incluso $fp!**)

- Deallocazione (pop) dello stack frame: sommare la dimensione del frame a `$sp`

```
$sp = $sp + DIM_FRAME
```

- Restituzione del controllo al chiamante con:

```
jr $ra
```

Esempio: chiamata della procedura «somma»

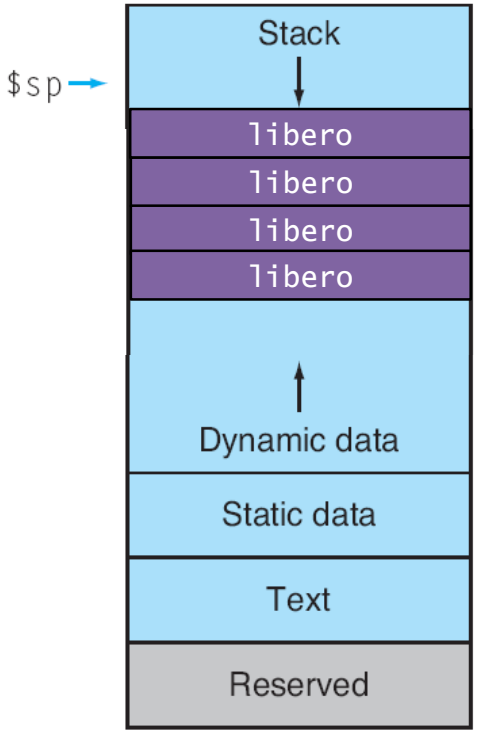
```
…

subu $sp, $sp, 4
sw $t0, 0($sp)
li $a0,5
li $a1,10
li $a2,7

jal Somma

# $v0 contiene il valore
della somma

…
```

Esempio: procedura «somma»

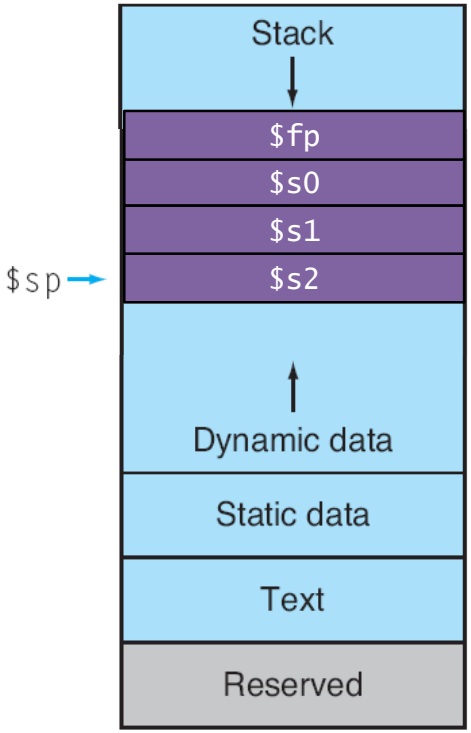
```
Somma:

# alloca nello stack lo spazio per i 3 registri
subu $sp,$sp,16
sw $fp, 12($sp)                 # salvataggio di $fp
sw $s0, 8($sp)                  # salvataggio di $s0
sw $s1, 4($sp)                  # salvataggio di $s1
sw $s2, 0($sp)                  # salvataggio di $s2
addiu $fp, $sp, 12

# esegue istruzioni
add $s0, $a0, $a1
add $s1, $a2, $a3
add $s2, $s0, $s1

#valore di ritorno
add $v0, $s2, $zero

# ripristino del vecchio contenuto dei registri estraendolo dallo stack
lw $s2, 0($sp)                  # ripristino di $s2
lw $s1, 4($sp)                  # ripristino di $s1
lw $s0, 8($sp)                  # ripristino di $s0
lw $fp, 12($sp)                 # ripristino di $fp
addi $sp, $sp, 16               # deallocazione stack frame

#restituzione controllo al chiamante
jr $ra
```

Esempio

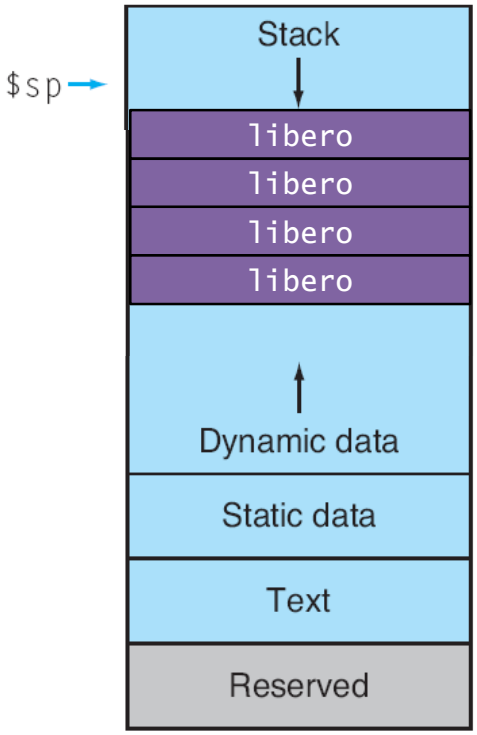
Prima della chiamata



Dopo la chiamata, durante l'esecuzione della procedura



Dopo il ritorno al chiamante



Esercizio 5.1

- Nome del file sorgente: *elaboratore.asm*

- Si scriva una procedura assembly, chiamata `Elaboratore`, che esegua somma, differenza, moltiplicazione e divisione tra due numeri interi

- Input: i due operandi e un terzo parametro per la selezione dell'operazione

- Output: risultato (nel caso della divisione restituisca anche il resto)

- Si scriva **poi** il `main` dove:
  - vengono chiesti all'utente operandi e operatore
  - il risultato dell'operazione è mostrato a terminale

Esercizio 5.2

- Nome del file sorgente: *sommaSelettiva.asm*

- Si scriva un programma che:
  - chieda all'utente di inserire un array di interi di dimensione arbitraria
  - invochi una procedura P
  - stampi il valore ritornato da P

- La procedura P è definita come segue:
  - Input: l'array inserito dall'utente e un parametro k
  - se k=0 la procedura calcola la somma di tutti gli interi in posizione (indice nell'array) dispari
  - se k=1 sommerà quelli in posizioni pari.

- *Suggerimento: allocare l'array staticamente in memoria e passare alla procedura il base address (passaggio per indirizzo)*

Esercizio 5.3

- nome del file sorgente: *subseteq.asm, belongs.asm*

Si implementi la procedura *subseteq* così definita:

- Input: due array di interi A1 e A2
- Output: 1 se ogni elemento di A1 è presente in A2, 0 altrimenti

La procedura *subseteq* dovrà a sua volta utilizzare un'altra procedura *belongs* così definita:

Input: un array A e un intero i

- Output: 1 se i è contenuto in A, 0 altrimenti
- Si implementi infine il *main* che acquisisca i dati, chiami *subseteq* e raccolga i risultati

Università degli Studi di Milano
Laboratorio di Architettura degli Elaboratori II
Corso di Laurea in Informatica, A.A. 2017-2018

Nicola Basilico
Dipartimento di Informatica
Via Comelico 39/41 - 20135 Milano (MI)
Ufficio S242
nicola.basilico@unimi.it
+39 02.503.16294

Hanno contribuito alla realizzazione di queste slides:

- Iuri Frosio
- Jacopo Essenziale