POLITECNICO
MILANO 1863

Architettura dei calcolatori e sistemi operativi

Sottoprogramma e MIPS

Capitolo 2 P&H

Sommario

Modello di chiamata

Area di attivazione

Sottoprogramma

Nei linguaggi di alto livello, per esempio in linguaggio C, la ***chiamata a sottoprogramma*** ha come effetto la creazione di un'***area (o record) di attivazione*** sulla ***pila (stack)***.

A ogni chiamata di sottoprogramma viene creata un'area di attivazione.

Quando il sottoprogramma termina l'area viene rilasciata dalla pila.

Con sottoprogrammi annidati le aree vengono tutte messe sulla pila e l'ultima messa (ossia quella in cima all apila) corrisponde al sottoprogramma correntemente in esecuzione.

L'area di attivazione di *main* è la prima ad essere creata quando il programma viene lanciato e l'ultima a essere rilasciata (deallocata).

Sottoprogramma (continua)

L’area di attivazione del sottoprogramma è associata in modo opportuno alle informazioni seguenti:

- i parametri formali (e I loro valori) passati al sottoprogramma
- l’indirizzo di ritorno al (sotto)programma chiamante
- informazioni per gestire lo spazio allocato per l’area di attivazione
- le variabili locali del sottoprogramma
- il valore restituito al (sotto)programma chiamante

Sottoprogramma e ISA

A livello ISA la chiamata a sottoprogramma di alto livello deve essere espansa in più istruzioni macchina, eseguite in ambiti diversi:

- il **chiamante** (***caller***) gestisce la parte relativa al passaggio dei parametri
- il **chiamante** attiva il sottoprogramma tramite l'***istruzione di chiamata ISA***
- l'**esecuzione dell'istruzione di chiamata ISA** gestisce la parte relativa al salvataggio dell'indirizzo di ritorno (***PC***) e attiva il sottoprogramma
- il **chiamato** (***callee***) gestisce l'allocazione delle variabili locali e del valore restituito

Modello di chiamata a sottoprogramma

La chiamata a sottoprogramma segue sempre l'espansione vista, però il ***modello di chiamata a sottoprogramma*** non è identico in tutti i processori.

A livello ISA:

- per vari aspetti, è necessario tenere conto anche dei registri del processore
- in generale è sempre possibile una certa flessibilità (il modello di chiamata non è totalmente vincolato)

Per esempio:

- Valori dei parametri attuali, valore restituito: registri o stack ?
- Indirizzo di ritorno: registro o stack ?
- Salvataggio dei registri usati nel modello di chiamata e nel sottoprogramma: chi li salva ?
- Come si definisce l'area di attivazione e come si gestiscono le variabili locali ?

Modello di chiamata a sottoprogramma in MIPS

L'architettura ISA di MIPS:

- vincola in modo forte il ***salvataggio dell'indirizzo di ritorno*** (*è fatto hardware*) tramite l'istruzione di chiamata (e quella di ritorno) da sottoprogramma
- definisce delle convenzioni sempre adottate per il ***passaggio dei parametri*** e per il ***valore restituito***
- caratterizza i ***gruppi di registri*** in base al fatto che i valori corrispondenti siano o meno da considerare preservati dal chiamato
- se un registro è definito "preservato dal chiamato", allora sarà compito del chiamato salvarne i valori per poi restituirlo integro al chiamante (***callee-saved registers***)

Istruzione di chiamata e ritorno da sottoprogramma

Chiamata a sottoprogramma

```
jal label      (jump and link)      # $ra ← PC + 4
                                    # PC ← i.e. label
```

L'etichetta ***label*** è il riferimento simbolico all'indirizzo della prima istruzione del sottoprogramma. Verrà tradotta in indirizzo effettivo (i.e.) dall'assemblatore e dal collegatore (linker).

Ritorno da sottoprogramma

```
jr $ra      (jump register)      # PC ← $ra
```

Convenzioni per il *passaggio dei parametri* e per il *valore restituito*

Passaggio dei parametri

- i primi quattro parametri (***argument***), numerati da sx a dx nella testata, vengono passati nei registri ***$a0-$a3***
  - se di tipo scalare o puntatore (a 32 bit)
  - il nome di vettore è considerato puntatore (al primo elemento)
- i parametri rimanenti, se presenti, sono passati sulla pila

  se un sottoprogramma ha 6 parametri i valori di *arg5* e *arg6* sono passati sulla pila

Valore restituito

- il valore restituito viene salvato nel registro ***$v0***
  - se di tipo scalare o puntatore (a 32 bit)
  - il nome di vettore è considerato puntatore (al primo elemento)
- se di tipo *double* (numero reale in vigola mobile) si usa anche ***$v1***

Convenzioni per il salvataggio dei registri

- L'esecuzione di un sottoprogramma non deve interferire con l'ambiente del (sotto)programma chiamante.
- I registri usati dal chiamato devono essere ripristinabili al rientro.

Convenzioni adottate da MIPS

- Per ottimizzare gli accessi alla memoria, il chiamante e il chiamato salvano (eventualmente) sulla pila soltanto un particolare gruppo di registri.
- Il chiamato può usare la pila per le strutture dati locali (p. es. array, strutture) e le variabili locali.

Preservato dal callee (registri callee-saved)	***Non preservato dal callee (registri caller-saved)***
registri saved: $s0-$7	registri temporanei: $t0-$t9
registro frame pointer: $fp	registri argomento: $a0-$a3
registro return address: $ra	registri di ritorno: $v0-$v1

La gestione della pila in MIPS

La pila cresce da indirizzi di memoria alti verso indirizzi di memoria bassi e il registro *stack pointer* **$sp** punta alla prima parola piena della pila.

L'inserimento di un dato nella pila (operazione di ***push***) avviene decrementando il registro **$sp** per allocare lo spazio, ed eseguendo l'istruzione **sw** per inserire il dato.

Esempio: salvare il registro $s0 sulla pila

```
addiu  $sp, $sp, -4
sw     $s0, 0($sp)
```

Il prelevamento di un dato nella pila (operazione di ***pop***) avviene eseguendo l'istruzione **lw** e incrementando il registro **$sp** (per eliminare il dato), riducendo così la dimensione della pila.

Esempio: prelevare la cima della pila e salvarla nel registro $s0

```
lw     $s0, 0($sp)
addiu  $sp, $sp, 4
```

Nota: `0($sp)` si può abbreviare in `($sp),` intendendo che lo spiazzamento sia 0.

Passi del modello di chiamata in MIPS

- La chiamata a funzione comporta queste fasi in successione:
  - prologo del chiamante
  - salto a chiamato
  - prologo del chiamato
  - corpo elaborativo del chiamato
  - epilogo del chiamato
  - rientro a chiamante
  - epilogo del chiamante
- Alcuni passaggi possono ridursi a poco, secondo le convenzioni o la situazione specifica.
- Relativamente alla pila:
  - il prologo del chiamante può comportare passaggio di parametri e salvataggio di registri
  - il prologo del chiamato può comportare il salvataggio di registri e l'allocazione di variabili locali
  - è necessario calcolare la ***dimensione in byte*** dell'area richiesta: quest'area è ***l'area (o record o frame) di attivazione*** della funzione

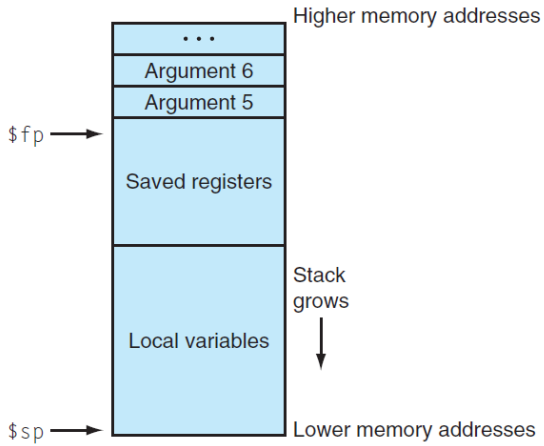
Convenzioni (ulteriori) adottate in ACSO

- Il salvataggio dei registri nella pila viene fatto seguendo il numero d'ordine crescente all'interno di ciascun gruppo: *t0, t1 …t9* ecc.
  - per l'ordine di salvataggio dei «gruppi» si veda il dettaglio dell'area di attivazione

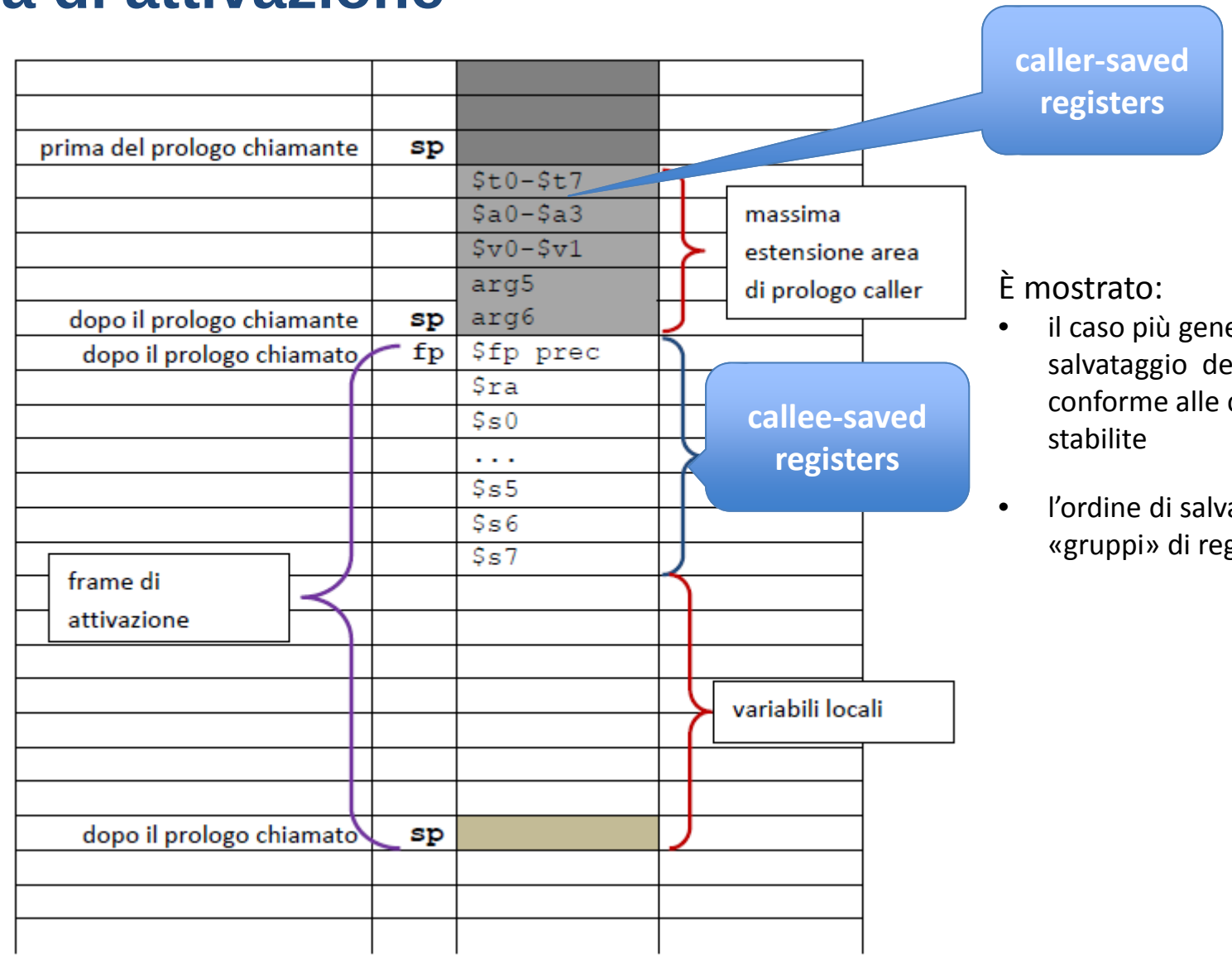
- Il salvataggio di un qualsiasi registro viene fatto solo se necessario. Per esempio:
  - ***fp*** è salvato solo se lo si usa per referenziare parole di memoria in pila
  - ***ra*** non viene salvato in procedure foglia (che non ne chiamano altre)

- Gli argomenti di un sottoprogramma vengono passati in ordine di elencazione:
  - primi quattro nei registri *a0 - a3*
  - poi in pila (p. es. *arg5* e *arg6*)

Nota bene: alcune convenzioni ACSO possono differire da quelle del libro di testo.



Area di attivazione



È mostrato:

- il caso più generale di salvataggio dei registri conforme alle convenzioni stabilite
- l'ordine di salvataggio dei «gruppi» di registri

Convenzioni di chiamata *caller* (1)

prologo del chiamante

- scrive nei registri *a*0 - *a*3 i paramenti in ingresso alla funzione
  - trascuriamo il caso di più di quattro parametri
- salva sulla pila i registri *t*0 – *t*9 che si vogliono riavere inalterati quando la funzione sarà rientrata (idem per i registri *v*0 – *v*1)
- se occorre preservarli, salva sulla pila gli argomenti *a*0 - *a*3
  - per ottimizzare un po', possiamo limitarci a salvare in pila, tra i parametri passati al chiamato, solo quelli che il chiamante non usa più a valle della chiamata, e che dunque esso non ha bisogno di riavere inalterati qualora il chiamato li avesse modificati (ma non è un errore salvarli sempre tutti)

salto a chiamato

- `jal FUNZ`

Convenzioni di chiamata *callee* (1)

prologo del chiamato

- crea area di attivazione:

```
addiu $sp, $sp, –dim. area in byte
```

- se il registro $*fp* è in uso:
  - viene salvato sulla pila (salva $*fp* precedente)
  - viene aggiornato in modo da puntare alla cima dell'area di attivazione, cioè alla parola di pila che contiene il valore di $*fp* appena salvato
- se la funzione chiamata non è foglia:
  - il registro $*ra* viene salvato sulla pila poiché sarà usato (e dunque sovrascritto) in chiamata annidata
- salva sulla pila i registri *s*0 – *s*7 assegnati a variabili locali

corpo elaborativo del chiamato

Convenzioni di chiamata *callee* (2)

epilogo del chiamato

- scrive nel registro *v*0 il valore di uscita
- ripristina dalla pila i registri *s*0 – *s*7 assegnati a variabili locali
- se il registro $*fp* è in uso, lo ripristina dalla pila
- se la funzione non è foglia, ripristina dalla pila il registro $*ra*
- elimina area di attivazione:

```
addiu $sp, $sp, dim. area in byte
```

rientro a chiamante

- `jr $ra`

Convenzioni di chiamata *caller* (2)

epilogo del chiamante

- ripristina dalla pila i registri *a0* – *a3* che erano stati preservati per il rientro dalla funzione chiamata
- ripristina dalla pila i registri *t*0 – *t*9 che erano stati preservati per il rientro dalla funzione chiamata (idem per i reg. *v*0 – *v*1)
- trova nel registro *v*0 il valore di uscita dalla funzione chiamata

Esempio 1

varloc *a* allocata in registro s0 (senza *fp*)

```
int f (int n) {
    int a
    ...
    a = a + n
    // chiama funz.
    ...
    return a
} /* f */
```

area di attivazione (dim. 8 byte)

```
      ra salvato   4($sp)
sp-> s0 salvato    0($sp)
```

```
F:  addiu $sp, $sp, -8    // crea area
    sw    $ra, 4($sp)     // salva ra
    sw    $s0, 0($sp)     // salva s0
    ...
    add   $s0, $s0, $a0   // calcola a = a + n
    // chiama un'altra funzione
    ...
    move  $v0, $s0        // valore uscita
    lw    $s0, 0($sp)     // ripristina s0
    lw    $ra, 4($sp)     // ripristina ra
    addiu $sp, $sp, 8     // elimina area
    jr    $ra             // rientra
```

Esempio 2

varloc *a* allocata in pila (senza *fp*)

```
int f (int n) {
   int a
   ...
   a = a + n
   // chiama funz.
   ...
   return a
} /* f */
```

area di attivazione (dim. 8 byte)

```
     ra salvato  4($sp)
sp-> varloc a    0($sp)
```

```
F:  addiu $sp, $sp, -8   // crea area
    .eqv  RA, 4  // spiazzamento in pila di ra
    .eqv  A, 0   // spiazzamento in pila di a
    sw    $ra, RA($sp)   // salva ra
    ...
    lw    $t0, A($sp)    // carica a
    add   $t0, $t0, $a0  // calcola a = a + n
    sw    $t0, A($sp)    // memorizza a
    // chiama un'altra funzione
    ...
    lw    $v0, A($sp)    // valore uscita
    lw    $ra, RA($sp)   // ripristina ra
    addiu $sp, $sp, 8    // elimina area
    jr    $ra            // rientra
```

similmente se ci sono più variabili locali

Esempio 3

```
int f (int n) {
    int a
    ...
    a = a + n
    // chiama funz.
    ...
    return a
} /* f */
```

area di attivaz. (dim. 12 byte)
con spiazzamenti riferiti a $*sp*

```
fp-> fp salvato   8($sp)
     ra salvato   4($sp)
sp-> varloc a     0($sp)
```

area di attivaz. (dim. 12 byte)
con spiazzamenti riferiti a $*fp*

```
fp-> fp salvato   0($fp)
     ra salvato  -4($fp)
sp-> varloc a    -8($fp)
```

```
F:  addiu $sp, $sp, -12   // crea area
    sw    $fp, 8($sp)     // salva fp precedente
    addiu $fp, $sp, 8     // sposta fp corrente
    sw    $ra, -4($fp)    // salva ra
    ...
    lw    $t0, -8($fp)    // carica a
    add   $t0, $t0, $a0   // calcola a = a + n
    sw    $t0, -8($fp)    // memorizza a
    // chiama un'altra funzione
    ...
    lw    $ra, -4($fp)    // ripristina ra
    lw    $fp, 8($sp)     // ripristina fp
    addiu $sp, $sp, 12    // elimina area
    jr    $ra             // rientra
```

ora $*sp* potrebbe anche cambiare durante l'esecuzione

attenzione: spiazzamenti di $*ra* e varloc *a* riferiti a $*fp*

similmente con le combinazioni viste prima e/o con ***.eqv***