POLITECNICO
MILANO 1863

Architettura dei calcolatori e sistemi operativi

Sottoprogrammi e MIPS

Espressioni algebriche

Capitolo 2 P&H

Sottoprogrammi

- Modello di chiamata
- Area di attivazione

Calcolare un’espressione algebrica

Sottoprogrammi

Nei linguaggi di alto livello, ad esempio in C, la ***chiamata a sottoprogramma*** ha come effetto la creazione di un ***record di attivazione*** sullo ***stack***

- ad ogni chiamata (attivazione) viene creato un record
- quando il sottoprogramma termina il record viene deallocato dallo stack
- con sottoprogrammi annidati i record vengono impilati sullo stack e l'ultimo corrisponde al sottoprogramma correntemente in esecuzione
- Il record di attivazione di main è il primo ad essere creato quando il programma viene lanciato e l'ultimo a essere deallocato

Sottoprogrammi (cont.)

Il record di attivazione è associato in modo opportuno alle seguenti informazioni

- i parametri formali (e loro valori)
- l'indirizzo di ritorno al chiamante
- informazioni per gestire lo spazio allocato per il record di attivazione
- le variabili locali
- il valore restituito

Sottoprogrammi 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, ma il ***modello di chiamata a sottoprogramma*** non è identico in tutti i processori

A livello ISA

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

Ad 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 il record di attivazione e come si gestiscono le variabili locali?

Modello di chiamata a sottoprogramma nel MIPS

L'architettura ISA del 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 corrispondenti valori siano o meno da considerarsi 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 in MIPS

Chiamata a sottoprogramma

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

dove ***label*** è il riferimento simbolico all'indirizzo della prima istruzione del sottoprogramma. Verrà tradotto in indirizzo effettivo (i.e.) dall'assemblatore e dal 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 rimanenti, se presenti, sono passati sullo stack
  - se un sottoprogramma ha 6 parametri i valori di Arg5 e Arg6 sono passati sullo stack

Valore restituito

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

Convenzioni per il salvataggio dei registri

- L'esecuzione di un sottoprogramma non deve interferire con l'ambiente del chiamante
- I registri usati dal chiamato devono poter essere ripristinati al rientro

Convenzioni adottate dal MIPS

- Per ottimizzare gli accessi alla memoria, il chiamante e il chiamato salvano sullo stack soltanto un particolare gruppo di registri
- Il chiamato può usare lo stack per le strutture dati locali (ad 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 dello stack in MIPS

Lo stack cresce da indirizzi di memoria alti verso indirizzi di memoria bassi e lo stack pointer punta alla prima parola piena dello stack

L'inserimento di un dato nello stack (operazione di ***push***) avviene decrementando **$sp** per allocare lo spazio ed eseguendo **sw** per inserire il dato

Esempio: ***salvare il registro*** $s0 nello stack

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

Il prelevamento di un dato nello stack (operazione di ***pop***) avviene eseguendo **lw** ed incrementando **$sp** (per eliminare il dato), riducendo così la dimensione dello stack

Esempio: ***ripristinare il valore del registro*** $s0, prelevandolo dalla cima dello stack

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

Passi del modello di chiamata in MIPS

- La chiamata a funzione comporta
  - 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 allo stack
  - 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. E' necessario calcolare la ***dimensione in byte*** dell'area richiesta: quest'area è ***l'area (frame) di attivazione*** della funzione

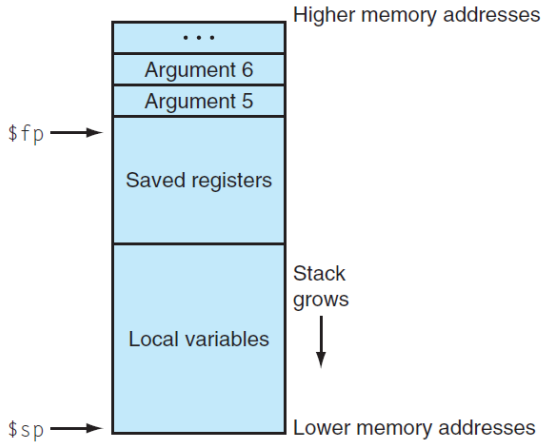
Convenzioni (ulteriori) adottate in ACSO

- Il salvataggio dei registri nello stack 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

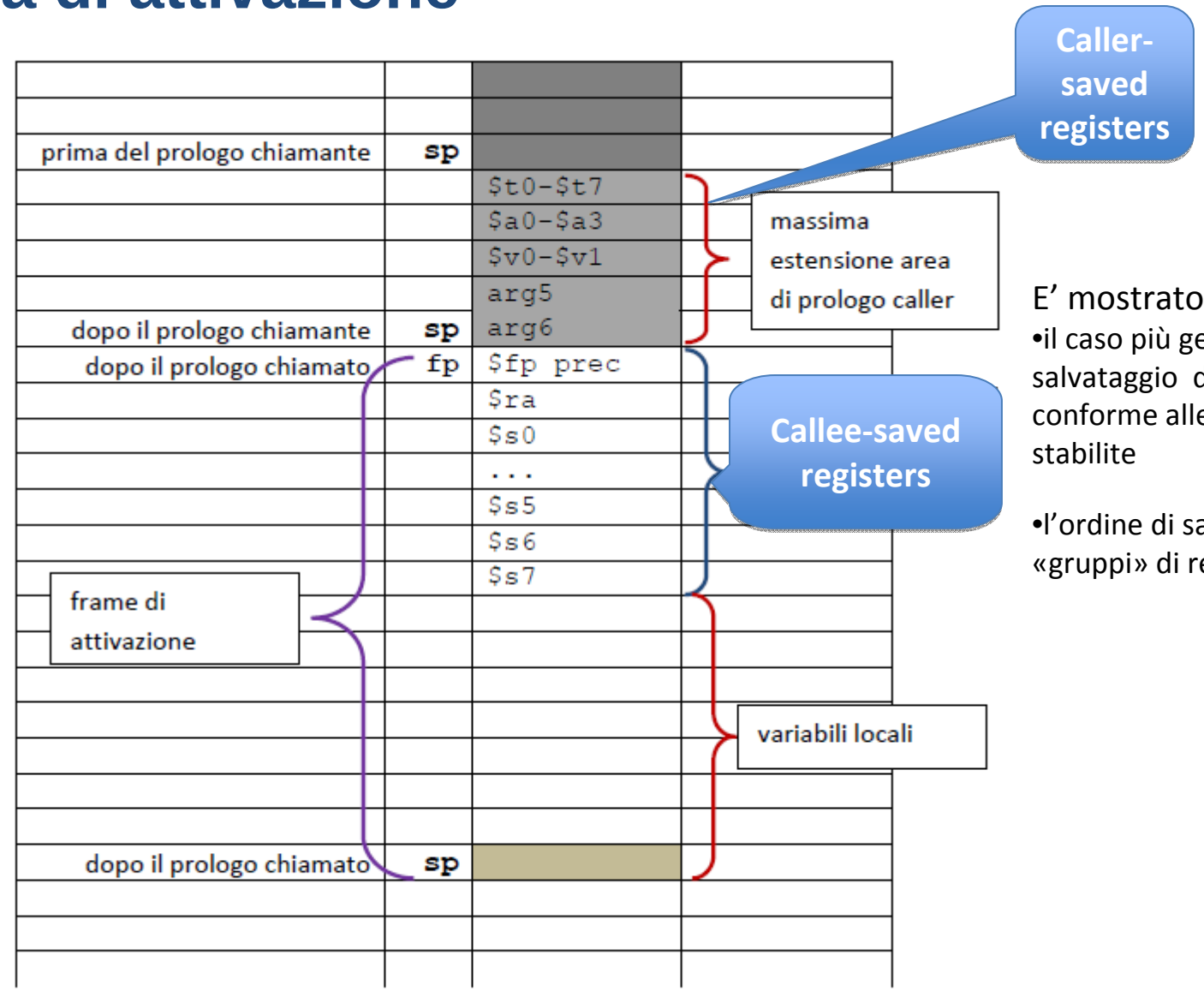
- Il salvataggio di un qualsiasi registro viene fatto solo se necessario.
  - ***fp*** è salvato solo se lo si usa per referenziare parole di memoria in stack
  - ***ra*** non viene salvato in procedure foglia

- Gli argomenti di un sottoprogramma vengono passati in ordine di elencazione
  - primi quattro in *a0-a3*
  - poi in stack (ad esempio, *arg5* e *arg6*)

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



Area di attivazione



E' mostrato

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

Variabile locale nominale

la variabile locale può essere gestita in vari modi, secondo il tipo di variabile e il grado di ottimizzazione del codice, e anche in dipendenza di come la variabile viene utilizzata

variabile di tipo scalare o puntatore – due modi

- associata a un registro del blocco *s*0 - *s*7, se per altro motivo non deve avere un indirizzo di memoria – vedi precisazioni sotto
- altrimenti nell'area di attivazione della funzione

variabile di tipo scalare, ma che viene ***anche acceduta tramite un puntatore*** – un solo modo

- nell'area di attivazione della funzione, perché deve avere un indirizzo

variabile di tipo array (o struct) – un solo modo

- sempre nell'area di attivazione della funzione
- convenzione: la variabile di tipo array (o struct) è allocata sullo stack a partire dagli indirizzi più bassi di memoria (quindi arry[0] sarà all'indirizzo di memoria più basso

Se una variabile locale è associata ad un registro del blocco *s*0 - *s*7, il registro deve essere preventivamente salvato

Convenzioni di chiamata Caller (1)

prologo del chiamante

- scrive in *a*0 - *a*3 i paramenti in ingresso alla funzione
  - trascuriamo il caso di più di quattro parametri
- se il chiamante intende usare, quando la funzione sarà rientrata – e quindi dopo l'istruzione di chiamata – i registri *t*0 – *t*9, o alcuni di essi, li deve salvare esplicitamente sullo stack e poi ripristinarli dopo il rientro del chiamato
- allo stesso modo, se occorre preservarli, salva sulla pila gli argomenti a0 - a3 e il valore dei registri v0 – v1

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 (fp prec)
  - viene aggiornato in modo da puntare alla cima dell'area (frame) di attivazione, cioè alla parola di stack che contiene il valore appena salvato
- se la funzione non è foglia
  - il registro $ra viene salvato sulla pila perché sarà usato (e quindi 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, si 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 della funzione
- ripristina dalla pila i registri *t*0 – *t*9 che con valori inalterati dopo il rientro della funzione
- trova nel registro *v*0 il valore di uscita dalla funzione

Esempio 1

varloc allocata in registro *s*0 (senza *fp*)

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

area di attivazione (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 allocata in pila (senza *fp*)

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

area di attivazione (8 byte)

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

```
F:  .eqv  RA, 4  // spiazzamento in pila di ra
    .eqv  A, 0   // spiazzamento in pila di a
    addiu $sp, $sp, -8   // crea area
    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

varloc allocata in pila (con *fp*)

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

area di attiv. (12 byte)

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

```
F:  addiu $sp, $sp, -12  // crea area
    sw    $fp, 8($sp)    // salva fp chiamante
    addiu $fp, $sp, 8    // sposta fp (punta a fp
salvato)
    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    $ra, -4($sp)   // 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: spiazzamento *ra* e varloc riferito a *fp*

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

Valutazione di espressioni algebriche

Le espressioni algebriche

I compilatori traducono in linguaggio macchina la valutazione di una espressione algebrica affrontando in modo ottimizzato i seguenti passi schematici:

- Derivazione dell'albero sintattico dell'espressione
- Ordinamento dei nodi dell'albero
- Assegnamento dei registri

noi adottiamo una «regola» semplice e intuitiva

che non richiede la derivazione dell'albero sintattico e non ottimizza la traduzione in modo completo

Valutazione di espressioni algebriche

espressione: a + b – 5 – (d – a) + c

ma anche: a + b – (c + funz(b – c, d))

Procedimento:

- Completa l'espressione con le parentesi associando da sinistra
- Ripeti fino ad aver esaminato tutta l'espressione
  - ✓ valuta il primo operatore valutabile (da sinistra)
  - ✓ un operatore è valutabile se le sue parti sinistra e destra sono: una variabile oppure una costante oppure una sottoespressione già calcolata in un registro di tipo *t*

Nota: le operazioni aritmetiche eseguono solo su registri

- ✓ se una variabile è in memoria deve essere prima caricata in un registro
- ✓ se è associata ad un registro (argomenti, var. locali …) si usa il registro senza modificarlo

Esempio 1: con variabili globali

espressione: a + b – 5 – (d – a) + c

➢Completa l'espressione con le parentesi associando da sinistra (la parentesi più esterna potremmo anche ometterla)

((((a + b) – 5) – (d – a))+ c)

➢Valuta l'espressione

```
lw      $t0, A
lw      $t1, B
add     $t0, $t0, $t1   # ( )
subi    $t0, $t0, 5     # ( )
lw      $t1, D
lw      $t2, A
sub     $t1, $t1, $t2   # ( )
sub     $t0, $t0, $t1   # ( )
lw      $t1, C
add     $t0, $t0, $t1   # ( )
```

Regole implicite utilizzate:

➢si riscrive il risultato nel registro con indice minore

➢si riutilizzano i registri

Esempio 2: con variabili globali e chiamata di sottoprogramma

espressione: a + b – (c + funz(b – c, d))

- Completa l'espressione con le parentesi associando da sinistra

((a + b) – (c + funz(b – c, d)))

- Valuta l'espressione

```
lw      $t0, A
lw      $t1, B
add     $t0, $t0, $t1  # ( )
lw      $t1, B
lw      $t2, C
sub     $a0, $t1, $t2  # primo parametro di funz
lw      $a1, D         # secondo parametro di funz
addiu   $sp, $sp, -4
sw      $t0, 0($sp)    # salva in stack t0 usato dopo invocazione
jal     FUNZ
lw      $t1, C
add     $t1, $v0, $t1  # ( )
sub     $t0, $t0, $t1  # ( )
```