Università degli Studi di Milano
Corso di Laurea in
Informatica per la Comunicazione Digitale
Architettura degli Elaboratori

Esempi di programmi in MIPS

Chiamate di funzione

Marco Tarini

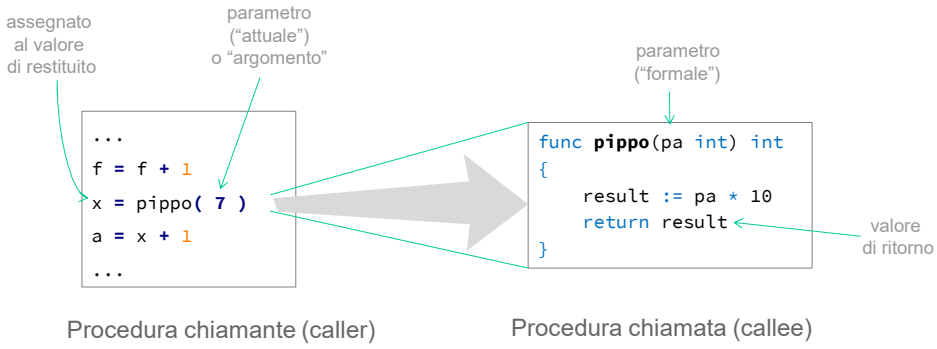
147

Procedure (o funzioni)

- Programmando ad alto livello, vogliamo strutturare il programma in unità funzionali dette **procedure**
  - o anche (nei vari linguaggi) **funzioni**, **routines**, **subroutines**, **subprograms**…
  - è un altro dei princìpi della *programmazione ben strutturata*
- Esempi:
  - procedura che volge in maiuscolo una data stringa,
  - procedura calcola l'interesse cumulato di una certa somma di denaro,
  - procedura che legge il nome dell'utente da tastiera
  - procedura che verifica una password…
- Le procedure vengono **invocate** all'occorrenza, ogni volta che sia necessario
  - dal programma principale,
  - oppure, da un'altra procedura

148

Chiamata a procedura ad alto livello (es: in Go)

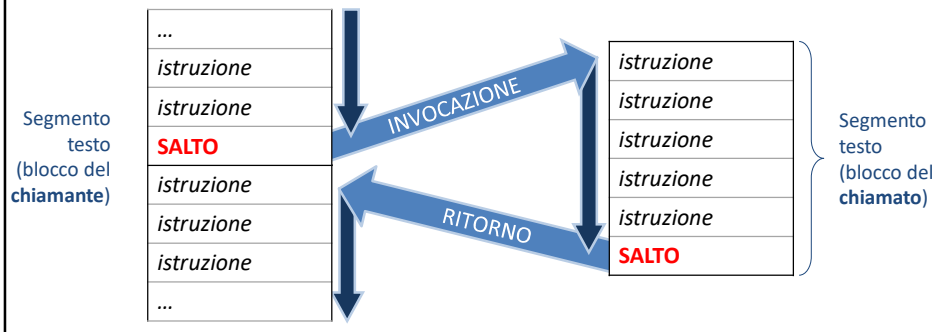


Caller e callee comunicano attraverso:

- passaggio di **parametri** (o **argomenti**) di input (dal caller al callee)
- restituzione di **valori (di ritorno)** di output (dal callee al caller)

150

Chiamata a procedura a basso livello



153

A quale indirizzo saltare per tornare al chiamante?

- Non può essere un indirizzo fissato (es, uno indicato da una label) perché la procedura può essere invocata da qualsiasi riga del programma!
  - o anche da più parti dello stesso programma
- Il MIPS dedica un registro a memorizzare l'indirizzo di ritorno:
  - `$ra` : «Return Address»
  - è il numero $31, cioè l'ultimo, ma non è necessario saperlo o ricordarselo, basta chiamarlo col suo sinonimo
- L'istruzione jump ha una variante «jump and link» che, prima di sovrascrivere il PC, salva in `$ra` il valore PC+4 :
  - `jal <indirizzo>` : Jump-and-link
- Il codice invocante salta all'indirizzo di partenza della procedura con `jal`
  - Come per la normale jump, si usa una label per specificare l'indirizzo di arrivo
- La procedura restituisce il controllo al chiamante con jump register: `jr $ra`

154

JAL: Jump and Link

Registro:	$0	$1	$2	$3	$4	$5	$6	$7
Sinonimo:	$r0	$at	$v0	$v1	$a0	$a1	$a2	$a3
Registro:	$8	$9	$10	$11	$12	$13	$14	$15
Sinonimo:	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7
Registro:	$16	$17	$18	$19	$20	$21	$22	$23
Sinonimo:	$s0	$s1	$s2	$s3	$s4	$s5	$s6	$s7
Registro:	$24	$25	$26	$27	$28	$29	$30	$31
Sinonimo:	$t8	$t9	$k0	$k1	$gp	$sp	$s8	$ra

RETURN ADDRESS

155

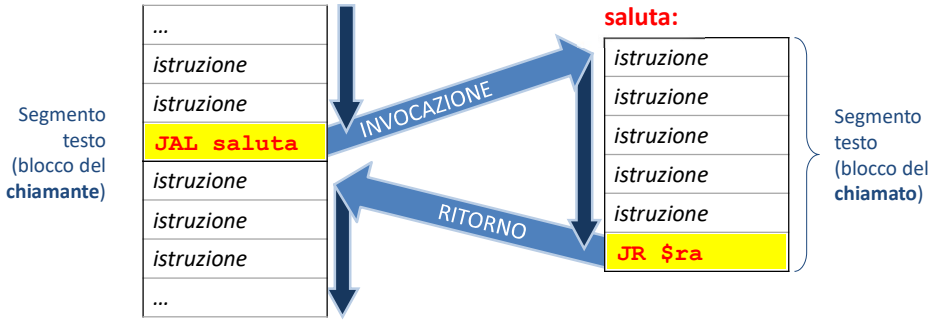
Salti di invocazione e ritorno

Salto di invocazione:

- Indicare l'inizio della procedura con una etichetta
- Saltare con una `jal` a quell'etichetta

Salto di ritorno

- Saltare con una `jr` al Return address



156

Implementazione e uso delle procedure

- Chi implementa una procedura (chi scrive il suo codice) è spesso una programmatore diverso da chi la usa (chi scrive il codice che la invoca)
  - per esempio: il primo è l'autore di una «libreria», il secondo è un utente di questa libreria
  - oppure: sono membri diversi di un team di sviluppo
- I linguaggi ad alto livello forniscono meccanismi con cui i due programmatori possono coordinarsi
  - ad esempio, per specificare…
  - …quali dati la procedura prenda in input (se alcuno)
  - …quali dati restituisca in output (se alcuno)
- a basso livello, si adottano invece una serie di convenzioni
  - che sta al programmatore (o al compilatore) rispettare
  - vediamo quelli usati nel MIPS

157

Come il chiamante comunica alla procedura i «parametri»

- Molte procedure si aspettano degli **input**
  - es: una procedura che volge in maiuscolo una stringa deve sapere l'indirizzo della stringa su cui lavorare
- Ad alto livello, questi sono detti gli **argomenti** (o i **parametri**) della procedura
  - Una procedura può averne nessuno, uno, o più di uno
- In MIPS, dedichiamo alcuni registri a memorizzare gli argomenti: `$a0`, `$a1`, `$a2`, `$a3` («a» sta per argomento)
- Convenzione:
  (che il programmatore assembly / il compilatore deve rispettare)
  - Il chiamante mette i valori dei parametri in `$a0`..`$a3` prima di invocare la procedura (quelli necessari)
  - La procedura assumerà di trovare il valore degli argomenti in `$a0`..`$a3`

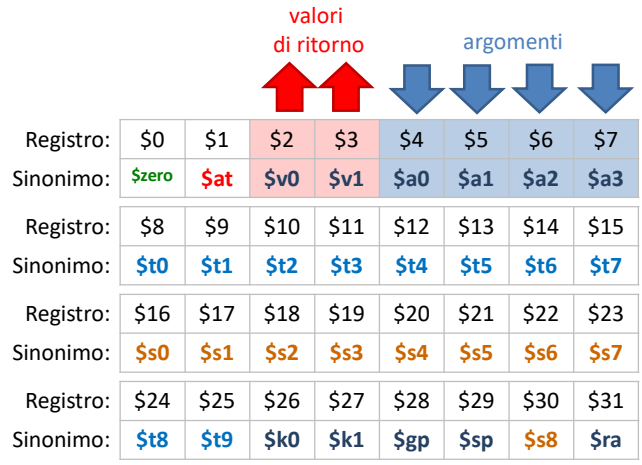
158

Come la procedura comunica al chiamante i suoi «valori di ritorno»

- Molte procedure restituiscono degli **output**
  - es: una procedura che calcola l'interesse cumulato deve comunicare al chiamate questo valore
- Ad alto livello, questo è detto il **valore di ritorno** della procedura
  - Una procedura può averne uno, nessuno, o più di uno
- In MIPS, dedichiamo alcuni registri a memorizzare gli eventuali valori di ritorno: `$v0`, `$v1` («v» sta per valore di ritorno)
- Convenzione
  (che il programmatore assembly / il compilatore deve rispettare)
  - Prima di restituire il controllo, la procedura mette in `$v0` (e/o `$v1`) gli eventuali valore/i da restituire
  - Al ritorno, il caller trova in `$v0` (e/o `$v1`) l'eventuale valore/i restituito/i dalla procedura

159

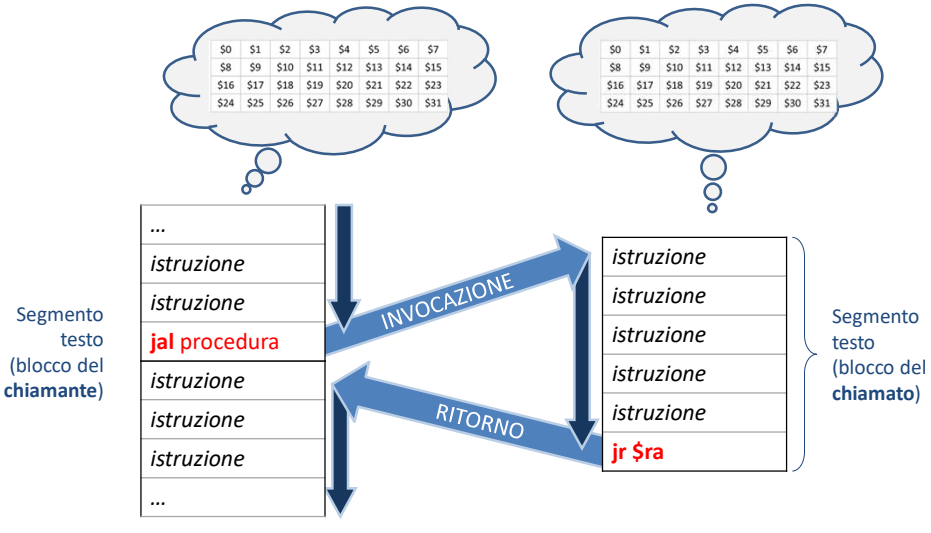
Input e output di una procedura



Registro:	$0	$1	$2	$3	$4	$5	$6	$7
Sinonimo:	$zero	$at	$v0	$v1	$a0	$a1	$a2	$a3
Registro:	$8	$9	$10	$11	$12	$13	$14	$15
Sinonimo:	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7
Registro:	$16	$17	$18	$19	$20	$21	$22	$23
Sinonimo:	$s0	$s1	$s2	$s3	$s4	$s5	$s6	$s7
Registro:	$24	$25	$26	$27	$28	$29	$30	$31
Sinonimo:	$t8	$t9	$k0	$k1	$gp	$sp	$s8	$ra

160

Problema:
i registri sono gli stessi!



161

Registri $s e $t

- Problema: i registri sono usati tanto dalla procedura quanto dal chiamante
  - Quindi, dopo una chiamata ad una procedura, il chiamante rischia di trovare i registri che stava utilizzando completamente cambiati («sporcati», «sovrascritti», «cancellati» dal chiamante)
- Ogni linguaggio assembly usa delle convenzioni per consentire a chiamante e chiamato di usare registri senza interferire uno con l'altro
- In MIPS, si adotta questa convenzione:
  - gli otto registri `$s0` .. `$s7` (s = save) devono essere **preservati** dalla procedura: quando la procedura restituisce il controllo, il chiamante deve trovare in questi registri gli stessi valori che avevano al momento dell'invocazione
  - i dieci registri `$t0` .. `$t9` (t = temp) possono invece essere modificati dalla procedura: il chiamante "sa" che invocare una procedura potrebbe modificare questi registri.

162

Convenzione sull'uso dei registri da parte delle procedure

Registro:	$0	$1	$2	$3	$4	$5	$6	$7
Sinonimo:	$zero	$at	$v0	$v1	$a0	$a1	$a2	$a3
Registro:	$8	$9	$10	$11	$12	$13	$14	$15
Sinonimo:	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7
Registro:	$16	$17	$18	$19	$20	$21	$22	$23
Sinonimo:	$s0	$s1	$s2	$s3	$s4	$s5	$s6	$s7
Registro:	$24	$25	$26	$27	$28	$29	$30	$31
Sinonimo:	$t8	$t9	$k0	$k1	$fp	$sp	$s8	$ra

deve rimanere invariato

può essere modificato dalla procedura

163

Progetti multi-sorgente e linking

- Un progetto multi-sorgente è costituito da più di un sorgente assembly
- L'assembler traduce separatamente ogni sorgente
  - producendo un codice in linguaggio macchina per ogni sorgente
- Linking: i codici prodotti vengono legati insieme (linked) in un unico codice binario finale
  - Semplicemente, concatenando le parti "text" fra loro, e le parti "data" fra loro
  - Nota: gli indirizzi finali a cui sono memorizzati istruzioni e dati divengono dunque noti solo *dopo* il linking
  - Solo dopo il linking, le nostre etichette (nell'assembly) vengono convertite in indirizzi (nel linguaggio macchina)

164

Progetti multi-sorgente e linking

- E' possibile condividere etichette fra sorgenti assembly diversi
  - Cioè, consentire ad un file sorgente di usare un'etichetta definita in un sorgente diverso dello stesso progetto "multi-sorgente"
  - Basta dichiarare che queste etichette sono "globali", inserendo, nel sorgente che le definisce, l'apposita direttiva:

```
.globl etichetta
```

  - Nota: etichette diverse usate da sorgenti diversi di uno stesso progetto possono condividere uno stesso nome (per es: "loop", "fine", "else", etc), se non sono globali

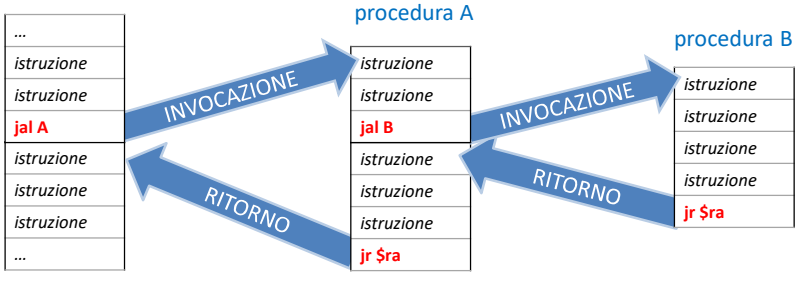
165

Progetti multi-sorgente e procedure

- Un caso classico è porre una procedura o un insieme di procedure in un sorgente assembly *separato*
  - Il progetto è "multi-sorgente"
  - Analogo ad una libreria ad alto livello
  - Per poter essere invocate dagli altri file, le etichette a cui iniziano le procedure devono essere dichiarate come "globali"
- Un unico sorgente conterrà la funzione globale "main"
  - da cui per convenzione inizia l'esecuzione
- Vedi il codice di esempio sul sito!

166

Nota: non stiamo valutando il caso delle «procedure annidate»



Riesci a vedere qual è il problema?
(Non ne vedremo la soluzione, in questo corso:
ma il compilatore ha il suo bel daffare per accertarsi che anche questo caso funzioni)

167