



Università degli Studi di Milano
Dipartimento di Informatica "Giovanni Degli Antoni"
Corso di Laurea Triennale in Informatica

Architettura degli Elaboratori II

Laboratorio

Procedure 1/2

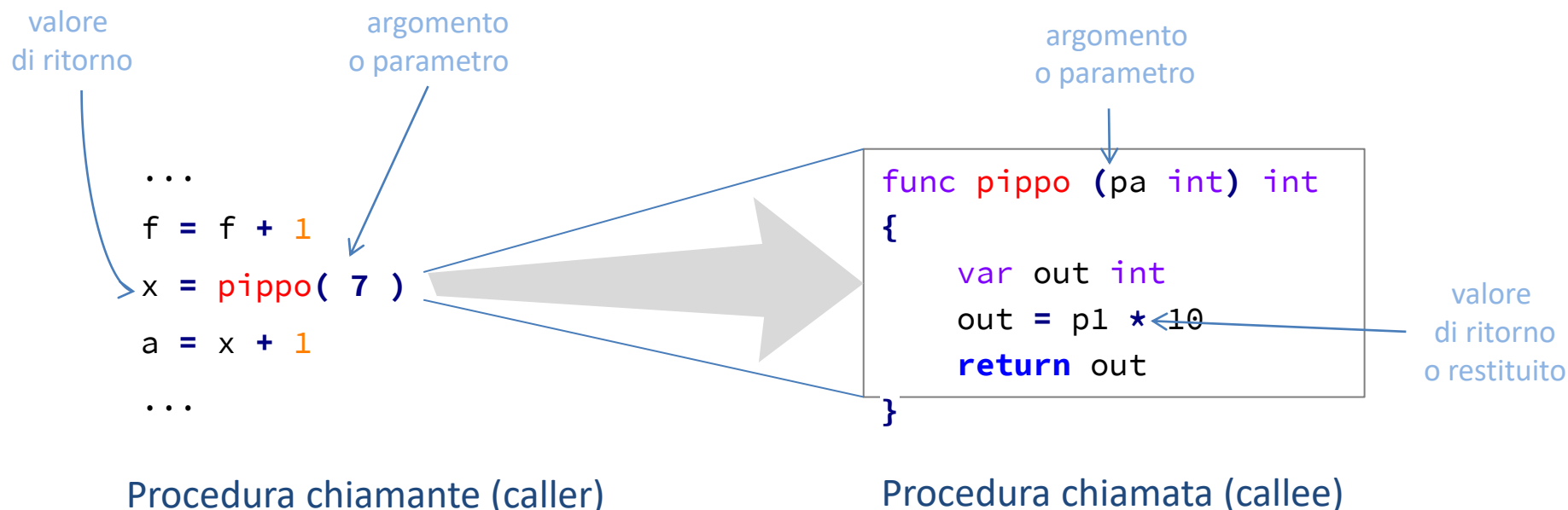
Procedure

- Programmando ad alto livello, spesso organizziamo il programma in unità funzionali dette **procedure** (o anche, nei vari linguaggi, funzioni, routines, subroutines, sottoprogrammi ...)
- Esempi:
 - procedura che volge in maiuscolo una data stringa
 - procedura che 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 **invoke** all'occorrenza, ogni volta che sia necessario
 - dal programma principale
 - da un'altra procedura

Procedure

- Chi implementa una procedura (ne scrive il codice) e chi la utilizza (scrive un codice che la invoca) sono spesso persone diverse, ad esempio:
 - l'autore di una libreria scrive una procedura
gli utente della libreria la utilizza
 - membri diversi di un team di sviluppo si accordano sulle procedure da usare, e uno sviluppatore scrive codice «come se» le funzioni esistessero già, mentre un altro scrive le procedure
- I linguaggi ad alto livello impongono regole fisse con cui sviluppatore e utilizzatore possono coordinarsi. Per esempio, la sintassi con cui dichiarare e invocare una procedura. Se non rispettiamo queste regole il codice non compila o genera errori
- **A basso livello**, non esistono regole! Si adottano invece una serie di convenzioni **autoimposte**: sta al programmatore (noi), o al compilatore, rispettarle

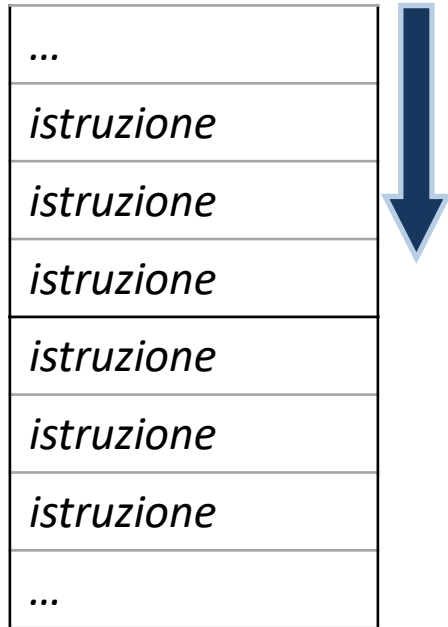
Chiamata a procedura ad alto livello (es: in Go)



Caller e callee interagiscono attraverso:

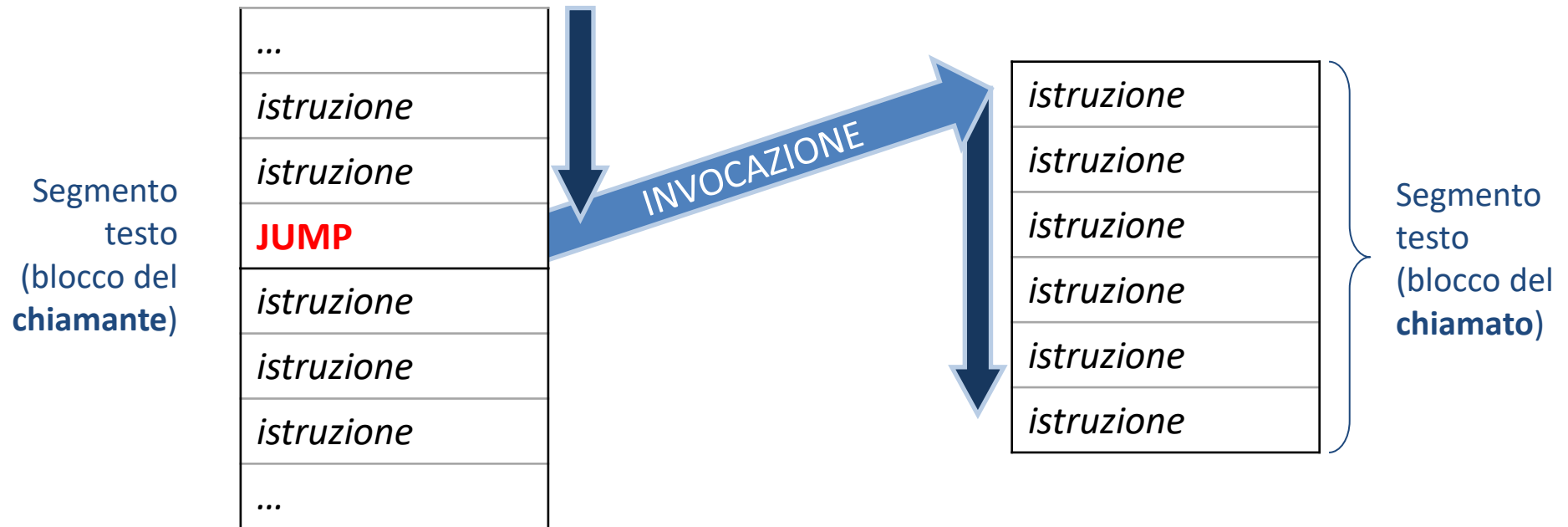
- passaggio di **parametri** di input (dal caller al callee)
- **ritorno di valori** di output (dal callee al caller)

Chiamata a procedura a basso livello

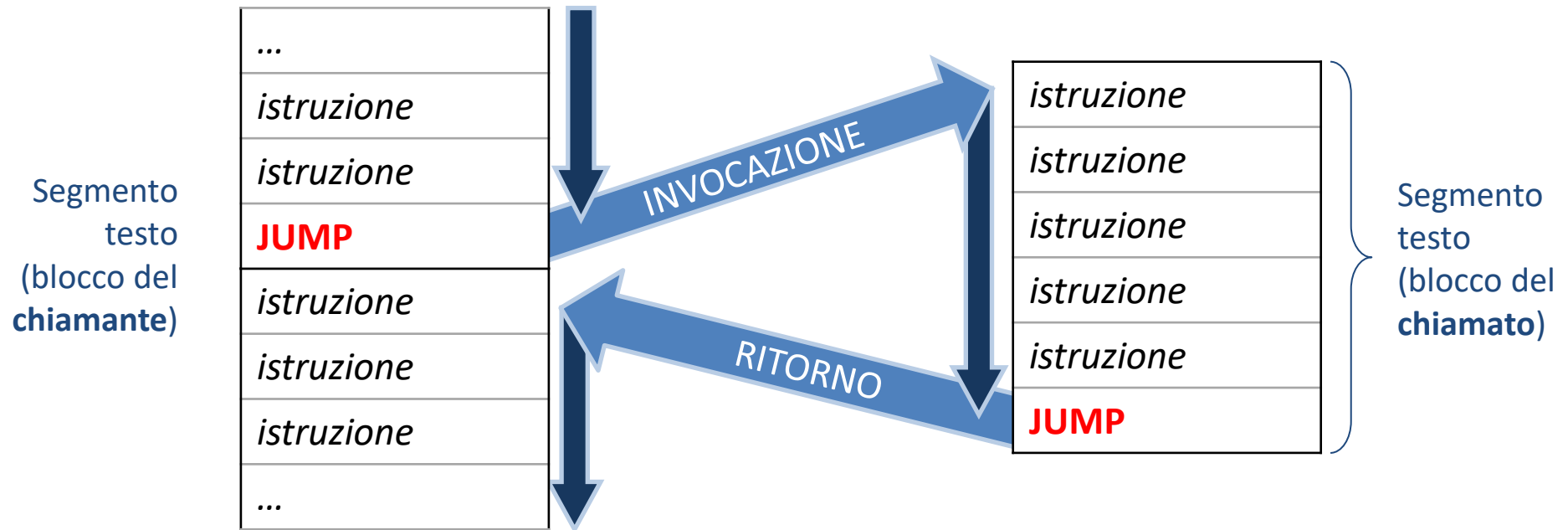


Segmento
testo
(blocco del
chiamante)

Chiamata a procedura a basso livello



Chiamata a procedura a basso livello



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

Salti di invocazione e ritorno

- In MIPS, un registro è dedicato a memorizzare l'indirizzo di ritorno:
 - **\$ra** : «Return Address»
- Le istruzioni di jump hanno una variante «and link» che, prima di sovrascrivere il PC (per fare il salto), salvano in **\$ra** il valore PC+4 (cioè l'indirizzo a cui tornare al rientro dalla procedura):
 - **jal** <label> : Jump-and-link (j con link)
 - **jalr** <registro> : Jump-and-link register (jr con link)

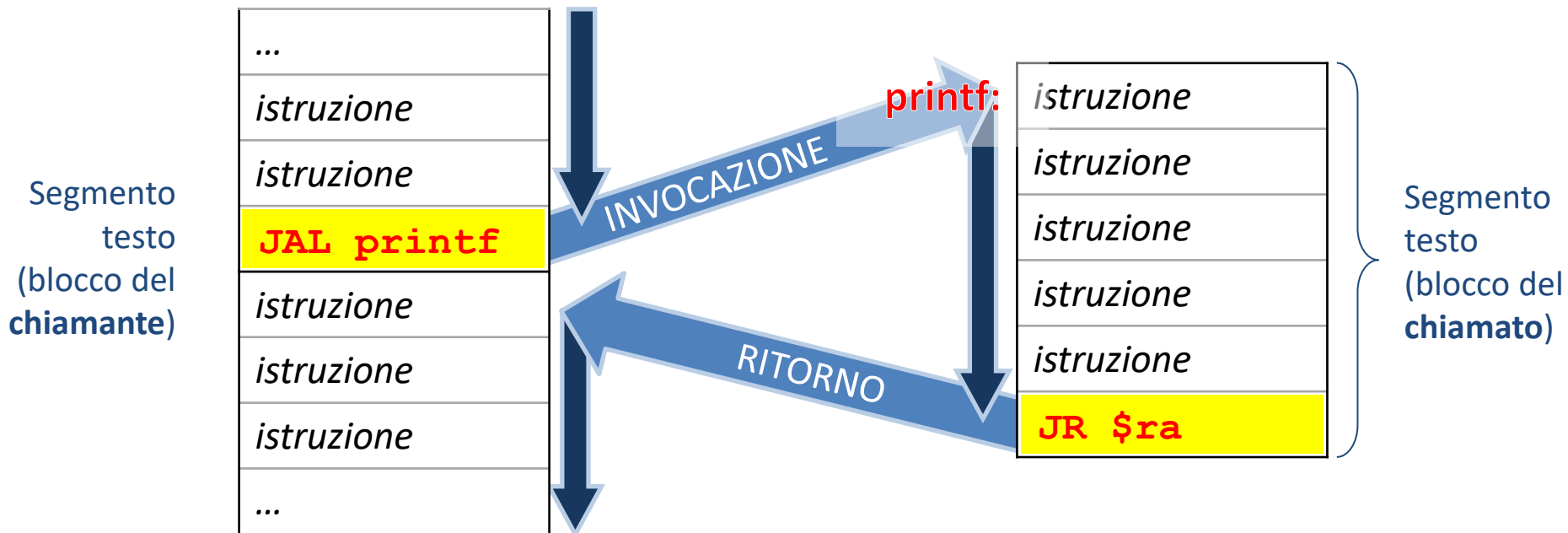
Salti di invocazione e ritorno

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



Comunicare alla procedura i suoi 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 gli **argomenti** (o i **parametri**) della procedura
- In MIPS, dedichiamo alcuni registri a memorizzare questi argomenti: **\$a0**, **\$a1**, **\$a2**, **\$a3** (a = argomento)
- Convenzione: (che sta ai programmatori / compilatori / studenti rispettare)
 - Il chiamante mette i valori dei parametri in **\$a0..\$a3** prima di invocare la procedura (quelli necessari)
 - La procedura assumerà di trovare gli input necessari in **\$a0..\$a3**

Comunicare al chiamante il valore di ritorno

- Molte procedure restituiscono degli output
 - es: una procedura che calcola l'interesse cumulato deve comunicare al chiamante il valore calcolato
- Ad alto livello, questo è il **valore di ritorno** della procedura (uno o più)
- In MIPS, dedichiamo alcuni registri a memorizzare i valori di ritorno: **\$v0**, **\$v1** (v = valore di ritorno)
- Convenzione: (che sta a noi rispettare)
 - Prima di restituire il controllo, la procedura mette in **\$v0** (e/o **\$v1**) il valore/i da restituire
 - Al ritorno il caller assume di avere in **\$v0** (e/o **\$v1**) il valore/i restituito/i dalla procedura

Come facciamo se abbiamo bisogno di più di 4 argomenti?

- Si usa lo stack
- Dal 5to argomento il poi:
 - Il chiamante: prima di invocare, fa una **push** nello stack dell'argomento
 - Il chiamato: per prima cosa, prende l'argomento dallo stack, con una **pop**
- Nota: chiamante e chiamato devono sapere che questo è il caso per ogni data funzione
 - Quindi, quanti argomenti servono oltre i primi 4
 - Lo stack viene compromesso irrimediabilmente se non si mettono d'accordo! (viene fatta una push o, peggio, una pop di troppo)
- Info: in altri ISA, questo è il modo convenzionale di passare *tutti* gli argomenti

```
addi $sp $sp -4  
sw $** ($sp)
```

```
lw $** ($sp)  
addi $sp $sp 4
```

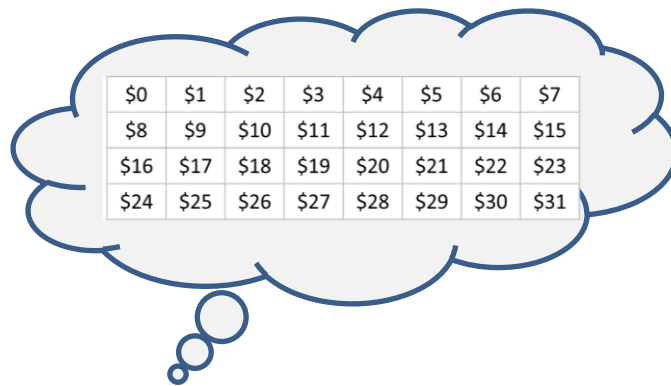
Come facciamo se abbiamo bisogno di più di 2 valori di ritorno?

- Si usa lo stack
- Dal 3zo valore di ritorno il poi:
 - Il chiamato: prima di restituire il controllo, fa una **push** nello stack del valore prodotto
 - Il chiamante: dopo l'invocazione, prende il valore dallo stack, con una **pop**
- Nota: chiamante e chiamato devono sapere che questo è il caso per ogni data funzione
 - Quindi, quanti valori vengono restituiti oltre i primi due
 - Lo stack viene compromesso irrimediabilmente se non si mettono d'accordo! (viene fatta una push o, peggio, una pop di troppo)
- Info: in altri ISA, questo è il modo convenzionale di restituire *tutti* i valori

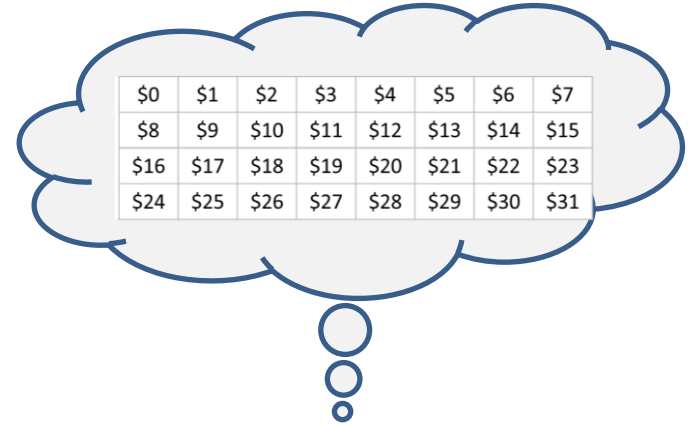
```
addi $sp $sp -4  
sw $** ($sp)
```

```
lw $** ($sp)  
addi $sp $sp 4
```

Problema: i registri sono gli stessi!



\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7
\$8	\$9	\$10	\$11	\$12	\$13	\$14	\$15
\$16	\$17	\$18	\$19	\$20	\$21	\$22	\$23
\$24	\$25	\$26	\$27	\$28	\$29	\$30	\$31



\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7
\$8	\$9	\$10	\$11	\$12	\$13	\$14	\$15
\$16	\$17	\$18	\$19	\$20	\$21	\$22	\$23
\$24	\$25	\$26	\$27	\$28	\$29	\$30	\$31

Segmento
testo
(blocco del
chiamante)

...
istruzione
istruzione
jal procedura
istruzione
istruzione
istruzione
...

INVOCAZIONE

RITORNO

Segmento
testo
(blocco del
chiamato)

istruzione
istruzione
istruzione
istruzione
istruzione
jr \$ra

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»)
- Ogni linguaggio assembly usa delle convenzioni per consentire a chiamante e chiamato usare i registri per non interferire
- In MIPS, adottiamo una 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 da una procedura: il chiamante sa che invocare una procedura potrebbe modificare («sporcare») questi registri

Convenzione sull'uso dei registri da parte delle procedure

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	\$fp	\$sp	\$s8	\$ra



deve rimanere
invariato



può essere modificato
dalla procedura

Anche detti: registri caller-saved e 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 \$sp \$fb

un callee non può sovrascrivere permanentemente 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

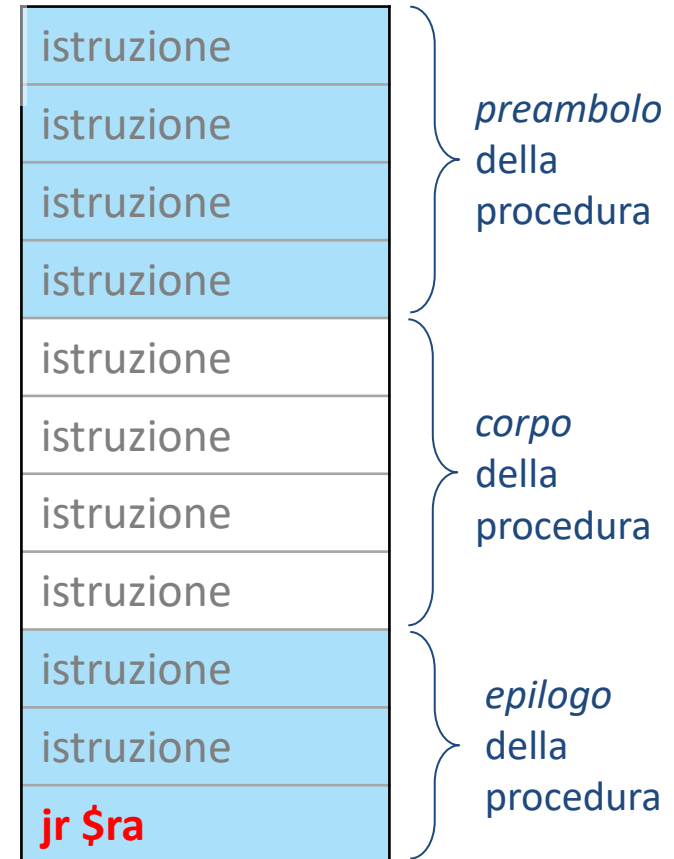
Esempio: `main` ha un dato importante nel registro `$s1` e invoca `f`; `f` salva `$s1` sullo stack prima di utilizzarlo, una volta terminato lo ripristina.

Registri \$s e \$t: per la procedura

La procedura può rispettare la convenzione attraverso diversi modi

- Modo 1: non scrivere mai i registri \$s
- Modo 2: salvare i registri \$s nello stack

miaFunz:

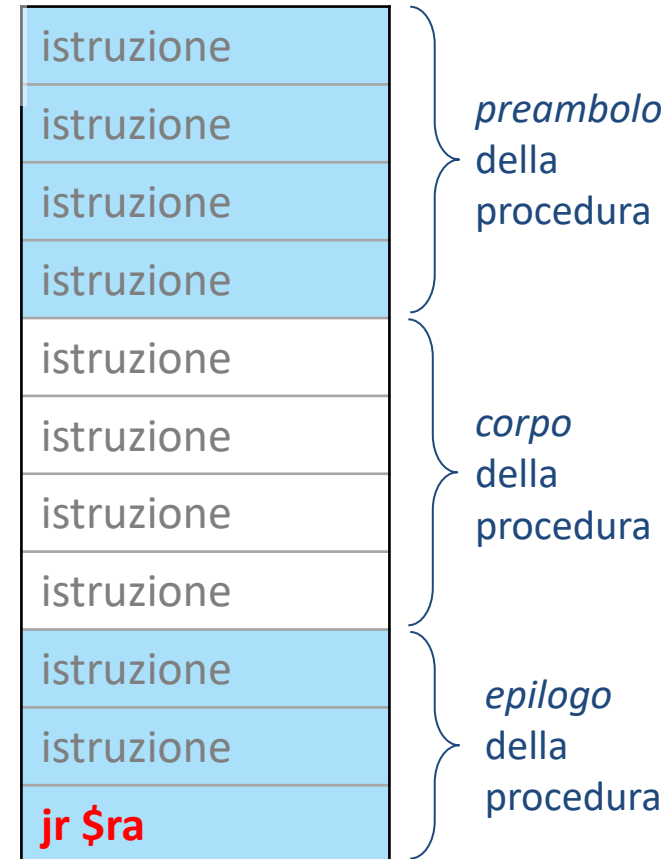


Registri \$s e \$t: per la procedura

La procedura può rispettare la convenzione attraverso diversi modi

- Modo 1: non scrivere mai i registri \$s
 - usare quindi solo i registri \$t
- Modo 2: salvare i registri \$s nello stack
 - Prima di scrivere su un dato registro \$s, (ad esempio, nel «*preambolo*» della proc.) o cmq salvarne una copia con una **push** nello stack
 - Poi, usare questi registri come normale
 - prima di restituire il controllo al chiamante, (ad esempio, nel «*epilogo*» della proc, subito prima della *jr \$ra* finale) ripristinare il valore originale di questi registri con una **pop**
 - Nota: dal punto di vista del chiamante, lo stack rimane inalterato

miaFunz:



Conclusione: manuale per invocare una procedura

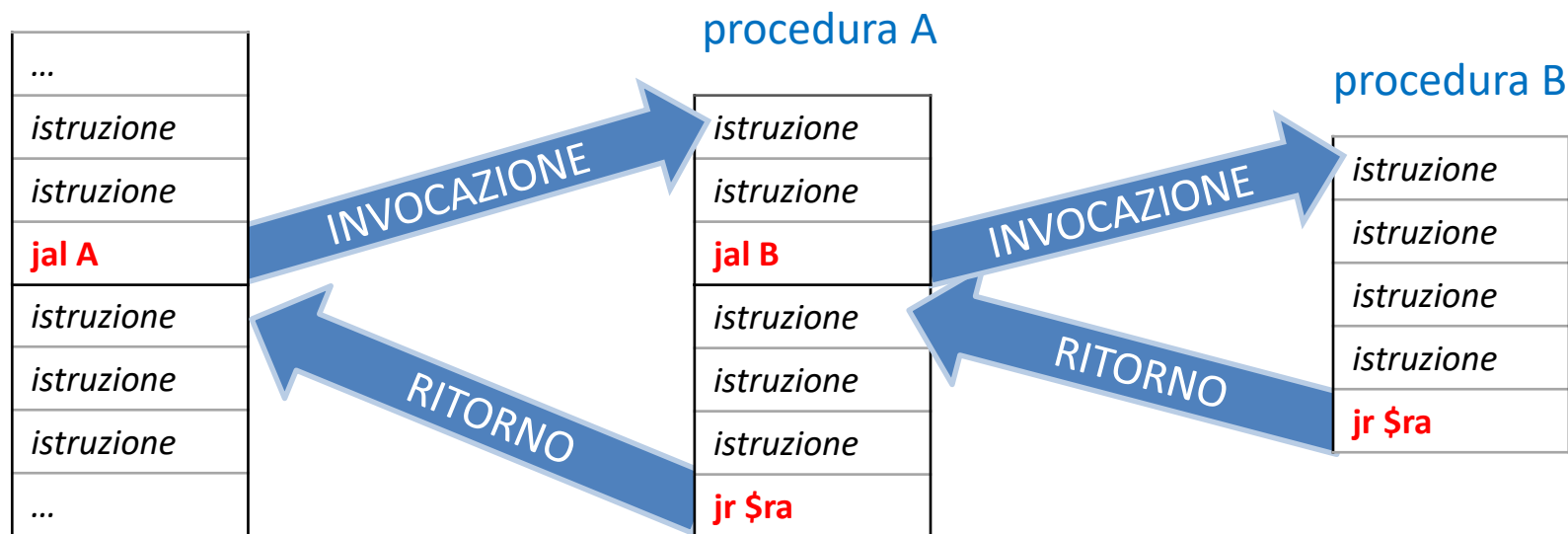
1. Caricare in \$a0.. \$a3 i parametri della procedura (se previsti)
2. Se necessario, salvare una copia dei registri \$t nei registri \$s oppure sullo stack frame (vale anche per \$a0..\$a3, \$v0 e \$v1)
3. Invocare la procedura: jal <label>
4. Trovare in \$v0 (o \$v1) l'eventuale valore restituito (se previsto)
5. Se necessario ripristinare il valore dei registri salvati nel passo 2

Manuale per scrivere una procedura

1. Preambolo:
salvare una copia dei registri $\$s$ che si intende usare nello stack frame
 - con store word agli indirizzi $\$sp - 4$, $\$sp - 8$, $\$sp - 12$, ...
2. implementare la procedura (scrivere codice)
 - leggendo gli eventuali input da $\$a0$.. $\$a3$
 - scrivendo liberamente su $\$t0$.. $\$t9$
 - scrivendo su $\$s0$.. $\$s8$ che siano stati salvati precedentemente
 - scrivere l'eventuale output in $\$v0$ (e/o $\$v1$)
3. Epilogo: ripristinare tutti i registri salvati nel passo 1
 - con altrettante load word agli stessi indirizzi
4. restituire il controllo al chiamante
 - jr $\$ra$

nota: se la procedura ne invoca un'altra, la situazione si complica. Vedi prossima lezione

Prossima lezione: invocazione di procedura annidate





Università degli Studi di Milano
Dipartimento di Informatica "Giovanni Degli Antoni"
Corso di Laurea Triennale in Informatica

Architettura degli Elaboratori II

Laboratorio