UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Procedure, linker e Assembly

Prof.ssa Giulia Cisotto

giulia.cisotto@units.it

Trieste, 12 marzo 2026

PROCEDURE

Una procedura è un meccanismo per organizzare in modo comprensibile e riutilizzabile il codice

Le procedure consentono ai programmatori di concentrarsi su una parte del problema alla volta
Possono essere chiamate senza bisogno di sapere come sono fatte “dentro”.

I 6 passi di una procedura:

- Setting dei parametri in un luogo accessibile alla procedura
- Trasferire il controllo alla procedura e salvare l’indirizzo dell’istruzione dove tornare dopo la chiamata della procedura (usare l’istruzione jal)
- Acquisire risorse per l’esecuzione della procedura
- Eseguire il compito richiesto
- Mettere il risultato in un luogo accessibile al chiamante
- Restituire il controllo al punto di partenza (usare l’istruzione jr)

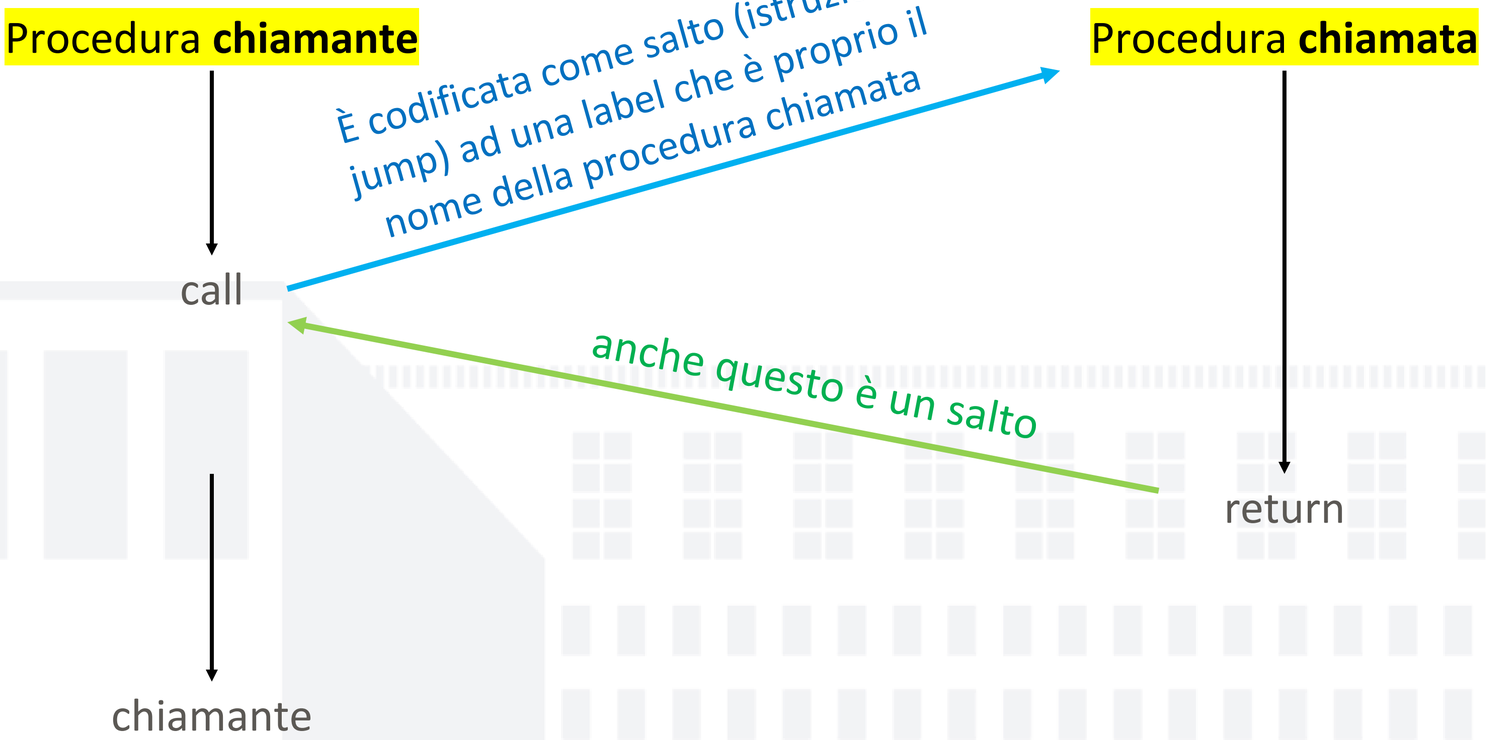
PROBLEMI APERTI

Cosa succede se una procedura ne chiama un’altra?

Se una procedura usa registri, cosa succede del contenuto lasciato nei registri dal chiamante?

Dove stanno le variabili locali della procedura?

CHIAMATA A FUNZIONI

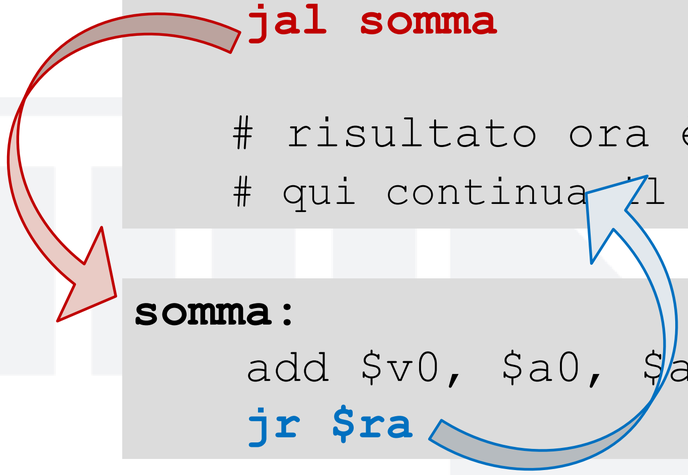


ESEMPIO

```
main:
    li $a0, 5           # primo parametro
    li $a1, 7           # secondo parametro

    jal somma           # chiama la procedura

    # risultato ora è in $v0
    # qui continua il programma
```



```
somma:
    add $v0, $a0, $a1   # v0 = a0 + a1
    jr $ra              # ritorna al chiamante
```

`jal somma`
salva l'indirizzo di ritorno in **$ra**
salta all'etichetta somma

La procedura «somma» usa i **parametri**:
$a0, $a1.
Il risultato viene messo in: **$v0.**
`jr $ra`: torna all'istruzione successiva alla jal.

L'alternativa sarebbe stata:

```
main:
    li $a0, 5
    li $a1, 7

    add $v0, $a0, $a1

    # qui continua il programma
```

Così però non posso riusare il codice che fa la somma in futuro, ma devo riscriverla

PROCEDURE INNESTATE

Procedure “foglia” e “non foglia”

- Una procedura foglia NON chiama altre procedure
- Una procedura non foglia chiama altre procedure

Cosa succede se una procedura ne chiama un’altra?

- Si perde il contenuto di $ra della prima chiamata??
- Procedure recorsive??
- Bisogna che una procedura “non foglia” salvi il contenuto di $ra e lo ripristini prima del ritorno

Dove salvare il contenuto dei registri $ra e $s?

- Uso dello stack

PROCEDURA CHIAMANTE

Prima di chiamare una procedura:

- impostare gli argomenti da passare alla procedura in **$a0-$a3;**
- eventuali altri argomenti sono nella memoria o nello stack
- salvare eventualmente i registri $a0-$a3 e $t0-$t9 in quanto la procedura chiamata puo' usare liberamente questi registri
- chiamare la procedura tramite l'istruzione **jal nome_procedura**

PROCEDURA CHIAMATA

Appena è stata chiamata:

- Allocare il suo stack frame ($sp = $sp – dimensione frame procedura)
- Salvare i valori disponibili nei registri $s0–$s7, $fp, $ra se intende usarle tali registri per la sua esecuzione; se per esempio la procedura non chiama un'altra procedura non è necessario salvare il registro $ra
- Settare il frame pointer (che indica l'indirizzo dell'ultima word del frame):
  $fp = $sp – dimensione frame procedura + 4

Quando ha finito la sua esecuzione:

- Mettere il valore di ritorno nel registri $v0, $v1
- Ripristinare i valori dei registri salvati sullo stack ($s0–$s7, $fp, $ra)
- Liberare lo spazio sullo stack: $sp = $sp + dimensione frame procedura
- Eseguire l'istruzione jr $ra

CONVENZIONI: REGISTRI TEMPORANEI SALVATI E NON SALVATI

Se una procedura usa registri, cosa succede del contenuto lasciato nei registri dal chiamante?

CONVENZIONI su uso dei registri $t e $s

- **I registri $t (“temporary”) non sono salvati dalla procedura**
- Il chiamante non si può aspettare di trovare immutati i contenuti dei registri $t dopo una chiamata a procedura
- I contenuti dei registri $t devono essere salvati dal chiamante prima della chiamata a procedura
- **I registri $s (“saved”) sono salvati dalla procedura**
- Il chiamante ha il diritto di aspettarsi che i contenuti dei registri $s siano immutati dopo una chiamata a procedura
- Se la procedura usa i registri $s deve salvarne il contenuto all’inizio e ripristinarlo prima del ritorno

Dove salvare il contenuto dei registri $s?

- Uso dello stack

PASSAGGIO PARAMETRI (CONVENZIONI BASE)

$a0 - $a3: registri argomento per il passaggio dei parametri

$v0 - $v1: registri valore per la restituzione dei risultati

Dal punto di vista hw sono registri come tutti gli altri, MA...
...il loro utilizzo per il passaggio di parametri e risultati è una convenzione programmativa che deve essere rispettata per consentire di scrivere procedure senza bisogno di sapere come è fatto il programma che le chiama

Passaggio parametri “per valore” o “per indirizzo”

- un parametro può essere un dato o un indirizzo!!!
- Confrontare le istruzioni la e lw. Esempio: la $s0, label

ISTRUZIONE JAL

jal <IndirizzoProcedura> (“jump and link”)

- Salta a una procedura indicata nell’istruzione e contemporaneamente crea un collegamento a dove deve ritornare per continuare l’esecuzione del chiamante
- Salva nel **registro $ra (registro 31) (“return address”)** l’indirizzo a cui tornare dopo l’esecuzione della procedura (è l’indirizzo successivo a quello dell’istruzione jal, cioè l’indirizzo in cui si trova la jal + 4)
- Tale indirizzo si trova nel registro PC (Program Counter)

ISTRUZIONE JR

jr <registro> (“jump register”)

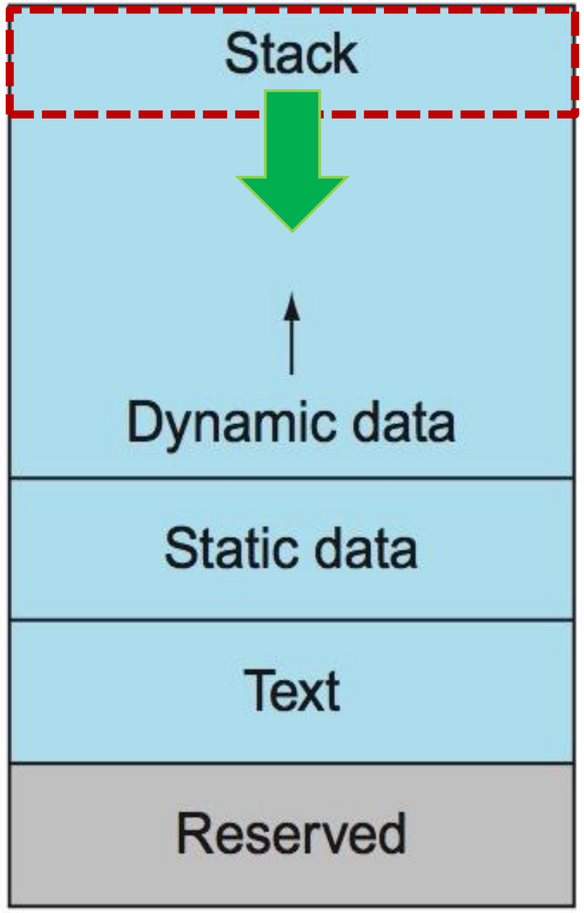
- Salta all’indirizzo contenuto in un registro
- È una istruzione **di uso generale** che consente di saltare a qualsiasi locazione di memoria, MA...

jr $ra

- Uno degli utilizzi tipici di jr
- Per realizzare il ritorno da procedura
- Saltando all’indirizzo precedentemente salvato da jal

CONVENZIONI SULL'USO DELLA MEMORIA DURANTE PROCEDURA

Usiamo lo stack!



Nota. Lo stack «cresce» verso indirizzi più bassi.

USO DELLO STACK: SALVATAGGIO REGISTRI

Cosa fa la procedura?

- **“alloca” spazio nello stack**
- decrementa **$sp** per lasciare in stack lo spazio necessario al salvataggio (1 word per ciascun registro da salvare). Nota: lo stack cresce “verso il basso”.
- salva **$ra**
- salva eventuali altri registri usando **$sp** come registro base
- ripristina i registri
- incrementa **$sp** per riportarlo alla situazione iniziale
- **jr $ra** (ritorno dalla procedura)

Approfondimenti:

- Parametri passati in stack
- “Procedure frame”: l’insieme dei dati locali (**$fp**)
- Come indirizzare variabili locali? (per fortuna ci pensa il compilatore...)

Nota. Si userà lo stack nelle procedure recorsive (e si chiarirà il suo uso grazie a QtSpim)

$sp ed $fp

Frame di stack (oppure di chiamata a procedura) è un blocco di memoria associato alla procedura

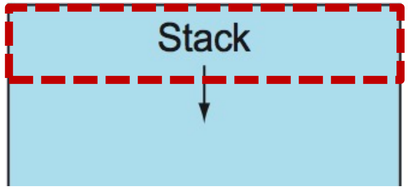
- **$sp – punta alla prima parola del frame**
- **$fp – punta all'ultima parola del frame**

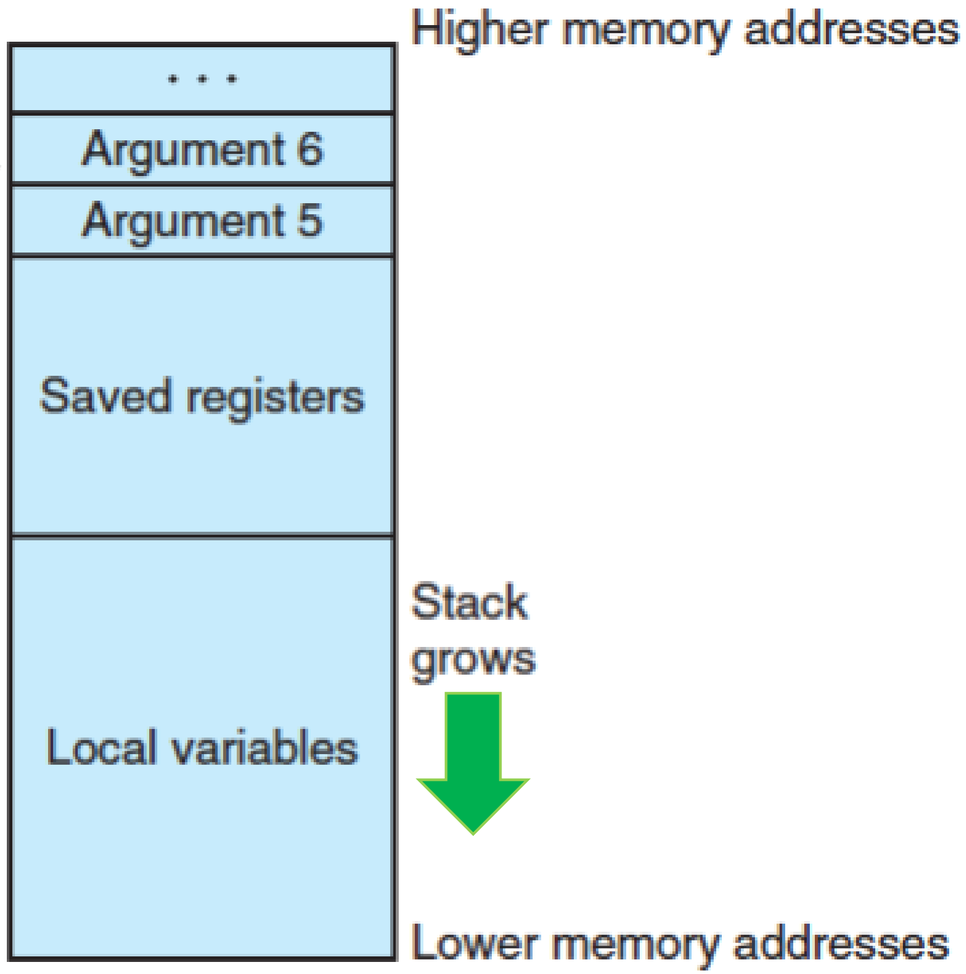
Esempio: un frame di 32 byte

```
addi $sp, $sp. -32      # frame di stack di 32 byte
addi $fp, $sp, 28       # imposta il frame pointer
sw $ra, 0($fp)          # salva l'indirizzo di ritorno come primo
                        # word nel frame sullo stack
```

Nota. Un frame di solito è multiplo della parola doppia (8 byte).

CONVENZIONI SULL'USO DELLA MEMORIA DURANTE PROCEDURA





STACK: CHIAMATA A PROCEDURA

Vuol dire interrompere il programma in esecuzione (il flusso *sequenziale* delle istruzioni) e *saltare* in un altro punto del codice, ***ricordandosi*** poi dove ci si era interrotti per poterci *tornare*.

Attenzione. Lo stiamo guardando dal basso!

```
addi $sp, $sp, -4      #faccio spazio per salvare $ra
                       #spostando $sp verso il basso
```

STACK: CHIAMATA A PROCEDURA

Vuol dire interrompere il programma in esecuzione (il flusso *sequenziale* delle istruzioni) e *saltare* in un altro punto del codice, ***ricordandosi*** poi dove ci si era interrotti per poterci *tornare*.

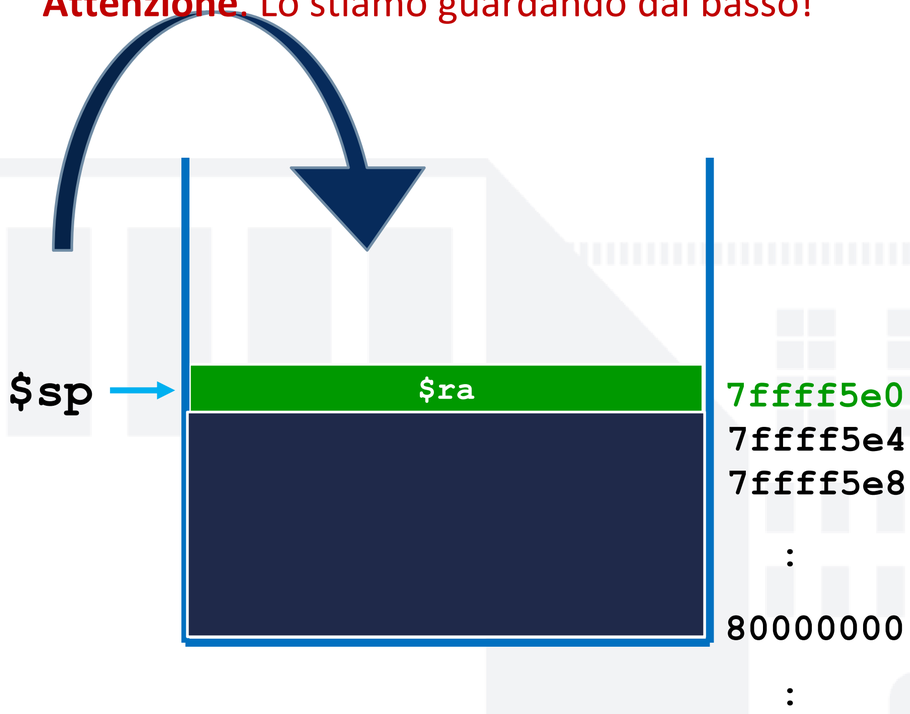
Attenzione. Lo stiamo guardando dal basso!

```
addi $sp, $sp, -4      #faccio spazio per salvare $ra
                       #spostando $sp verso il basso
```

STACK: CHIAMATA A PROCEDURA

Vuol dire interrompere il programma in esecuzione (il flusso *sequenziale* delle istruzioni) e *saltare* in un altro punto del codice, ***ricordandosi*** poi dove ci si era interrotti per poterci *tornare*.

Attenzione. Lo stiamo guardando dal basso!




```
addi $sp, $sp, -4      #faccio spazio per salvare $ra
                       #spostando $sp verso il basso
sw $ra, 0($sp)         #salvo $ra nello spazio allocato
                       #prima della jal
```

PC → istruzione successiva a jal

```
jal procedura_figlia
```

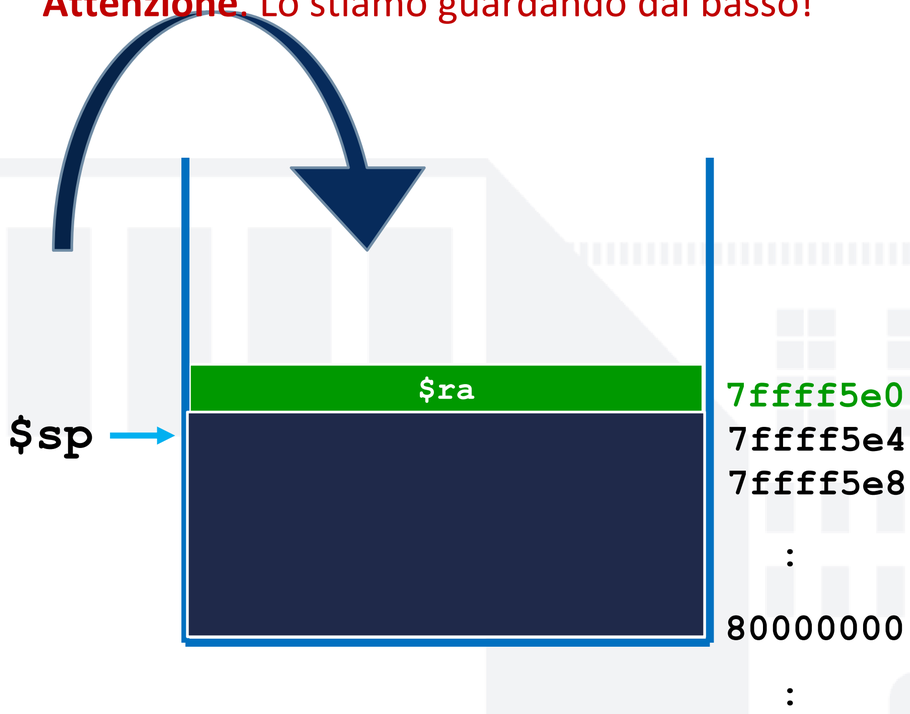
PC → indirizzo prima istruzione di procedura_figlia

$ra → precedente valore di PC

STACK: CHIAMATA A PROCEDURA

Vuol dire interrompere il programma in esecuzione (il flusso *sequenziale* delle istruzioni) e *saltare* in un altro punto del codice, ***ricordandosi*** poi dove ci si era interrotti per poterci *tornare*.

Attenzione. Lo stiamo guardando dal basso!




```
addi $sp, $sp, -4      #faccio spazio per salvare $ra
                       #spostando $sp verso il basso
sw $ra, 0($sp)         #salvo $ra nello spazio allocato
                       #prima della jal
```

PC → istruzione successiva a jal

```
jal procedura_figlia
```

PC → indirizzo prima istruzione di procedura_figlia

$ra → precedente valore di PC

STACK: RITORNO DA PROCEDURA

Vuol dire ritornare al programma «chiamante» (con istruzione di salto), ***ripristinando le condizioni ($sp, registri)*** che c'erano prima della chiamata. Se la procedura ha prodotto dei risultati, allora metterli in $v0, $v1 (convenzione).

Attenzione. Lo stiamo guardando dal basso!


```
lw $ra, 0($sp)   #recupera valore di $ra dallo stack
addi $sp, $sp, 4 #libera spazio allocato nello stack


jr $ra           #salta a istruzione il cui indirizzo
                 #è in $ra
```

PROCEDURE ANNIDATE: ESEMPIO#1

```
# main chiama procedura1 che chiama procedura2
# Data segment
.data
msg1: .asciiz "Esecuzione Main...\n"
msg2: .asciiz "Esecuzione Proc1...\n"
msg3: .asciiz "Esecuzione Proc2...\n"
msg4: .asciiz "Fine Proc2.\n"
msg5: .asciiz "Fine Proc1.\n"
msg6: .asciiz "Fine Main.\n"
```

```
# main e chiamata a procedura1
.text
main:                   #Codice del main
       li $v0, 4        #Stampa stringa
       la $a0, msg1     #Parametro della procedura di stampa (msg1)
       syscall          #anche main e' una procedura chiamata, che al termine dovra' fare jr $ra al chiamante

       addi $sp, $sp, -4    #faccio spazio per salvare $ra, spostando $sp verso il basso
       sw $ra, 0($sp)       #salvo $ra prima di jal nello spazio allocato
       jal procedura1

RIENTRODAPROCEDURA1:        #etichetta del punto di rientro da procedura1
       li $v0, 4            #stampa stringa
       la $a0, msg6         #parametro della procedura di stampa (msg6)
       syscall

       lw $ra, 0($sp)       #ripristino $ra prima di jr $ra
       addi $sp, $sp, 4     #riposiziono puntatore a stack pointer
       jr $ra               #salta a istruzione il cui indirizzo e' in registro $ra
```

Prima di chiamare una procedura

EFFETTO DELLA jump-and-link:

- scrivere in PC l'indirizzo dove si trova la prima istruzione della procedura chiamata
- salvare in $ra l'indirizzo dell'istruzione di codice che seguiva la jal

```
# chiusura programma main
li $v0, 10              #Exit
syscall
```

PROCEDURE ANNIDATE: ESEMPIO#1

Prima di chiamare una procedura

```
# procedura1 e chiamata a procedura2
procedura1:                     # Codice della procedura procedura1
          li $v0, 4             # Stampa msg2
          la $a0, msg2
          syscall
          addi $sp, $sp, -4     # Faccio spazio nello stack (spostando $sp verso il basso)
          sw $ra, 0($sp)        # e ci salvo $ra, cioe' salvo il punto di rientro chiamante
          jal procedura2        # Salto a proc2 e in $ra etichetta RIENTRODAPROCEDURA2

RIENTRODAPROCEDURA2:
          li $v0, 4             # Stampa msg5
          la $a0, msg5
          syscall

          lw $ra, 0($sp)        # Recupera il valore corretto di $ra dallo stack
          addi $sp, $sp, 4      # libera spazio allocato nello stack
          jr $ra                # salta a istruzione cui indirizzo e' in $ra
```

```
# procedura2
procedura2:                     # Codice della procedura procedura2 (foglia)
          li $v0, 4             # Stampa msg3
          la $a0, msg3
          syscall

          li $v0, 4             # Stampa msg4
          la $a0, msg4
          syscall

          jr $ra                # Restituisce il controllo al chiamante
```

Avvisi

- Tutorato extra LUNEDI’ 16/03 dalle 11 alle 12 → rispondere al sondaggio su Moodle

Materiale per la lezione

- *Hennessy-Patterson, cap. 1 pp.14-16*
- *Appendix A.2, A.3, A.4, A.5*

Prossima lezione: 17 ***marzo, h.14:00, aula 4C***