UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Procedure, linker e Assembly

Prof.ssa Giulia Cisotto

giulia.cisotto@units.it

Trieste, 2 aprile 2025

RECAP

Nomi riservati

Nome Simbolico	Numero	Uso
$zero	0	Costante 0
$at	1	Assembler temporary
$v0-$v1	2-3	Functions and expressions evaluation
$a0-$a3	4-7	Arguments
$t0-$t7	8-15	Temporaries
$s0-$s7	16-23	Saved Temporaries
$t8-$t9	24-25	Temporaries
$k0-$k1	26-27	Reserved for OS kernel
$gp	28	Global pointer
$sp	29	Stack pointer
$fp	30	Frame pointer
$ra	31	Return address

LINKER

Il **linker** è uno strumento che prende **più file oggetto** (generati da più file .asm) e:

1. **Unisce** il codice e i dati in un *unico programma eseguibile*.
2. **Risolve i riferimenti tra simboli** (es. se un file usa un'etichetta *func* definita in un altro).
3. **Assegna indirizzi finali** a tutte le etichette, dati e istruzioni, aggiornando la memoria.

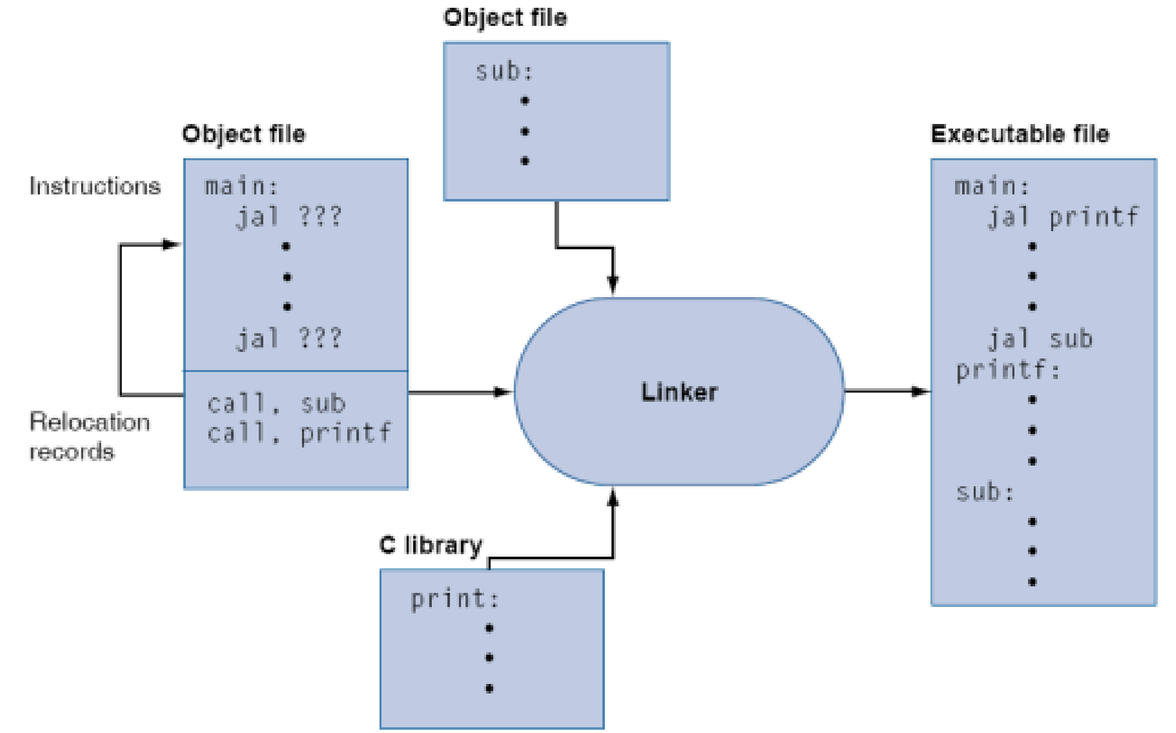


FIGURE A.3.1 The linker searches a collection of object files and program libraries to find nonlocal routines used in a program, combines them into a single executable file, and resolves references between routines in different files.

SYS-CALL: chiamata al sistema (operativo)

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $a0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

FIGURE A.9.1 System services.

UNIVERSITÀ DEGLI STUDI DI TRIESTE

ESITO QUIZ AUTOVALUTAZIONE

UNIVERSITÀ DEGLI STUDI DI TRIESTE

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
Comparazione tra procedure e spie: le spie lavorano in segreto, acquisiscono risorse, svolgono il compito, nascondono le tracce, e tornano al punto di partenza con i risultati richiesti

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)

REGISTRI TEMPORANEI SALVATI E NON SALVATI

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

- Convenzioni: registri $s e $t

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

PROBLEMI APERTI

Cosa succede se una procedura ne chiama un'altra?

- Si perde il contenuto di $ra della prima chiamata?
- Procedure recursive?
- -> uso dello stack

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

- Convenzioni: registri $s e $t

Dove stanno le variabili locali della procedura?

- Stack frame ...

PROCEDURE INNESTATE

Procedure “foglia” e “non foglia”

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

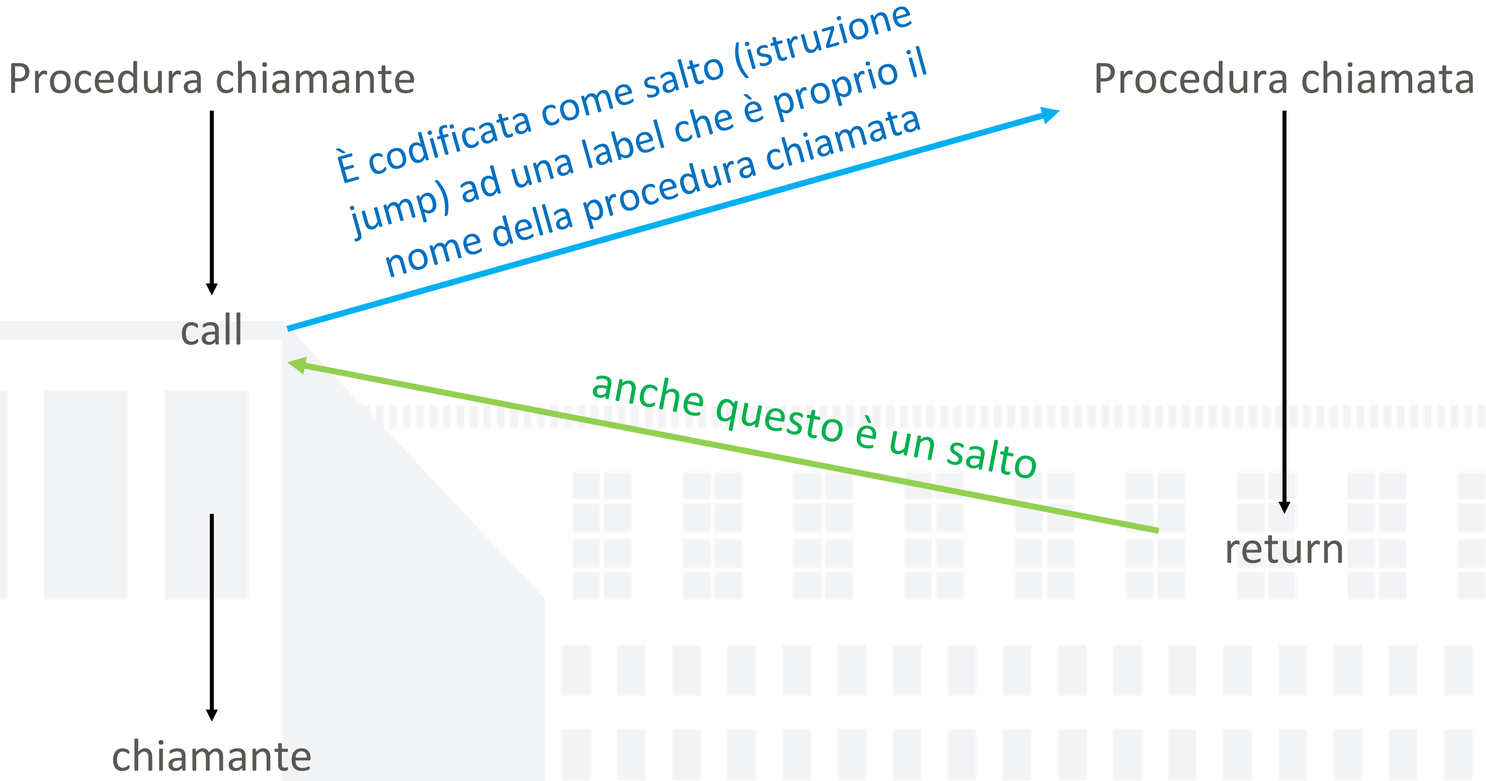
Cosa succede se una procedura ne chiama un’altra?

- Si perde il contenuto di $ra della prima chiamata??
- Procedure recursive??
- 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

CHIAMATA A FUNZIONI



UNIVERSITÀ DEGLI STUDI DI TRIESTE

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 parola 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

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 che possono essere scritte senza bisogno di sapere come è fatto il programma che le chiama
- Possono essere chiamate senza bisogno di sapere come sono fatte “dentro”

Passaggio parametri “per valore” o “per indirizzo”

- un parametro può essere un dato o un indirizzo!!!
- Confrontare le istruzioni la e lw
- 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 (1/2)

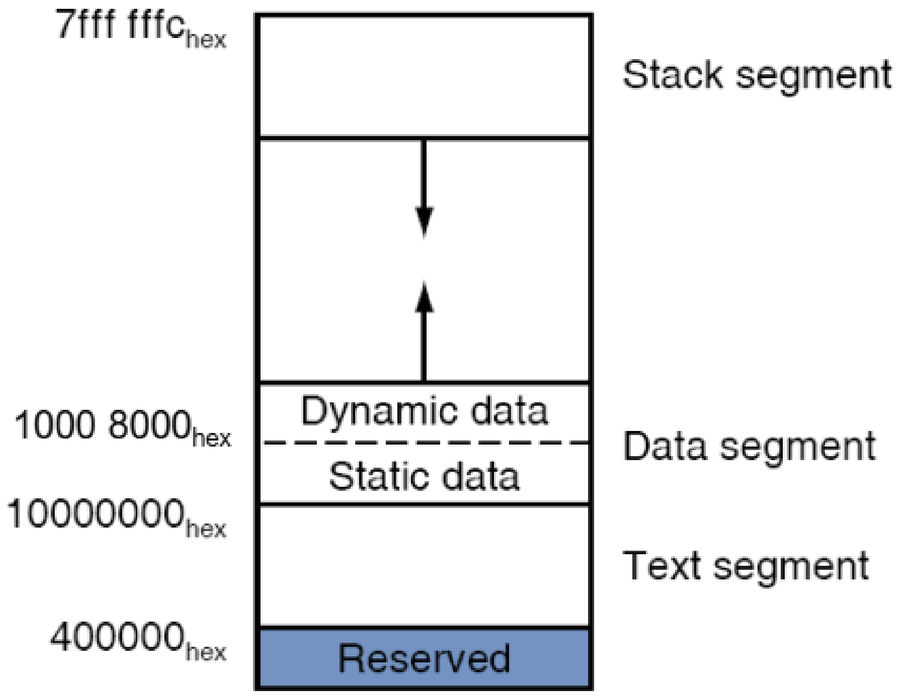
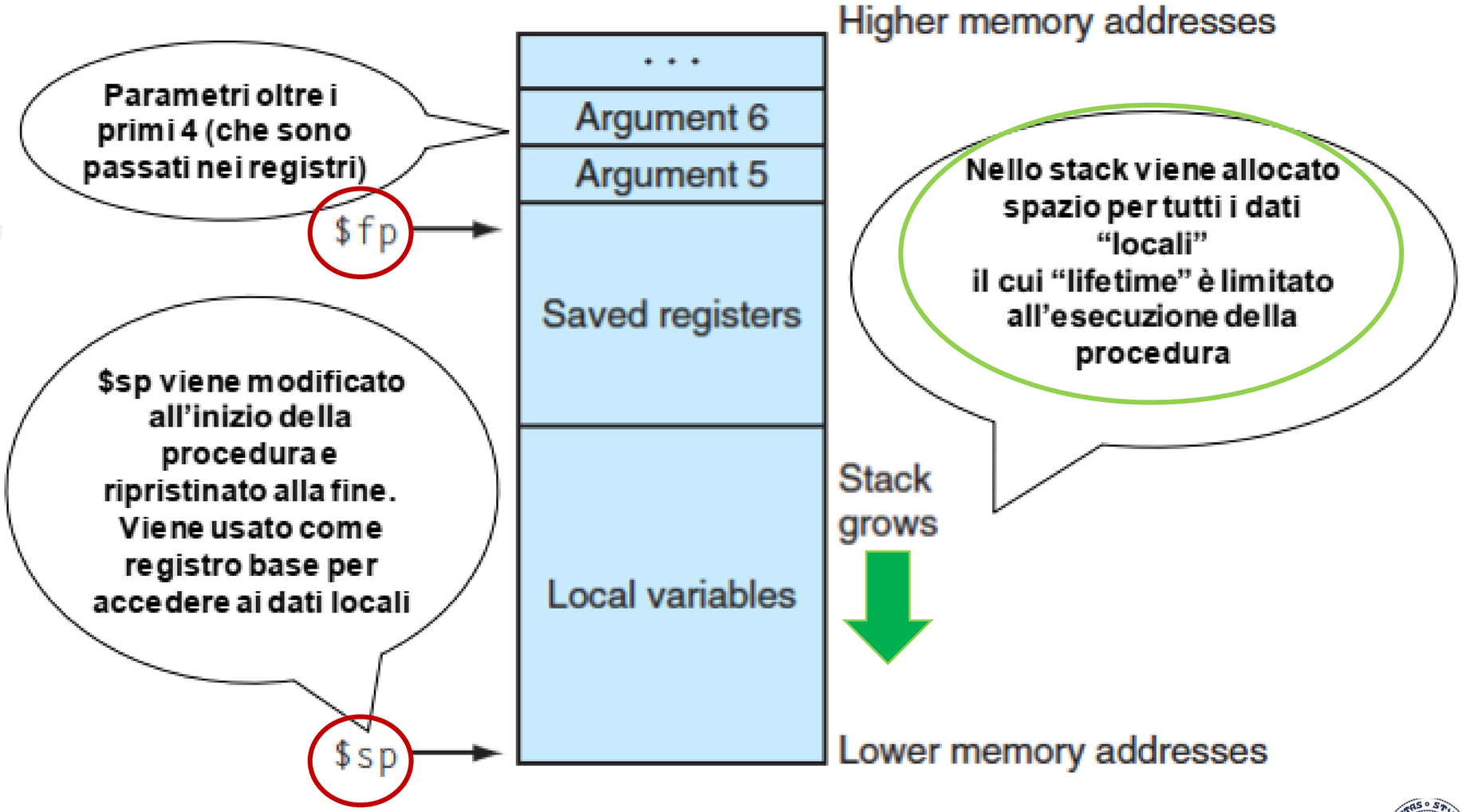


FIGURE A.5.1 Layout of memory.

CONVENZIONI SULL'USO DELLA MEMORIA (2/2)



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**

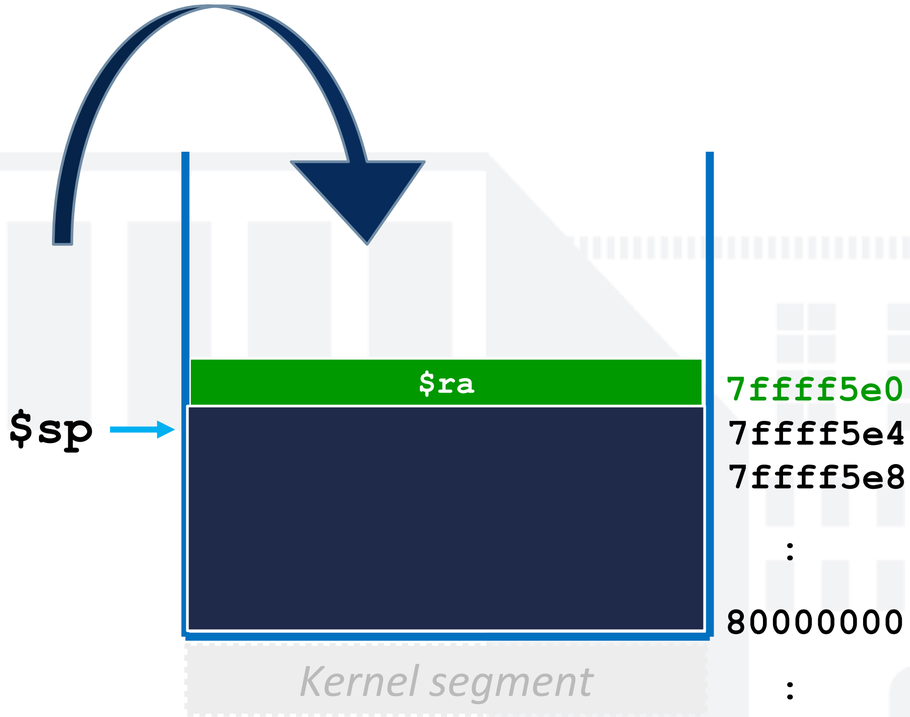
Un frame di solito è multiplo della parola doppia (8 byte).

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
```

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*.




```
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

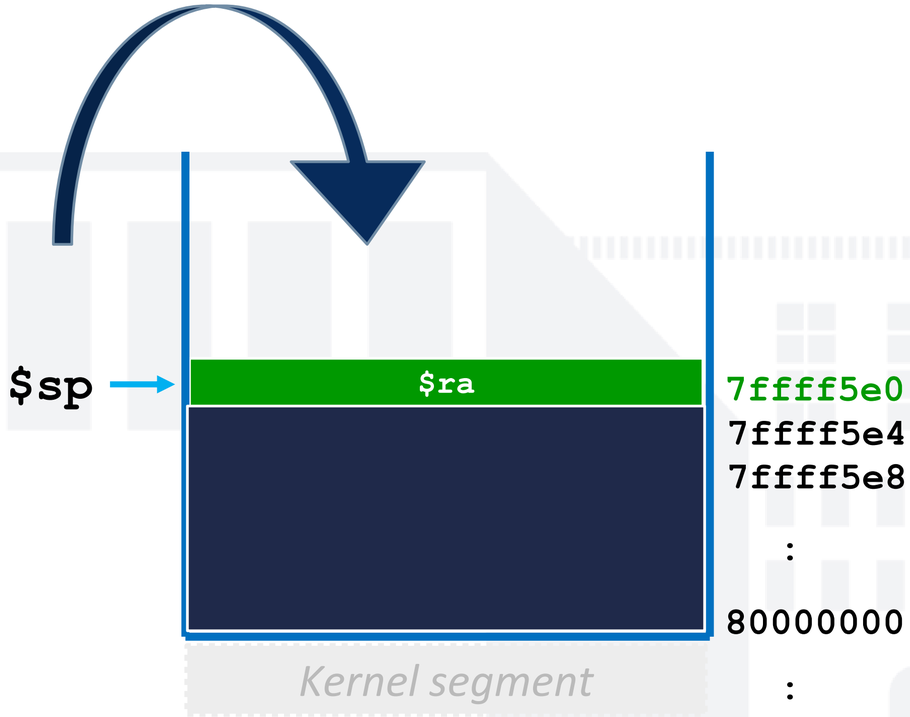
```
jal procedura_figlia
```

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*.




```
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

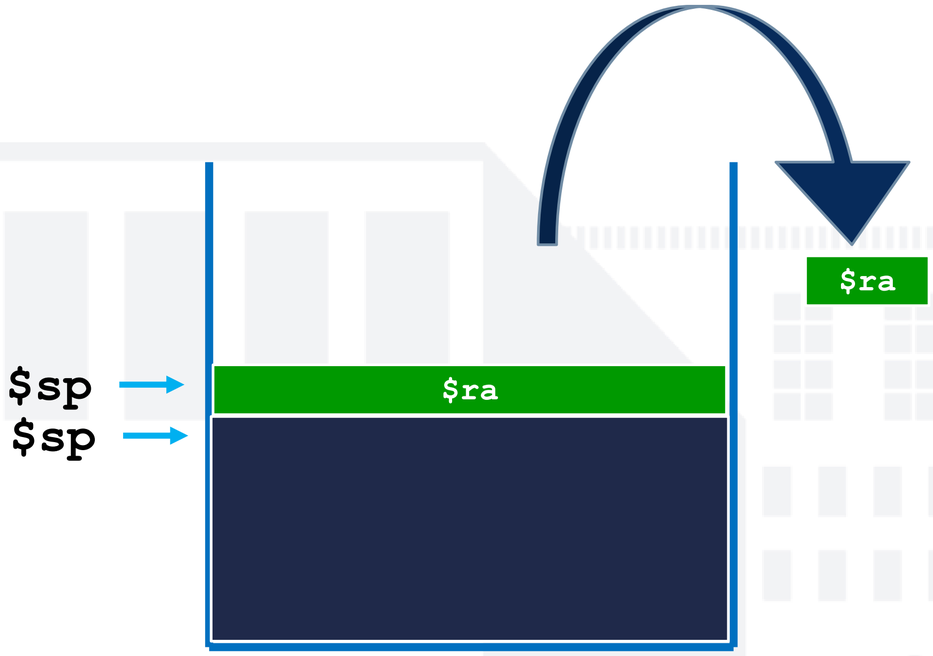
```
jal procedura_figlia
```

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).




```
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

su singolo file sorgente

```
# 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

su singolo file sorgente

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
```

PROGRAMMARE IN ASSEMBLY

Ricorsione: calcolo del fattoriale

fattoriale(5) = 5*4*3*2*1

```
.data
n:      .word 3          # Valore di partenza (n = 3)
res:    .word 0          # Spazio per il risultato

.text
.globl main

main:
    lw   $a0, n          # Carica n in $a0 (argomento per fact)
    jal  fact            # Chiama la funzione ricorsiva fact(n)
    sw   $v0, res        # Salva il risultato in memoria
    li   $v0, 10         # Termina il programma
    syscall

# Fattoriale ricorsivo: fact(n)
# se n <= 1 → ritorna 1
# altrimenti → ritorna n * fact(n - 1)
```

```
fact:
    addi $sp, $sp, -8          # Alloca spazio per $ra e $a0
    sw   $ra, 4($sp)           # Salva $ra sullo stack
    sw   $a0, 0($sp)           # Salva n ($a0) sullo stack

    li   $t0, 1                # $t0 = 1
    ble  $a0, $t0, base_case   # Se n <= 1, salta al caso base

    addi $a0, $a0, -1          # $a0 = n - 1
    jal  fact                  # Chiama ricorsivamente fact(n - 1)

    lw   $a0, 0($sp)           # Ripristina n originale
    mul  $v0, $a0, $v0         # $v0 = n * fact(n - 1)

    lw   $ra, 4($sp)           # Ripristina $ra
    addi $sp, $sp, 8           # Libera spazio nello stack
    jr   $ra                   # Ritorna

base_case:
    li   $v0, 1                # Caso base: fact(0) = 1
    lw   $ra, 4($sp)           # Ripristina $ra
    addi $sp, $sp, 8           # Libera spazio
    jr   $ra                   # Ritorna
```

Lo **stack ($sp)** viene usato per:

- salvare il valore attuale di $a0 prima della chiamata ricorsiva
- salvare e ripristinare $ra per tornare al punto giusto

Materiale per la lezione

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

Prossima lezione: 3 ***aprile, h.9:00, aula 4C***