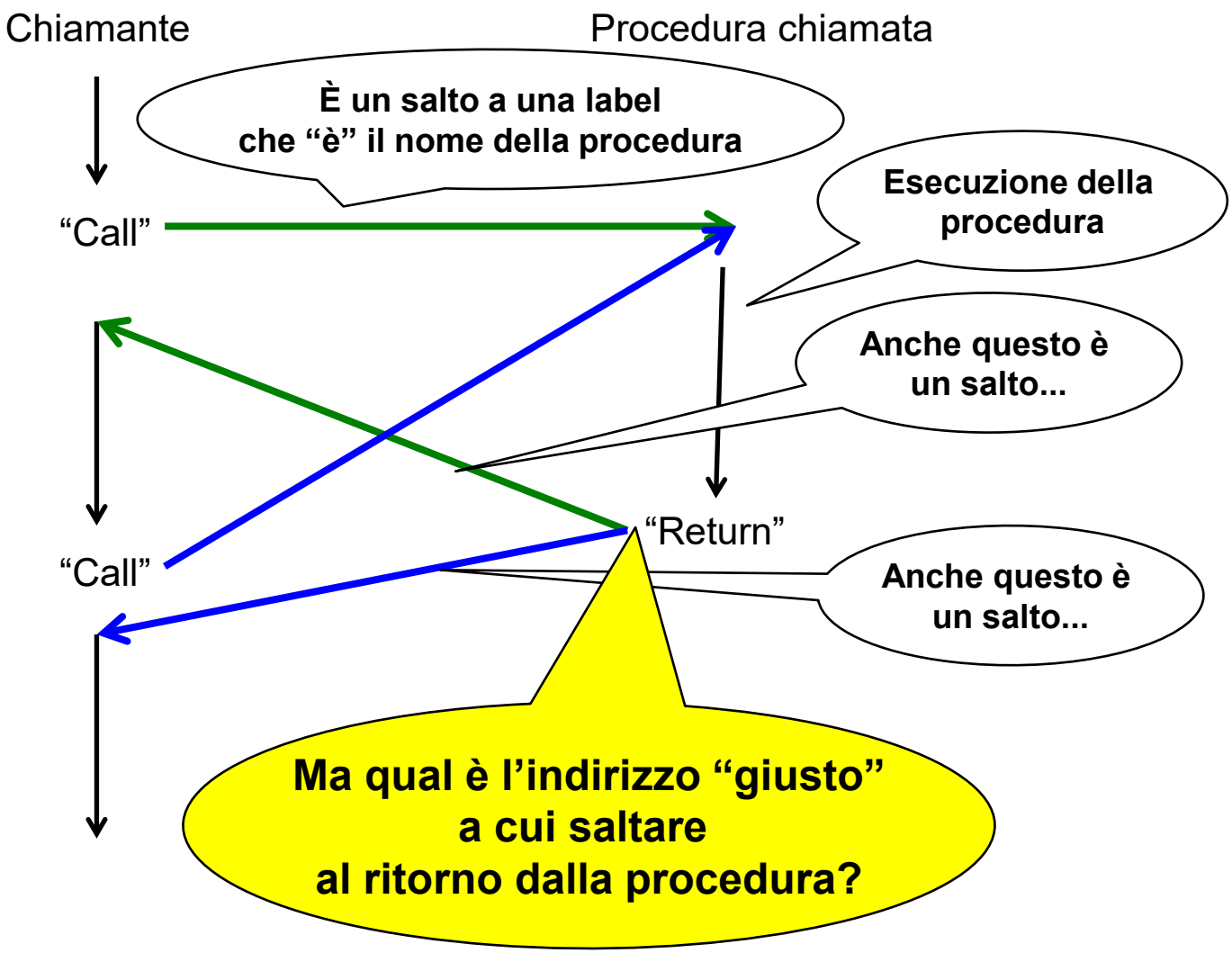
Corso di Laurea in Informatica
Architettura degli elaboratori
a.a. 2025-2026

Programmazione Assembly: procedure

Passaggio parametri attraverso registri
Syscall

Claudia Raibulet
claudia.raibulet@unimib.it

Flusso di controllo



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

Passaggio parametri (convenzioni base)

- **$a0 - $a3: registri argomento per il passaggio dei parametri**
- **$v0 - $v1: registri valore per la restituzione dei risultati**
- **NB: 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"
- **NB: un parametro può essere un dato o un indirizzo!!!**
  - Rivedere passaggio parametri "per valore" o "per indirizzo" dall'insegnamento di Programmazione 1
  - Confrontare le istruzioni la e lw

Osservazione

- **Una procedura – 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
  - 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

Riferimenti principali

- **Patterson – Hennessy, Computer Organization and Design, Morgan Kaufmann**
  - Capitolo 2, Sezione 8
- **Appendice A**

Esempio elementare (1): dati

Il programma definisce quattro numeri num1, num2, num3 e num4.
Definisce una procedura “somma” che riceve due numeri come parametri e restituisce la loro somma.
Chiama due volte la procedura passando come parametri prima num1 e num2 (memorizzando il risultato in result1), poi num3 e num4 (memorizzando il risultato in result2)

```
.data
num1: .word 50
num2: .word 14
result1: .word 0
num3: .word 50
num4: .word -66
result2: .word 0
```

Esempio elementare (2): programma principale

```
.text
.globl main
main:

#prima chiamata della procedura
la $t0, num1
lw $a0, 0($t0)  #predisposizione del primo parametro
lw $a1, 4($t0) #predisposizione del secondo parametro

jal somma
#l'indirizzo dell'istruzione successiva viene salvato in $ra e si
#salta alla procedura.

sw $v0, 8(4t0) #memorizzazione del primo risultato

#seconda chiamata della procedura
lw $a0, 12($t0)
lw $a1, 16($t0)
jal somma
sw $v0, 20($t0)
```

Esempio elementare (3): la procedura

```
#Procedura
somma: #questo è l'indirizzo iniziale della procedura

add $v0, $a0, $a1
# convenzione:
# I registri $a0-$a3 si usano per passare i parametri
# I registri $v0 e $v1 si usano per restituire i risultati

jr $ra
# il registro $ra contiene l'indirizzo di ritorno
```

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 ...
- **Di tutto questo parleremo più avanti...**
- **...ma cominciate a porvi i problemi**

Sistema Operativo e Syscall

- **In un sistema reale esiste il Sistema Operativo...**
- **...che è un insieme di programmi che:**
  - stanno in un'area (protetta) di memoria
  - svolgono funzioni di utilità generale (in particolare, I/O) richiamabili dai programmi utente
- **La struttura generale e le funzioni di un SO saranno trattate nell'insegnamento di Reti e Sistemi Operativi (II anno)**
- **I meccanismi base di chiamata al SO sono trattati nel seguito di questo insegnamento**
- **Il simulatore MIPS fornisce alcune funzioni elementari che simulano alcune funzionalità base del SO...**
- **...richiamabili attraverso il meccanismo di syscall, concettualmente analogo a una chiamata a procedura**

Syscall

- **Analogo a una chiamata a procedura**
- **Convenzioni per le syscall: Tabella a pag. A43 (Appendice A)**
- **Impostare nel registro $v0 il codice della chiamata**
- **Impostare i parametri nei registri $a0-$a3 (come da tabella)**
- **Syscall**
- **Syscall essenziali:**
  - exit2 codice 17: terminazione del programma!
  - read_int
  - print_int codice 1 (parametro passato per valore!!)
  - read_string (guardare e capire bene i parametri!!!)
  - print_string (parametro passato per indirizzo!!)
  - ....

Syscall

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.

Syscall (esempio)

```
# il codice seguente stampa la stringa «La risposta è 5 »:
            .data
str:                    .asciiz “La risposta è “
numero:                 .word 5
.globl main
            .text
main:
            li $v0, 4       # Codice della chiamata di sistema per print_str
            la $a0, str  # Indirizzo della stringa da stampare (passato per indirizzo!!!)
            syscall       # Stampa la stringa

            li $v0, 1           # Codice della chiamata di sistema per print_int
            la $t0, numero
            lw $a0, 0(4t0) # Numero intero da stampare (passato per valore!!!)
            syscall           # Stampa l’intero
```

Esercizio

- **Si chiede di calcolare la lunghezza di una stringa memorizzata partendo dall'indirizzo "stringa". La stringa finisce con il carattere '\0'. Si chiede di memorizzare la dimensione della stringa in memoria dopo la stringa in un word.**
- **Si chiede di scrivere la stringa e la sua dimensione sullo schermo tramite le syscall.**

Esercizio – Soluzione senza procedure

```
            .data
stringa:    .asciiz "Hello World!"
#13 byte per memorizzare la stringa
#12 byte lunghezza della stringa
dim:        .word 0
charfine:   .asciiz ""
            .text
            .globl main
main:
            la $t0, stringa
            add $t2, $zero, $zero
            la $s5, charfine
            lb $s1, 0($s5)
ciclo:
            lb $s7, 0($t0)
            beq $s7, $s1, fine
            addi $t2, $t2, 1
            addi $t0, $t0, 1
            j ciclo
fine:
            la $t3, dim
            sw $t2, 0($t3)

            # scrive la stringa sullo schermo
            li $v0, 4
            la $a0, stringa
            syscall

            # scrive la dim della
            # stringa sullo schermo
            li $v0, 1
            la $t0, dim
            lw $a0, 0($t0)
            syscall
```

Esercizio – Soluzione con procedure

```
            .data
stringa:    .asciiz "Hello World!"
dim:        .word 0
charfine:   .asciiz ""
            .text
            .globl main
main:
            la $a0, stringa
            jal calcola_dim_stringa
            # la dimensione della stringa
            # e' in $v0
            # salva la dim in memoria
            la $t7, dim
            sw $v0, 0($t7)
            # scrive la stringa sullo schermo
            li $v0, 4
            la $a0, stringa
            syscall

            # scrive la dim della
            # stringa sullo schermo
            li $v0, 1
            la $t0, dim
            lw $a0, 0($t0)
            syscall
```

```
.globl calcola_dim_stringa

calcola_dim_stringa:
            add $t2, $zero, $zero
            la $t5, charfine
            lb $t1, 0($t5)

ciclo:      lb $t7, 0($a0)
            beq $t7, $t1, fine
            addi $t2, $t2, 1  # incrementa la dim
            addi $a0, $a0, 1 # incrementa l'indirizzo
            # per il prossimo elemento della stringa
            j ciclo
fine:       move $v0, $t2
            jr $ra
```