

## Modalità di Indirizzamento + Programmi

### Architettura degli Elaboratori e delle Reti Turno I



Alberto Borghese  
Università degli Studi di Milano  
Dipartimento di Scienze dell'Informazione  
email: borghese@dsi.unimi.it

1

## Modalità di indirizzamento

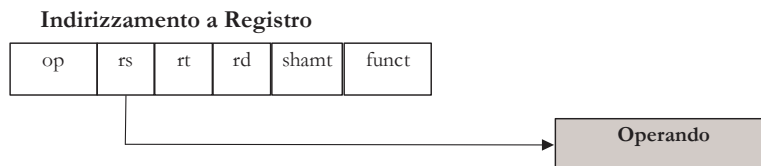
- Le modalità di indirizzamento indicano le diverse modalità attraverso le quali far riferimento agli operandi nelle istruzioni.
- L'esempio più comune di modalità di indirizzamento è l'indirizzamento **a registro** nel quale gli operandi dell'istruzione sono contenuti nei registri: ad esempio `add $s0, $s1, $s2`.

## Modalità di indirizzamento

- MIPS ha solo 5 modalità di indirizzamento:
  - ◆ A registro
  - ◆ Immediato
  - ◆ Con base o spiazzamento
  - ◆ Relativo al Program Counter
  - ◆ Pseudo-diretto
- Una singola istruzione può usare più di una modalità di indirizzamento.

## Indirizzamento a registro

- L'operando (l'indirizzo) è il contenuto di un registro della CPU: il nome (numero = indirizzo) del registro è specificato nell'istruzione.



## Indirizzamento a registro

- Le istruzioni che usano questo tipo di indirizzamento hanno formato di tipo R.
- Esempio: istruzione aritmetico-logica:

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
<code>add \$s1, \$s2, \$s3</code>	000000	10010	10011	10001	00000	100000

## Indirizzamento immediato

- L'operando è una costante il cui valore è contenuto nell'istruzione.
- L'indirizzamento immediato si usa per specificare il valore di un operando sorgente, non ha senso usarlo come destinazione.

### Indirizzamento Immediato



- Le istruzioni che usano questo tipo di indirizzamento hanno formato I
  - ◆ La costante è memorizzata nel campo a 16-bit

## Indirizzamento immediato

- Esempio: operazione aritmetico-logica con operando immediato (formato tipo I):

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
<code>addi \$s1, \$s1, 4</code>	001000	10001	10001	0000 0000 0000 0100

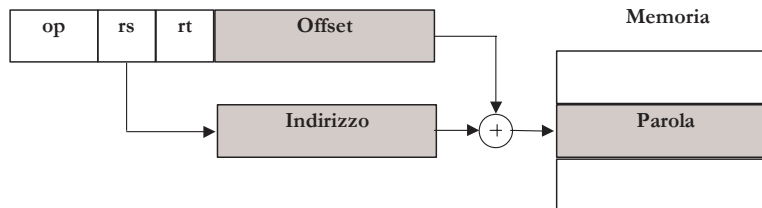
- Esempio: operazione di confronto con operando immediato (formato tipo I):

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
<code>slti \$t0, \$s2, 8</code>	001010	10010	01000	0000 0000 0000 1000

## Indirizzamento con base

- L'operando è in una locazione di memoria il cui indirizzo si ottiene sommando il contenuto di un registro base ad un valore costante (*offset o spiazzamento*) contenuto nell'istruzione.

Indirizzamento con base



- Le istruzioni che usano questo tipo di indirizzamento hanno formato di tipo I.

## Indirizzamento con base

- Esempio: istruzione di load

```
lw $t0, 32 ($s3)
```

- ◆ L'operando di trova in memoria all'indirizzo 32+[\$s3]

- Esempio: istruzione di store

```
sw $t0, 32 ($s3)
```

- ◆ L'operando viene copiato in memoria all'indirizzo 32+[\$s3]

L'indirizzo e' espresso in numero di byte.

## Indirizzamento con base

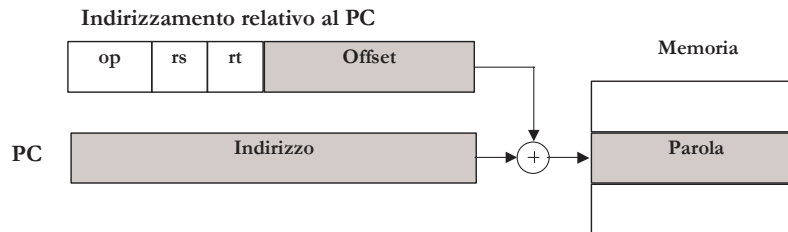
- Esempio: Istruzioni di load/store (formato tipo I):

Nome campo	op	rs	rt	indirizzo			
Dimensione	6-bit	5-bit	5-bit	16-bit			
lw \$t0, 32 (\$s3)	10011	10011	01000	0000	0000	0010	0000

Nome campo	op	rs	rt	indirizzo			
Dimensione	6-bit	5-bit	5-bit	16-bit			
sw \$t0, 32 (\$s3)	101011	10011	01000	0000	0000	0010	0000

## Indirizzamento relativo al PC

- L'**istruzione** è in una locazione di memoria il cui indirizzo si ottiene sommando il contenuto del *Program Counter* ad un valore costante (*offset o spiazzamento*) contenuto nell'istruzione:



- Le istruzioni che usano questo tipo di indirizzamento hanno formato di tipo I.

## Indirizzamento relativo al PC

- Esempio: Operazione di salto condizionato (formato tipo I):

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
<code>beq \$s1, \$s2, 100</code>	000100	10001	10010	0000 0000 0001 1001

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
<code>bne \$s1, \$s2, 100</code>	000101	10001	10010	0000 0000 0001 1001

## Indirizzamento relativo al PC

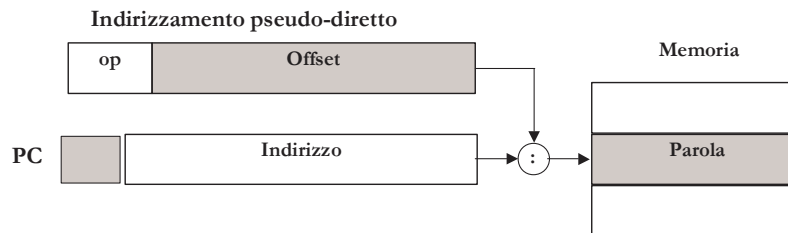
- Si usa l'indirizzamento relativo al PC nei salti condizionati in quanto la destinazione del salto in tali istruzioni è in genere prossima al punto di salto.
- Avendo a disposizione 16 bit di *Offset*  $\Rightarrow$  è possibile saltare in un'area tra  $-2^{15}$  e  $+2^{15}-1$  parole rispetto all'istruzione corrente.

## Indirizzamento pseudo-diretto

- Una parte dell'indirizzo è presente come valore costante (offset) nell'istruzione ma deve essere completato.
- Le istruzioni che usano questo tipo di indirizzamento hanno formato di tipo J.

## Indirizzamento pseudo-diretto

- L'indirizzo di salto si calcola facendo uno shift a sinistra di 2 bit dei 26-bit di offset contenuti nell'istruzione (aggiungendo 00 nei bit meno significativi per passare da 26 a 28-bit) e concatenando i 28-bit con i 4-bit più significativi del Program Counter.



## Indirizzamento pseudo-diretto

- Esempio: operazione di salto incondizionato (formato J)

Nome campo	op	indirizzo						
Dimensione	6-bit	26-bit						
j 32	000010	00	0000	0000	0000	0000	0000	1000



## Utilizzo di costanti

- Spesso le operazioni richiedono l'uso di costanti (ad esempio: somma del valore decimale 4 al contenuto di un registro).
- Possibili **3** opzioni:
  - ◆ le costanti risiedono in memoria e sono caricate con `lw`
  - ◆ utilizzo di registri speciali (es: `$zero`)
  - ◆ utilizzare versioni alternative delle **istruzioni di tipo I** in cui un operando è una **costante**
    - > utilizzo di modalità di **indirizzamento immediato**
- La costante è memorizzata nel campo di **16 bit** denominato **indirizzo**.

## Esempi

```
addi $s0, $s0, 4    # $s0 ← $s0 + 4 (sign-extended)
slti $t0, $s2, 10   # $t0 = 1 if $s2 < 10
andi $s0, $s0, 6    # $s0 ← $s0 and 6 (zero-extended)
ori $s0, $s0, 10    # $s0 ← $s0 or 10 (zero-extended)
li $s0, 20          # $s0 ← 20 (pseudo-instruction)
```

- Le istruzioni di tipo I consentono di rappresentare costanti esprimibili in 16 bit.
- I valori immediati sono espressi in assembly in rappresentazione decimale ma possono anche essere esadecimali o binari

## Esempio di indirizzamento a registri & immediato

```
# Somma

.text          #Definizione segmento codice
.globl main    #Definizione del main

main:
    li $t1,10  # carica il valore decimale 10 nel reg. $t1
    li $t2,15  # carica il valore decimale 15 nel reg. $t2
    add $a0,$t2,$t1 # $a0 ← $t2 + $t1
print_result:
    li $v0,1   # stampa risultato (= 25)
    syscall
```

## Esempio di indirizzamento a registri & immediato

```
# L'accesso immediato è usato anche dalle operazioni
# aritmetiche

.text          #Definizione segmento codice
.globl main

main:
    li $t1,10  # $t1 ← 10
    addi $a0,$t1,15 # $a0 ← $t1 + 15
# stampa risultato
print_result:
    li $v0,1
    syscall
```

## Esempio di indirizzamento a registri & immediato

```
# I valori immediati possono essere dichiarati come
# esadecimali: 10 e 15 espressi in esadecimale con
# 0xA e 0xF.

        .text
        .globl main
main:
        li $t1,0xA
        addi $a0,$t1,0xF
# stampa risultato
print_result:
        li $v0,1
        syscall
```

## Gestione di costanti su 32-bit

- Le istruzioni di **tipo I** consentono di rappresentare costanti esprimibili in 16 bit (valore massimo 65535 unsigned).
- Se 16 bit non sono sufficienti per rappresentare la costante, l'assemblatore (o il compilatore) deve fare due passi:
  - ◆ si utilizza l'istruzione **lui (load upper immediate)** per caricare i 16 bit più significativi della costante nei 16-bit più significativi di un registro. I rimanenti 16-bit meno significativi del registro sono posti a 0.
  - ◆ una successiva istruzione, ad esempio, di **or** specifica i rimanenti 16 bit meno significativi della costante.
- Il registro **\$at** è riservato all'assemblatore per creare costanti su 32-bit (costanti 'lunghe').

## Istruzione lui: formato tipo I

- L'istruzione **load upper immediate**: `lui rt, imm` carica i 16-bit del campo immediato nei 16-bit più significativi del registro `rt`. I rimanenti 16-bit meno significativi del registro `rt` sono posti a 0.

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
<code>lui \$at, 61</code>	001111	00000	00001	0000 0000 0011 1101

- Pseudo-istruzione: `li rdest, imm` carica il valore `imm` nel registro `rdest`.

## Caricamento costante di 32 bit Esempio 1

Si consideri la costante a 32 bit:

0000 0000 0011 1101 0000 1001 0000 0000

`lui $s0, 61 (lui $zero,$s0,61) #61 = 0000 0000 0011 1101`

valore di `$s0`:

0000 0000 0011 1101 0000 0000 0000 0000

`addiu $s0, $s0, 2304 # 2304 = 0000 1001 0000 0000`

valore di `$s0`:

0000 0000 0011 1101 0000 1001 0000 0000

400,000 in decimale

## Caricamento costante di 32 bit Esempio 2

Si consideri la costante a 32-bit:  $118345_{10}$  ( $0x1CE49$ )

```
0000 0000 0000 0001 1100 1110 0100 1001
```

16-bit più significativi      16-bit meno significativi  
corrispondenti al valore  $1_{10}$       corrispondenti al valore  $52809_{10}$

$$1 \times 2^{16} + 52809 = 118345$$

Si consideri l'istruzione:

```
li $t1, 118345                      # $t1 ← 118345
```

## Pseudo-istruzione li: esempio

L'assemblatore sostituisce l'istruzione originale con le seguenti istruzioni:

```
lui $at, 1                      # 1 = 0000 0000 0000 0001
```

valore di \$at:

```
0000 0000 0000 0001 0000 0000 0000 0000
```

```
ori $t1, $at, 52809      # $t1 ← $at or 52809
```

valore di \$t1:

```
0000 0000 0000 0001 1100 1110 0100 1001
```

## Verso i programmi in SPIM

## Il simulatore SPIM

▪ **SPIM: A MIPS R2000/R3000 Simulator :**  
**PCSPIM version 6.3**

▪ <http://www.cs.wisc.edu/~larus/spim.html>

▪ **Piattaforme:**

- Unix or Linux system
- Microsoft Windows  
(Windows 95, 98, NT, 2000)
- Microsoft DOS

**Esempio:**

```
int main(int argc, char* argv[])  
{ int i,n,t = 0; for (i=0;i<n;i++) t +=i*i; return(0); }
```

```

# Programma che somma i quadrati dei primi N numeri
# N e' memorizzato in t1.

.data
.align 2
str:
    .asciiz "La soma da 0 a N vale "
.text
.globl main

main:
    li    $t1, 2    # N interi di cui calcolare la somma dei
                   # quadrati
    li    $t6, 0    # t6 e' indice di ciclo indice di ciclo
    li    $t8, 0    # t8 contiene la somma dei quadrati
Loop:
    mul   $t7, $t6, $t6    # t7 = t6 x t6
    addu  $t8, $t8, $t7    # t8 = t8 + t7
    addi  $t6, $t6, 1      # t6 = t6 + 1
    ble   $t6, $t1, Loop  # if t6 < N stay in loop

    la    $a0, str
    li    $v0, 4          #print
    syscall

    li    $v0, 1          #print
    add   $a0, $t8, $zero
    syscall

    li    $v0, 10         # $v0 codice della exit
    syscall               # esce dal programma

```

**Codice assembly**

## Sommario

- Introduzione a SPIM
- **Direttive**
- Chiamate di sistema (system call)
- Chiamate a procedura

## Direttive

- Le direttive (data layout directives) danno delle indicazioni all'assemblatore sul contenuto di un file (istruzioni, strutture dati, ecc.)
- Sintatticamente le direttive iniziano tutte con il carattere "."

## Direttive

- `.data <addr>`
  - Gli elementi successivi sono memorizzati nel segmento dati a partire dall'indirizzo `addr`, facoltativo.
- `.ascii str`
  - Memorizza la stringa `str` terminandola con il carattere `Null` (`.ascii str` ha lo stesso effetto, ma non aggiunge alla fine il carattere `Null`)
- `.byte b1,...,bn`
  - Memorizza gli `n` valori `b1, ..., bn` in byte consecutivi di memoria



## Direttive

`.word w1, ..,wn`

- Memorizza gli  $n$  valori su 32-bit  $w1, \dots, wn$  in parole consecutive di memoria.

`.half h1, ..,hn`

- Memorizza gli  $n$  valori su 16-bit  $h1, \dots, hn$  in halfword (mezze parole) consecutive di memoria

`.space n`

Alloca uno spazio pari ad  $n$  byte nel segmento dati

### *SEGMENTO TESTO*

`.text <addr>`

- Memorizza gli elementi successivi nel segmento testo dell'utente a partire dall'indirizzo `addr`. (Questi elementi possono essere solo istruzioni o parole).

## Direttive

`.globl sym`

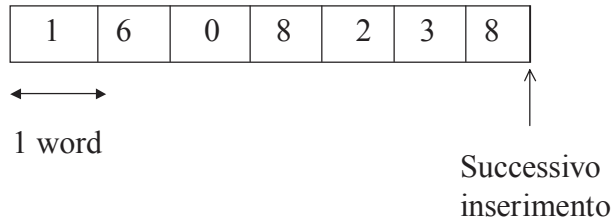
- Dichiarare `sym` come etichetta globale (ad essa è possibile fare riferimento da altri file)

`.align n`

- Allinea il dato successivo a blocchi di  $2^n$  byte: ad esempio
  - ◆ `.align 2 = .word` allinea alla parola il valore successivo
  - ◆ `.align 1 = .half` allinea alla mezza parola il valore successivo
  - ◆ `.align 0` elimina l'allineamento automatico delle direttive `.half, .word, .float, e .double` fino a quando compare la successiva direttiva `.data`

## **.word: esempio**

`.word 1, 6, 0, 8, 2, 3, 8`

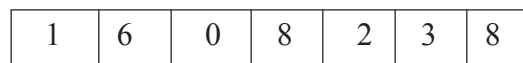


@ N. A. Borghese – Università degli Studi di Milano 02/04/2003

35/48

## **.word + etichetta: esempio**

`array: .word 1, 6, 0, 8, 2, 3, 8`



←→  
1 word

*⇒ array rappresenta l'indirizzo del primo elemento*

@ N. A. Borghese – Università degli Studi di Milano 02/04/2003

36/48

```

# Programma che somma i quadrati dei primi N numeri
# N e' memorizzato in t1.

.data
.align 2
str:
    .asciiz "La soma da 0 a N vale "
.text
.globl main

main:
    li    $t1, 2    # N interi di cui calcolare la somma dei
                  # quadrati
    li    $t6, 0    # t6 e' indice di ciclo indice di ciclo
    li    $t8, 0    # t8 contiene la somma dei quadrati
Loop:
    mul   $t7, $t6 $t6    # t7 = t6 x t6
    addu  $t8, $t8, $t7    # t9 = t8 + t7
    addi  $t6, $t6, 1
    ble   $t6, $t1, Loop  # if t6 < N stay in loop

    la    $a0, str
    li    $v0, 4          #print
    syscall

    li    $v0, 1          #print
    add   $a0, $t8, $zero
    syscall

    li    $v0, 10         # $v0 codice della exit
    syscall               # esce dal programma

```

**Codice assembly**

## Sommario

- Introduzione a SPIM
- Direttive
- Chiamate di sistema (system call)
- Chiamate a procedura

## System call

- Sono disponibili delle **chiamate di sistema (system call)** predefinite che implementano particolari servizi (ad esempio: stampa a video)
- Ogni system call ha:
  - ◆ un codice
  - ◆ degli argomenti (opzionali)
  - ◆ dei valori di ritorno (opzionali)

## System call

- **print\_int**: stampa sulla console il numero intero che le viene passato come argomento;
- **print\_float**: stampa sulla console il numero in virgola mobile con singola precisione che le viene passato come argomento;
- **print\_double**: stampa sulla console il numero in virgola mobile con doppia precisione che le viene passato come argomento;
- **print\_string**: stampa sulla console la stringa che le è stata passata come argomento terminandola con il carattere **Null**;

## System call

- **read\_int**: legge una linea in ingresso fino al carattere a capo incluso (i caratteri che seguono il numero sono ignorati);
- **read\_float**: leggono una linea in ingresso fino al carattere a capo incluso (i caratteri che seguono il numero sono ignorati);
- **read\_double**: leggono una linea in ingresso fino al carattere a capo incluso (i caratteri che seguono il numero sono ignorati);
- **read\_string**: legge una stringa di caratteri di lunghezza \$a1 da una linea in ingresso scrivendoli in un buffer (\$a0) e terminando la stringa con il carattere `Null` (se ci sono meno caratteri sulla linea corrente, li legge fino al carattere a capo incluso e termina la stringa con il carattere `Null`);

## System call

- **sbrk** restituisce il puntatore (indirizzo) ad un blocco di memoria;
- **exit** interrompe l'esecuzione di un programma;

## System call

Nome	Codice (\$v0)	Argomenti	Risultato
print_int	1	\$a0	
print_float	2	\$f12	
print_double	3	\$f12	
print_string	4	\$a0	
read_int	5		\$v0
read_float	6		\$f0
read_double	7		\$f0
read_string	8	\$a0,\$a1	
sbrk	9	\$a0	\$v0
exit	10		

## System call

- Per richiedere un servizio ad una chiamata di sistema (`syscall`) occorre:
  - ◆ Caricare il **codice** della `syscall` nel registro `$v0`
  - ◆ Caricare gli **argomenti** nei registri `$a0 - $a3` (oppure nei registri `$f12 - $f15` nel caso di valori in virgola mobile)
  - ◆ Eseguire `syscall`
  - ◆ L'eventuale **valore di ritorno** è caricato nel registro `$v0` (`$f0`)

## Esempio

```
#Programma che stampa: la risposta è 5
.data
str: .asciiz "la risposta è "
.text
.globl main

Main:
li $v0, 4          # $v0 ← codice della print_string
la $a0, str        # $a0 ← indirizzo della stringa
syscall           # stampa della stringa

li $v0, 1          # $v0 ← codice della print_integer
li $a0, 5          # $a0 ← intero da stampare
syscall           # stampa dell'intero

li $v0, 10         # $v0 ← codice della exit
syscall           # esce dal programma
```

## Esempio

```
#Programma che stampa "Dammi un intero: "
# e che legge un intero
.data
prompt:.asciiz "Dammi un intero: "
.text
.globl main

main:
li $v0, 4          # $v0 ← codice della print_string
la $a0, prompt     # $a0 ← indirizzo della stringa
syscall           # stampa la stringa

li $v0, 5          # $v0 ← codice della read_int
syscall           # legge un intero e lo carica in $v0

li $v0, 10         # $v0 ← codice della exit
syscall           # esce dal programma
```

## Esempio, fattoriale

```
# Programma che calcola il fattoriale iterativamente
.data
prompt: .ascii "Inserisci un numero intero"
output: .ascii "Il fattoriale è:"
.text
.globl main
main:
    li $v0, 4          # $v0 ← codice della print_string
    la $a0, prompt     # $a0 ← indirizzo della stringa
    syscall           # stampa la stringa

    li $v0, 5          # $v0 ← codice della read_int
    syscall           # legge l'intero e lo carica in $v0
```

## Esempio, fattoriale (calcolo)

```
# calcola il fattoriale

    li $t0, 1          # inizializzo $t0 contatore cicli
    li $t1, 1          # inizializzo $t1 accumulatore
                        # che conterrà il risultato n!

loop:
    mul $t1, $t1, $t0 # $t1 ← $t1 * $t0
    addi $t0, $t0, 1  # incremento $t0 contatore cicli
    ble $t0, $v0, loop # se $t0 ≤ $v0 go to loop
```



## Esempio, fattoriale (stampa risultato)

```
# stampa il risultato

    move $a0, $v0      # $a0 ← $v0
    li $v0, 1         # $v0 ← codice della print_int
    syscall           # stampa l'intero in input

    li $v0, 4         # $v0 ← codice della print_string
    la $a0, output    # $a0 ← indirizzo della stringa
    syscall           # stampa la stringa

    li $v0, 1         # $v0 ← codice della print_int
    move $a0, $t1     # $a0 ← n!
    syscall           # stampa n!

    li $v0, 10        # $v0 ← codice della exit
    syscall           # esce dal programma
```