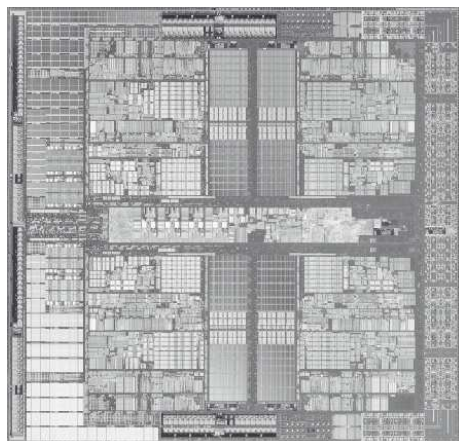




Università degli Studi di Cassino e del Lazio Meridionale



Corso di Calcolatori Elettronici

*Indirizzamento
Pseudo istruzioni*

Anno Accademico 2013/2014

Alessandra Scotto di Freca

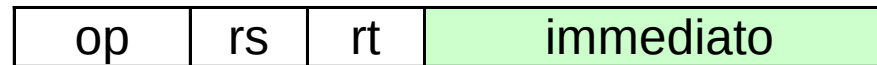
Si ringrazia il prof. Francesco Tortorella per il materiale didattico

Modi di indirizzamento

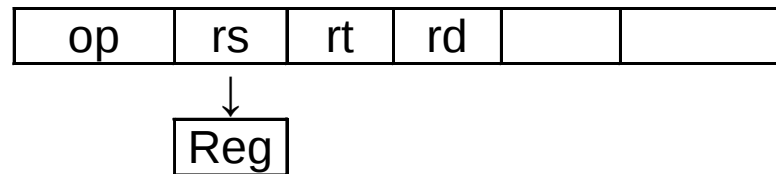
- Indicano come accedere all'operando di interesse dell'istruzione. Il MIPS ha 5 modi di indirizzamento:
- **immediato** (immediate)
 sli \$1,\$2,100
- **registro** (register)
 sli \$1,\$2,100
- **base/spiazzamento** (base/displacement)
 lw \$t0,4(\$t1)
- **relativo rispetto al PC** (PC-relative)
 beq \$1,\$2,label
- **pseudodiretto** (pseudo direct)
 j label

Modi di indirizzamento

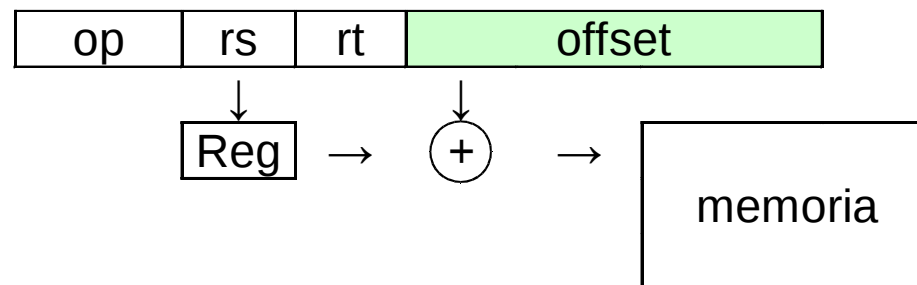
immediato



registro



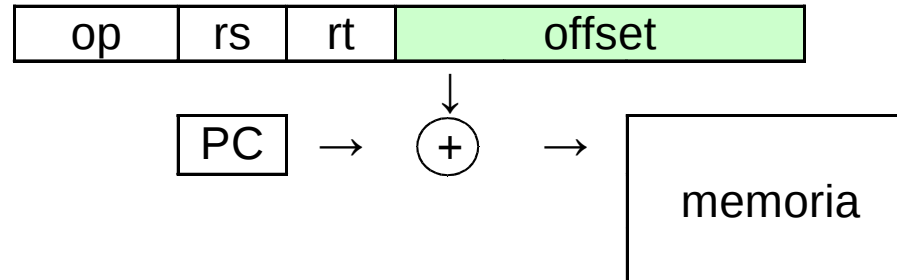
base (+offset)



Modi di indirizzamento

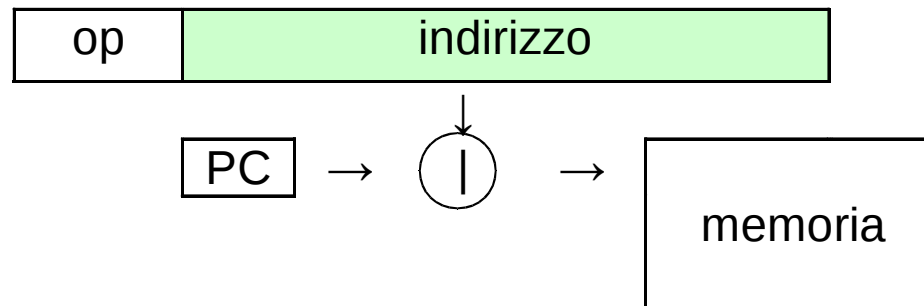
PC-relative (*branch*)

± 128 KB da PC + 4



PC-relative (*jump*)

entro la pagina di 256
MB a partire da PC + 4



Pseudoistruzioni

Sono istruzioni accettate dall'assemblatore MIPS alle quali non corrisponde un effettivo codice operativo in linguaggio macchina.

Prima della traduzione del programma in linguaggio macchina, le pseudo istruzioni vengono espanse dall'assemblatore in sequenze di istruzioni ammesse, usando il registro \$1 (\$at), riservato a questo scopo.

Una stessa pseudo istruzione può essere espansa in modi diversi, a seconda degli operandi contenuti.

Es.:

li \$s0,0x1234 → ori \$s0,\$0,0x1234

li \$s0,-1 → lui \$1, -1
ori \$s0,\$1,-1

li \$s0,0x12345678 → lui \$1,0x1234
ori \$s0,\$1,0x5678

Pseudoistruzioni aritmetiche

<i>Pseudoistr.</i>	<i>Significato</i>	<i>Esempio</i>	
abs <i>rd,rs</i>	$rd = \text{ABS}(rs)$	abs \$1,\$2	
div <i>rd,rs,src</i>	$rd = rs \div src$	div \$1,\$2,100	<i>eccezione possibile</i>
divu <i>rd,rs,src</i>	$rd = rs \div src$	divu \$1,\$2,\$3	<i>nessuna eccezione</i>
mul <i>rd,rs,src</i>	$rd = rs \times src$	mul \$1,\$2,100	<i>eccezione possibile</i>
mulo <i>rd,rs,src</i>	$rd = rs \times src$	mulo \$1,\$2,\$3	<i>eccezione possibile</i>
mulou <i>rd,rs,src</i>	$rd = rs \times src$	mulou \$1,\$2,\$3	<i>unsigned</i> <i>eccezione possibile</i>
rem <i>rd,rs,src</i>	$rd = rs \bmod src$	rem \$1,\$2,100	<i>signed</i>
remu <i>rd,rs,src</i>	$rd = rs \bmod src$	remu \$1,\$2,100	<i>unsigned</i>

Altre pseudoistruzioni

Confronto

seq *rd,rs1,rs2*
sge *rd,rs1,rs2*
sgeu *rd,rs1,rs2*
sgt *rd,rs1,rs2*
sgtu *rd,rs1,rs2*
sle *rd,rs1,rs2*
sleu *rd,rs1,rs2*

Controllo

b *label*
beqz *rs,label*
bge *rs1,rs2,label*
bgeu *rs1,rs2,label*
bgt *rs1,rs2,label*
bgtu *rs1,rs2,label*
ble *rs1,rs2,label*
bleu *rs1,rs2,label*
blt *rs1,rs2,label*
bltu *rs1,rs2,label*
bnez *rs,label*

Load/Store

la *rd,address*
li *rd,constant*
ld *rd,address*
ulh *rd,address*
ulhu *rd,address*
ulw *rd,address*
sd *rs,address*
ush *rs,address*
usw *rs,address*

Move

move *rd,rs*

Altri modi di indirizzamento

L'assembler MIPS permette anche altri modi di indirizzamento oltre quelli direttamente implementati in hardware (*pseudo-indirizzamento*).
Vengono realizzati con i modi disponibili in una o più istruzioni.

diretto

lw \$t0,vett → lui \$1,vett_{HIGH}
lw \$8,vett_{LOW}(\$1)

registro indiretto

lw \$t0,(\$t1) → lw \$8,0(\$9)

base+registro indiretto

lw \$t1, vett+4(\$t0) → lui \$1, vett_{HIGH}
addu \$1, \$1, \$8
lw \$9, {4+vett_{LOW} }(\$1)

Direttive

Forniscono all'assemblatore istruzioni relative all'assemblaggio del programma. La loro interpretazione **non genera codice**, ma provoca lo svolgimento di particolari azioni da parte dell'assemblatore.

Principali direttive

<code>.align n</code>	<code>.half h1,...,hn</code>
<code>.ascii str</code>	<code>.kdata <addr></code>
<code>.asciiz str</code>	<code>.ktext <addr></code>
<code>.byte b1,...,bn</code>	<code>.set noat</code>
<code>.data <addr></code>	<code>.set at</code>
<code>.double d1,...,dn</code>	<code>.space n</code>
<code>.extern sym</code>	<code>.text <addr></code>
<code>.float f1,...,fn</code>	<code>.word w1,...,wn</code>
<code>.globl sym</code>	

.align n

Allinea il dato successivo a blocchi di 2^n byte.

Es.: .align 2 allinea alla word il dato successivo

 .align 0 elimina l'allineamento automatico

.ascii str

.asciiz str

Mette in memoria la stringa str (non) terminata dal carattere null

.byte b1, ..., bn

.half h1,...,hn

.word w1,...,wn

Memorizza n byte (halfword,word) in parole consecutive della memoria

.space n

Alloca n byte a partire dall'indirizzo corrente.

.data <addr>

.kdata <addr>

Gli elementi successivi sono memorizzati nel segmento dati utente (kernel)

.text <addr>

.ktext <addr>

Gli elementi successivi sono memorizzati nel segmento testo utente (kernel)

Definizione costanti

Costanti Numeriche

12 decimale

0x2F esadecimale

Costanti Carattere

Delimitate da doppi apici. Generano la sequenza di byte corrispondenti ai codici ASCII dei relativi caratteri

mesg: .asciiz "ciao"

mesg: .byte \$43,\$49,\$41,\$4F,0

Chiamate di sistema (syscall)

- Sono un insieme di servizi di sistema messi a disposizione dall'ambiente di simulazione (sia SPIM che MARS) invocabili con l'istruzione **syscall**. La maggior parte di esse è dedicata all'input/output.
- Uso delle chiamate di sistema:
 1. Si carica il codice della system call in \$v0
 2. Si caricano gli argomenti, se ci sono, in \$a0, \$a1, \$a2 o \$f12
 3. Si esegue l'istruzione **syscall**
 4. Si recuperano i risultati, se ci sono, nei registri specificati

Chiamate di sistema (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	

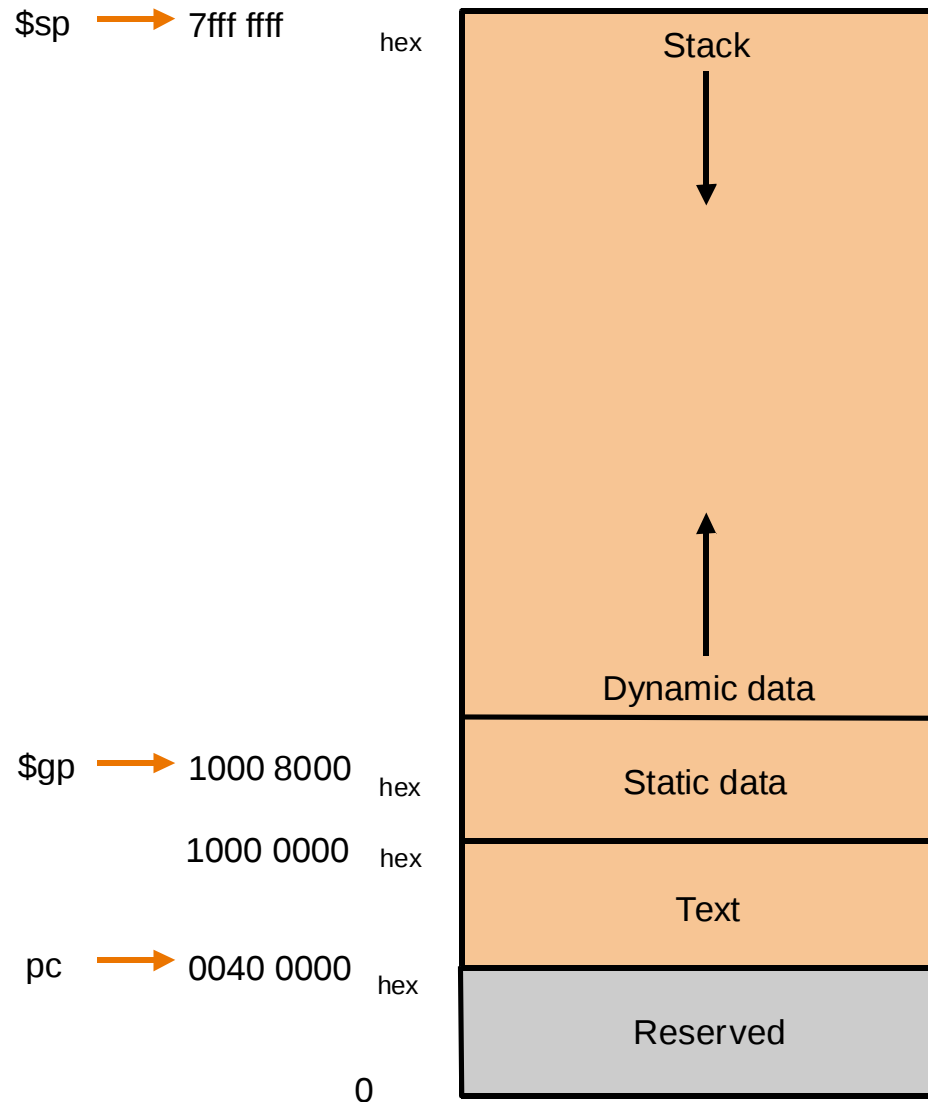
Esempio di chiamate di sistema

```
.data
str:      .ascii "the answer is "
        .align 2
W1: .word 5

.text
li $v0, 4      # system call code for print_str
la $a0, str     # address of string to print
syscall        # print the string

li $v0, 1      # system call code for print_int
lw $a0, W1     # integer to print
syscall        # print it
```

Allocazione di memoria del MIPS



\$sp → 7fff ffff hex

hex

Stack



Dynamic data

Static data

Text

Reserved

0

```
.data
W1: .word 10
W2: .word 20,30
buf: .space 12
```

```
.text
la $t0,buf
lw $t1,W1
sw $t1,($t0)
lw $t1,W2
sw $t1,4($t0)
la $t2,W2
lw $t1,4($t2)
sw $t1,8($t0)
```

```
li $v0,10
syscall
```

\$gp → 1000 8000 hex

hex

1000 0000 hex

hex

pc → 0040 0000 hex

hex