MIPS & SPIM

Modulo del Corso di Architettura degli Elaboratori

Nicola Paoletti

Università di Camerino
Scuola di Scienze e Tecnologie

24 Aprile 2013
AA 2012/2013

Lezioni precedenti

- **Lezione 1:** linguaggio macchina/assembly/alto livello; RISC vs CISC.
- **Lezione 2:** introduzione al MIPS Instruction Set; registri; istruzioni aritmetiche e di data transfer.

Riepilogo

Come traduciamo un'istruzione in linguaggio macchina?

Tre tipi di formati (32 bit):

- **Formato R-type:** per le istruzioni del tipo *add*, *sub*, *mul*, *div*, *and*, *or*, ... (R-format: Registry-format)
- **Formato I-type:** per le istruzioni del tipo *addi*, *lw*, *sw*, ... (I-format: Immediate-format)
- **Formato J-type:** per le istruzioni di jump (J-format: Jump-format)

Riepilogo

R-format

op	rs	rt	rd	shamt	funct
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

- **op:** opcode
- **rs:** primo registro sorgente
- **rt:** secondo registro sorgente
- **rd:** registro di destinazione
- **shamt:** shift amount; =0 se non si tratta di un'istruzione di shift
- **funct:** funzione; i campi *op* e *funct* identificano la particolare funzione da eseguire.

I-format (1/2)

op	**rs**	**rt**	**address**
6 bit	5 bit	5 bit	16 bit

- In questo caso solo *opcode* identifica l'istruzione mnemonica.
- Si hanno 16 bit per indirizzare la memoria fisica.

I-format (2/2)

Domanda:

Ricordandoci che la memoria è indirizzata byte per byte, con 16 bit per specificare l'indirizzo,
qual'è l'intervallo a cui possiamo accedere?

I-format (2/2)

Domanda:

Ricordandoci che la memoria è indirizzata byte per byte, con 16 bit per specificare l'indirizzo,
qual'è l'intervallo a cui possiamo accedere?

Dato un offset `$s`, dal byte `$s` al byte $2^{16} - 1 +$ `$s`.

J-format

op	**address**
6 bit	26 bit

Address specifica l'indirizzo dell'istruzione a cui saltare.

Una vista d'insieme

Type	**Format(Bits)**					
R	opcode(6)	rs(5)	rt(5)	rd(5)	shamt(5)	funct(6)
I	opcode(6)	rs(5)	rt(5)	immediate(16)		
J	opcode(6)	address(26)				

noop

noop

Alto livello: ; oppure { }
Assembly: `noop`
Macchina: `0x0`

Si utilizza per questioni di temporizzazione (far passare cicli di clock), per allineare la memoria e altre "manutenzioni"

Riepilogo

Addition

add

Alto livello: `$d = $s + $t`

Assembly: `add $d, $s, $t`

Macchina:

0x0	$s	$t	$d	0x0	0x20

Subtraction

sub

Alto livello: `$d = $s - $t`

Assembly: `sub $d, $s, $t`

Macchina:

0x0	$s	$t	$d	0x0	0x22

Multiplication

mult

Alto livello: `$LO = (($s * $t) << 32)>>32`
`$HI = ($s * $t)>>32`

Assembly: `mult $s, $t`

Macchina:

0x0	$s	$t	0x0	0x0	0x18

Moltiplica il contenuto dei due registri a 32 bit (`$s` e `$t`). Il risultato (64 bit) viene messo in due registri speciali:

- `$LO`: contiene i 32 bit meno significativi del risultato; si recupera attraverso l'istruzione `mflo $s`.
- `$HI`: contiene i 32 bit più significativi del risultato; si recupera attraverso l'istruzione `mfhi $s`.

Bitwise and

and

Alto livello: `$d = $s & $t`

Assembly: `and $d, $s, $t`

Macchina:

0x0	$s	$t	$d	0x0	0x24

Bitwise or

or

Alto livello: `$d = $s | $t`

Assembly: `or $d, $s, $t`

Macchina:

0x0	$s	$t	$d	0x0	0x25

Exclusive or

xor

Alto livello: `$d = $s ^ $t`

Assembly: `xor $d, $s, $t`

Macchina:

0x0	$s	$t	$d	0x0	0x26

Set on less than

slt

Alto livello: `$d=($s < $t)`

Assembly: `slt $d, $s, $t`

Macchina:

0x0	$s	$t	$d	0x0	0x2A

Riepilogo

Add immediate

addi

Alto livello: `$t = $s + C`

Assembly: `addi $t, $s, C`

Macchina:

0x8	$s	$t	C

Load word

lw

Alto livello: `$t = MEM[$s + C]`

Assembly: `lw $t, C($s)`

Macchina:

0x23	$s	$t	$s + C

Store word

SW

Alto livello: `MEM[$s + C] = $t`

Assembly: `sw $t, C($s)`

Macchina:

0x2B	$s	$t	$s + C

Branch on equal

beq

Alto livello: `if ($s == $t) goto C`

Assembly: `beq $s,$t, C`

Macchina:

0x4	$s	$t	C

Example

```
# se $s1 è uguale a $s2, li sottraggo; altrimenti li
sommo.

        beq $s1, $s2, L1
        add $t1, $s1, $s2
        ...
  L1:   sub $t1, $s1, $s2
```

Branch on not equal

bne

Alto livello: `if ($s != $t) goto C`

Assembly: `bne $s,$t, C`

Macchina:

0x5	$s	$t	C

Example

```
# se $s1 è diverso a $s2, li sommo; altrimenti li
sottraggo.

        bne $s1, $s2, L1
        sub $t1, $s1, $s2
        ...
  L1:   add $t1, $s1, $s2
```

Riepilogo

Unconditional jump

jump

Alto livello: `goto C`

Assembly: `j C`

Macchina:

0x2	C

Example

```
# goto L1

        j L1
        ...
  L1:   ...
```

Esercizi (1/2)

Massimo

Scrivere una procedura MIPS per il possibile codice dell'istruzione `max $v0, $a0, $a1`.

Esercizi (1/2)

Massimo

Scrivere una procedura MIPS per il possibile codice dell'istruzione `max $v0, $a0, $a1`.

```
        slt $t0, $a0, $a1
        beq $t0, $0, L1
        addi $v0, $a1, 0
        ...
  L1:   addi $v0, $a0, 0
```

Esercizi (2/2)

Minimo

Scrivere una procedura MIPS per il possibile codice dell'istruzione `min $v0, $a0, $a1`.

Esercizi (2/2)

Minimo

Scrivere una procedura MIPS per il possibile codice dell'istruzione `min $v0, $a0, $a1`.

```
        slt $t0, $a0, $a1
        beq $t0, $0, L1
        addi $v0, $a0, 0
        ...
  L1:   addi $v0, $a1, 0
```