Architettura degli Elaboratori

MIPS:

Istruzioni logiche e codifica in linguaggio macchina

Barbara Masucci
UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

Punto della situazione

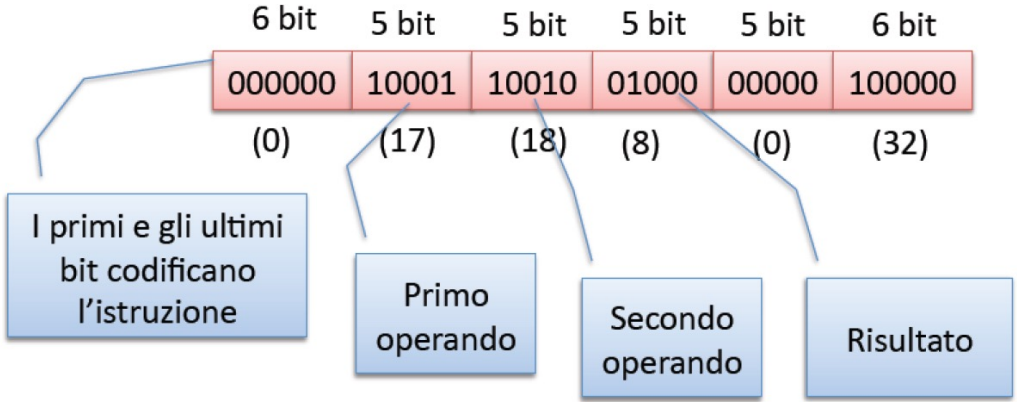
- Stiamo studiando l'Instruction Set del MIPS
- Abbiamo visto
  - Istruzioni aritmetiche: add, sub
  - Istruzioni di trasferimento dati: lw, sw
  - L'organizzazione dei registri e della memoria

- Obiettivo di oggi
  - Codifica delle istruzioni in linguaggio macchina
  - Studio delle istruzioni per effettuare operazioni logiche: AND, OR, NOR, XOR, shift logico e aritmetico

Codifica delle istruzioni

- Abbiamo studiato l'istruzione aritmetica in assembly MIPS add $t0, $s1, $s2

- Come viene convertita dall'assemblatore in linguaggio macchina?



$zero	0
$at	1
$v0 - $v1	2-3
$a0 - $a3	4-7
$t0 - $t7	8-15
$s0 - $s7	16-23
$t8 - $t9	24-25
$k0 - $k1	26-27
$gp	28
$sp	29
$fp	30
$ra	31

- Gli operandi corrispondono alla numerazione dei registri (fissa nel MIPS)
  - Nel secondo campo, $10001_2=17_{10}$ corrisponde al registro $s1

Codifica delle istruzioni

- Il formato binario a 32 bit usato per le istruzioni MIPS a tre operandi viene detto "formato R" (da Registro)

  - OP: codice operativo dell'istruzione
  - rs: primo operando sorgente
  - rt: secondo operando sorgente
  - rd: operando destinazione
  - shamt: shift amount (per alcune istruzioni)
  - funct: definisce la funzionalità specifica dell'istruzione (insieme ad OP)

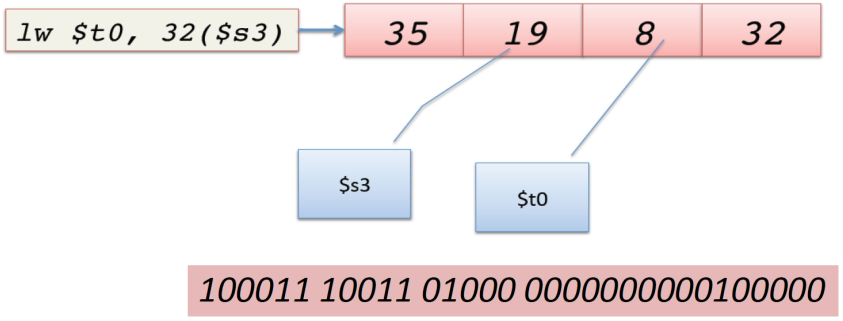
Codifica delle istruzioni

- Esiste anche un altro formato, detto "formato I", per l'indirizzamento Immediato (operazioni con costanti, operazioni lw e sw)

- Ad esempio, l'istruzione

  lw $t0, 32($s3)

  viene convertita secondo il formato I



Codifica delle istruzioni

- Sul libro sono riportati i formati e i codici operativi di tutte le istruzioni

Istruzione	Formato	op	rs	rt	rd	shamt	funct	indirizzo
add	R	0	reg	reg	reg	0	32_{dec}	n.a.
sub (sottrazione)	R	0	reg	reg	reg	0	34_{dec}	n.a.
addi (addizione immediata)	I	8_{dec}	reg	reg	n.a.	n.a.	n.a.	costante
lw (load word)	I	35_{dec}	reg	reg	n.a.	n.a.	n.a.	indirizzo
sw (store word)	I	43_{dec}	reg	reg	n.a.	n.a.	n.a.	indirizzo

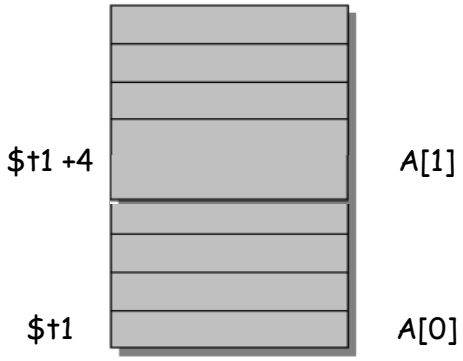
- Nota: il campo op di add e sub è lo stesso (000000), ma cambia il campo funct
  - funct=32_{10} per add
  - funct=34_{10} per sub

Codifica delle istruzioni

- Supponiamo di voler tradurre in codice macchina la seguente istruzione:

  A[300] = h + A[300]

- Inoltre, supponiamo che $t1 contenga l'indirizzo base del vettore A e che $s2 contenga h



- Il codice assembler MIPS corrispondente è

```
lw $t0, 1200($t1) #$t0 assume il valore A[300]
add $t0, $s2, $t0 #$t0 assume il valore h+A[300]
sw $t0, 1200($t1) #memorizza h+A[300] in A[300]
```

Codifica delle istruzioni

- Vediamo prima i codici decimali corrispondenti

Istruzione	op	rs	rt	rd	Ind/Shamt	Funct
lw $t0, 1200($t1)	35	9	8	1200		
add $t0, $s2, $t0	0	18	8	8	0	32
sw $t0, 1200($t1)	43	9	8	1200		

- E poi quelli in binario

op	rs	rt	rd	Ind/Shamt	Funct
100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Operazioni logiche bit a bit

- Il MIPS offre operazioni logiche che operano su ciascun bit
  - AND
  - OR
  - NOR
  - XOR
- Il formato è sempre a tre operandi (registri), dove il primo indica la destinazione
  - Quindi queste istruzioni sono in formato R

AND bit a bit

- Supponiamo che il contenuto dei due registri $t1 e $t2 sia il seguente

```
t1 = 0000 0000 0000 0000 0000 1101 0000 0000
```

```
t2 = 0000 0000 0000 0000 0011 1100 0000 0000
```

- L'istruzione and $t0, $t1, $t2 ha questo effetto

```
t0 = 0000 0000 0000 0000 0000 1100 0000 0000
```

  - Notare che se si desidera forzare alcuni bit di $t0 a 0 basta porre a 0 i bit corrispondenti in $t2 (in modo che il risultato dell'AND sia 0)

OR bit a bit

- Supponiamo che il contenuto dei due registri $t1 e $t2 sia il seguente

```
t1 = 0000 0000 0000 0000 0000 1101 0000 0000
```

```
t2 = 0000 0000 0000 0000 0011 1100 0000 0000
```

- L'istruzione or $t0, $t1, $t2 ha questo effetto

```
t0 = 0000 0000 0000 0000 0011 1101 0000 0000
```

  - Notare che se si desidera forzare alcuni bit di $t0 a 1 basta porre a 1 i bit corrispondenti in $t2 (in modo che il risultato dell'OR sia 1)

AND e OR immediati

- Le costanti sono molto utili nelle operazioni logiche

- Per questo motivo il MIPS fornisce le istruzioni di
  - AND immediato (andi)
  - OR immediato (ori)

  per effettuare l'operazione logica corrispondente tra il registro sorgente e una costante, salvando il risultato nel registro destinazione

NOR bit a bit

- Supponiamo che il contenuto dei due registri $t1 e $t2 sia il seguente

```
t1 = 0000 0000 0000 0000 0000 1101 0000 0000
```

```
t2 = 0000 0000 0000 0000 0011 1100 0000 0000
```

- L'istruzione nor $t0, $t1, $t2 ha questo effetto

```
t0 = 1111 1111 1111 1111 1100 0010 1111 1111
```

NOT bit a bit

- Mediante il NOR è facile realizzare anche il NOT
  - NOT è un operatore unario e per i progettisti MIPS tutte le operazioni aritmetiche e logiche hanno tre operandi

```
t0 = 1111 1111 1111 1111 1100 0010 1111 1111
```

- L'istruzione nor $t0, $t0, $zero ha questo effetto

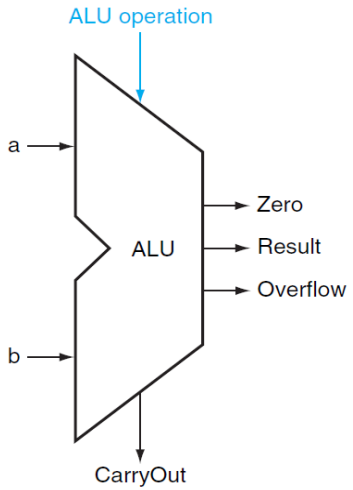
```
t0 = 0000 0000 0000 0000 0011 1101 0000 0000
```

x_2	x_1	nor
0	0	1
0	1	0
1	0	0
1	1	0

Non è stata implementata la versione immediata dell'operazione NOR nel MIPS

Il ruolo dell'ALU

- La ALU a 32 bit che abbiamo studiato effettua le istruzioni aritmetiche e logiche viste finora
  - add, addi, sub, and, andi, or, ori, nor



- Con una piccola modifica al progetto dell'ALU è possibile supportare anche operazioni aggiuntive
  - Ad esempio lo XOR tra due stringhe a 32 bit
  - Che tipo di modifica?

XOR bit a bit

- Supponiamo che il contenuto dei due registri $t1 e $t2 sia il seguente

```
t1 = 0000 0000 0000 0000 0000 1101 0000 0000
```

```
t2 = 0000 0000 0000 0000 0011 1100 0000 0000
```

- L'istruzione xor $t0, $t1, $t2 ha questo effetto

```
t0 = 0000 0000 0000 0000 0011 0001 0000 0000
```

XOR bit a bit

- Supponiamo che il contenuto del registro $t0 sia il seguente

```
t0 = 0000 0000 0000 0000 0011 0001 0000 0000
```

- L'istruzione xor $t0, $t0, $t0 ha questo effetto

```
t0 = 0000 0000 0000 0000 0000 0000 0000 0000
```

  - E' un modo molto veloce ed efficiente per annullare un registro

Altre operazioni logiche

- Il MIPS offre anche diversi tipi di operazioni di scorrimento o traslazione (shift) sul contenuto dei registri
  - Shift logici
  - Shift aritmetici

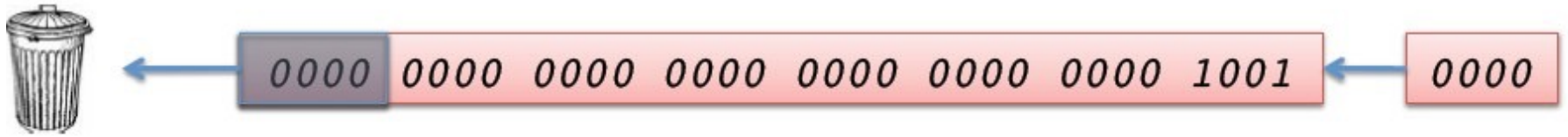
- Queste operazioni sono effettuate tramite hardware aggiuntivo che non studieremo (shift register)

Shift logico a sx

- Corrisponde ad inserire un certo numero n di zeri a partire dalla posizione meno significativa di una word, traslandola a sx di n posti
  - Gli n bit più significativi vengono persi
- Ad esempio, se il contenuto del registro $s0 è

0000 0000 0000 0000 0000 0000 0000 1001

l'istruzione sll $t2, $s0, 4 ha questo effetto:



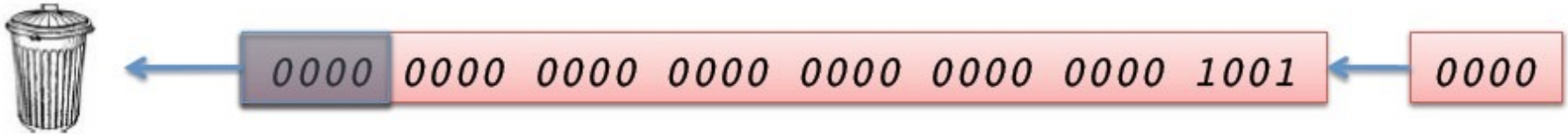
Il risultato viene memorizzato nel registro $t2

Shift logico a sx

- Un effetto collaterale di questa istruzione è che il risultato corrisponde a moltiplicare l'operando per 2^n
- Esempio: se il contenuto del registro $s0 è

0000 0000 0000 0000 0000 0000 0000 1001 cioè 9_{10}

l'istruzione sll $t2, $s0, 4 produce in $t2



che corrisponde a $2^4 + 2^7 = 144_{10}$

Notiamo che $144_{10} = 9_{10} \times 2^4 = 9_{10} \times 16_{10}$

Shift logico a sx

- Vediamo se questo effetto collaterale si verifica sempre
  - Supponiamo che il contenuto del registro $s0 sia 10000000000000000000000000000100, cioè -2147483644_{10}
  - Effettuiamo l'istruzione sll $t2, $s0, 1
  - Il contenuto del registro $t2 sarà

    00000000000000000000000000001000 cioè 8_{10}
  - L'effetto collaterale non si è verificato!
- Il problema è che il numero che si otterrebbe dall'operazione $-2147483644_{10} \times 2$ non è rappresentabile con 32 bit in complemento a 2

Shift logico a sx

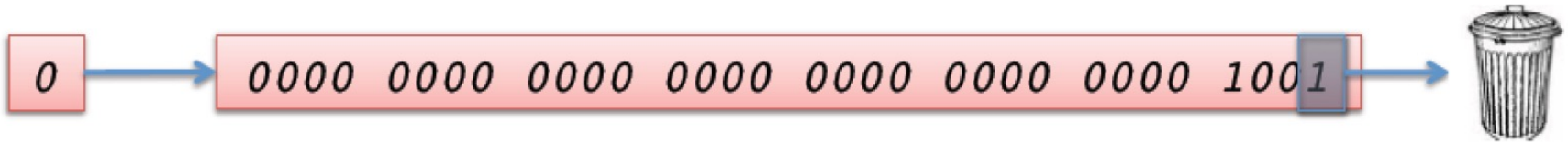
- Se invece restiamo nel range di rappresentabilità $[-2^{31}, 2^{31}-1]$, la moltiplicazione per potenze di 2 tramite shift a sx va a buon fine:
  - Supponiamo che il contenuto del registro $s0 sia 11111111111111111111111111111110, cioè -2_{10}
  - Effettuiamo l'istruzione sll $t2, $s0, 1
  - Il contenuto del registro $t2 sarà 11111111111111111111111111111100 cioè -4_{10}
  - Corretto!

Shift logico a dx

- Corrisponde ad inserire un certo numero n di zeri a partire dalla posizione più significativa di una word, traslandola a dx di n posti
  - Gli n bit meno significativi vengono persi
- Ad esempio, se il contenuto del registro $s0 è

0000 0000 0000 0000 0000 0000 0000 1001

l'istruzione srl $t2, $s0, 1 ha questo effetto:



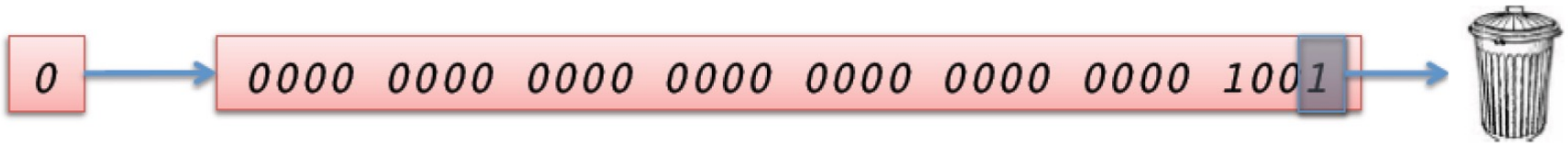
Il risultato viene memorizzato nel registro $t2

Shift logico a dx

- Un effetto collaterale di questa istruzione è che il risultato corrisponde a dividere l'operando per 2^n
- Esempio: se il contenuto del registro $s0 è

`0000 0000 0000 0000 0000 0000 0000 1001` cioè 9_{10}

l'istruzione srl $t2, $s0, 1 produce in $t2



che corrisponde a $4_{10} = 9_{10} / 2$ (divisione intera)

Shift logico a dx

- Vediamo se questo effetto collaterale si verifica sempre
  - Supponiamo che il contenuto del registro $s0 sia 11111111111111111111111111111100 cioè -4_{10}
  - Effettuiamo l'istruzione srl $t2, $s0, 1
  - Il contenuto del registro $t2 sarà 01111111111111111111111111111110 cioè un numero positivo, nonostante il risultato dell'operazione sia rappresentabile!
  - Scorretto!
  - Possiamo risolvere il problema?

Shift aritmetico

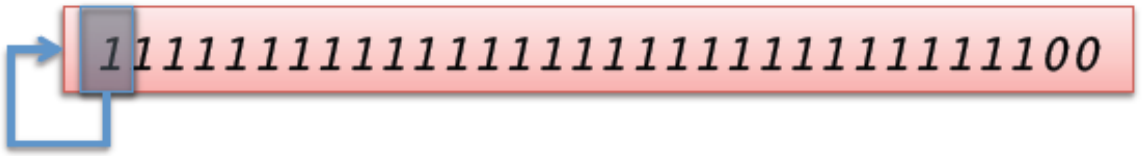
- Corrisponde ad inserire un certo numero n di bit, tutti uguali al bit di segno, a partire dalla posizione più significativa di una word, traslandola a dx di n posti
- Ad esempio, se il contenuto del registro $s0 è

`11111111111111111111111111111100`

l'istruzione sra $t2, $s0, 1 memorizza in $t2

`11111111111111111111111111111110`

che corrisponde ad effettuare



Shift aritmetico

- Notare che lo shift aritmetico a sinistra non ha nessun senso
- Infatti in quel caso,
  - Se il risultato è rappresentabile, lo shift logico funziona come aritmetico
  - Se il risultato non è rappresentabile, non c'è altro da fare
- Alcune architetture offrono entrambi i tipi di shift
  - Il MIPS, facendo parte della famiglia RISC, fornisce solo l'essenziale

Codifica delle istruzioni

- Le istruzioni di shift sono in formato R
- Ad esempio, consideriamo l'istruzione

sll $t2, $s0, 4

- I vari campi del codice macchina corrispondente sono

op	rs	rt	rd	Ind/Shamt	Funct
0	0	16	10	4	0
		$s0	$t2		

$zero	0
$at	1
$v0 - $v1	2-3
$a0 - $a3	4-7
$t0 - $t7	8-15
$s0 - $s7	16-23
$t8 - $t9	24-25
$k0 - $k1	26-27
$gp	28
$sp	29
$fp	30
$ra	31

Da qui si comprende l'utilita' del campo "shift amount"

Riassumendo

- Ciascuna istruzione MIPS viene espressa come una parola di 32 bit
- Un programma quindi è una sequenza di parole di 32 bit
- Tale sequenza viene scritta in locazioni consecutive di memoria

In momenti diversi,
la stessa memoria può contenere
programmi diversi!

Riepilogo e riferimenti

- Codifica delle istruzioni in linguaggio macchina
  - [PH] par. 2.5
- Istruzioni MIPS per operazioni logiche
  - [PH] par. 2.6