Università degli Studi di Cassino e del Lazio Meridionale

Corso di Calcolatori Elettronici

MIPS: modello di programmazione

Anno Accademico 2016/2017

Francesco Tortorella

CPU

- Compito della CPU: eseguire **istruzioni**
- Le istruzioni costituiscono le operazioni primitiva eseguibili dalla CPU
- CPU diverse implementano differenti insiemi di istruzioni. L'insieme di istruzioni che una particolare CPU implementa è detto **Instruction Set Architecture (ISA).**
  - Esempi: Intel 80x86 (Pentium 4), IBM/Motorola PowerPC (Macintosh), MIPS, Intel IA64, ...

Instruction Set Architectures

- La tendenza progettuale iniziale era quella di aggiungere sempre più numerose e complesse istruzioni in modo da poter eseguire operazioni complesse:
  CISC (Complex Instruction Set Computing)
  - L'architettura VAX aveva un'istruzione per moltiplicare i polinomi !
- Negli anni 80 si afferma una nuova filosofia che mira a costruire processori più efficienti in base a due principi progettuali:
  - Mantenere il set di istruzioni piccolo e semplice: ciò rende più facile costruire hardware veloce
  - Non implementare operazioni complesse (e raramente eseguite) tramite istruzioni dirette, ma realizzarle via software, componendo istruzioni più semplici

- **RISC (Reduced Instruction Set Computing)**

MIPS

- **MIPS**: un'azienda che ha costruito una delle prime architetture RISC commerciali
- Studieremo l'architettura MIPS in qualche dettaglio
- Perchè MIPS invece di (es.) Intel 80x86 ?
  - L'architettura e l'ISA del MIPS sono molto più semplici ed eleganti
  - Il MIPS è largamente utilizzato in applicazioni "embedded", contrariamente all'INTEL 80x86 che è praticamente limitato al solo segmento del personal computer

Architettura del processore MIPS

Microprocessor without Interlocking Pipe Stages

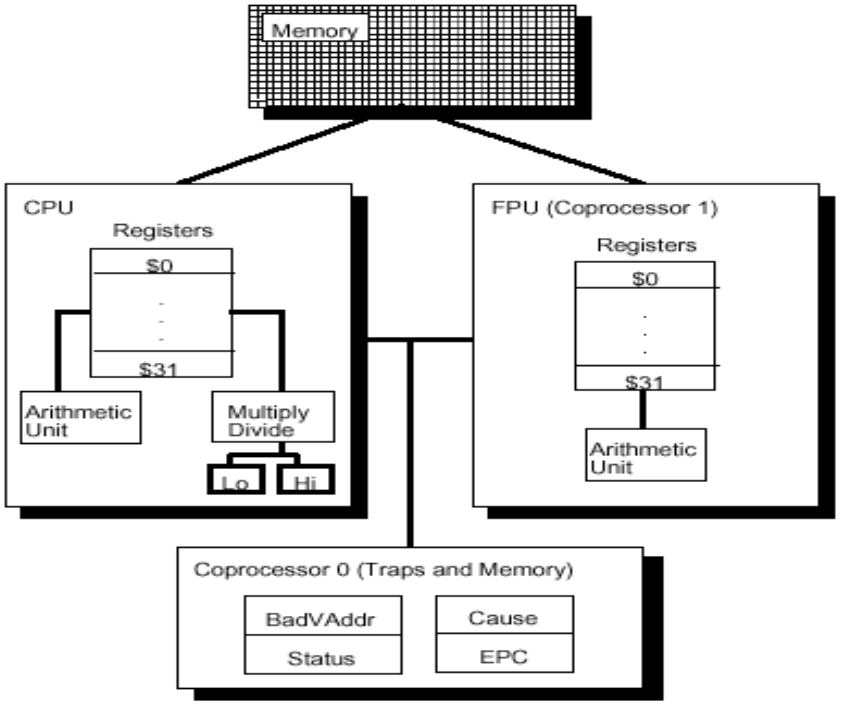
- Architettura Load/Store con istruzioni aritmetiche registro-registro a 3 operandi
- Istruzioni di 32-bit - 3 Formati (R, I, J)
- 32 registri generali di 32 bit (R0 contiene 0, R31 riceve l'indirizzo di ritorno) (+ HI, LO)
- Modi d'indirizzamento: Register, Immediate, Base+Offset, PC-relative
- Immediati a 16-bit + istruzione LUI

Architettura del processore MIPS

- Supporto per interi in complemento a 2 di 8 (byte), 16 (halfword) e 32 (word) bit e, con coprocesore opzionale, per numeri floating-point IEEE 754 singola e doppia precisione
- Branch semplici senza codici di condizione
- *Delayed branch* (l'istruzione dopo il salto viene comunque eseguita) e *Delayed load* (l'istruzione dopo una load non deve usare il registro caricato), senza interlock

Coprocessori

- Può supportare fino a 4 coprocessori, numerati da 0 a 3
- Il coprocessore di controllo del sistema (coprocessore 0) è integrato nel chip e gestisce la memoria e le eccezioni
- Il coprocessore floating-point (coprocessore 1) opzionale ha 32 registri di 32-bit ($f0 - $f31), di cui sono utilizzabili quelli di posto pari in semplice o doppia precisione



Registri del MIPS e convenzione di uso

32 registri generali da 32 bit

registri speciali

PC

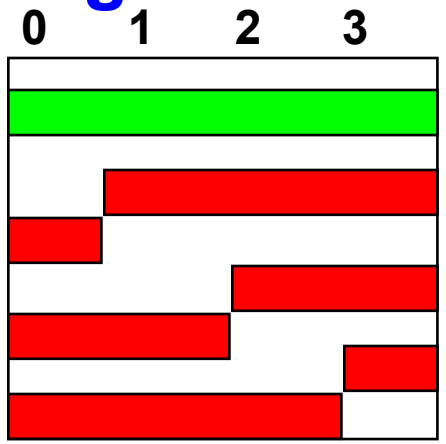
HI
LO

Register name	Number	Usage
$zero	0	constant 0
$at	1	reserved for assembler
$v0	2	expression evaluation and results of a function
$v1	3	expression evaluation and results of a function
$a0	4	argument 1
$a1	5	argument 2
$a2	6	argument 3
$a3	7	argument 4
$t0	8	temporary (not preserved across call)
$t1	9	temporary (not preserved across call)
$t2	10	temporary (not preserved across call)
$t3	11	temporary (not preserved across call)
$t4	12	temporary (not preserved across call)
$t5	13	temporary (not preserved across call)
$t6	14	temporary (not preserved across call)
$t7	15	temporary (not preserved across call)
$s0	16	saved temporary (preserved across call)
$s1	17	saved temporary (preserved across call)
$s2	18	saved temporary (preserved across call)
$s3	19	saved temporary (preserved across call)
$s4	20	saved temporary (preserved across call)
$s5	21	saved temporary (preserved across call)
$s6	22	saved temporary (preserved across call)
$s7	23	saved temporary (preserved across call)
$t8	24	temporary (not preserved across call)
$t9	25	temporary (not preserved across call)
$k0	26	reserved for OS kernel
$k1	27	reserved for OS kernel
$gp	28	pointer to global area
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address (used by function call)

Gestione degli indirizzi di memoria

- Spazio di indirizzi di 2^{32} byte (4 Gigabyte, con i 2 superiori riservati al S.O.), ossia 2^{30} word
- L'indirizzamento è al byte (incremento di 4 per passare da una word alla successiva)
- L'indirizzo di una word è quello del suo primo byte (byte di indirizzo minore)
- Negli accessi, l'indirizzo di un dato di s byte deve essere allineato, ossia A mod s = 0 (esistono istruzioni per accedere a dati disallineati)
- L'ordinamento dei byte in una word può essere sia *big-endian* (il primo byte è quello più significativo) che *little-endian* (il primo byte è quello meno significativo), in dipendenza del valore logico su di un pin

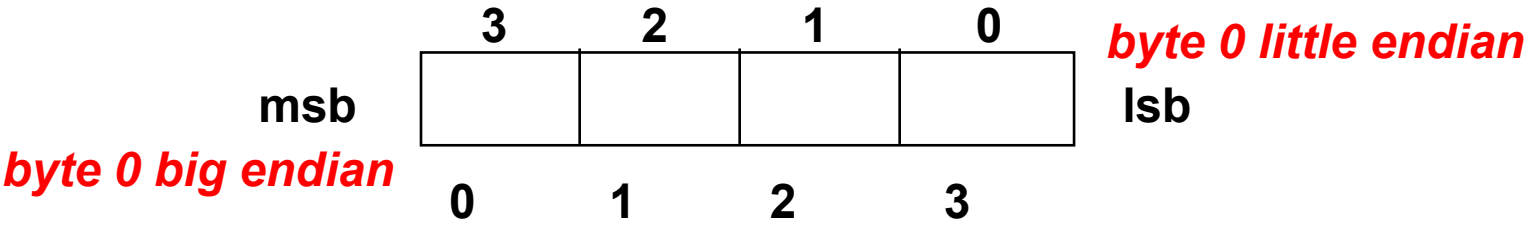
Scelte implementative nella gestione degli indirizzi



- può una word essere memorizzata in qualunque indirizzo della memoria ?

- come si distribuiscono gli indirizzi dei byte appartenenti alla word ?

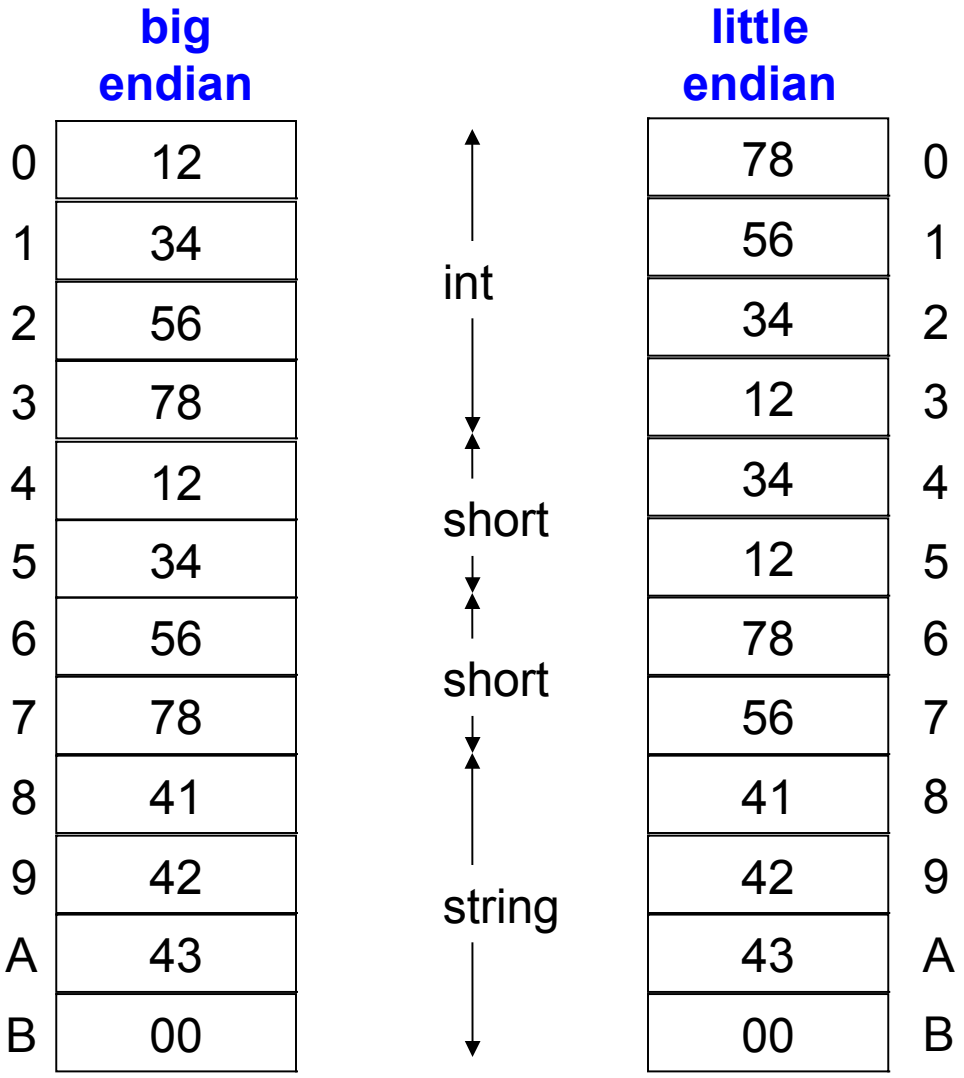
Big Endian (IBM 360/370, Motorola 68k, Sparc, HP PA)
Little Endian (Intel 80x86, DEC Vax, DEC Alpha)



```
int i = 0x12345678;

struct s_point{
    short x;
    short y;
} p =
{0x1234,0x5678};

char str[]="ABC";
```

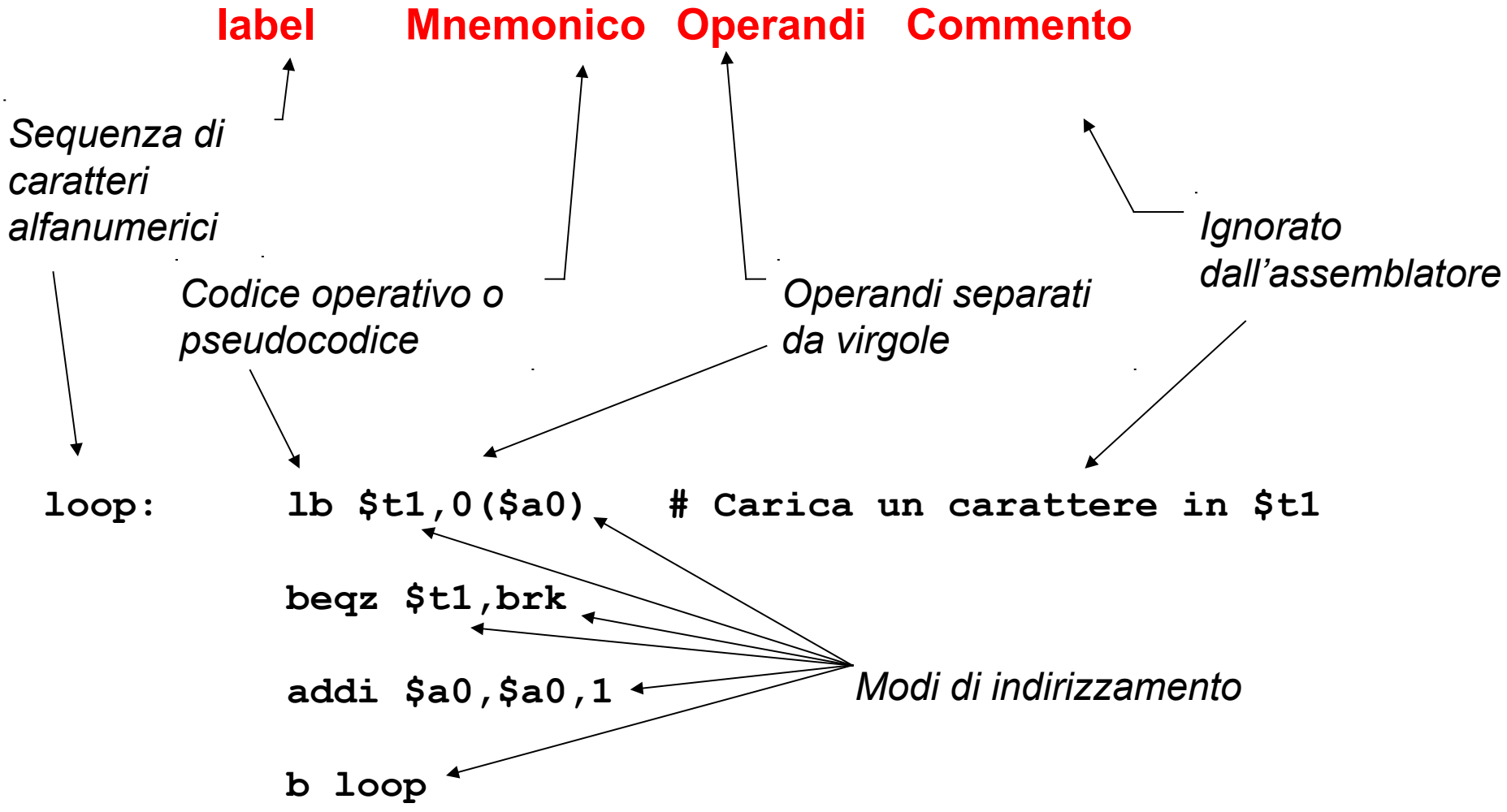


Il linguaggio Assembly del MIPS

- Linguaggio macchina
- Linguaggio definito da un insieme di istruzioni, codificate come stringhe di bit, che il processore può interpretare ed eseguire direttamente

- Linguaggio Assembly
- Linguaggio simbolico, vicino al linguaggio macchina, che definisce:
  - Uno *mnemonico* per ogni istruzione in L.M.
  - Un *formato* per le linee di programma
  - Formati per la specifica *della modalità di indirizzamento*
  - *Direttive*

Sintassi istruzioni



Classi di Istruzioni

- Istruzioni aritmetiche
- Istruzioni logiche
- Istruzioni di movimento dati
- Istruzioni di confronto
- Istruzioni per il controllo di flusso

Istruzioni aritmetiche

Istruzione	*Significato*	
add $t1,$t2,$t3	$t1 = $t2 + $t3	*eccezione possibile*
addi $t1,$t2,100	$t1 = $t2 + 100	*eccezione possibile*
addu $t1,$t2,$t3	$t1 = $t2 + $t3	*nessuna eccezione*
addiu $t1,$t2,100	$1 = $2 + 100	*nessuna eccezione*
sub $t1,$t2,$t3	$t1 = $t2 – $t3	*eccezione possibile*
subu $t1,$t2,$t3	$t1 = $t2 – $t3	*nessuna eccezione*
mult $t2,$t3	Hi, Lo = $t2 x $t3	*64-bit con segno*
multu $t2,$t3	Hi, Lo = $t2 x $t3	*64-bit senza segno*
div $t2,$t3	Lo = $t2 ÷ $t3 Hi = $t2 mod $t3	*Lo = quoziente, Hi = resto*
divu $t2,$t3	Lo = $t2 ÷ $t3 Hi = $t2 mod $t3	*Quoziente & resto* *senza segno*
sra $t1,$t2,10	$1 = $2 >> 10	*Shift aritmetico a destra*
srav $t1,$t2, $t3	$1 = $2 >> $3	*Shift aritm. a destra di # bit variabile*

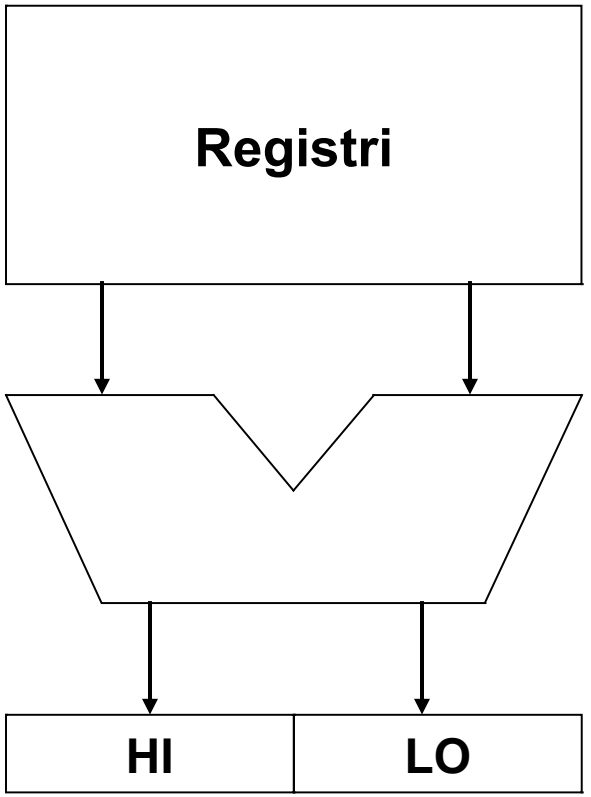
Nota: gli immediati sono valori a 16 bit (con sign extension)

Multiply/Divide

° Avviano l'operazione
  - MULT rs, rt
  - MULTU rs, rt
  - DIV rs, rt
  - DIVU rs, rt

° Trasferiscono il risultato
  - MFHI rd
  - MFLO rd

° Caricano HI e LO
  - MTHI rd
  - MTLO rd



- Durano più cicli e vengono eseguite in parallelo con l'esecuzione di altre istruzioni, da un'unità indipendente
- I tentativi di accesso prematuro a Lo e Hi sono interlocked
- DIV non verifica /0 e overflow (min neg/-1)
- Le 2 istruzioni che seguono MFHI e MFLO non devono modificare Hi e Lo

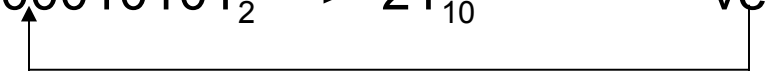
Istruzioni di scorrimento aritmetiche

Uno scorrimento a destra di n bit, può essere visto come una divisione per 2^n:

Es.:

01010110_2 --> 86_{10} con uno scorrimento a destra di 2 bit si ottiene

00010101_2 --> 21_{10} vengono inseriti due 0 a sinistra



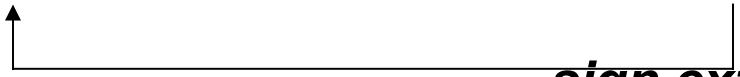
Che cosa succede se si considerano numeri signed ?

Es.:

10010110_2 --> -106_{10} con uno scorrimento a destra di 2 bit si ottiene

00100101_2 --> $+37_{10}$ **errato !**

11100101_2 --> -27_{10} vengono inseriti due 1 a sinistra **corretto**



sign extension

Istruzioni logiche

Istruzione	*Significato*	
and $t1,$t2,$t3	$t1 = $t2 & $t3	*AND bit a bit*
andi $t1,$t2,10	$t1 = $t2 & 10	*AND reg, costante*
or $t1,$t2,$t3	$t1 = $t2 | $t3	*OR bit a bit*
ori $t1,$t2,10	$t1 = $t2 | 10	*OR reg, costante*
xor $t1,$t2,$t3	$t1 = $t2 ⊕ $t3	*XOR bit a bit*
xori $t1, $t2,10	$t1 = $t2 ⊕ 10	*XOR reg, costante*
nor $t1,$t2,$t3	$t1 = ~($t2 | $t3)	*NOR bit a bit*
sll $t1,$t2,10	$t1 = $t2 << 10	*Shift a sinistra di # bit costante*
sllv $t1,$t2,$t3	$t1 = $t2 << $t3	*Shift a sinistra di # bit variabile*
srl $t1,$t2,10	$t1 = $t2 >> 10	*Shift a destra di # bit costante*
srlv $t1,$t2, $t3	$t1 = $t2 >> $t3	*Shift a destra di # bit variabile*

Uso delle istruzioni logiche

Le istruzioni logiche sono tipicamente utilizzate per accedere e manipolare i singoli bit all'interno delle word.

Es.:

A	1	0	1	0	1	0	0	1
B	0	1	1	1	1	0	1	1

Dati iniziali

M	0	0	0	0	1	1	1	1
N	1	1	1	1	0	0	0	0

Maschere

A and M --> C

B and N --> D

C or D --> E

C	0	0	0	0	1	0	0	1
D	0	1	1	1	0	0	0	0
E	0	1	1	1	1	0	0	1

Similmente, lo XOR è utile per invertire il valore di singoli bit, mentre le istruzioni di shift possono essere impiegate per le conversioni serie-parallelo.

Istruzioni di movimento dati

- Il processore MIPS ha due gruppi distinti di istruzioni per i movimenti dati memoria → registro (**load**) e registro → memoria (**store**).
- E' possibile spostare dati di dimensione pari a 1, 2 o 4 byte; in ogni caso, i trasferimenti da/verso la memoria sono sempre da 32 bit.
- Per i dati non allineati in memoria esistono apposite istruzioni che permettono il trasferimento tramite trasferimenti parziali successivi, ma lasciando all'utente la responsabilità di costruire la sequenza di istruzioni corretta.
- Sono inoltre disponibili istruzioni particolari per lo spostamento da/verso i registri speciali HI, LO.
  - Non esistono istruzioni per il movimento dati tra registri generali. Perché ? Come si possono realizzare ?

Istruzioni load

Istruzione	*Significato*	
lb $t1,*address*	Mem[*address*](8 bit) -> $t1	*esteso con segno*
lbu $t1,*address*	Mem[*address*](8 bit) -> $t1	*esteso con 0*
lh $t1,*address*	Mem[*address*](16 bit) -> $t1	*esteso con segno*
lhu $t1,*address*	Mem[*address*](16 bit) -> $t1	*esteso con 0*
lw $t1,*address*	Mem[*address*](32 bit) -> $t1	
lwl $t1,*address*	Mem[*address*] -> $t1	*Carica $t1 con la parte sinistra della word all'indirizzo non allineato*
lwr $t1,*address*	Mem[*address*] -> $t1	*Carica $t1 con la parte destra della word all'indirizzo non allineato*
lui $t1,*costant*	*costant* x 2^{16} -> $t1	*Carica una costante da 16 bit nella parte alta del registro, azzerando la parte bassa*

Istruzioni store

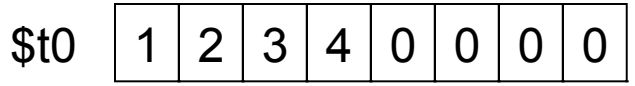
Istruzione	*Significato*	
sb $t1,*address*	$t1(0:7)->Mem[*address*]	
sh $t1,*address*	$t1(0:15)->Mem[*address*]	
sw $t1,*address*	$t1(0:31)->Mem[*address*]	
swl $t1,*address*	$t1->Mem[*address*]	*Carica da $t1 la parte sinistra della word all'indirizzo non allineato*
swr $t1,*address*	$t1->Mem[*address*]	*Carica da $t1 la parte destra della word all' indirizzo non allineato*

Caricamento di una costante

Per trasferire il valore di una costante in un registro, esiste l'istruzione lui che assegna alla parte alta di un registro una costante da 16 bit:

Es.:

lui $t0,0x1234



Come fare per caricare una costante da 16 bit nella parte bassa ?

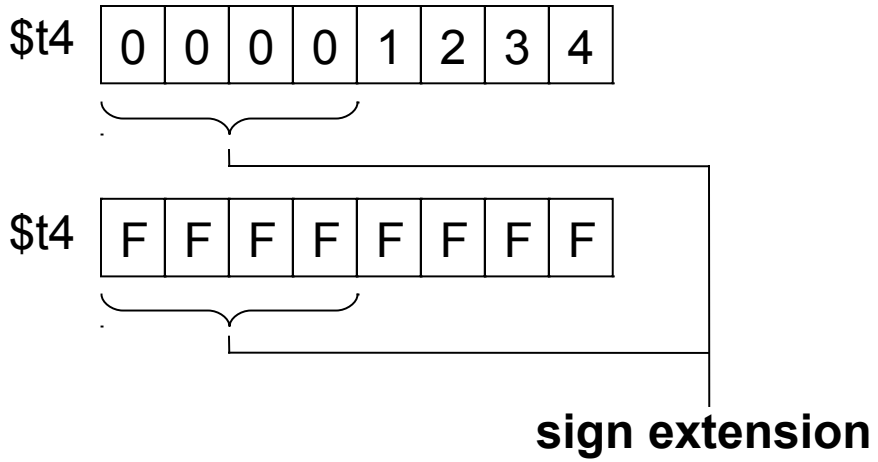
Come fare per caricare una costante da 32 bit nell'intero registro?

Caricamento di una costante

Caricamento di una costante da 16 bit

addi $t4,$zero,0x1234

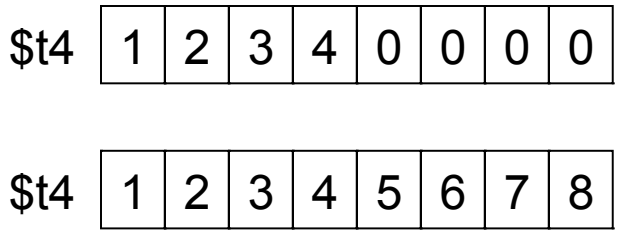
addi $t4,$zero,-1



Caricamento di una costante da 32 bit (0x12345678)

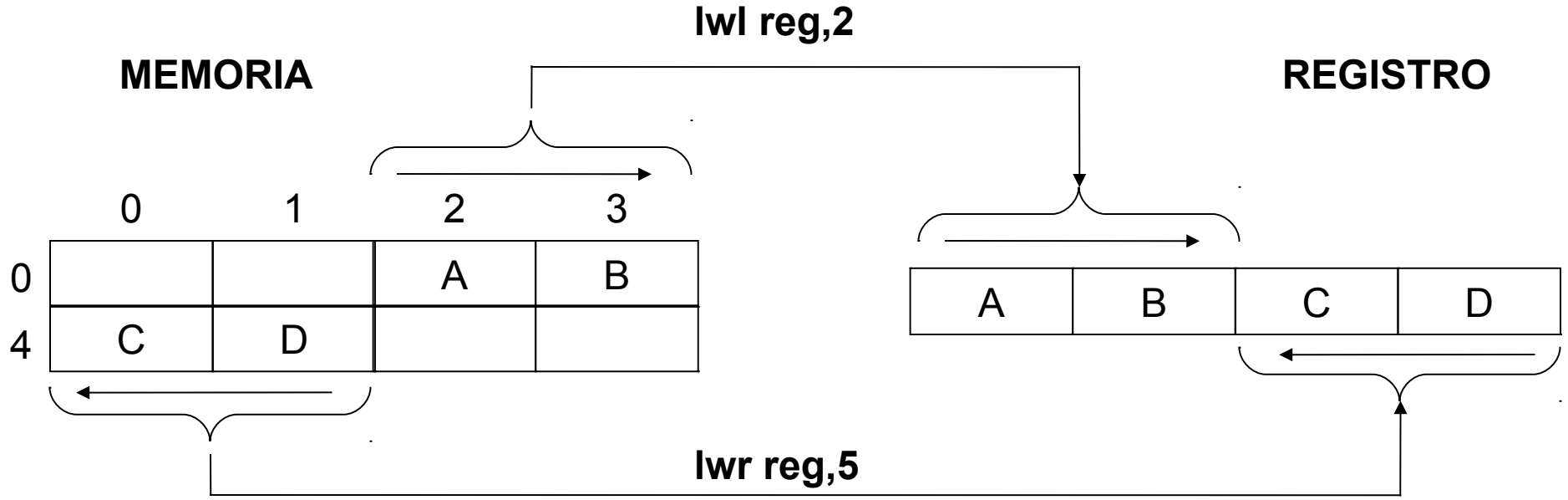
lui $t4,0x1234

ori $t4,$t4,0x5678

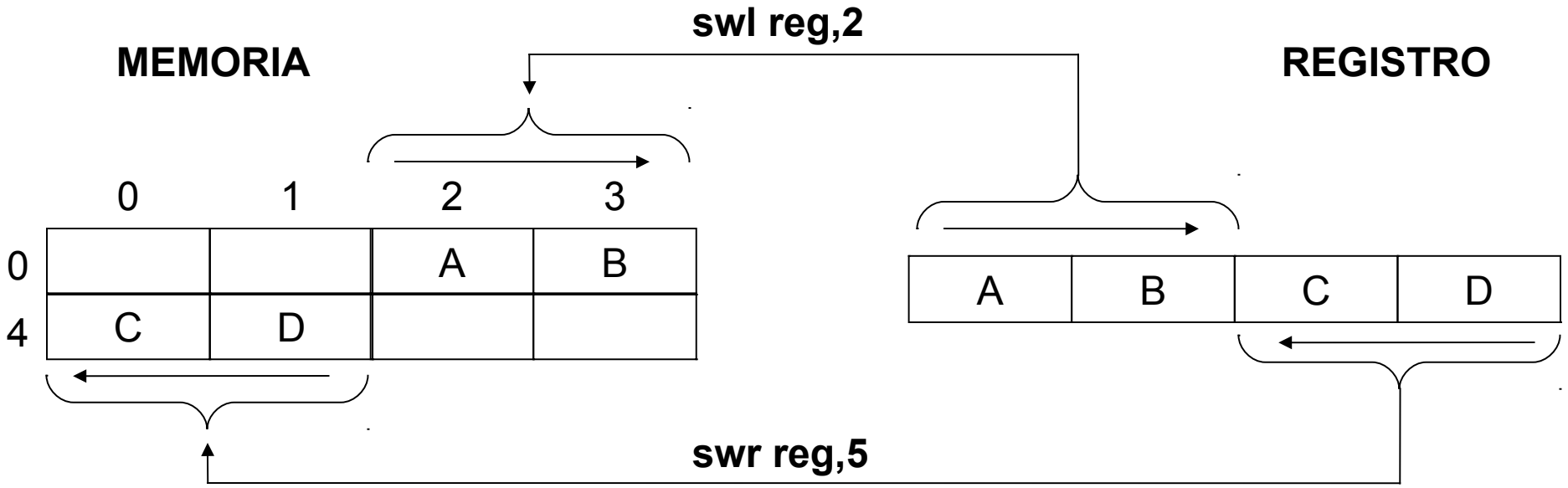


Istruzioni load/store per dati non allineati

Queste istruzioni si usano in coppia per trasferire un dato da/verso un indirizzo non allineato.



Istruzioni load/store per dati non allineati



Istruzioni di move da/verso HI,LO

Istruzione	*Significato*	
mfhi $t0	$t0 = Hi	*Copia Hi in $t0*
mflo $t0	$t0 = Lo	*Copia Lo in $ t0*
mthi $t0	Hi = $t0	*Copia $ t0 in Hi*
mtlo $t0	Lo = $t0	*Copia $ t0 in Lo*

Esempi di uso delle istruzioni viste

- Provare a scrivere le istruzioni MIPS equivalenti alle seguenti istruzioni C (ogni variabile corrisponda ad un registro):
  - a=0;
  - b=a+1;
  - b=b*4;
  - c=a+b;
- Utilizzare opportune istruzioni per stabilire se il valore in un registro
  - è dispari
  - è negativo