

## Architettura del processore MIPS

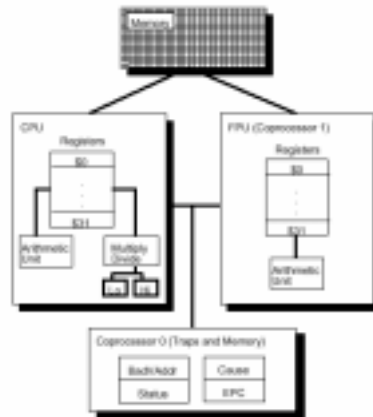
### *Microprocessor without Interlocking Pipe Stages*

- Architettura Load/Store con istruzioni aritmetiche registro-registro a 3 indirizzi
- 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

- Supporto per interi in complemento a 2 di 8 (byte), 16 (halfword) e 32 (word) bit e, con coprocessore 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



F. Tortorella

Corso di Calcolatori Elettronici

Università degli Studi di Cassino

## 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 0
\$a1	5	argument 1
\$a2	6	argument 2
\$a3	7	argument 3
\$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 temporaries (preserved across call)
\$s1	17	saved temporaries (preserved across call)
\$s2	18	saved temporaries (preserved across call)
\$s3	19	saved temporaries (preserved across call)
\$s4	20	saved temporaries (preserved across call)
\$s5	21	saved temporaries (preserved across call)
\$s6	22	saved temporaries (preserved across call)
\$s7	23	saved temporaries (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 (saved by function call)

F. Tortorella

Corso di Calcolatori Elettronici

Università degli Studi di Cassino

## 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  $A \bmod s = 0$  (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

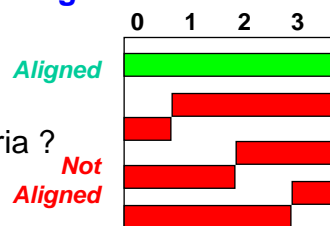
F. Tortorella

Corso di Calcolatori Elettronici

Università degli Studi  
di Cassino

## Scelte implementative nella gestione degli indirizzi

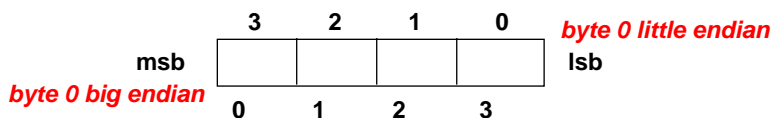
→ può una word essere memorizzata in qualunque indirizzo della memoria ?



→ 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)



F. Tortorella

Corso di Calcolatori Elettronici

Università degli Studi  
di Cassino

```

int i = 0x12345678;
struct s_point{
    short x;
    short y;
} p =
{0x1234,0x5678};
char str[]="ABC";

```

