

Architettura degli Elaboratori

Tipologie di indirizzamento e instruction set MIPS



SAPIENZA
UNIVERSITÀ DI ROMA

Alessandro Checco

alessandro.checco@uniroma1.it

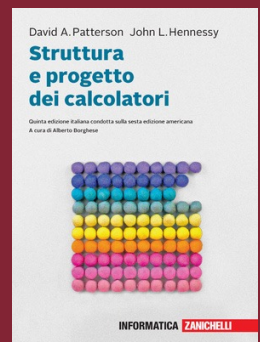
alessandrochecco.github.io

Special thanks and credits:

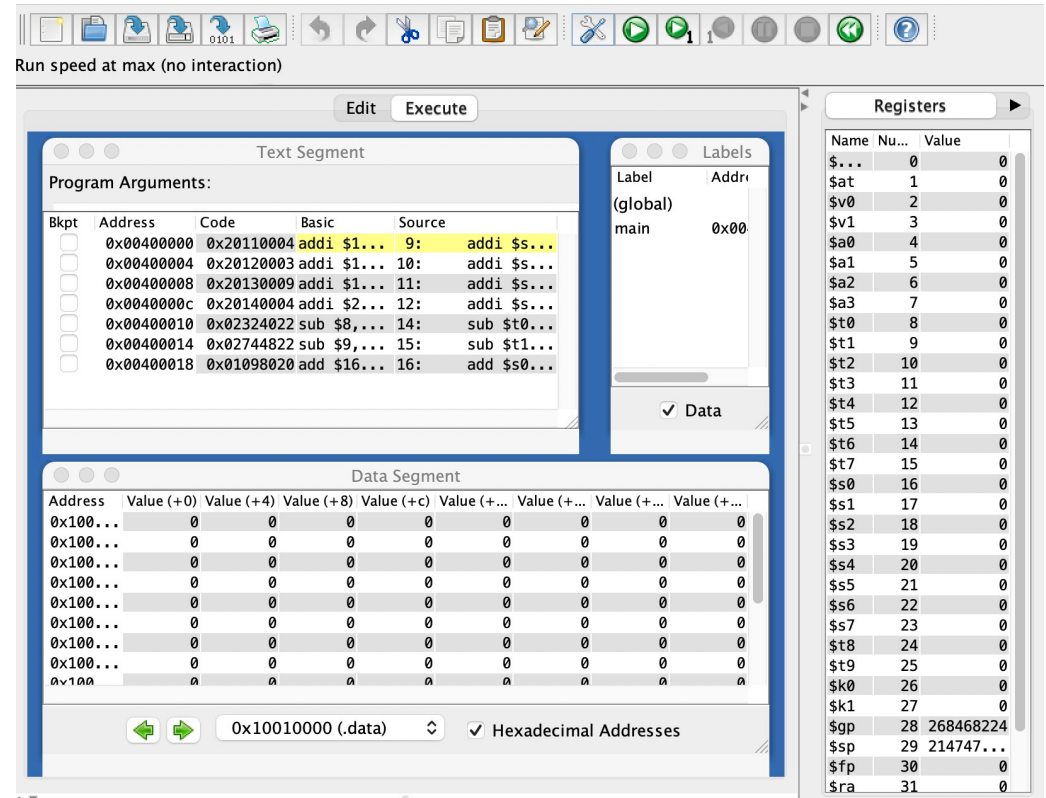
Andrea Sterbini, Iacopo Masi,

Claudio di Ciccio

[S&PdC] 2.1 – 2.8



Download MARS

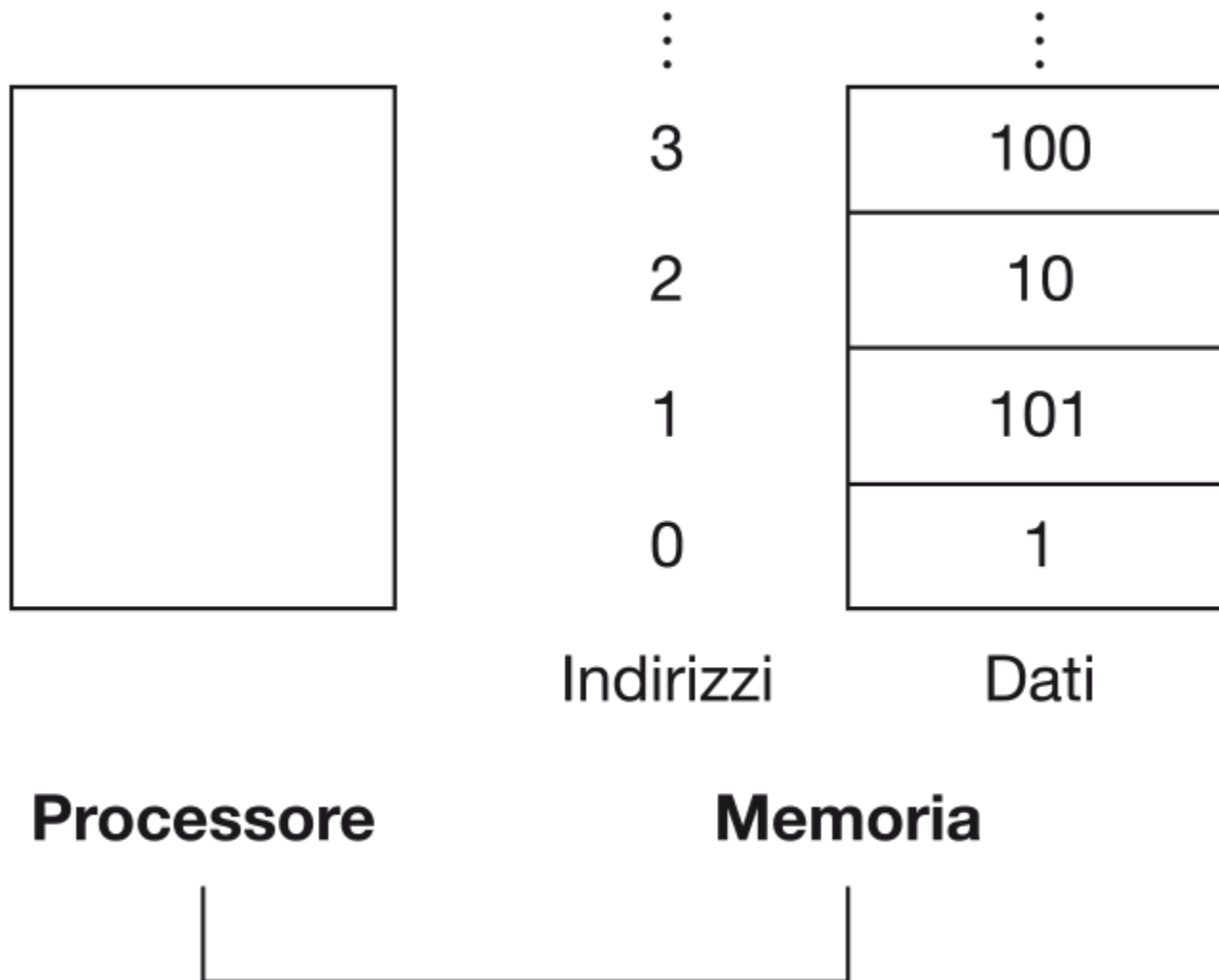


<https://courses.missouristate.edu/KenVollmar/mars/download.htm>

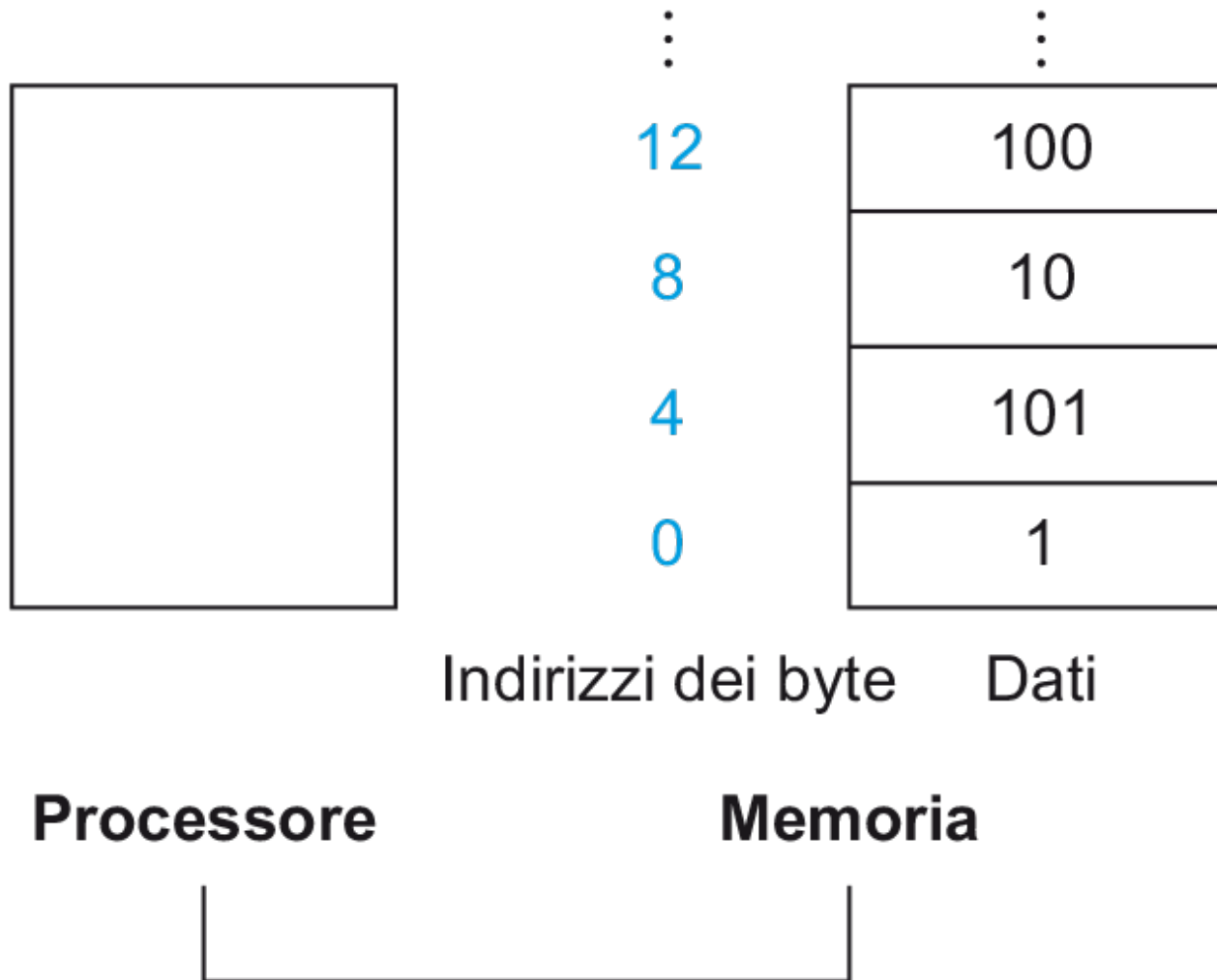
Principi di progettazione

1. La semplicità favorisce la regolarità
2. Minori sono le dimensioni, maggiore è la velocità
3. Un buon progetto richiede buoni compromessi

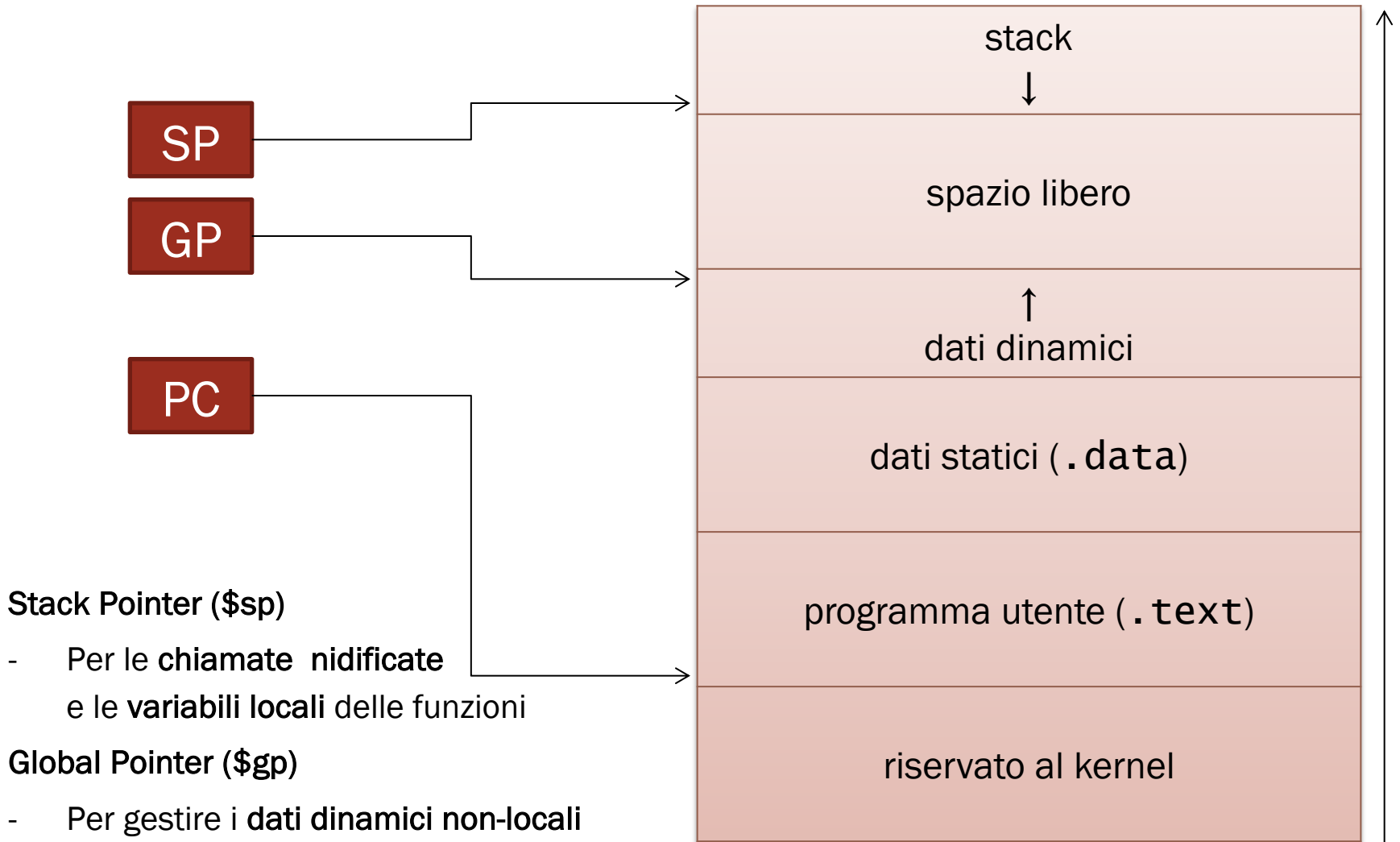
Memoria



Memoria (vincolo di allineamento)



Organizzazione della memoria



Il set di istruzioni (cosa la CPU «sa fare»)

Come abbiamo visto nella IAS machine

Le **fasi di esecuzione** di una istruzione sono:

- **Fetch**/caricamento della istruzione

Dalla posizione indicata dal Program Counter

- **Decodifica**/riconoscimento della istruz.

La Control Unit attiva le parti funzionali necessarie

- **Load**/caricamento di eventuali **argomenti**

A seconda dei modi di indirizzamento (vedi dopo)

- **Esecuzione** della istruzione

In genere da parte dell'ALU

- **Store**/salvataggio del risultato

In memoria o in un registro

- **Aggiornamento del Program Counter**

(contemporaneamente ad altre fasi)

Tipologie di istruzioni:

- **LOAD/STORE**

Trasferiscono dati da/verso la memoria

Es: LW, LH, LB, SW, SH, SB, LWC1, SWC1

- **Logico/Aritmetiche**

Svolgono i calcoli aritmetici e logici

Es: ADD, SUB, MUL, DIV, SLL, SRL, SRA

- **Salti** condizionati e incondizionati

Controllano il flusso logico di esecuzione

Es: J(ump), JAL, BEQZ, BLT, BLE, ...

- **Gestione delle eccezioni/interrupt**

Salvataggio dello stato e suo ripristino

Es: ERET

- **Istruzioni di trasferimento dati**

Non necessarie col memory-mapping

Codifica delle istruzioni (come indicare «cosa fare»)

La **codifica della istruzione** deve indicare:

- Quale operazione va svolta (**Opcode**)

Se necessario quale sotto-operazione

- Quali **argomenti** sono necessari

- Dove mettere il **risultato**

Modi di indirizzamento (come codificare gli argomenti della istruzione):

- **Implicito**

Sorgente/destinazione fissa (p.es. l'AC della IAS)

0 accessi alla memoria

- **Immediato**

Valore costante codificato nella istruzione stessa



0 accessi alla memoria, limitato dalla dim. della istr.

- **Diretto**



1 accesso alla memoria

- **Indiretto**



2 accessi alla memoria

- **A Registro**

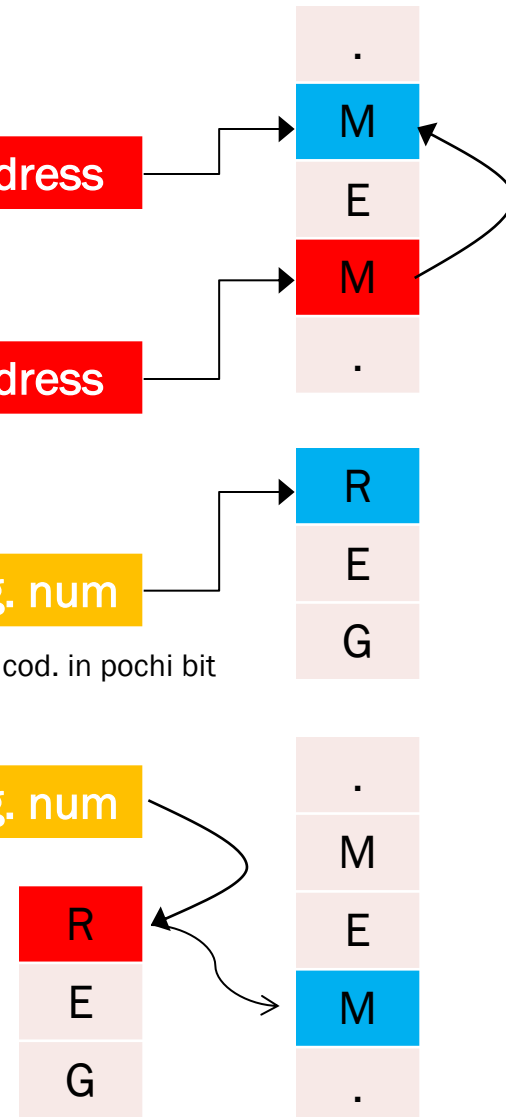


0 accessi alla memoria, cod. in pochi bit

- **A Registro Indiretto**

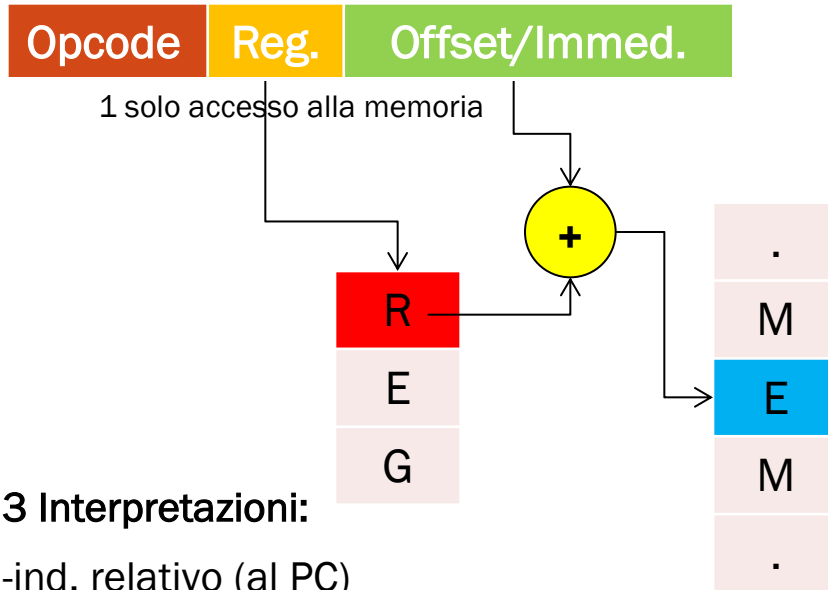


1 accesso alla memoria



Modi di indirizzamento complessi

- Con spiazzamento (offset)



3 Interpretazioni:

-ind. relativo (al PC)

-ind. relativo a registro base

Il registro è un indirizzo, l'offset una distanza

-indicizzazione di un vettore

Il registro contiene un offset, la parte immediata è l'indirizzo della struttura indicizzata (vettore)

- Altri modi di indirizzamento
(non usati in questo corso):

- post indicizzato

subito dopo il registro viene aggiornato

- pre/post incrementato

il registro è incrementato subito prima/dopo

- Istruzioni condizionate (ARM)

La ALU produce dei condition code (alcuni bit)

Ogni istruzione può essere disattivata

- ...

Somma e sottrazione

Testo

Siano

a, b, c, d ed e

variabili contenute rispettivamente in

\$s0, \$s1, \$s2, \$s3, \$s4

Scrivere le istruzioni in codice MIPS che
eseguano questo comando ad alto livello:

$a = (b - c) + (d - e)$



\$t0



\$t1

Soluzione

```
sub $t0,$s1,$s2
```

```
sub $t1,$s3,$s4
```

```
add $s0,$t0,$t1
```

Test

```
addi $s1,$zero,4
```

```
addi $s2,$zero,3
```

```
addi $s3,$zero,9
```

```
addi $s4,$zero,4
```

```
sub $t0,$s1,$s2
```

```
sub $t1,$s3,$s4
```

```
add $s0,$t0,$t1
```

esempio MARS

Movimentazione di dati da/per memoria

Testo	Soluzione
Sia n una variabile il cui valore è nella locazione di memoria indicata da \$s6	lw \$t0,24(\$s5) lw \$t1,(\$s6) # lw \$t1,0(\$s6) add \$t0,\$t1,\$t0 sw \$t0,48(\$s5)
e sia v un vettore la cui base è registrata in \$s5	
Scrivere le istruzioni in codice MIPS che eseguano questo comando ad alto livello: $v[12] = n + v[6]$	

Architetture CISC e RISC (memorandum)

Architettura CISC

(Complex Instruction Set Computer)

- Istruzioni di **dimensione variabile**

Per il fetch della successiva è **necessaria la decodifica**

- **Formato variabile**

Decodifica complessa

- Operandi in memoria

Molti accessi alla memoria per istruzione

- **Pochi registri** interni

Maggior numero di accessi in memoria

- **Modi di indirizzamento complessi**

Maggior numero di accessi in memoria

Durata variabile della istruzione

Conflitti tra istruzioni più complicati

- Istruz. Complesse: pipeline più complicata

Architettura RISC

(Reduced Instruction Set Computer)

- Istruzioni di **dimensione fissa**

Fetch della successiva **senza decodifica della prec.**

- Istruzioni di **formato uniforme**

Per semplificare la fase di decodifica

- **Operazioni ALU solo tra registri**

Senza accesso a memoria

- **Molti registri** interni

Per i risultati parziali senza accessi alla memoria

- **Modi di indirizzamento semplici**

Con spiazzamento, 1 solo accesso a memoria

Durata fissa della istruzione

Conflitti semplici

- Istruz. semplici => pipeline più veloce

L'architettura MIPS 2000

Word da 32 bit

Spazio di indirizzamento da 32 bit (4GByte)

indirizzamento con spiazzamento

(include anche gli ind. più semplici)

Interi in complemento a 2 su 32 bit

32 registri di uso generale

3 CPU

-CPU: ALU e programma

32 registri + Hi/Lo usati da mult e div

Ha accesso alla Memoria

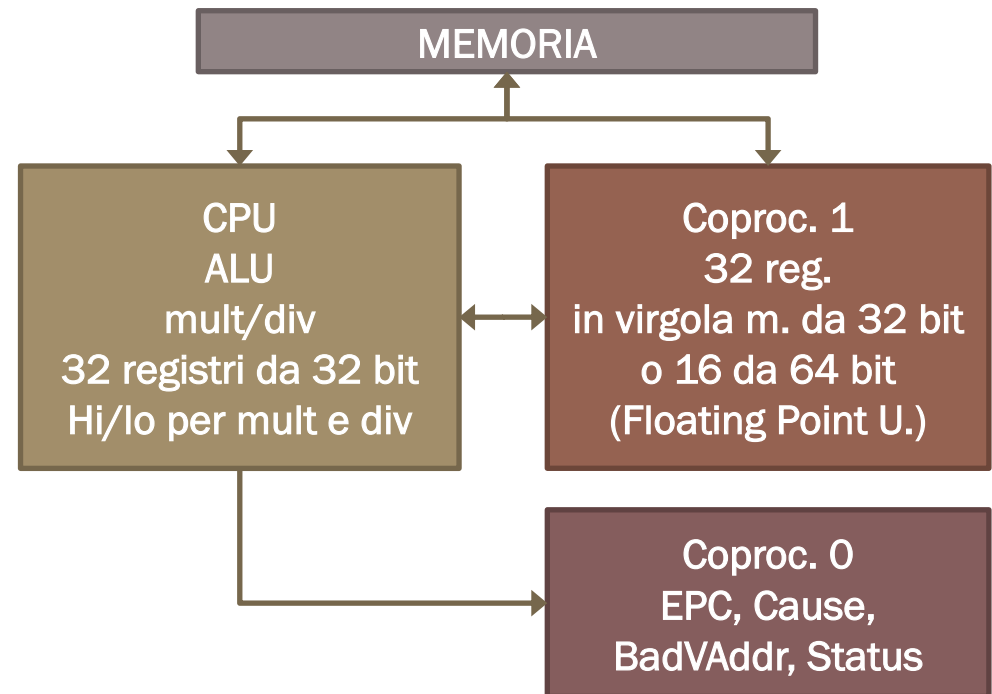
-Coproc. 0: trap, eccezioni, Virtual Mem.

Cause, EPC, Status, BadVAddr

-Coproc. 1: per calcoli in virgola mobile

32 registri da 32 bit usabili come 16 da 64 bit

Ha accesso alla memoria



L'architettura MIPS 2000

Word da 32 bit

Spazio di indirizzamento da 32 bit (4GByte)

indirizzamento con spiazzamento

(include anche gli ind. più semplici)

Interi in complemento a 2 su 32 bit

32 registri di uso generale

3 CPU

-CPU: ALU e programma

32 registri + Hi/Lo usati da mult e div

Ha accesso alla Memoria

-Coproc. 0: trap, eccezioni, Virtual Mem.

Cause, EPC, Status, BadVAddr

-Coproc. 1: per calcoli in virgola mobile

32 registri da 32 bit usabili come 16 da 64 bit

Ha accesso alla memoria

i 32 Registri della CPU

\$zero la costante **zero** (immutabile)

\$at usato dalle **pseudoistruzioni**

\$v0,\$v1 **risultati** delle procedure/funzioni

\$a0..\$a3 **argomenti** delle proc./funzioni

\$t0..\$t7 **temporanei** (salvati dal chiamante)

\$s0..\$s7 **temporanei** (salvati dal chiamato)

\$t8,\$t9 **temporanei** (s. dal chiamante)

\$k0,\$k1 **kernel** (interruzioni/eccezioni)

\$gp **global pointer** (memoria dinamica)

\$sp **stack pointer** (stack delle funzioni)

\$fp **frame pointer** (funzioni)

\$ra **return address**

Formato delle istruzioni MIPS

Istruzioni della CPU tutte da 32 bit con formato molto simile

R-type: (tipo a Registro)

- SENZA accesso alla memoria
- Istruzioni Aritmetico/Logiche

Esempio: **add \$t0, \$t1, \$t2**
sll \$t0, \$t1, 5

Op	Rs	Rt	Rd	Shamt	Func
6 bit	5bit	5bit	5bit	5 bit	6 bit

$\$t0 \leftarrow \$t1 + \$t2$

$\$t0 \leftarrow \$t1 \ll 5$

I-type: (tipo Immediato)

- Load/Store
- Salti condizionati (salto relativo al PC)

Esempi: **lw \$t1, vettore(\$t0)**
beq \$t0, \$t1, offset

Op	Rs	Rt/d	Immediate
6 bit	5bit	5bit	16 bit

$\$t1 \leftarrow \text{MEM}[\text{vettore} + \$t0]$

$\text{PC} \leftarrow \text{PC} + 4 + \text{offset} \times 4 \text{ se } \$t0 = \$t1$

J-type: (tipo Jump)

- Salti non condizionati (salto assoluto)

Esempio: **j destinazione**

Op	Address
6 bit	26 bit

$\text{PC} \leftarrow \text{destinazione} \times 4$

Formati del Coproc. 1

Anche le istruzioni per le operazioni in virgola mobile sono da 32 bit (fetch senza decode)

I formati più importanti sono:

FR-type: (tipo a Registro Float)

- SENZA accesso alla memoria
- Istruzioni della FPU

Esempio: **add.s \$f0, \$f1, \$f2**

div.d \$f0, \$f2, \$f4

Op	Fmt	F1	F2	F3	Func
6 bit	5bit	5bit	5bit	5 bit	6 bit

$\$f0 \leftarrow \$f1 + \$f2$ (in singola precisione)

$\$f0 \leftarrow \$f2 / \$f4$ (in doppia precisione)

FI-type: (tipo Immediato Float)

- Load/store
- Salti condizionati (salto relativo al PC)

Esempi: **lwc1 \$f1, indirizzo**

bc1f cc, offset

Op	Fmt	F1	Immediate
6 bit	5bit	5bit	16 bit

$\$f1 \leftarrow \text{MEM}[\text{indirizzo}]$

$\text{PC} \leftarrow \text{PC} + 4 + \text{offset} \times 4$ SE il bit **cc** (F1) è falso

Alcuni operatori logici

- `sll $t0,$s1,0x1`
- `srl $t0,$s1,0x2`
- `and $t0,$s1,$s2`
`andi $t0,$s1,0x3`
- `or $t0,$s1,$s2`
`ori $t0,$s1,0x4`
- `nor $t0,$s1,$s2`

Ne faremo un
uso particolare

Ideale per
bitmask

E il NOT? Per mantenere il formato di istruzioni a due operandi, (v. [principio di progettazione 1](#)):

$$\text{NOT}(A) = A \text{ NOR } 0 = \text{NOT}(A \text{ OR } 0)$$

- # shift / left / logic
- # shift / right / logic
- # Logical bitwise “and”
- # Logical bitwise “and”, I-type
- # Logical bitwise “or”
- # Logical bitwise “or”, I-type
- # Logical bitwise “or”, negated

Esercizio: siano

`u` e `v`

due variabili salvate nei registri
`$s0` e `$s1`

rispettivamente. Si assume `u` positivo.

Scrivere il codice MIPS che esegua:

$$v = u \times 256$$

Domanda: da quale valore di `u` in poi il risultato non corrisponde a quello atteso?



Comparazione e salto condizionato

- `beq $s1,$s2,C` # Branch on equal
- `bne $s1,$s2,C` # Branch on not equal
- `blez $s1,C` # Branch on less-than or eq. 0
- `bgez $s1,C` # Branch on greater-than or eq.0
- `bltz $s1,C` # Branch on less than 0
- `bgtz $s1,C` # Branch on greater than 0

C è una ETICHETTA
che identifica
un'istruzione

Pseudocodice delle azioni sottostanti
`PC += 4`
`if ($s1 == $s2) { PC += C << 2 }`

Comparazione e salto condizionato

- `beq $s1,$s2,C` # Branch on equal
- `bne $s1,$s2,C` # Branch on not equal
- `blez $s1,C` # Branch on less-than or eq. 0
- `bgez $s1,C` # Branch on greater-than or eq.0
- `bltz $s1,C` # Branch on less than 0
- `bgtz $s1,C` # Branch on greater than 0

- `slt $s0,$s1,$s2` # Set \$s0 to 1 if \$s1 l.than \$s2
- `slti $s0,$s1,10` # Set \$s0 to 1 if \$s1 l.than 10

Esempio: max di 4 numeri

Codice C

Supponiamo che i valori siano in a, b, c, d

E che il risultato sarà in max

```
max = a;
if (b>max) {
    max = b;
}
if (c>max) {
    max = c;
}
if (d>max) {
    max = d;
}
```


Assembly MIPS

Supponendo che i valori siano nei registri

\$s0, \$s1, \$s2, \$s3

E che il risultato vada nel registro \$s4

```
and $s4,$s4,$zero
or $s4,$zero,$s0
CheckB: slt $t0,$s4,$s1
        beq $t0,$zero,CheckC
        or $s4,$zero,$s1
CheckC: slt $t0,$s4,$s2
        beq $t0,$zero,CheckD
        or $s4,$zero,$s2
CheckD: slt $t0,$s4,$s3
        beq $t0,$zero,End
        or $s4,$zero,$s3
End:    sw $s4,Result
```



Assembly MIPS

Direttive principali per l'assemblatore

- .data** definizione dei dati statici
- .text** definizione del programma

- .ascii** stringa terminata da `\0`
- .byte** sequenza di byte
- .double** sequenza di double
- .float** sequenza di float
- .half** sequenza di half words
- .word** sequenza di words

Codici mnemonici delle **istruzioni**

`add, sub, beq ...`

Codifica mnemonica dei **registri**

`$a0, $sp, $ra ... $s0, $s7`

Etichette (per calcolare gli indirizzi relativi)

`label:`

L'assemblatore converte

il testo del programma in assembly
in codice macchina

e dalle etichette calcola gli indirizzi

Dei salti relativi

Delle strutture dati in memoria (offset)

Dei salti assoluti

NOTA: le **strutture di controllo del flusso** del programma vanno realizzate «a mano» usando i **salti condizionati** e le etichette

Riepilogo – salto assoluto

Salti incondizionati	Salto incondizionato	j 2500	Vai a 10000	Salto all'indirizzo della costante
	Salto indiretto	jr \$ra	Vai all'indirizzo contenuto in \$ra	Salto all'indirizzo contenuto nel registro, utilizzato per il ritorno da procedura e per i costrutti <i>switch</i>
	Salta e collega	jal 2500	\$ra = PC+4; vai a 10000	Chiamata a procedura

j 2500
non è proprio così...

Riepilogo – salto assoluto

j 2500 # In realtà lo usiamo sempre con etichette!

