



SAPIENZA  
UNIVERSITÀ DI ROMA  
DIPARTIMENTO DI INFORMATICA

# Architettura degli Elaboratori

## Lez. 2 – Le istruzioni della CPU

Prof. Andrea Sterbini – [sterbini@di.uniroma1.it](mailto:sterbini@di.uniroma1.it)



# Argomenti

---

## ► Motivazioni

► Nel resto del corso vedremo nel dettaglio la progettazione di una CPU MIPS, che:

- Ha un set di istruzioni ristretto (RISC)
- È volutamente semplice
- È facile da velocizzare con la pipeline
- È facile da parallelizzare
- È un esempio eclatante di progettazione coordinata dell'hardware col software

► Esempi di CPU con architettura RISC:

- **SPARC** di Sun, **MIPS** in sistemi embedded
- **PowerPC** nei Mac e server IBM
- **Cell** delle Play Station Sony
- **ARM** di quasi tutti i cellulari e tablet

## ► Argomenti della lezione

- Tipi di istruzioni
- Modi di indirizzamento degli operandi
- Confronto tra architettura CISC ed architettura RISC
- L'architettura RISC 2000
- Organizzazione della memoria
- Formati delle istruzioni RISC
- L'assemblatore
- Esempio di programma

# Esempio IAS machine: $C=A+B$

- ▶ Se A, B e C sono le locazioni 101, 102, 103 in memoria, il codice per calcolare  $C=A+B$  è

- ▶ LD            101  
                  AC  $\leftarrow$  Mem[101]

- ▶ ADD          102  
                  AC  $\leftarrow$  AC+Mem[102]

- ▶ SD            103  
                  Mem[103]  $\leftarrow$  AC

## Esecuzione delle istruzioni

**Fetch** della istruzione dalla memoria

MAR  $\leftarrow$  PC

IR, IBR  $\leftarrow$  MBR  $\leftarrow$  Mem[MAR]

**Decodifica** della istruzione (da IR)

MAR  $\leftarrow$  IR.Address; CU  $\leftarrow$  IR.Opcode

Sua **esecuzione**

AC  $\leftarrow$  MBR  $\leftarrow$  Mem[101]

**Fetch** della istr. successiva (da IBR)

MAR  $\leftarrow$  IBR.Address ; CU  $\leftarrow$  IBR.Op

sua **esecuzione**

AC  $\leftarrow$  AC + MBR  $\leftarrow$  Mem[102]

**Aggiornamento** del PC

PC  $\leftarrow$  PC + I

**Fetch** della istruzione successiva

MAR  $\leftarrow$  PC

IR, IBR  $\leftarrow$  MBR  $\leftarrow$  Mem[MAR]

**Decodifica** della istruzione (da IR)

MAR  $\leftarrow$  IR.Address; CU  $\leftarrow$  IR.Opcode

Sua **esecuzione**

Mem[103]  $\leftarrow$  MBR  $\leftarrow$  AC

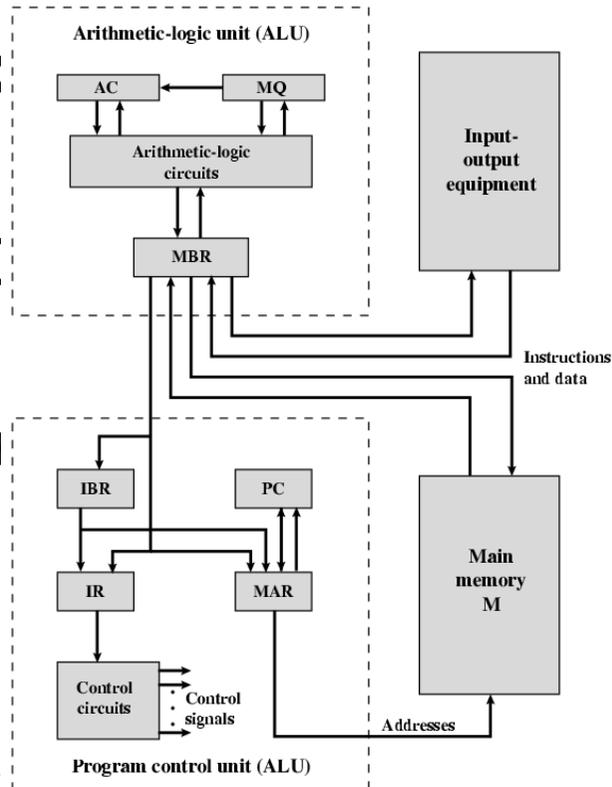
# Esempio IAS machine: $C=A+B$

- ▶ Se A, B e C sono le locazioni 101, 102, 103 in memoria, il codice per calcolare  $C=A+B$  è

▶ LD

▶ AC

▶ SD



## Esecuzione delle istruzioni

**Fetch** della istruzione dalla memoria

$MAR \leftarrow PC$

$IR, IBR \leftarrow MBR \leftarrow Mem[MAR]$

**Decodifica** della istruzione (da IR)

$MAR \leftarrow IR.Address; CU \leftarrow IR.Opcode$

Sua **esecuzione**

$AC \leftarrow MBR \leftarrow Mem[101]$

**Fetch** della istr. successiva (da IBR)

$MAR \leftarrow IBR.Address; CU \leftarrow IBR.Op$

sua **esecuzione**

$AC \leftarrow AC + MBR \leftarrow Mem[102]$

**Aggiornamento** del PC

$PC \leftarrow PC + I$

**Fetch** della istruzione successiva

$MAR \leftarrow PC$

$IR, IBR \leftarrow MBR \leftarrow Mem[MAR]$

**Decodifica** della istruzione (da IR)

$MAR \leftarrow IR.Address; CU \leftarrow IR.Opcode$

Sua **esecuzione**

$Mem[103] \leftarrow MBR \leftarrow AC$

# 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

# 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

# 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)

# 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

# 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

# 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)

# 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, LWCI, SWCI

# 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, LWCI, SWCI

- **Logico/Aritmetiche**

- ▶ Svolgono i calcoli aritmetici e logici
  - ▶ Es: ADD, SUB, MUL, DIV, SLL, SRL, SRA

# 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, LWCI, SWCI

- **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, ...

# 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, LWCI, SWCI

- **Logico/Aritmetiche**

- ▶ Svolgono i calcoli aritmetici e logici
  - ▶ Es: ADD, SUB, MUL, DIV, SLL, SRL, SRA

- **Salta** 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

# 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, LWCI, SWCI

- **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**

# 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):

# 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**

**Opcode**

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

▶ 0 accessi alla memoria

# 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**

**Opcode**

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

▶ 0 accessi alla memoria

- **Immediato** **Opcode** **Operando**

▶ Valore costante codificato nella istruzione stessa

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

# Codifica delle istruzioni (come indicare «cosa fare»)

▶ La **codifica della istruzione** deve indicare: - **Diretto**



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

▶ Se necessario quale sotto-operazione

- Quali **argomenti** sono necessari

- Dove mettere il **risultato**

▶ 1 accesso alla memoria



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

- **Implicito**

**Opcode**

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

▶ 0 accessi alla memoria

- **Immediato** **Opcode** **Operando**

▶ Valore costante codificato nella istruzione stessa

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

# 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**

**Opcode**

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

▶ 0 accessi alla memoria

- **Immediato** **Opcode** **Operando**

▶ Valore costante codificato nella istruzione stessa

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

- **Diretto**

**Opcode**

**Address**

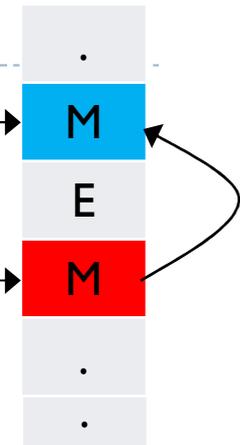
▶ 1 accesso alla memoria

- **Indiretto**

**Opcode**

**Address**

▶ 2 accessi alla memoria



# 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**

**Opcode**

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

▶ 0 accessi alla memoria

- **Immediato** **Opcode** **Operando**

▶ Valore costante codificato nella istruzione stessa

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

- **Diretto**

**Opcode**

**Address**

▶ 1 accesso alla memoria

- **Indiretto**

**Opcode**

**Address**

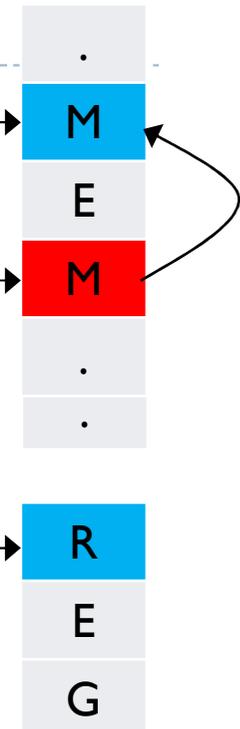
▶ 2 accessi alla memoria

- **A Registro**

**Opcode**

**Reg. num**

▶ 0 accessi alla memoria, cod. in pochi bit



# 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**

**Opcode**

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

- ▶ 0 accessi alla memoria

- **Immediato** **Opcode** **Operando**

- ▶ Valore costante codificato nella istruzione stessa

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

- **Diretto**

**Opcode**

**Address**

- ▶ 1 accesso alla memoria

- **Indiretto**

**Opcode**

**Address**

- ▶ 2 accessi alla memoria

- **A Registro**

**Opcode**

**Reg. num**

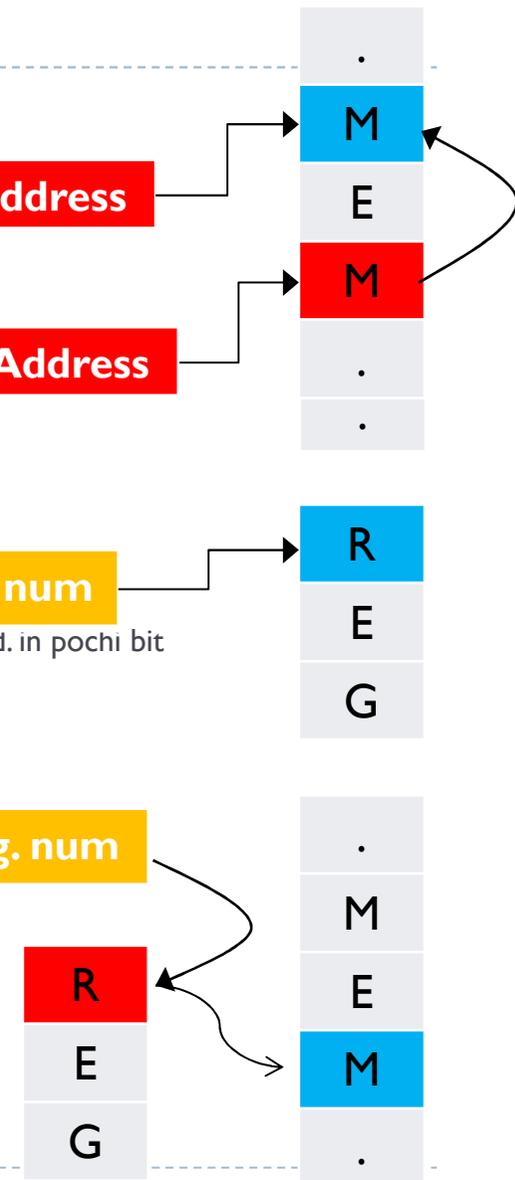
- ▶ 0 accessi alla memoria, cod. in pochi bit

- **A Registro Indiretto**

- ▶ 1 accesso alla memoria

**Opcode**

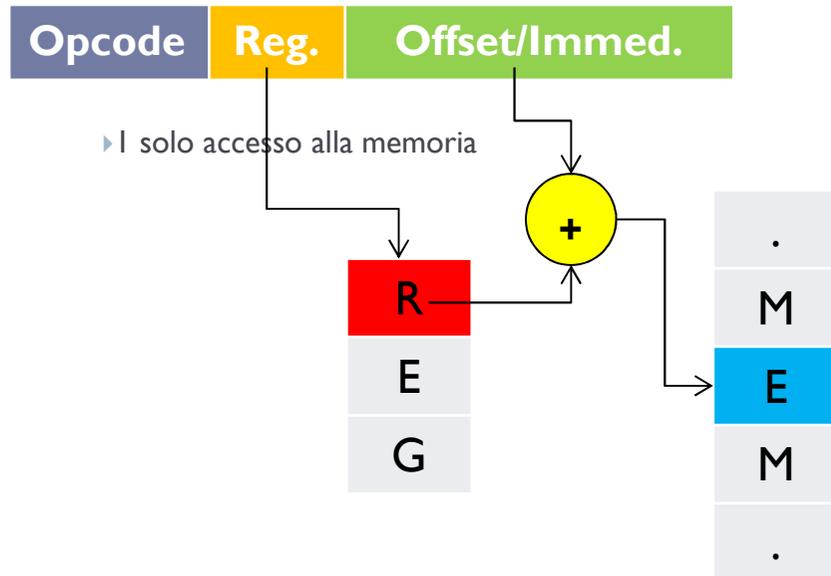
**Reg. num**



# Modi di indirizzamento complessi

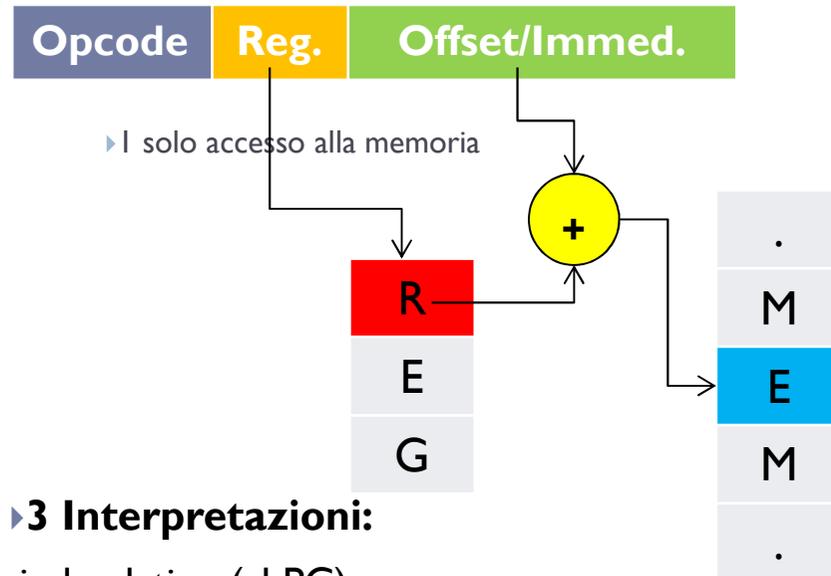
---

## - Con spiazamento (offset)



# Modi di indirizzamento complessi

## - Con spiazamento (offset)



### ▶ 3 Interpretazioni:

-ind. relativo (al PC)

-ind. relativo a registro base

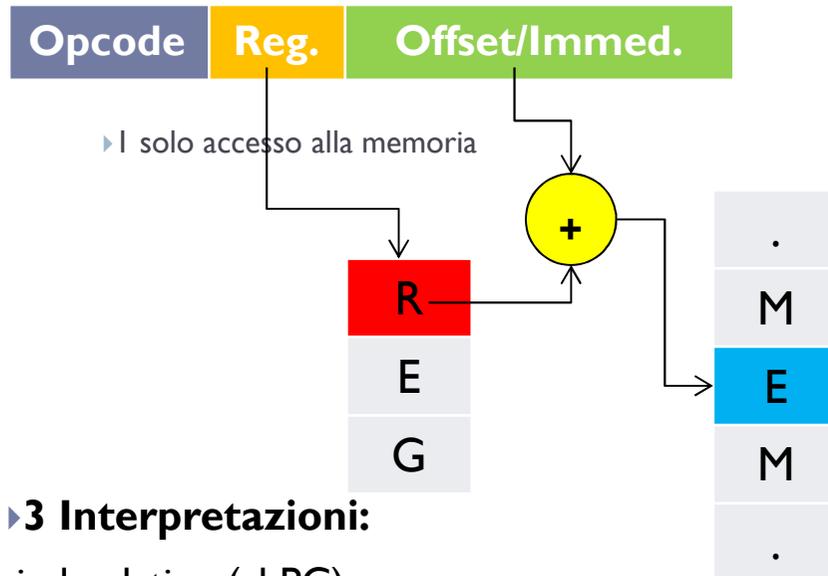
▶ Il registro è un indirizzo, l'offset una distanza

-indicizzazione di un vettore

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

# 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

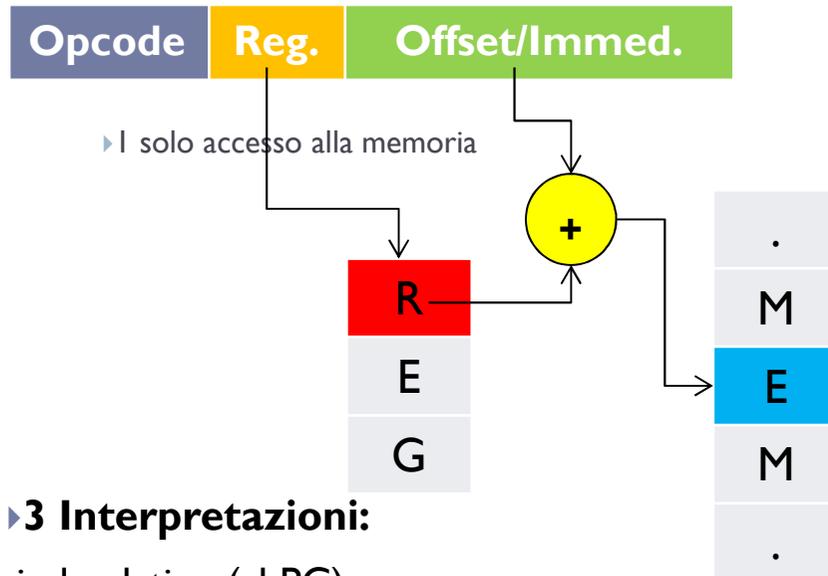
-indicizzazione di un vettore

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

## ▶ - Altri modi di indirizzamento:

# Modi di indirizzamento complessi

## - Con spiazamento (offset)



### ▶ 3 Interpretazioni:

-ind. relativo (al PC)

-ind. relativo a registro base

▶ Il registro è un indirizzo, l'offset una distanza

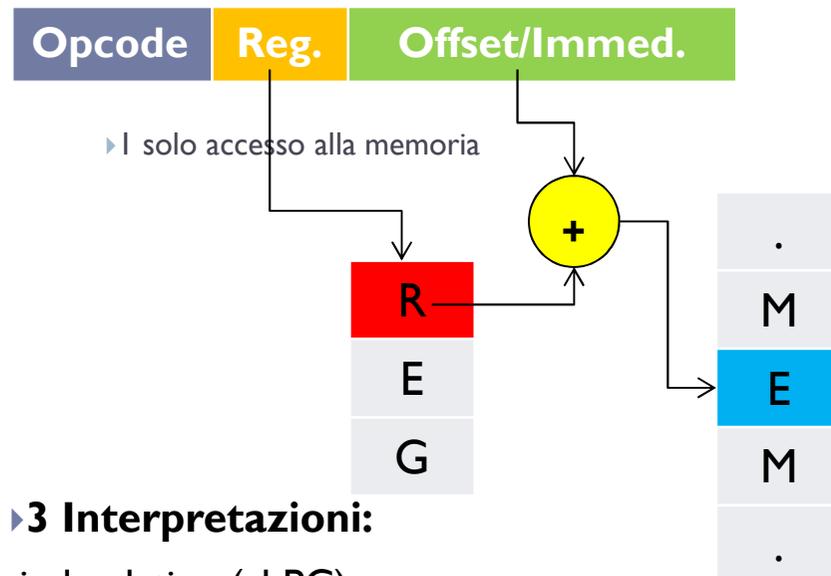
-indicizzazione di un vettore

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

▶ - Altri modi di indirizzamento:  
post indicizzato

# 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)

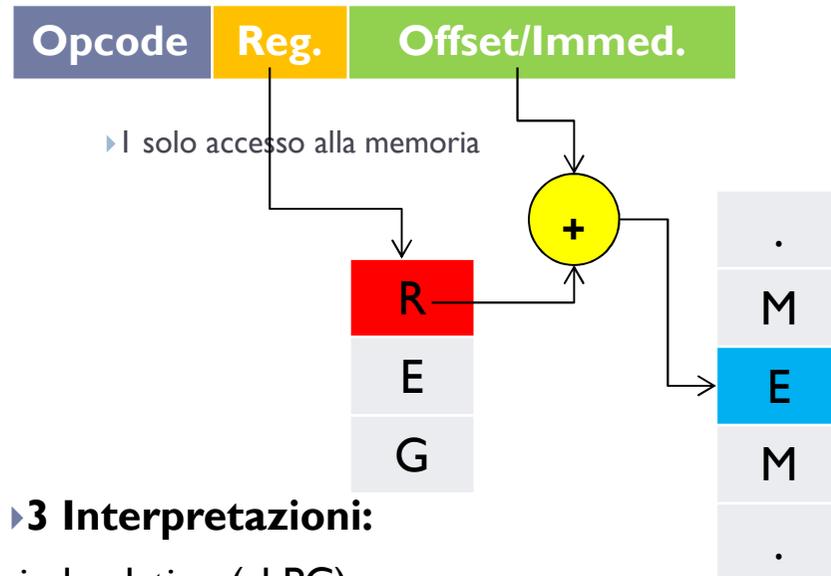
## ▶ - Altri modi di indirizzamento:

post indicizzato

subito dopo il registro viene aggiornato

# 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:

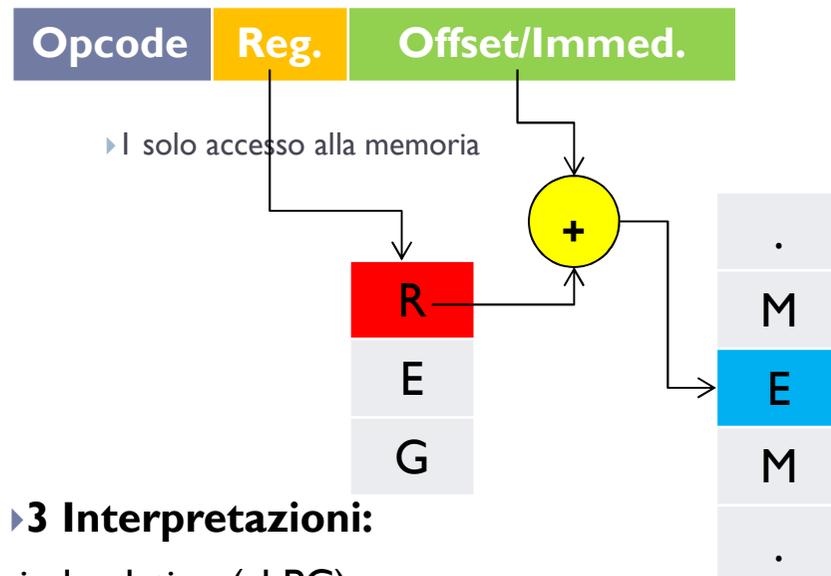
post indicizzato

subito dopo il registro viene aggiornato

pre/post incrementato

# Modi di indirizzamento complessi

## - Con spiazamento (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:

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

▶- ...

# Architetture CISC e RISC

---

## ▶Architettura CISC

### ▶(Complex Instruction Set Computer)

#### ▶- Istruzioni di **dimensione variabile**

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

## ▶Architettura RISC

### ▶(Reduced Instruction Set Computer)

#### - Istruzioni di **dimensione fissa**

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

# Architetture CISC e RISC

---

## ▶ **Architettura CISC**

### ▶ **(Complex Instruction Set Computer)**

#### ▶- Istruzioni di **dimensione variabile**

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

#### ▶- **Formato variabile**

▶ Decodifica complessa

## ▶ **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

# Architetture CISC e RISC

---

## ▶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

## ▶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

# Architetture CISC e RISC

---

## ▶ **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

## ▶ **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**

# Architetture CISC e RISC

---

## ▶ **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

## ▶ **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

# Architetture CISC e RISC

---

## ▶ **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**

# Architetture CISC e RISC

---

## ▶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**

**Istruz. complesse => molte più istruz. semplici ... MA: girano più velocemente !!!!**

---

# L'architettura MIPS 2000

---

- ▶ Word da 32 bit
- ▶ Spazio di indirizzamento da 32 bit (4GByte)
- ▶ Indirizzamento con spiazzamento
  - ▶ (che include anche gli ind. più semplici)
- ▶ Interi in Complemento a 2 su 32 bit
- ▶ 32 registri di uso generale

# L'architettura MIPS 2000

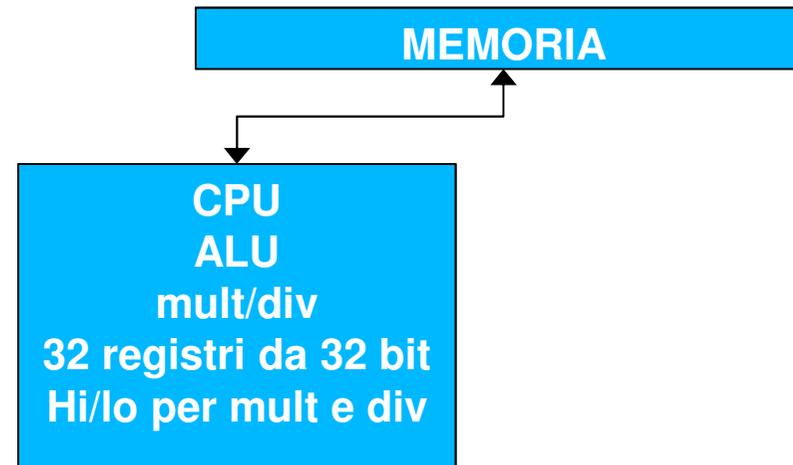
---

- ▶ Word da 32 bit
- ▶ Spazio di indirizzamento da 32 bit (4GByte)
- ▶ Indirizzamento con spiazamento
  - ▶ (che 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



# L'architettura MIPS 2000

- ▶ Word da 32 bit
- ▶ Spazio di indirizzamento da 32 bit (4GByte)
- ▶ Indirizzamento con spiazzamento
  - ▶ (che include anche gli ind. più semplici)
- ▶ Interi in Complemento a 2 su 32 bit
- ▶ 32 registri di uso generale

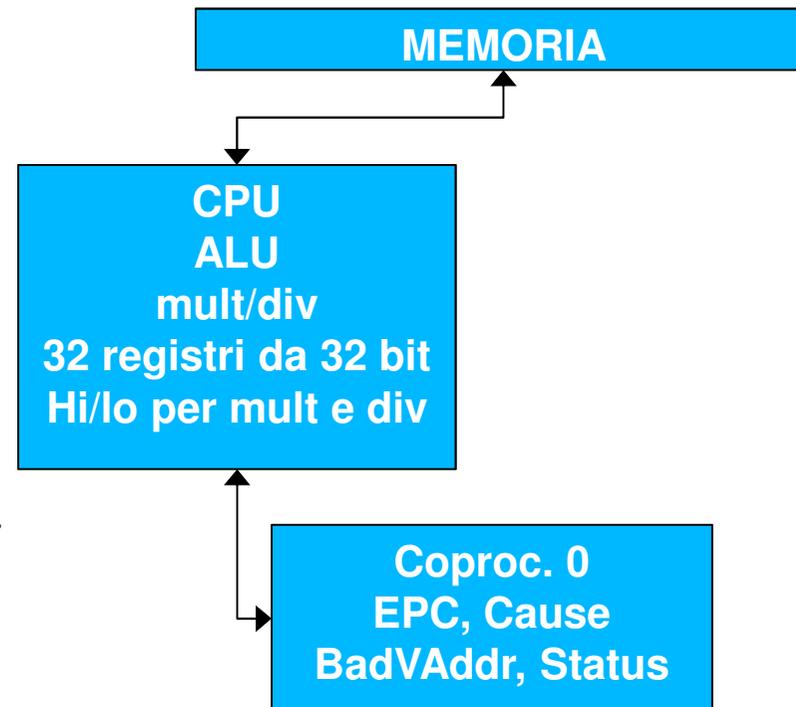
## ▶ 3 CPU

### -CPU: ALU e programma

- ▶ 32 registri + Hi/Lo usati da mult e div
- ▶ Ha accesso alla Memoria

### -Coproc. 0: traps, eccezioni, Virtual Mem.

- ▶ Cause, EPC, Status, BadVAddr



# L'architettura MIPS 2000

- ▶ Word da 32 bit
- ▶ Spazio di indirizzamento da 32 bit (4GByte)
- ▶ Indirizzamento con spiazzamento
  - ▶ (che 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

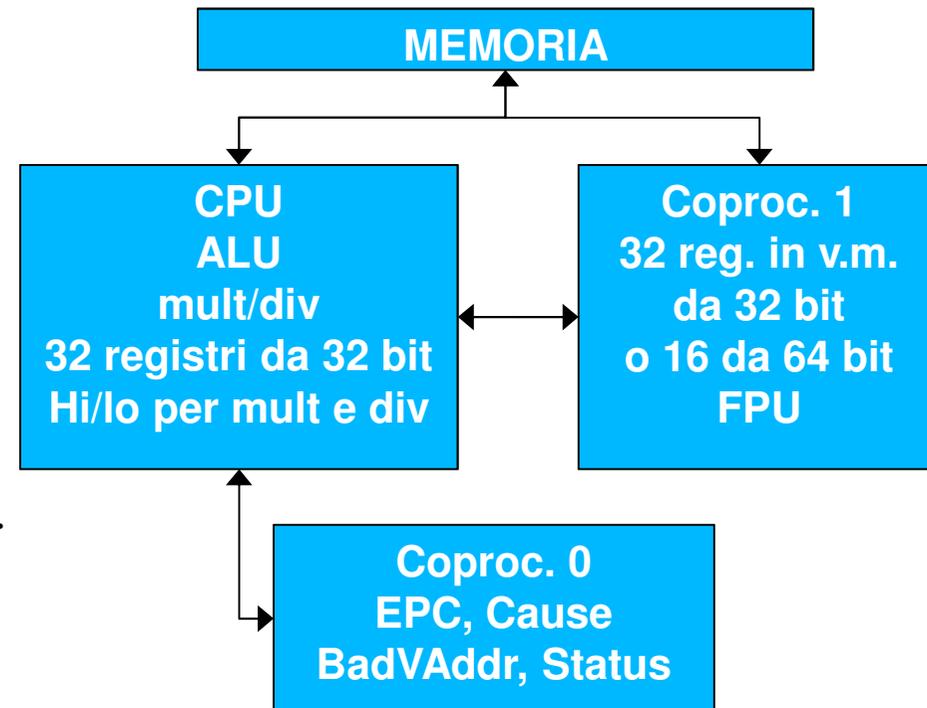
- ▶ 32 registri + Hi/Lo usati da mult e div
- ▶ Ha accesso alla Memoria

### -Coproc. 0: traps, 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
    - ▶ (che include anche gli ind. più semplici)
  - ▶ Interi in Complemento a 2 su 32 bit
  - ▶ 32 registri di uso generale
- ▶ **i 32 Registri della CPU**
  - ▶ **\$zero** la costante **zero** (immutabile)
  - ▶ **\$at** usato dalle **pseudoistruzioni**

## ▶ 3 CPU

### -CPU: ALU e programma

- ▶ 32 registri + Hi/Lo usati da mult e div
- ▶ Ha accesso alla Memoria

### -Coproc. 0: traps, eccezioni, Virtual Mem.

- ▶ Cause, EPC, Status, BadVAddr

### -Coproc. I: 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
    - ▶ (che include anche gli ind. più semplici)
  - ▶ Interi in Complemento a 2 su 32 bit
  - ▶ 32 registri di uso generale
- ▶ **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

## ▶ 3 CPU

### -CPU: ALU e programma

- ▶ 32 registri + Hi/Lo usati da mult e div
- ▶ Ha accesso alla Memoria

### -Coprocc. 0: traps, eccezioni, Virtual Mem.

- ▶ Cause, EPC, Status, BadVAddr

### -Coprocc. I: 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
  - ▶ (che 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: traps, eccezioni, Virtual Mem.**
  - ▶ Cause, EPC, Status, BadVAddr
- **Coproc. I: 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)

# L'architettura MIPS 2000

---

- ▶ Word da 32 bit

- ▶ Spazio di indirizzamento da 32 bit (4GByte)

- ▶ Indirizzamento con spiazzamento

  - ▶ (che 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:** traps, eccezioni, Virtual Mem.

  - ▶ Cause, EPC, Status, BadVAddr

- **Coproc. I:** 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)

# L'architettura MIPS 2000

---

- ▶ Word da 32 bit

- ▶ Spazio di indirizzamento da 32 bit (4GByte)

- ▶ Indirizzamento con spiazzamento

  - ▶ (che 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:** traps, eccezioni, Virtual Mem.

  - ▶ Cause, EPC, Status, BadVAddr

- **Coproc. I:** 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)

# L'architettura MIPS 2000

---

- ▶ Word da 32 bit

- ▶ Spazio di indirizzamento da 32 bit (4GByte)

- ▶ Indirizzamento con spiazzamento

  - ▶ (che 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:** traps, eccezioni, Virtual Mem.

  - ▶ Cause, EPC, Status, BadVAddr

- **Coproc. I:** 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)

# L'architettura MIPS 2000

---

- ▶ Word da 32 bit

- ▶ Spazio di indirizzamento da 32 bit (4GByte)

- ▶ Indirizzamento con spiazzamento

  - ▶ (che 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:** traps, eccezioni, Virtual Mem.

  - ▶ Cause, EPC, Status, BadVAddr

- **Coproc. I:** 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)

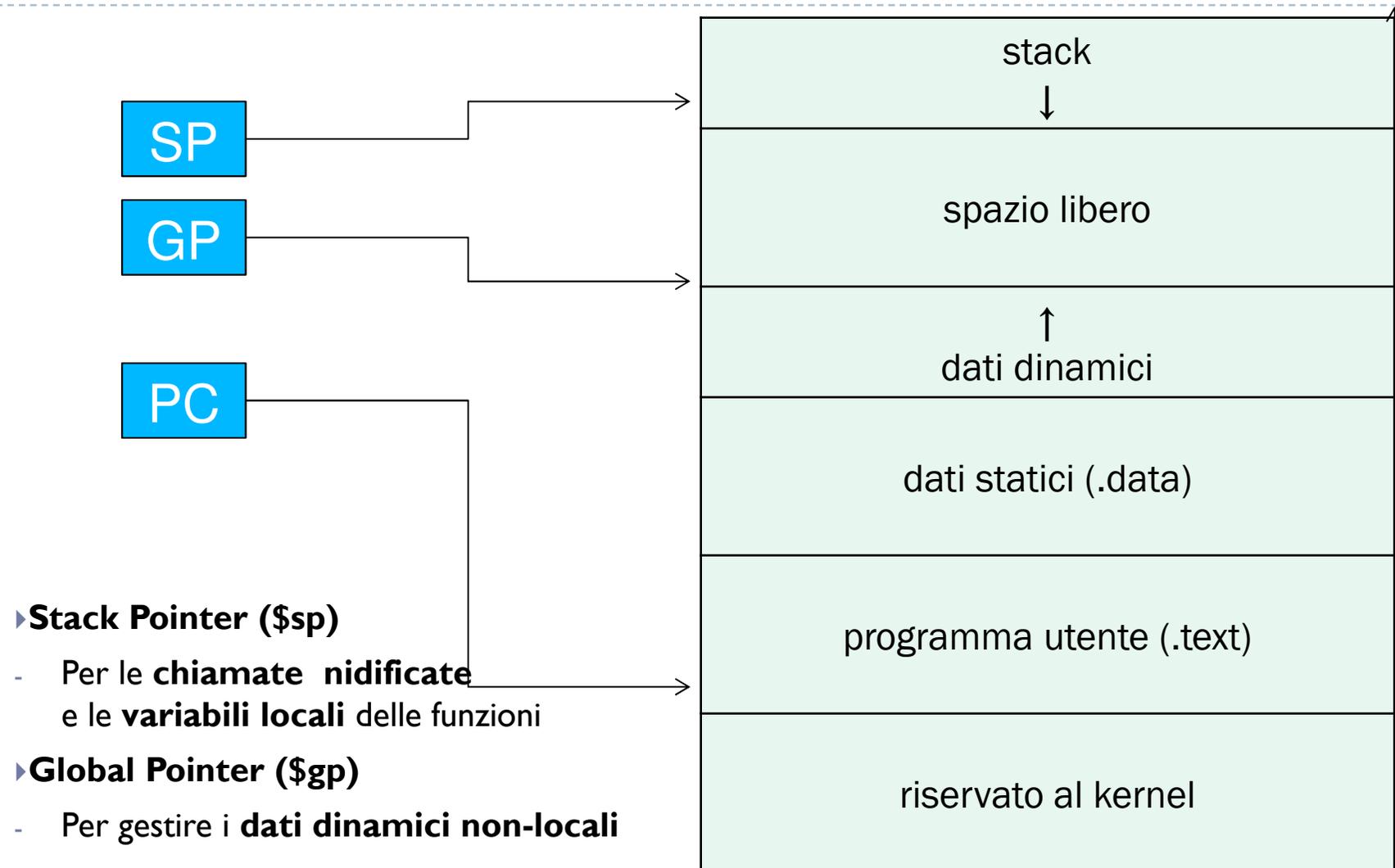
- ▶ **\$gp** **global pointer** (memoria dinamica)

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

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

- ▶ **\$ra** **return address**

# Organizzazione della memoria



# 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

▶ Esempi: **add \$t0, \$t1, \$t2**

**sll \$t0, \$t1, 5**

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

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

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

# 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

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

- ▶ Esempi: **add \$t0, \$t1, \$t2**  
**sll \$t0, \$t1, 5**

$\$t0 \leftarrow \$t1 + \$t2$   
 $\$t0 \leftarrow \$t1 \ll 5$

- ▶ **I-type:** (tipo Immediato)
  - Load/Store
  - Salti condizionati (salto relativo al PC)

Op	Rs	Rd	Immediate
6 bit	5bit	5bit	16 bit

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

$\$t1 \leftarrow \text{MEM}[\text{vettore} + \$t0]$   
 $\text{PC} \leftarrow \text{PC} + 4 + \text{offset} * 4$  **SE**  $\$t0 == \$t1$

# 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

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

- ▶ Esempi: **add \$t0, \$t1, \$t2**  
**sll \$t0, \$t1, 5**

$\$t0 \leftarrow \$t1 + \$t2$   
 $\$t0 \leftarrow \$t1 \ll 5$

- ▶ **I-type:** (tipo Immediato)
  - Load/Store
  - Salti condizionati (salto relativo al PC)

Op 6 bit	Rs 5bit	Rd 5bit	Immediate 16 bit
-------------	------------	------------	---------------------

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

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

$PC \leftarrow PC + 4 + \text{offset} * 4$     ~~SE \$t0 == \$t1~~

Op 6 bit	Address 26 bit
-------------	-------------------

- ▶ **J-type:** (tipo Jump)
  - Salti NON condizionati (salto assoluto)
- ▶ Esempio:

▶ 10      **j destinazione**

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

# Formati del Coprocessore 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

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

- ▶ Esempio:

**add.s \$f0, \$f1, \$f2**       $\$f0 \leftarrow \$f1 + \$f2$  (in singola precisione)  
**div.d \$f0, \$f2, \$f4**       $\$f0 \leftarrow \$f2 / \$f4$  (in doppia precisione)

- ▶ **FI-type:** (tipo Immediato Float)

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

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

- ▶ Esempi:

**lwc1 \$f1, indirizzo**       $\$f1 \leftarrow \text{MEM}[\text{indirizzo}]$   
**bclf cc, offset**       $\text{PC} \leftarrow \text{PC}+4 + \text{offset} * 4$  **SE** il bit **cc** è falso

# 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 \$a0, \$a1, \$a2, \$a3
- ▶ E che il risultato vada nel registro \$v0

```
move $v0, $a0  
bge $v0, $a1, checkC  
move $v0, $a1  
checkC: ←  
  
bge $v0, $a2, checkD  
move $v0, $a2  
checkD: ←  
  
bge $v0, $a3, fine  
move $v0, $a3  
fine: ←
```

# Assembly MIPS

---

## ► **Direttive** principali per l'assemblatore

<b>.data</b>	definizione dei dati statici
<b>.text</b> programma	definizione del programma
<b>.ascii</b>	stringa terminata da zero
<b>.byte</b>	sequenza di byte
<b>.double</b>	sequenza di double
<b>.float</b>	sequenza di float
<b>.half</b>	sequenza di half words
<b>.word</b>	sequenza di words

## Codici mnemonici delle **istruzioni**

add, sub, div, beq ...

## Codifica mnemonica dei **registri**

\$a0, \$sp, \$ra ... \$f0, \$f31

## **Etichette** (per calcolare gli indirizzi relativi)

nome:

## ► L'assemblatore converte

- dal testo del programma in assembly
- al programma in codice macchina

## Dalle etichette presenti **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**

# Esempio completo

---

Problema:

trovare il massimo di 4 valori  
che si trovano **in memoria**  
e memorizzare il risultato **in memoria**

**.globl main**

# Esempio completo

---

Problema:

trovare il massimo di 4 valori  
che si trovano **in memoria**  
e memorizzare il risultato **in memoria**

**.globl**    **main**

**.data**    **# definizione statica**

# Esempio completo

---

Problema:

trovare il massimo di 4 valori  
che si trovano **in memoria**  
e memorizzare il risultato **in memoria**

```
.globl    main  
.data    # definizione statica  
          # delle variabili usate  
A:      .word    42  
B:      .word    800  
C:      .word    90  
D:      .word   -56  
Risultato:      .word    0
```

# Esempio completo

---

Problema:	<b>.text</b>		
trovare il massimo di 4 valori	<b>main:</b>	lw \$a0, A	# carico A
che si trovano <b>in memoria</b>		lw \$a1, B	# carico B
e memorizzare il risultato <b>in memoria</b>		lw \$a2, C	# carico C
		lw \$a3, D	# carico D
<b>.globl</b>	<b>main</b>		
<b>.data</b>	<b># definizione statica</b>		
	<b># delle variabili usate</b>		
<b>A:</b>	.word	42	
<b>B:</b>	.word	800	
<b>C:</b>	.word	90	
<b>D:</b>	.word	-56	
<b>Risultato:</b>	.word	0	

# Esempio completo

---

Problema:

trovare il massimo di 4 valori  
che si trovano **in memoria**  
e memorizzare il risultato **in memoria**

**.text**

```
main: lw $a0, A      # carico A
        lw $a1, B      # carico B
        lw $a2, C      # carico C
        lw $a3, D      # carico D
        move $v0, $a0  # max=A
```

**.globl main**

**.data # definizione statica**

**# delle variabili usate**

**A:** .word 42

**B:** .word 800

**C:** .word 90

**D:** .word -56

**Risultato:** .word 0

# Esempio completo

---

Problema:

trovare il massimo di 4 valori  
che si trovano **in memoria**  
e memorizzare il risultato **in memoria**

```
.globl    main  
.data    # definizione statica  
          # delle variabili usate  
A:      .word    42  
B:      .word    800  
C:      .word    90  
D:      .word   -56  
Risultato: .word    0
```

**.text**

```
main:    lw $a0,A      # carico A  
          lw $a1,B      # carico B  
          lw $a2,C      # carico C  
          lw $a3,D      # carico D  
          move $v0,$a0   # max=A  
          bge $v0,$a1,checkC # se max>=B  
          move $v0,$a1   # max=B
```

# Esempio completo

---

Problema:

trovare il massimo di 4 valori  
che si trovano **in memoria**  
e memorizzare il risultato **in memoria**

```
.globl   main  
.data   # definizione statica  
         # delle variabili usate  
A:     .word   42  
B:     .word   800  
C:     .word   90  
D:     .word  -56  
Risultato: .word  0
```

**.text**

```
main:   lw $a0,A           # carico A  
         lw $a1,B           # carico B  
         lw $a2,C           # carico C  
         lw $a3,D           # carico D  
         move $v0,$a0       # max=A  
         bge $v0,$a1,checkC # se max>=B  
         move $v0,$a1       # max=B  
checkC: bge $v0,$a2,checkD # se max>=C  
         move $v0,$a2       #max=C
```

# Esempio completo

---

Problema:

trovare il massimo di 4 valori  
che si trovano **in memoria**  
e memorizzare il risultato **in memoria**

```
.globl    main
.data    # definizione statica
           # delle variabili usate
A:      .word    42
B:      .word    800
C:      .word    90
D:      .word   -56
Risultato: .word    0
```

**.text**

```
main:    lw $a0,A           # carico A
           lw $a1,B           # carico B
           lw $a2,C           # carico C
           lw $a3,D           # carico D
           move $v0,$a0       # max=A
           bge $v0,$a1,checkC  # se max>=B
           move $v0,$a1       # max=B
checkC: bge $v0,$a2,checkD  # se max>=C
           move $v0,$a2       #max=C
checkD: bge $v0,$a3,fin     # se max >= D
           move $v0,$a3       #max=D
```

# Esempio completo

---

Problema:

trovare il massimo di 4 valori  
che si trovano **in memoria**  
e memorizzare il risultato **in memoria**

```
.globl    main
.data    # definizione statica
           # delle variabili usate
A:      .word    42
B:      .word    800
C:      .word    90
D:      .word   -56
Risultato: .word    0
```

**.text**

```
main:    lw $a0,A           # carico A
           lw $a1,B           # carico B
           lw $a2,C           # carico C
           lw $a3,D           # carico D
           move $v0,$a0       # max=A
           bge $v0,$a1,checkC  # se max>=B
           move $v0,$a1       # max=B
checkC: bge $v0,$a2,checkD  # se max>=C
           move $v0,$a2       #max=C
checkD: bge $v0,$a3,fine    # se max >= D
           move $v0,$a3       #max=D
fine:   sw $v0,Risultato   # mem. il risult.
           li $v0,10          # fine
           syscall           # del prog.
```