# MIPS

## Registri

- 32 registri di 32 bit (registro 0 contiene sempre il valore 0)
- Architettura Load / Store
  - Istruzioni di trasferimento per muovere i dati tra memoria e registri
  - Istruzioni per la manipolazione di dati operano sui valori dei registri
  - Nessuna operazione memoria ↔ memoria
- Quindi: le istruzioni operano su registri
  (registro i riferito con $i)
- Esempio: add $0, $1, $2



# MIPS

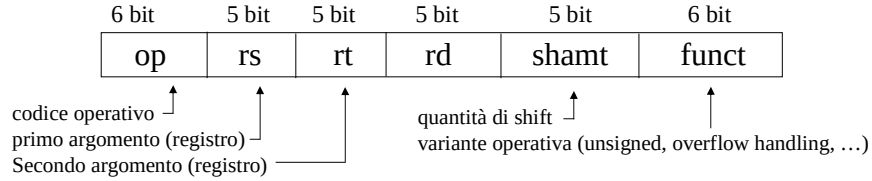
## Dati e modi di indirizzamento

- Registri possono essere caricati con byte, mezze parole, e parole (riempiendo con 0 quando necessario o estendendo, cioè replicando, il segno sui bit non coinvolti del registro)
- Modalità di indirizzamento ammesse (con campi di 16 bit):
  - Immediata es. add $2, $2, 0004
  - Displacement es. sw $1, 000c($1)
- Altre modalità derivabili:
  - Indiretta registro (displacement a 0) es. sw $2, 0000($3)
  - Assoluta (registro 0 come registro base) es. lw $1, 00c4($0)

# MIPS

## Formato Istruzioni

- 32 bit per tutte, 3 formati diversi (formato R, formato I, formato J)
- **Formato R (registro)**




Es: **`0x02484020`** ( **`add $8,$18,$8`** `[R8] ← [R18] + [R8]` )

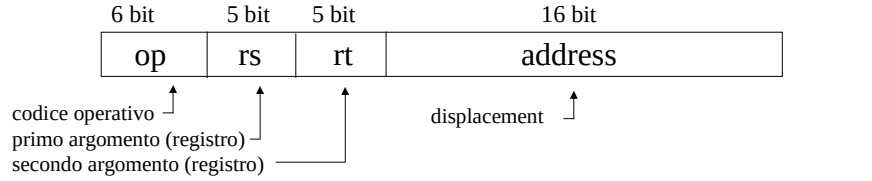
| op | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| 000000 | 10010 | 01000 | 01000 | 00000 | 100000 |



# MIPS

## Formato Istruzioni

- **Formato I (istruzioni load / store)**



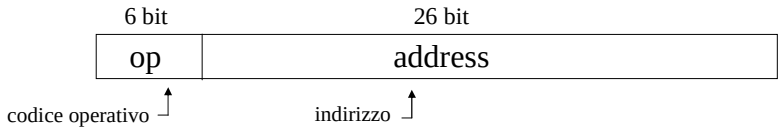

Es: **`0x8D2804B0`** ( **`lw $8, 1200($9)`** `[R8] ← Mem[1200+[R9]]` )

| op | rs | rt | address |
|---|---|---|---|
| 100011 | 01001 | 01000 | 0000 0100 1011 0000 |

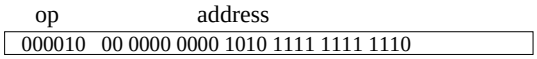
# MIPS

## Formato Istruzioni

- **Formato J (istruzioni jump)**




Es: **`0x0800AFFE`** ( **`j 45054`** `[PC] ← 45054` )






# Fasi (MIPS)

Fasi senza pipeline:

**IF (istruction fetch):**

- IR ← Mem[PC] ;
- NPC ← PC + 4 ;

Dove NPC è un registro temporaneo

PC è il program counter

# Fasi (MIPS)

**ID (instruction decode/register fetch cycle):**

- A ← Regs[rs] ;
- B ← Regs[rt] ;
- Imm ← campo immediato di IR con segno esteso ;

Dove A, B, Imm sono registri temporanei



# Fasi (MIPS)

**EX (execution/effective address cycle):**

1. ***Riferimento a memoria***
   - ALUOutput ← A + Imm ;
2. ***Istruzione ALU registro-registro***
   - ALUOutput ← A ***func*** B ;
3. ***Istruzione ALU registro-immediato***
   - ALUOutput ← A ***op*** Imm ;
4. ***Salto***
   - ALUOutput ← NPC + (Imm << 2);
   - Cond ← (A = = 0 ) ;

shift a sinistra

# Fasi (MIPS)

## MEM (memory access/branch completion cycle):

- PC ← NPC ;  in tutti i casi

1. ***Riferimento a memoria***
   - LMD ← Mem[ALUOutput] or
     Mem[ALUOutput] ← B;
2. ***Salto***
   - if (Cond)  PC ← ALUOutput ;

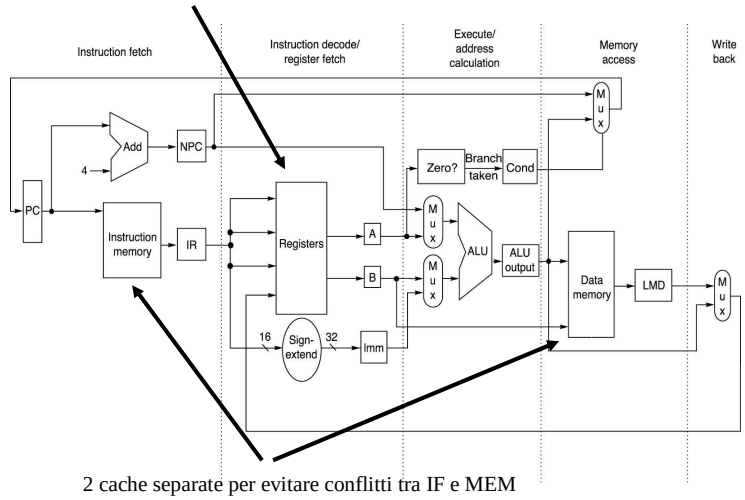


# Fasi (MIPS)

## WB (write/back cycle):

1. ***Istruzione ALU registro-registro***
   - Regs[rd] ← ALUOutput ;
2. ***Istruzione ALU registro-immediato***
   - Regs[rt] ← ALUOutput ;
3. ***Istruzione Load***
   - Regs[rt] ← LMD ;

registri letti (anche 2 volte) e scritti nello stesso ciclo di clock per evitare conflitti fra ID e WB




2 cache separate per evitare conflitti tra IF e MEM

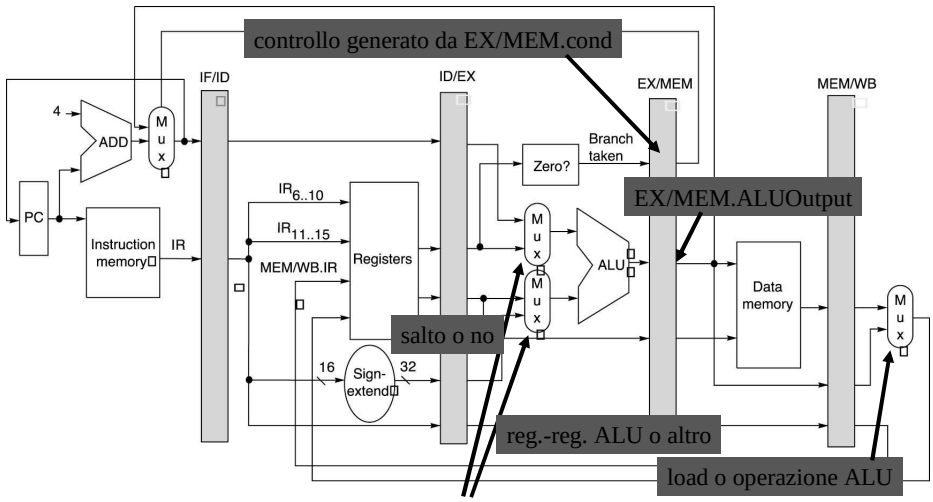




# Pipeline (MIPS)

- Architettura che si presta ad una facile introduzione della pipeline: uno stadio per fase, 1 ciclo di clock per stadio
- Occorre memorizzare i dati fra una fase e la successiva: si introducono opportuni registri (denominati pipeline registers o pipeline latches) fra i vari stadi della pipeline
- Tali registri memorizzano sia dati che segnali di controllo che devono transitare da uno stadio al successivo
- Dati che servono a stadi non immediatamente successivi vengono comunque copiati nei registri dello stato successivo per garantire la correttezza dei dati

# Pipeline (MIPS)




**Settati a seconda del tipo di istruzione (codificato nel campo ID/EX.IR)**