famiglia MIPS

- Studiamo i processori MIPS come esempio di architettura RISC
- architettura sperimentale sviluppata a Stanford negli anni '80 e poi sviluppata commercialmente
- Riferimenti: *Hennessy & Patterson "Struttura e progetto dei calcolatori",* in Biblioteca

- Architettura molto regolare
- architettura progettata per un'implementazione efficiente della pipline
  - MIPS = ***m****icroprocessor without* ***i****nterlocked* ***p****ipeline* ***s****tages*

MIPS (a 32 bit)

Istruzioni:

- Tutte le istruzioni di dimensione 32 bit
- tutte le operazioni sui dati sono da registro a registro
  - le istruzioni che manipolano i dati si usano i valori dei registri
- le operazioni sulla memoria:
  - solo *load* e *store*, per trasferire dati tra memoria e registri
  - nessuna operazione memoria-memoria
- quindi **tutte le istruzioni operano su registri**, es `add $1, $2, $3`

Registri:

- 32 registri di 32 bit
- si indicano con $1, $2, $3…. **$0** contiene sempre 0

MIPS

Dati:

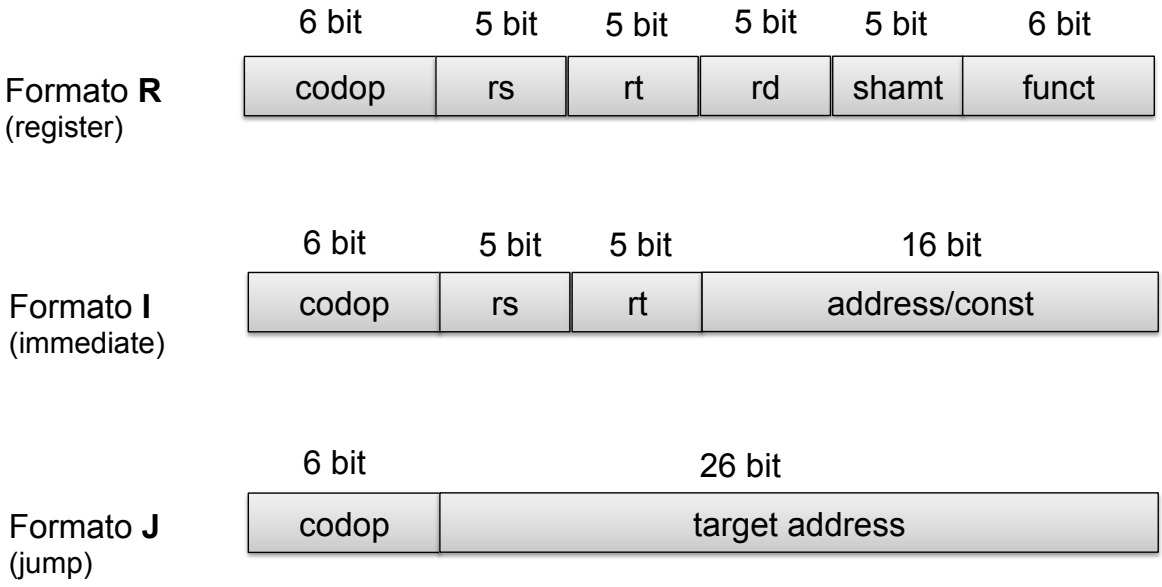
- Registri possono essere caricati con byte, mezze parole, e parole
- i registri sono a 32 bit, quindi il dato può essere “allungato” riempiendo i bit rimanenti con 0 o estendendo il segno (cioè replicandolo)

Modi di indirizzamento:

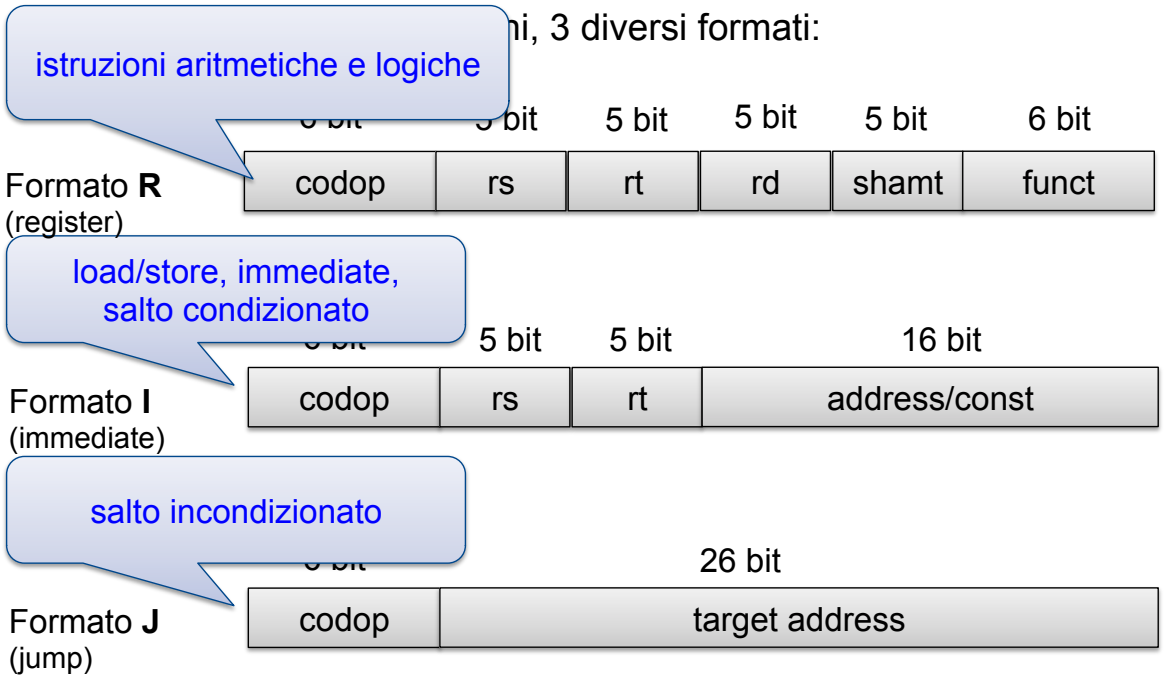
- Immediata es. `addi $2, $2, 0004` ← 16 bit
- Displacement es. `sw $1, 000c($1)`
- Altre modalità derivabili:
  - Indiretta registro (displacement a 0) es. `sw $2, 0000($3)`
  - Assoluta (registro 0 come registro base) es. `lw $1, 00c4($0)`

MIPS – Formato Istruzioni

- 32 bit per tutte le istruzioni, 3 diversi formati:

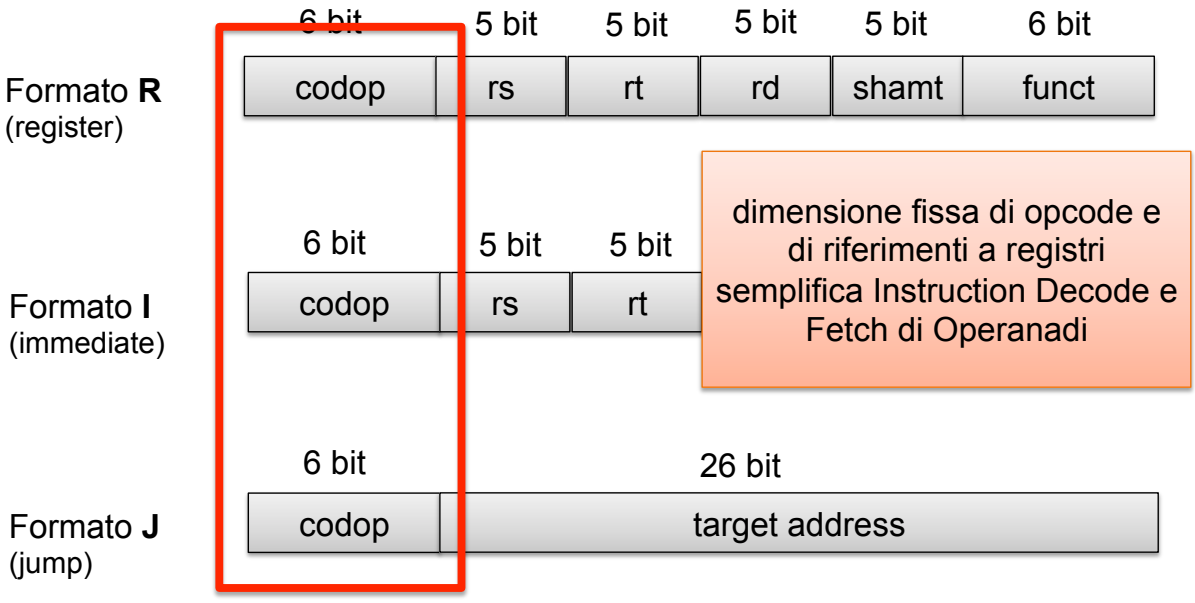


MIPS – Formato Istruzioni



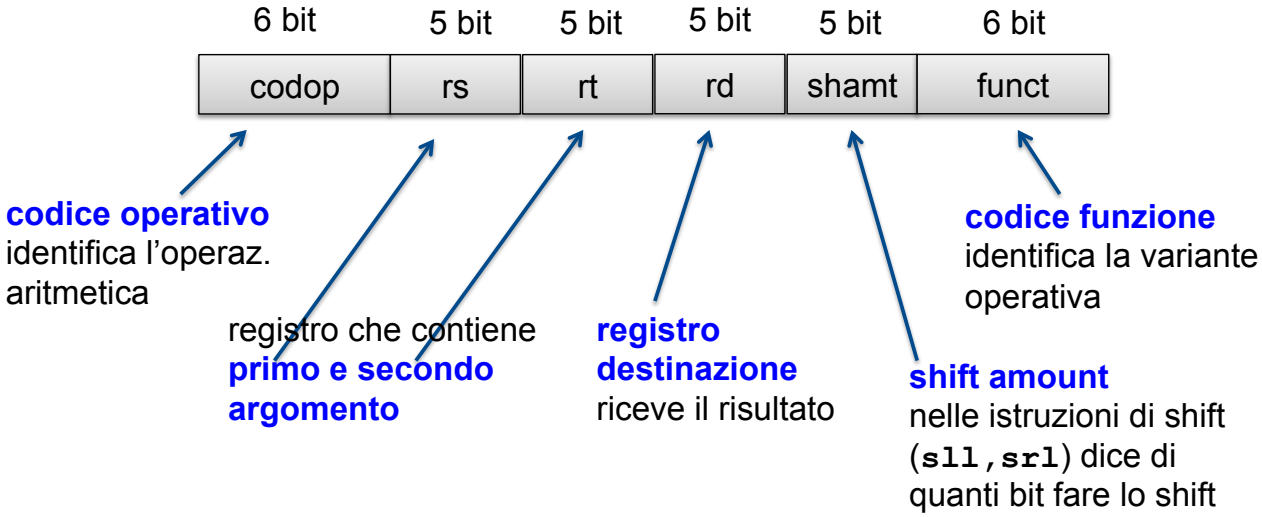
MIPS – Formato Istruzioni

- 32 bit per tutte le istruzioni, 3 diversi formati:

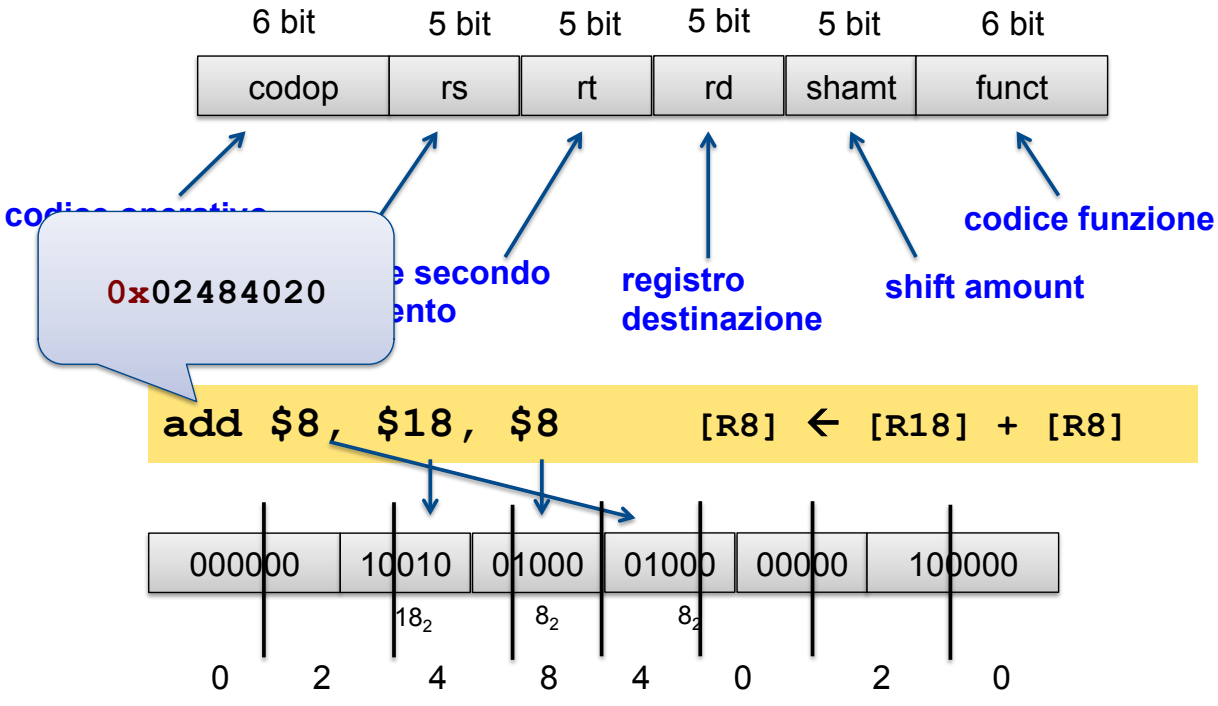


Istruzioni – Formato R

istruzioni aritmetiche e logiche. 6 campi a dimensione fissa

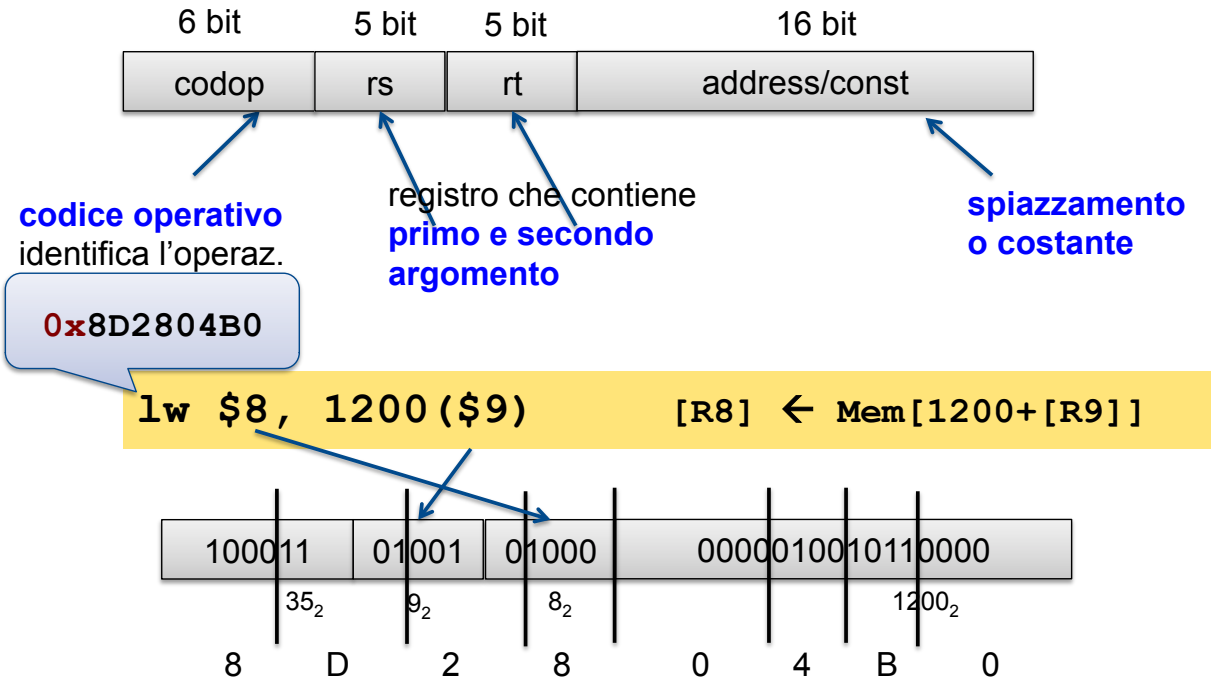


Istruzioni – Formato R



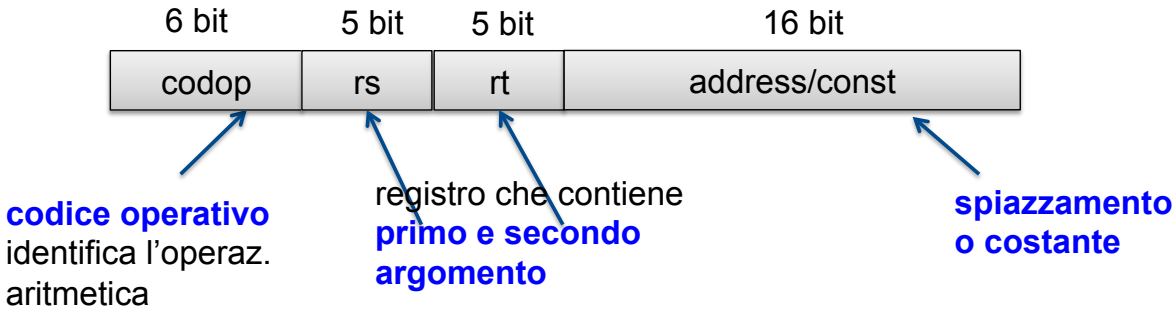
Istruzioni – Formato I

istruzioni load/store, immediate e salto condizionato



Istruzioni – Formato I

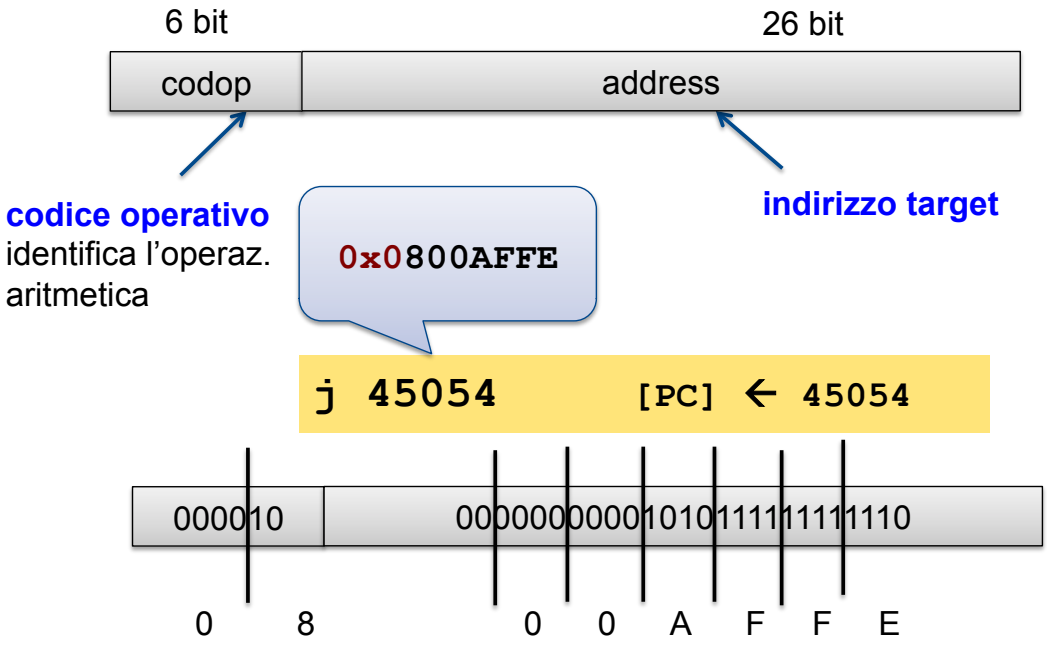
istruzioni load/store, immediate e salto condizionato




```
lw $8, 1200($9)        [R8] ← Mem[1200+[R9]]
sw $8, 1200($9)        Mem[1200+[R9]] ← [R8]
beq $1, $2, 16         if ($1==$2) PC=PC+16
addi $1, $8, 4         [R1] ← [R8]+4
```

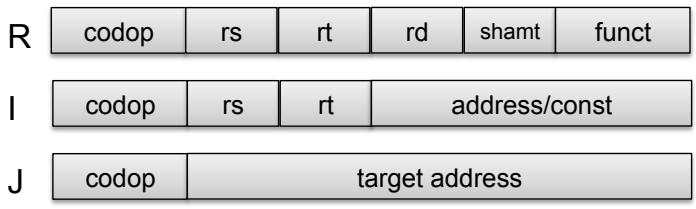
Istruzioni – Formato J

istruzioni salto incondizionato



ciclo esecutivo di un'istruzione MIPS

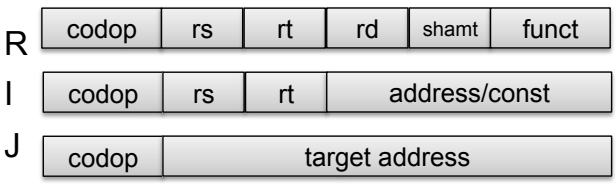
1. **IF** Instruction Fetch
2. **ID** Instruction Decode / register fetch
3. **EX** Execution / address calculation
   - tutte le istruzioni usano la ALU (tranne salto incondizionato):
   - operazioni logico-aritmentiche
   - load/store e jump per calcolare l'indirizzo
   - salti condizionati per calcolare la condizione
4. **MEM** Memory access / branch completion
5. **WB** Write Back: scrittura del risultato nei registri



ciclo esecutivo di un'istruzione MIPS

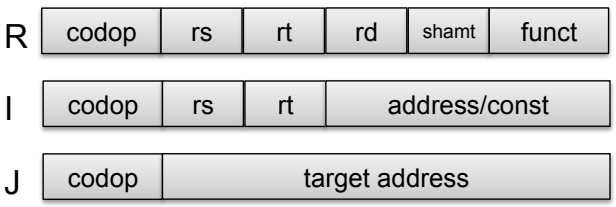
1. **IF** Instruction Fetch
   - IR ← Mem[PC] *(IR = instruction register, PC = program counter)*
   - NPC ← PC+4 *(NPC è un registro temporaneo)*

2. **ID** Instruction Decode / register fetch
   - **Formato R** A ← Regs[rs]; B ← Regs[rt]
   - **Formato I** A ← Regs[rs]; B ← Regs[rt] ; Imm ← campo immediato di IR
   - **Formato J** Imm ← campo immediato di IR
   - il campo immediato di IR va esteso a 32 bit, estendendo il segno
   - A, B, Imm registri temporanei



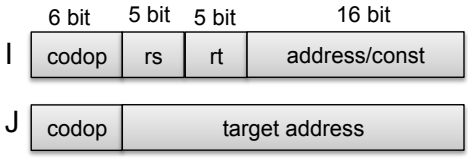
ciclo esecutivo di un'istruzione MIPS

3. **EX** Execute / address calculation
   - **Riferimento a memoria**
     - `lw $8, 1200($9)` Formato I A=R[rs]=[$9] Imm=1200*(esteso)*
     - ALUOutput ← A + Imm //address calculation
   - **Istruzione ALU registro-registro**
     - `add $8, $18, $8` Formato R A=R[rs]=[$18] B=R[rt]=[$8]
     - ALUOutput ← A ***funct*** B
   - **Istruzione ALU registro-immediato**
     - `addi $8, $18,4` Formato I A=R[rs]=[$18] Imm=4*(esteso)*
     - ALUOputpu ← A ***op*** Imm
   - **Salto**
     - `j 45054` Formato J
     - `beq $1, $2, 16` Formato I



ciclo esecutivo di un'istruzione MIPS

3. **EX** Execute / address calculation
   - **Salto incondizionato**
     - `j 45054` Formato J Imm = 45054
     - l'indirizzo nell'istruzione si riferisce al numero di **parole** di cui far avanzare il PC (NPC), ma gli indirizzi MIPS sono al **byte**
     - quindi il numero di parole va trasformato in numero di byte: moltilplicando per 4 cioè **due shift a sinistra <<2**
     - Target ← NPC + (Imm << 2)
   - **Salto condizionato**
     - `beq $1, $2, 16` Formato I A=R[rs]=[$1] B=R[rt]=[$2] Imm=16
     - deve valutare la condizione e calcolare l'indirizzo di salto
     - Cond ← (A-B)==0 la ALU fornisce in uscita un segnale che indica se il risultato è 0
     - Target ← NPC + (Imm <<2)

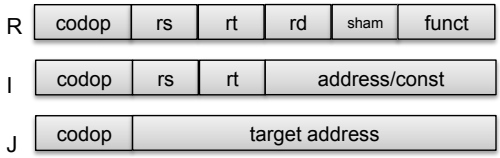


ciclo esecutivo di un'istruzione MIPS

4. **MEM** Memory access / branch completion
   - PC ← NPC sempre
   - **Riferimento a memoria**
     - `lw $8, 1200($9)` A=[$9] B=[$8] Imm=1200 ALUOutput=A+Imm
     - LMD ← Mem[ALUOutput] (Load Memory Data register)
     - `sw $8, 1200($9)` A=[$9] B=[$8] Imm=1200 ALUOutput=A+Imm
     - Mem[ALUOutput] ← B
   - **Salto**
     - **incondizionato** PC ← Target
     - **condizionato** if (Cond) PC ← Target

ciclo esecutivo di un'istruzione MIPS

5. **WB** Write back: scrittura nei registri
   - **Riferimento a memoria (solo load)**
     - `lw $8, 1200($9)` A=$9 B=Reg[rt]=$8 LMD ← Mem[ALUOutput]
     - Regs[rt] ← LMD
   - **Istruzione ALU registro-registro**
     - `add $8, $18, $8` Formato R A=R[rs]=$18 B=R[rt]=$8
     - Regs[rd] ← ALUOutput
   - **Istruzione ALU registro-immediato**
     - `addi $8, $18,4` Formato I A=R[rs]=$18 B=R[rt] Imm=4
     - Regs[rt] ← ALUOutput



ciclo esecutivo di un'istruzione MIPS

`add $8, $18, $8`

R

IF
- IR ← Mem[PC]
- NPC ← PC+4

ID
- A ← Regs[rs] = [$18]
- B ← Regs[rt] = [$8]

EX
- ALUOutput ← A funct B

MEM
- PC ← NPC

WB
- Regs[rd]=$8 ← ALUOutput

ciclo esecutivo di un'istruzione MIPS

`lw $8, 1200($9)`

IF
- IR ← Mem[PC]
- NPC ← PC+4

ID
- A ← Regs[rs] = [$9] B ← Regs[rt]=[$8]
- Imm ← campo immediato di IR = 1200 *(esteso)*

EX
- ALUOutput ← A + Imm

MEM
- PC ← NPC
- LMD ← Mem[ALUOutput]

WB
- Regs[rt]=$8 ← LMD

ciclo esecutivo di un'istruzione MIPS

`sw $8, 1200($9)`

IF
- IR ← Mem[PC]
- NPC ← PC+4

ID
- A ← Regs[rs] = [$9] B ← Regs[rt]=[$8]
- Imm ← campo immediato di IR = 1200 *(esteso)*

EX
- ALUOutput ← A + Imm

MEM
- PC ← NPC
- Mem[ALUOutput] ← B

WB
- nessuna operazione

ciclo esecutivo di un'istruzione MIPS

`beq $1,$2,16` I

IF
- IR ← Mem[PC]
- NPC ← PC+4

ID
- A ← Regs[rs] = [$1] B ← Regs[rt]=[$2]
- Imm ← campo immediato di IR = 16 *(esteso)*

EX
- Cond ← (A-B)==0
- Target ← NPC + (Imm <<2)

MEM
- if (Cond) PC ← Target

WB
- nessuna operazione

ciclo esecutivo di un'istruzione MIPS

`j 45054` J

IF
- IR ← Mem[PC]
- NPC ← PC+4

ID
- Imm ← campo immediato di IR = 45054 *(esteso)*

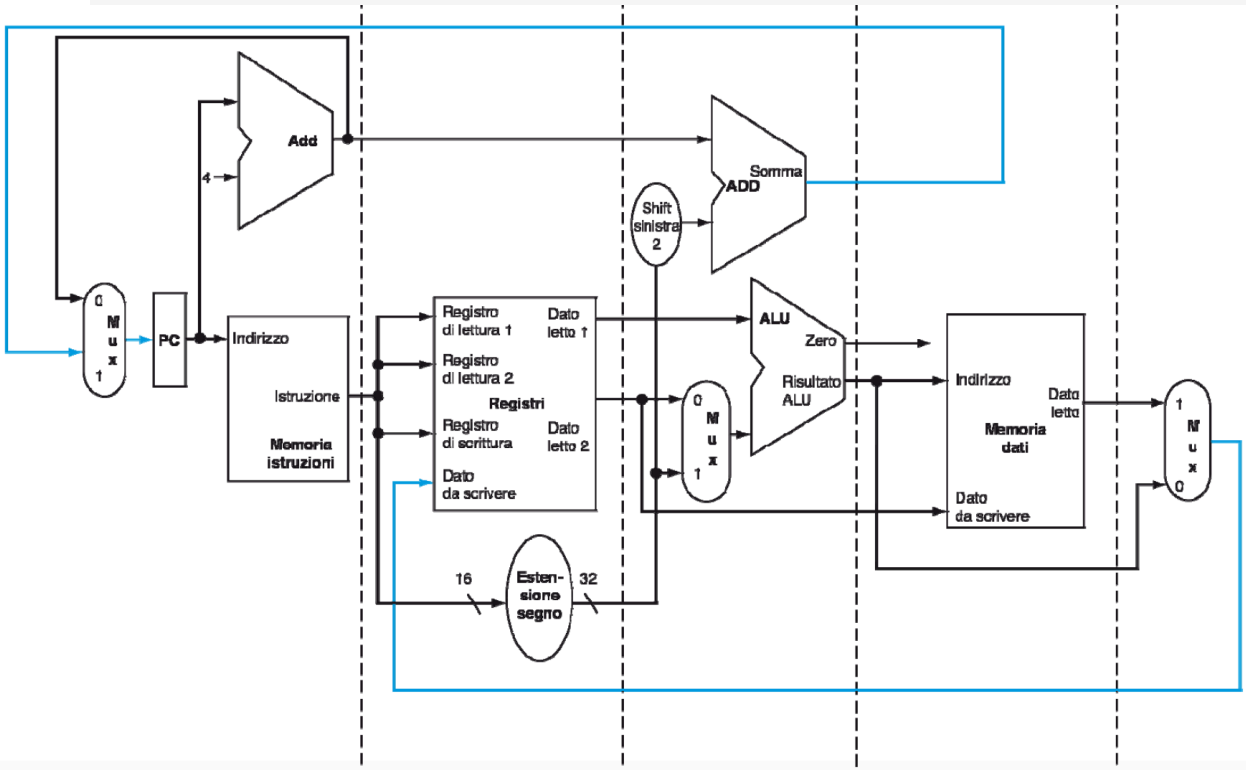
EX
- Target ← NPC + (Imm <<2)

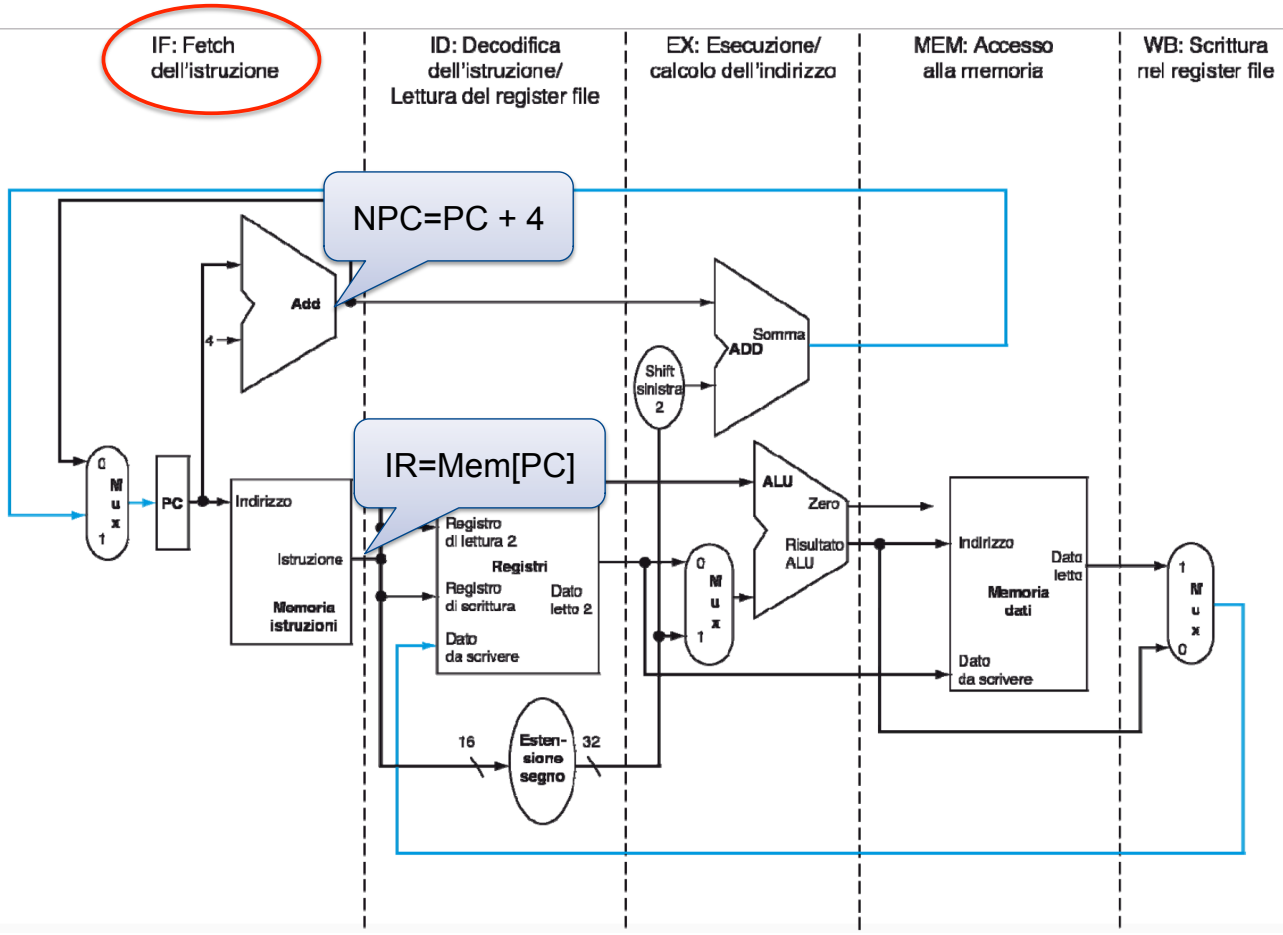
MEM
- PC ← Target

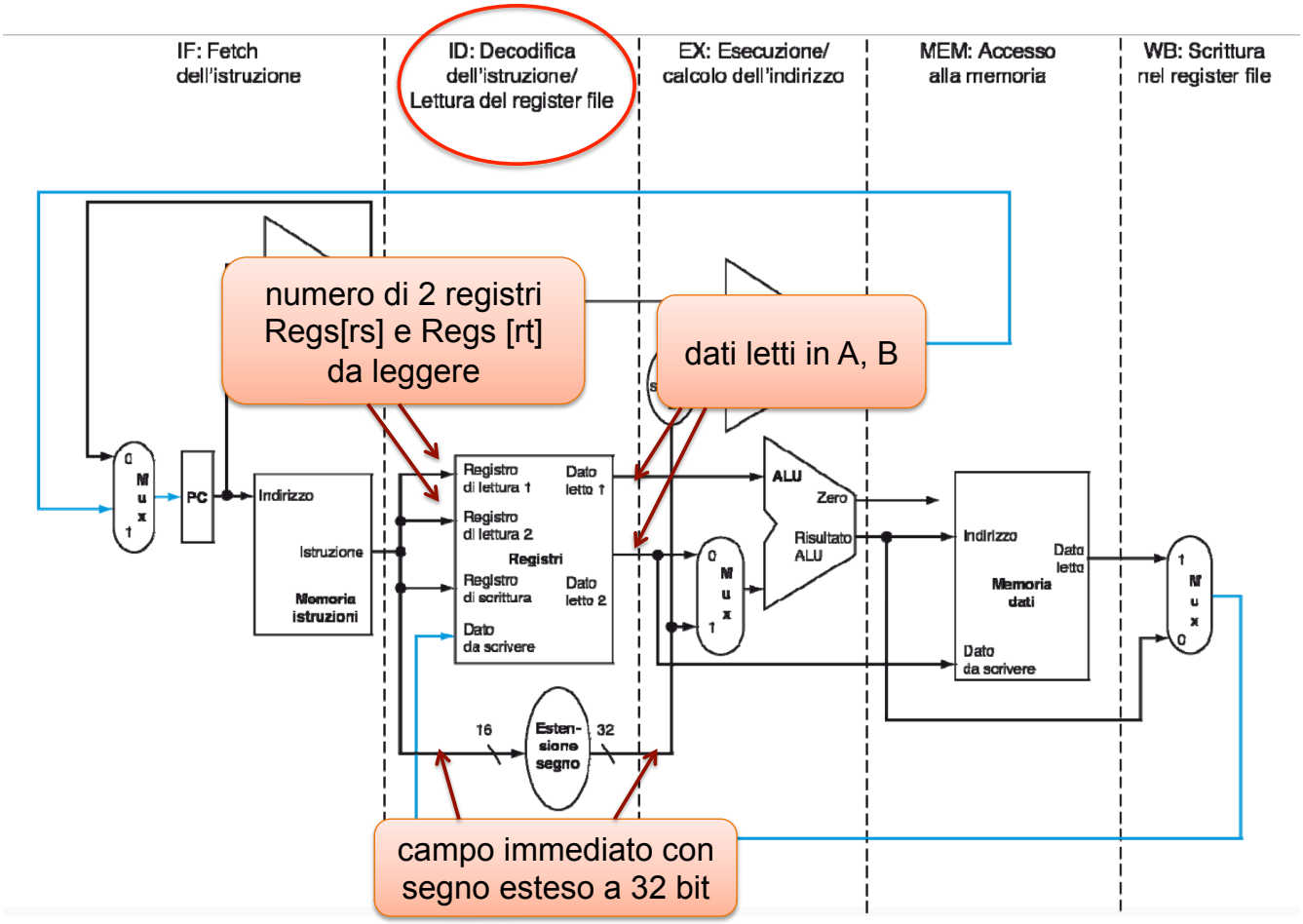
WB
- nessuna operazione

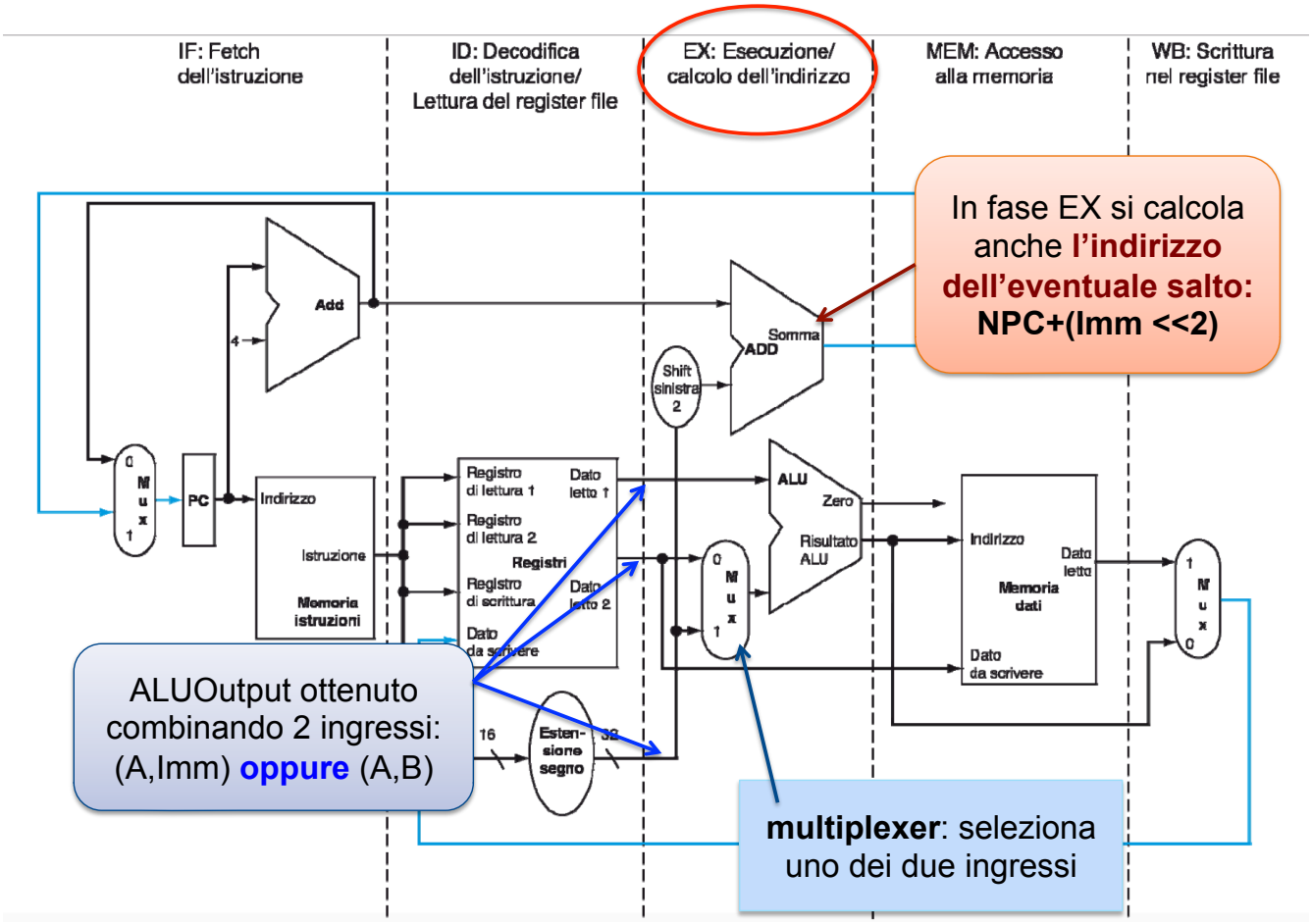
schema di implementazione di MIPS:

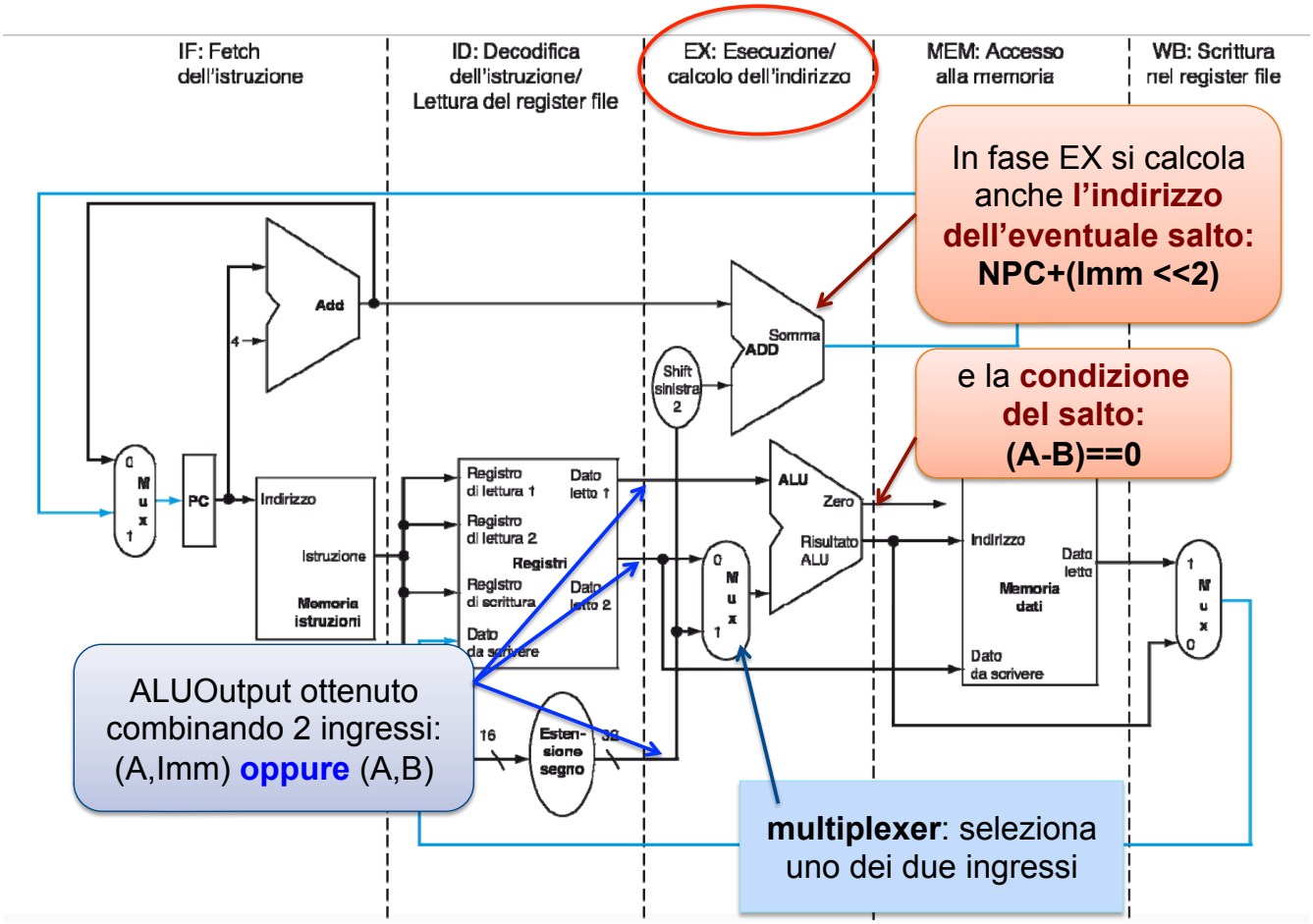
- diverse unità funzionali (es. banco di registri, ALU, memoria...)
- loro connessioni
- manca unità di controllo e linee di controllo

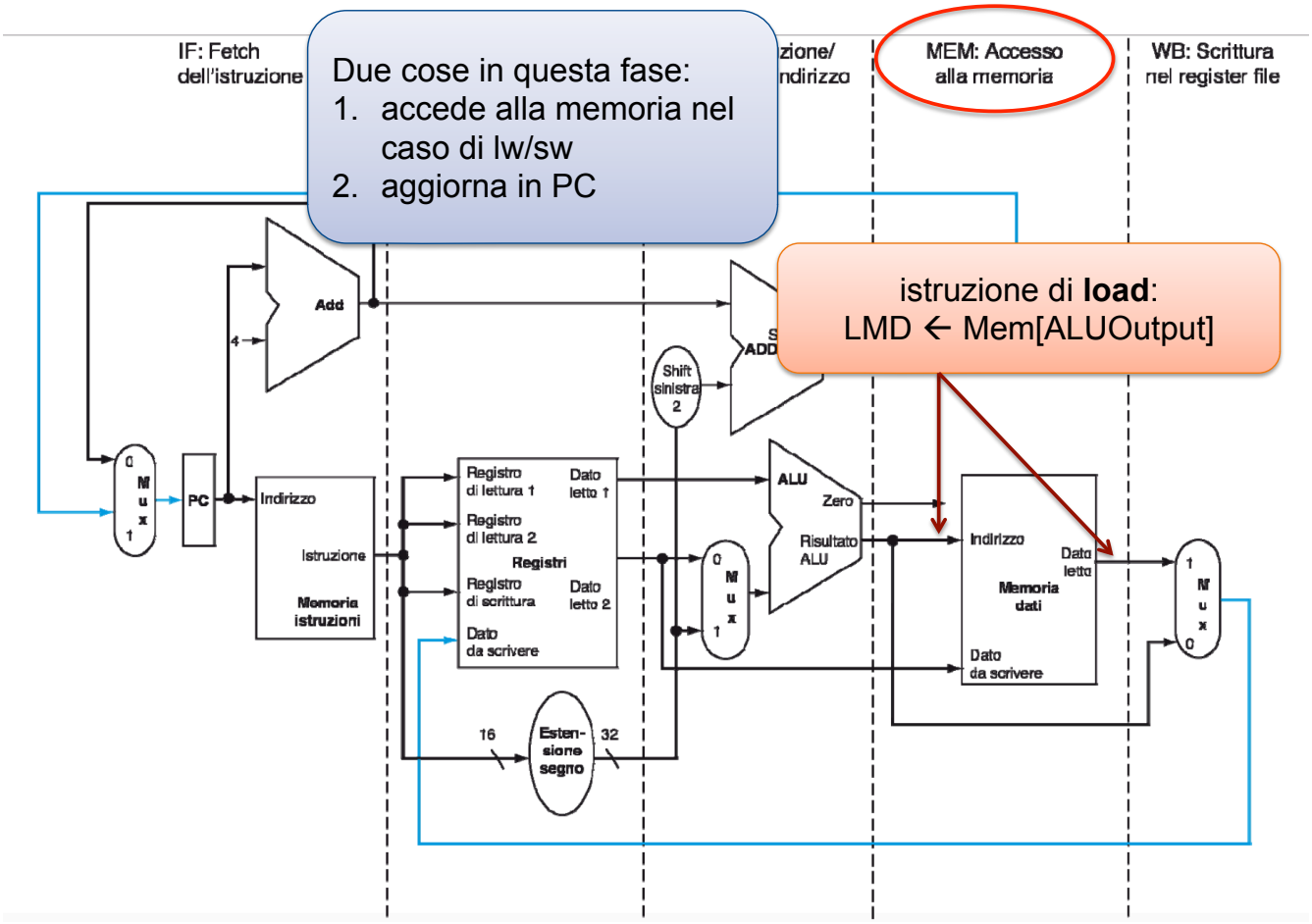


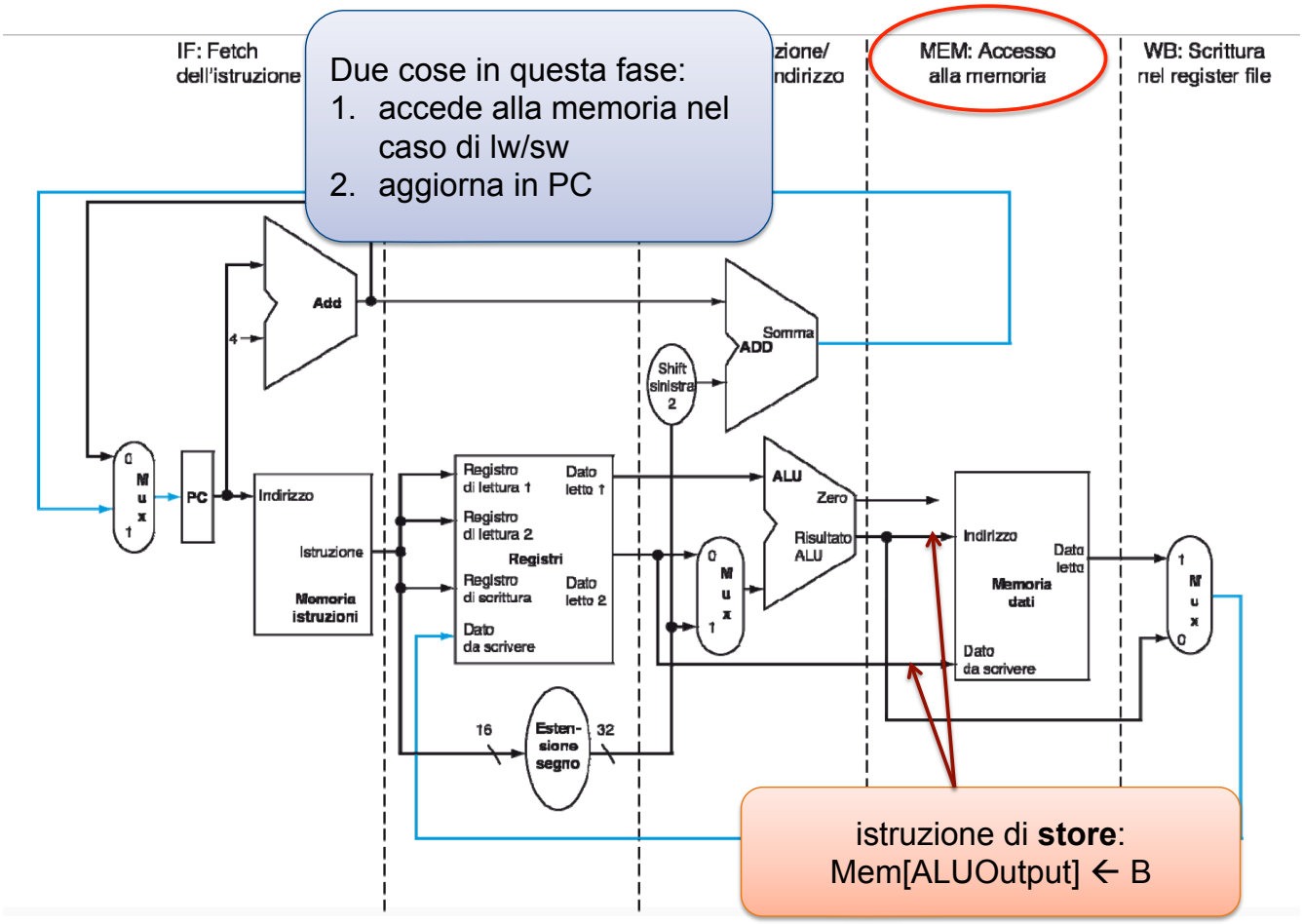


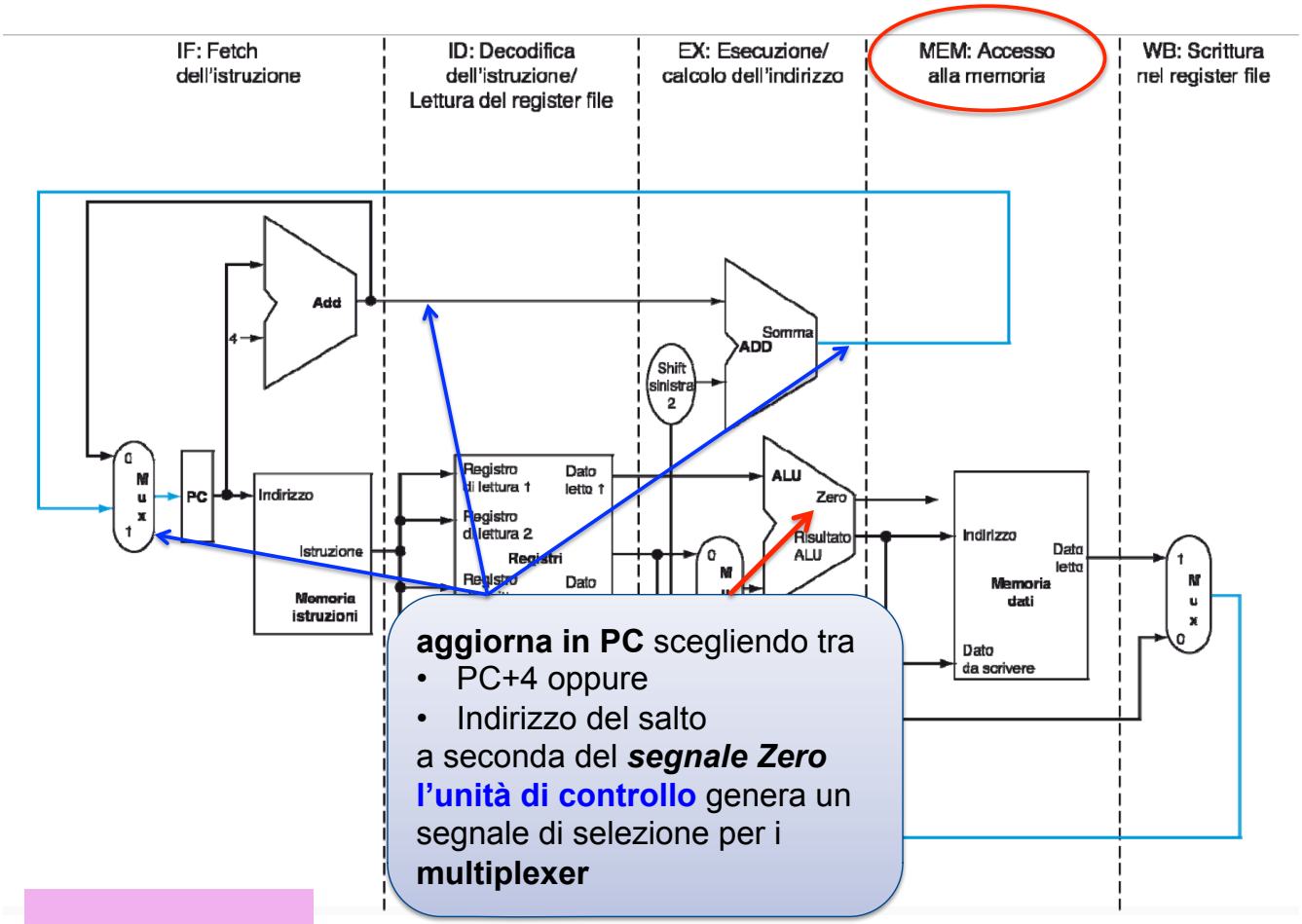

IF: Fetch dell'istruzione
ID: Decodifica dell'istruzione/ Lettura del register file
EX: Esecuzione/ calcolo dell'indirizzo
MEM: Accesso alla memoria
WB: Scrittura nel register file
numero di 2 registri Regs[rs] e Regs [rt] da leggere
dati letti in A, B
campo immediato con segno esteso a 32 bit
M u x
PC
Indirizzo
Istruzione
Memoria istruzioni
Registro di lettura 1
Registro di lettura 2
Registri
Registro di scrittura
Dato da scrivere
Dato letto 1
Dato letto 2
ALU
Zero
Risultato ALU
Indirizzo
Dato letto
Memoria dati
Dato da scrivere
16
Estensione segno
32

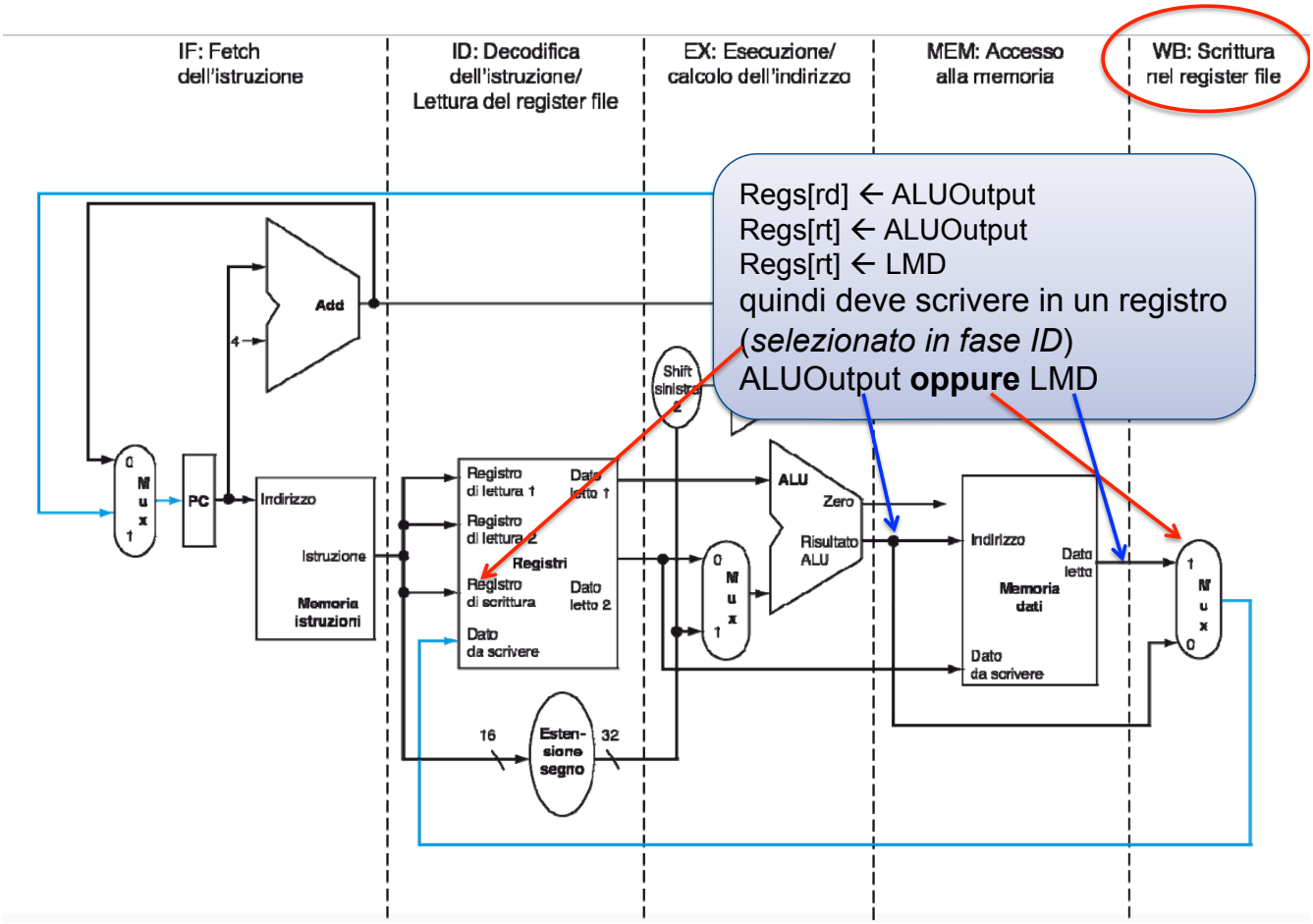

IF: Fetch dell'istruzione
ID: Decodifica dell'istruzione/ Lettura del register file
EX: Esecuzione/ calcolo dell'indirizzo
MEM: Accesso alla memoria
WB: Scrittura nel register file
In fase EX si calcola anche l'indirizzo dell'eventuale salto: NPC+(Imm <<2)
ALUOutput ottenuto combinando 2 ingressi: (A,Imm) oppure (A,B)
multiplexer: seleziona uno dei due ingressi
Add
4
Somma ADD
Shift sinistra 2
M u x
PC
Indirizzo
Istruzione
Memoria istruzioni
Registro di lettura 1
Registro di lettura 2
Registri
Registro di scrittura
Dato da scrivere
Dato letto 1
Dato letto 2
ALU
Zero
Risultato ALU
Indirizzo
Dato letto
Memoria dati
Dato da scrivere
16
Estensione segno
32

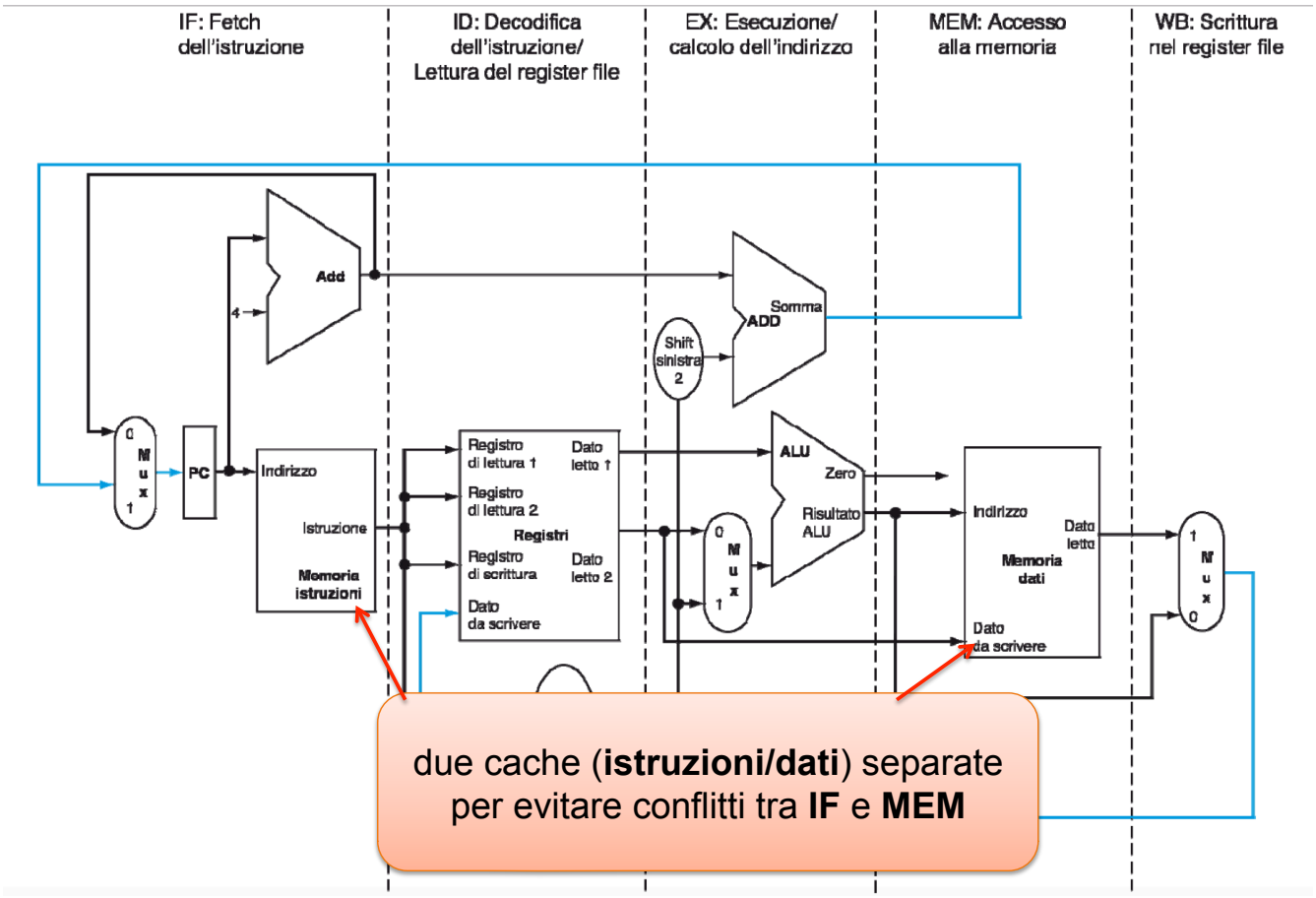

IF: Fetch dell'istruzione
ID: Decodifica dell'istruzione/ Lettura del register file
EX: Esecuzione/ calcolo dell'indirizzo
MEM: Accesso alla memoria
WB: Scrittura nel register file
In fase EX si calcola anche l'indirizzo dell'eventuale salto: NPC+(Imm <<2)
e la condizione del salto: (A-B)==0
ALUOutput ottenuto combinando 2 ingressi: (A,Imm) oppure (A,B)
multiplexer: seleziona uno dei due ingressi
Add
Somma ADD
Shift sinistra 2
PC
Indirizzo
Istruzione
Memoria istruzioni
Registro di lettura 1
Registro di lettura 2
Registri
Registro di scrittura
Dato da scrivere
Dato letto 1
Dato letto 2
ALU
Zero
Risultato ALU
Indirizzo
Dato letto
Memoria dati
Dato da scrivere
16
Estensione segno
32

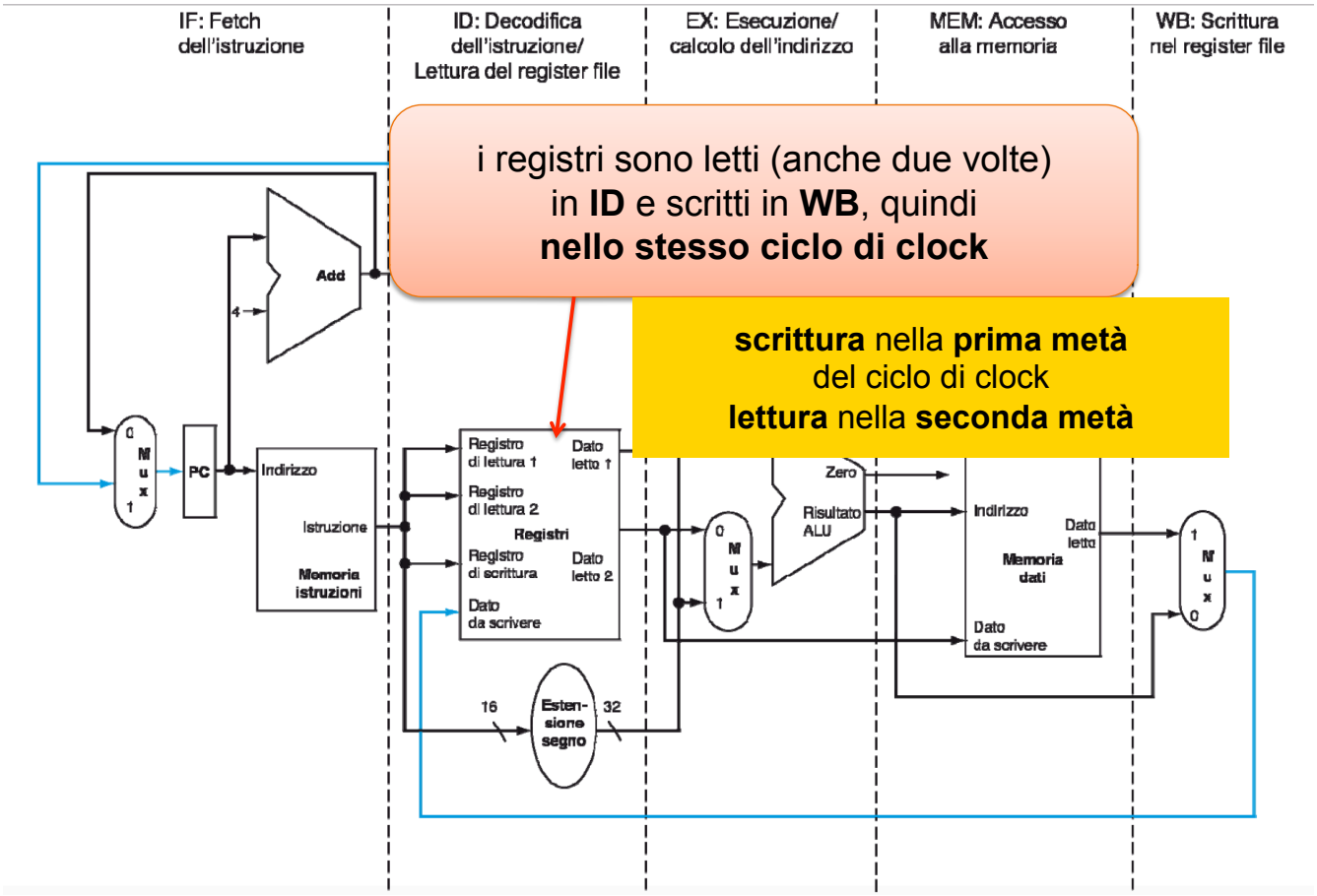

IF: Fetch dell'istruzione
MEM: Accesso alla memoria
WB: Scrittura nel register file
Due cose in questa fase:
1. accede alla memoria nel caso di lw/sw
2. aggiorna in PC
istruzione di load:
LMD ← Mem[ALUOutput]
Add
Shift sinistra 2
PC
Indirizzo
Istruzione
Memoria istruzioni
Registro di lettura 1
Registro di lettura 2
Registri
Registro di scrittura
Dato da scrivere
Dato letto 1
Dato letto 2
ALU
Zero
Risultato ALU
Indirizzo
Dato letto
Memoria dati
Dato da scrivere
16
Estensione segno
32


IF: Fetch dell'istruzione
MEM: Accesso alla memoria
WB: Scrittura nel register file
Due cose in questa fase:
1. accede alla memoria nel caso di lw/sw
2. aggiorna in PC
Add
4
Somma ADD
Shift sinistra 2
M u x
PC
Indirizzo
Istruzione
Memoria istruzioni
Registro di lettura 1
Registro di lettura 2
Registri
Registro di scrittura
Dato da scrivere
Dato letto 1
Dato letto 2
ALU
Zero
Risultato ALU
Indirizzo
Dato letto
Memoria dati
Dato da scrivere
16
Estensione segno
32
istruzione di store:
Mem[ALUOutput] ← B


IF: Fetch dell'istruzione
ID: Decodifica dell'istruzione/ Lettura del register file
EX: Esecuzione/ calcolo dell'indirizzo
MEM: Accesso alla memoria
WB: Scrittura nel register file
Add
4
Somma ADD
Shift sinistra 2
M u x
PC
Indirizzo
Istruzione
Memoria istruzioni
Registro di lettura 1
Registro di lettura 2
Registri
Dato letto 1
ALU
Zero
Risultato ALU
Indirizzo
Dato letto
Memoria dati
Dato da scrivere
aggiorna in PC scegliendo tra
• PC+4 oppure
• Indirizzo del salto
a seconda del segnale Zero
l'unità di controllo genera un segnale di selezione per i multiplexer


IF: Fetch dell'istruzione
ID: Decodifica dell'istruzione/ Lettura del register file
EX: Esecuzione/ calcolo dell'indirizzo
MEM: Accesso alla memoria
WB: Scrittura nel register file
Regs[rd] ← ALUOutput
Regs[rt] ← ALUOutput
Regs[rt] ← LMD
quindi deve scrivere in un registro (*selezionato in fase ID*)
ALUOutput **oppure** LMD
Add
4
M u x
PC
Indirizzo
Istruzione
Memoria istruzioni
Registro di lettura 1
Registro di lettura 2
Registri
Registro di scrittura
Dato da scrivere
Dato letto 1
Dato letto 2
Shift sinistra 2
ALU
Zero
Risultato ALU
Indirizzo
Dato letto
Memoria dati
Dato da scrivere
16
Estensione segno
32


IF: Fetch dell'istruzione
ID: Decodifica dell'istruzione/ Lettura del register file
EX: Esecuzione/ calcolo dell'indirizzo
MEM: Accesso alla memoria
WB: Scrittura nel register file
Add
4
Somma ADD
Shift sinistra 2
M u x
PC
Indirizzo
Istruzione
Memoria istruzioni
Registro di lettura 1
Registro di lettura 2
Registri
Registro di scrittura
Dato da scrivere
Dato letto 1
Dato letto 2
ALU
Zero
Risultato ALU
Indirizzo
Dato letto
Memoria dati
Dato da scrivere
due cache (**istruzioni/dati**) separate per evitare conflitti tra **IF** e **MEM**


IF: Fetch dell'istruzione
ID: Decodifica dell'istruzione/ Lettura del register file
EX: Esecuzione/ calcolo dell'indirizzo
MEM: Accesso alla memoria
WB: Scrittura nel register file
i registri sono letti (anche due volte) in ID e scritti in WB, quindi nello stesso ciclo di clock
scrittura nella prima metà del ciclo di clock
lettura nella seconda metà
Add
4
M u x
PC
Indirizzo
Istruzione
Memoria istruzioni
Registro di lettura 1
Registro di lettura 2
Registri
Registro di scrittura
Dato da scrivere
Dato letto 1
Dato letto 2
16
Esten-sione segno
32
Zero
Risultato ALU
Indirizzo
Dato letto
Memoria dati
Dato da scrivere