SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Architettura degli Elaboratori

Progetto della pipeline RISC senza forwarding

Prof. Andrea Sterbini – sterbini@di.uniroma1.it

Argomenti

- Come realizzare una pipeline
- Progetto della CPU MIPS con pipeline (senza gestione hazard)

Argomenti

- Come realizzare una pipeline
- Progetto della CPU MIPS con pipeline (senza gestione hazard)

- Come in una catena di montaggio, in una pipeline:

Argomenti

- Come realizzare una pipeline
- Progetto della CPU MIPS con pipeline (senza gestione hazard)

Come in una catena di montaggio, in una pipeline:

- **Obiettivo: fasi veloci (periodo di clock = durata della fase più lenta)**

Argomenti

- Come realizzare una pipeline
- Progetto della CPU MIPS con pipeline (senza gestione hazard)

- Come in una catena di montaggio, in una pipeline:
  - **Obiettivo: fasi veloci (periodo di clock = durata della fase più lenta)**
  - **Ciascuna fase realizza un solo compito** (o più compiti ma in parallelo)

Argomenti

- Come realizzare una pipeline
- Progetto della CPU MIPS con pipeline (senza gestione hazard)

Come in una catena di montaggio, in una pipeline:

- **Obiettivo: fasi veloci (periodo di clock = durata della fase più lenta)**
- **Ciascuna fase realizza un solo compito** (o più compiti ma in parallelo)
- Ciascuna fase **riceve informazioni e segnali di controllo**

Argomenti

- Come realizzare una pipeline
- Progetto della CPU MIPS con pipeline (senza gestione hazard)

Come in una catena di montaggio, in una pipeline:

- **Obiettivo: fasi veloci (periodo di clock = durata della fase più lenta)**
- **Ciascuna fase realizza un solo compito** (o più compiti ma in parallelo)
- Ciascuna fase **riceve informazioni e segnali di controllo**
- Ciascuna fase **passa alla successiva le informazioni e segnali di controllo**

Argomenti

- Come realizzare una pipeline
- Progetto della CPU MIPS con pipeline (senza gestione hazard)

▸ Come in una catena di montaggio, in una pipeline:

- **Obiettivo: fasi veloci (periodo di clock = durata della fase più lenta)**
- **Ciascuna fase realizza un solo compito** (o più compiti ma in parallelo)
- Ciascuna fase **riceve informazioni e segnali di controllo**
- Ciascuna fase **passa alla successiva le informazioni e segnali di controllo**
- I segnali necessari **devono restare stabili** durante tutta la fase

Argomenti

- Come realizzare una pipeline
- Progetto della CPU MIPS con pipeline (senza gestione hazard)

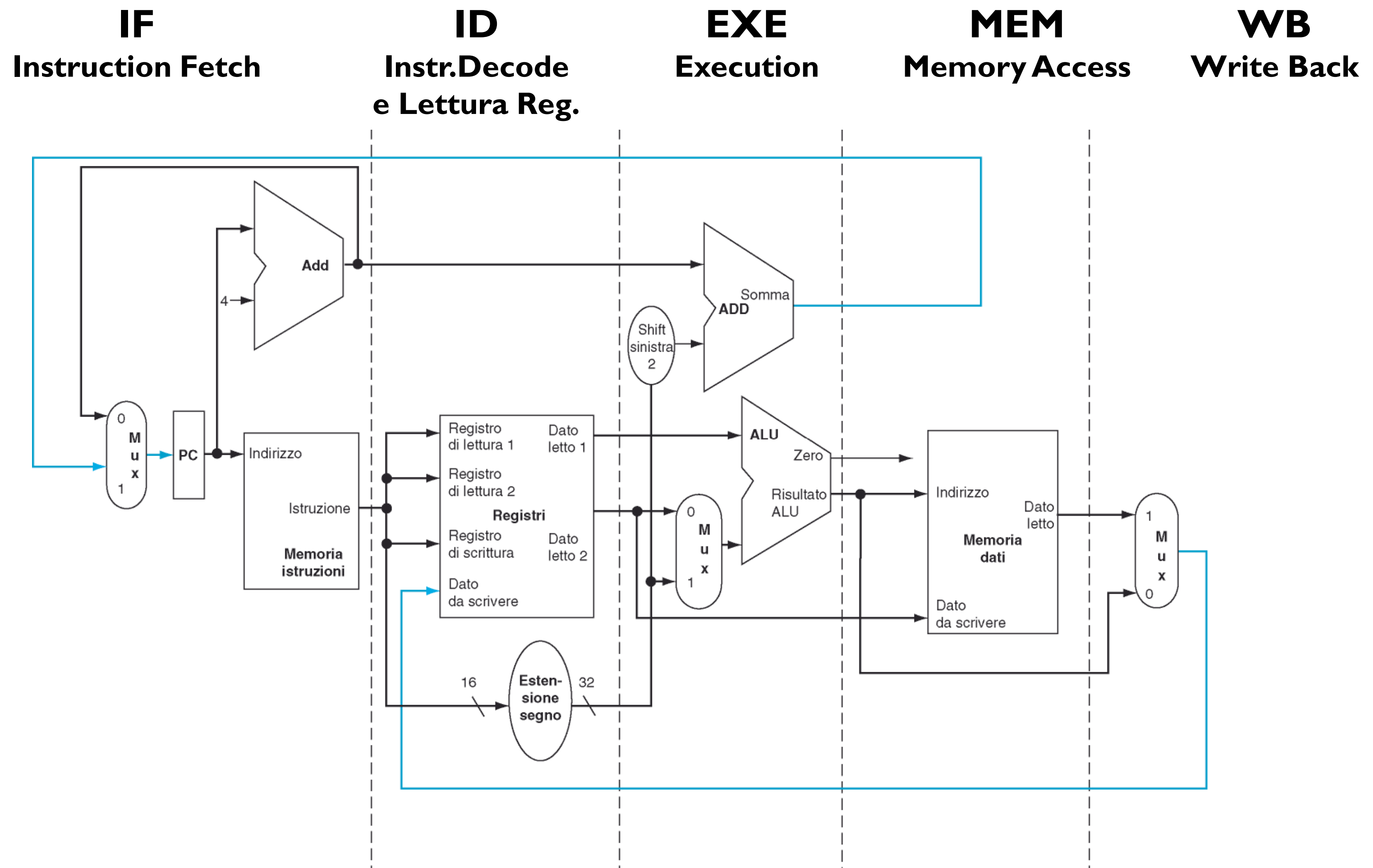
- Come in una catena di montaggio, in una pipeline:
  - **Obiettivo: fasi veloci (periodo di clock = durata della fase più lenta)**
  - **Ciascuna fase realizza un solo compito** (o più compiti ma in parallelo)
  - Ciascuna fase **riceve informazioni e segnali di controllo**
  - Ciascuna fase **passa alla successiva le informazioni e segnali di controllo**
  - I segnali necessari **devono restare stabili** durante tutta la fase
- Per rendere stabili dei segnali si usano LATCH oppure registri che cambiano solo alla transizione del clock

Argomenti

- Come realizzare una pipeline
- Progetto della CPU MIPS con pipeline (senza gestione hazard)

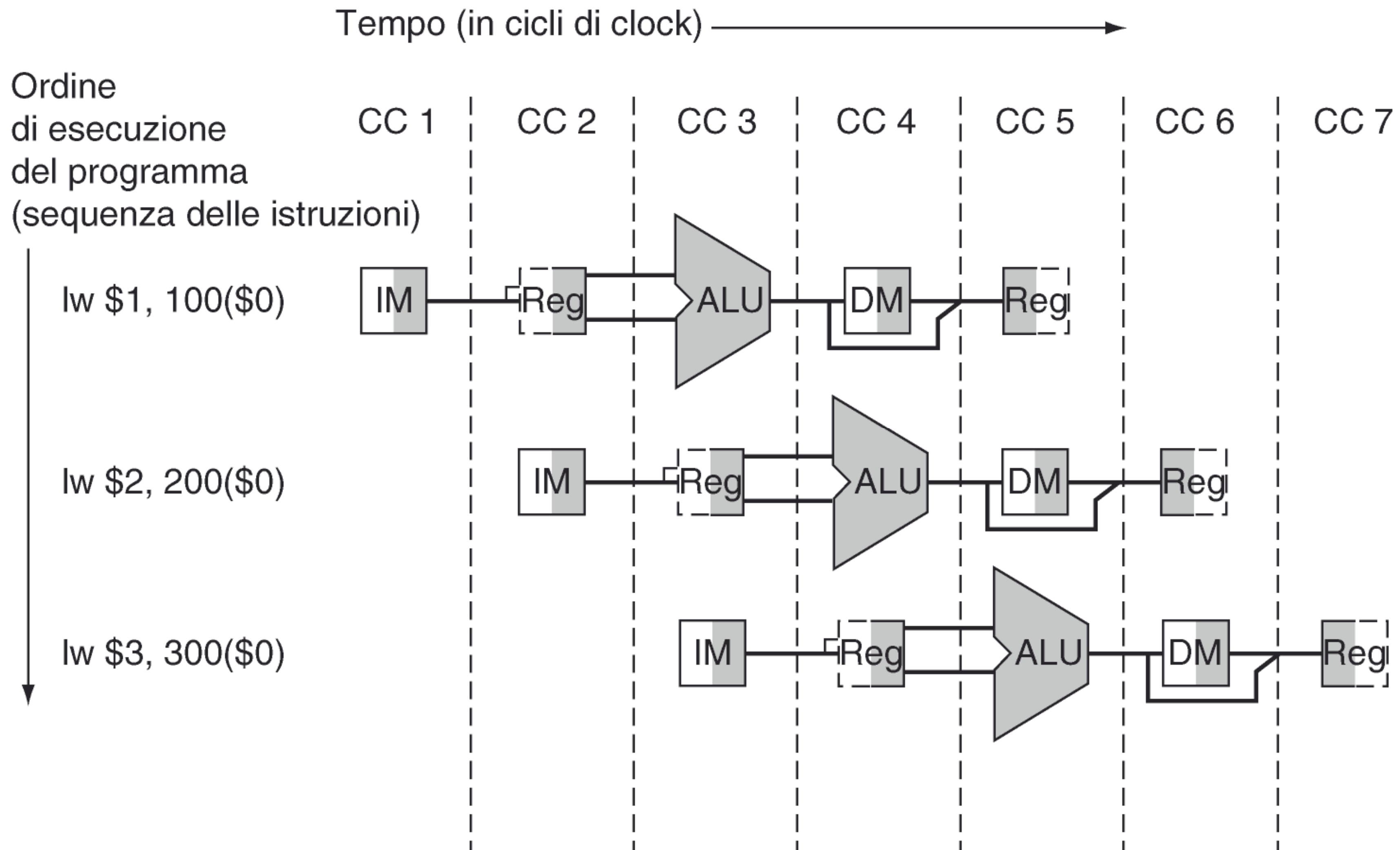
- Come in una catena di montaggio, in una pipeline:
  - **Obiettivo: fasi veloci (periodo di clock = durata della fase più lenta)**
  - **Ciascuna fase realizza un solo compito** (o più compiti ma in parallelo)
  - Ciascuna fase **riceve informazioni e segnali di controllo**
  - Ciascuna fase **passa alla successiva le informazioni e segnali di controllo**
  - I segnali necessari **devono restare stabili** durante tutta la fase
- Per rendere stabili dei segnali si usano LATCH oppure registri che cambiano solo alla transizione del clock
- **Soluzione:** separare ciascuna fase dalla successiva con un registro che riceve informazioni e segnali di controllo dalla fase precedente e li mette a disposizione alla fase successiva.

Unità funzionali delle 5 fasi



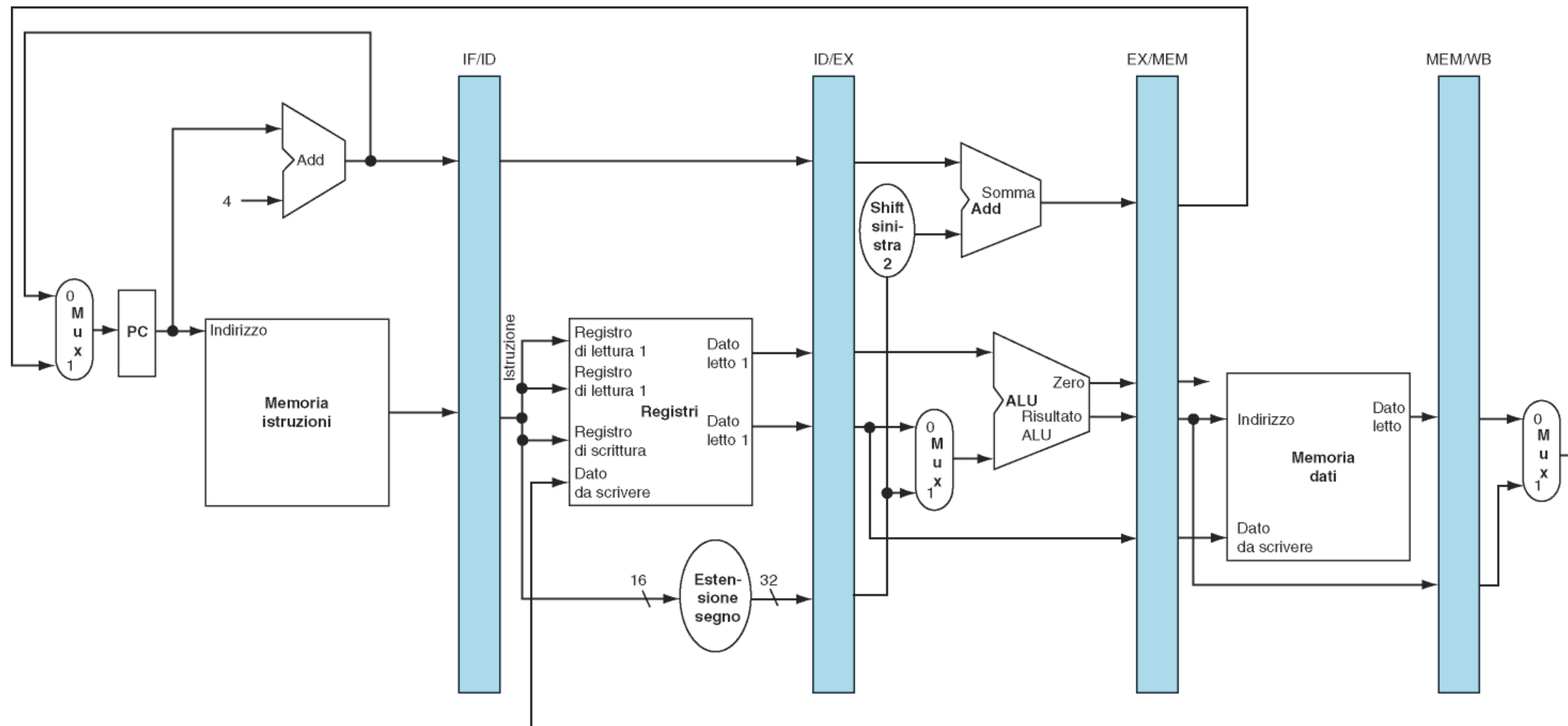
Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Esempio di esecuzione



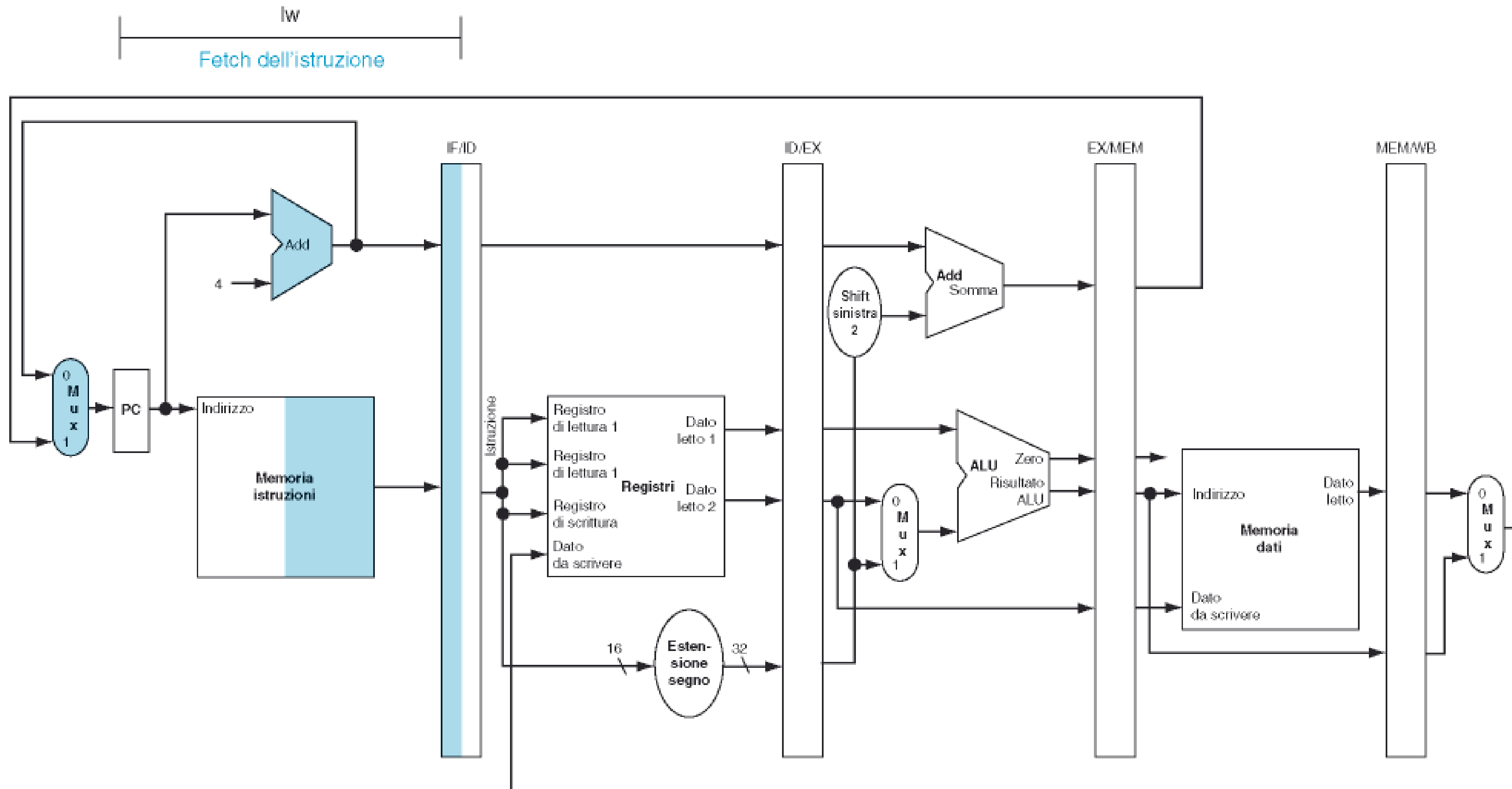
Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Inseriamo i registri tra le fasi



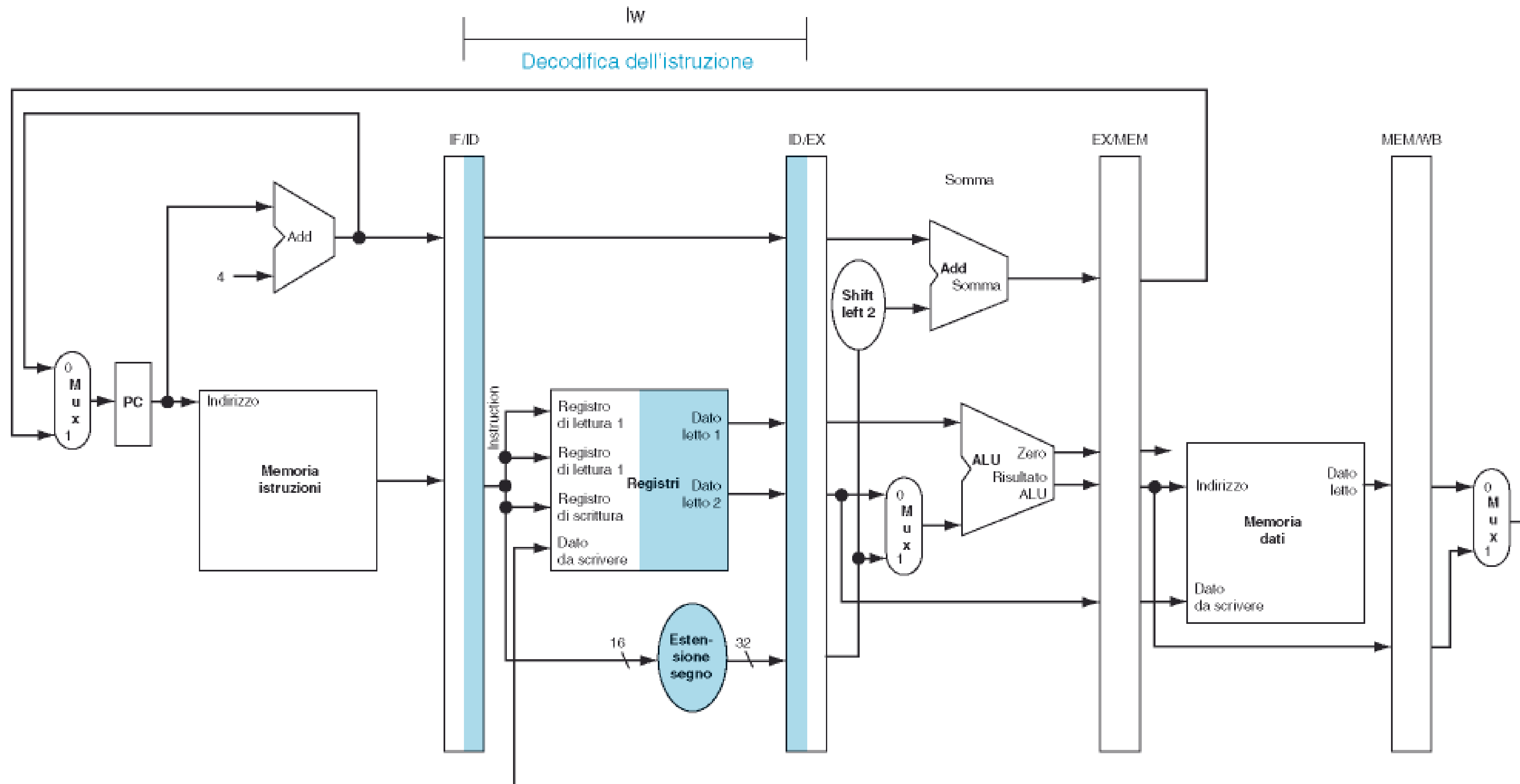
Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Fetch della istruzione



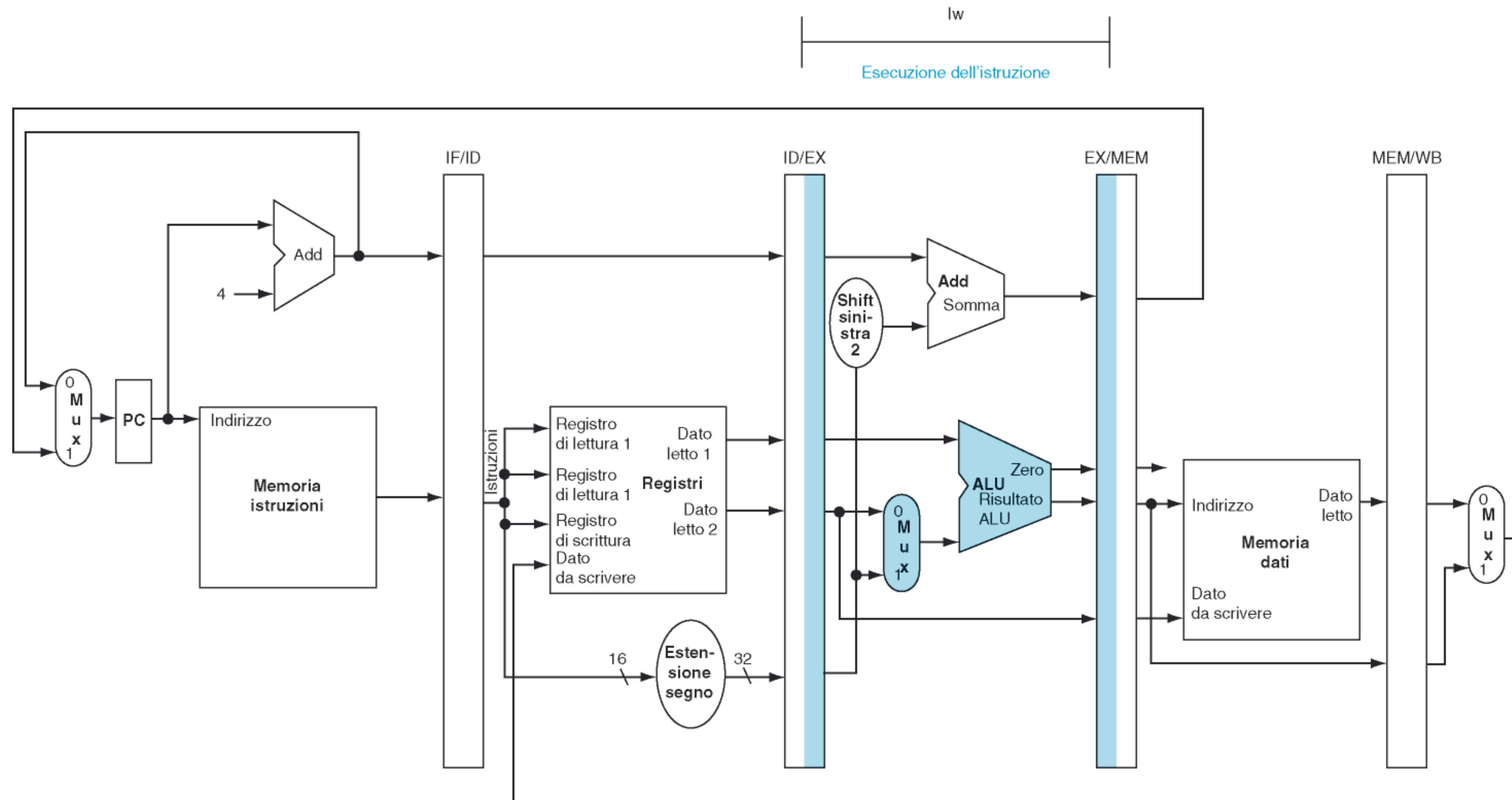
Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Decodifica e lettura registri



Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Esecuzione operazione (o calcolo indirizzo)



Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

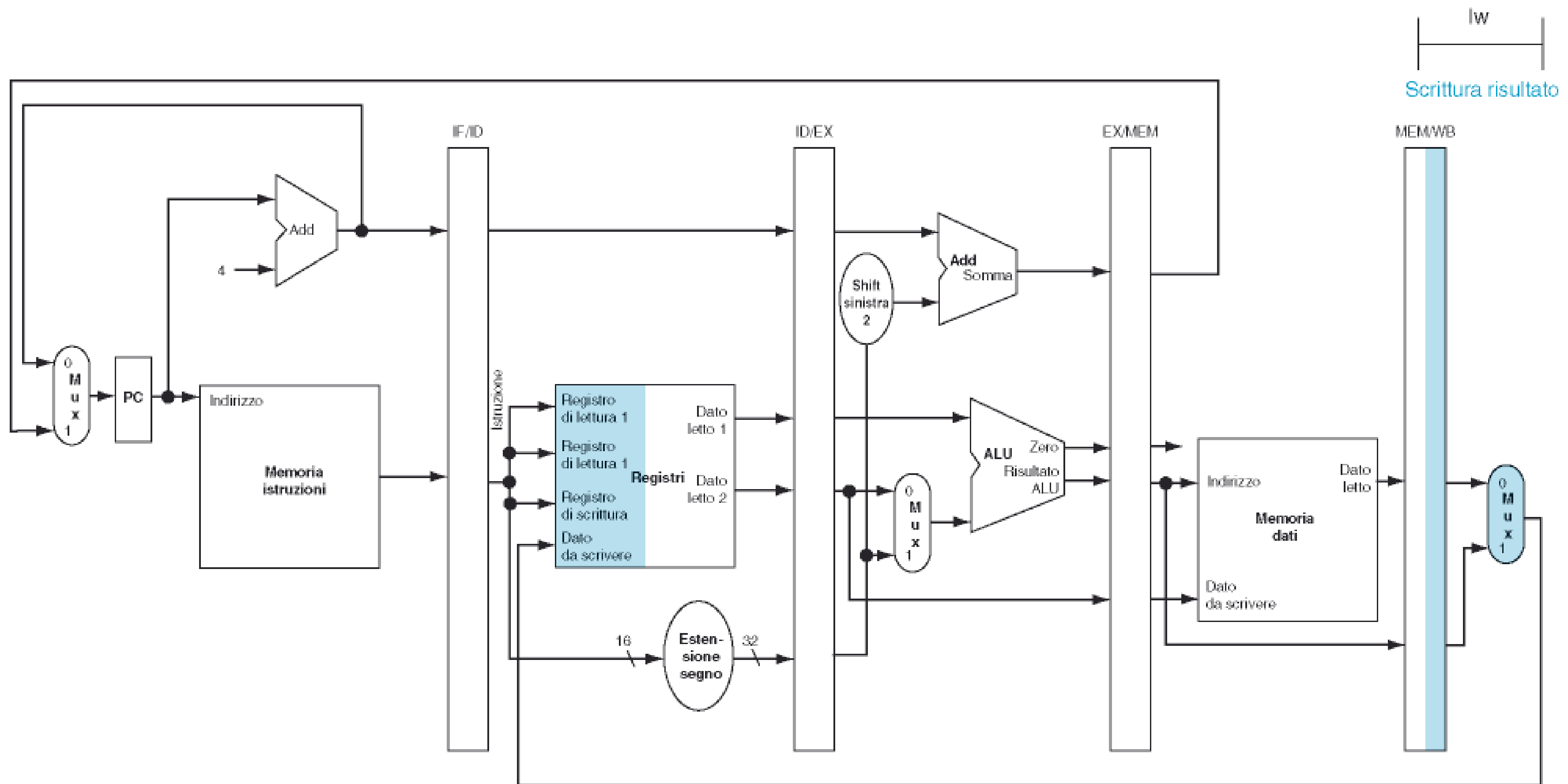
Accesso alla memoria

lw

Memoria dati

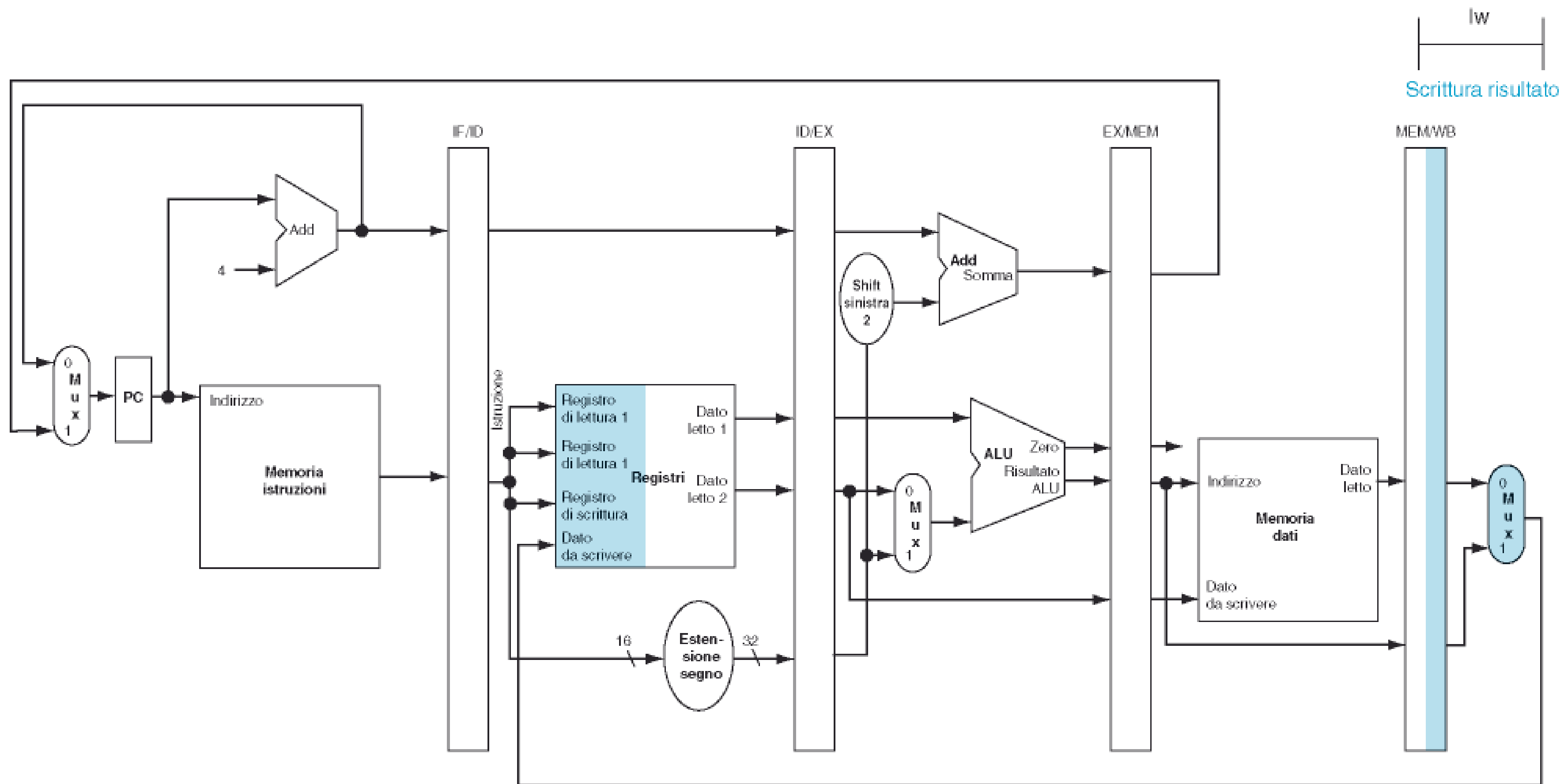
Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Write Back (scrittura nei registri)



Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

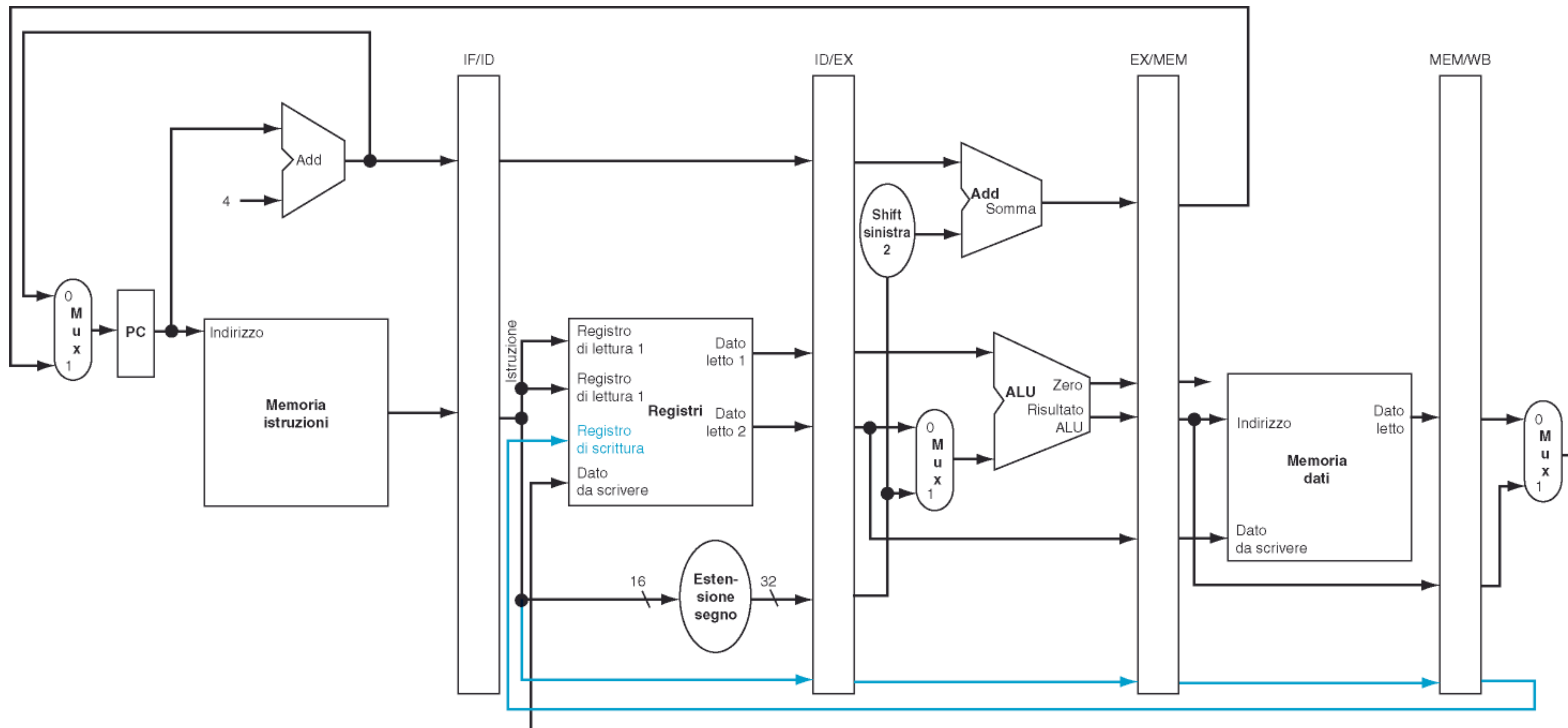
Write Back (scrittura nei registri)



- **NOTA: fatto così, il write-back avviene sul registro sbagliato!!!**
- **Il Registro Destinazione è quello indicato dalla istruzione che si trova 3 passi dopo! (e che intanto è stata caricata ed è arrivata alla fase di Instruction Decode)**

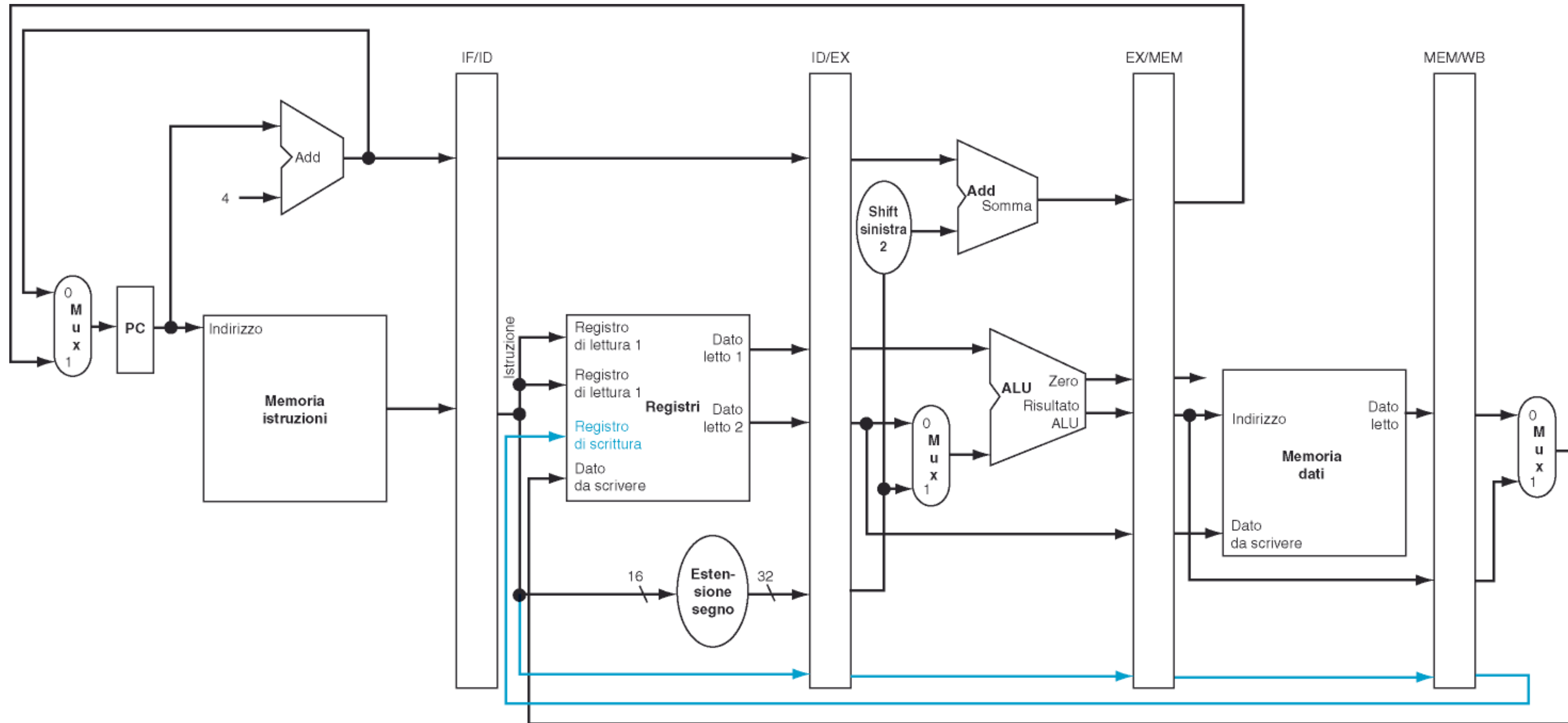
Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Write Back corretto



Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

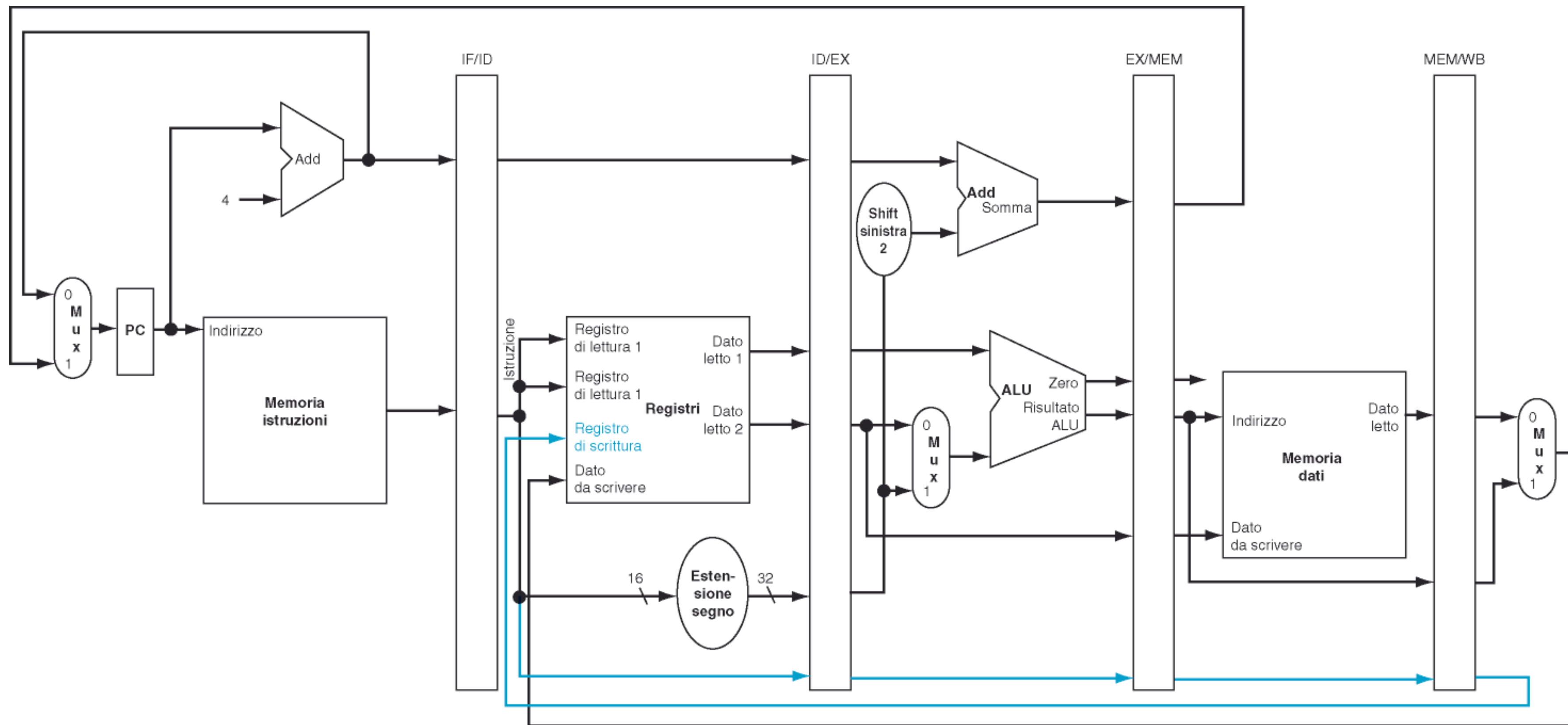
Write Back corretto



- **TUTTE le informazioni e TUTTI i segnali di controllo DEVONO essere nel registro della pipeline precedente alla fase**

Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

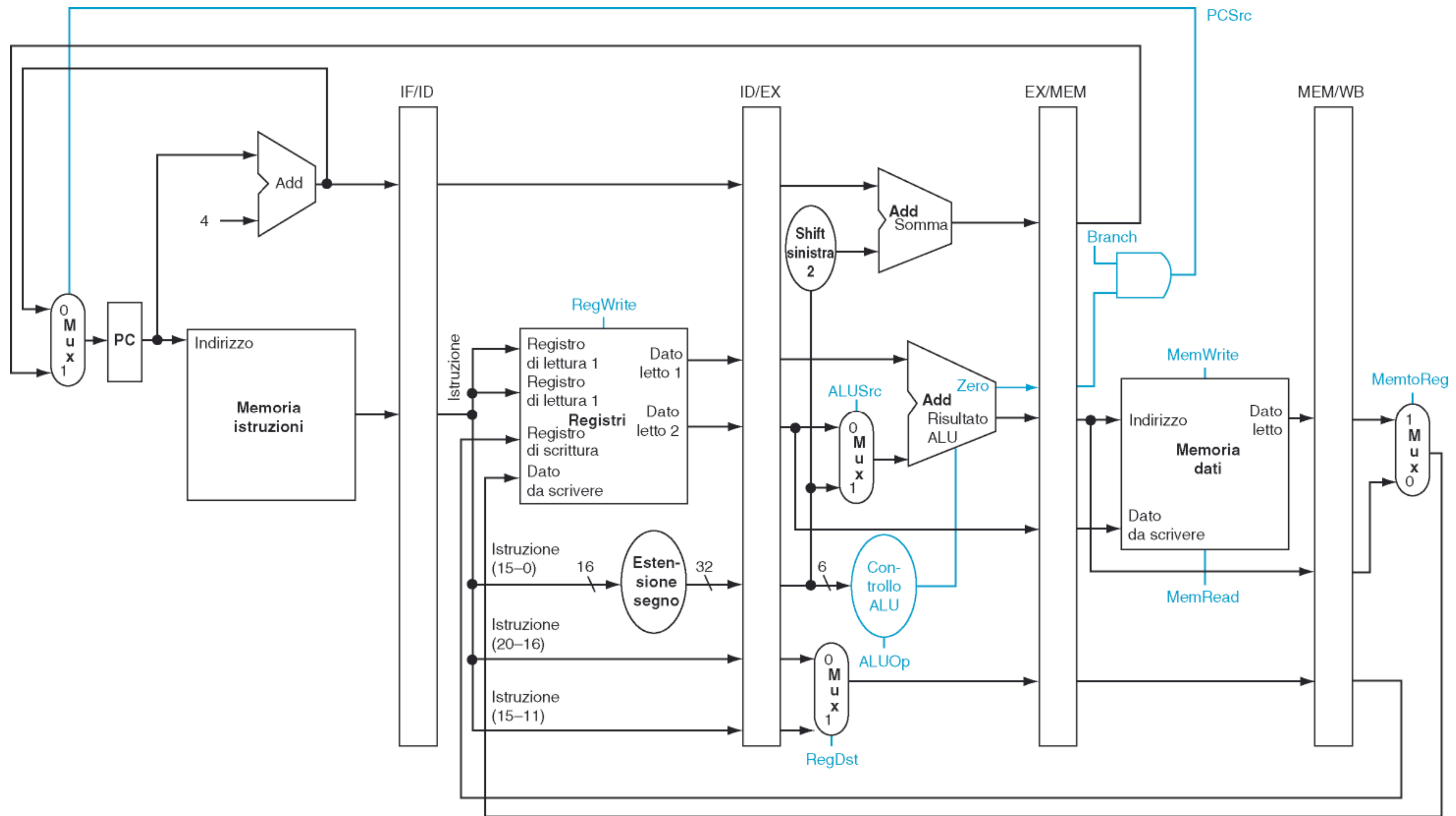
Write Back corretto



- **TUTTE le informazioni e TUTTI i segnali di controllo DEVONO essere nel registro della pipeline precedente alla fase**
  - **(mandiamo mano a mano avanti il reg. destinazione assieme all'istruzione fino ad arrivare alla fase di Write Back)**

Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Con la logica dei salti (beq)



Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

I segnali di controllo della CU

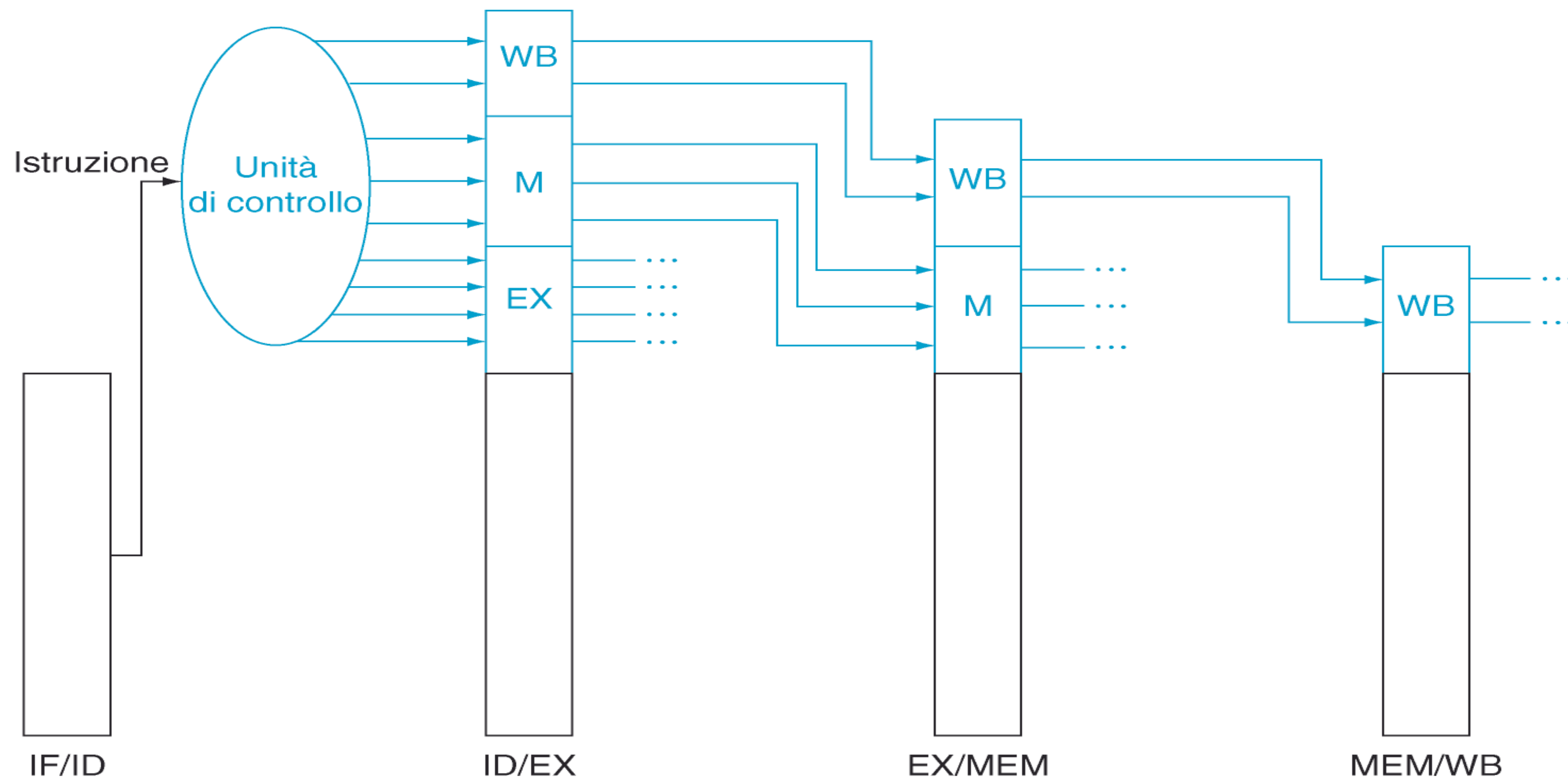
- Possiamo dividere i segnali di controllo delle fasi esecutive in 3 gruppi:
- per la fase EXE
- per la fase MEM
- per la fase WB

Istruzione	Segnali di controllo dello stadio di esecuzione/calcolo dell'indirizzo				Segnali di controllo dello stadio di accesso alla memoria dati			Segnali di controllo dello stadio di *Write-back*	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch Mem	Read Mem	Write	RegWrite	Memto-Reg
Formato-R	1	1	0	0	0	0	0	1	0
`lw`	0	0	0	1	0	1	0	1	1
`sw`	X	0	0	1	0	0	1	0	X
`beq`	X	0	1	0	1	0	0	0	X

- Questi segnali devono essere passati di fase in fase da ciascuno dei registri della pipeline al successivo

Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

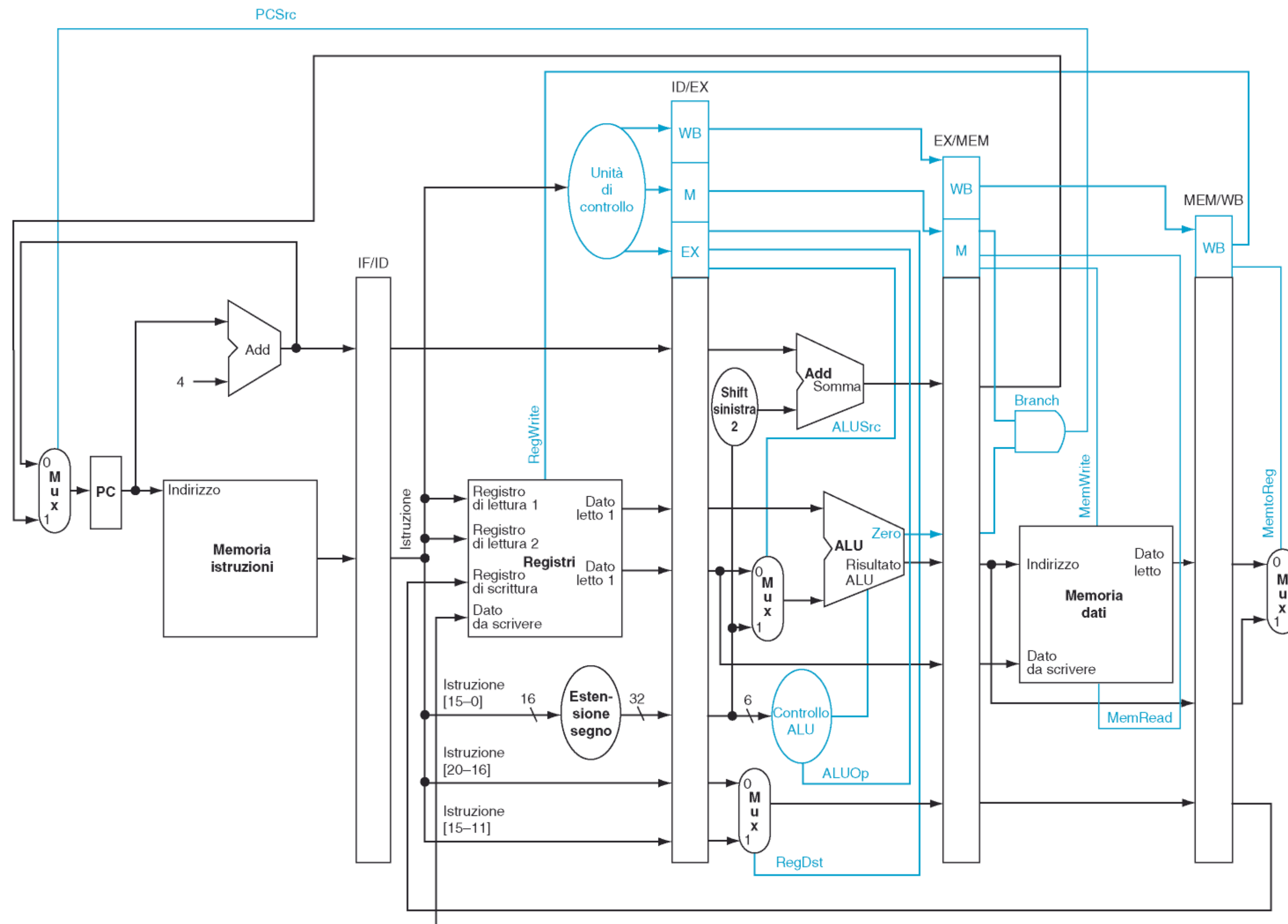
CU e registri della pipeline



- Ad ogni fase ci portiamo appresso i DATI (letti dai registri o dalla parte immediata)
- e i segnali di CONTROLLO della stessa istruzione (generati dalla CU)

Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

La CPU (quasi) completa



Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Soluzione esercizio

- Calcolate il numero di cicli necessari a eseguire le istruzioni seguenti:
  - individuate i **data e control hazard**
  - per **determinare se con il forwarding possono essere risolti** tracciate il diagramma temporale della pipeline
  - determinate **quali non possono essere risolti e necessitano di stalli** (e quanti stalli)
  - tenente conto del tempo necessario a **caricare la pipeline**
- Assumete:
  che la beq salti alla fine della fase EXE,
  che il salto beq non sia ritardato,
  che j non introduca stalli

```
# Sommo un vettore di word
sommavettore:
        li   $t0,   0     # somma
        li   $t1,   40    # fine
        li   $t2,   0     # offset
ciclo:  beq  $t2, $t1, fine
        lw   $t3,   vettore($t2)
        add  $t0, $t0, $t3
        addi $t2, $t2, 4
        j ciclo
fine:   li   $v0,   10
        syscall
```

Soluzione

```
sommavettore:                  # Sottolineato = hazard
        li   $t0,   0          #     fine istruzione al 5° colpo di clock
        li   $t1,   40         #                        6°
        li   $t2,   0          # IF ID EX MM WB         7°
ciclo:  beq  $t2, $t1, fine    #    IF ID EX MM WB      8° 14° 20° ... 68°
        lw   $t3,   vettore($t2)# IF ID EX MM WB        9° 15° ...
        add  $t0, $t0, $t3     #    → IF ID EX MM WB   11° 17° ...
        addi $t2, $t2, 4       #                       12° 18° ...
        j ciclo                #                       13° 19° ...
fine:   li   $v0,   10         #                       71°
        syscall                #                       72°
```

- Il ciclo impiega 6 colpi di clock perché è necessario 1 stallo tra lw e add
- Tutti gli altri data-hazard sono risolvibili con il forwarding
- C'è un control-hazard sul salto beq alla fine del ciclo (che inserisce 2 stalli)
- Quindi in totale ci vogliono: 8 + 10*(5+1) + 2 + 2 = **72 colpi di clock**
- **Esercizio per casa: cosa sarebbe successo SENZA FORWARDING?**