

Architettura degli Elaboratori

Pipelining:
Unità di Elaborazione e di Controllo



Barbara Masucci

UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

DIPARTIMENTO DI ECCELLENZA

Punto della situazione

- Abbiamo studiato
 - Una prima implementazione hardware (a ciclo singolo) di un sottoinsieme dell'IS del MIPS e visto che è inefficiente
- Obiettivo di oggi e della prossima lezione
 - Studiare una seconda implementazione, più efficiente, basata su pipeline
 - In particolare, vedremo la realizzazione dell'Unità di Elaborazione con pipeline e dell'Unità di Controllo ad essa associata



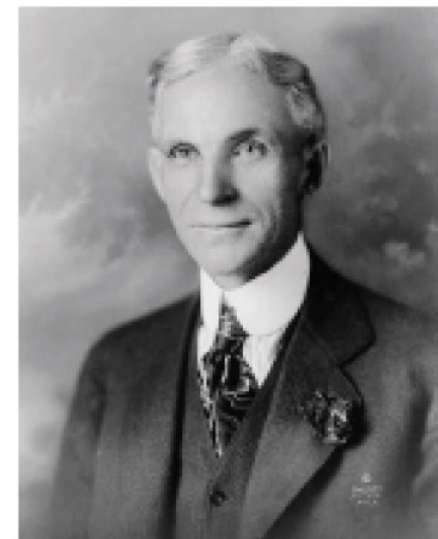
Implementazione a Ciclo Singolo

- L'implementazione a **ciclo singolo** non viene utilizzata perché è **inefficiente**
 - Motivo: la **lunghezza del ciclo di clock** (uguale per tutte le istruzioni) è **determinata dal cammino di elaborazione più lungo** all'interno del processore
 - L'istruzione più lenta è la **lw**
 - Utilizza 5 unità funzionali in sequenza: memoria istruzioni, register file, ALU, memoria dati, register file
 - Questo comporta un **ciclo di clock molto lungo**, mentre molte istruzioni possono essere eseguite in tempo minore
 - Se si aggiungono istruzioni più complesse di quelle considerate finora (ad esempio quelle in virgola mobile), le cose peggiorano
 - **Come risolvere il problema?**







La Pipeline

- E' una **tecnica di implementazione hardware** di un set di istruzioni, utilizzata da quasi tutti gli elaboratori
- Prevede la **sovrapposizione temporale** dell'esecuzione delle varie istruzioni
- L'idea su cui si basa è la stessa utilizzata nelle **catene di montaggio** (assembly line)
 - Introdotte da Henry Ford ai primi del '900
 - Consentono di ridurre i tempi necessari per la realizzazione di un manufatto complesso, **dividendo il lavoro** degli operai **in compiti**



Esempio: il bucato

- Supponiamo di dover fare il bucato e che questo consista nelle seguenti attività
 - Mettere la biancheria nella lavatrice 
 - Terminato il lavaggio, mettere la biancheria bagnata nell'asciugatrice 
 - Terminata l'asciugatura, procedere alla stiratura 
 - Terminata la stiratura, riporre la biancheria nell'armadio 
- L'approccio non basato su pipeline **procede ad un nuovo ciclo** per il bucato **solo quando il precedente è terminato**



L'approccio **basato su pipeline** consente di risparmiare tempo, **effettuando più cicli di bucato contemporaneamente**

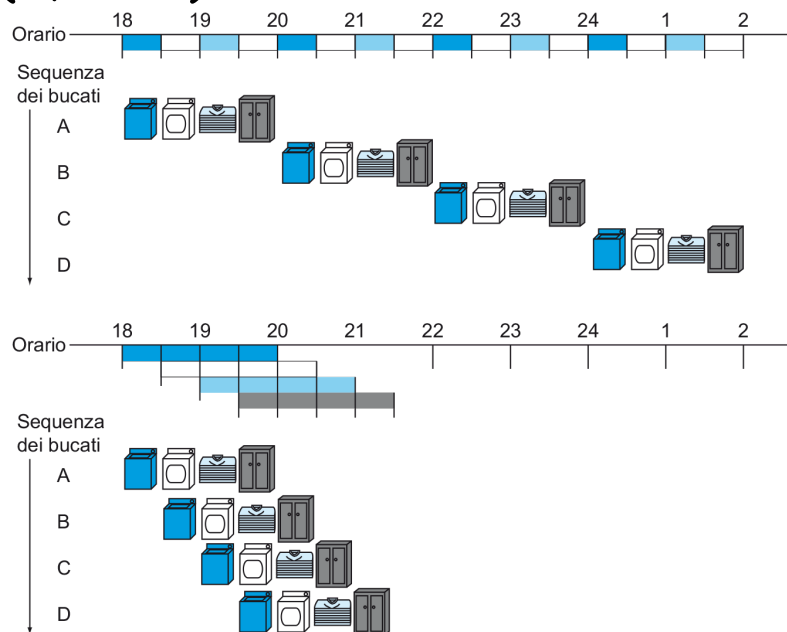
Il bucato con pipeline

- Nell'approccio **basato su pipeline**
 - Quando la lavatrice termina il lavaggio del **primo carico**, viene subito inserito il **secondo carico**, mentre l'asciugatrice asciuga il **primo carico**
 - Quando l'asciugatrice termina il **primo carico**, viene subito inserito il **secondo carico**, mentre viene stirato il **primo carico** e la lavatrice procede al **terzo carico**
 - Quando la stiratura del **primo carico** viene terminata e i panni vengono riposti nell'armadio, si passa a stirare il **secondo carico**, mentre nell'asciugatrice viene inserito il **terzo carico** e nella lavatrice il **quarto carico**



Tempo di esecuzione per il bucato

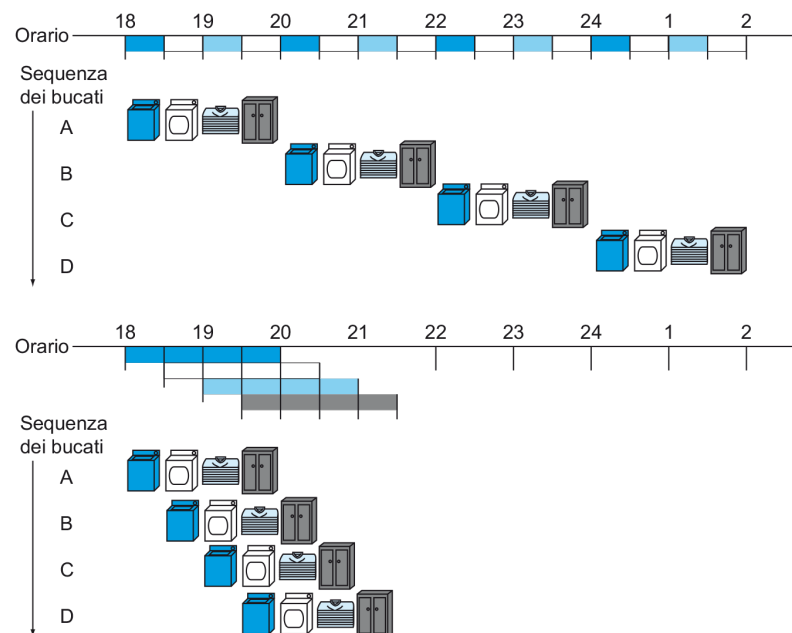
- Supponendo che ciascuna attività duri 30 minuti, **l'intero ciclo per il bucato dura due ore**
 - Con il primo metodo, tra le 18:00 e le 20:00 viene eseguito **un solo ciclo per il bucato** e sono richieste 8 ore per completare 4 cicli
 - Con il secondo metodo tra le 18:00 e le 21:30 vengono eseguiti **4 cicli per il bucato** (3,5 ore)



Tempo di esecuzione per il bucato

➤ La pipeline

- Non riduce il tempo totale richiesto per un ciclo di bucato (consiste sempre in 2 ore)
- Migliora il **throughput** (numero di cicli di bucato completati nell'unità di tempo)



Il bucato con pipeline

- Nell'approccio **basato su pipeline** sono utilizzati 4 **stadi** diversi
 - Lavaggio
 - Asciugatura
 - Stiratura
 - Sistemazione della biancheria
- Se i cicli di bucato da fare sono molti, **l'incremento di velocità diventa pari al numero degli stadi della pipeline** (4 in questo caso)



II MIPS con pipeline

- Possiamo usare l'approccio con pipeline per eseguire le istruzioni del **processore MIPS**
- Saranno necessari **tanti stadi quante sono le fasi richieste** per l'esecuzione delle istruzioni
 - **Prelievo** (fetch) dell'istruzione dalla memoria istruzioni
 - **Decodifica** dell'istruzione e **lettura** dei registri
 - **Esecuzione** di un'operazione o calcolo di un indirizzo
 - **Accesso** a un operando nella **memoria dati**
 - **Scrittura** del risultato in un registro del register file
- Quindi avremo bisogno di una **pipeline con 5 stadi**



Confronto di prestazioni

- Consideriamo una pipeline che sia in grado di eseguire le istruzioni seguenti:
 - lw
 - sw
 - Formato R
 - beq
- Confrontiamo il tempo medio necessario per l'esecuzione di queste istruzioni nelle due implementazioni
 - A ciclo singolo
 - Con pipeline



Confronto di prestazioni

- Supponiamo che i **tempi richiesti** per le varie fasi siano rappresentati in tabella
 - Non si considerano i ritardi introdotti dai multiplexer, dall'unità di controllo, dall'accesso al registro PC e dall'estensione del segno

Tipo di istruzione	Lettura dell'istruzione	Lettura dei registri	Operazione con la ALU	Accesso ai dati in memoria	Scrittura del register file	Tempo totale
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
Formato R (add, sub, and, or, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Salto condizionato (beq)	200 ps	100 ps	200 ps			500 ps

1 ps (picosecondo) = 10^{-12} s

- Nell'implementazione a ciclo singolo bisogna adeguarsi all'istruzione più lenta (**lw**) per stabilire la durata del ciclo di clock: quindi **800 ps**



Confronto di prestazioni

- Consideriamo una sequenza di **tre** istruzioni **lw**
- L'**implementazione a ciclo singolo** richiede $800 \times 3 = 2400$ ps
- Nell'**implementazione con pipeline**
 - Ciascuno dei 5 stadi impiega un ciclo di clock
 - La durata del ciclo di clock va tarata in base alla durata dello stadio più lento (200 ps), anche se alcuni stadi impiegano meno tempo
 - Ad esempio, la lettura dai registri richiede 100 ps
 - Quindi i 5 stadi della pipeline in sequenza richiedono 5 cicli di clock, per un totale di $200 \times 5 = 1000$ ps
 - Il tempo richiesto per eseguire le tre istruzioni è **1400 ps**
 - Ogni istruzione successiva alla prima fa aumentare il tempo di esecuzione di 200 ps, per cui il tempo richiesto dalle tre istruzioni è
$$200 \times 5 + 200 + 200 = 1400 \text{ ps}$$



Confronto di prestazioni

Ordine di esecuzione
del
programma

(sequenza delle istruzioni)

200

400

600

800

1000

1200

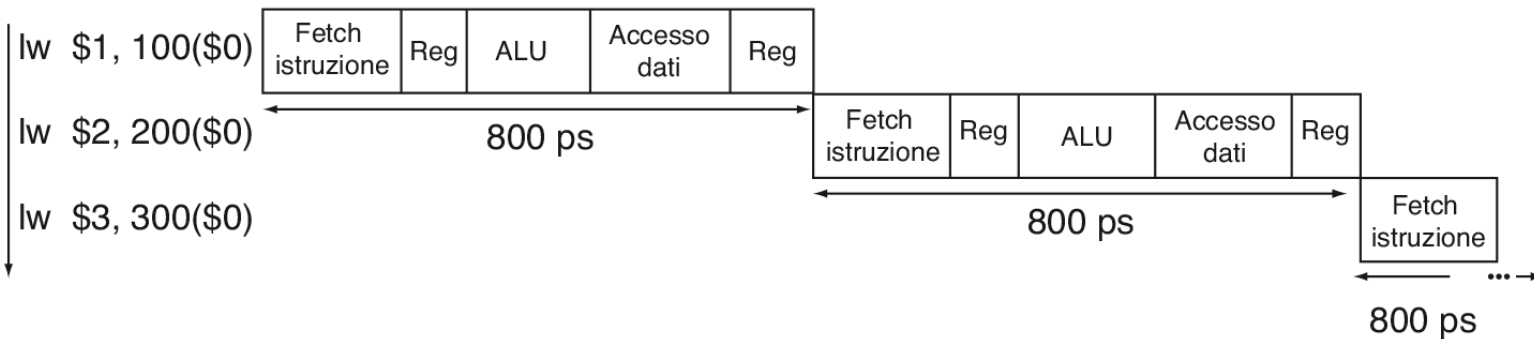
1400

1600

1800

Tempo

Implementazione a ciclo singolo



Implementazione con pipeline

Ordine di esecuzione
del
programma

(sequenza delle istruzioni)

200

400

600

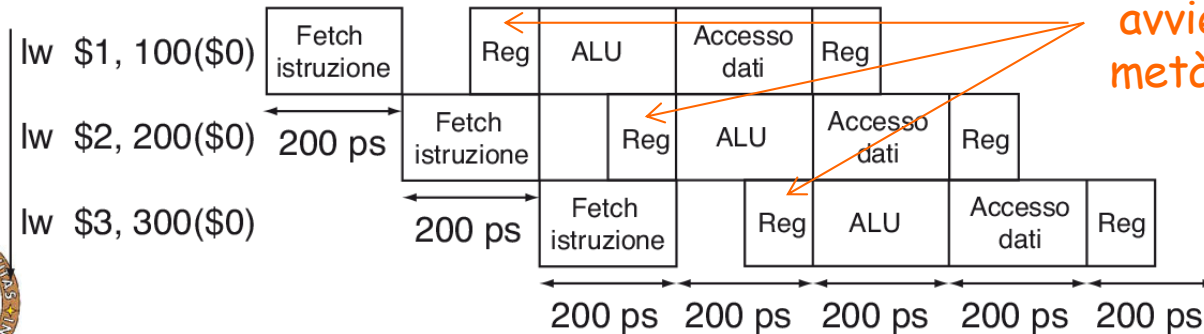
800

1000

1200

1400

Tempo



La lettura dal register file
avviene nella seconda
metà del ciclo di clock

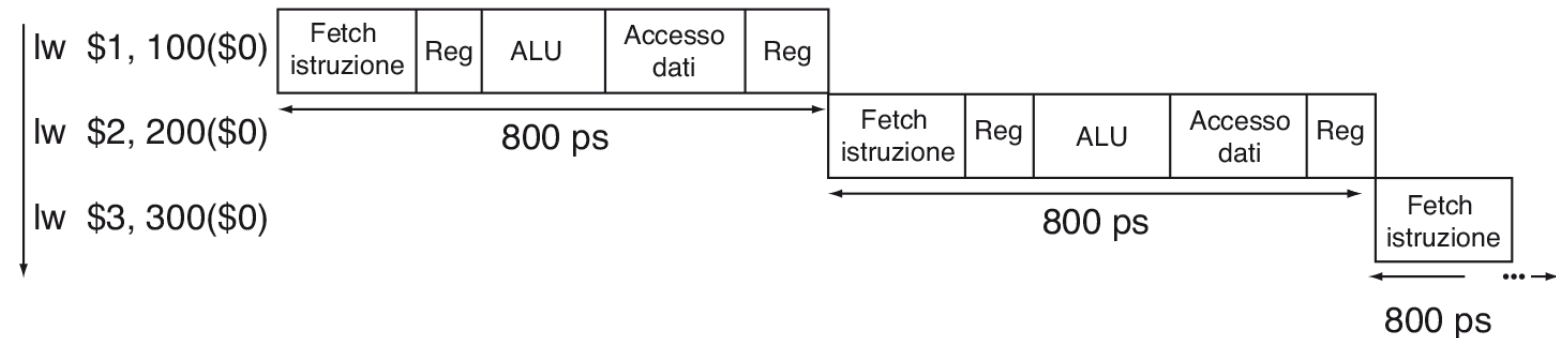


Confronto di prestazioni

Ordine di esecuzione
del
programma
(sequenza delle istruzioni)

200 400 600 800 1000 1200 1400 1600 1800 Tempo

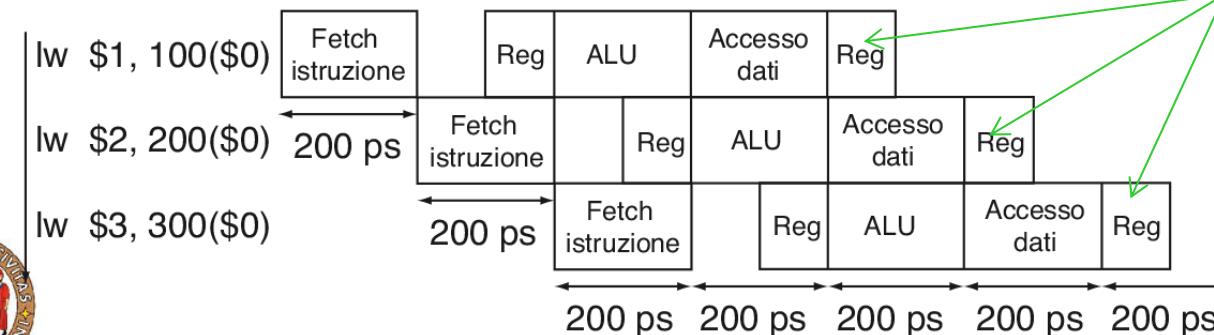
Implementazione a ciclo singolo



Implementazione con pipeline

Ordine di esecuzione
del
programma
(sequenza delle istruzioni)

200 400 600 800 1000 1200 1400 Tempo



La scrittura dal register file
avviene nella prima
metà del ciclo di clock



Confronto di prestazioni

- Se la pipeline opera in condizioni ideali, con **un numero molto grande di istruzioni**, il tempo di esecuzione viene calcolato con la formula seguente:

$$\text{Tempo tra due istruzioni con pipeline} \geq \frac{\text{Tempo tra due istruzioni senza pipeline}}{\text{\# stadi della pipeline}}$$

- Quindi una pipeline a 5 stadi può consentire un incremento di velocità pari a 5
 - Considerando l'esempio precedente, si può passare da 2400 ps a 480 ps ($480 = 2400/5$)
 - Perché noi abbiamo ottenuto 1400 ps invece di 480 ps?
 - Abbiamo considerato solo poche istruzioni



Confronto di prestazioni

- Vediamo che accade se **aumentiamo il numero di istruzioni**
 - Aggiungiamo 1.000.000 di istruzioni di tipo **lw** alle 3 precedenti, per un totale di 1.000.003 istruzioni
 - In assenza di pipeline, il tempo di esecuzione è $1.000.003 \times 800$ ps, cioè 800.002.400 ps
 - Con pipeline, ogni istruzione in più fa aumentare il tempo di esecuzione di 200 ps, per un totale di $1.000.000 \times 200 + 1400$ ps, cioè 200.001.400 ps
 - Quindi il rapporto tra i tempi di esecuzione nelle due implementazioni è

$$\frac{800.002.400}{200.001.400} = 3,999$$



Osservazioni

- L'implementazione con pipeline aumenta il **numero di istruzioni contemporaneamente in esecuzione**
- Quindi, la pipeline
 - Aumenta il **throughput**
(numero di istruzioni eseguite nell'unità di tempo)
 - Non diminuisce la **latenza**
(tempo di esecuzione della singola istruzione), che resta sempre la stessa



Pipeline nel MIPS

- Abbiamo già detto che per implementare la **pipeline nel MIPS** abbiamo bisogno di **5 stadi**
 - **Primo stadio:** Prelievo dell'istruzione dalla memoria istruzioni
 - **Secondo stadio:** Decodifica dell'istruzione e lettura dei registri
 - **Terzo stadio:** Esecuzione di un'operazione o calcolo di un indirizzo
 - **Quarto stadio:** Accesso a un operando nella memoria dati
 - **Quinto stadio:** Scrittura del risultato in un registro

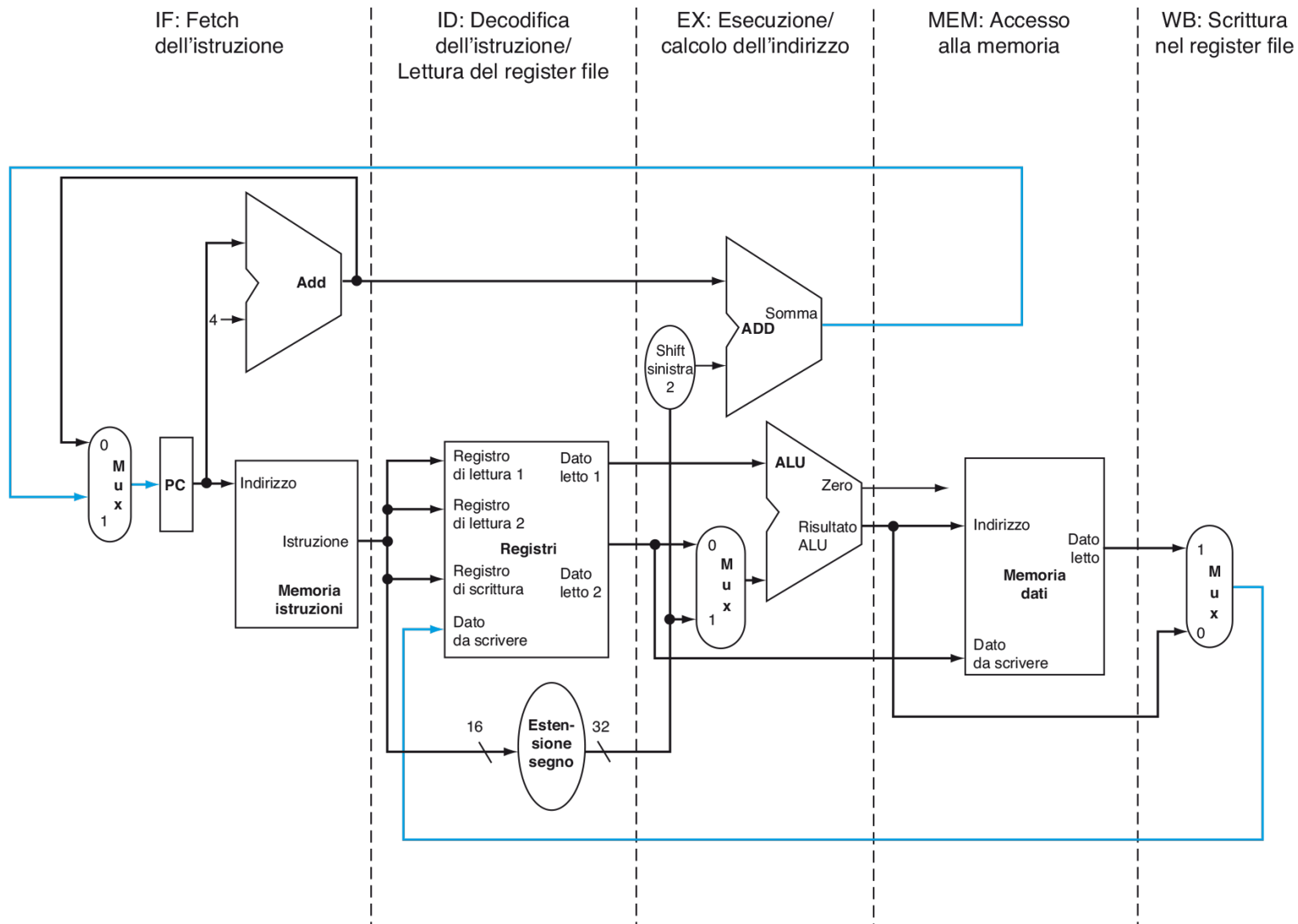


Pipeline nel MIPS

- Vediamo come realizzare l'Unità di Elaborazione con pipeline e l'Unità di Controllo ad essa associata
- Ripartiamo dall'Unità di Elaborazione a ciclo singolo e identifichiamo i 5 stadi della pipeline
 - **IF** (**I**nstruction **F**etch): prelievo istruzione ed incremento PC
 - **ID** (**I**nstruction **D**ecode): Decodifica istruzione e lettura registri
 - **EX** (**EX**ecution): Esecuzione operazione / calcolo indirizzo
 - **MEM** (**MEM**ory): Accesso in memoria
 - **WB** (**W**rite **B**ack): Scrittura banco registri



L'unità di Elaborazione a Ciclo Singolo

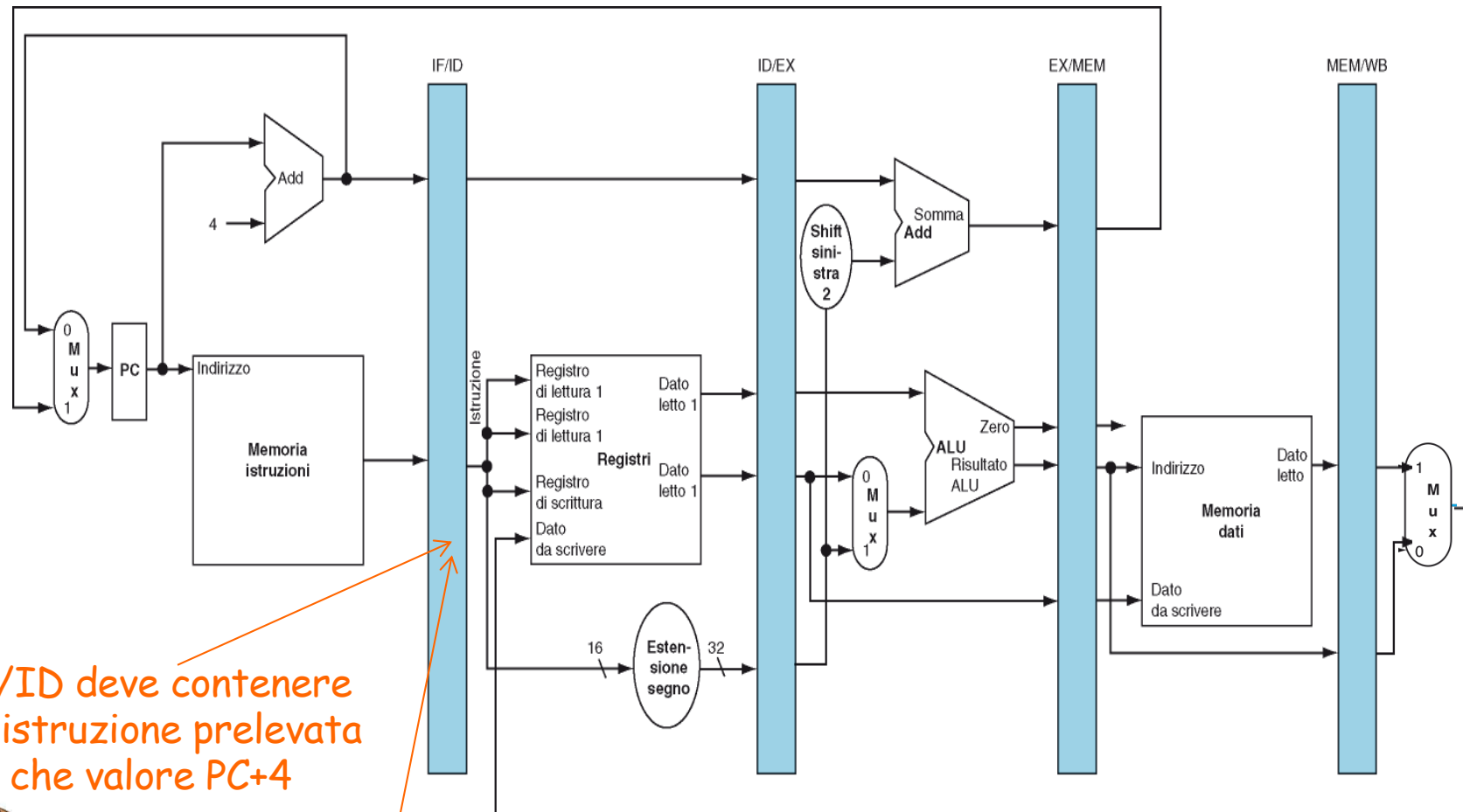


L'unità di Elaborazione con pipeline

- Per consentire la condivisione delle varie unità funzionali tra i vari stadi della pipeline, aggiungiamo dei registri per salvare i dati (**registri di pipeline o registri interstadio**)
 - Ad ogni ciclo di clock le informazioni procedono da un registro di pipeline al successivo
 - I registri devono avere **ampiezza sufficiente** a memorizzare tutti i dati necessari
- Il nome del registro è dato dai nomi dei due stadi che esso separa
 - Registro IF/ID (Instruction Fetch / Instruction Decode)
 - Registro ID/EX (Instruction Decode / EXecute)
 - Registro EX/MEM (EXecute / MEMory Access)
 - Registro MEM/WB (MEMory Access / Write Back)



L'unità di Elaborazione con pipeline



IF/ID deve contenere
sia istruzione prelevata
che valore PC+4

IF/ID deve
essere ampio 64 bit



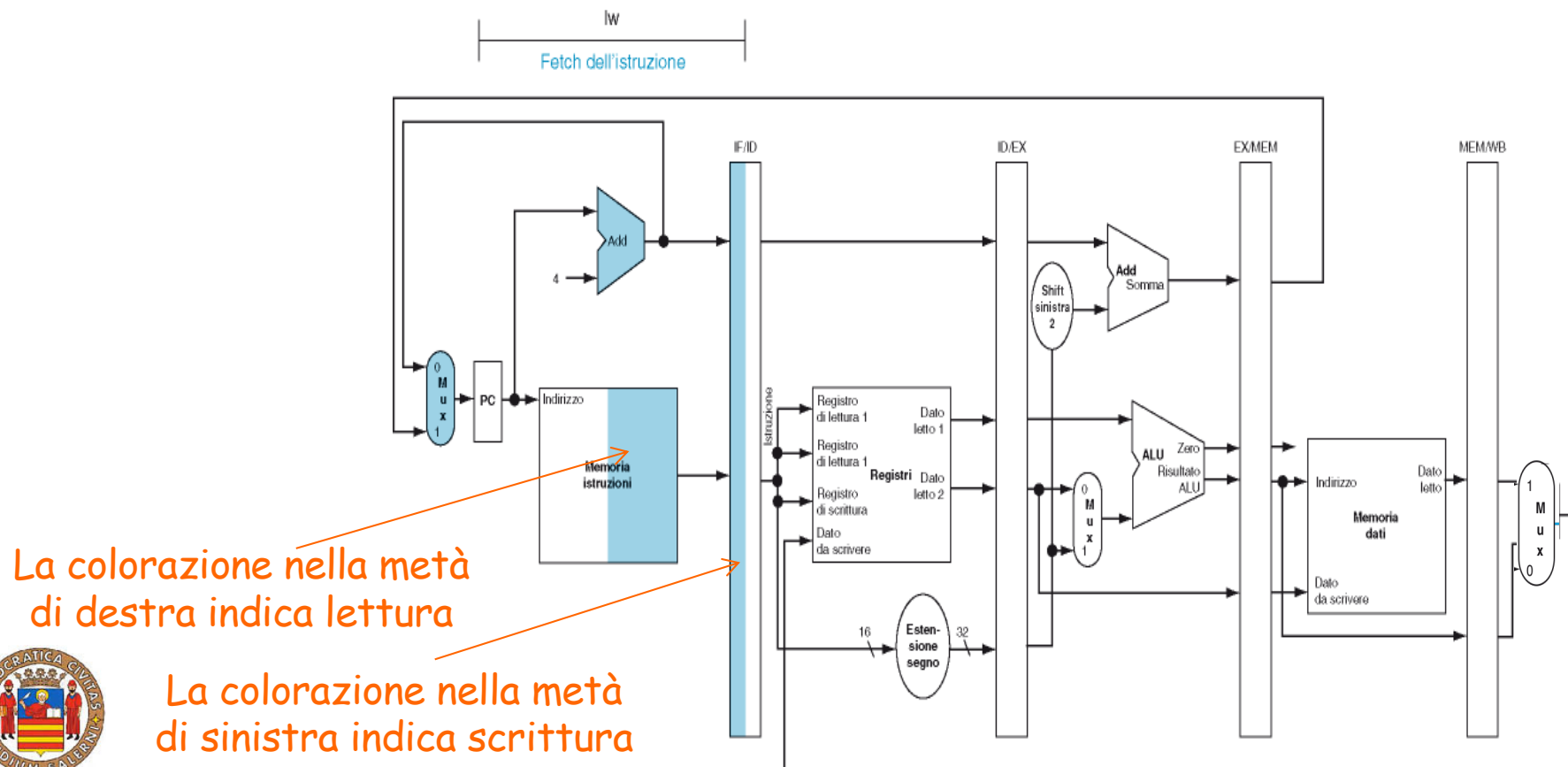
L'unità di Elaborazione con pipeline

- Vediamo come viene eseguita l'istruzione **lw** nei vari stadi della pipeline
 - **IF** (**I**nstruction **F**etch): Prelievo istruzione ed incremento PC
 - **ID** (**I**nstruction **D**ecode): Decodifica istruzione e lettura registri
 - **EX** (**EX**ecution): Esecuzione operazione / calcolo indirizzo
 - **MEM** (**MEM**ory): Accesso in memoria
 - **WB** (**W**rite **B**ack): Scrittura banco registri



Esecuzione di lw: primo stadio

- L'istruzione viene **prelevata (letta)** dalla memoria istruzioni, usando il contenuto del PC, e **scritta** nel registro **IF/ID**
- Il **PC** viene **aggiornato a PC+4** e il nuovo valore viene **scritto** in **IF/ID**



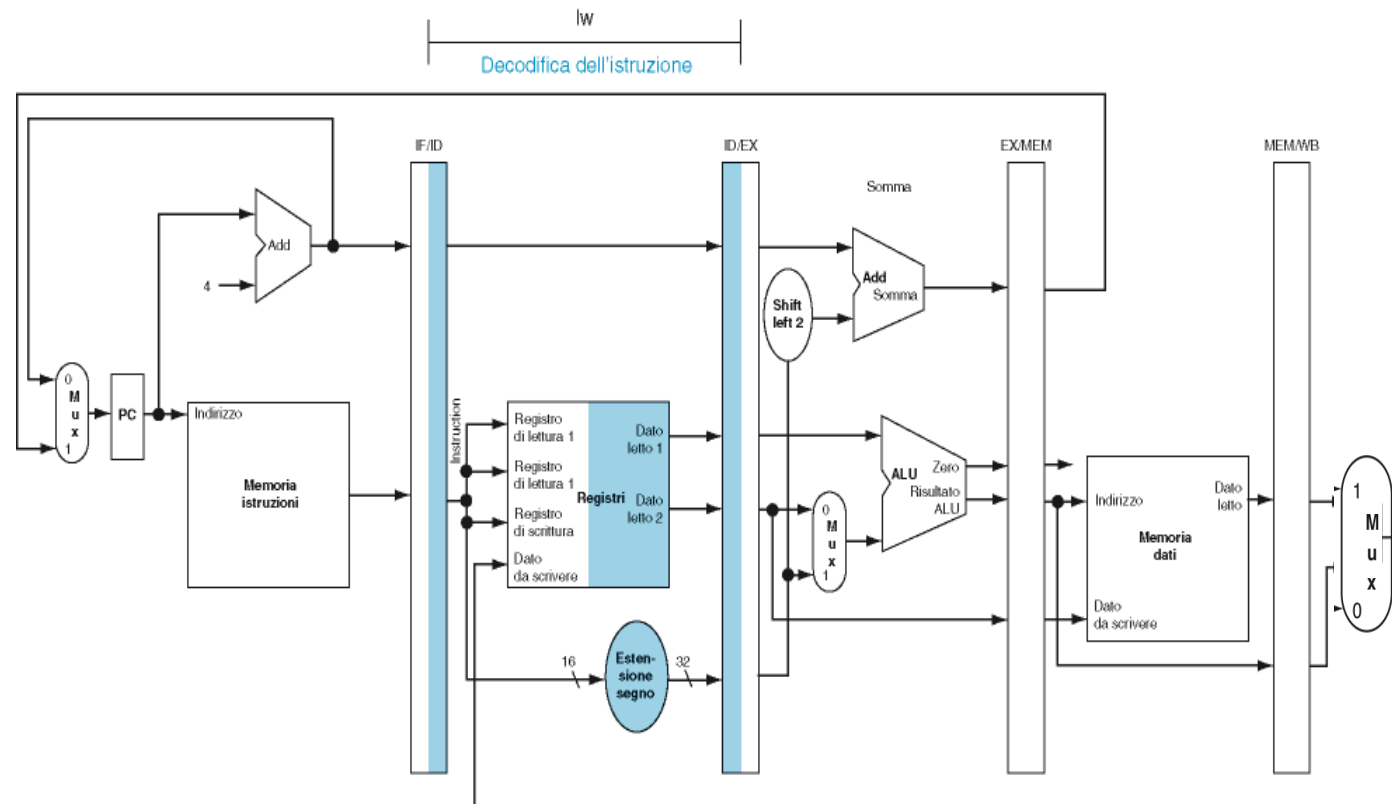
Esecuzione di lw: primo stadio

- **Nota:** L'aggiornamento del PC in questo stadio consente di poter prelevare l'istruzione successiva al prossimo ciclo di clock
 - Tale istruzione si troverà in memoria istruzioni all'indirizzo dato dal contenuto del PC incrementato di 4
- **Nota:** Il valore $PC+4$ viene scritto anche in **IF/ID** perché potrebbe servire successivamente
 - Poiché nello stadio di fetch il processore non sa ancora se l'istruzione corrente è una **beq**, prepara i dati per l'esecuzione di tutte le istruzioni possibili e li trasporta lungo la pipeline
 - Se si tratta di una **beq**, successivamente il dato $PC+4$ potrebbe servire per il calcolo dell'indirizzo di salto
 - Se $PC+4$ non viene salvato da qualche parte, dopo non sarà più possibile recuperarlo (perché al ciclo di clock successivo PC sarà nuovamente aggiornato)



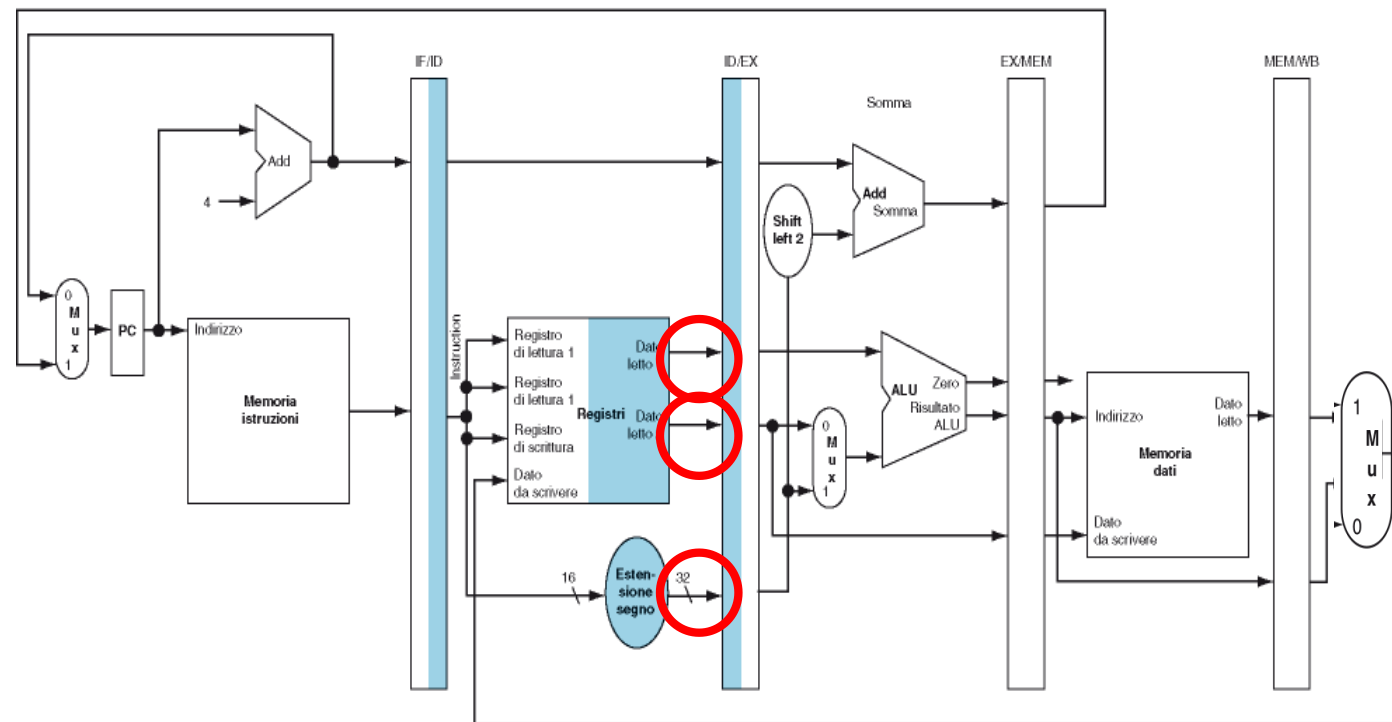
Esecuzione di lw: secondo stadio

- Il campo offset e i numeri dei due registri da leggere vengono **letti** dal registro **IF/ID**, contenente l'istruzione prelevata
- I due dati **letti** dai registri e l'offset esteso a 32 bit vengono **scritti** nel registro **ID/EX**



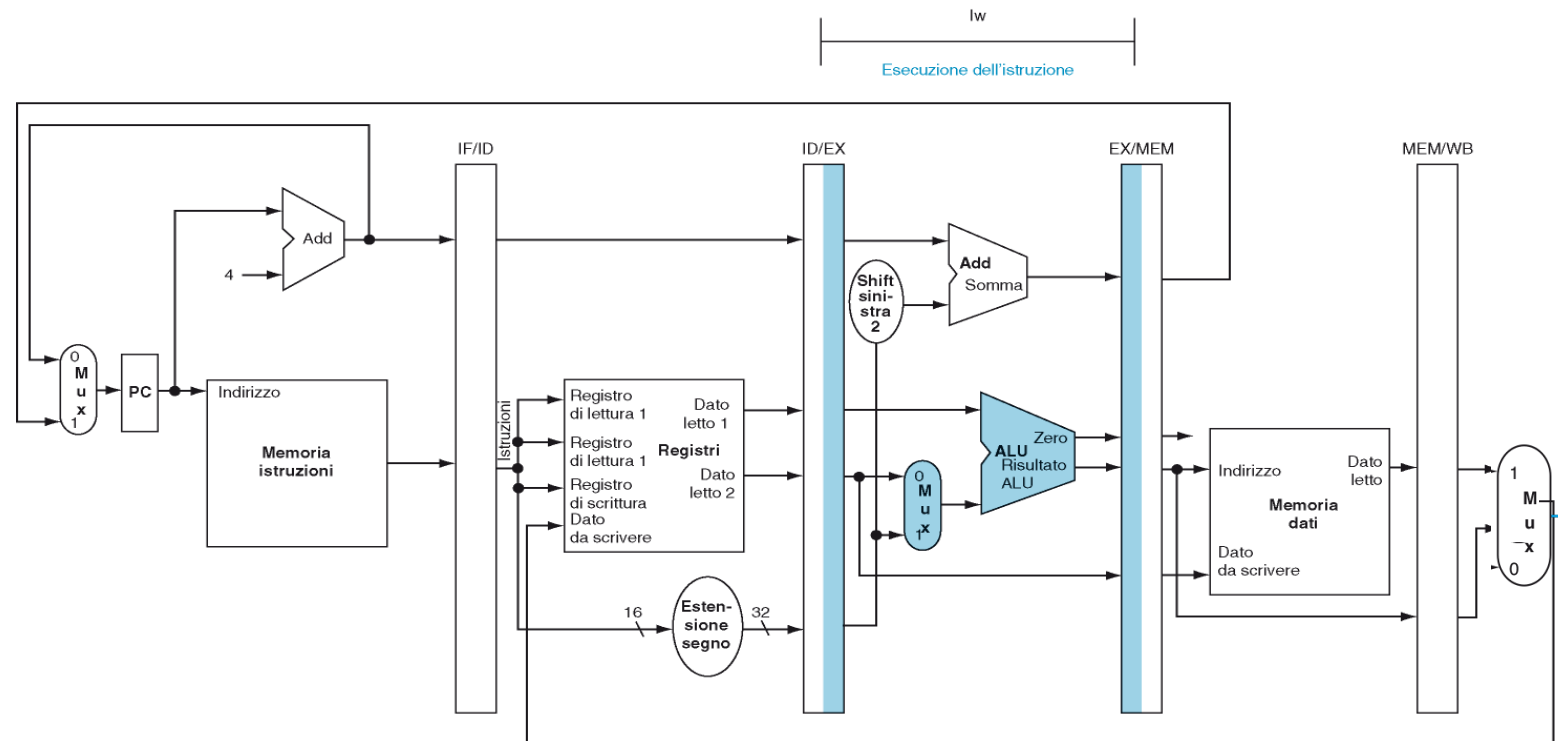
Esecuzione di lw: secondo stadio

- **Nota:** nel registro ID/EX vengono scritti sia il campo offset esteso a 32 bit che i numeri dei due registri da leggere!
- Ma la load richiede la lettura di un solo registro!
- Tuttavia, l'implementazione del processore è più semplice se tutti e tre i dati vengono scritti in ID/EX



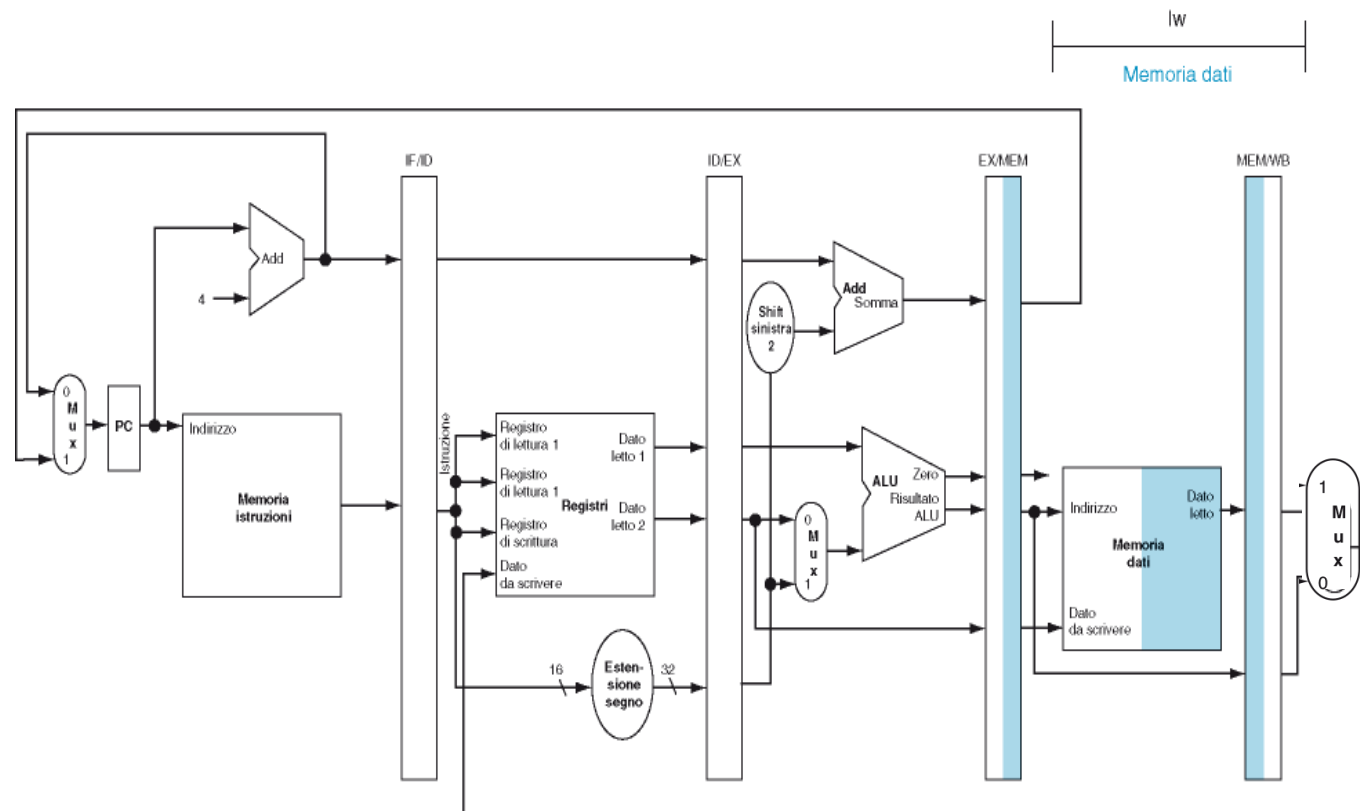
Esecuzione di lw: terzo stadio

- I due dati provenienti dai registri e l'offset esteso a 32 bit vengono **letti** dal registro **ID/EX**
- Viene **effettuata la somma** tramite l'ALU del contenuto del primo registro e dell'offset esteso a 32 bit
- Il risultato viene **scritto** nel registro **EX/MEM**



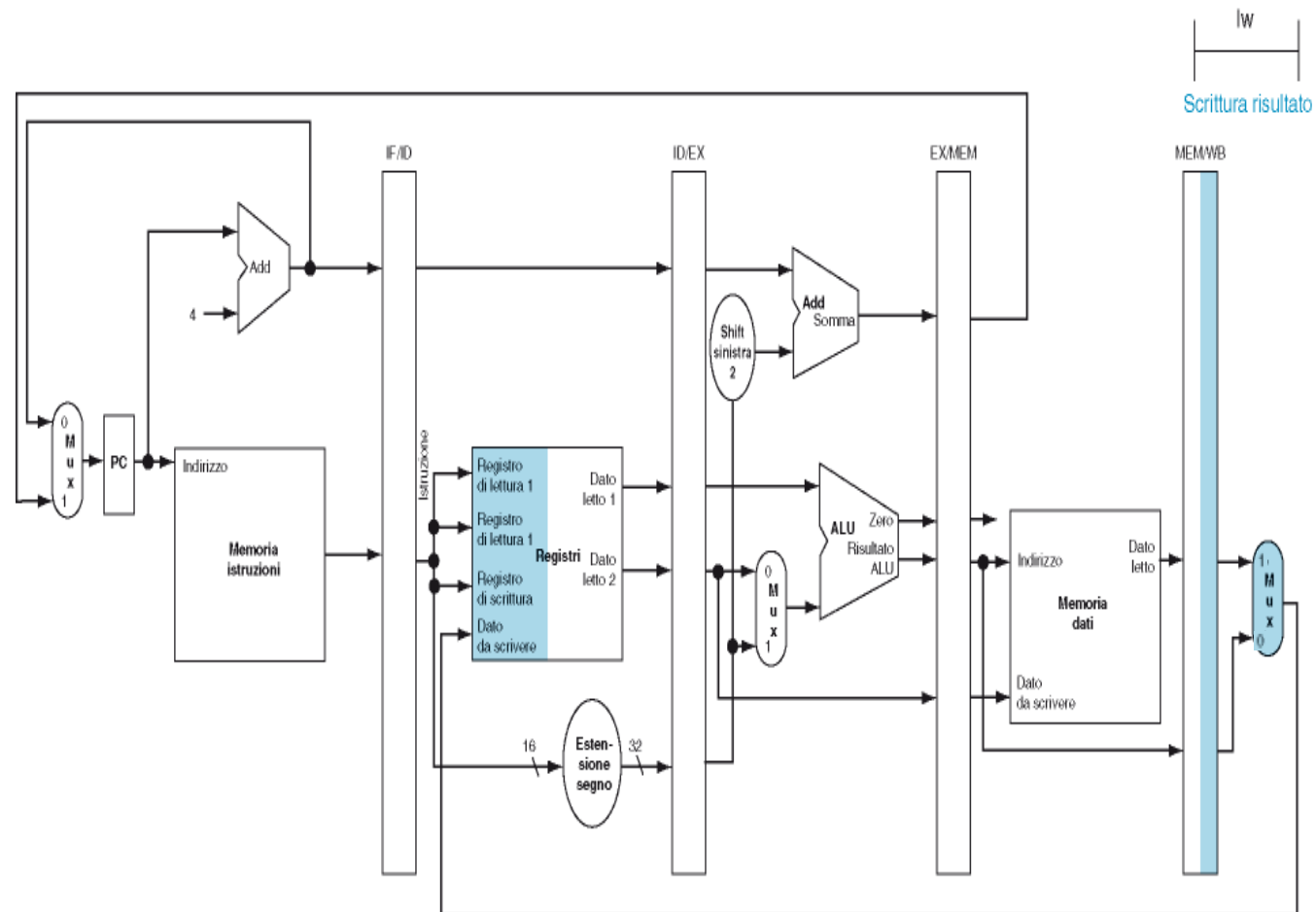
Esecuzione di lw: quarto stadio

- L'indirizzo a cui accedere nella memoria dati viene **letto** dal registro **EX/MEM**
- Il dato letto in memoria viene **scritto** nel registro **MEM/WB**

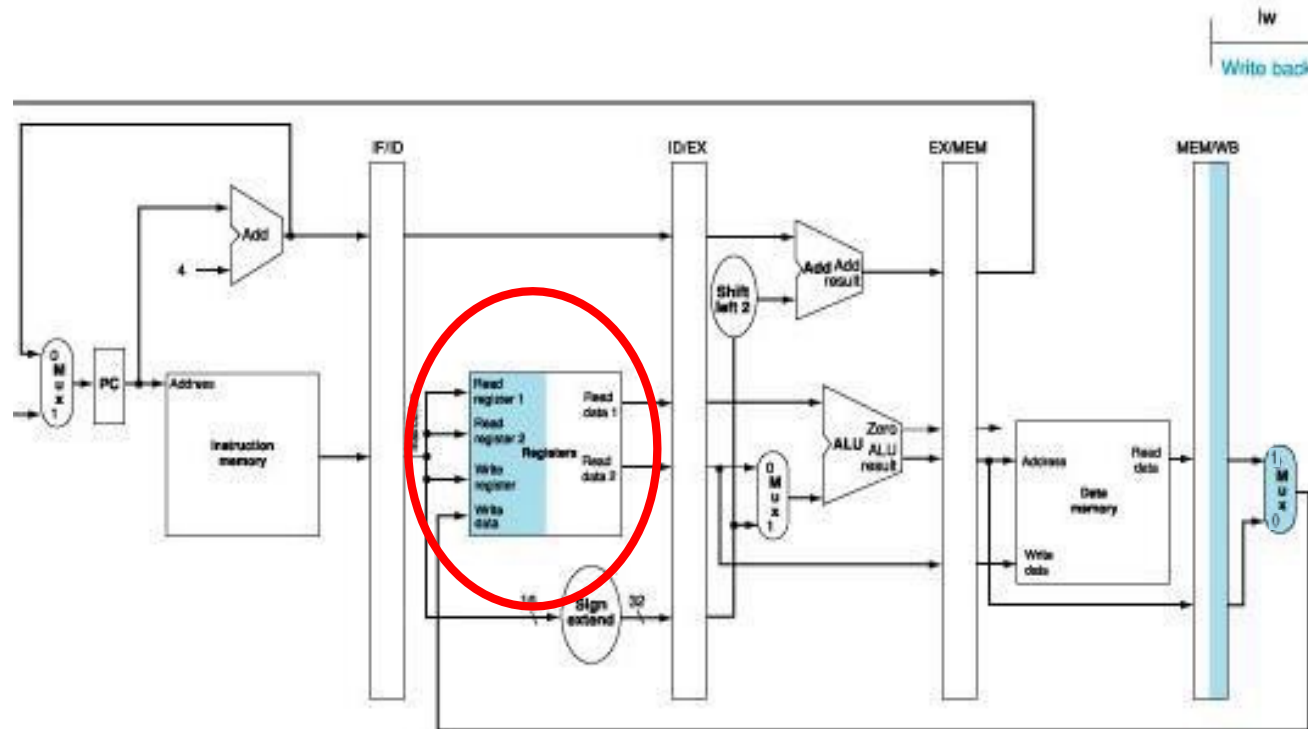


Esecuzione di lw: quinto stadio

- Il dato viene **letto** dal registro **MEM/WB** e **scritto** nel register file



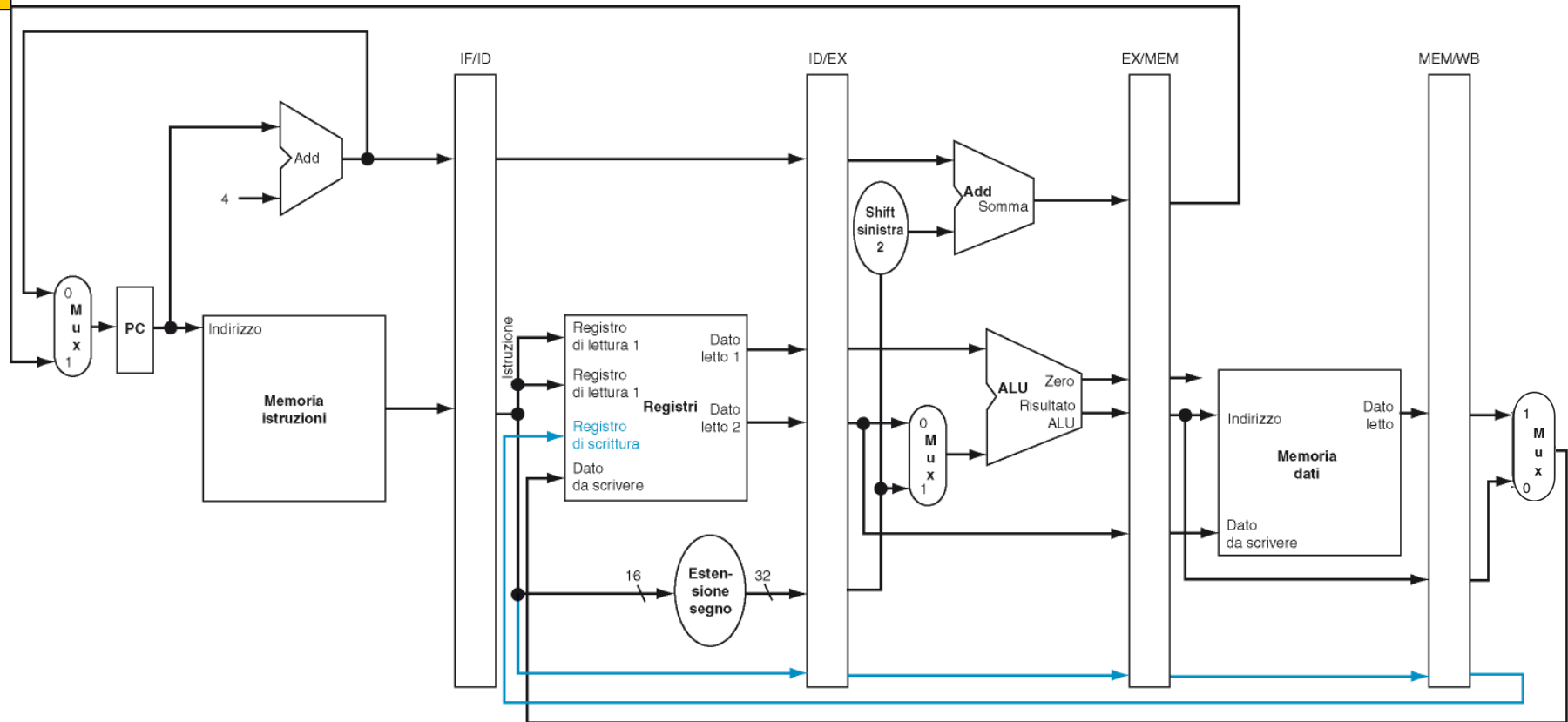
Caccia all'errore



- Quale **registro di destinazione** viene scritto?
- Il numero del registro da scrivere è contenuto nel registro **IF/ID**
- Ma tale registro contiene un'istruzione successiva alla **lw**
- Soluzione: **preservare il numero del registro di destinazione** di **lw**



Soluzione



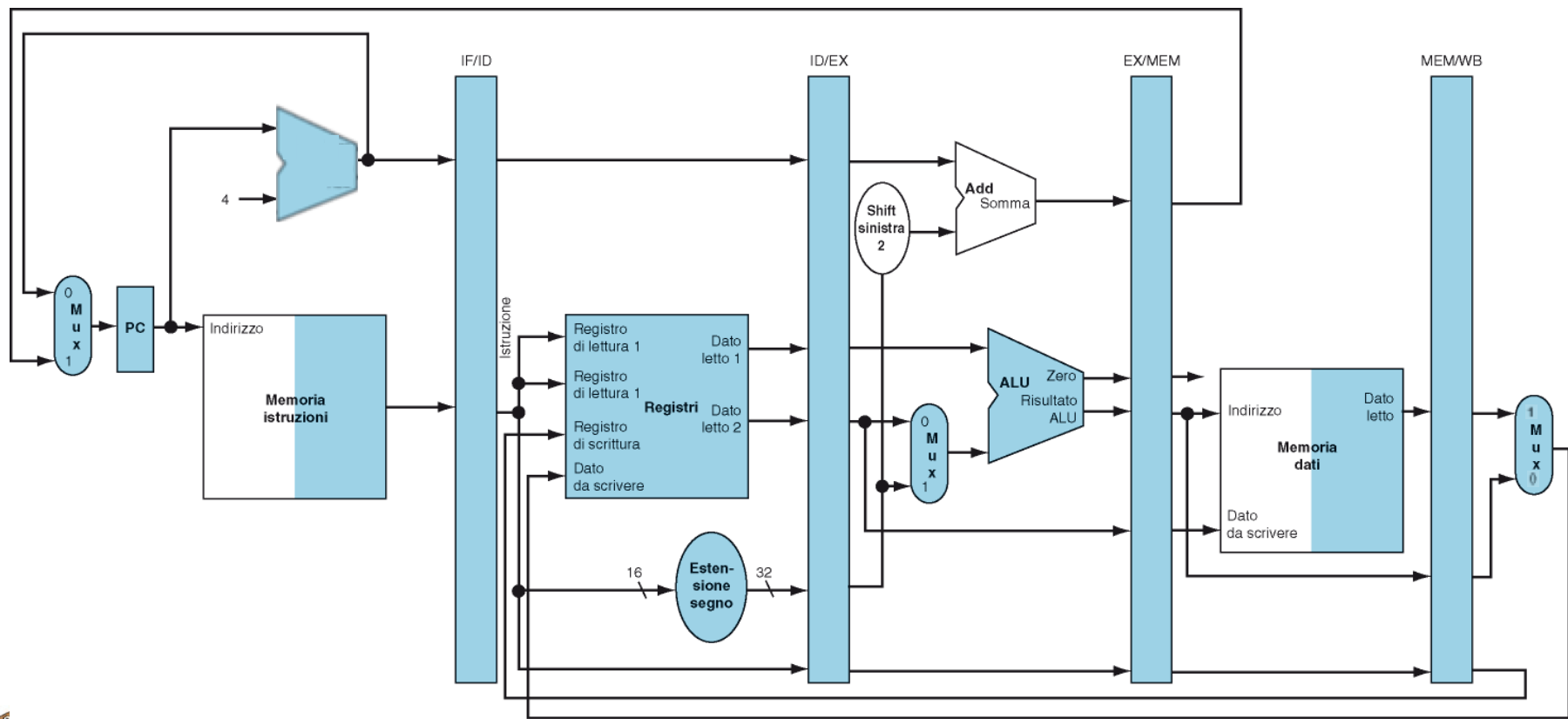
➤ Il numero del registro di destinazione viene scritto/trasportato nei registri di pipeline

➤ Prima in **ID/EX**, poi in **EX/MEM**, infine in **MEM/WB**



Schema corretto per lw

➤ Mettendo insieme i 5 stadi della pipeline per l'istruzione **lw** si ha



L'Unità di controllo con pipeline

- Aggiungiamo l'unità di controllo all'unità di elaborazione con pipeline
 - Cerchiamo di riutilizzare il più possibile l'Unità di Controllo dell'implementazione a ciclo singolo
 - In particolare, l'Unità di Controllo della ALU, i multiplexer e la logica di controllo per i salti saranno gli stessi
- Innanzitutto notiamo che
 - I registri di pipeline IF/ID, ID/EX, EX/MEM, MEM/WB, così come il registro PC, sono scritti ad ogni ciclo di clock
 - Non sono quindi necessari segnali di controllo per la scrittura di tali registri



L'Unità di controllo con pipeline

- Raggruppiamo i segnali di controllo in base agli stadi della pipeline e vediamo quali devono essere impostati
 - **Prelievo dell'istruzione**
 - La memoria istruzioni viene letta ad ogni ciclo di clock e il **PC viene scritto ad ogni ciclo di clock**: **non c'è niente da controllare** in questo stadio
 - **Decodifica dell'istruzione/lettura del register file**
 - Il register file viene letto ad ogni ciclo di clock: **non c'è niente da controllare**
 - **Esecuzione/calcolo dell'indirizzo**
 - I segnali da impostare sono **RegDst**, **ALUOp**, **ALUSrc**
 - **Accesso alla memoria**
 - I segnali da impostare sono **Branch**, **MemRead**, **MemWrite**
 - **Scrittura del risultato**
 - I segnali da impostare sono **MementoReg**, **RegWrite**



L'Unità di controllo con pipeline

- Implementare l'unità di controllo significa **impostare il valore dei segnali** per ciascuno stadio della pipeline e per ciascuna istruzione
 - Ciò può essere fatto **salvando i segnali all'interno dei registri di pipeline**
 - I valori necessari allo stadio successivo **sono propagati** dal registro di pipeline corrente a quello successivo

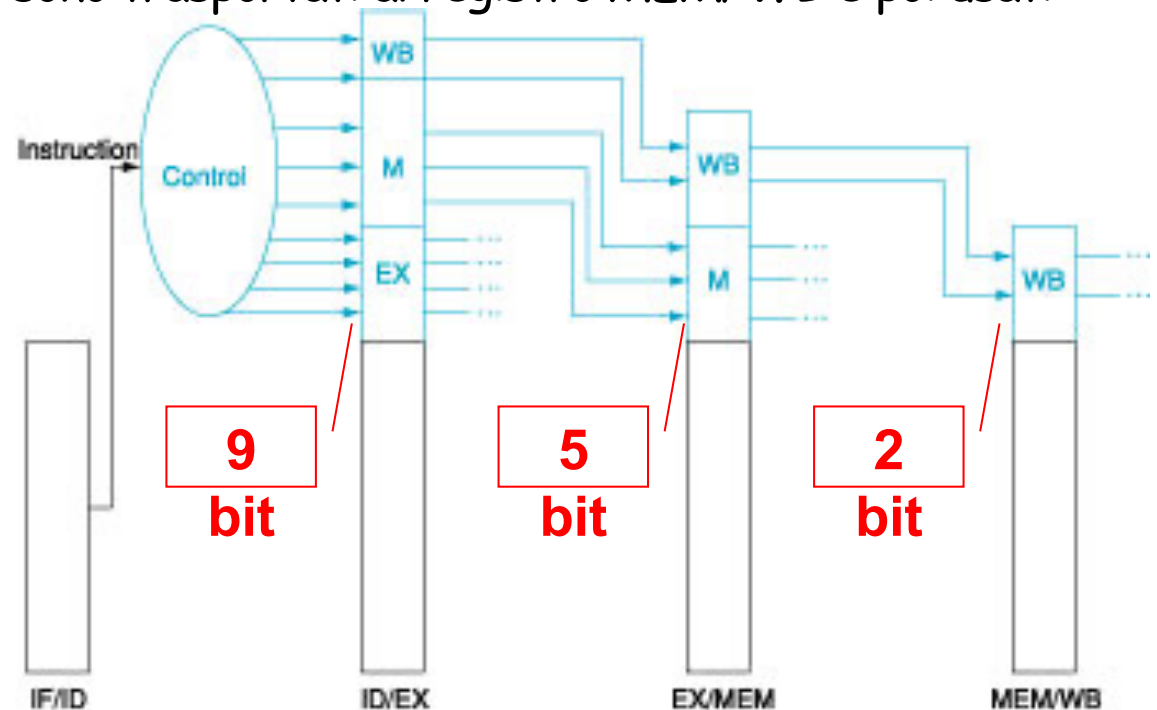
Istruzione	Segnali di controllo dello stadio di esecuzione/calcolo dell'indirizzo				Segnali di controllo dello stadio di accesso alla memoria dati			Segnali di controllo dello stadio di scrittura	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Mem-Read	Mem-Write	RegWrite	Memto-Reg
Formato R	1	1	0	0	0	0	0	1	0
lw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



Sono gli stessi segnali di controllo dell'implementazione a ciclo singolo, ma divisi in tre gruppi, ciascuno attivo in uno degli ultimi tre stadi della pipeline

L'Unità di controllo con pipeline

- Il registro di pipeline ID/EX contiene anche i valori dei segnali di controllo (9 bit)
 - Di questi, 4 sono utilizzati nello stadio EX, mentre i rimanenti sono trasportati al registro di pipeline EX/MEM
 - Dei segnali presenti nel registro EX/MEM, 3 sono utilizzati nello stadio MEM, mentre i rimanenti sono trasportati al registro MEM/WB e poi usati nello stadio WB

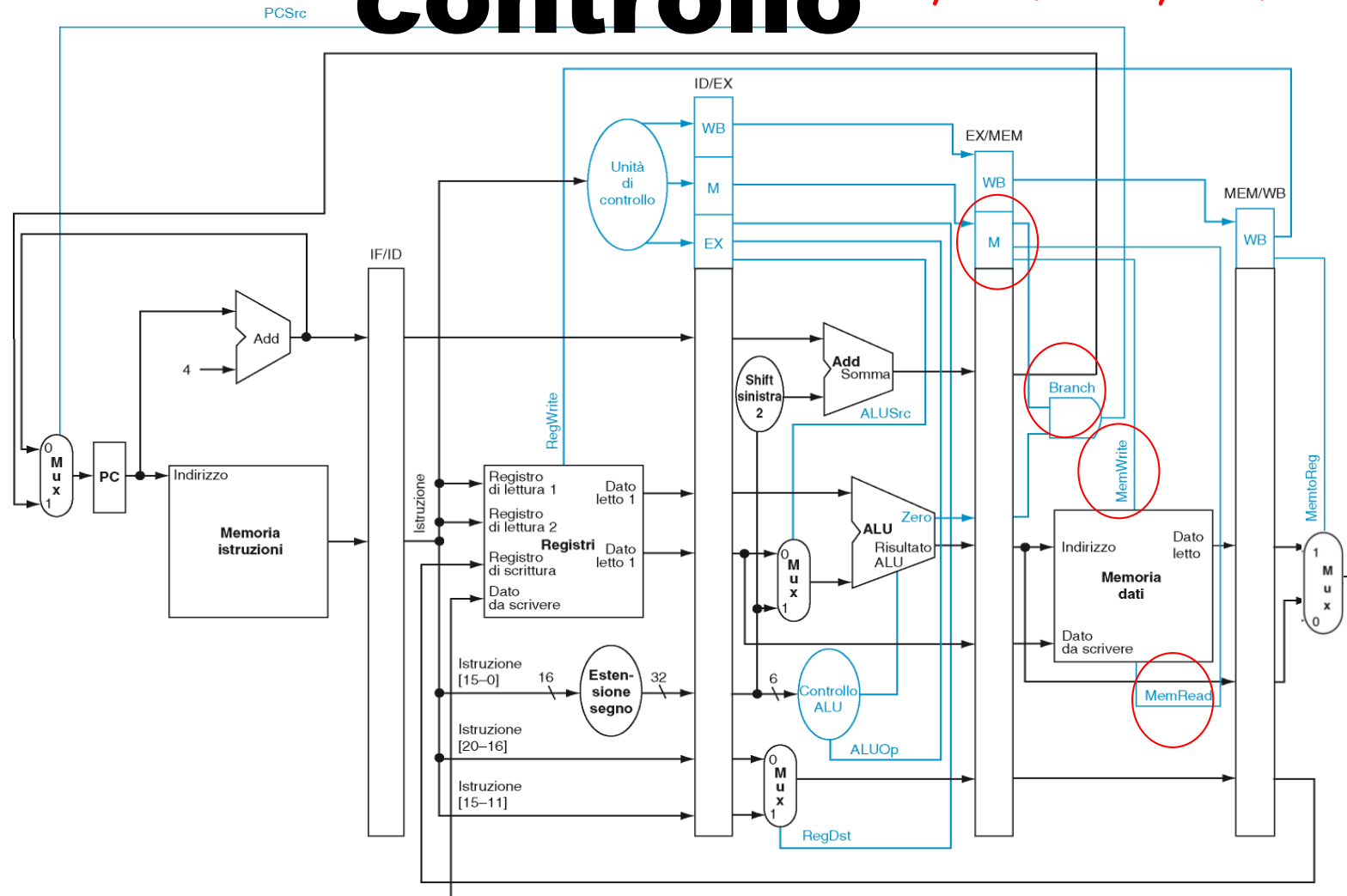


I segnali di controllo usati nello stato EXE: RegDest, AluOp, AluSrc



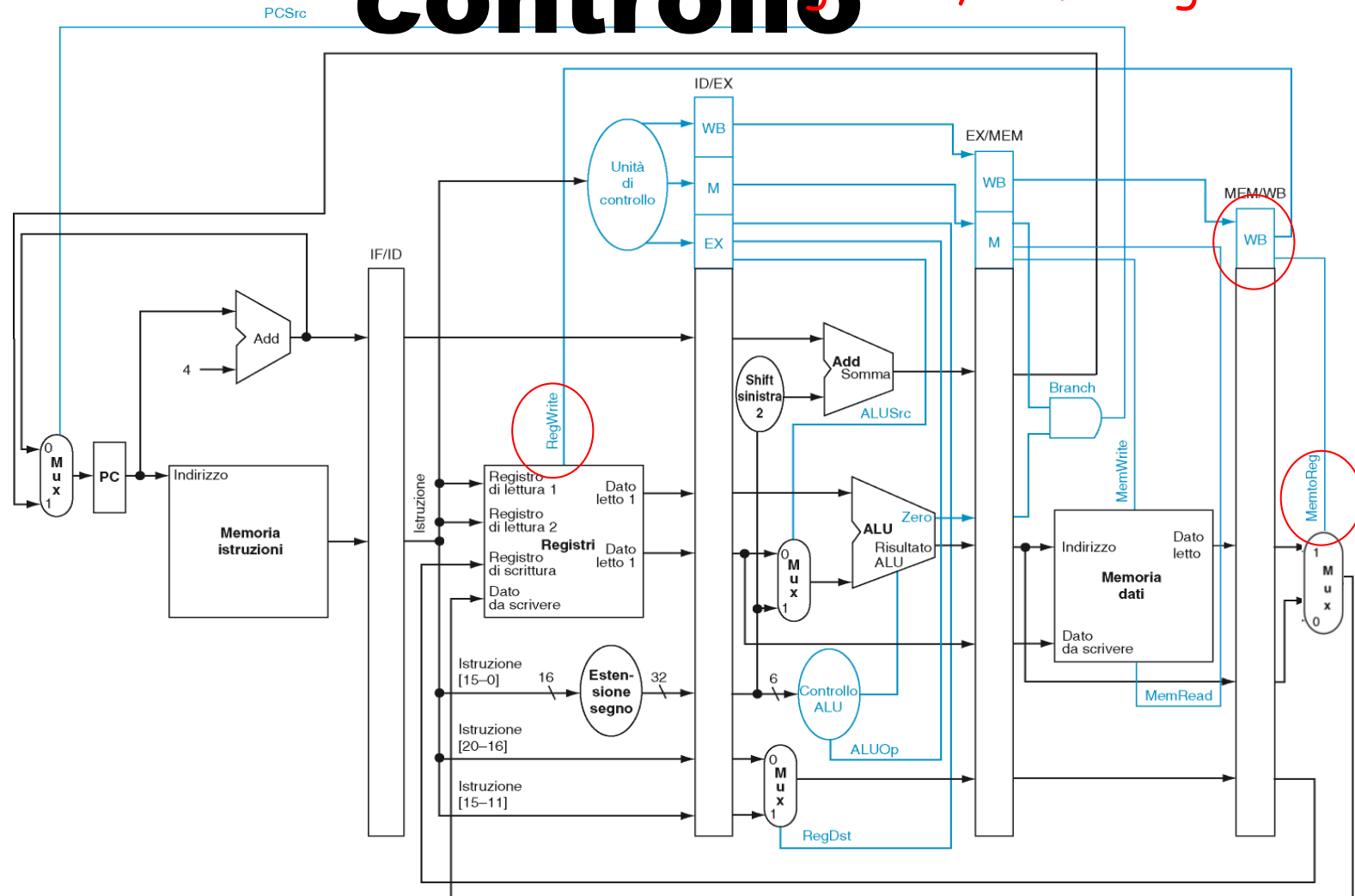
Unità di elaborazione con pipeline e

I segnali di controllo usati nello stato MEM: Branch, MemRead, MemWrite



Unità di elaborazione con pipeline e

I segnali di controllo usati nello stato WB: RegWrite, MemtoReg



Riepilogo e riferimenti

- Abbiamo visto una **seconda implementazione per il MIPS**, basata su **pipeline**
 - Introduzione alla pipeline: [PH] par. 4.5 (esclusi gli hazard nelle pipeline)
 - L'unità di Elaborazione con pipeline e l'Unità di Controllo associata: [PH] par. 4.6
- Nella prossima lezione
 - Studieremo le **criticità (hazard)** che sorgono in una pipeline quando non è possibile eseguire l'istruzione successiva nel ciclo di clock successivo
 - Analizzeremo alcune **soluzioni** per risolvere tali criticità

