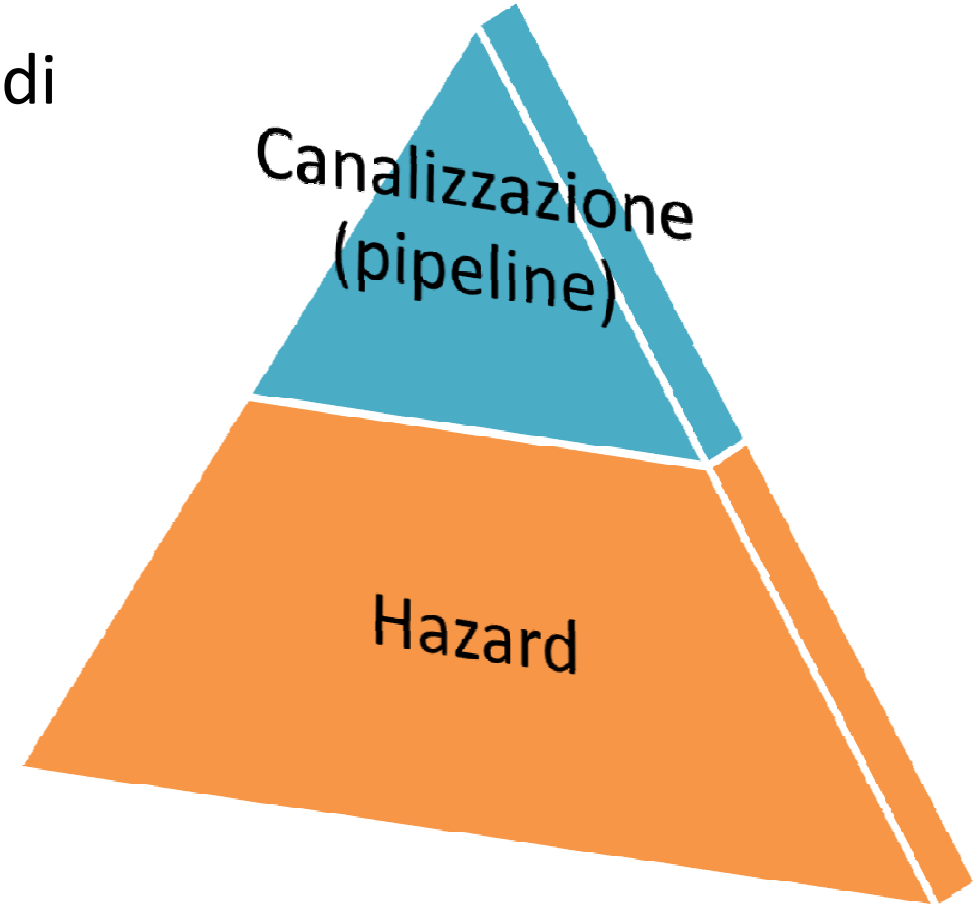
Architettura degli elaboratori

Canalizzazione/Pipeline

Dott. Franco Liberati

ARGOMENTI DELLA LEZIONE

- Miglioramenti Macchina di Von Neumann
  - Canalizzazione o pipeline
  - Hazard



Architettura degli elaboratori

PIPELINE

PIPELINE

Generalità

- Il **pipeline**, o **canalizzazione**, è una tecnica che consiste nella scomporre una rete logica (combinatoria) in una serie di reti logiche più semplici mediante l'inserimento di opportuni **registri di disaccoppiamento (interlock)**
- Questo accorgimento permette di aumentare la frequenza di clock e svolgere più operazioni in **parallelo**
- L'esecuzione di una istruzione, in una macchina è di solito suddivisa in **fasi** (o **sezioni** o ***stage***):
  - **F - Fetch**: prelievo istruzione (con incremento del Program Counter)
  - **L - Load**: prelievo operandi
  - **D - Decode**: decodifica/riconoscimento istruzione
  - **E - Execution**: esecuzione istruzione
  - **M - Memory**: accesso in memoria per scrittura o lettura (*load* o *store*)

PIPELINE

Generalità RISC

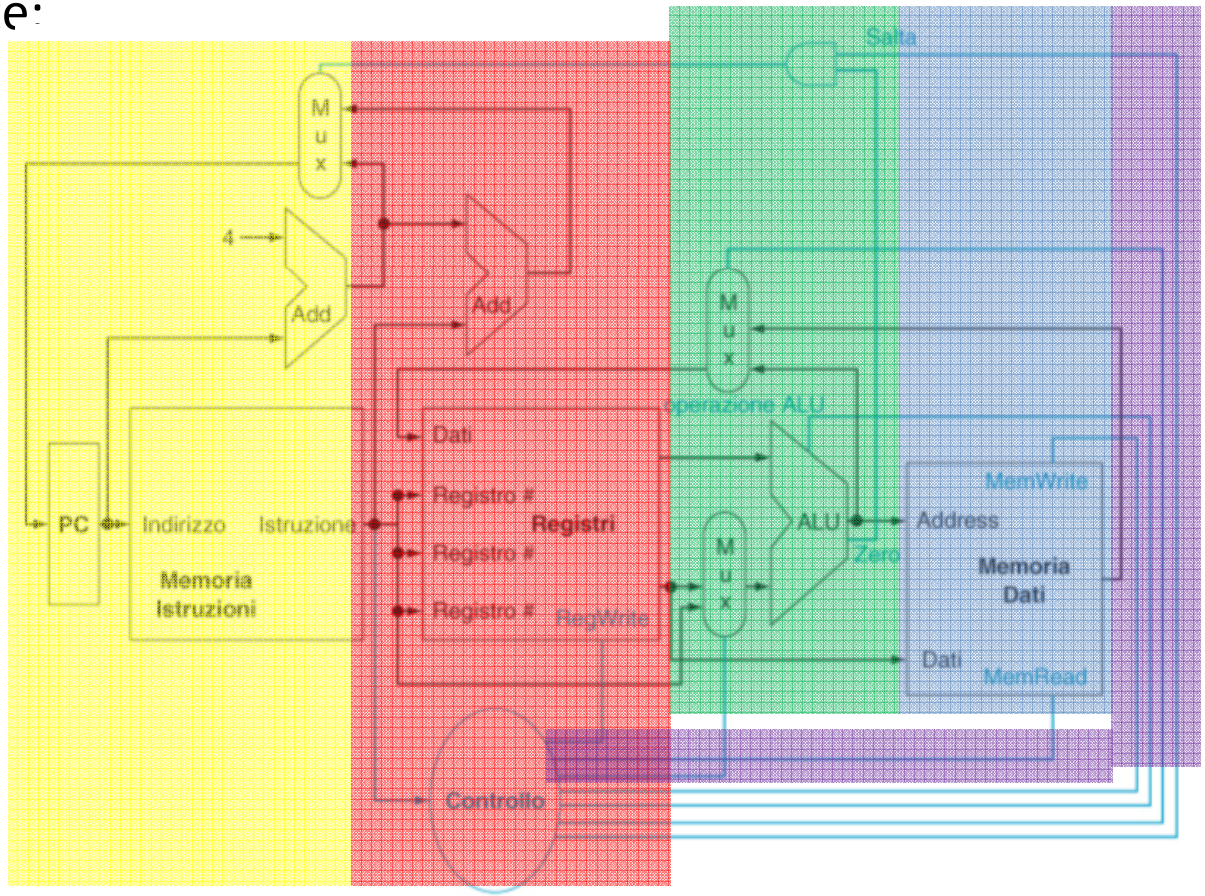
- L'esecuzione di una istruzione in una macchina RISC come il MIPS ha una suddivisione in cinque **fasi**:
  - **F - Fetch**: prelievo istruzione (con incremento del Program Counter)
  - **D - Decode**: decodifica/riconoscimento istruzione
  - **E - Execution**: esecuzione istruzione
  - **M - Memory**: accesso in memoria per scrittura o lettura (*load* o *store*)
  - **WB - Write Back**: quando il risultato dell'ALU o quello letto dalla memoria viene messo nel registro destinazione

PIPELINE

Macchina senza pipeline

In una macchina senza pipeline si evidenzia che ad ogni momento **solo una unità funzionale è attiva** e le altre non sono impegnate:

- **Fetch**: Memoria Istruzioni (e aggiornamento PC)
- **Decode**: Blocco registri (e CU)
- **Execute**: ALU
- **Memory**: Memoria dati
- **Write Back**: Blocco registri



PIPELINE

Macchina senza pipeline

In ogni momento **solo una unità funzionale è attiva** e le altre non sono impegnate:

- **Fetch**: Memoria Istruzioni (e aggiornamento PC)
- **Decode**: Blocco registri (e CU)
- **Execute**: ALU
- **Memory**: Memoria dati
- **Write Back**: Blocco registri

Esecuzione sequenziale																								
Δt	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Istruzione 1	F	D	E	M	WB																				
Istruzione 2						F	D	E	M	WB															
Istruzione 3											F	D	E	M	WB										
Istruzione 4																F	D	E	M	WB					
Istruzione 5																					F	D	E	M	WB

PIPELINE

Macchina con pipeline

In una **macchina canalizzata**, cioè con pipeline, ogni unità funzionale elabora la fase che gli corrisponde e poi passa all'istruzione successiva.

Esecuzione sequenziale																				
Δt	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Istruzione 1	**F**	**D**	**E**	**M**	**WB**																
Istruzione 2		**F**	**D**	**E**	**M**	**WB**															
Istruzione 3			**F**	**D**	**E**	**M**	**WB**														
Istruzione 4				**F**	**D**	**E**	**M**	**WB**													
Istruzione 5					**F**	**D**	**E**	**M**	**WB**												

In questo modo una volta che la canalizzazione arriva **a regime**, cioè quando tutte le unità funzionali sono occupate e cioè si svolgono fino a *n* istruzioni contemporaneamente in *n* fasi diverse, alla fine di una fase si completa una istruzione

PIPELINE

Tempi delle fasi

- La canalizzazione, per funzionare adeguatamente, richiede che le varie unità siano sincronizzate e che i dati delle varie istruzioni non si sovrappongano, pertanto è necessario imporre che tutte quante le fasi **abbiano la stessa durata** (stabilita valutando tutte le istruzioni e analizzando i relativi tempi delle fasi)
- Questo determina un **clock totale più lungo** (si attribuisce a ciascuna fase il tempo della classe di istruzione più dispendiosa temporalmente; c'è un periodo più lungo); ma il tempo complessivo di esecuzione del programma è migliorato (frequenza maggiore una volta a regime)

PIPELINE

Incremento della velocità

- Per determinare il **periodo di clock uniforme** che permette di svolgere le fasi in modo da poterle sovrapporre si deve individuare l'istruzione - o meglio la classe di istruzione - più lenta. In pratica si cerca sempre di progettare la CPU in modo che le fasi abbiano tempi simili (nell'esempio sottostante il tempo di fase è 200ps)

Classe Istruzione	Fetch dell'istruzione	Lettura registri	ALU	Accesso in memoria	Scrittura nel registro	TOTALE
LOAD	200ps	100ps	200ps	200ps	100ps	800ps
STORE	200ps	100ps	200ps	200ps		700ps
FORMATO R	200ps	100ps	200ps		100ps	600ps
SALTO CONDIZIONATO	200ps	100ps	200ps			500ps

- Si può ridurre il periodo di clock da 800ps (durata massima di una istruzione) a 200ps (durata massima di una fase), ovvero quadruplicare la velocità del clock

PIPELINE

Esempio

Macchina non canalizzata (tempo=4·800ps=3200ps)	
lw $t0,batman	
lw $t1,superman	
add $t2,$t0,$t1	
sw $t2,joker	

Macchina canalizzata (tempo=800ps+200ps+200ps+200ps=1400ps)	
lw $t0,batman	
lw $t1,superman	
add $t2,$t0,$t1	
sw $t2,joker	

PIPELINE

Incremento della velocità

- **In generale** per determinare il **periodo di clock** si individua la fase più lenta di ciascuna istruzione e si uniformano a quel clock tutte le fasi delle diverse istruzioni
- La singola istruzione sarà computata in un tempo più lungo, ma grazie alla canalizzazione ad ogni colpo di clock si avrà, una volta a regime, l'elaborazione di una istruzione e un guadagno pari al numero di fasi

Macchina canalizzata									
I1									
I2									
I3									
I4									

PIPELINE

Applicazione e guadagno

- Il guadagno offerto da una canalizzazione è, una volta “a regime”, dato del numero delle fasi elementari legate alla esecuzione dell’istruzione (in un macchina con cinque fasi il guadagno è di cinque volte)
- In alcune macchine c’è una canalizzazione fino a 20 fasi che consente uno sfruttamento completo di tutte le unità funzionali in gioco (es.: l’accesso a delle memorie ausiliare come la *cache*)
  - L’Intel Core 2 Duo ha 14 fasi di canalizzazione
- Nei calcolatori il **compilatore** assume un ruolo fondamentale: deve infatti presiedere alla suddivisione delle istruzioni complesse in gruppi di istruzioni semplici ed ottimizzare la sequenza di esecuzione, sfruttando appieno i vantaggi della pipeline
- La canalizzazione, infatti, è un accorgimento **trasparente** al programmatore

PIPELINE

Canalizzazione nelle macchine CISC

- L'implementazione di pipeline su **calcolatori CISC** risulta difficoltosa a causa della intrinseca complessità delle istruzioni
- L'esecuzione di una istruzione, nelle macchine CISC, è di solito suddivisa in sei fasi (o sezioni) la cui durata è molto variabile:
  - **F – Fetch**: prelievo istruzione (con incremento del Program Counter)
  - **L – Load**: prelievo operandi dalla memoria (tipico delle istruzioni con modi di indirizzamento complesso)
  - **D – Decode**: decodifica/riconoscimento istruzione
  - **E – Execution**: esecuzione istruzione
  - **M – Memory**: accesso in memoria per scrittura o lettura (*load* o *store*)
  - **WB – Write Back**: quando il risultato dell'ALU o quello letto dalla memoria viene messo nel registro destinazione

PIPELINE

Canalizzazione nelle macchine CISC

- La fase Fetch può sovrapporsi alla fase Decode nelle macchine CISC solamente se l'istruzione ha **dimensione fissa** e quindi è possibile calcolare il prossimo valore del Program Counter senza sapere la classe di istruzione
- Altrimenti nelle architetture CISC con **istruzioni a dimensione variabile**, oltre alla presenza di una eventuale fase di **Load** (recupero degli operandi), la fase di **decodifica** è variabile così come quelle di esecuzione e scrittura dei dati

PIPELINE

Canalizzazione in RISC e CISC (differenze)

Esecuzione RISC																				
Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Istruzione 1	F	D	E	M	WB																
Istruzione 2		F	D	E	M	WB															
Istruzione 3			F	D	E	M	WB														
Istruzione 4				F	D	E	M	WB													
Istruzione 5					F	D	E	M	WB												

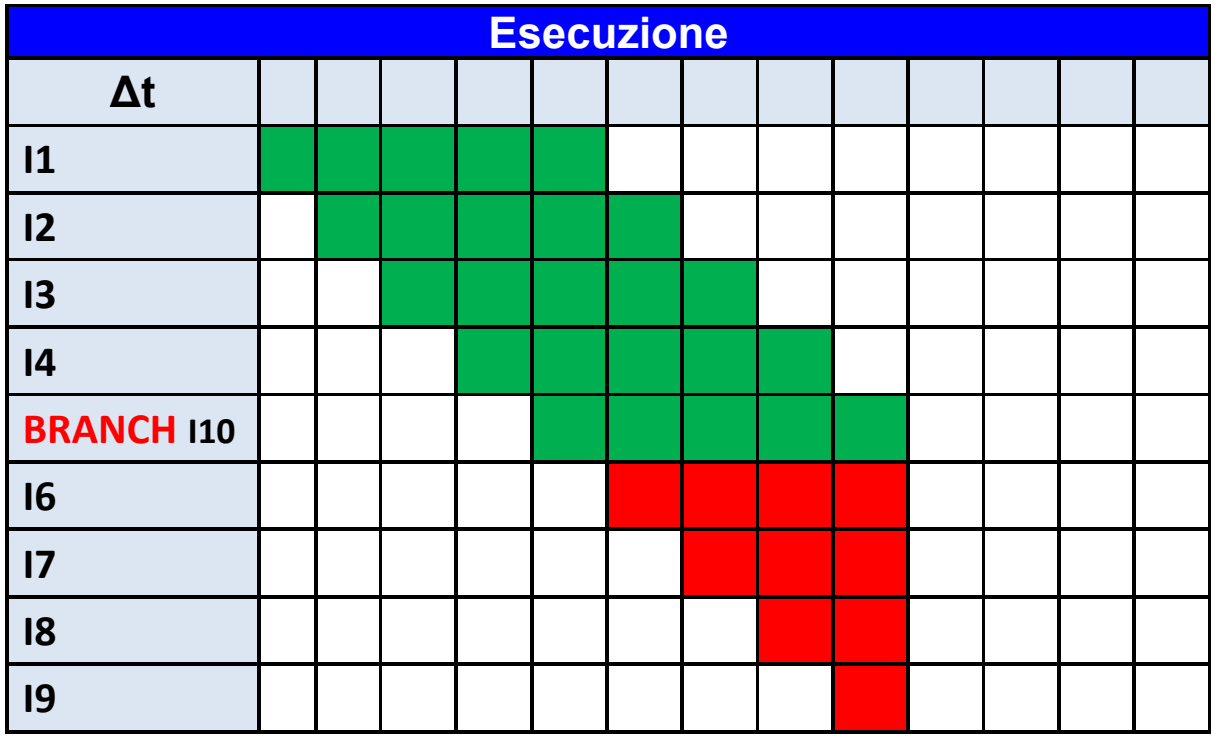
Esecuzione CISC																				
Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Istruzione 1	F	L	L	L	D	D	E	E	M	WB											
Istruzione 2				F	L	L	L	D	D	E	M	WB									
Istruzione 3							F	L	L	D	D	D	D	E	M	M	WB				
Istruzione 4										F	F	L	L	L	D	E	M	WB			
Istruzione 5												F	F	L	L	L	D	D	E	M	WB

NB: non ci devono essere fasi uguali sivrapposte

PIPELINE

Degrado delle prestazioni: hazard

- La regolarità delle istruzioni permette di utilizzare e ottimizzare il funzionamento della pipeline, fino a quando non si verifica una **criticità nella esecuzione (hazard)**
- Ad esempio nel caso di un salto è necessario svuotare il pipe per far si che siano caricate le istruzioni che fanno riferimento dal punto di arrivo del salto in poi



PIPELINE
HAZARD

- Le **criticità nella esecuzione** (**hazard**) possono essere:
  - **SUL CONTROLLO (*control hazard*)**: un salto cambia il flusso sequenziale di esecuzione delle istruzioni (jump/beq ma anche una interruzione o un salto a subroutine)
  - **SUI DATI (*data hazard*)**: necessità di inserire una attesa perché il dato che deve essere elaborato in una istruzione non è ancora pronto perché in corso di calcolo nell'istruzione precedente
  - **STRUTTURALI**: le risorse HW non sono sufficienti (es.: l'ALU sta ancora svolgendo una operazione precedente e quindi non è libera)

Esempio data hazard sul registro $s0:

add $s0, $t0, $t1
sub $t2, $s0, $t3

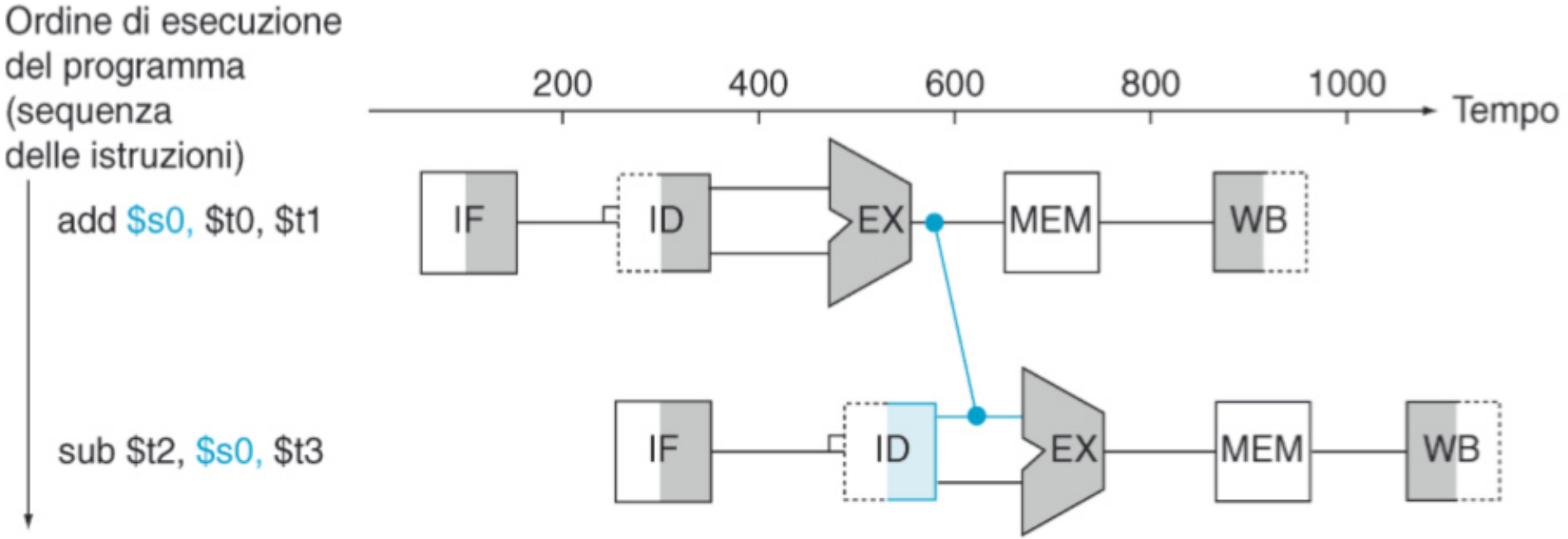
Δt	0	1	2	3	4	5	6	7
I1	F	D	E	M	WB			
I2		F	D	E	M	WB		

La seconda istruzione non può eseguire la lettura degli argomenti se la prima non fa WB (le due fasi **WB** e **D** potrebbero essere sovrapposte se **WB** avviene nella prima metà e **D** nella seconda metà del periodo di clock)

PIPELINE

Soluzioni alle criticità: propagazione (forwarding)

- In alcuni casi l'informazione necessaria è già presente nella pipeline ben prima del **WB**
- Si inseriscono nel datapath delle "scorciatoie" che recapitano il dato all'unità funzionale che ne ha bisogno senza aspettare la fase di **WB**



PIPELINE

Soluzioni alle criticità: stallo

- Se la fase che ha bisogno del dato si trova prima (nel tempo) di quella che lo produce, il forwarding non è possibile e quindi è necessario inserire una attesa (**stallo** o **bolla**)

Δt	**0**	**1**	**2**	**3**	**4**	**5**
lw $t0,20($s0)	F	D	E	M	WB	
add $t1,$t0,$t2		F	D	E	M	WB

Δt	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
lw $t0,20($s0)	F	D	E	M	WB			
add $t1,$t0,$t2		➲	➲	F	D	E	M	WB

PIPELINE

Soluzioni alle criticità: istruzione di stallo

- Possibile implementazione di una bolla: uso dell'istruzione **NOP** (fa perdere due fasi: **F** e **D**)

lw $t0,20($s0)	F	D	E	M	WB	
add $t1,$t0,$t2		F	D	E	M	WB

lw $t0,20($s0)	F	D	E	M	WB							
NOP		F	D									
NOP			F	D								
add $t1,$t0,$t2				F	D	E	M	WB				

PIPELINE

Degrado delle prestazioni: eliminazione stalli

- Si possono **evitare stalli** riordinando le istruzioni ma bisogna mantenere la stessa semantica (possibile risposta ai data hazard)

A=B+E e C=B+F

I1: lw $t1, 0($t0)

I2: lw $t2, 4($t0)

I3: add $t3, $t1, $t2

I4: sw $t3, 12($t0)

I5: lw $t4, 8($t0)

I6: add $t5, $t1, $t4

I7: sw $t5, 16($t0)

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1	F	D	E	M	WB										
I2		F	D	E	M	WB									
I3			➲	➲	F	D	E	M	WB						
I4						F	D	E	M	WB					
I5							F	D	E	M	WB				
I6								➲	➲	F	D	E	M	WB	
I7											F	D	E	M	WB

PIPELINE

Degrado delle prestazioni: eliminazione stalli

- Riordino delle istruzioni con assenza di stalli e semantica corretta

A=B+E e C=B+F

I1: lw $t1, 0($t0)

I2: lw $t2, 4($t0)

I3: lw $t4, 8($t0)

I4: add $t3, $t1, $t2

I5: sw $t3, 12($t0)

I6: add $t5, $t1, $t4

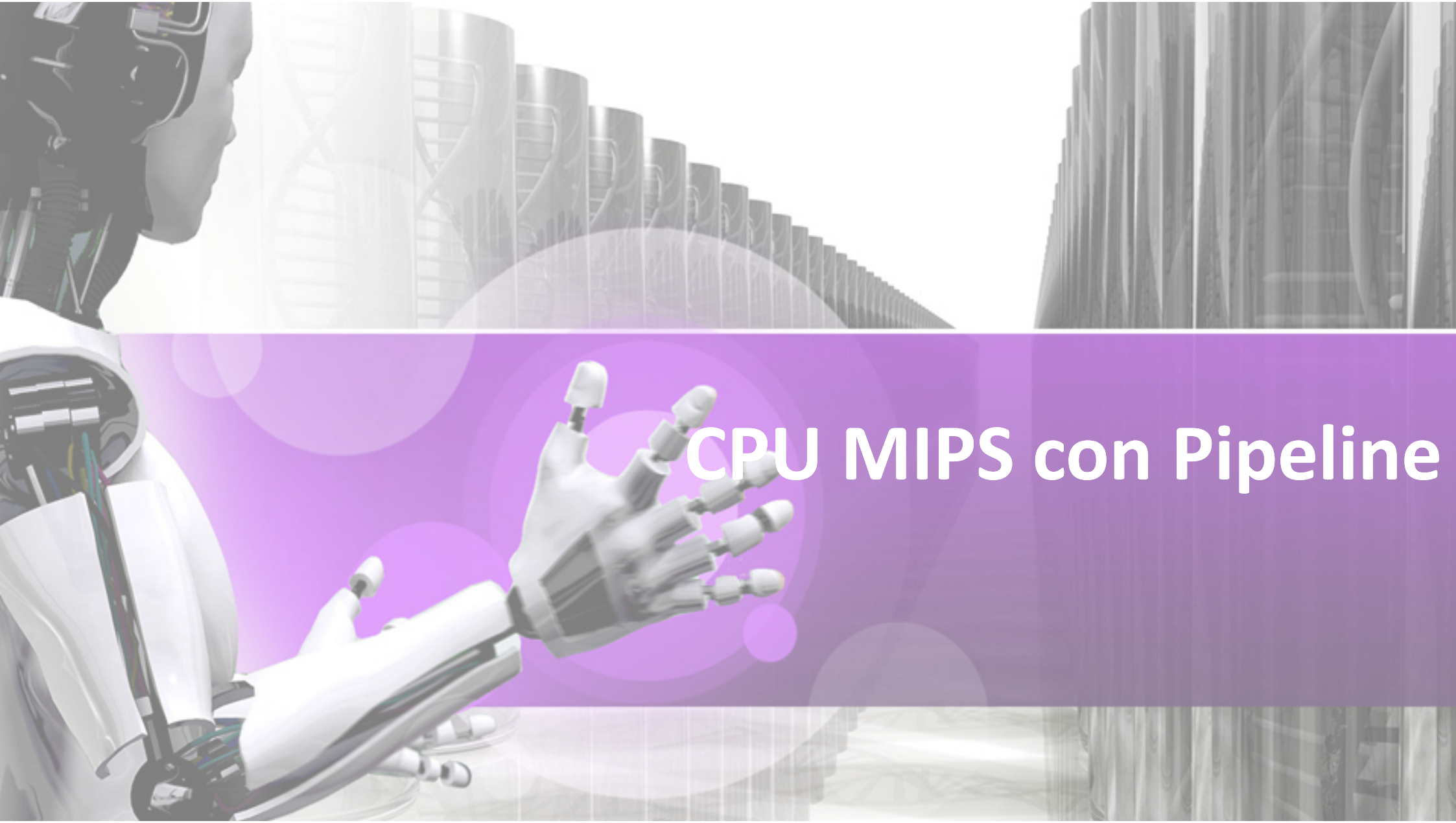
I7: sw $t5, 16($t0)

Δt	0	1	2	3	4	5	6	7	8	9	10
I1	F	D	E	M	WB						
I2		F	D	E	M	WB					
I3			F	D	E	M	WB				
I4				F	D	E	M	WB			
I5					F	D	E	M	WB		
I6						F	D	E	M	WB	
I7							F	D	E	M	WB

PIPELINE

Degrado delle prestazioni: mitigare i control hazard

- **Anticipare la decisione:**
  - se il salto condizionato è calcolato dall'ALU nella fase **E** ci saranno due istruzioni da «scartare» perché si deve aspettata la fine della terza fase; se invece il salto condizionato è deciso nella fase **D** è sufficiente «scartare» una istruzione
- **Branch prediction (previsione del salto):**
  - la CPU osserva i salti eseguiti e cerca di pre-caricare l'istruzione (seguente o di destinazione) che deve essere eseguita
- **Ritardare il salto:**
  - se è possibile anticipare l'istruzione che segue il salto condizionato senza conseguenze semantiche e logiche si elimina lo stallo



CPU MIPS con Pipeline

PIPELINE MIPS

Generalità

- Il MIPS con datapath monociclo funziona correttamente, ma non è implementato nelle architetture moderne, perché risulta inefficiente. Infatti, ciascuna istruzione ha un tempo di elaborazione diverso in relazione alla lunghezza del datapath che deve attraversare. Poiché il ciclo macchina deve essere stabilito a priori, il suo tempo deve essere sufficiente a permettere l'esecuzione dell'istruzione con la durata maggiore, comportando uno spreco temporale quando è elaborata una istruzione che attraversa un percorso molto breve
- Per questo motivo si ricorre ad una **macchina multiciclo**, in cui il progettista scompone l'esecuzione di ciascuna istruzione in un insieme di passi, detti anche **fasi** (o *stage*), ognuno dei quali è eseguibile in un singolo ciclo macchina. Ogni fase ricalca un passaggio dell'elaborazione di una istruzione (nel MIPS: caricamento dell'istruzione F; decodifica, D; reperimento operandi, R; esecuzione, E; salvataggio del risultato, M)
- Con questa organizzazione architetturale è possibile eseguire più istruzioni su ciascuna unità funzionale che realizza una fase; a fronte, però, di alcune modifiche fisiche rispetto alla struttura monociclo. Questa sovrapposizione delle istruzioni nelle diversi fasi è detta **canalizzazione** (o *pipeline*)

PIPELINE MIPS

Generalità

- La **canalizzazione** per funzionare adegautamente richiede che le varie unità siano sincronizzate e che i dati delle varie istruzioni non si sovrappongano, quindi all'interno delle pipeline sono posti dei blocchi (**interlock**) che sorvegliano il completamento delle varie istruzioni e fanno procedere il pipeline solamente quando tutti gli stadi sono pronti. Questo meccanismo garantisce la corretta esecuzione del programma ma introduce spesso stalli nella CPU che deprimono le prestazioni
- La caratteristica distintiva del progetto MIPS è che tutte le istruzioni sono completate dagli stadi della pipeline in un solo ciclo di clock in modo da non introdurre ritardi e stalli nella pipeline

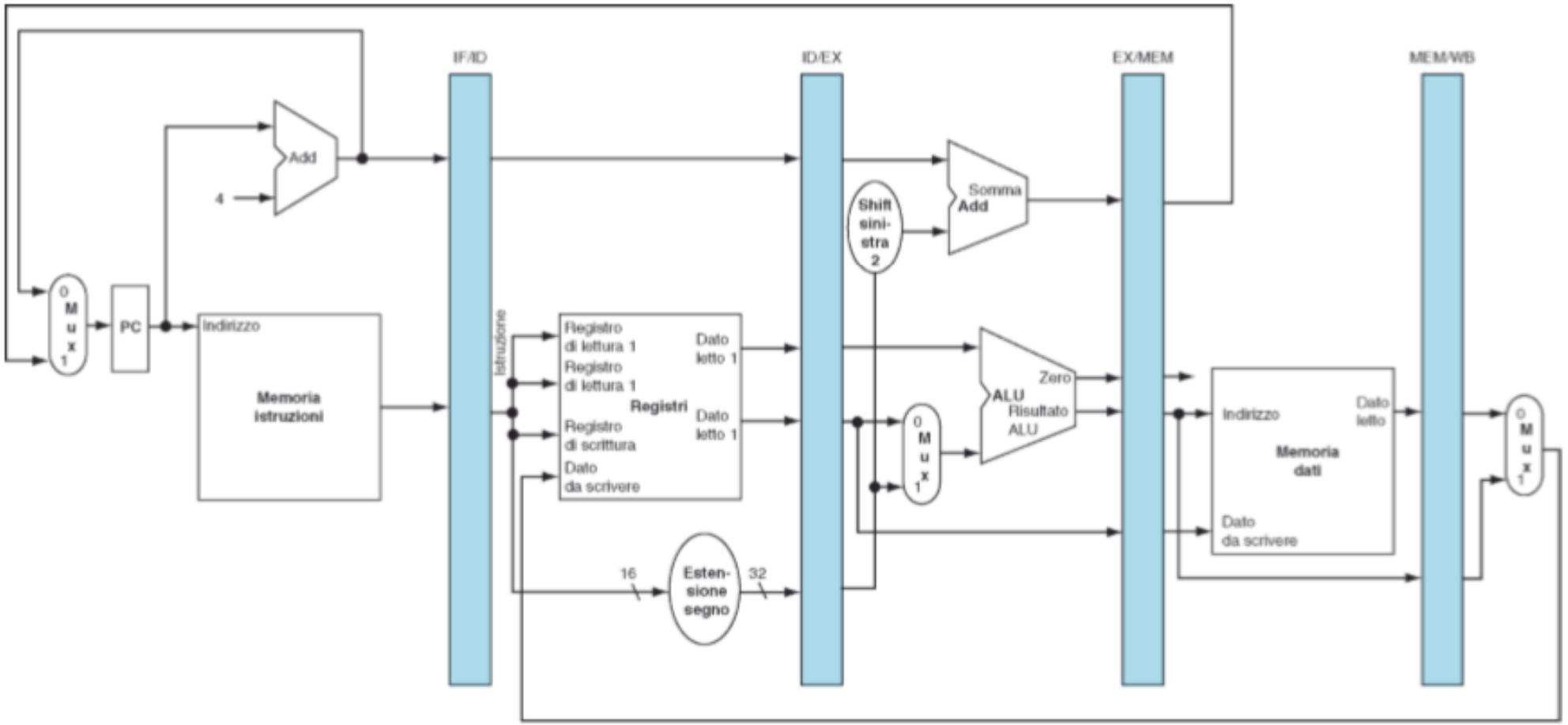
PIPELINE MIPS

Generalità

- **CPU MIPS con pipeline (senza gestione hazard)**
- **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** (***interblock***) che riceve informazioni e segnali di controllo dalla fase precedente e li mette a disposizione alla fase successiva
- MIPS ***m****icroprocessor without* ***i****nterlocked* ***p****ipeline* ***s****tages* (microprocessore senza fasi bloccanti nel pipeline)

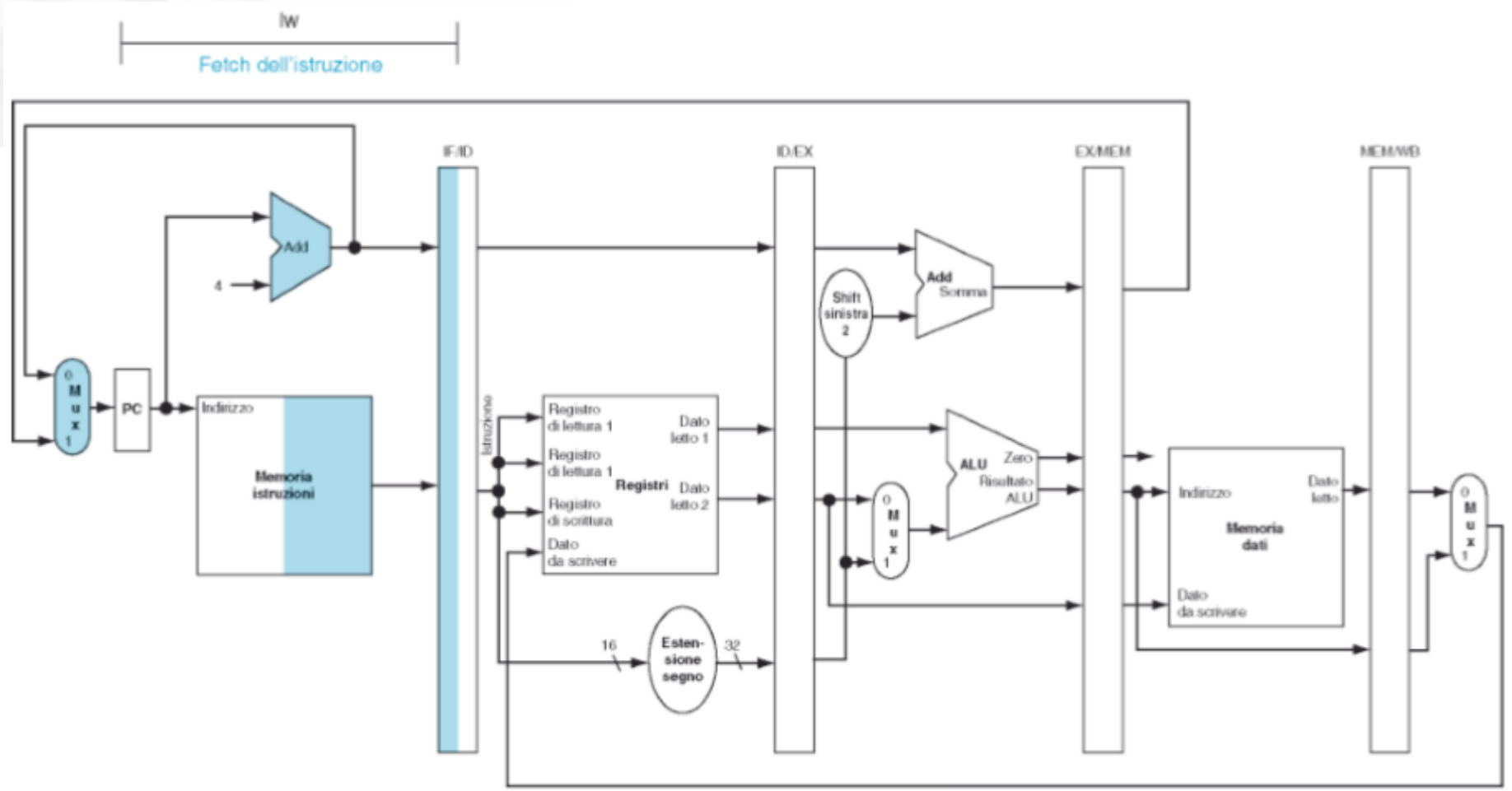
PIPELINE MIPS

Struttura di base



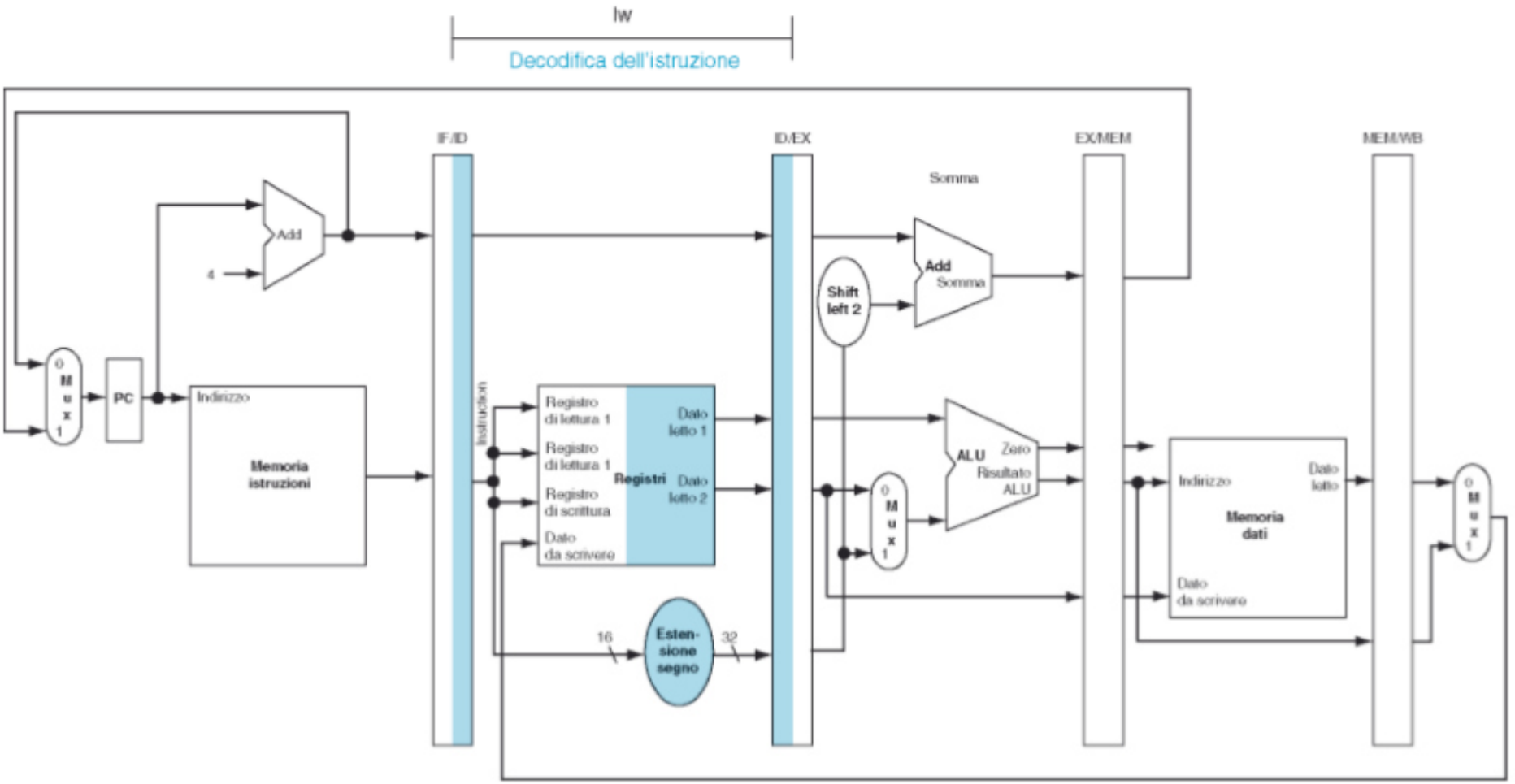
PIPELINE MIPS

Componenti coinvolte nel fetch



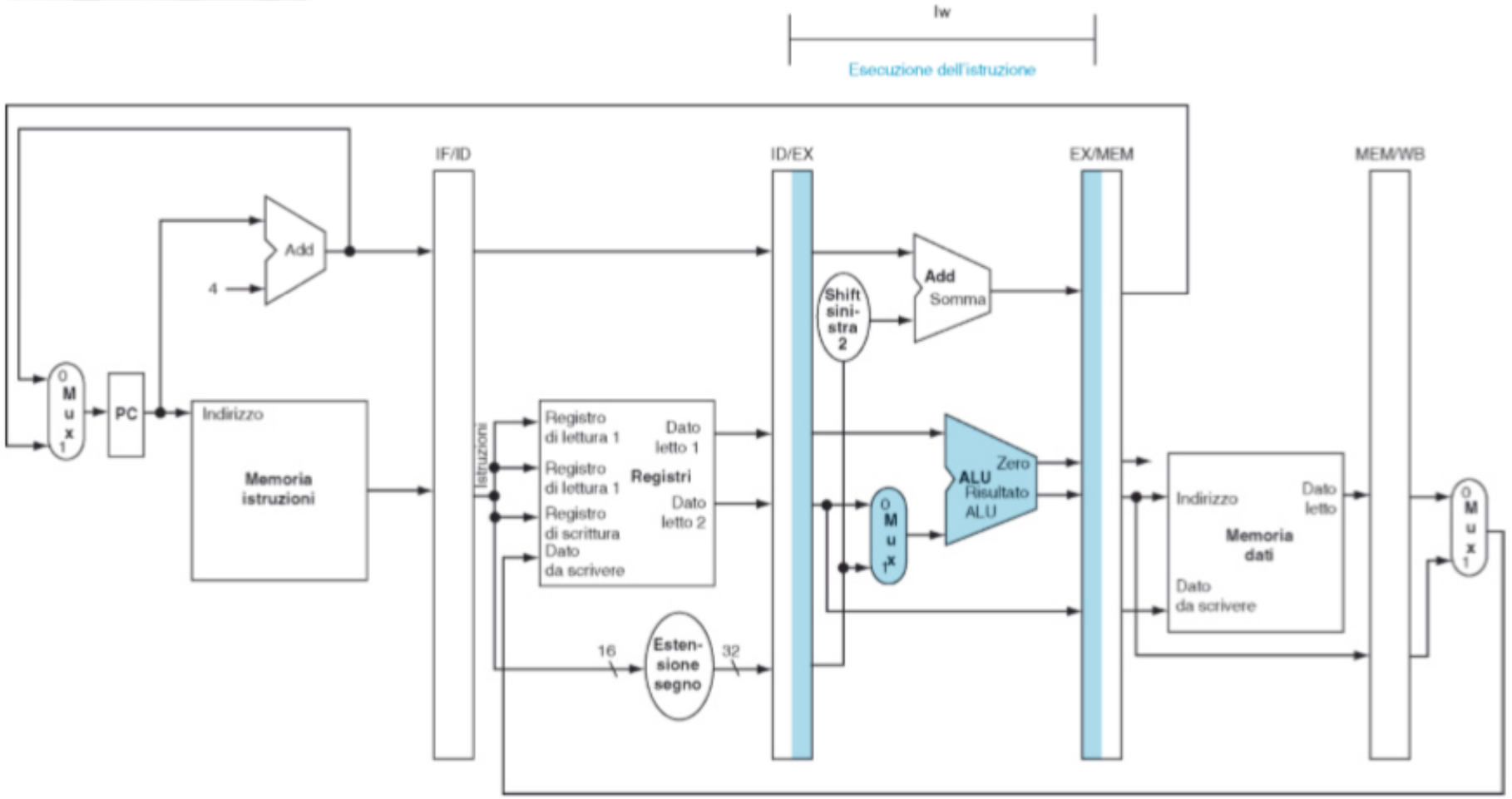
PIPELINE MIPS

Componenti coinvolte nella decodifica e lettura registri



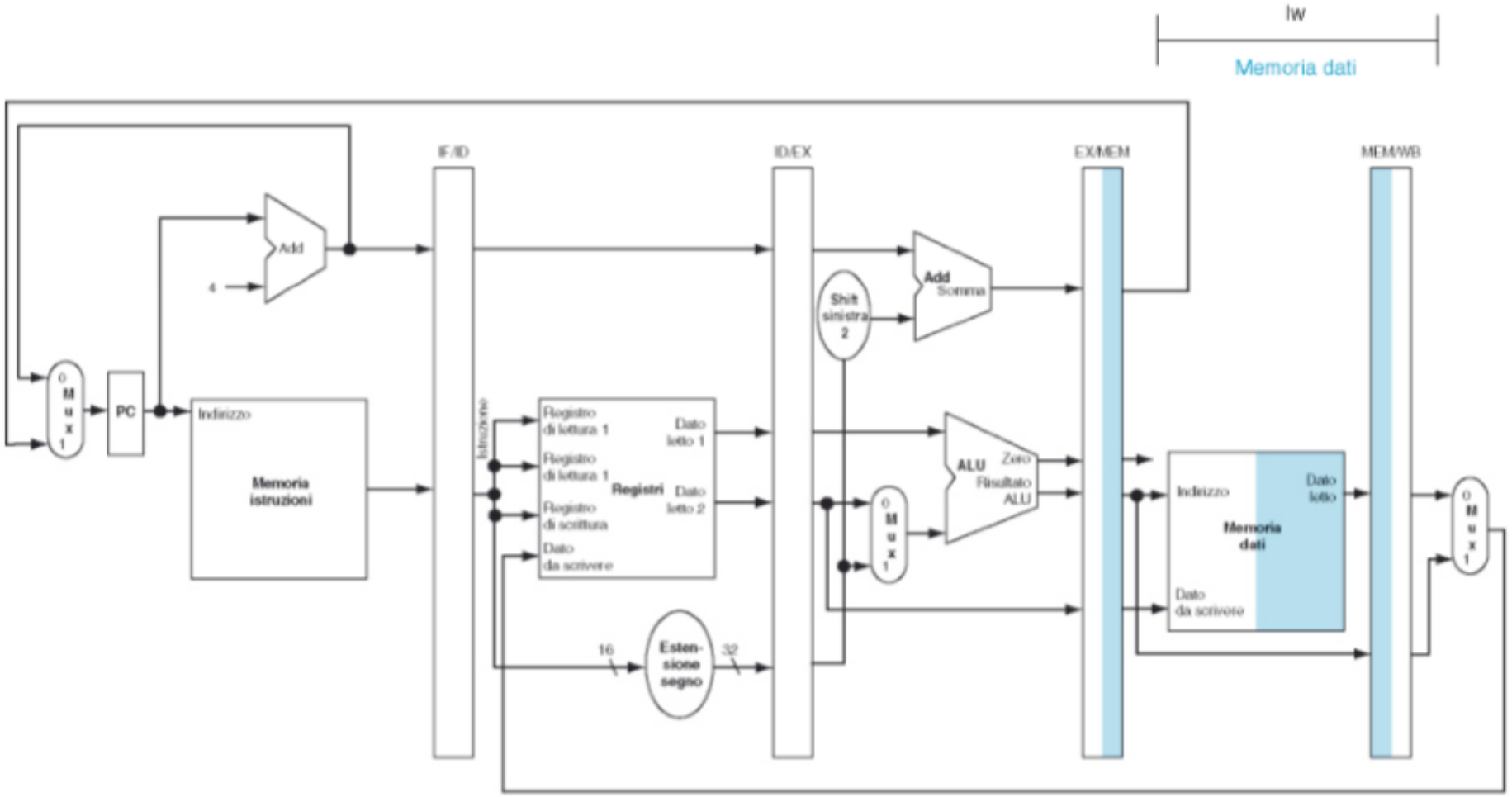
PIPELINE MIPS

Componenti coinvolte nell'esecuzione o calcolo indirizzo



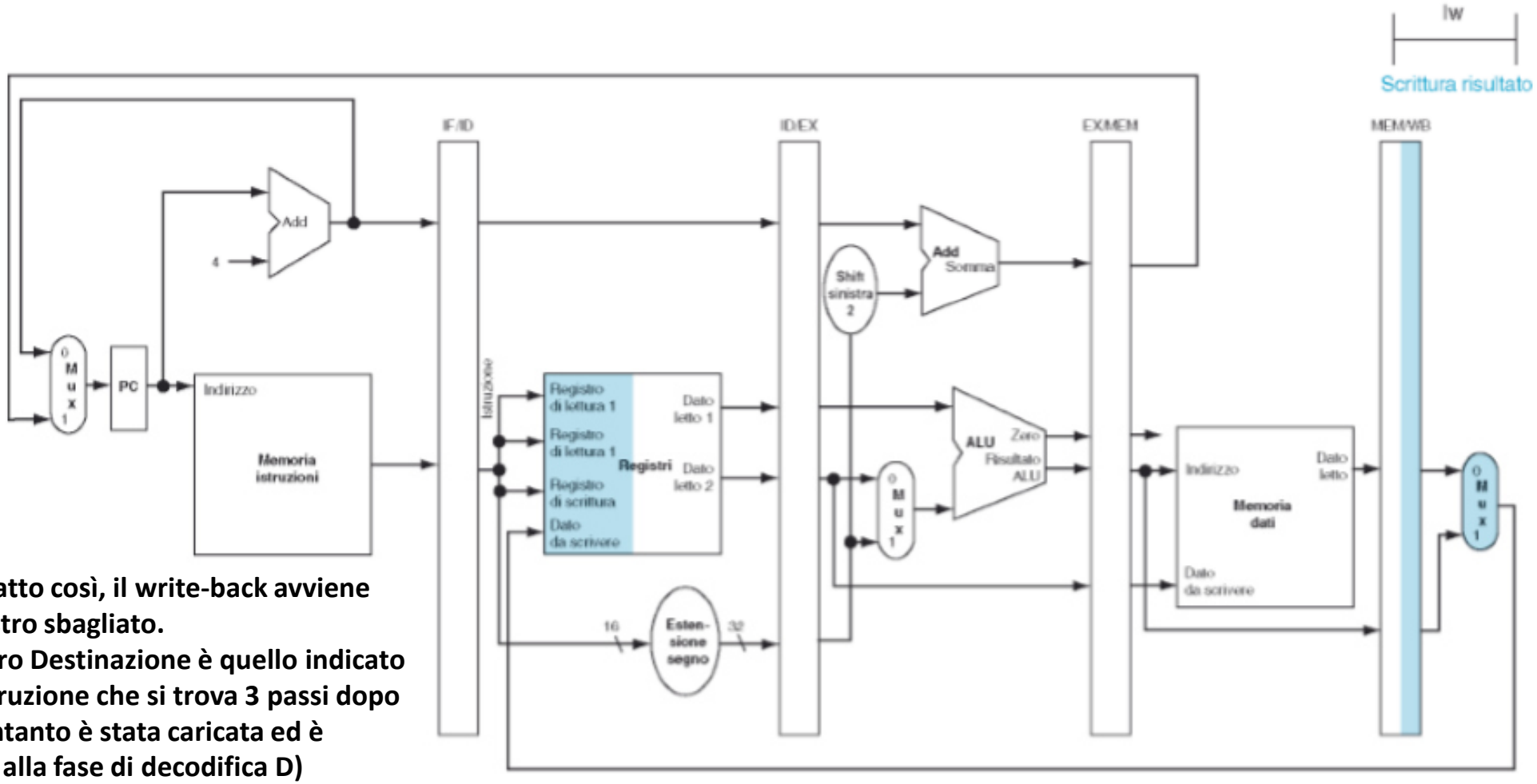
PIPELINE MIPS

Componenti coinvolte nell'accesso alla memoria



PIPELINE MIPS

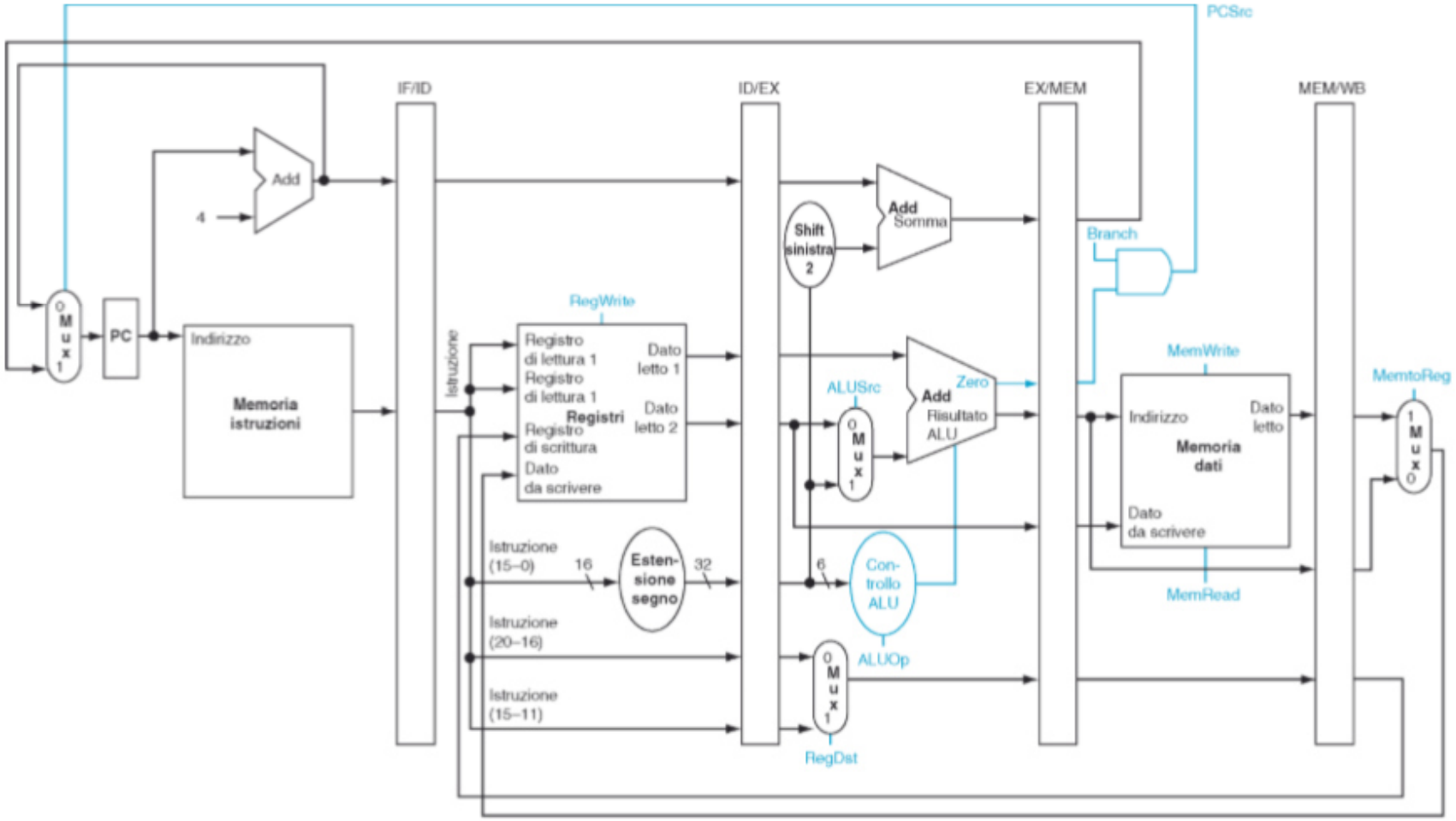
Componenti coinvolte nella scrittura registri (WB)



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 decodifica D)

PIPELINE MIPS

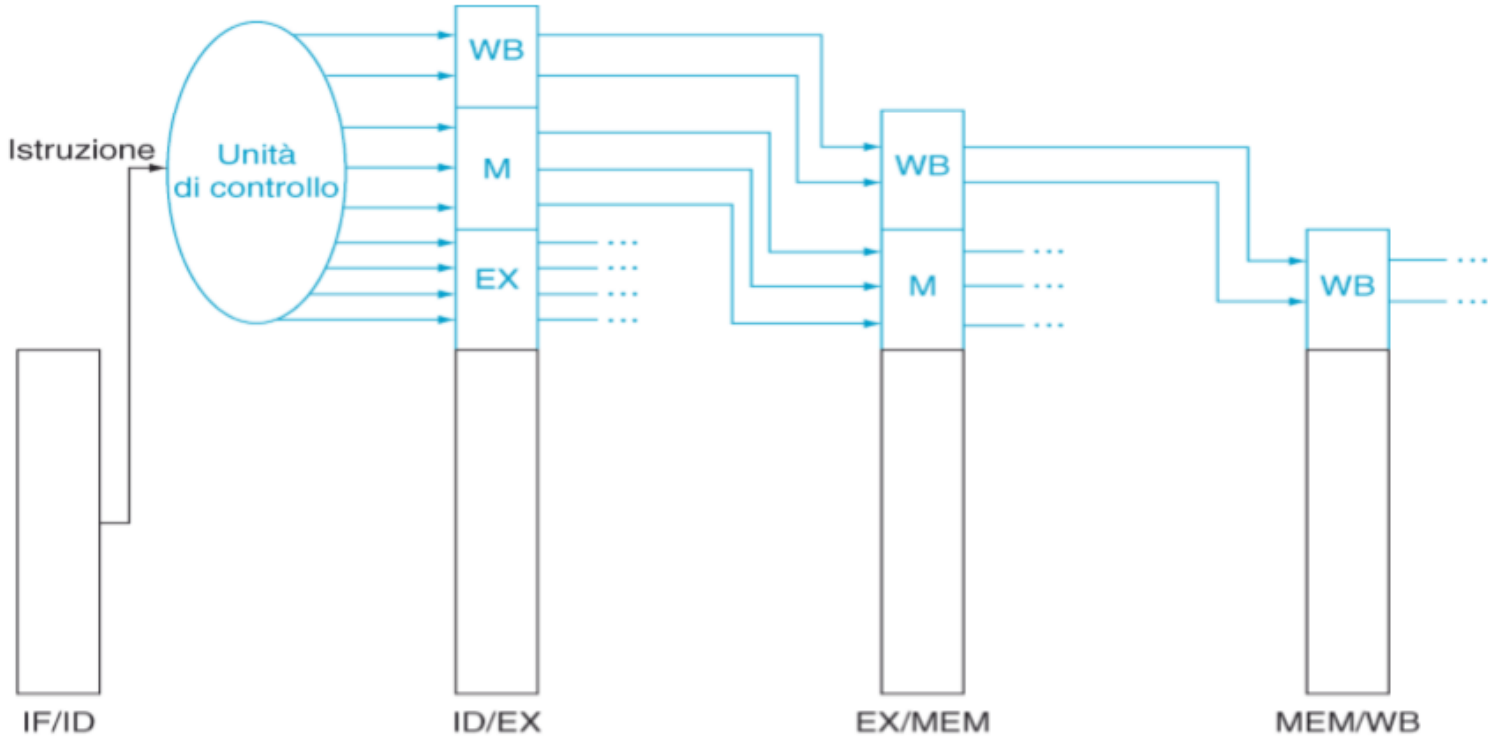
Pipeline con WB corretto e Salti



PIPELINE MIPS

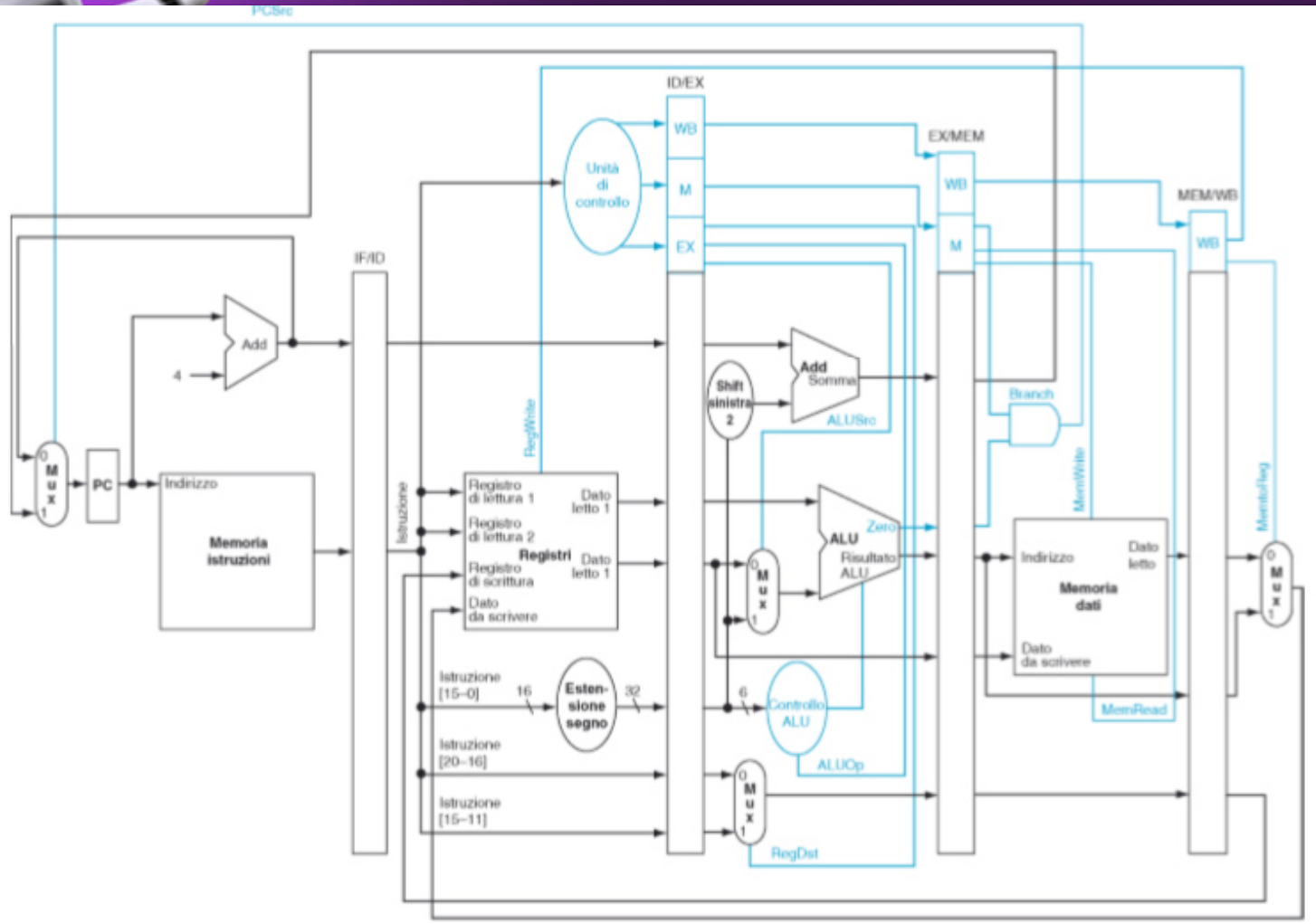
I registri del pipeline MIPS

- Ad ogni fase nei **registri** del pipeline sono presenti i **dati** (letti dai registri o dalla parte immediata) e i **comandi** (generati dalla CU)



PIPELINE MIPS

CPU MIPS



Criticità/Hazard

HAZARD

Generalità

- Le **criticità nella esecuzione** (**hazard**) possono essere:
  - **SUI DATI (*data hazard*)**: necessità di inserire una attesa perché il dato che deve essere elaborato in una istruzione non è ancora pronto perché in corso di calcolo nell'istruzione precedente
  - **SUL CONTROLLO (*control hazard*)**: un salto cambia il flusso sequenziale di esecuzione delle istruzioni (jump/beq ma anche una interruzione o un salto a subroutine)
  - **STRUTTURALI**: le risorse HW non sono sufficienti (es.: l'ALU sta ancora svolgendo una operazione precedente e quindi non è libera)

DATA HAZARD

Generalità

- Un **Data-hazard** si verifica quando una istruzione dà il valore ad una delle due istruzioni successive (prima di WB) ovvero: il registro destinazione della istruzione precedente è uguale ad uno dei due registri argomenti delle due istruzioni successive ed inoltre l'istruzione precedente ha il comando *RegWrite* impostato a *true*
  - **Problema: stallo**
  - **Soluzione: FORWARDING o eliminazione stallo**

DATA HAZARD

Soluzioni alle criticità: propagazione (forwarding)

- Nel MIPS, per risolvere lo stallo, si può (nella fase di Decodifica):
  - Annullare l'istruzione che deve attendere (immettere una bolla che continuerà senza fare nulla) ovvero azzerare i segnali di controllo *MemWrite* e *RegWrite* e bloccare il registro F/D
  - Ripetere l'istruzione che è stata letta e deve entrare nella fase Decodifica ovvero impedire l'aggiornamento del registro F/D della pipeline
  - Rileggere la stessa istruzione di nuovo perché possa essere rieseguita un clock dopo, ovvero impedire che il PC si aggiorni

CONTROL HAZARD

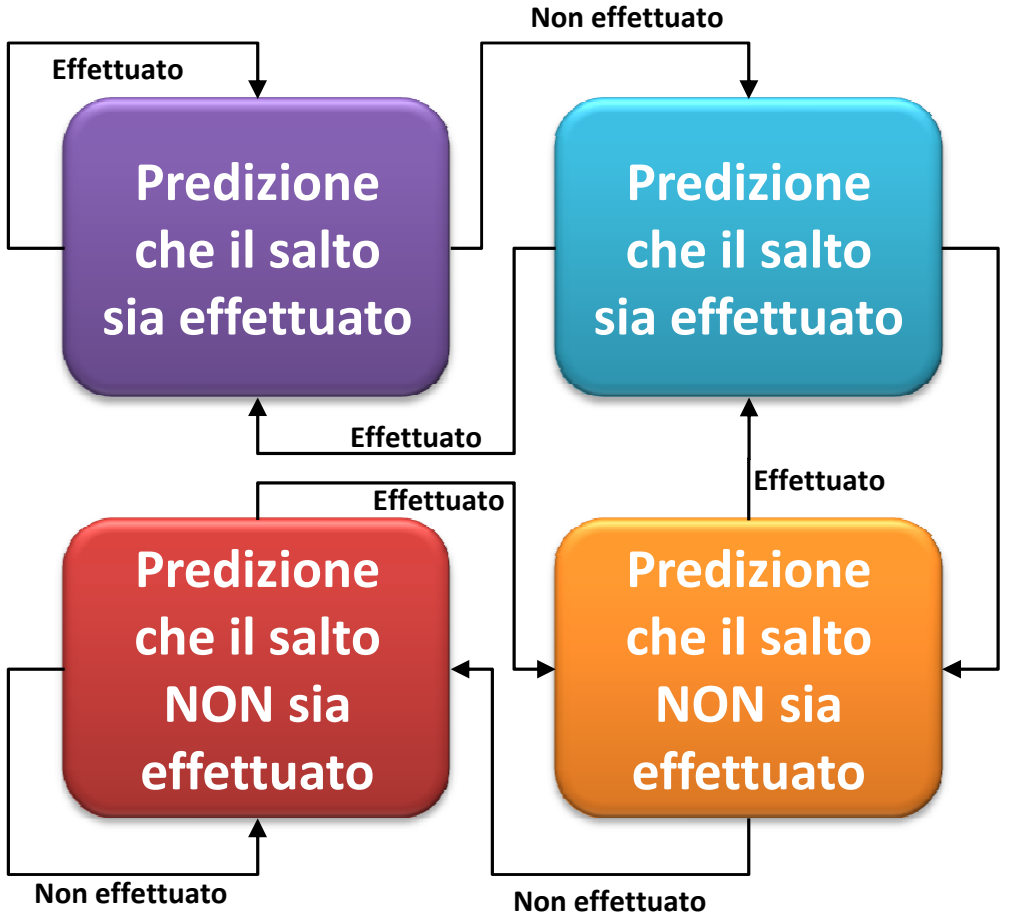
Generalità

- **Salto incondizionato:**
  - **Anticipo**: si costruisce una circuiteria che analizza l'opcode dell'istruzione e se è un salto incondizionato si aggiorna il Program Counter dopo il fetch senza decodificare l'istruzione saltando alla parte di codice indicato dal salto (non si scarta nessuna istruzione perchè la successiva è quella corretta)
- **Salto condizionato**:
  - **Predizione**
  - **Ritardo del salto (delay slot)**

CONTROL HAZARD

Predizione

- Ad ogni istruzione di salto si possono associare alcuni bit che «**predicono**» (rispetto alla storia dei salti già fatti) contando se è più probabile che il salto venga fatto oppure no
  - con un solo bit si può rappresentare l'informazione «l'ultima volta il salto è stato fatto» nel realizzare un ciclo la previsione sarà sbagliata 2 volte: entrando e uscendo
  - con due bit si può realizzare la macchina a stati finiti (figura laterale), che ha bisogno di due sbagli consecutivi per cambiare previsione e quindi sbaglia meno nei cicli a forte prevalenza di uno specifico tipo di scelta (fa una sola predizione errata)



CONTROL HAZARD

Ritardo del salto

- Per recuperare il tempo perso dallo stallo si può **ritardare il salto** (***delay slot***)ovvero eseguire sempre l'istruzione che segue il salto condizionato (prima di quest'ultimo). Questo richiede una scrittura del codice assembly diversa, oppure l'uso di un compilatore capace di automatizzare le necessarie modifiche al codice macchina prodotto

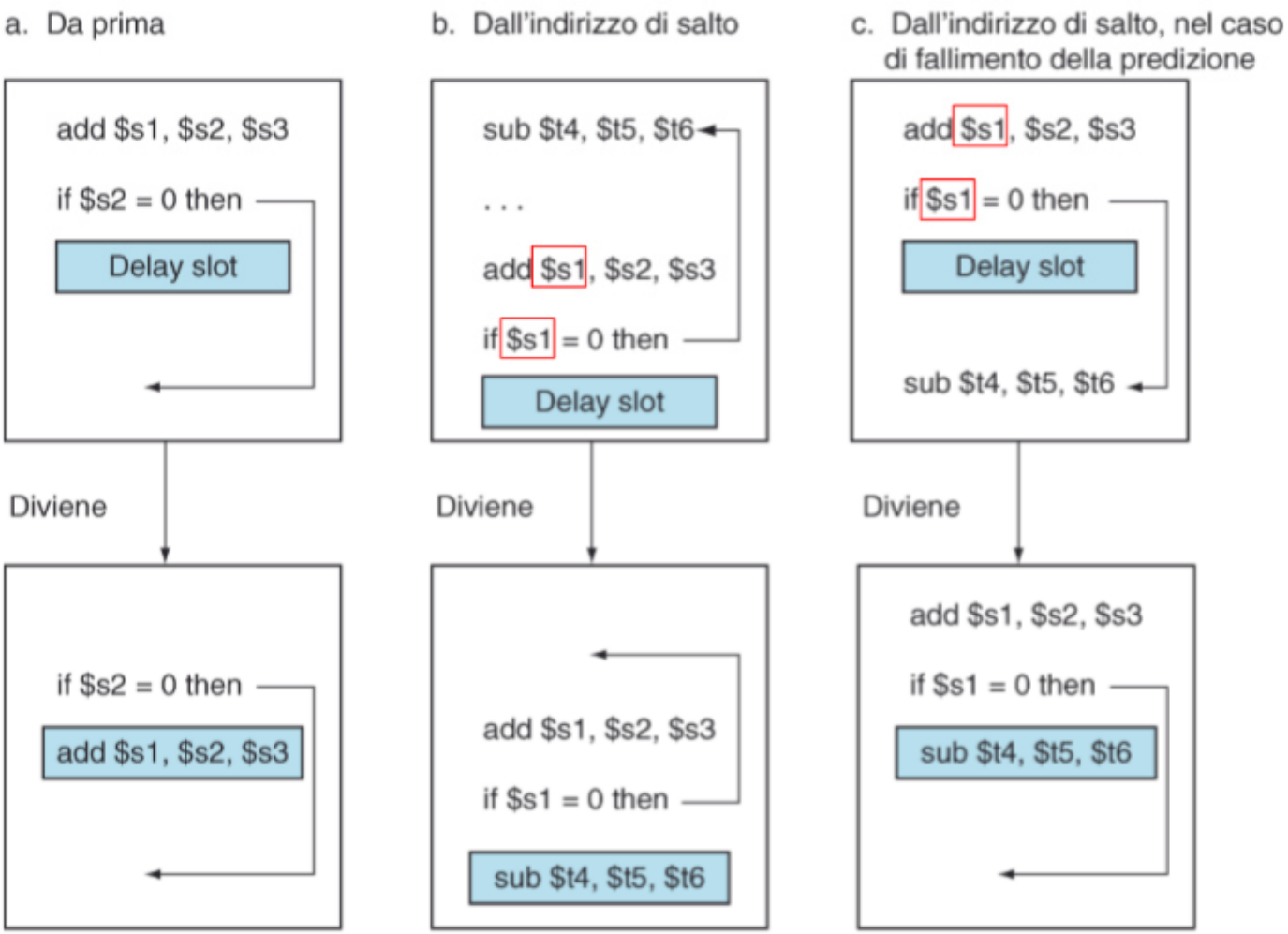
- Nel delay slot è possibile copiare una delle istruzioni che vanno sempre eseguite:

  Caso 1: una istruzione precedente che non abbia dipendenze (anche indirette) con il salto condizionato
  NOTA: l'istruzione viene copiata perché potrebbe far parte di altri flussi di controllo

- Se non ci sono istruzioni precedenti senza dipendenze:
  Caso 2: si copia l'istruzione alla destinazione del salto
  NOTA: quando il salto NON viene fatto l'istruzione scelta (che viene sempre eseguita) però non deve creare problemi:
  - ad esempio può calcolare un valore non più necessario nel codice seguente
  - l'importante è che non sia dannosa per l'esecuzione successiva

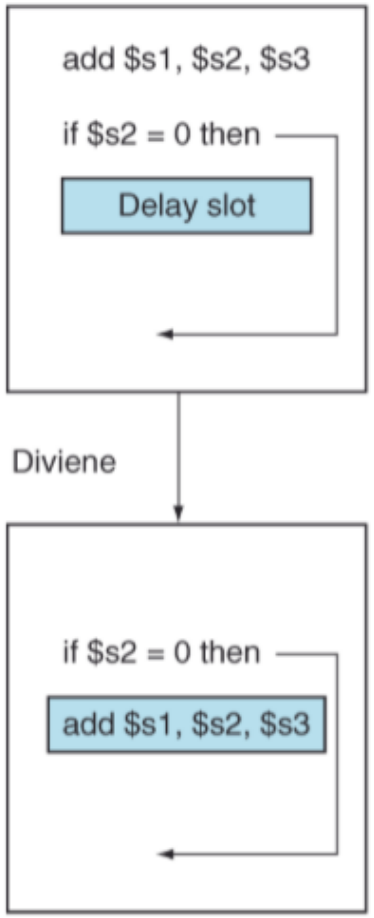
CONTROL HAZARD

Ritardo del salto



CONTROL HAZARD
Ritardo del salto

a. Da prima



CONTROL HAZARD

Ritardo del salto: caso iniziale

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	F	D	E	M	W														
I2		F	D	E	M	W													
I3			F	D	E	M	W												
I4				F	D	E	M	W											
B10					F	D	E	M	W										
I6						F	D	E	M										
I7							F	D	E										
I8								F	D										
I9									F										
I10										F	D	E	M	W					

CONTROL HAZARD

Ritardo del salto:posticipo istruzione

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	F	D	E	M	W														
I2		F	D	E	M	W													
I3			F	D	E	M	W												
B10				F	D	E	M	W											
I4					F	D	E	M	W										
I6						F	D	E	M										
I7							F	D	E										
I8								F	D										
I9									F	D	E	M	W						

CONTROL HAZARD

Ritardo del salto: analisi del salto

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	F	D	E	M	W														
I2		F	D	E	M	W													
I3			F	D	E	M	W												
B10				F	D	E	*M*	*W*											
I4					F	D	E	M	W										
I6						F	D												
I10							F	D	E	M	W								

CONTROL HAZARD

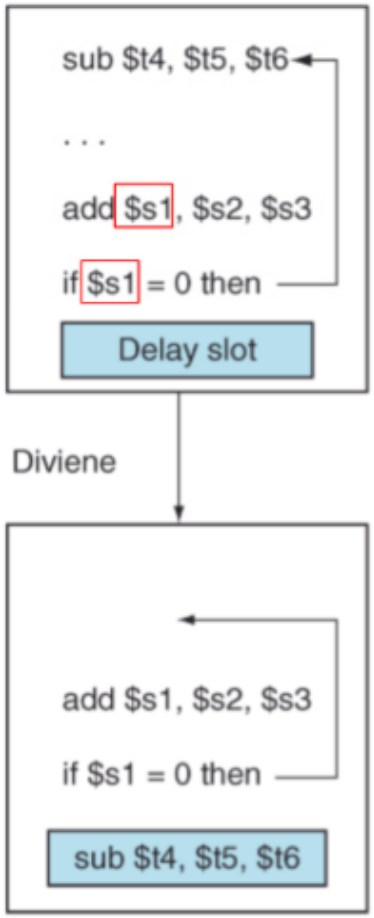
Ritardo del salto: forwarding

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	F	D	E	M	W														
I2		F	D	E	M	W													
I3			F	D	E	M	W												
B10				F	D	*E*	*M*	*W*											
I4					F	D	E	M	W										
I10						F	D	E	M	W									

CONTROL HAZARD

Posticipo istruzione sempre eseguita

b. Dall'indirizzo di salto



PIPELINE MIPS

Posticipo istruzione sempre eseguita: caso iniziale

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	F	D	E	M	W														
I2		F	D	E	M	W													
I3			F	D	E	M	W												
I4				F	D	E	M	W											
I5					F	D	E	M	W										
I6						F	D	E	M	W									
I7							F	D	E	M	W								
B(I4)								F	D	E	M	W							
I9									F	D	E	M							
I10										F	D	E							
I11											F	D							
I12												F							
I4													F	D	E	M	W		

PIPELINE MIPS

Posticipo istruzione sempre eseguita:spostamento

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	F	D	E	M	W														
I2		F	D	E	M	W													
I3			F	D	E	M	W												
I5				F	D	E	M	W											
I6					F	D	E	M	W										
I7						F	D	E	M	W									
B(I4)							F	D	E	M	W								
I4								F	D	E	M	W							
I9									F	D	E								
I10										F	D								
I11											F								
I5											F	D	E	M	W				

PIPELINE MIPS

Posticipo istruzione sempre eseguita: analisi del salto

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	F	D	E	M	W														
I2		F	D	E	M	W													
I3			F	D	E	M	W												
I5				F	D	E	M	W											
I6					F	D	E	M	W										
I7						F	D	E	M	W									
B(I4)							F	D	E	*M*	*W*								
I4								F	D	E	M	W							
I9									F										
I5										F	D	E	M	W					
I6											F	D	E	M	W				
I7												F	D	E	M	W			

PIPELINE MIPS

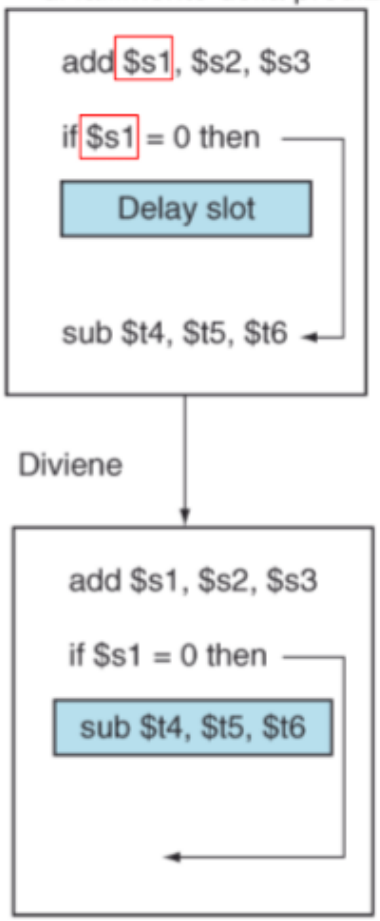
Posticipo istruzione sempre eseguita: forwarding

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	F	D	E	M	W														
I2		F	D	E	M	W													
I3			F	D	E	M	W												
I5				F	D	E	M	W											
I6					F	D	E	M	W										
I7						F	D	E	M	W									
B(I4)							F	D	E	*M*	*W*								
I4								F	D	E	M	W							
I5									F	D	E	M	W						
I6										F	D	E	M	W					

PIPELINE MIPS

Fallimento predizione

c. Dall'indirizzo di salto, nel caso di fallimento della predizione



PIPELINE MIPS

Fallimento predizione: caso iniziale

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	F	D	E	M	W														
I2		F	D	E	M	W													
I3			F	D	E	M	W												
I4				F	D	E	M	W											
B(I11)					F	D	E	M	W										
I6						F	D	E	M	W									
I7							F	D	E	M	W								
I8								F	D	E	M	W							
I9									F	D	E	M	W						
I10										F	D	E	M	W					
I11											F	D	E	M	W				
I12												F	D	E	M	W			
I13													F	D	E	M	W		

PIPELINE MIPS

Fallimento predizione: spostamento

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	F	D	E	M	W														
I2		F	D	E	M	W													
I3			F	D	E	M	W												
I4				F	D	E	M	W											
B(I11)					F	D	E	M	W										
I11						F	D	E	M	W									
I7							F	D	E										
I8								F	D										
I9									F										
I12										F	D	E	M	W					
I13											F	D	E	M	W				

PIPELINE MIPS

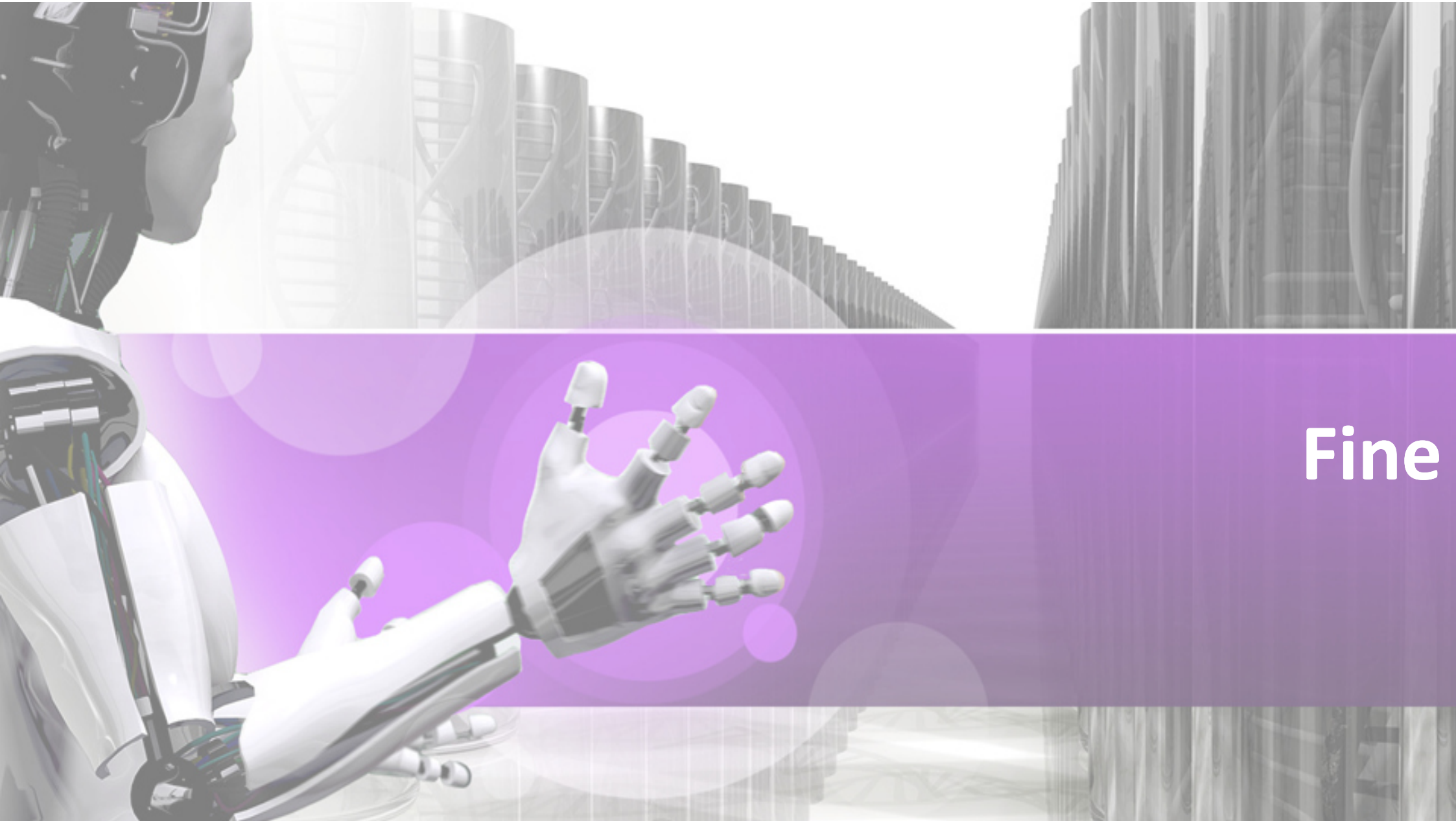
Fallimento predizione: analisi salto

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	F	D	E	M	W														
I2		F	D	E	M	W													
I3			F	D	E	M	W												
I4				F	D	E	M	W											
B(I11)					F	D	E	*M*	*W*										
I11						F	D	E	M	W									
I7							F	D	E										
I12								F	D	E	M	W							
I13									F	D	E	M	W						

PIPELINE MIPS

Fallimento predizione: forwarding

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	F	D	E	M	W														
I2		F	D	E	M	W													
I3			F	D	E	M	W												
I4				F	D	E	M	W											
B(I11)					F	D	*E*	*M*	*W*										
I11						F	D	E	M	W									
I12							F	D	E	M	W								
I13								F	D	E	M	W							


Fine