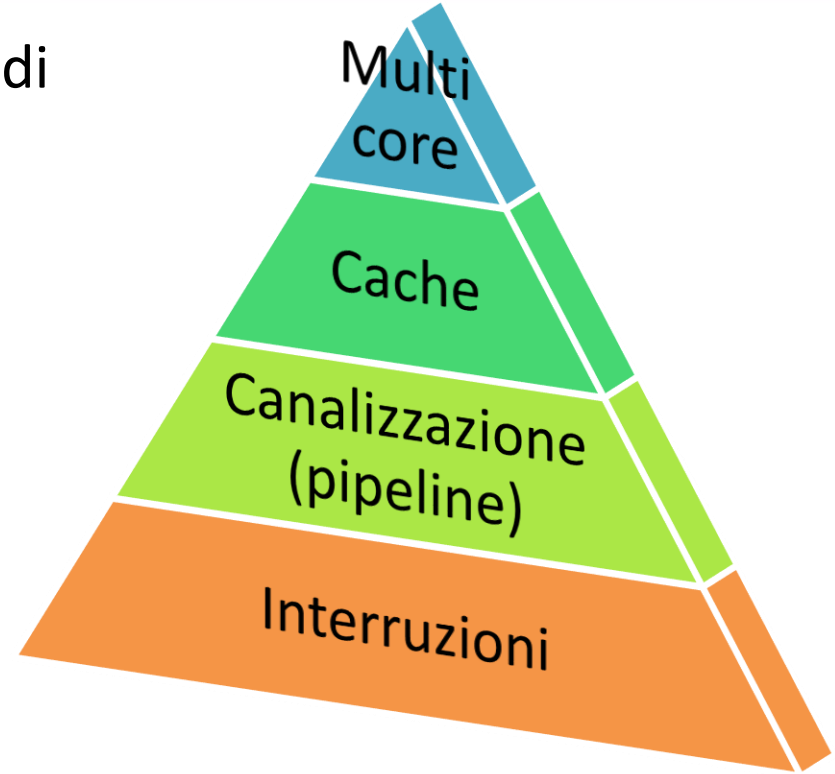
Architettura degli Elaboratori Elettronici

Canalizzazione

ARGOMENTI DELLA LEZIONE

- Miglioramenti Macchina di Von Neumann
  - Interruzione
  - Canalizzazione o pipeline
  - Cache
  - Multi core (*cenni*)



Canalizzazione (pipeline)

PIPELINE

Macchina senza pipeline

In una architettura che segue il modello della MdVN le istruzioni sono eseguite una dopo l'altra

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

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 fasi in parallelo**

PIPELINE

Generalità

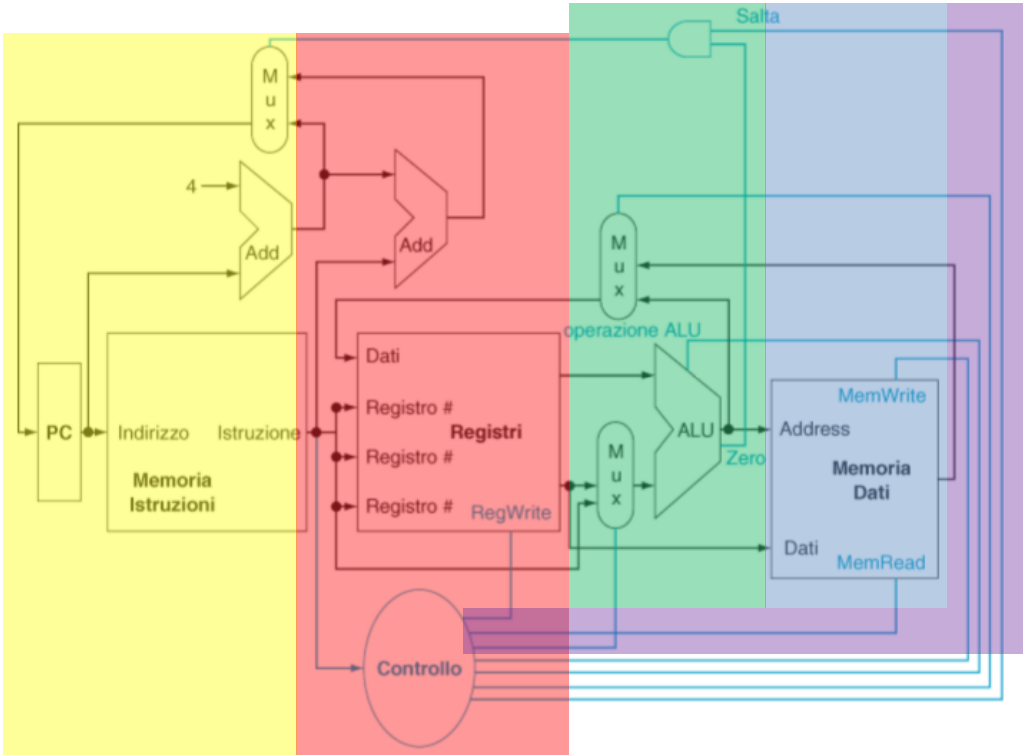
- L'esecuzione di una istruzione, in una macchina RISC (es.: MIPS), è di solito suddivisa in cinque **fasi** (o **sezioni** o ***stage***):
  - **F – Fetch**: prelievo istruzione (con incremento del Program Counter)
  - **D – Decode**: decodifica/riconoscimento istruzione
  - **E – Execution**: esecuzione istruzione
  - **M – Memory**: accesso in memoria per scrittura o lettura (*load* o *store*)
  - **WB – Write Back**: quando il risultato dell'ALU (o durante una operazione di load) viene messo nel registro destinazione

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 ●
- **MEM**: Memoria dati ●
- **Write Back**: Blocco registri ●



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

Generalità

- Poiché ciascuna fase deve poter essere svolta contemporaneamente a quelle delle altre istruzioni, è necessario imporre che tutte quante **abbiano la stessa durata** (stabilita valutando la fase più lenta durante la progettazione dell'architettura con test temporali)
- Questo determina un **clock totale più lungo** (si attribuisce a tutte le classi di istruzione il tempo della classe di istruzione più dispendiosa temporalmente; c'è un periodo più lungo); ma il tempo complessivo di esecuzione del programma si migliora (si ha una frequenza di elaborazione maggiore una volta che la canalizzazione è a regime)

PIPELINE

Calcolo del periodo di clock

- 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 progetta la CPU in modo che le fasi abbiano tempi simili

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

- In questo esempio, grazie alla canalizzazione 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

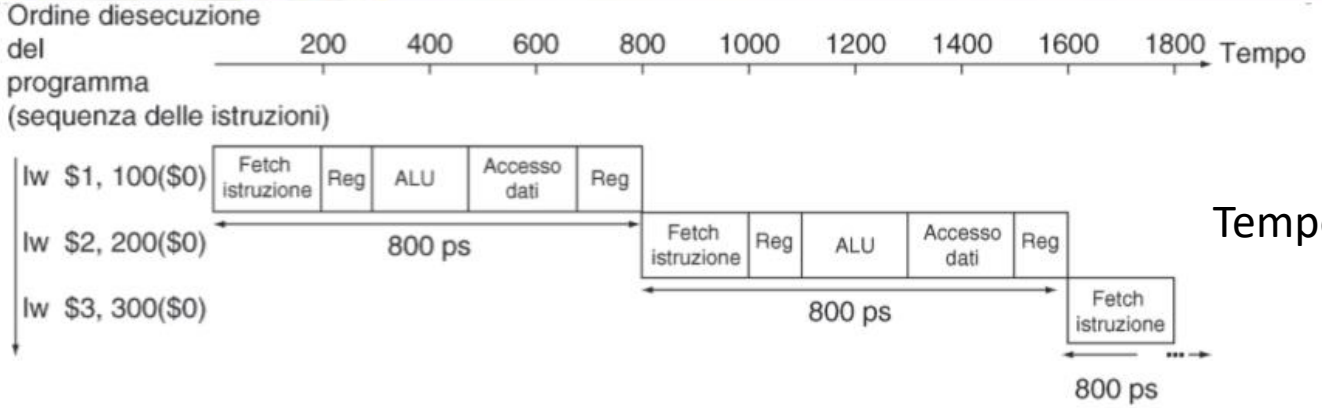
PIPELINE

Applicazione e guadagno

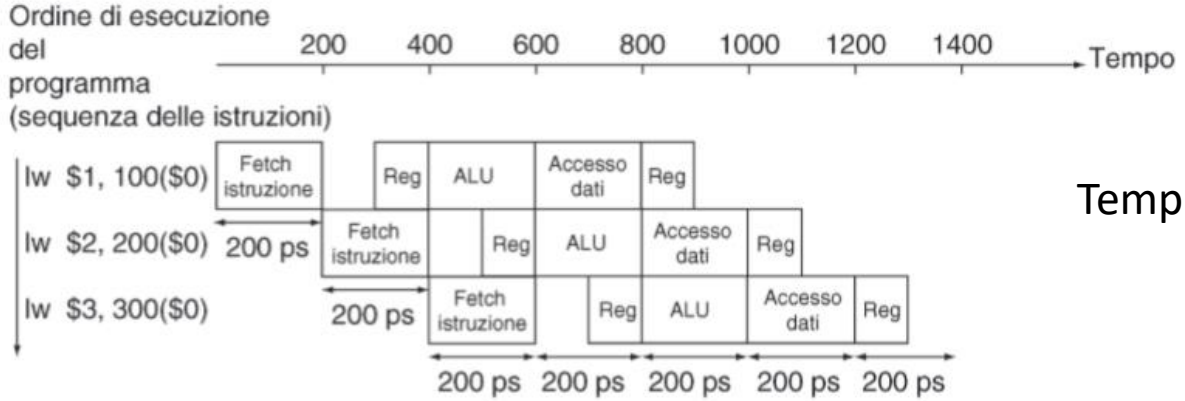
- Il guadagno offerto da una canalizzazione è, una volta “a regime”, dato del numero delle fasi elementari legate alla esecuzione dell’istruzione (in un caso con cinque fasi il guadagno è di cinque volte)
- In alcune macchine c’è una canalizzazione fino a 20 fasi che consente uno sfruttamento completo 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 per singolo core

PIPELINE

Esempio dell'applicazione e del guadagno



Tempo=3x800ps=2400ps



Tempo=1000+200ps+200ps=1400ps

PIPELINE

Compilatore in macchine canalizzate

- 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 **elaboratori 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)
  - **D – Decode**: decodifica/riconoscimento istruzione
  - **L – Load**: prelievo operandi dalla memoria (tipico delle istruzioni con modi di indirizzamento complesso)
  - **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 di scrittura dei dati
- L'alternativa è commisurare il tempo più lungo di ciascuna fase delle diverse classi di istruzioni ed uniformare ogni fase di tutte le classi di istruzioni a quelle più lunghe (esattamente come avviene per le macchine RISC); con un dispendio temporale più lungo per le istruzioni "meno complesse"

PIPELINE

Canalizzazione nelle macchine CISC

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	D	D	L	L	L	E	E	M	WB											
Istruzione 2				F	D	D	L	L	L	E	M	WB									
Istruzione 3						F	D	D	D	D	L	L	E	M	M	WB					
Istruzione 4										F	F	D	L	L	L	E	M	WB			
Istruzione 5												F	F	D	D	L	L	L	E	M	WB

PIPELINE

Canalizzazione nelle macchine CISC

- Seppur fattibile, la canalizzazione nelle macchine CISC è molto più complicata da realizzare
- Nel contempo una istruzione CISC compie funzioni più "articolate" che in una macchina RISC richiederebbero più istruzioni
- Malgrado questi due punti, uno sfavorevole ma fattibile e l'altro estremamente vantaggioso, si predilige l'uso di Macchine RISC perché, da analisi statistiche, il 90% dei programmi utilizza istruzioni semplici mentre l'impiego di istruzioni complesse è riservato a pochi applicativi specifici

ISTRUZIONE COMPLESSA

ADD #300,#100,#200

ISTRUZIONE SEMPLICE AFFINE

lw $t1,100

lw $t2,200

add $t0,$t1,$t2

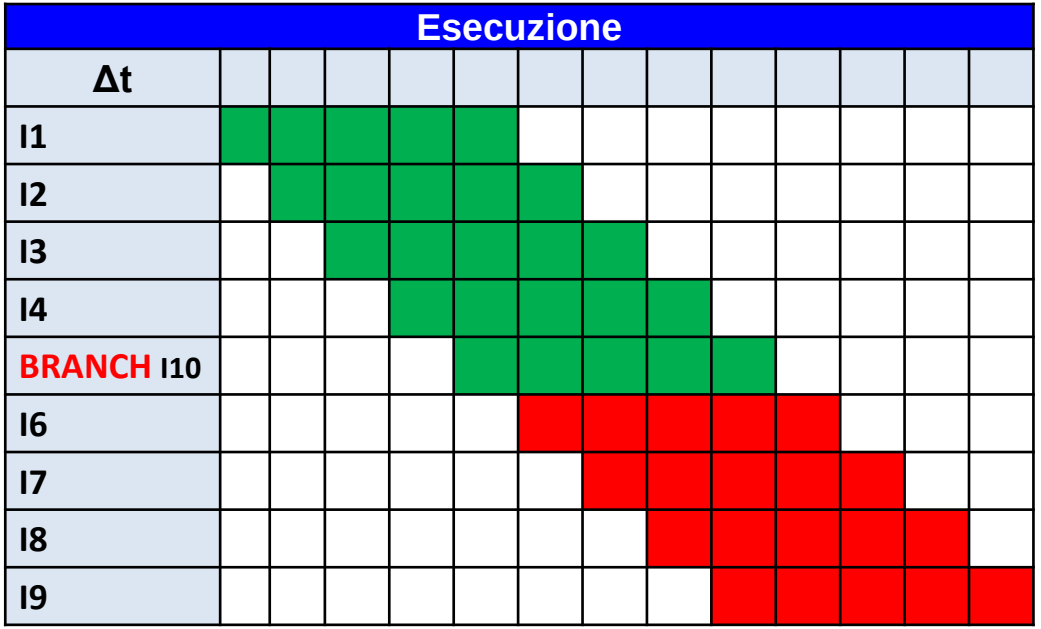
sw $t0,300

Condizioni svantaggiose per il Pipeline
Criticità della canalizzazione o hazard

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 al punto di arrivo del salto e le successive



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*)**: quando un dato (istruzione o operando) 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

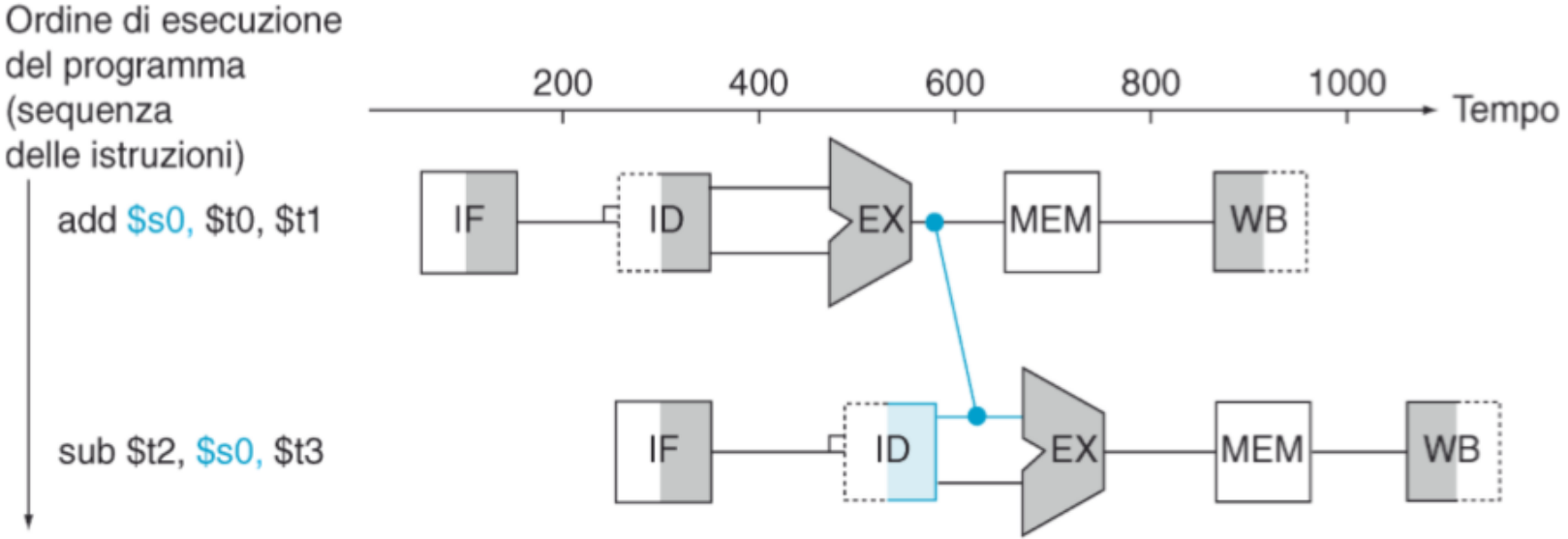
Δt	0	1	2	3	4	5	6	7
I1	F	D	E	M	WB			
I2		Stallo1	Stallo2	F	D	E	M	WB

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). Per questo si ricorre ad uno/due Stallo

PIPELINE

Soluzioni alle criticità: propagazione (forwarding)

- In alcuni casi l'informazione utile all'istruzione successiva 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: propagazione (forwarding)

- L’adozione del forwarding evita i problemi correlati ad una canalizzazione con solo interlocked block.
- Una canalizzazione interlocked block le fasi della pipeline NON sono interconnesse in modo tale che l'esecuzione di un'istruzione può dipendere dal completamento di un'altra istruzione nella pipeline. Questo tipo di organizzazione può introdurre ritardi e complicazioni nel design del processore
- Con il termine "without Interlocked Pipeline Stages" si indica che il design del processore MIPS è stato sviluppato senza questa caratteristica, adottando un approccio più semplice e diretto alla progettazione della pipeline. Questo ha contribuito a rendere il MIPS più efficiente e più facile da implementare rispetto ad altre architetture di processore che utilizzano l'interlocking delle fasi della pipeline.

PIPELINE

Soluzioni alle criticità: propagazione (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 un tempo di 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: propagazione (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

- Una ulteriore soluzione ai dati hazard (e in parte ai control hazard) si ottiene **riordinando le istruzioni** (ed evitando gli stalli).
  La condizione da rispettare è mantenere la stessa semantica

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	15
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	

In questo esempio è integrato il forwarding

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)

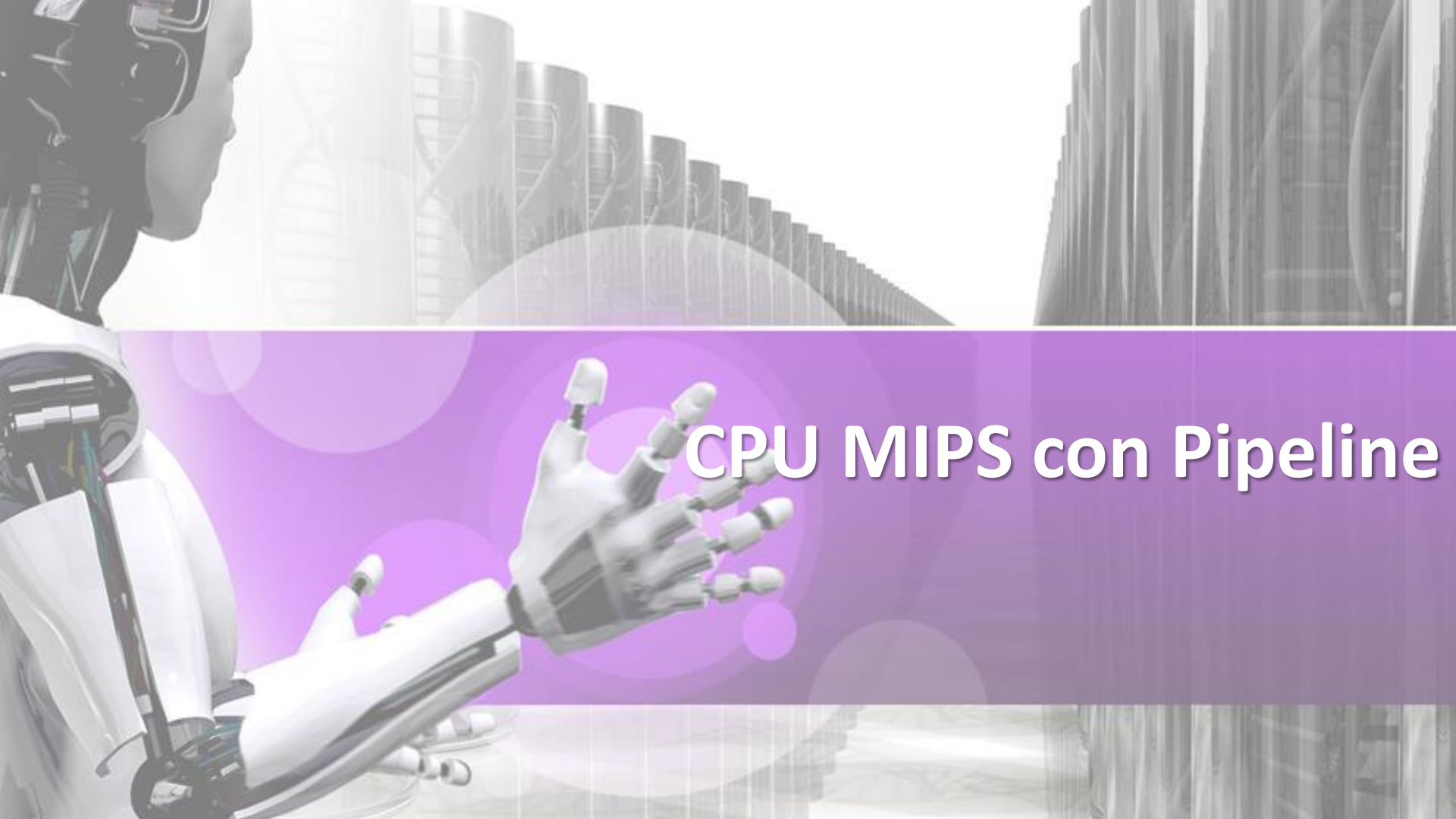
Δt	0	1	2	3	4	5	6	7	8	9	10
I1	F	D	E	M	WB						
I2		F	D	E	M	WB					
I3			F	D	E	M	WB				
I4				F	D	E	M	WB			
I5					F	D	E	M	WB		
I6						F	D	E	M	WB	
I7							F	D	E	M	WB

In questo esempio è integrato il forwarding

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 è incondizionato il riconoscimento alla fase **D** consente di «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

- 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

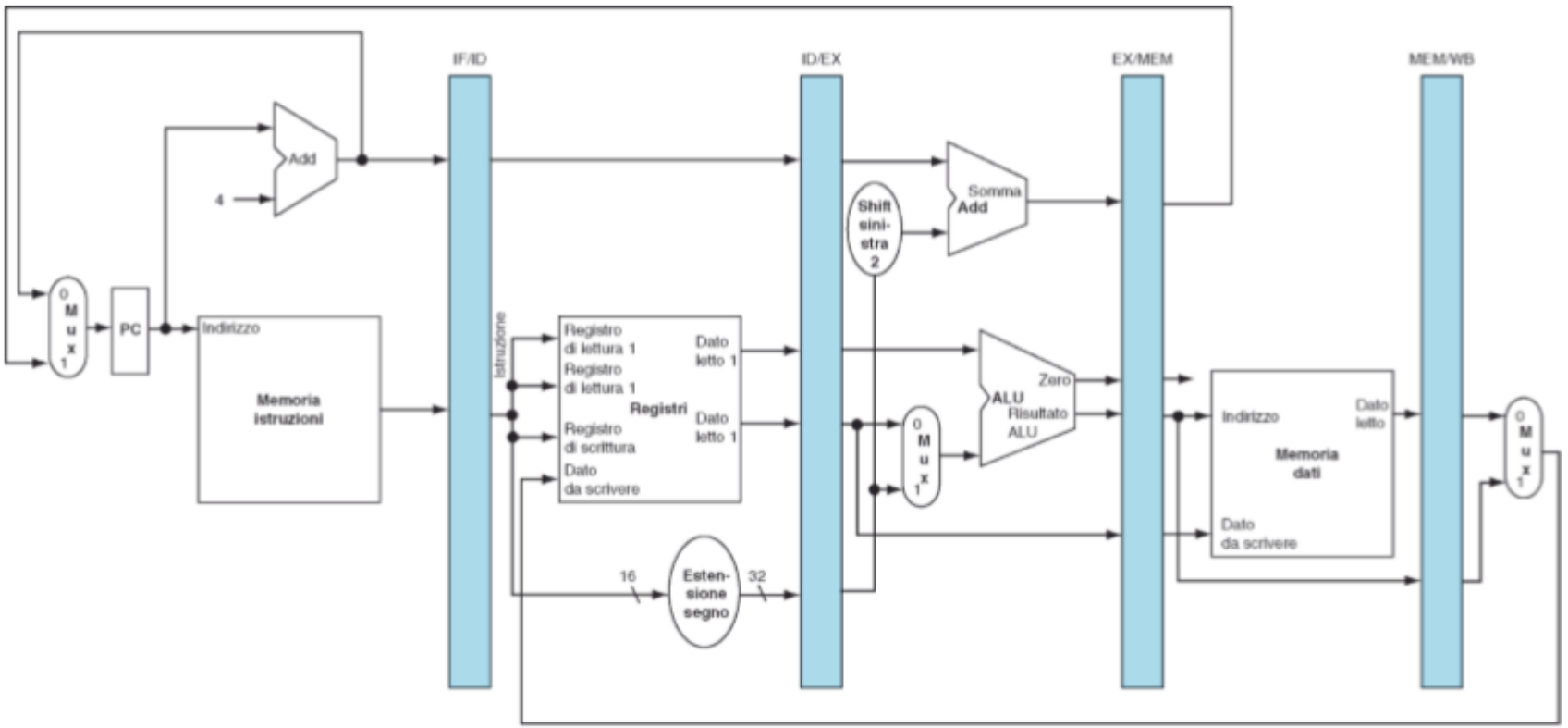
PIPELINE MIPS

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)

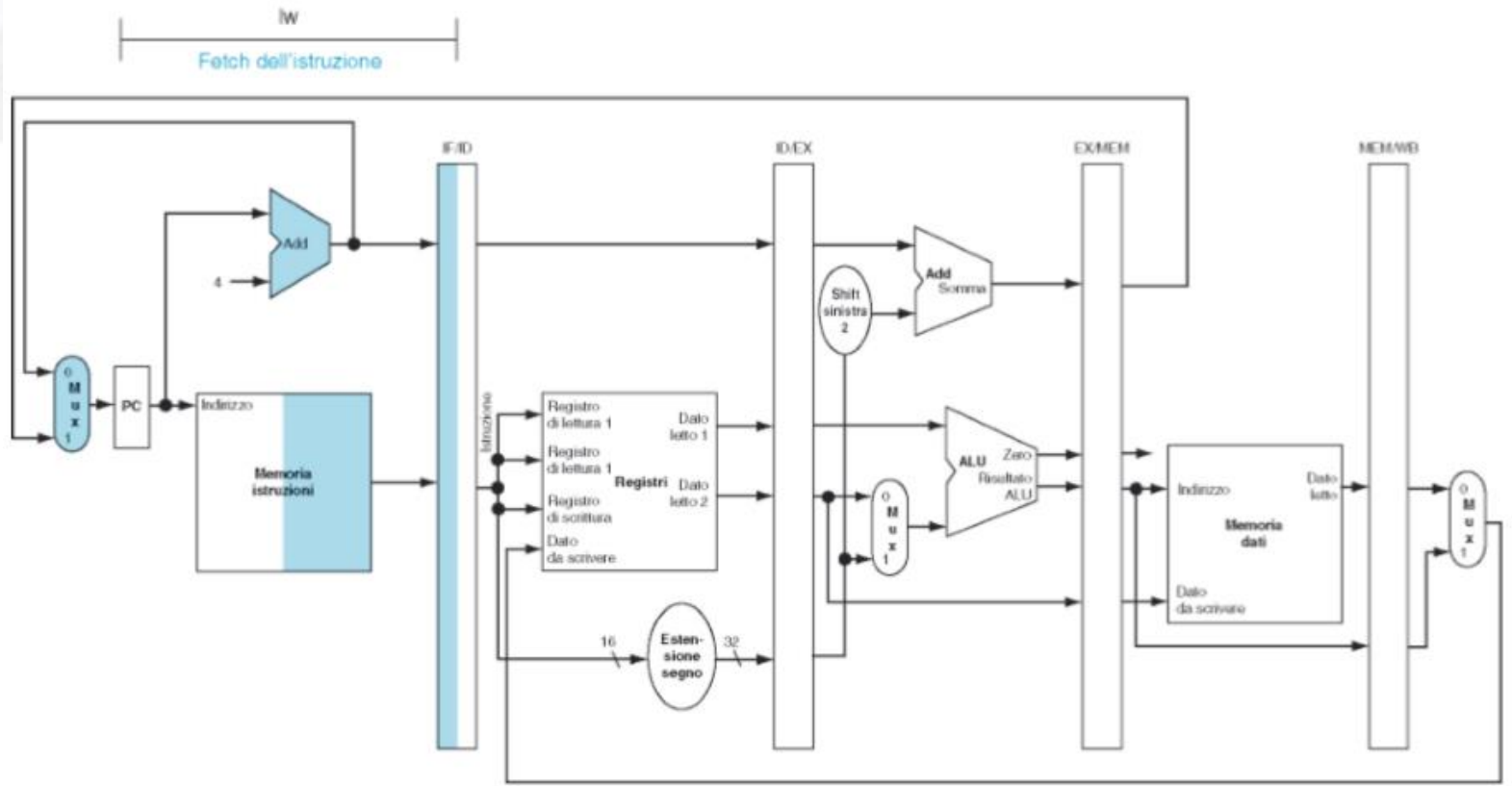
PIPELINE MIPS

Struttura di base



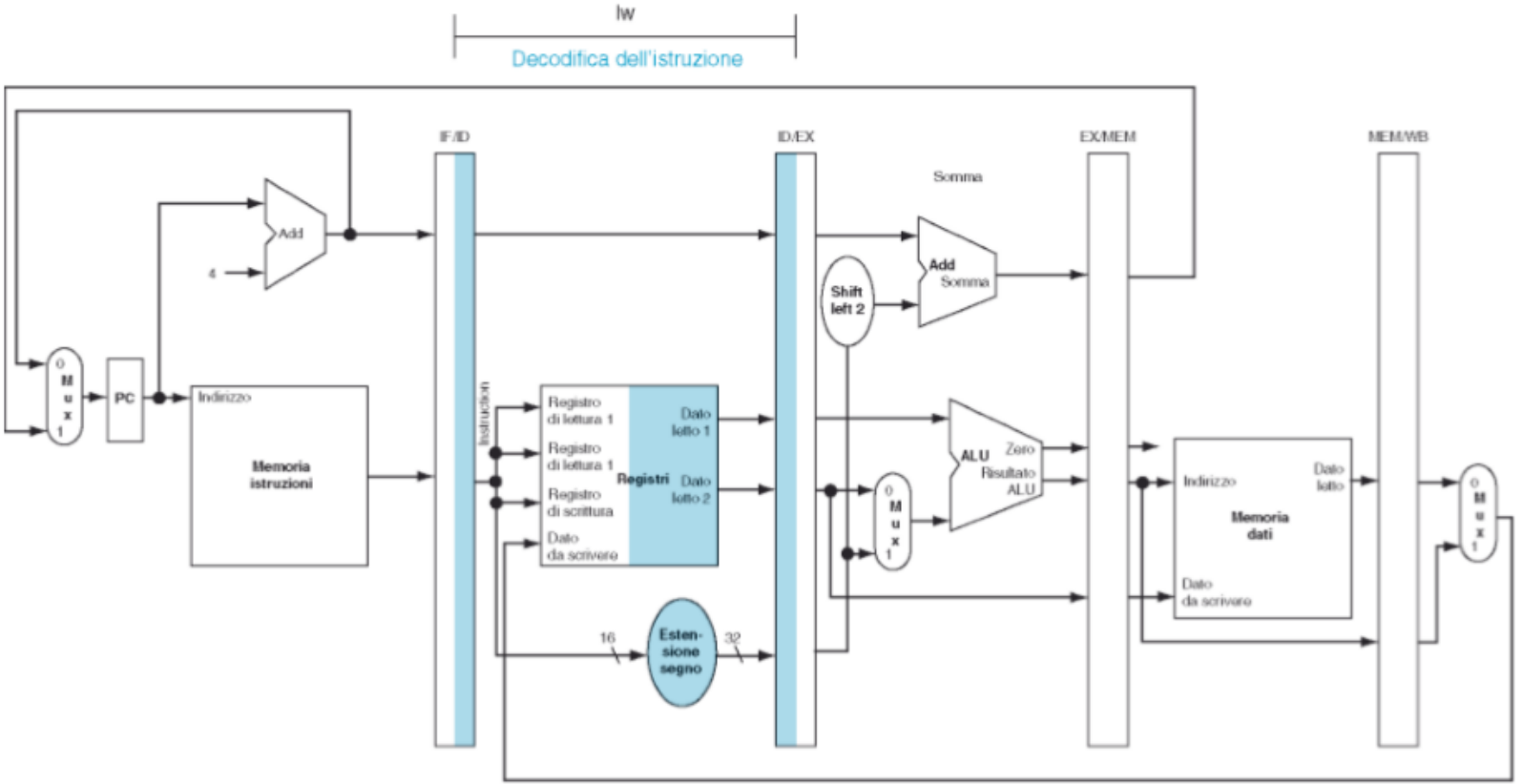
PIPELINE MIPS

Componenti coinvolte nel fetch



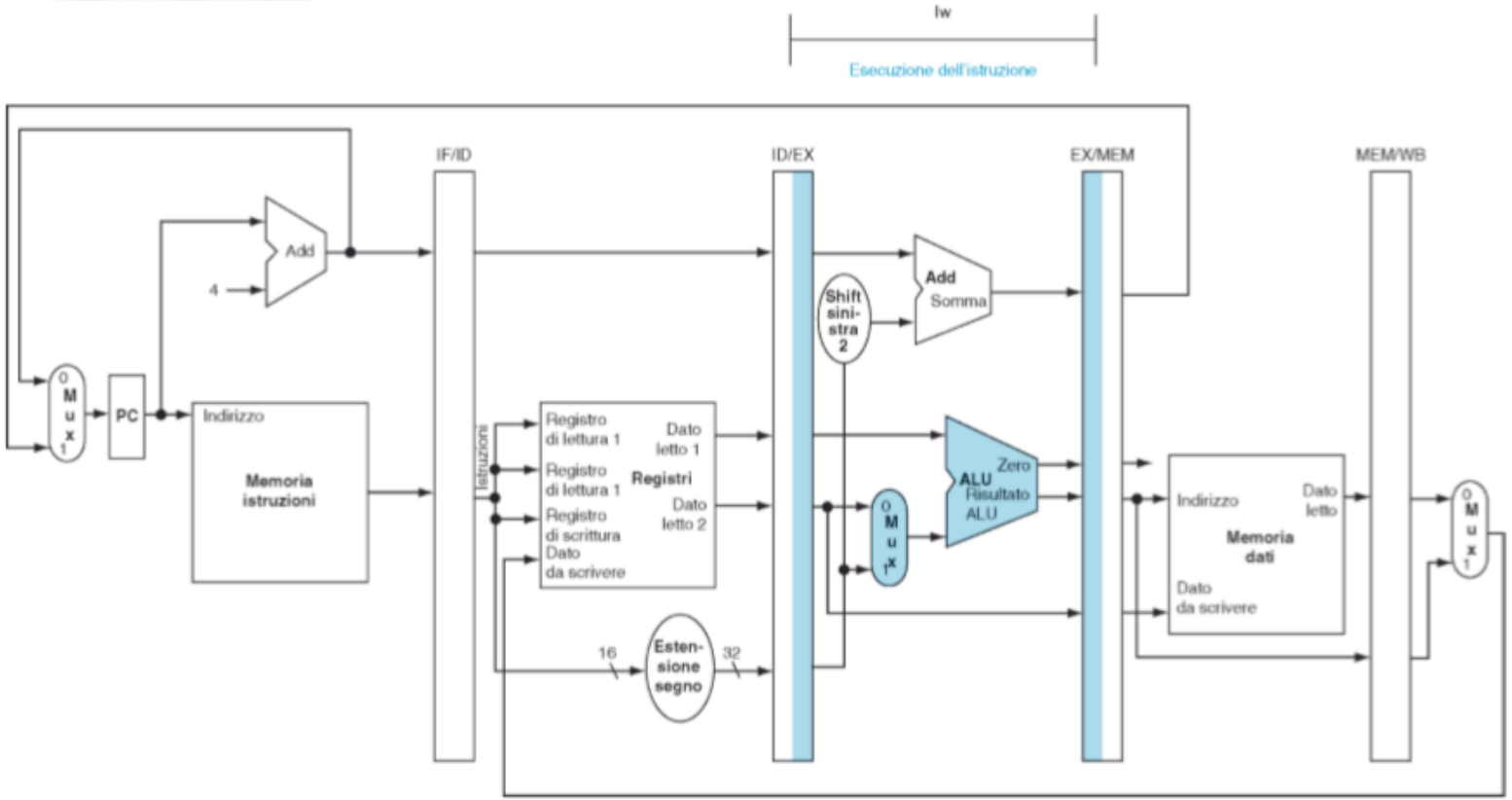
PIPELINE MIPS

Componenti coinvolte nella decodifica e lettura registri



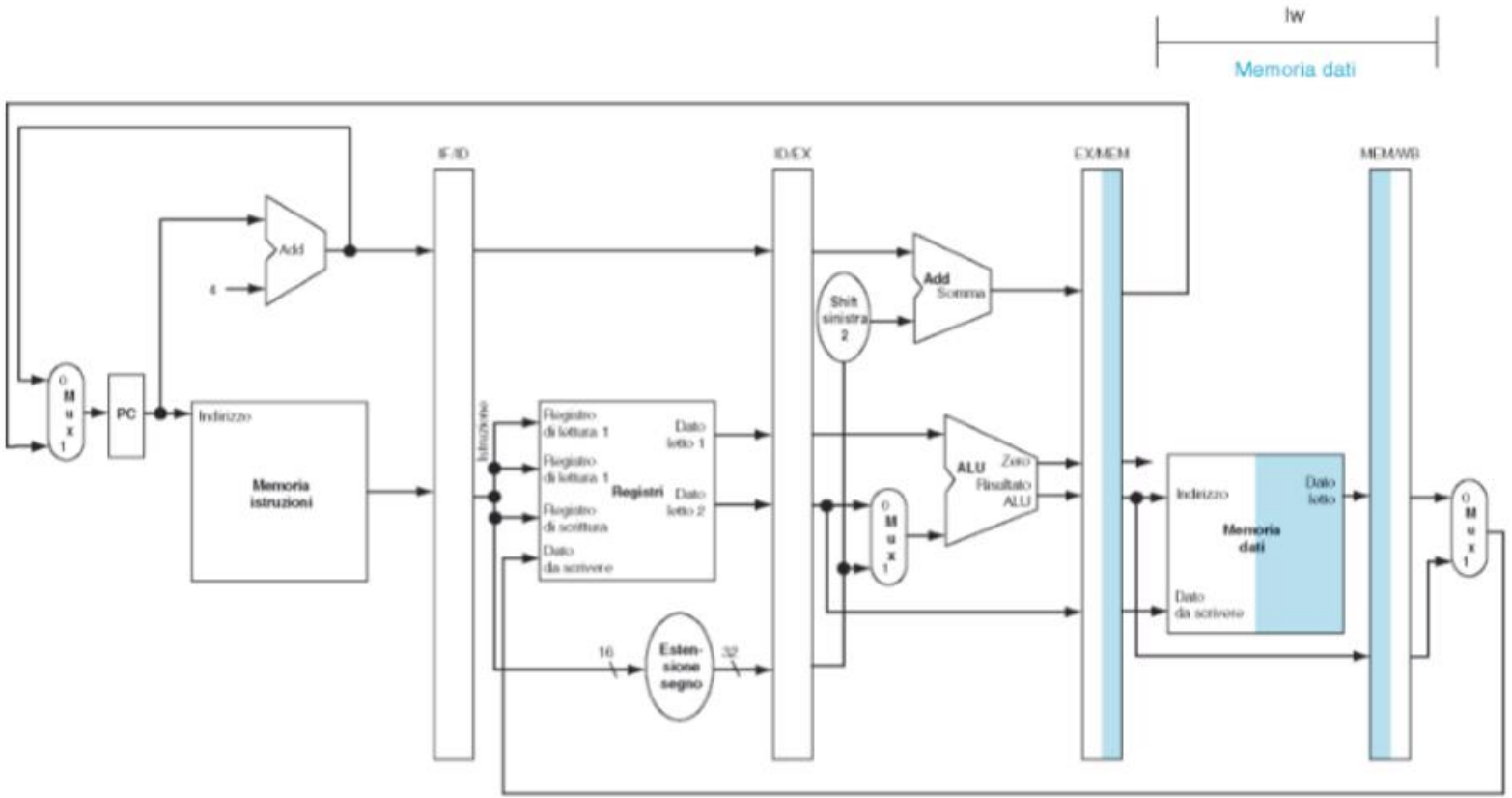
PIPELINE MIPS

Componenti coinvolte nell'esecuzione o calcolo indirizzo



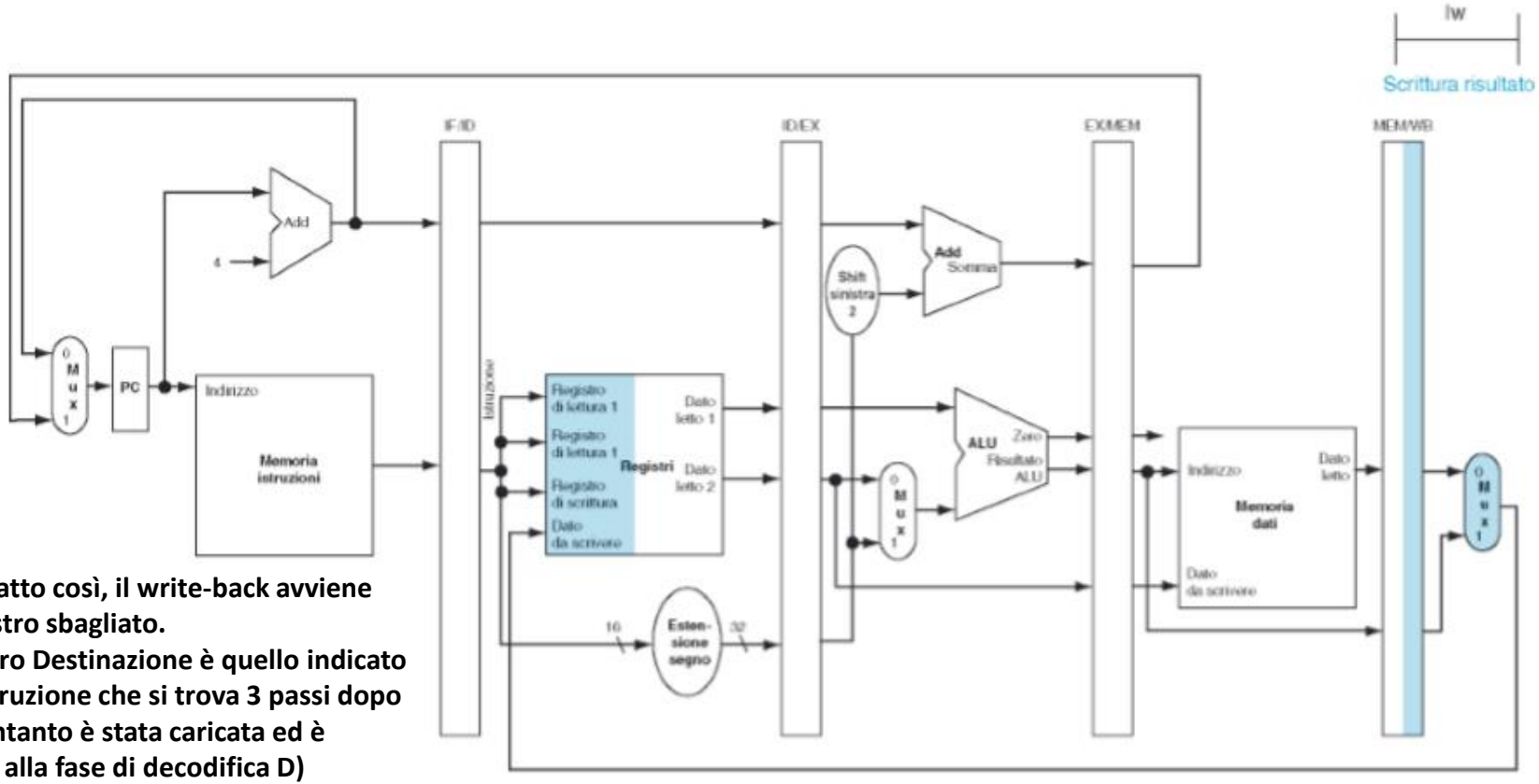
PIPELINE MIPS

Componenti coinvolte nell'accesso alla memoria



PIPELINE MIPS

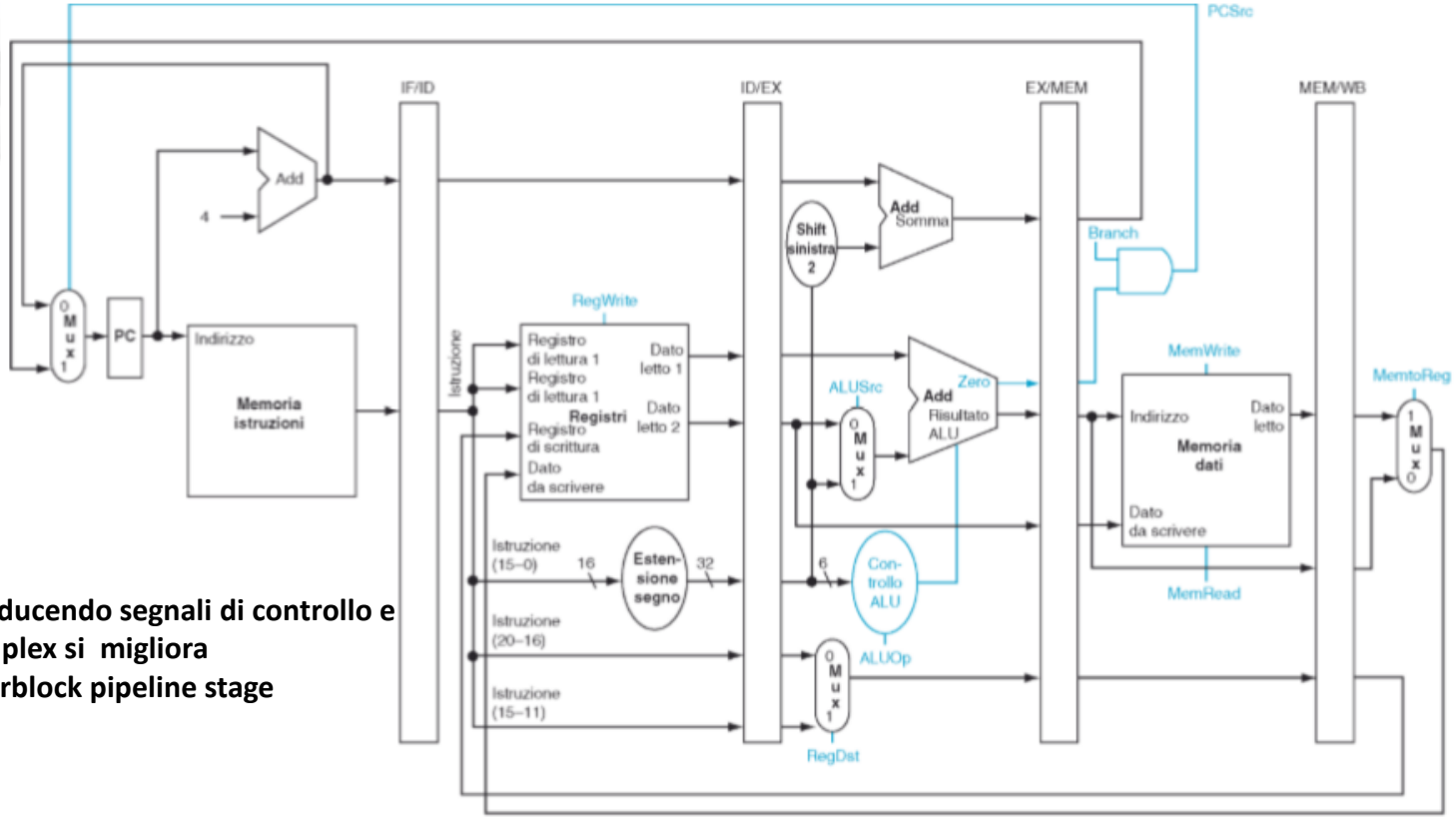
Componenti coinvolte nella scrittura registri (WB)



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

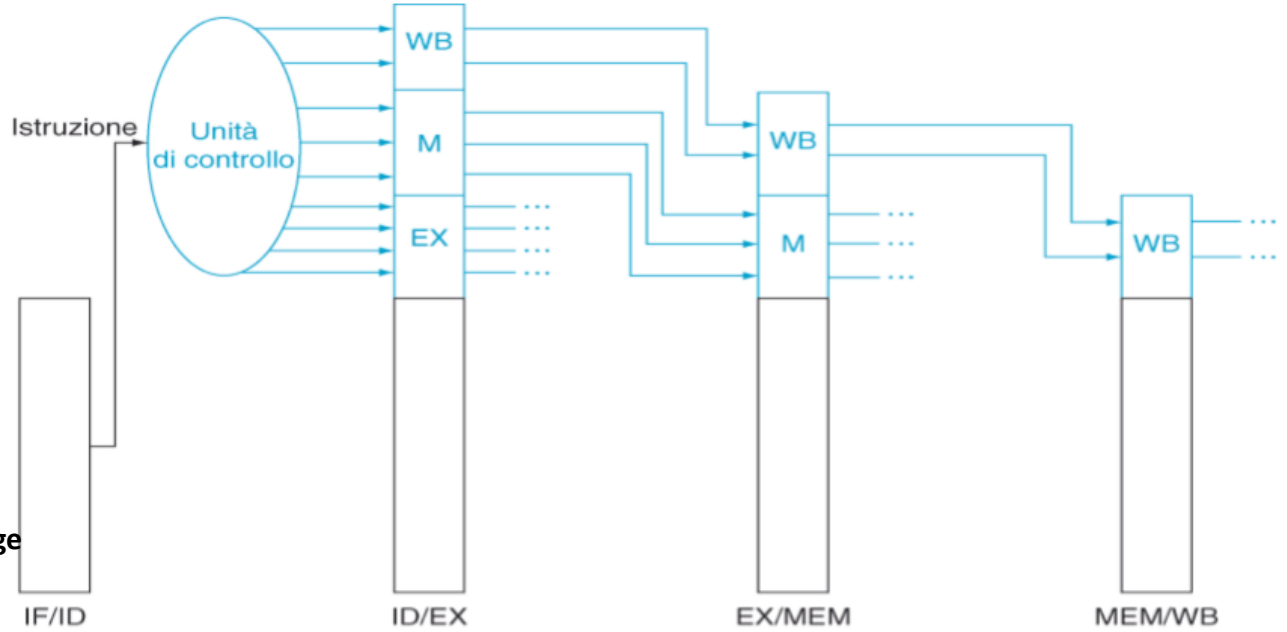


Introducendo segnali di controllo e multiplex si migliora l'interblock pipeline stage

PIPELINE MIPS

I registri del pipeline MIPS

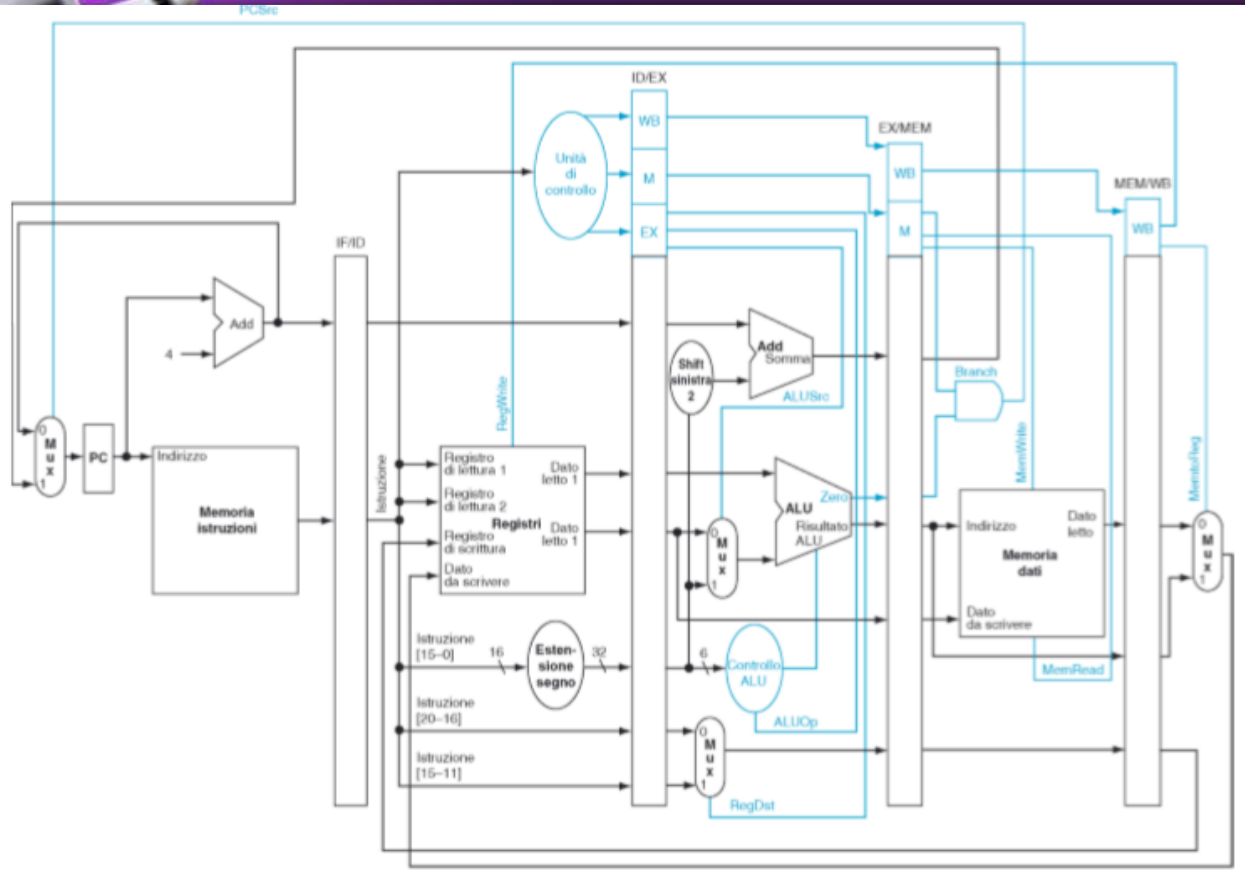
- Ad ogni fase nei **registri** del pipeline sono presenti i **dati** (letti dai registri o dalla parte immediata) e i **comandi** (generati dalla CU)



L'impiego di registri e di comandi migliora l'interblock pipeline stage

PIPELINE MIPS

CPU MIPS



Considerazioni sugli hazard

PIPELINE MIPS

CPU MIPS: una considerazione sui data hazard

- 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: la istruzione precedente ha RegWrite = true
  - **Problema: stallo**
  - **Soluzione: FORWARDING o eliminazione stallo**

PIPELINE MIPS

Soluzioni alle criticità: propagazione (forwarding)

- Nel MIPS, per fermare l'istruzione con uno stallo, si deve (nella fase di Decodifica):
  - Annullare l'istruzione che deve attendere (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

PIPELINE MIPS

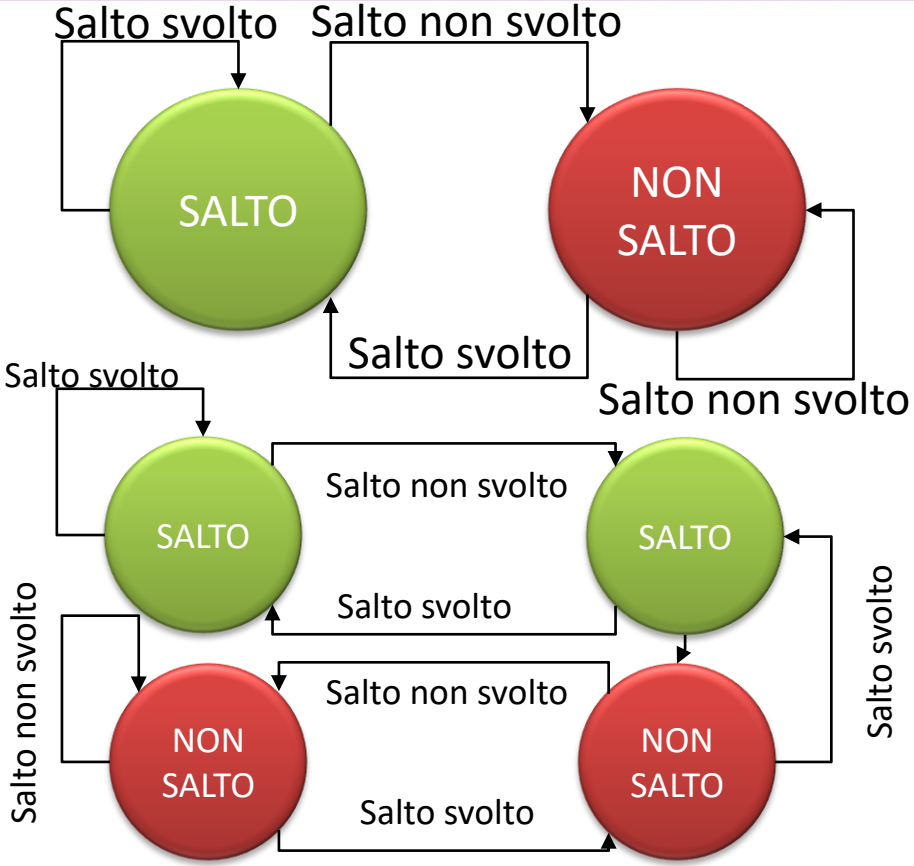
CPU MIPS: una considerazione sul control hazard

- **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)**

PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

- 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 sia 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)



PIPELINE MIPS

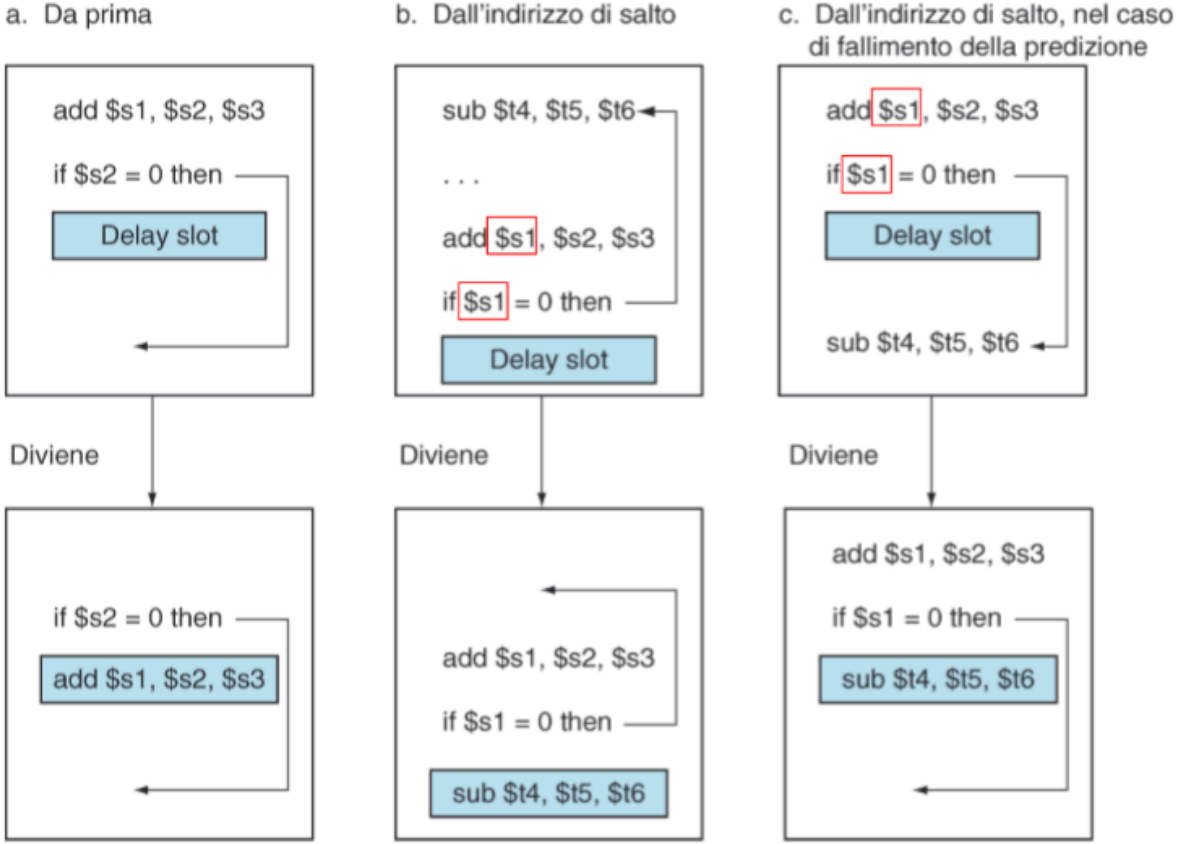
CPU MIPS: una considerazione sul control hazard

- Per recuperare il tempo perso dallo stallo si può **ritardare il salto** (***delay slot***) ovvero eseguire in ogni caso l'istruzione che segue il salto condizionato
- 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)

PIPELINE MIPS

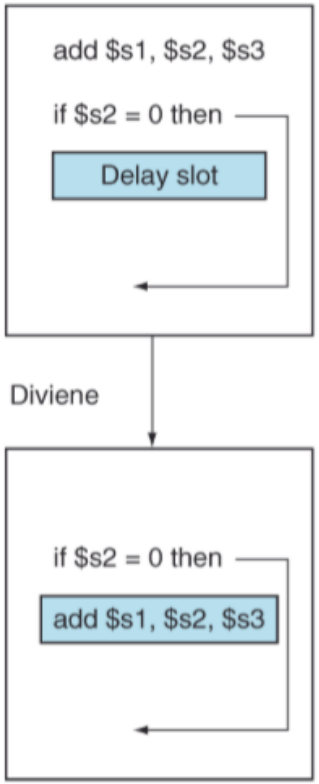
CPU MIPS: una considerazione sul control hazard



PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

a. Da prima



PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

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					

4 istruzioni caricate ma inutili

PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

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										
I10									F	D	E	M	W						

Delay slot

3 istruzioni caricate ma inutili

PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

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								

Forwarding

1 istruzione caricata ma inutile

PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

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									

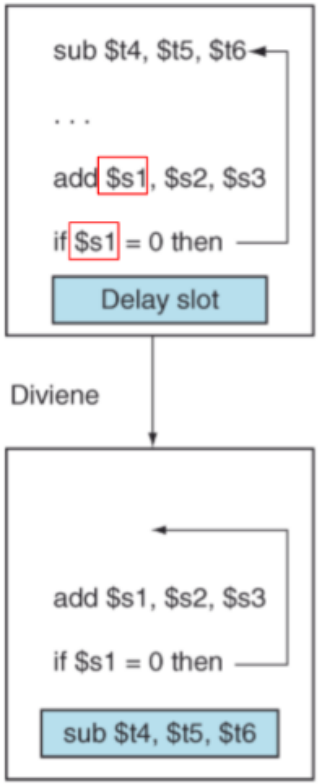
nessuna istruzione caricata Inutilmente se predizione confermata

Predizione salto condizionato o riconoscimento salto incondizionato

PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

b. Dall'indirizzo di salto



PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

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		

4 istruzioni caricate ma inutili

PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

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			

Delay slot

3 istruzioni caricate ma inutili

PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

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			

Forwarding

1 istruzioni caricate ma inutili

PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

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					

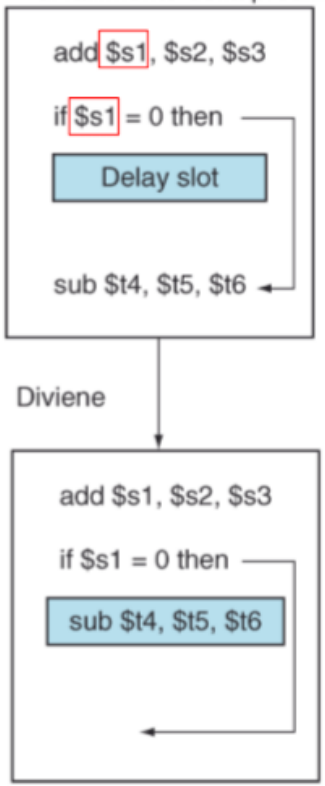
nessuna istruzione caricata Inutilmente se predizione confermata

Predizione salto condizionato o riconoscimento salto incondizionato

PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

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



PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

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

CPU MIPS: una considerazione sul control hazard

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				

Delay slot

3 istruzioni caricate ma inutili

PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

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						

Forwarding

1 istruzione caricate ma inutile

PIPELINE MIPS

CPU MIPS: una considerazione sul control hazard

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							

nessuna istruzione caricata Inutilmente se predizione confermata

Predizione salto condizionato o riconoscimento salto incondizionato

PIPELINE MIPS

CPU MIPS: una considerazione sullo hazard strutturale

- L'hazard strutturale MIPS può verificarsi quando l'architettura del processore non è in grado di gestire correttamente le dipendenze strutturali tra le istruzioni
- Le dipendenze strutturali si verificano quando più istruzioni richiedono lo stesso hardware (ad esempio, la stessa unità di esecuzione) contemporaneamente
- Un esempio di hazard strutturale MIPS potrebbe verificarsi in una situazione in cui il processore deve eseguire due istruzioni che richiedono l'accesso allo stesso modulo di memoria o allo stesso registro simultaneamente. Se il design del processore non è in grado di gestire efficacemente questa situazione, potrebbe verificarsi un hazard strutturale, rallentando l'esecuzione delle istruzioni o causando errori nell'esecuzione del programma
- Per mitigare gli hazard strutturali MIPS, sono state sviluppate varie tecniche di progettazione dei processori, come l'implementazione di buffer e pipeline più complessi, l'uso di tecniche di predizione delle dipendenze, l'allocazione dei registri e l'implementazione di tecniche di scheduling delle istruzioni

PIPELINE MIPS

CPU MIPS: una considerazione sullo hazard strutturale

Il seguente codice tenta di caricare due valori da memoria e memorizzarli in due registri diversi:

```
add $t0, $s0, $zero # Aggiunge il contenuto di $s0 a zero e memorizza il risultato in $t0
add $t1, $s1, $zero # Aggiunge il contenuto di $s1 a zero e memorizza il risultato in $t1
```

Se il processore MIPS ha un'unica ALU allora si verificherà un hazard strutturale. Questo accade perché entrambe le istruzioni devono usufruire della ALU e non possono essere eseguite contemporaneamente

Per risolvere questo problema, potrebbe essere necessario aggiungere cicli di attesa o utilizzare altre tecniche di progettazione per gestire correttamente gli accessi alla ALU in modo che non si verifichino hazard strutturali. Ad esempio, si potrebbe ristrutturare il codice in modo che gli accessi alla ALU non siano dipendenti o si deve ricorrere al forwarding


Fine