

Esercizi sulla Pipeline

Architettura degli elaboratori

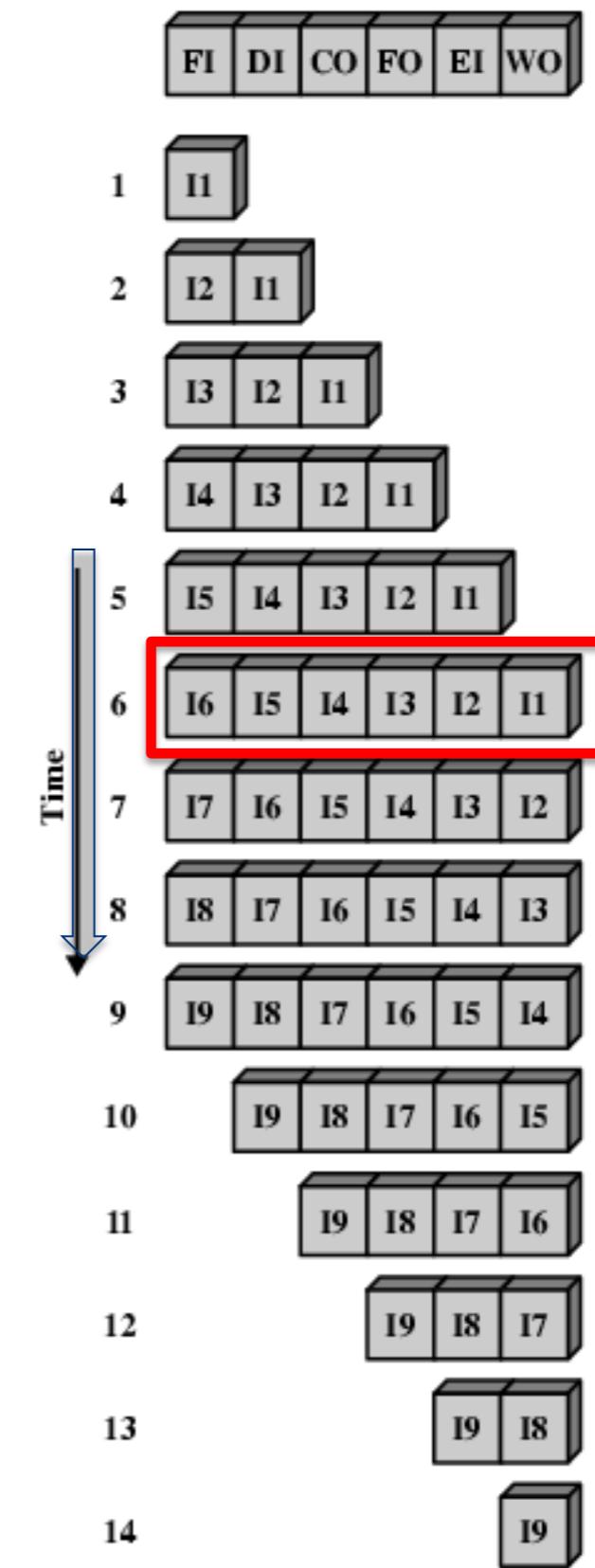
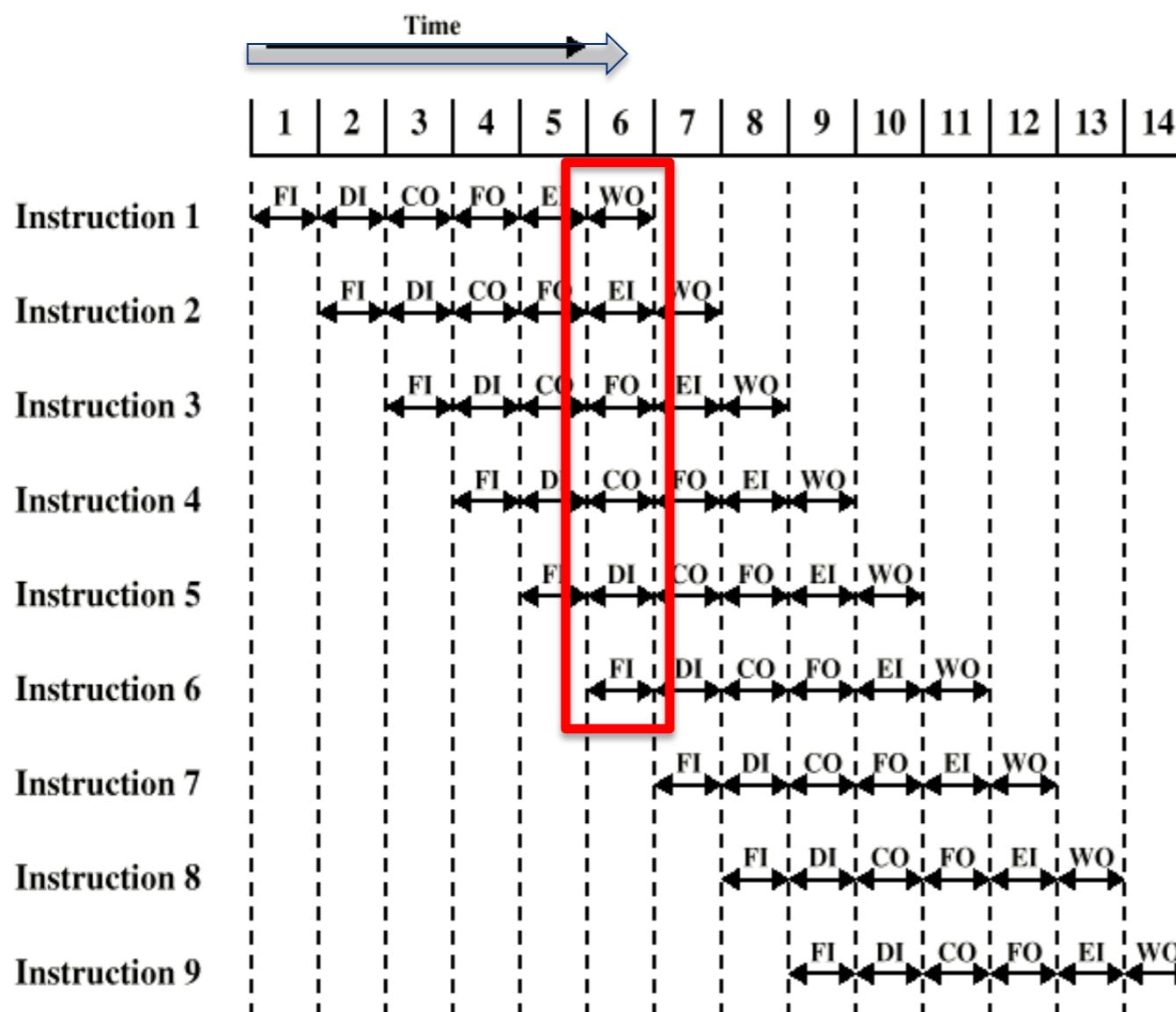
Laurea in Informatica

Docente: Federico Corò

Pipeline: overview

- Decomporre il lavoro in fasi
 - Indipendenti e di durata simile
- Esecutori specializzati, che possono eseguire in parallelo.
- Ad esempio:
 - **fetch (FI)** lettura dell'istruzione
 - **decodifica (DI)** decodifica dell'istruzione
 - **calcolo ind. op. (CO)** calcolo indirizzo effettivo operandi
 - **fetch operandi (FO)** lettura degli operandi in memoria
 - **esecuzione (EI)** esecuzione dell'istruzione
 - **scrittura (WO)** scrittura del risultato in memoria

Pipeline



Tempi Pipeline

- **Tempo di ciclo:** tempo per far avanzare di uno stadio l'esecuzione

$$\tau = \max_i(\tau_i) + d \quad 1 \leq i \leq k$$

Tempo di esecuzione dello stadio più oneroso

Numero di stadi pipeline

Ritardo di commutazione di registro

- **Performance ideali:** tempo totale per eseguire n istruzioni (no salti)

$$T_k = (k + (n - 1))\tau$$

- k cicli per la prima istruzione
- n-1 cicli per le altre n-1 istruzioni

Non diminuisce il tempo di esecuzione della singola istruzione, ma il throughput

Speedup Pipeline

- Fattore di velocizzazione: tempo per far avanzare di uno stadio l'esecuzione

n istruzioni senza pipeline: 1
singolo stadio di durata $k \tau$

$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{(k + n - 1)\tau} = \frac{nk}{k + n - 1}$$

Pipeline Hazards: criticità

- Situazioni in cui l'istruzione successiva non può essere eseguita nel ciclo di clock immediatamente successivo
 - Serve introdurre stalli (o bolle)
 - Non si raggiunge lo speedup massimo

Vari tipi di Hazards:

- Sbilanciamento delle fasi: duree di attesa
- Problemi strutturali: due fasi competono per la stessa risorsa
- Dipendenza dai dati
- Dipendenza dal controllo

Nella pipeline RISC-V
non le consideriamo

Pipeline Hazards: Dipendenza dai dati

- una fase non può essere eseguita in un certo ciclo di clock perchè **dati** di cui ha bisogno **non sono ancora disponibili**
 - deve attendere che termini l'elaborazione di un'altra fase
 - un dato modificato nell'esecuzione dell'istruzione **corrente** può dover essere utilizzato dalla fase Fetch (FO) dell'istruzione **successiva**
 - add **x1, x2, x3** $R1 \leftarrow [R2] + [R3]$
 - sub **x4, x1, x5** $R4 \leftarrow [R1] - [R5]$
 - la seconda istruzione dipende dal risultato della prima, che si trova ancora all'interno della pipeline!

Data Hazards

istruzione i
istruzione $i+1$

- **Read after Write** : “lettura dopo scrittura” (esempio di prima – $i+1$ legge **prima** che i abbia scritto)
- **Write after Write** : “scrittura dopo scrittura” – $i+1$ scrive **prima** che i abbia scritto (problema con parallelismo)
- **Write after Read**: “scrittura dopo lettura”
– $i+1$ scrive **prima** che i abbia letto (caso raro in pipeline)

Soluzioni:

1. Introduzione di stalli
2. Data forwarding (In Prossima esercitazione vado richiesto)
3. Riordino delle istruzioni

Pipeline Hazards: dipendenza dal controllo

- istruzioni che alterano la sequenzialità, es. salti
- Es. Per salto condizionato: quando è chiaro che il salto deve essere preso, tutto il lavoro che ha fatto la pipeline nel frattempo sulle istruzioni successive va scartato
 - Peggiora il parallelismo
- Si possono individuare le istruzioni critiche e aggiungere un'apposita logica di controllo. si ***complica il compilatore e hardware*** specifico
 - Ad esempio, predizione dei salti: cerca di predire se un salto verrà preso o no

Esercizio 1: Dipendenze

Che tipo di dipendenze si possono prevedere guardando questo codice sorgente?

```
if (a > c) {  
    d = d + 5;  
    a = b + d + e;  
}  
  
else {  
    e = e + 2;  
    f = f + 2;  
    c = c + f;  
}  
  
b = a + f;
```

N.B. Le dipendenze non dipendono dalla particolare architettura/pipeline

- Sono proprietà del programma!

N.B.2: dipendenze aggiuntive potrebbero presentarsi nel programma assembly!

Sol Esercizio 1: Dipendenze

- Dipendenze dal controllo

```
if (a > c) {  
    d = d + 5;  
    a = b + d + e;  
}  
else {  
    e = e + 2;  
    f = f + 2;  
    c = c + f;  
}  
b = a + f;
```



Sol Esercizio 1: Dipendenze

- Dipendenze dai dati

```
if (a > c) {  
    d = d + 5;  
    a = b + d + e;  
}  
else {  
    e = e + 2;  
    f = f + 2;  
    c = c + f;  
}  
b = a + f;
```

ES1 B

Che tipo di dipendenze si possono prevedere guardando questo codice sorgente?

```
ble x1 x2 else ← Branch if less or equal
    addi x3, x3, 5
    add x1, x3, x4
    add x1, x1, x5
    j end
else: addi x4, x4, 2
    addi x6, x6, 2
    add x2, x2, x6
end: add x5, x1, x6
```

N.B. Le dipendenze non dipendono dalla particolare pipeline

- Sono proprietà del programma!

SOL ES1 B

- Stesse dipendenze di prima + altre:

```
ble x1 x2 else
    addi x3, x3, 5
    add x1, x3, x4
    add x1, x1, x5
    j end
else: addi x4, x4, 2
      addi x6, x6, 2
      add x2, x2, x6
end:  add x5, x1, x6
```

Esercizio 2: Dipendenze

Si consideri il seguente frammento di codice

```
loop:    lw x1, 0(x2)      # x1← mem[0+[x2]]  
  
          addi x1, x1, 1    # x1← [x1]+1  
  
          sw x1, 0(x2)      # mem[0+[x2]] ← [x1]  
  
          add x2, x1, x2    # x2← [x1] + [x2]  
  
          sub x4, x3, x2    # x4← [x3] - [x2]  
  
          bne x4,x0, loop   # if ([x4] != 0) PC ← indirizzo(loop)
```

Si individuino le dipendenze Read After Write (RAW) e Write after Write (WAW)

Soluzione Esercizio 2: Dipendenze

- Si consideri il seguente frammento di codice

#	codice	R1	R2	R3	R4	commento
1	LOOP: lw x1, 0(x2)	W	R			legge x2, scrive x1
2	addi x1, x1, 1	RW				legge e scrive x1
3	sw x1, 0(x2)	R	R			legge x1 e x2
4	add x2, x1, x2	R	RW			legge x1, legge e scrive x2
5	sub x4, x3, x2		R	R	W	legge x2 e x3, scrive x4
6	bne x4,x0, LOOP				R	legge x4

Linee codice	Spiegazione dipendenza	Tipo
2←1	ADDI legge x1 che è scritto da LW	RAW
2←1	ADDI scrive x1 che è scritto da LW	WAWS
3←2, 3←1	SW legge x1 che è scritto da ADDI, e prima da LW	RAW
4←2, 4←1	ADD legge x1 che è scritto da ADDI, e prima da LW	RAW
5←4	SUB legge x2 che è scritto da ADD	RAW
6←5	BNE legge x4 che è scritto da SUB	RAW

Pipeline RISC-V

Stadi della pipeline:

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

Pipeline RISC-V

- Le fasi IF e ID non dipendono dai segnali di controllo
- Nella Pipeline RISC-V è possibile individuare **tutte le dipendenze dai dati nella fase ID**
- Hazard Detection Unit
 - Inserisce bolle o attiva meccanismi di data forwarding

Pipeline RISC-V

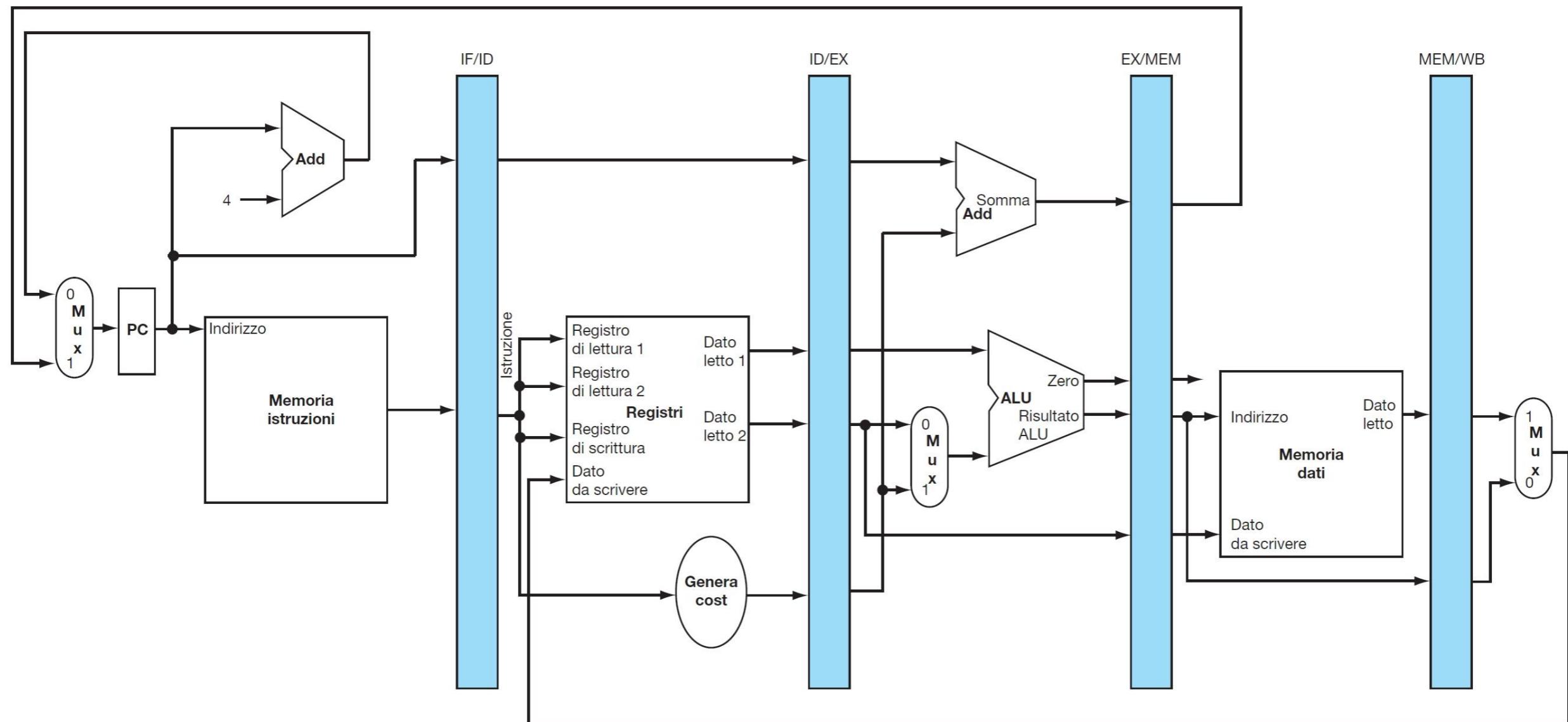
IF

ID

EX

MEM

WB



Pipeline RISC-V

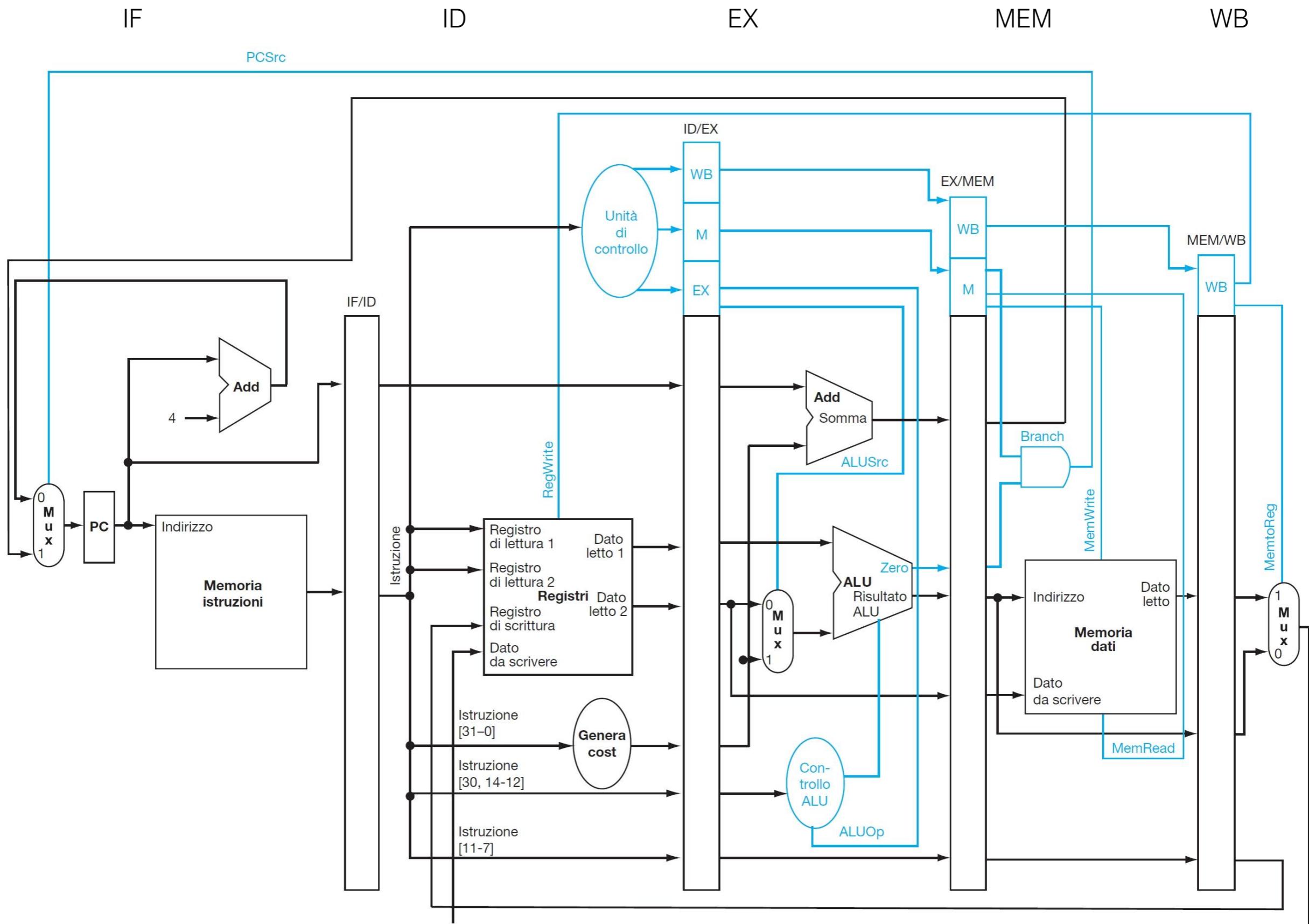


Figura 4.53 L'unità di elaborazione con pipeline della Figura 4.48 completa dei segnali di controllo uscenti dalla porzione dei registri di

Esercizio 3: Pipeline RISC-V e riordino

Si consideri la pipeline RISC-V a 5 stadi vista a lezione, SENZA data-forwarding e con possibilità di scrittura e successiva lettura dei registri in uno stesso ciclo di clock.

Per ognuna delle seguenti sequenze di istruzioni assembler:

1. indicare quali dipendenze dai dati sono presenti;
2. mostrare come evolve la pipeline durante l'esecuzione del codice, spiegando nel dettaglio i motivi di un eventuale stallo
3. indicare se è possibile riordinare le istruzioni in modo da ridurre le dipendenze dai dati.

Sequenza 1

SUB x2, x7, x5
LW x1, 7 (x2)
ADD x2, x1, x8
SW x3, 73 (x1)
ADDI x2, x3, -4
ADDI x7, x3, 8
ADD x1,x7,x2

Sequenza 2

LW x3, 80 (x0)
ADD x2, x3, x1
LW x1, 800(x2)
ADDI x1, x1, -3
ADDI x2, x2, 4
SW x1, 108(x2)
SUBx4,x3,x1

Sequenza 3

SW x9, 0 (x1)
LW x1, 7 (x9)
SUB x9, x1, x8
SW x3, 73 (x9)
ADDI x9, x3, -9
SW x7, 78 (x9)
LW x9,A(x7)

Esercizio 3: Pipeline RISC-V e riordino

1. indicare quali dipendenze dai dati sono presenti;
2. mostrare come evolve la pipeline durante l'esecuzione del codice, spiegando nel dettaglio i motivi di un eventuale stallo

SUB x2, x7, x5

LW x1, 7 (x2)

ADD x2, x1, x8

SW x3, 73 (x1)

ADDI x2, x3, -4

ADDI x7, x3, 8

ADD x1, x7, x2

R2 <- [R7]-[R5]

R1 <- mem[7+[R2]]

R2 <- [R1]+[R8]

mem[73+[R1]] <- [R3]

R2 <- [R3]-4

R7 <- [R3]+8

R1 <- [R7]+[R2]

Esercizio 3: Pipeline RISC-V e riordino

SUB x2, x7, x5

LW x1, 7 (x2)

ADD x2, x1, x8

SW x3, 73 (x1)

ADDI x2, x3, -4

ADDI x7, x3, 8

ADD x1, x7, x2

1	2	3	4	5	6	7	8	9	10
IF	ID	EX	MEM	WB					

Esercizio 3: Pipeline RISC-V e riordino

SUB x2, x7, x5

RAW LW x1, 7 (x2)

ADD x2, x1, x8

SW x3, 73 (x1)

ADDI x2, x3, -4

ADDI x7, x3, 8

ADD x1, x7, x2

1	2	3	4	5	6	7	8	9	10
IF	ID	EX	MEM	WB					
	IF	ID	RD	RD	EX	MEM	WB		

Qui si accorge che c'è la dipendenza
e vanno inseriti stalli

Esercizio 3: Pipeline RISC-V e riordino

RAW: qui si accorge che c'è la dipendenza e vanno inseriti stalli

1	2	3	4	5	6	7	8	9	10	11
IF	ID	EX	MEM	WB						
	IF	ID	IF	IF	EX	MEM	WB			
		IF	IF	IF	ID	IF	IF	EX	MEM	WB

SUB x2, x7, x5

LW x1, 7 (x2)

WAW/
RAW ADD x2, x1, x8

SW x3, 73 (x1)

ADDI x2, x3, -4

ADDI x7, x3, 8

ADD x1, x7, x2

ID non può iniziare prima che l'istruzione precedente passi alla fase successiva

In RISC-V la lettura degli operandi avviene nello stadio ID

Esercizio 3: Pipeline RISC-V e riordino

SUB x2, x7, x5
LW x1, 7 (x2)
ADD x2, x1, x8
RAW SW x3, 73 (x1)
ADDI x2, x3, -4
ADDI x7, x3, 8
ADD x1, x7, x2

1	2	3	4	5	6	7	8	9	10	11	12
IF	ID	EX	MEM	WB							
	IF	ID	IF	IF	EX	MEM	WB				
		IF	IF	IF	ID	IF	IF	EX	MEM	WB	
					IF	IF	IF	ID	EX	MEM	WB

ID non può iniziare prima che l'istruzione precedente passi alla fase successiva

OK: ID è già dopo la WB della LW

Esercizio 3: Pipeline RISC-V e riordino

SUB x2, x7, x5
LW x1, 7 (x2)
ADD x2, x1, x8
SW x3, 73 (x1)
ADDI x2, x3, -4
ADDI x7, x3, 8
ADD x1, x7, x2

WAW
/RAR

1	2	3	4	5	6	7	8	9	10	11	12	13
IF	ID	EX	MEM	WB								
	IF	ID	IF	IF	EX	MEM	WB					
		IF	IF	IF	ID	IF	IF	EX	MEM	WB		
					IF	IF	IF	ID	EX	MEM	WB	
								IF	IF	EX	MEM	WB

IF non può iniziare prima che
l'istruzione precedente passi alla fase
successiva

OK: dipendenze non problematiche

Esercizio 3: Pipeline RISC-V e riordino

SUB x2, x7, x5
LW x1, 7 (x2)
ADD x2, x1, x8
SW x3, 73 (x1)
ADDI x2, x3, -4
RAR ADDI x7, x3, 8
ADD x1, x7, x2

1	2	3	4	5	6	7	8	9	10	11	12	13	14
IF	ID	EX	MEM	WB									
	IF	ID	IF	IF	EX	MEM	WB						
		IF	IF	IF	ID	IF	IF	EX	MEM	WB			
					IF	IF	IF	ID	EX	MEM	WB		
								IF	ID	EX	MEM	WB	
									IF	ID	EX	MEM	WB



OK: x3 è già stato usato in lettura

Esercizio 3: Pipeline RISC-V e riordino

SUB x2, x7, x5
LW x1, 7 (x2)
ADD x2, x1, x8
SW x3, 73 (x1)
ADDI x2, x3, -4
ADDI x7, x3, 8
RAW ADD x1, x7, x2

5	6	7	8	9	10	11	12	13	14	15	16	17
WB												
ID	EX	MEM	WB									
IF	ID	ID	ID	EX	MEM	WB						
IF	IF	IF	ID	EX	MEM	WB						
			IF	ID	EX	MEM	WB					
				IF	ID	EX	MEM	WB				
					IF	ID	EX	MEM	WB			
						IF	ID	ID	ID	EX	MEM	WB

RAW: x7 è scritto dall'istruzione precedente.
Vanno inseriti stalli.
RAW anche su x2, ma risolto dagli stalli
inseriti per x7

Esercizio 3 seq 1: riordino?

- Idea: riordino le istruzioni per diminuire il tempo totale di esecuzione
 - Devo ridurre le dipendenze tra istruzioni che generano stalli

SUB **x2**, **x7**, x5

LW **x1**, 7 (x2)

ADD **x2**, **x1**, x8

SW **x3**, 73 (x1)

ADDI **x2**, **x3**, -4

ADDI **x7**, **x3**, 8

ADD **x1**, **x7**, **x2**

Esercizio 3 seq 1: riordino?

```
    SUB x2, x7, x5  
    LW x1, 7 (x2)  
    ADD x2, x1, x8  
    SW x3, 73 (x1)  
    ADDI x2, x3, -4  
    ADDI x7, x3, 8  
    ADD x1, x7, x2
```

deve leggere il contenuto di x2, che è definito nell'istruzione precedente, quindi non si può anticipare

se si anticipa ADD prima di LW, si modifica il contenuto di x2, quindi LW carica indirizzo diverso e il programma cambia semantica

SW si può scambiare con ADD, ma non elimina la necessità dello stalllo perché anche SW legge x1

deve venire dopo le due precedenti perché legge x7 e x2

Esercizio 3: riordino?

SUB x2, x7, x5

LW x1, 7 (x2)

ADD x2, x1, x8

SW x3, 73 (x1) ← posso posticiparla?

ADDI x2, x3, -4

ADDI x7, x3, 8 ← Non creava problemi, posso anticiparla?

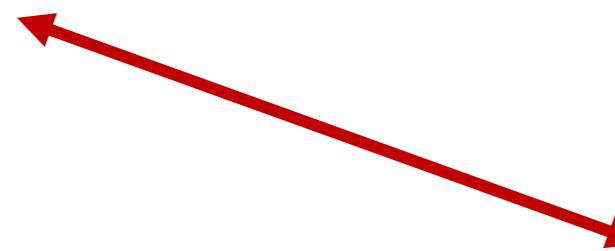
ADD x1, x7, x2

Entrambe le possibilità migliorano il tempo totale

Esercizio 3 sequenza 1: riordino?

SUB x2, x7, x5
LW x1, 7 (x2)
ADD x2, x1, x8
SW x3, 73 (x1)
ADDI x2, x3, -4
ADDI x7, x3, 8
ADD x1, x7, x2

SUB x2, x7, x5
LW x1, 7 (x2)
ADD x2, x1, x8
ADDI x2, x3, -4
ADDI x7, x3, 8
SW x3, 73 (x1)
ADD x1, x7, x2



Esercizio 3: Pipeline RISC-V e riordino

SUB x2, x7, x5
 LW x1, 7 (x2)
 ADD x2, x1, x8
 ADDI x2, x3, -4
 ADDI x7, x3, 8
 SW x3, 73 (x1)
 ADD x1, x7, x2

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
IF	ID	EX	MEM	WB											
	IF	ID	IF	IF	EX	MEM	WB								
		IF	IF	IF	ID	IF	IF	EX	MEM	WB					
					IF	IF	IF	ID	EX	MEM	WB				
								IF	ID	EX	MEM	WB			
									IF	ID	ME	WB			
										IF	ID	IF	EX	MEM	WB

Esercizio 3: Sequenza 2

1. indicare quali dipendenze dai dati sono presenti;
2. mostrare come evolve la pipeline durante l'esecuzione del codice, spiegando nel dettaglio i motivi di un eventuale stallo

LW x3, 80 (x0)

R1 <- mem[80+[R0]]

ADD x2, x3, x1

R2 <- [R3]+[R1]

LW x1, 800(x2)

R1 <- mem[800+[R2]]

ADDI x1, x1, -3

R1 <- [R1]-3

ADDI x2, x2, -4

R2 <- [R2]-4

SW x1, 108(x2)

mem[108+[R2]] <- [R1]

SUB x4,x3,x1

R4 <- [R3]-[R1]

Soluzione esercizio 3 sequenza 2

LW x3, 80 (x0)
ADD x2, x3, x1
LW x1, 800(x2)
ADDI x1, x1, -3
ADDI x2, x2, 4
SW x1, 108(x2)
SUB x4,x3, x1

The diagram illustrates the execution of seven instructions across 19 steps. The timeline is divided into four main phases: IF (Instructions Fetch), ID (Instruction Decode), EX (Execution), and MEM (Memory Access). The first three rows show the initial execution of the first three instructions. The next four rows show the execution of the remaining four instructions. Red annotations highlight dependencies and RAW hazards.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
IF	IF	ID	EX	MEM	WB														
IF	IF	ID	ID	ID	EX	MEM	WB												
IF	IF	ID	ID	ID	ID	EX	MEM	WB											
			IF	IF	IF	ID	EX	ID	ID	ID	EX	MEM	WB						
							IF	IF	IF	IF	ID	EX	MEM	WB					
								IF	IF	IF	ID	EX	MEM	WB					
									IF	IF	ID	ID	ID	IF	IF	IF	IF		

Esercizio 3 sequenza 2: riordino?

LW **x3**, 80 (**x0**)

ADD **x2**, **x3**, **x1**

LW **x1**, 800(**x2**)

ADDI **x1**, **x1**, -3

ADDI **x2**, **x2**, 4

SW **x1**, 108(**x2**)

SUB **x4**,**x3**, **x1**

Riordino esercizio 3 seq 2

LW **x3**, 80 (**x0**)

ADD **x2**, **x3**, **x1**

LW **x1**, 800(**x2**)

ADDI **x1**, **x1**, -3

ADDI **x2**, **x2**, 4

SW **x1**, 108(**x2**)

SUB **x4**,**x3**, **x1**

LW **x3**, 80 (**x0**)

ADD **x2**, **x3**, **x1**

LW **x1**, 800(**x2**)

ADDI **x1**, **x1**, -3

ADDI **x2**, **x2**, 4

SUB **x4**,**x3**, **x1**

SW **x1**, 108(**x2**)

Riordino esercizio 3 seq 2

LW x3, 80(x0)
ADD x2, x3, x1
LW x1, 800(x2)
ADDI x1, x1, -3
ADDI x2, x2, 4
SUB x4,x3, x1
SW x1, 108(x2)

