

Soluzione esercizi

Sequenza 1

SUB \$2, \$7, \$5	R2 <- [R7] - [R5]
LW \$1, 7 (\$2)	R1 <- mem[7 + [R2]]
ADD \$2, \$1, \$8	R2 <- [R1] + [R8]
SW \$3, 73 (\$1)	mem[73 + [R1]] <- [R3]
SUBI \$2, \$3, 4	R2 <- [R3] - 4
ADDI \$7, \$3, 8	R7 <- [R3] + 8
ADD \$1, \$7, \$2	R1 <- [R7] + [R2]

Sequenza 1

SUB \$2, \$7, \$5
 LW \$1, 7 (\$2)
 ADD \$2, \$1, \$8
 SW \$3, 73 (\$1)
 SUBI \$2, \$3, 4
 ADDI \$7, \$3, 8
 ADD \$1, \$7, \$2

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

si accorge che il registro di **lettura** IF/ID.IR[rs]
 è uguale al registro di **scrittura** ID/EX.IR[rd] => **RAW**

risolvibile con data forwarding:

EX/MEM.AluOutput inviato a TopAluInput

SUB

R

codop	rs	rt	rd	sham	funct
-------	----	----	----	------	-------

LW

I

codop	rs	rt	address/const
-------	----	----	---------------

Sequenza 1

SUB \$2, \$7, \$5
 LW \$1, 7 (\$2)
 ADD \$2, \$1, \$8
 SW \$3, 73 (\$1)
 SUBI \$2, \$3, 4
 ADDI \$7, \$3, 8
 ADD \$1, \$7, \$2

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

si accorge che il registro di **lettura** IF/ID.IR[rs]
 è uguale al registro di **scrittura** ID/EX.IR[rt] => **RAW su \$1**

ADD

R

codop	rs	rt	rd	sham	funct
-------	----	----	----	------	-------

LW

I

codop	rs	rt	address/const
-------	----	----	---------------

Sequenza

quando ADD scrive su \$2, non c'è alcun conflitto con le precedenti operazioni su \$2

SUB \$2, \$7, \$5
 LW \$1, 7 (\$2)
 ADD \$2, \$1, \$8
 SW \$3, 73 (\$1)
 SUBI \$2, \$3, 4
 ADDI \$7, \$3, 8
 ADD \$1, \$7, \$2

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

si accorge che il registro di **lettura** IF/ID.IR[rs]
 è uguale al registro di **scrittura** ID/EX.IR[rt] => **RAW su \$1**

LW determina il valore da scrivere in \$1 in fase MEM
 quindi serve **1 stallo + data forward**:

MEM/WB.LMD inviato a TopAluInput

Sequenza 1

SUB \$2, \$7, \$5
 LW \$1, 7 (\$2)
 ADD \$2, \$1, \$8
 SW \$3, 73 (\$1)
 SUBI \$2, \$3, 4
 ADDI \$7, \$3, 8
 ADD \$1, \$7, \$2

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

si accorge che il registro di **lettura** IF/ID.IR[rs]
 è uguale al registro di **scrittura della LW** MEM/WB.IR[rt]
 => **RAW su \$1**

ma nella **prima metà** del ciclo 6 WB di LW scrive \$1
 e nella **seconda metà** ID legge \$1, quindi **tutto ok**

Sequenza 1

SUB \$2, \$7, \$5
 LW \$1, 7 (\$2)
 ADD \$2, \$1, \$8
 SW \$3, 73 (\$1)
 SUBI \$2, \$3, 4
 ADDI \$7, \$3, 8
 ADD \$1, \$7, \$2

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

tutto OK:

- \$3 è usato in **lettura** sia da SUBI che da SW
- SUBI scrive su \$2 in fase WB, senza conflitto con i precedenti

Sequenza 1

SUB \$2, \$7, \$5
 LW \$1, 7 (\$2)
 ADD \$2, \$1, \$8
 SW \$3, 73 (\$1)
 SUBI \$2, \$3, 4
 ADDI \$7, \$3, 8
 ADD \$1, \$7, \$2

...	5	6	7	8	9	10	11	12
	WB							
	MEM	WB						
	ID	EX	MEM	WB				
	IF	ID	EX	MEM	WB			
		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
				IF	ID			

tutto OK: \$3 è usato in **lettura** sia da SUBI che da SW

si accorge di **RAW** su \$7 e **RAW** su \$2

- \$7 = IF/ID.IR[rs] = ID/EX.IR[rt]
- \$2 = IF/ID.IR[rt] = EX/MEM.IR[rt]

Sequenza 1

...	5	6	7	8	9	10	11	12
SUB \$2, \$7, \$5	WB							
LW \$1, 7 (\$2)	MEM	WB						
ADD \$2, \$1, \$8	ID	EX	MEM	WB				
SW \$3, 73 (\$1)	IF	ID	EX	MEM	WB			
SUBI \$2, \$3, 4		IF	ID	EX	MEM	WB		
ADDI \$7, \$3, 8			IF	ID	EX	MEM	WB	
ADD \$1, \$7, \$2				IF	ID	EX	MEM	WB

si accorge di **RAW** su \$7 e **RAW** su \$2 risolti con forward:

- \$7 = IF/ID.IR[rs] = ID/EX.IR[rt] EX.AluOutput inviato a TopAluInput
- \$2 = IF/ID.IR[rt] = EX/MEM.IR[rt] MEM.AluOutput va a BottomAluInput

Sequenza 1: Riordino?

- SUB \$2, \$7, \$5 ← deve leggere il contenuto di \$2, che è definito nell'istruzione precedente, quindi non si può anticipare
- LW \$1, 7 (\$2) ←
- ADD \$2, \$1, \$8 ← se si anticipa ADD prima di LW, si modifica il contenuto di \$2, quindi LW carica indirizzo diverso e il programma cambia semantica
- SW \$3, 73 (\$1) ← SW si può scambiare con ADD, ma non elimina la necessità dello stallo perché anche SW legge \$1
- SUBI \$2, \$3, 4
- ADDI \$7, \$3, 8
- ADD \$1, \$7, \$2 ← deve venire dopo le due precedenti perché legge \$7 e \$2

Sequenza 1: Riordino?

SUB \$2, \$7, \$5

LW \$1, 7 (\$2)

ADD \$2, \$1, \$8

SW \$3, 73 (\$1)

SUBI \$2, \$3, 4

ADDI \$7, \$3, 8

ADD \$1, \$7, \$2

non avevano dipendenze, provo ad anticiparle per allontanare la dipendenza RAW su \$1 che genera lo **stallo** (ricorda che SW legge \$3)

SUB \$2, \$7, \$5

LW \$1, 7 (\$2)

SUBI \$2, \$3, 4

ADDI \$7, \$3, 8

ADD \$2, \$1, \$8 ← legge \$1 corretto

SW \$3, 73 (\$1) ←

ADD \$1, \$7, \$2 ← NO: legge \$2 definito dalla ADD invece che da SUBI

e se anticipo
anche questa subito dopo ADDI, allora sovrascrive \$1

Sequenza 1: Riordino?

SUB \$2, \$7, \$5

LW \$1, 7 (\$2)

ADD \$2, \$1, \$8

SW \$3, 73 (\$1)

SUBI \$2, \$3, 4

ADDI \$7, \$3, 8

ADD \$1, \$7, \$2

SUB \$2, \$7, \$5

LW \$1, 7 (\$2)

ADD \$2, \$1, \$8

SUBI \$2, \$3, 4

ADDI \$7, \$3, 8

SW \$3, 73 (\$1)

ADD \$1, \$7, \$2

SUB \$2, \$7, \$5

LW \$1, 7 (\$2)

ADDI \$7, \$3, 8

ADD \$2, \$1, \$8

SW \$3, 73 (\$1)

SUBI \$2, \$3, 4

ADD \$1, \$7, \$2

toglie il forward doppio

toglie lo stallo

si possono fare entrambi

Sequenza 2

```

LW $3, 80 ($0)
ADD $2, $3, $1
LW $1, 800($2)
SUBI $1, $1, 3
ADDI $2, $2, 4
SW $1, 108($2)
SUB $4, $3, $1

```

Sequenza 2

```

LW $3, 80 ($0)
ADD $2, $3, $1
LW $1, 800($2)
SUBI $1, $1, 3
ADDI $2, $2, 4
SW $1, 108($2)
SUB $4, $3, $1

```

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

si accorge che il registro di **lettura** IF/ID.IR[rs]
 è uguale al registro di **scrittura** ID/EX.IR[rt] => **RAW**

LW determina il valore da scrivere in \$3 in fase MEM
 quindi serve **1 stallo + data forward**:

MEM/WB.LMD inviato a **TopAluInput**

ADD	R	codop	rs	rt	rd	sham	funct
LW	I	codop	rs	rt	address/const		

Sequenza

quando LW scrive su \$1, non c'è alcun conflitto con la precedente lettura su \$1

```
LW $3, 80 ($0)
ADD $2, $3, $1
LW $1, 800 ($2)
SUBI $1, $1, 3
ADDI $2, $2, 4
SW $1, 108 ($2)
SUB $4, $3, $1
```

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

si accorge che il registro di **lettura** IF/ID.IR[rs]
è uguale al registro di **scrittura** ID/EX.IR[rd] => **RAW** su \$2

risolvibile con data forwarding:

EX/MEM.AluOutput inviato a TopAluInput

Sequenza 2

```
LW $3, 80 ($0)
ADD $2, $3, $1
LW $1, 800 ($2)
SUBI $1, $1, 3
ADDI $2, $2, 4
SW $1, 108 ($2)
SUB $4, $3, $1
```

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

si accorge che il registro di **lettura** IF/ID.IR[rs]
è uguale al registro di **scrittura** ID/EX.IR[rt] => **RAW** su \$1

LW determina il valore da scrivere in \$1 in fase MEM
quindi serve **1 stallo + data forward**:

MEM/WB.LMD inviato a TopAluInput

SUBI/LW

codop	rs	rt	address/const
-------	----	----	---------------

Sequenza 2

```

LW $3, 80 ($0)
ADD $2, $3, $1
LW $1, 800($2)
SUBI $1, $1, 3
ADDI $2, $2, 4
SW $1, 108($2)
SUB $4, $3, $1

```

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

...	5	6	7	8	9	10	11
WB							
EX		MEM	WB				
ID		EX	MEM	WB			
IF		ID	ID	EX	MEM	WB	
		IF	IF	ID	EX	MEM	WB
				IF	ID		

tutto OK

si accorge di **RAW su \$2**, risolvibile con **forward** di EX/MEM.AluOutput verso TopAluInput

ma c'è un'altra **RAW su \$1**:

SW in fase ID deve anche **leggere** \$1=IF/ID.IR[rt] e **propagarlo** nei registri di pipeline **fino alla fase MEM** ma il valore corretto di \$1 sarà scritto in fase WB di SUBI

l'esecuzione di SW non può procedere
-> **stallo**

Sequenza 2

```

LW $3, 80 ($0)
ADD $2, $3, $1
LW $1, 800($2)
SUBI $1, $1, 3
ADDI $2, $2, 4
SW $1, 108($2)
SUB $4, $3, $1

```

...	5	6	7	8	9	10	11	12	13
WB									
EX		MEM	WB						
ID		EX	MEM	WB					
IF		ID	ID	EX	MEM	WB			
		IF	IF	ID	EX	MEM	WB		
				IF	ID	ID	EX	MEM	WB

lo stallo **ripete la fase ID**:

- nella prima metà del ciclo SUBI scrive \$1
- nella seconda metà del ciclo SW può leggere \$1

e risolve **RAW su \$2**, con **forward** di MEM/WB.AluOutput verso TopAluInput

Sequenza 2

```

LW $3, 80 ($0)
ADD $2, $3, $1
LW $1, 800($2)
SUBI $1, $1, 3
ADDI $2, $2, 4
SW $1, 108($2)
SUB $4, $3, $1
    
```

...	5	6	7	8	9	10	11	12	13	14
	WB									
	EX	MEM	WB							
	ID	EX	MEM	WB						
	IF	ID	ID	EX	MEM	WB				
		IF	IF	ID	EX	MEM	WB			
				IF	ID	ID	EX	MEM	WB	
					IF	IF	ID	EX	MEM	WB

tutto ok: sia SUB che SW fanno lettura di \$1

Sequenza 2: Riordino?

LW \$3, 80 (\$0)

ADD \$2, \$3, \$1

LW \$1, 800(\$2)

SUBI \$1, \$1, 3

ADDI \$2, \$2, 4

SW \$1, 108(\$2)

SUB \$4, \$3, \$1

legge il contenuto di \$3, che è definito nell'istruzione precedente, quindi non si può anticipare

legge il contenuto di \$2, definito nella ADD precedente, quindi non si può anticipare



leggere il contenuto di \$1, definito nella LW precedente, quindi non si può anticipare

si può anticipare prima di SUBI, ma non prima di LW perche' questa ADDI modifica \$2

legge il contenuto di \$2, definito nella ADDI precedente, quindi deve stare dopo ADDI e legge \$1, definito in SUBI, quindi anche dopo SUBI

basta che stia dopo la SUBI che definisce il contenuto di \$1, nota che SW legge \$1



Sequenza 2: Riordino?

LW \$3, 80 (\$0)		LW \$3, 80 (\$0)
ADD \$2, \$3, \$1		ADD \$2, \$3, \$1
LW \$1, 800 (\$2)		LW \$1, 800 (\$2)
SUBI \$1, \$1, 3		ADDI \$2, \$2, 4
ADDI \$2, \$2, 4		SUBI \$1, \$1, 3
SW \$1, 108 (\$2)		SW \$1, 108 (\$2)
SUB \$4, \$3, \$1		SUB \$4, \$3, \$1

basta che stia dopo la SUBI che definisce il contenuto di \$1, nota che SW legge \$1

deve leggere \$1 in fase ID, ma \$1 è scritto in fase WB di SUBI quindi **peggiora la situazione!**

Sequenza con branch

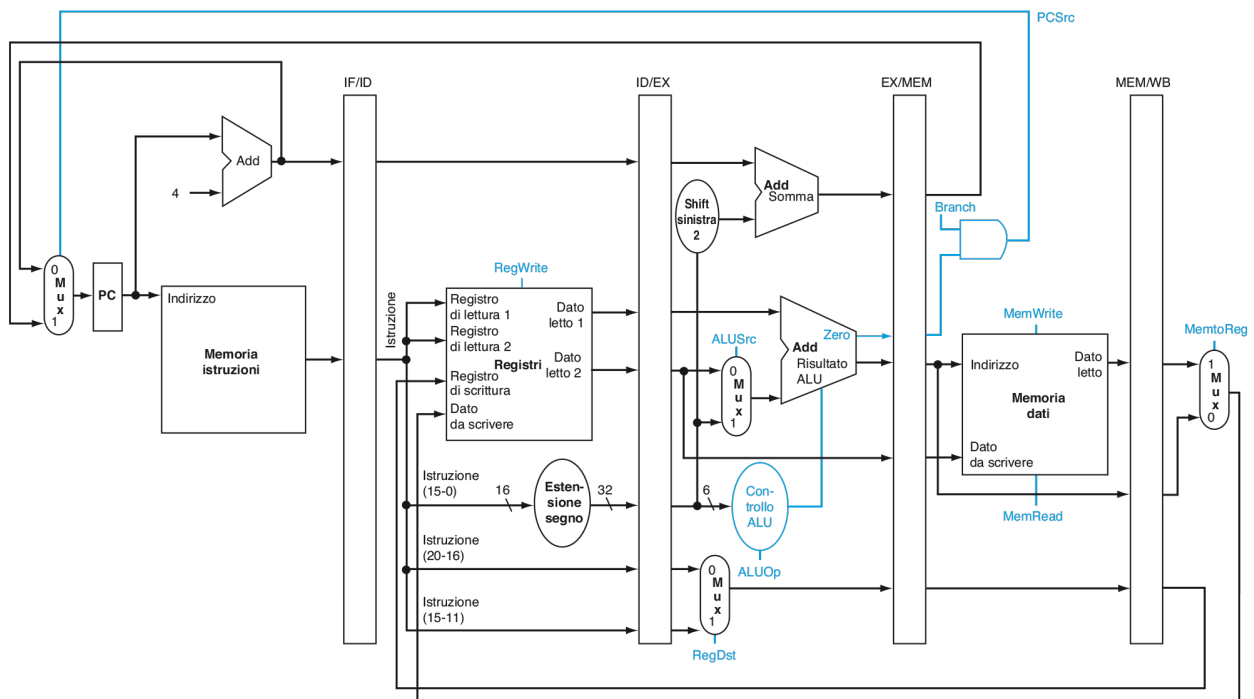
LW \$2, 0 (\$1)	assumiamo:
Label1: BEQ \$2, \$0, Label2	 preso solo la prima volta
LW \$3, 0 (\$2)	
BEQ \$3, \$0, Label1	 preso sempre
ADD \$1, \$3, \$1	
Label2: SW \$1, 0 (\$2)	

per semplicità

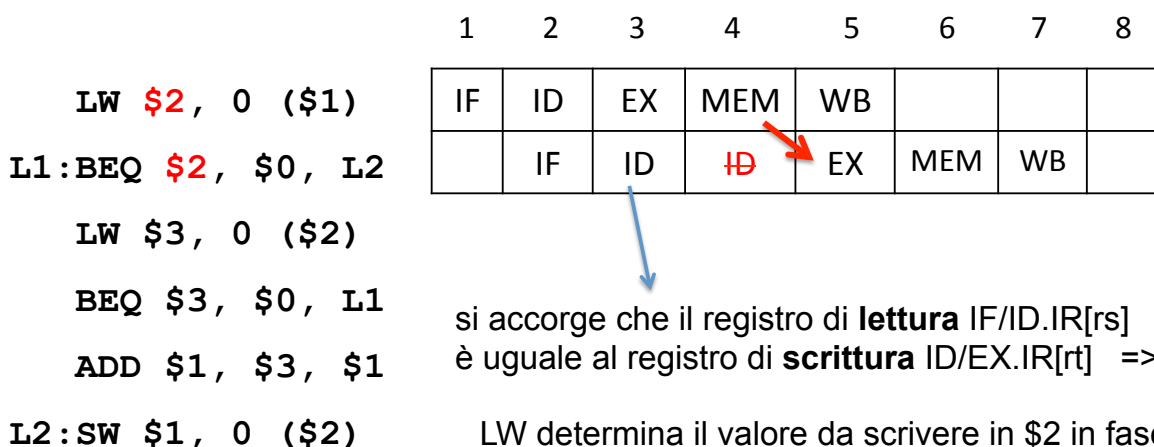
- usiamo Label al posto del campo Imm (esteso a 32 bit) da usare come offset per calcolare l'indirizzo del salto:
`beq $1, $2, Imm if ([R1]-[R2]==0) then PC=NPC+(Imm<<2)`
- non assumiamo alcuna tecnica di predizione dei salti
- in fase EX di istruzione beq calcola la condizione e il target, ma è in fase **MEM** che decide se saltare, cioè usa la condizione per decidere il valore di PC

Segnali di controllo

in fase MEM decide come aggiornare il PC a seconda di salto preso o no



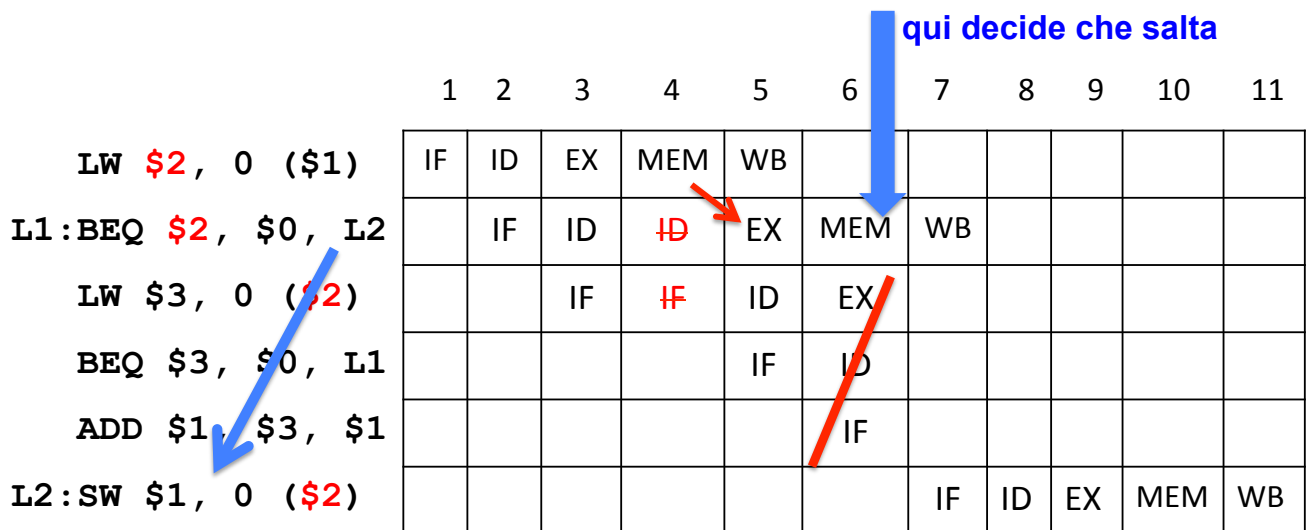
Sequenza con branch



BEQ/LW

codop	rs	rt	address/const
-------	----	----	---------------

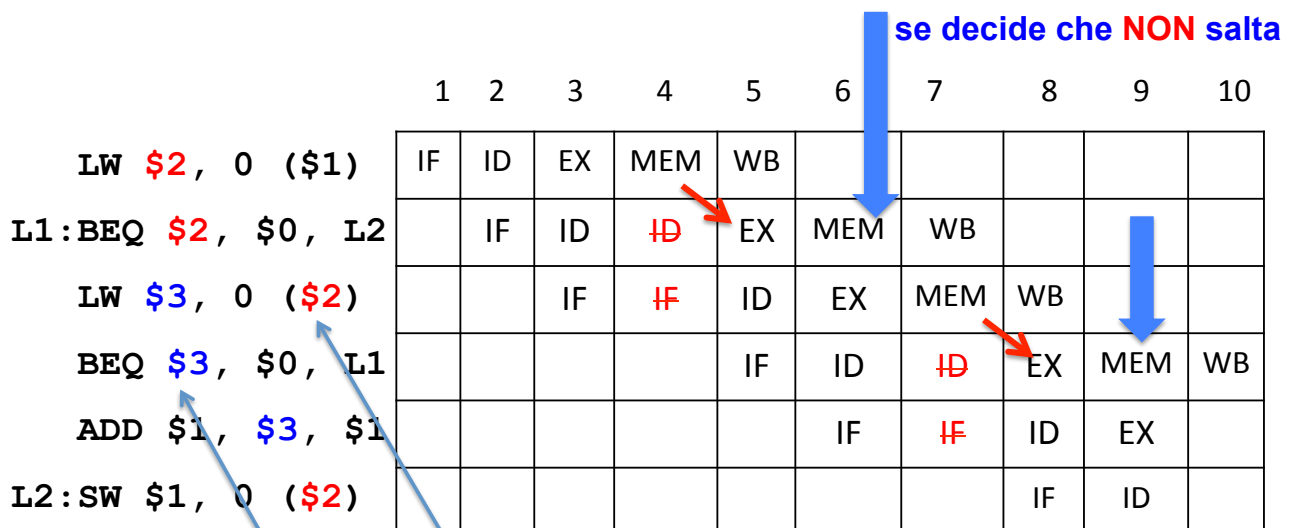
Sequenza con branch



scarta il contenuto della pipeline e
inizia IF dell'istruzione in L2
nessun problema di dipendenza

branch penalty = n. di cicli in cui non conclude nessuna istruzione = 3

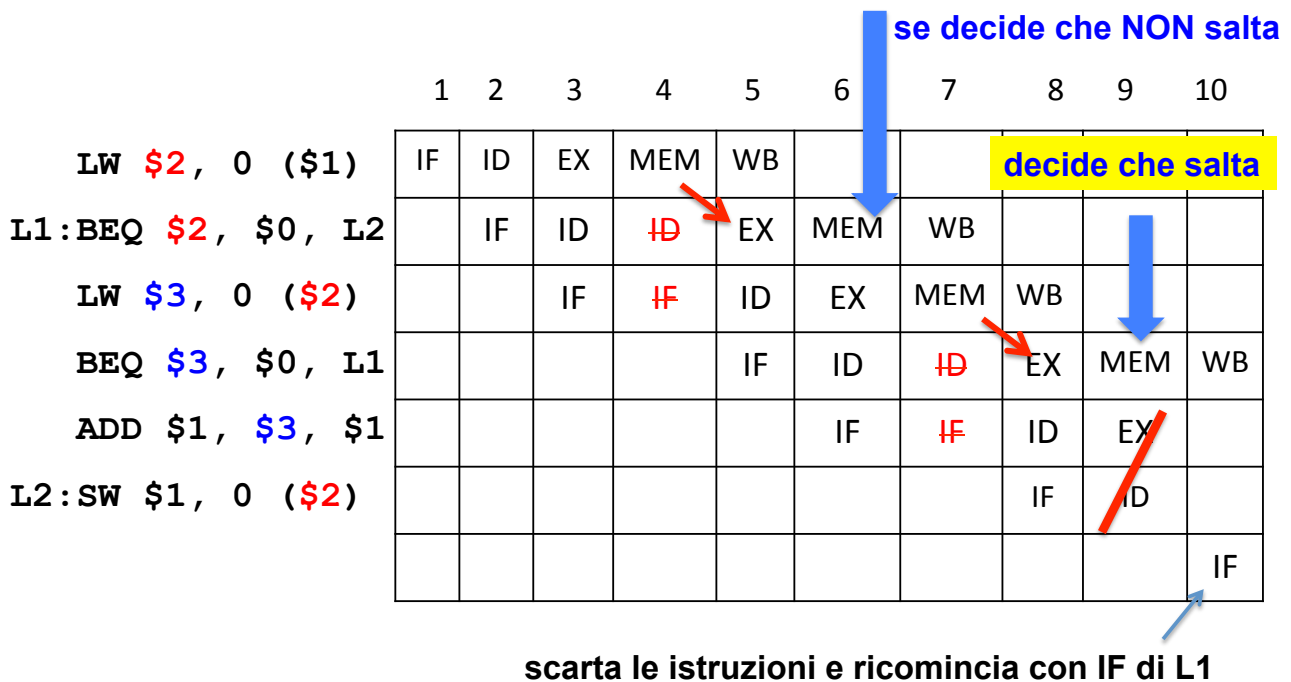
Sequenza con branch



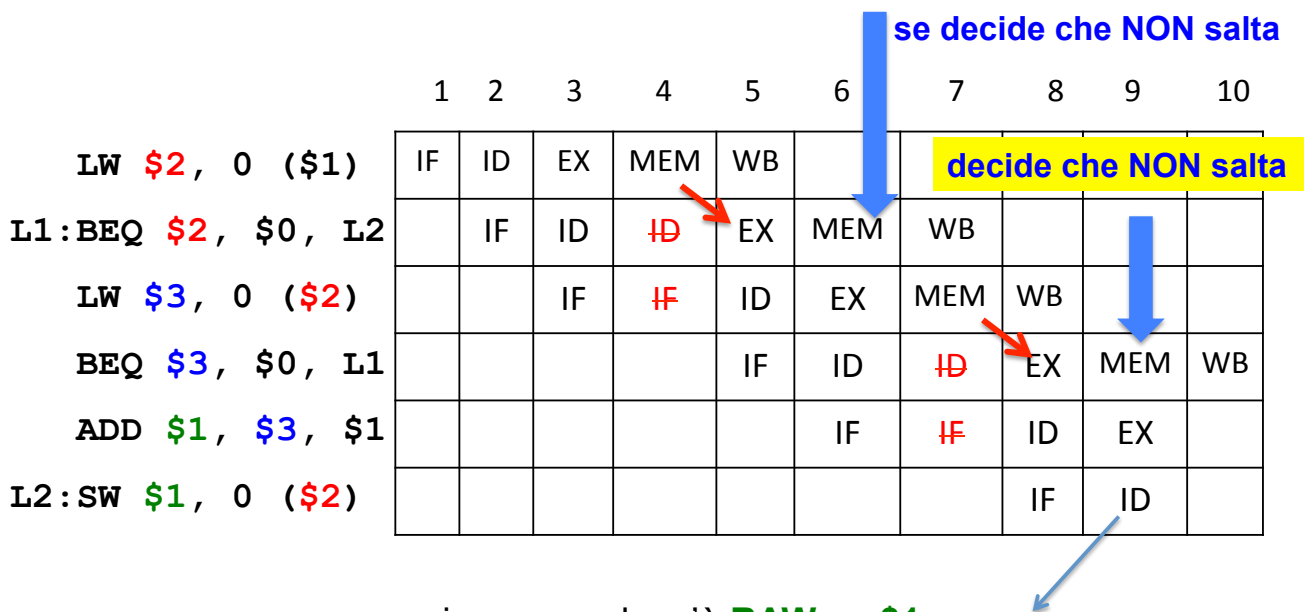
tutto ok: no RAW su \$2 poiche' LW legge \$2 in
seconda metà di ciclo 5, e in prima metà di ciclo 5
LW completa la scrittura

RAW su \$3: come prima stallo più forward

Sequenza con branch



Sequenza con branch



si accorge che c'è RAW su \$1:

SW in fase ID deve **leggere** \$1 e **propagarlo** nei registri di pipeline **fino alla fase MEM** ma il valore corretto di \$1 sarà scritto in fase WB di ADD
non c'è forward verso fase ID, quindi 2 stalli

Sequenza con branch



ripete la fase ID:

- nella prima metà del ciclo ADD scrive \$1
- nella seconda metà del ciclo SW può leggere \$1