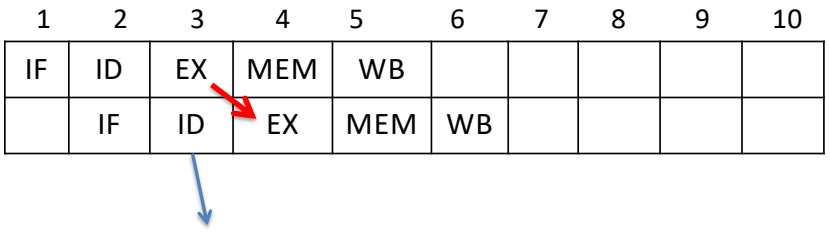
Sequenza 1

```
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
```



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

risovibile con data forwarding:
EX/MEM.AluOutput 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
SUB $2, $7, $5	IF	ID	EX	MEM	WB					
LW $1, 7 ($2)		IF	ID	EX	MEM	WB				
ADD $2, $1, $8			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**

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
SUB $2, $7, $5	IF	ID	EX	MEM	WB					
LW $1, 7 ($2)		IF	ID	EX	MEM	WB				
ADD $2, $1, $8			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
SUB	WB							
LW	MEM	WB						
ADD	~~ID~~	EX	MEM	WB				
SW	~~IF~~	ID	EX	MEM	WB			
SUBI		IF	ID	EX	MEM	WB		
ADDI			IF	ID	EX	MEM	WB	
ADD				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

```
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
SUB	WB							
LW	MEM	WB						
ADD	~~ID~~	EX	MEM	WB				
SW	~~IF~~	ID	EX	MEM	WB			
SUBI		IF	ID	EX	MEM	WB		
ADDI			IF	ID	EX	MEM	WB	
ADD				IF	ID	EX	MEM	WB

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]

risolti con forward:

EX/MEM.AluOutput inviato a TopAluInput

MEM/WB.AluOutput va a BottomAluInput

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
```

deve leggere il contenuto di $2, che è definito nell'istruzione precedente, quindi non si può anticipare

se si anticipa ADD prima di LW, si modifica il contenuto di $2, quindi LW carica indirizzo diverso e il programma cambia semantica

SW si può scambiare con ADD, ma non elimina la necessità dello stallo perché anche SW legge $1

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
SW $3, 73 ($1)
ADD $1, $7, $2
```

legge $1 corretto

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
```

toglie il forward doppio

```
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 lo stallo

si possono fare entrambi
altri riordini sono possibili

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
```

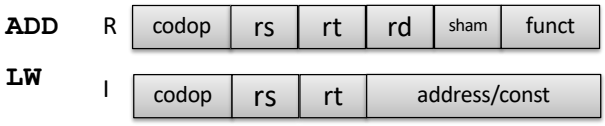
Sequenza 2

```
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



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		

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

risovibile 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

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	
WB							
EX	MEM	WB					
ID	EX	MEM	WB				
IF	ID	~~ID~~	EX	MEM	WB		
	IF	~~IF~~	ID	EX	MEM	WB	
			IF	ID			

tutto OK

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

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

Pipeline **SENZA** data forward

	1	2	3	4	5	6	7	8	9	10	11
`lw $3 80($0)`	IF	ID	EX	MEM	WB						
`add $2 $3 $1`		IF	ID	ID	ID	EX	MEM	WB			
`lw $1 38($2)`			IF	IF	IF	ID	ID	ID	EX	MEM	WB
`subi $1 $1 3`						IF	IF	IF	ID		
`addi $2 $2 4`											
`sw $1 11($2)`											
`sub $4 $3 $1`											

Pipeline **SENZA** data forward

	5	6	7	8	9	10	11	12	13	14	15
`lw $3 80($0)`	WB										
`add $2 $3 $1`	ID	EX	MEM	WB							
`lw $1 38($2)`	IF	ID	ID	ID	EX	MEM	WB				
`subi $1 $1 3`		IF	IF	IF	ID	ID	ID	EX	MEM	WB	
`addi $2 $2 4`					IF	IF	IF	ID	EX	MEM	WB
`sw $1 11($2)`											
`sub $4 $3 $1`											

Pipeline **SENZA** data forward

	9	10	11	12	13	14	15	16	17	18	19
`lw $3 80($0)`											
`add $2 $3 $1`											
`lw $1 38($2)`	EX	MEM	WB								
`subi $1 $1 3`	ID	ID	ID	EX	MEM	WB					
`addi $2 $2 4`	IF	IF	IF	ID	EX	MEM	WB				
`sw $1 11($2)`				IF	ID	ID	ID	EX	MEM	WB	
`sub $4 $3 $1`					IF	IF	IF	ID	EX	MEM	WB

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 perché 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 sche stia dopo la SUBI che definisce il contenuto di $1, nota che SW legge $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
```

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

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

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

Sequenza 3

```
lw   $1  0 ($2)
add  $2  $3  $1
sw   $2  21 ($1)
beq  $2  $1  11
add  $3  $2  $2
add  $1  $1  $3
sw   $3  0 ($1)
```

Pipeline CON data forward

	1	2	3	4	5	6	7	8	9	10	11
lw $1 0 ($2)	IF	ID	EX	MEM	WB						
add $2 $3 $1		IF	ID	ID	EX	MEM	WB				
sw $2 21 ($1)											
beq $2 $1 11											
add $3 $2 $2											
add $1 $1 $3											
sw $3 0 ($1)											

istruzione 2

IF/ID.IR[rt] = ID/EX.IR[rt] ➔ RAW $1

1 stallo e forward:
MEM/WB.**LMD** → BottomALUInput

Pipeline CON data forward

	1	2	3	4	5	6	7	8	9	10	11
lw $1 0 ($2)	IF	ID	EX	MEM	WB						
add $2 $3 $1		IF	ID	ID	EX	MEM	WB				
sw $2 21 ($1)			IF	IF	ID	ID	ID	EX	MEM	WB	
beq $2 $1 11											
add $3 $2 $2											
add $1 $1 $3											
sw $3 0 ($1)											

istruzione 3

IF/ID.IR[rs] = MEM/WB.IR[rt] ➔ RAW $1

IF/ID.IR[rt] = ID/EX.IR[rd] → RAW $2

deve leggere $2 in fase ID
quindi **2 stalli** e lettura in seconda metà di ciclo
così si risolve anche RAW $1

Pipeline CON data forward

	1	2	3	4	5	6	7	8	9	10	11
lw $1 0 ($2)	IF	ID	EX	MEM	WB						
add $2 $3 $1		IF	ID	ID	EX	MEM	WB				
sw $2 21 ($1)			IF	IF	ID	ID	ID	EX	MEM	WB	
beq $2 $1 11					IF	IF	IF	ID	EX	MEM	WB
add $3 $2 $2											
add $1 $1 $3											
sw $3 0 ($1)											

istruzione 4

beq legge sia $2 che $1, che erano scritte da istruzioni 2 e 1, ma al ciclo 8 sono già completate

in ciclo 10 usa la condizione di salto.
Supponiamo sia falsa

Pipeline CON data forward

	1	2	3	4	5	6	7	8	9	10	11
lw $1 0 ($2)	IF	ID	EX	MEM	WB						
add $2 $3 $1		IF	ID	ID	EX	MEM	WB				
sw $2 21 ($1)			IF	IF	ID	ID	ID	EX	MEM	WB	
beq $2 $1 11					IF	IF	IF	ID	EX	MEM	WB
add $3 $2 $2								IF	ID	EX	..
add $1 $1 $3											
sw $3 0 ($1)											

istruzione 5

tutto ok

Pipeline CON data forward

	5	6	7	8	9	10	11	12	13	14
lw $1 0 ($2)	WB									
add $2 $3 $1	EX	MEM	WB							
sw $2 21 ($1)	ID	ID	ID	EX	MEM	WB				
beq $2 $1 11	IF	IF	IF	ID	EX	MEM	WB			
add $3 $2 $2				IF	ID	EX	MEM	WB		
add $1 $1 $3					IF	ID	EX	MEM	WB	
sw $3 0 ($1)										

istruzione 6

IF/ID.IR[rt] = ID/EX.IR[rd] → RAW $3

si risolve con forward:
EX/MEM.**ALUOutput** → BottomALUInput

Pipeline CON data forward

	5	6	7	8	9	10	11	12	13	14	15
lw $1 0 ($2)	WB										
add $2 $3 $1	EX	MEM	WB								
sw $2 21 ($1)	ID	ID	ID	EX	MEM	WB					
beq $2 $1 11	IF	IF	IF	ID	EX	MEM	WB				
add $3 $2 $2				IF	ID	EX	MEM	WB			
add $1 $1 $3					IF	ID	EX	MEM	WB		
sw $3 0 ($1)						IF	ID	ID	EX	MEM	WB

istruzione 7

IF/ID.IR[rs] = ID/EX.IR[rd] → RAW $1

IF/ID.IR[rt] = EX/MEM.IR[rd] → RAW $3

deve leggere $3 in fase ID
quindi 1 stalli e r/w in ciclo 12
più forward:
MEM/WB.AluOutput →TopAluInput

Sequenza con branch

```
        LW   $2, 0 ($1)
Label1: BEQ  $2, $0, Label2
        LW   $3, 0 ($2)
        BEQ  $3, $0, Label1
        ADD  $1, $3, $1
Label2: SW   $1, 0 ($2)
```

assumiamo:

← preso solo la prima volta

← preso sempre

per semplicità

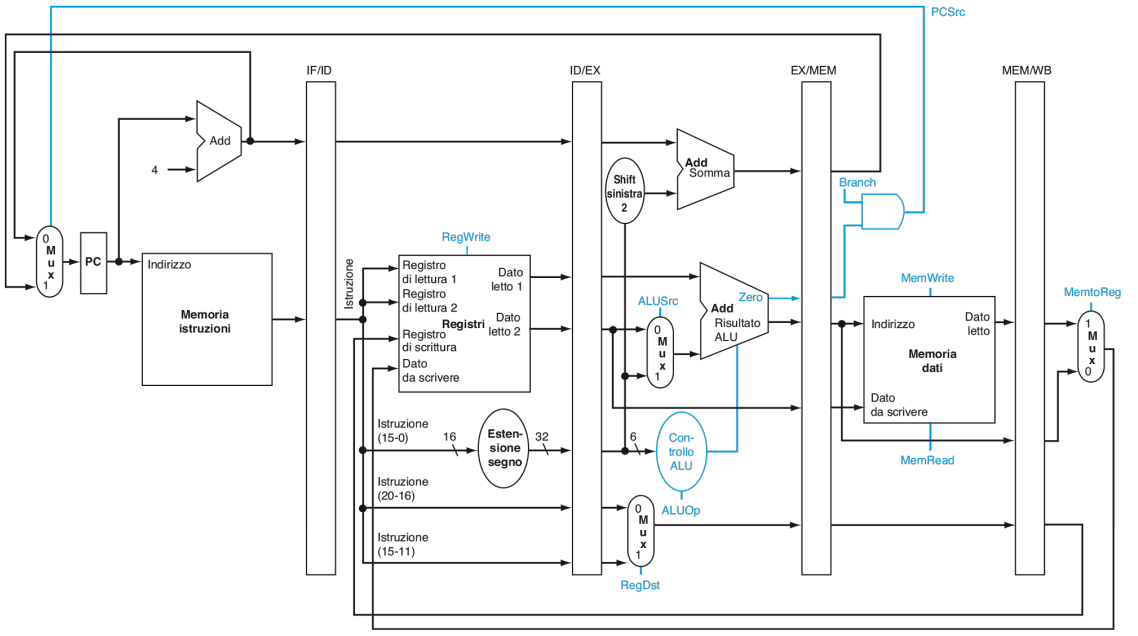
- usiamo Label al posto del campo Imm (esteso a 32 bit) da usare come offset per calcolare l'indirizzo del salto:

  `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

```
    LW   $2, 0 ($1)
L1: BEQ  $2, $0, L2
    LW   $3, 0 ($2)
    BEQ  $3, $0, L1
    ADD  $1, $3, $1
L2: SW   $1, 0 ($2)
```

	1	2	3	4	5	6	7	8
LW $2, 0 ($1)	IF	ID	EX	MEM	WB			
L1: BEQ $2, $0, L2		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 $2 in fase MEM
quindi serve **1 stallo + data forward**:
MEM/WB.LMD inviato a TopAluInput

BEQ/LW

Sequenza con branch

qui decide che salta

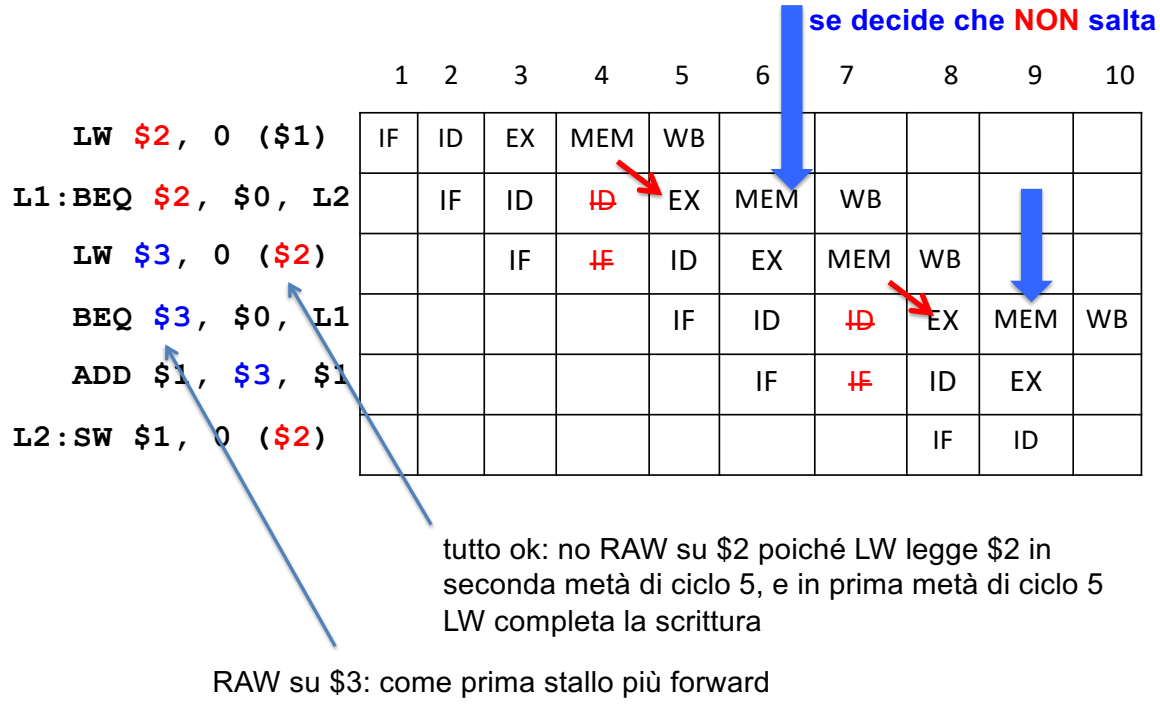
```
    LW  $2, 0 ($1)
L1: BEQ $2, $0, L2
    LW  $3, 0 ($2)
    BEQ $3, $0, L1
    ADD $1, $3, $1
L2: SW  $1, 0 ($2)
```

	1	2	3	4	5	6	7	8	9	10	11
LW $2, 0 ($1)	IF	ID	EX	MEM	WB						
L1: BEQ $2, $0, L2		IF	ID	~~ID~~	EX	MEM	WB				
LW $3, 0 ($2)			IF	~~IF~~	ID	EX					
BEQ $3, $0, L1					IF	ID					
ADD $1, $3, $1						IF					
L2: SW $1, 0 ($2)							IF	ID	EX	MEM	WB

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

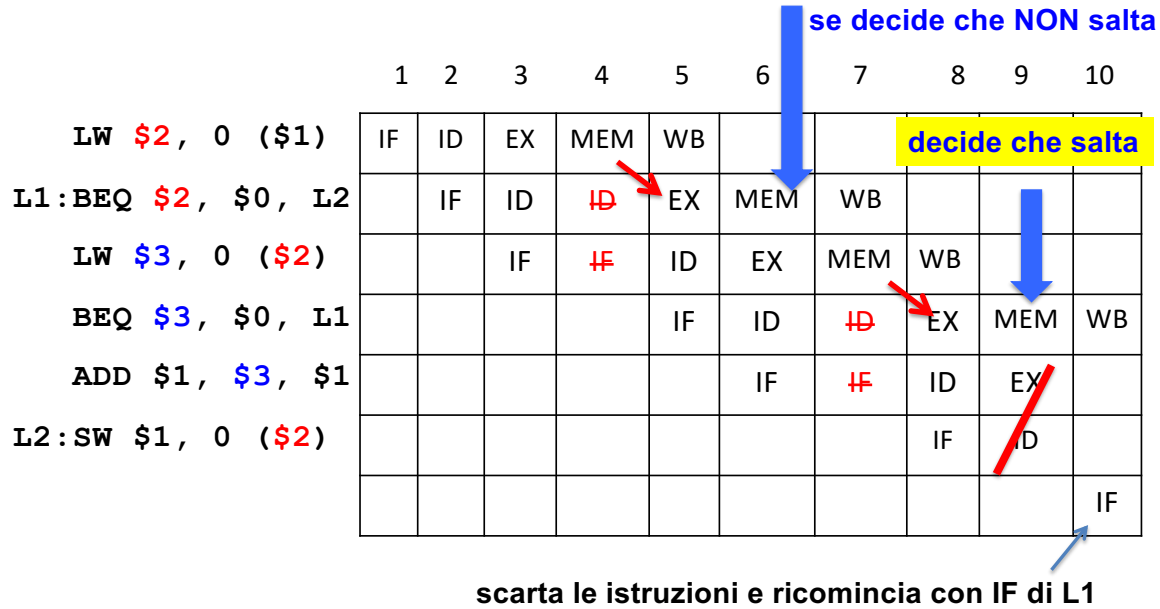
Sequenza con branch



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

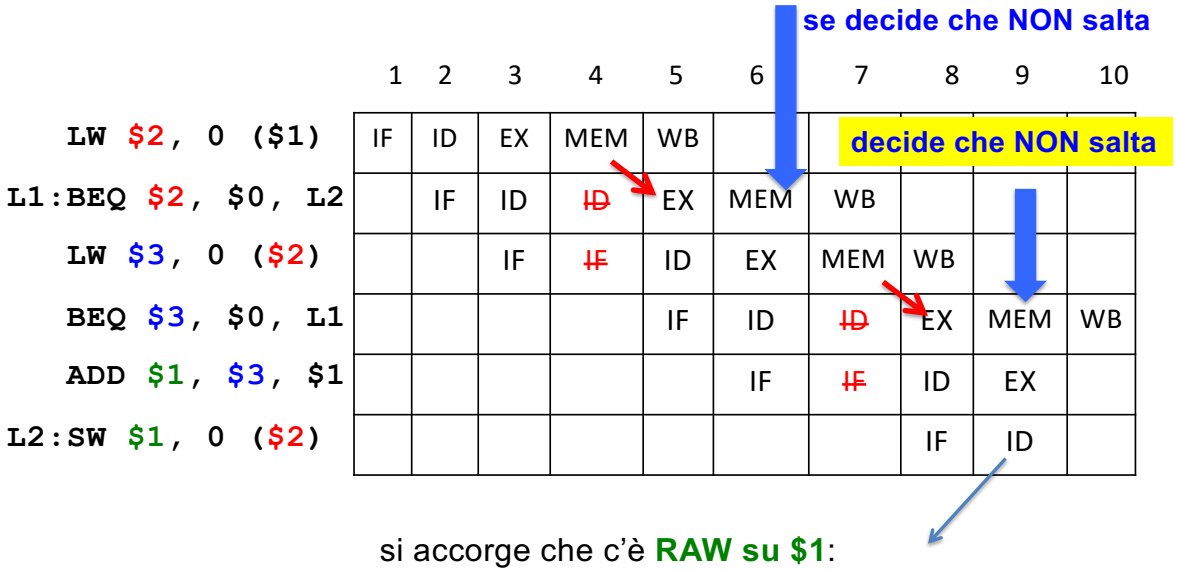
RAW su $3: come prima stallo più forward

Sequenza con branch



scarta le istruzioni e ricomincia con IF di L1

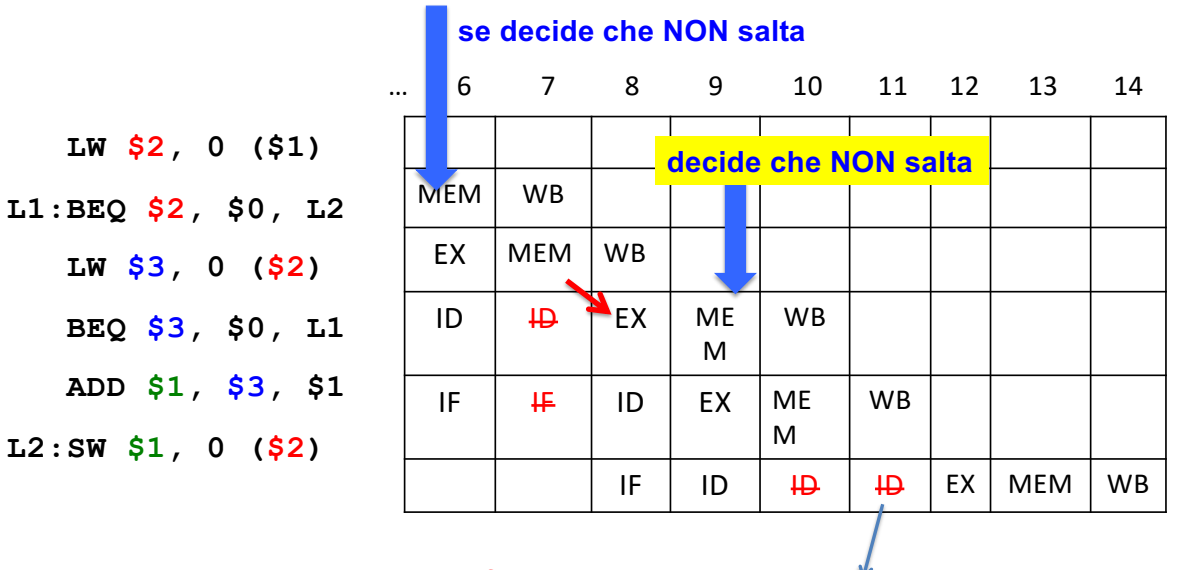
Sequenza con branch



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