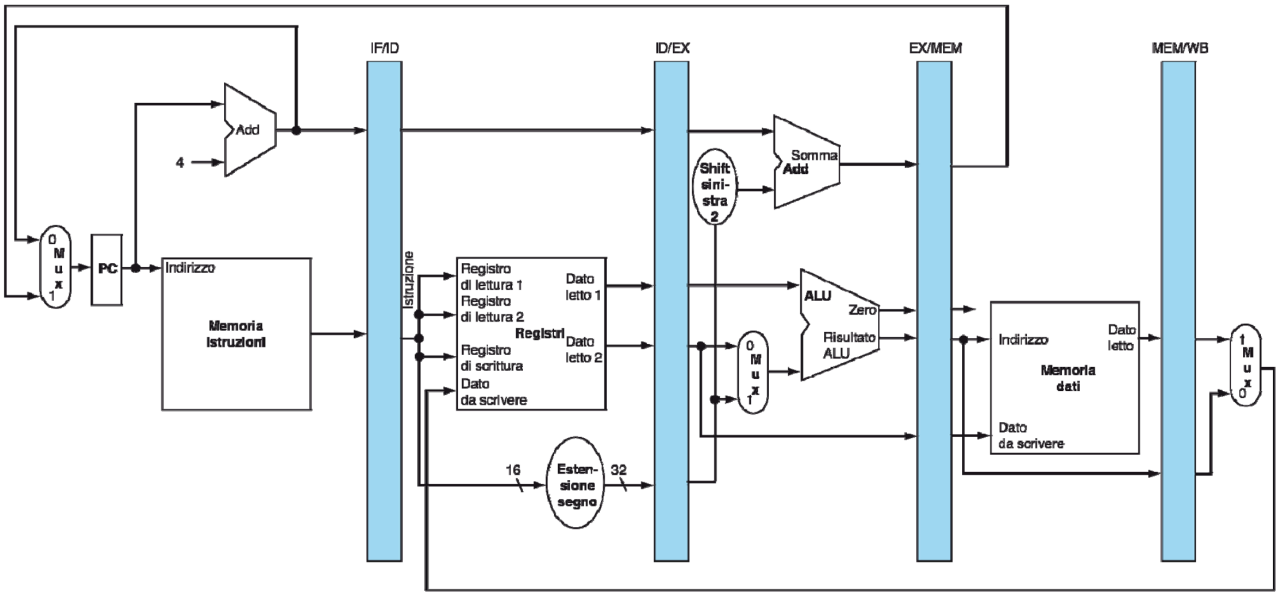
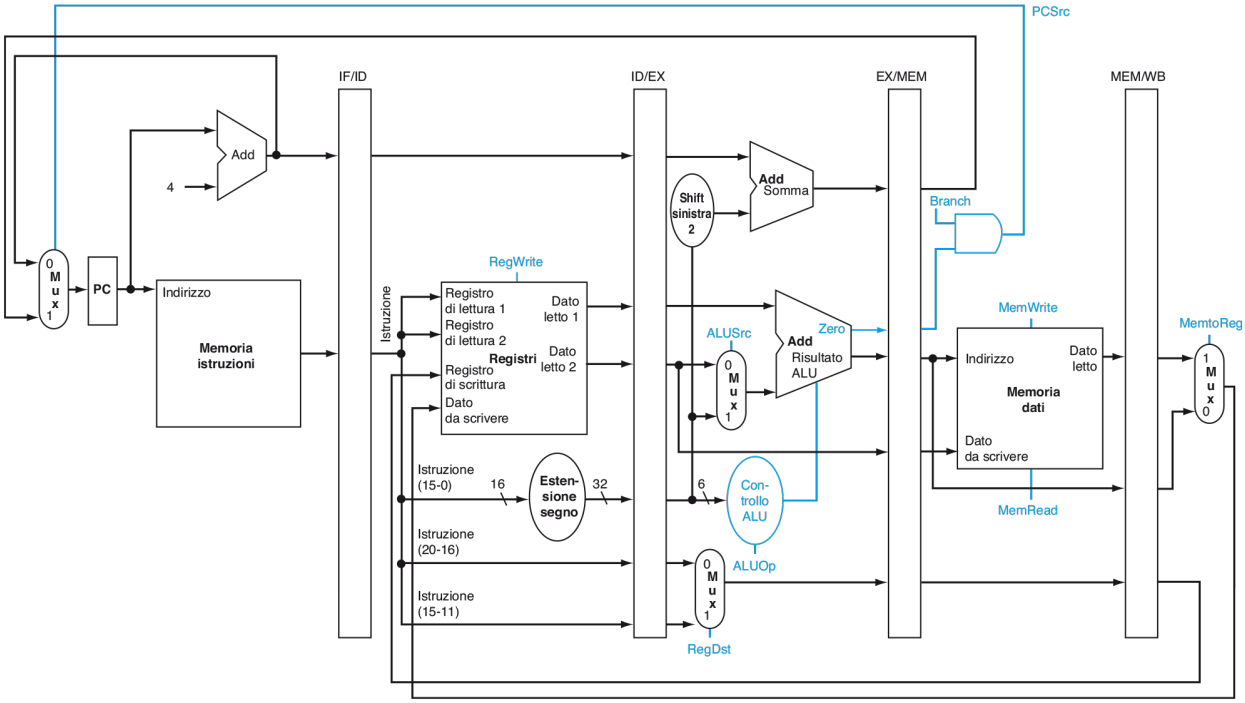
IF ID EX MEM WB

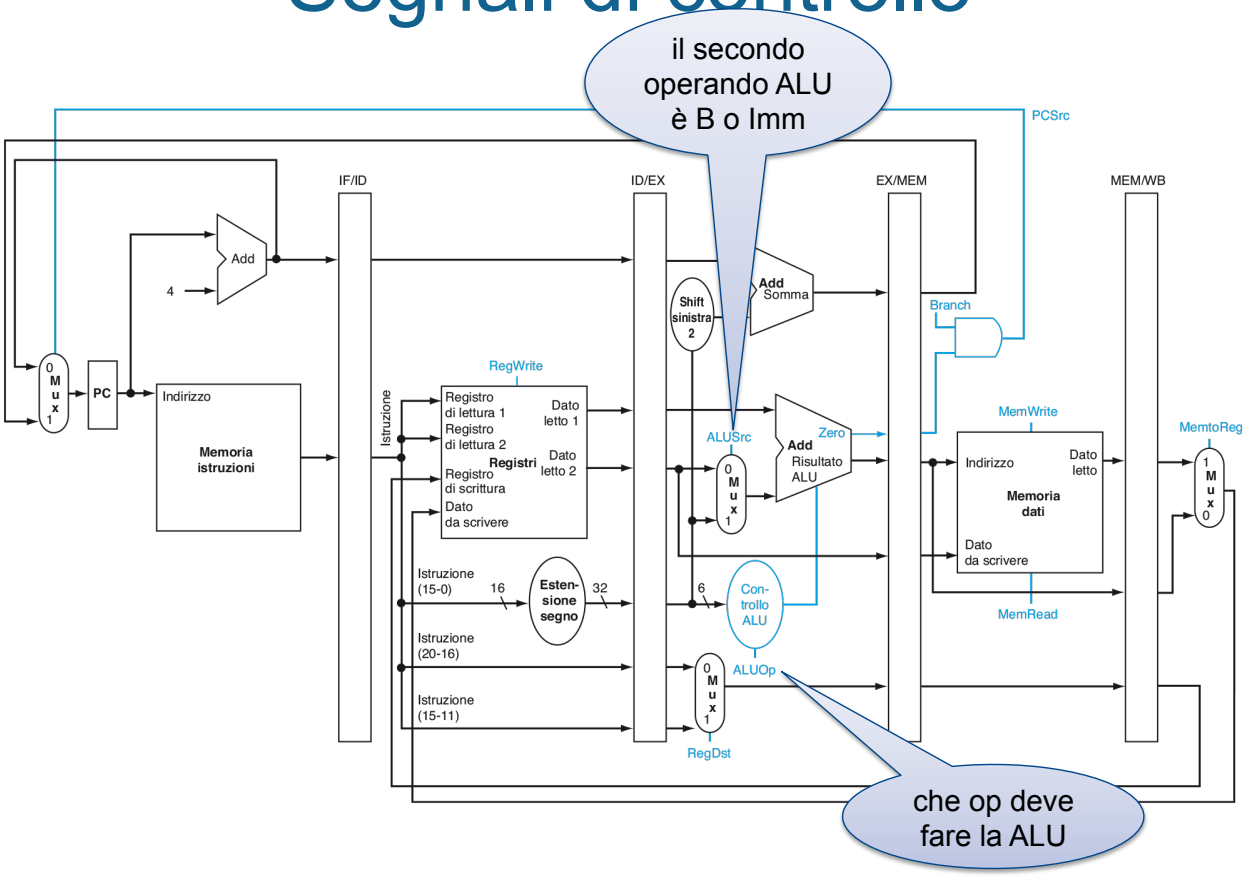


- **`Regs[MEM/WB.IR[rd]]`** (opp **`IR[rt]`**) ← **`MEM/WB.ALUOutput`**; per istr arit-log
- **`Regs[MEM/WB.IR[rt]`** ← **`MEM/WB.LMD`**; per load

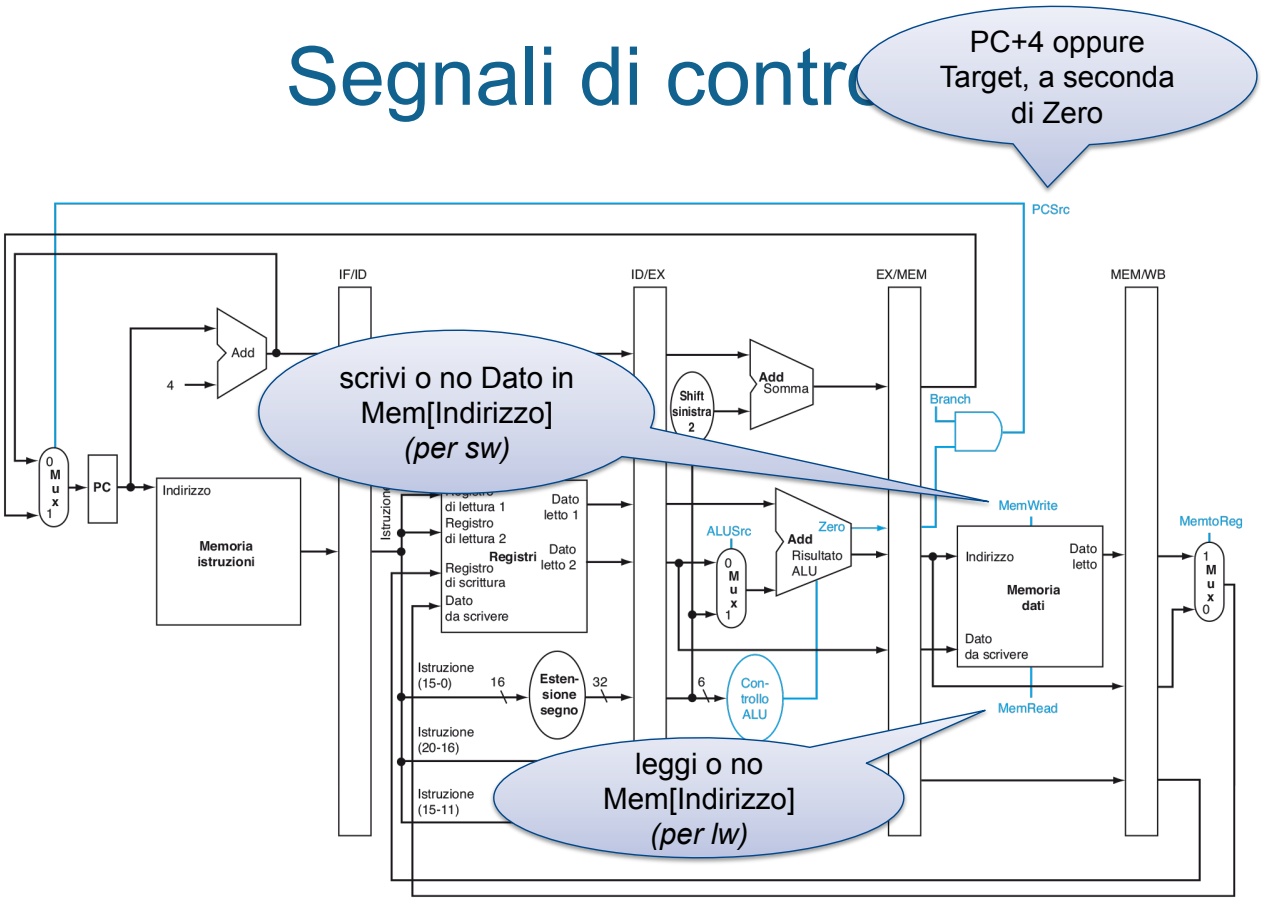
Segnali di controllo



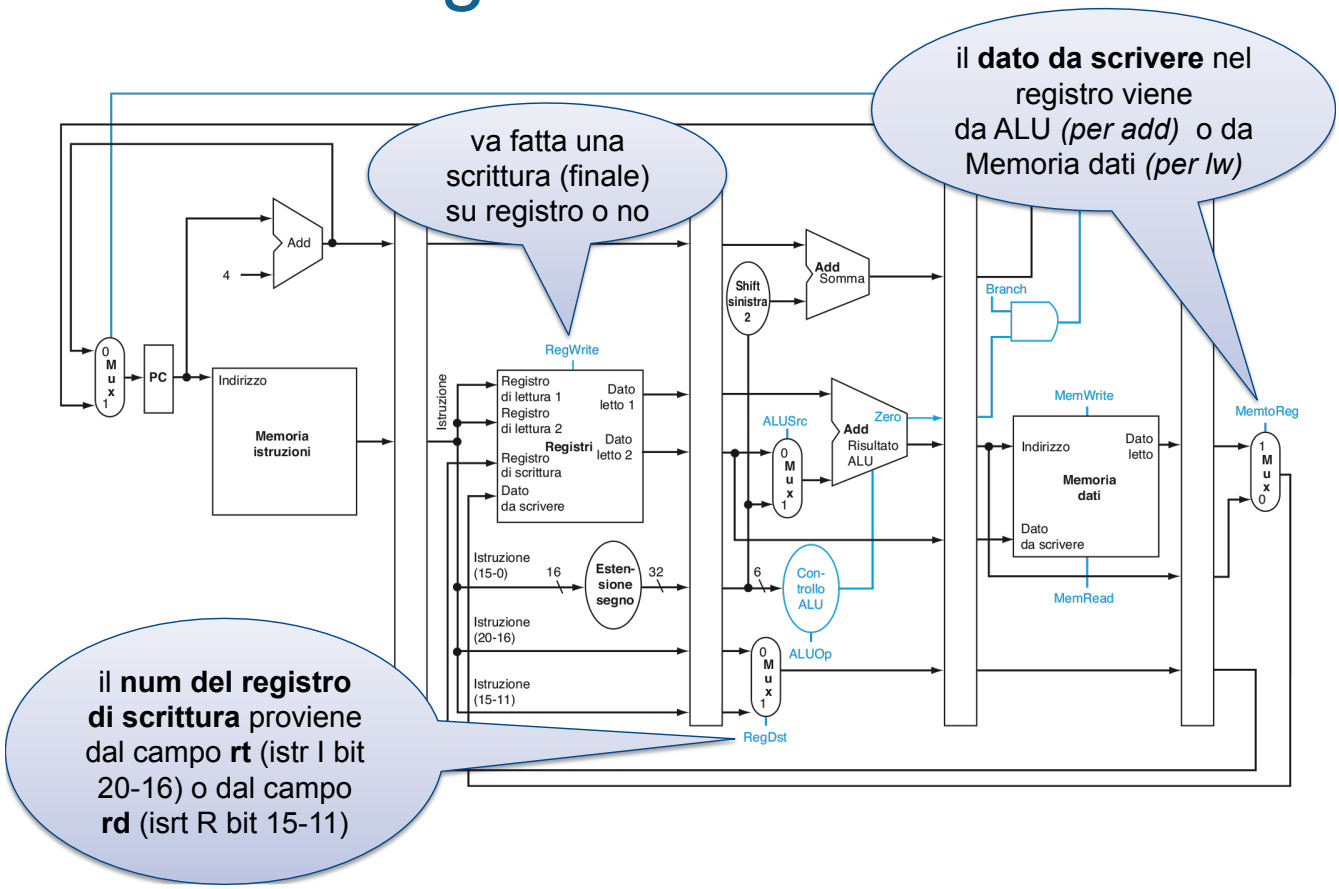
Segnali di controllo



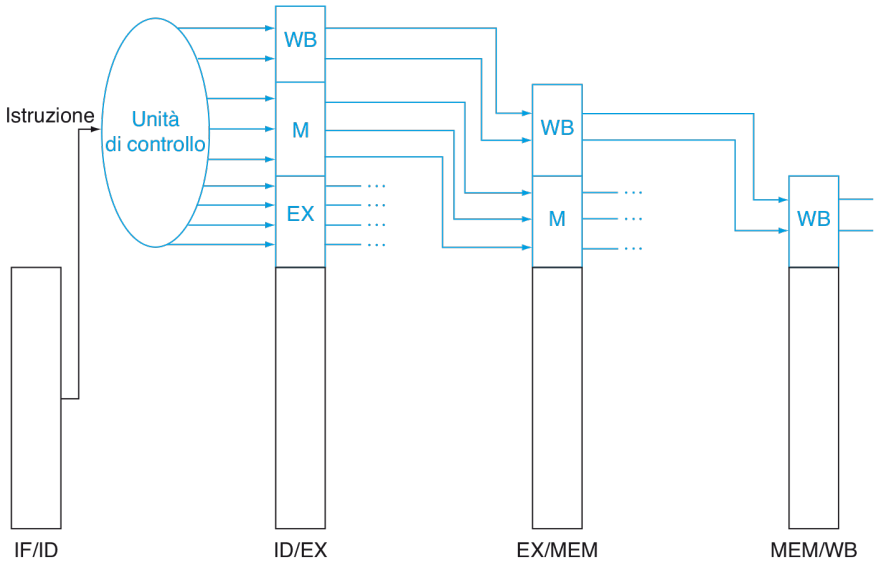
Segnali di contro



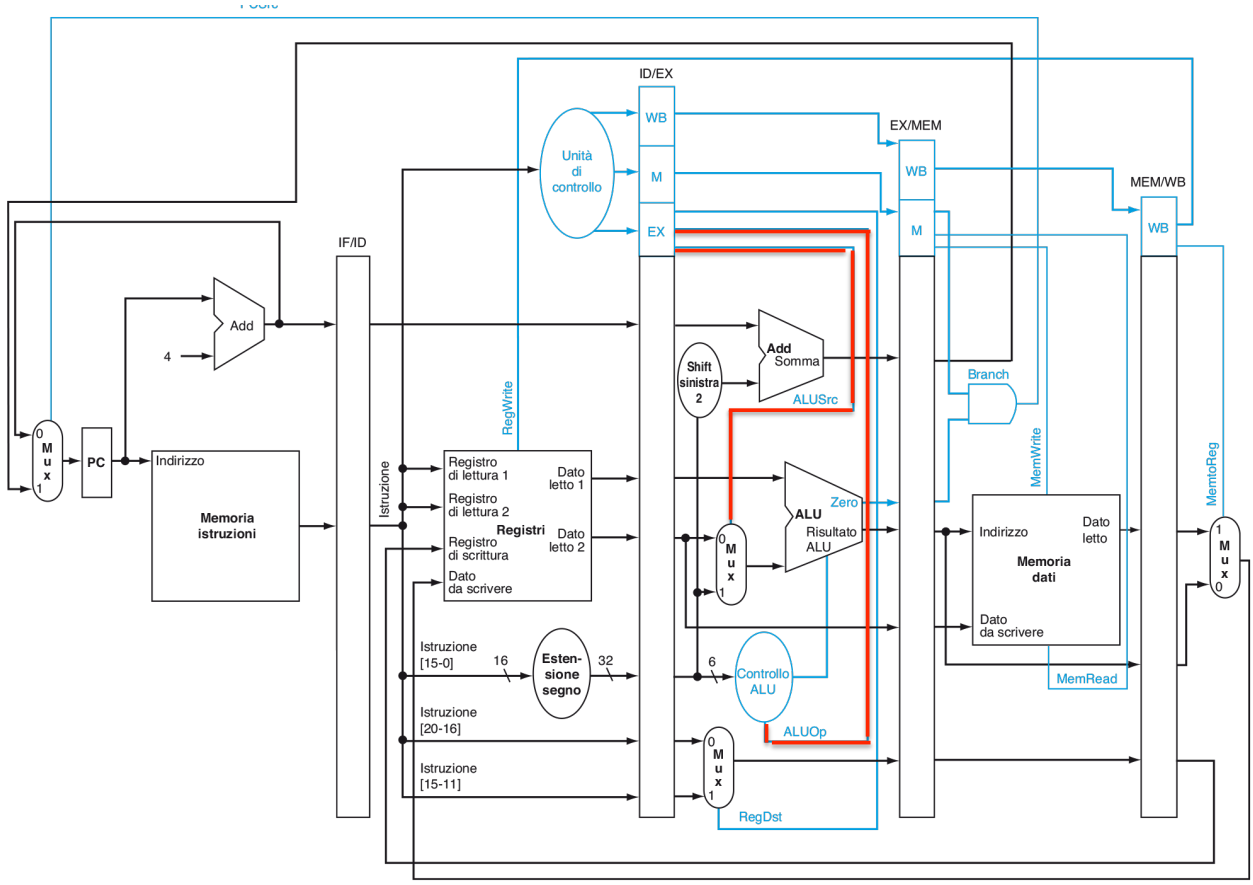
Segnali di controllo



Unità di controllo



- le fasi IF e ID non dipendono dai valori dei segnali di controllo
- in fase ID si possono calcolare i segnali corretti per le fasi successive
  - i segnali sono **calcolati in ID e propagati** attraverso i registri di pipeline



Pipeline e dipendenze dai dati

- Nella Pipeline MIPS **è possibile individuare tutte le dipendenze dai dati nella fase ID**
  - Se si rileva una dipendenza dai dati per una istruzione, questa **va in stallo prima di essere rilasciata** (issued, cioè quando una passa dalla fase ID a quella EX)
  - Inoltre, sempre nella fase ID, è possibile determinare che tipo di **data forwarding** adottare per evitare lo stallo ed anche predisporre gli opportuni segnali di controllo

- Esempio: realizziamo un forwarding nella fase EX per una dipendenza di tipo RAW (Read After Write) con sorgente che proviene da una istruzione **load** (*load interlock*)

dipendenze RAW da istr Load

Possibili casi

Situazione	Esempio di codice	Azione

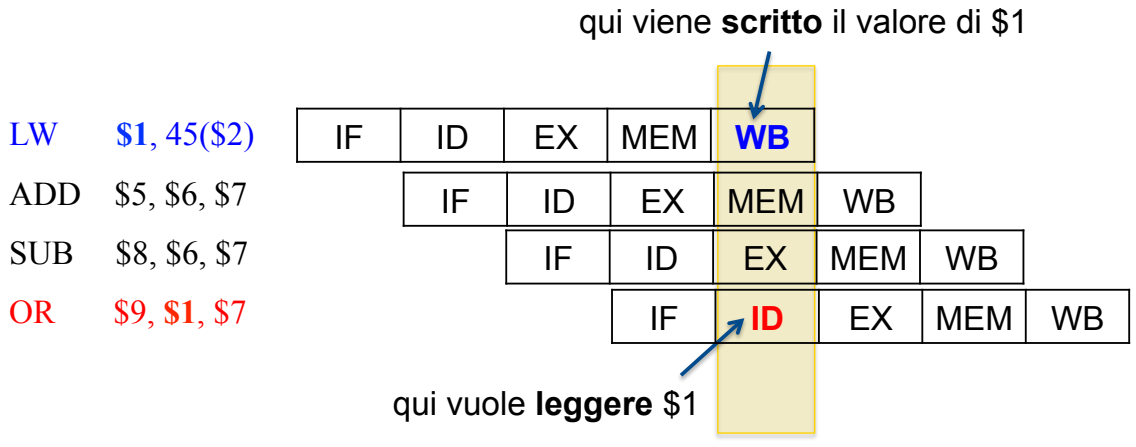
Nessuna dipendenza	LW **$1**, 45($2)
	ADD $5, $6, $7
	SUB $8, $6, $7
	OR $9, $6, $7
		Non occorre fare nulla

	LW **$1**, 45($2)
	ADD $5, **$1**, $7
	SUB $8, $6, $7
	OR $9, $6, $7	

	LW **$1**, 45($2)
	ADD $5, $6, $7
	SUB $8, **$1**, $7
	OR $9, $6, $7	

	LW **$1**, 45($2)
	ADD $5, $6, $7
	SUB $8, $6, $7
	OR $9, **$1**, $7	

RAW



la scrittura avviene nella **prima metà** del ciclo di clock,
la lettura nella **seconda metà**

nessun problema

dipendenze RAW da istr Load

Possibili casi

Situazione	Esempio di codice	Azione

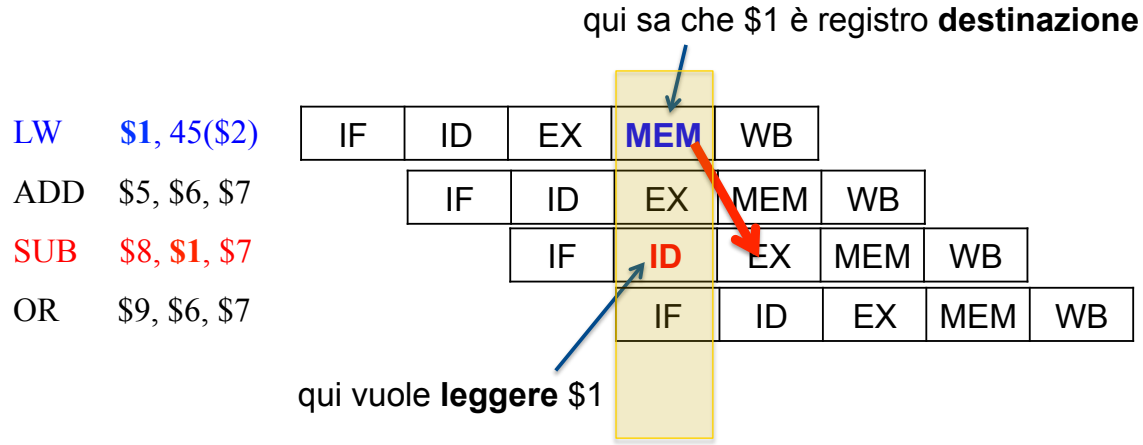
Nessuna dipendenza	LW **$1**, 45($2)
	ADD $5, $6, $7
	SUB $8, $6, $7
	OR $9, $6, $7
		Non occorre fare nulla

	LW **$1**, 45($2)
	ADD $5, **$1**, $7
	SUB $8, $6, $7
	OR $9, $6, $7	

	LW **$1**, 45($2)
	ADD $5, $6, $7
	SUB $8, **$1**, $7
	OR $9, $6, $7	

Dipendenza con accessi in ordine	LW **$1**, 45($2)
	ADD $5, $6, $7
	SUB $8, $6, $7
	OR $9, **$1**, $7
		Non occorre fare nulla perché la lettura di $1 in OR avviene dopo la scrittura del dato caricato

RAW

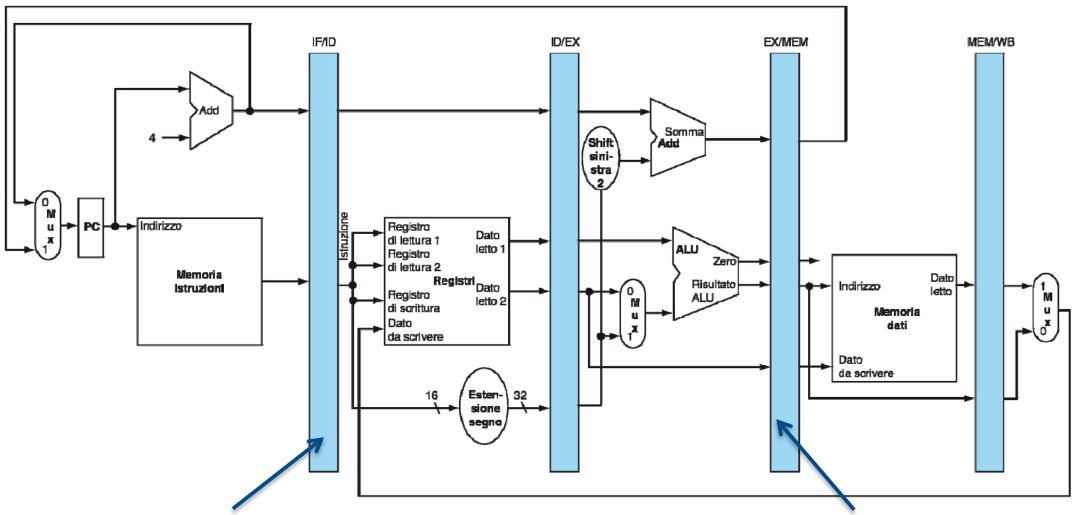


in questo ciclo si **identifica la dipendenza** RAW

il data forwarding
(il contenuto della memoria è inviato alla ALU prima della scrittura nel registro $1)
evita lo stallo

IF ID EX MEM WB

OR $9, $6, $7 SUB $8, $1, $7 ADD $5, $6, $7 LW $1, 45($2)



IF/ID.IR[opcode] =Reg-Reg ALU
IF/ID.IR[rs] = $1

EX/MEM.IR[opcode] = LW
EX/MEM.IR[rt] =$1

sono uguali!

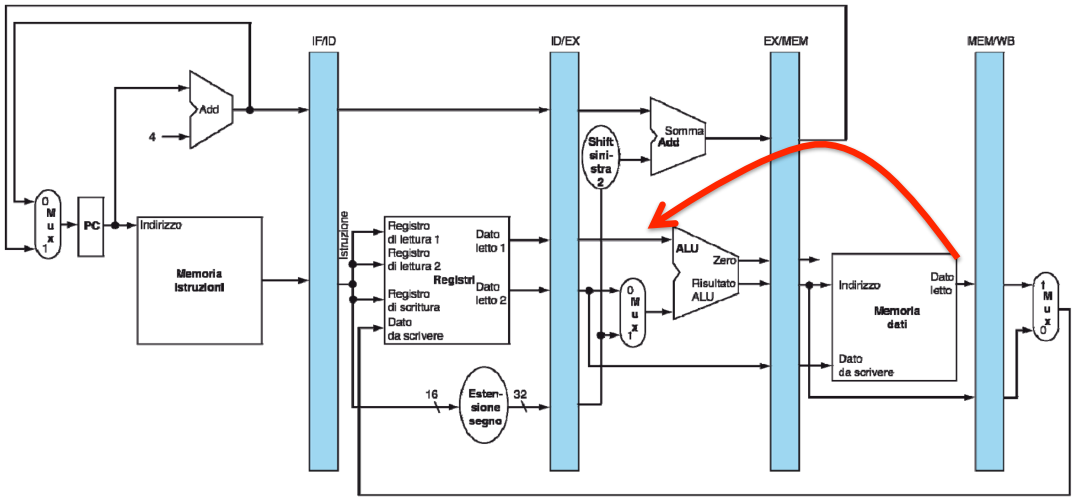
riconosciuta dipendenza RAW
risolvibile con forward

Read after ***Write***

ciclo successivo

IF ID EX MEM WB

OR $9, $6, $7 SUB $8, $1, $7 ADD $5, $6, $7 LW $1, 45($2)

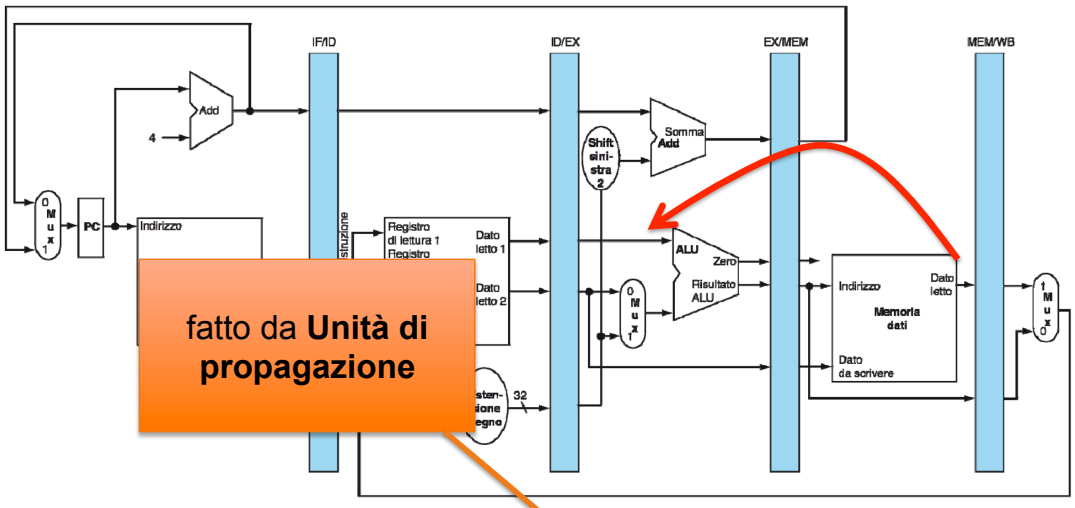


Effettua il forward da Load a Reg-Reg-ALU

- se MEM/WB.IR[rt] == ID/EX.IR[rs] allora **manda MEM/WB.LMD a Top ALU Input**

ciclo successivo

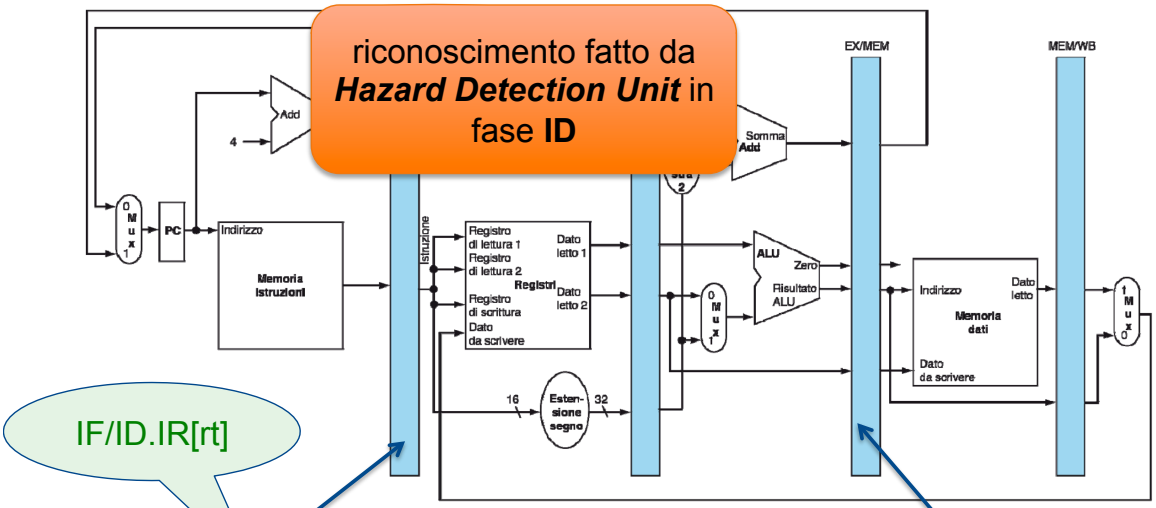
IF	ID	EX	MEM	WB
	OR $9, $6, $7	SUB $8, $1, $7	ADD $5, $6, $7	LW $1, 45($2)



Effettua il forward da Load a Reg-Reg-ALU

- se MEM/WB.IR[rt] == ID/EX.IR[rs] allora **manda MEM/WB.LMD a Top ALU Input**
- se MEM/WB.IR[rt] == ID/EX.IR[**rt**] allora **manda MEM/WB.LMD a Bottom ALU Input**
- istruzioni diverse hanno condizioni diverse

IF	ID	EX	MEM	WB
OR $9, $6, $7	SUB $8, $1, $7	ADD $5, $6, $7	LW $1, 45($2)	



IF/ID.IR[rt]

IF/ID.IR[opcode] =Reg-Reg ALU
IF/ID.IR[rs] = $1

EX/MEM.IR[opcode] = LW
EX/MEM.IR[rt] =$1

sono uguali!
riconosciuta dipendenza RAW risolvibile con forward

Read after ***Write***

dipendenze RAW da istr Load

Possibili casi

Situazione	Esempio di codice	Azione

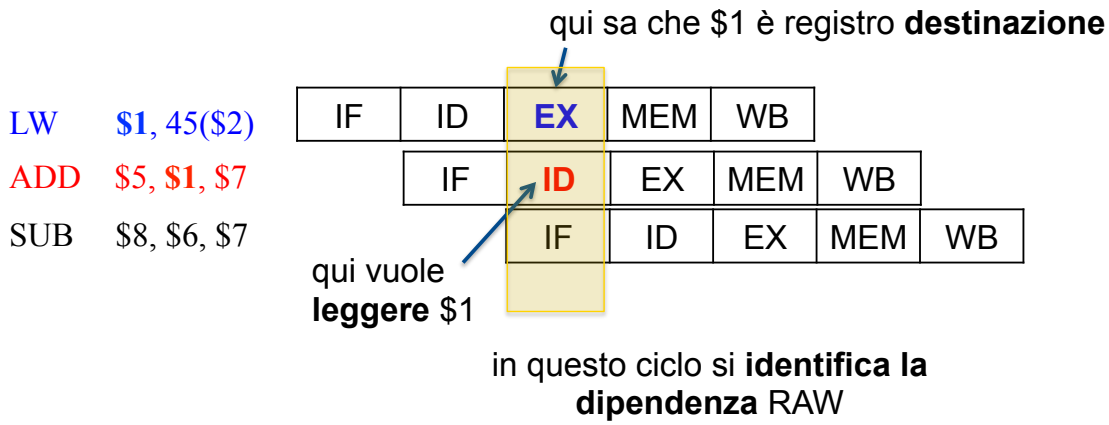
Nessuna dipendenza	LW $1, 45($2)
	ADD $5, $6, $7
	SUB $8, $6, $7
	OR $9, $6, $7
		Non occorre fare nulla

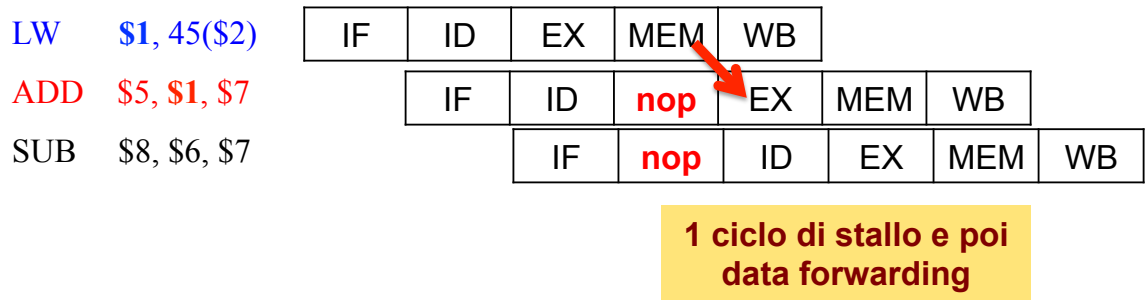
	LW $1, 45($2)
	ADD $5, $1, $7
	SUB $8, $6, $7
	OR $9, $6, $7	

Dipendenza **risolvibile con un forwarding**	LW $1, 45($2)
	ADD $5, $6, $7
	SUB $8, $1, $7
	OR $9, $6, $7
		Opportuni comparatori rilevano l'uso di $1 in SUB e inoltrano il risultato della load alla ALU in tempo per la fase EX di SUB

Dipendenza con accessi in ordine	LW $1, 45($2)
	ADD $5, $6, $7
	SUB $8, $6, $7
	OR $9, $1, $7
		Non occorre fare nulla perché la lettura di $1 in OR avviene dopo la scrittura del dato caricato

RAW

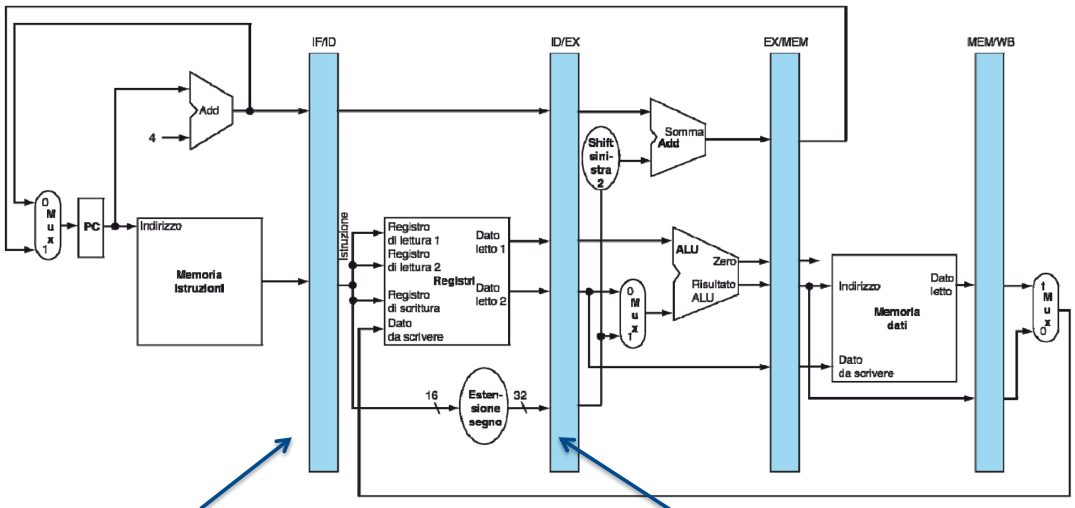




1 ciclo di stallo e poi data forwarding

IF ID EX MEM WB

SUB $8, $6, $7 ADD $5, **$1**, $7 LW **$1**, 45($2)



IF/ID.IR[opcode] =Reg-Reg ALU
IF/ID.IR[rs] = $1

ID/EX.IR[opcode] = LW
ID/EX.IR[rt] =$1

sono uguali!

Read after

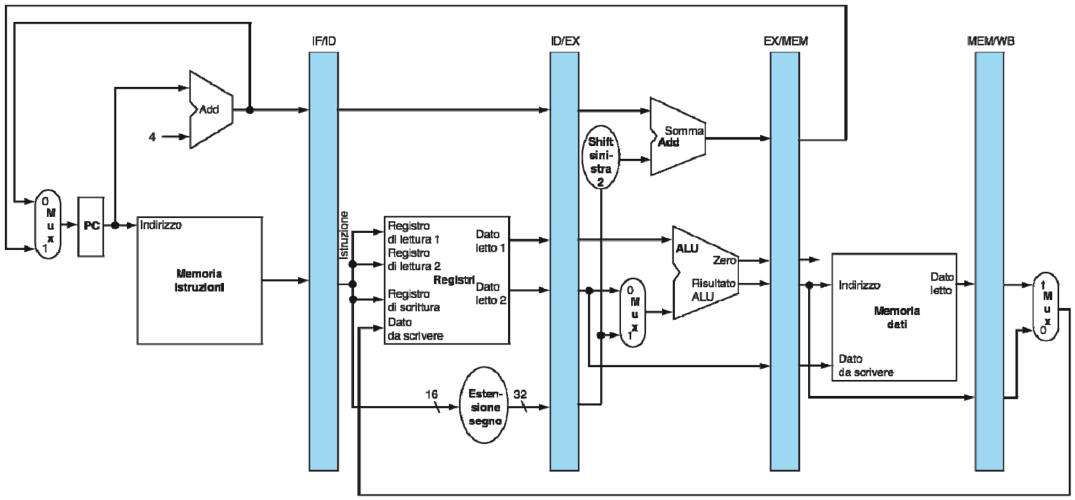
riconosciuta dipendenza RAW che richiede stallo

Write

ciclo successivo

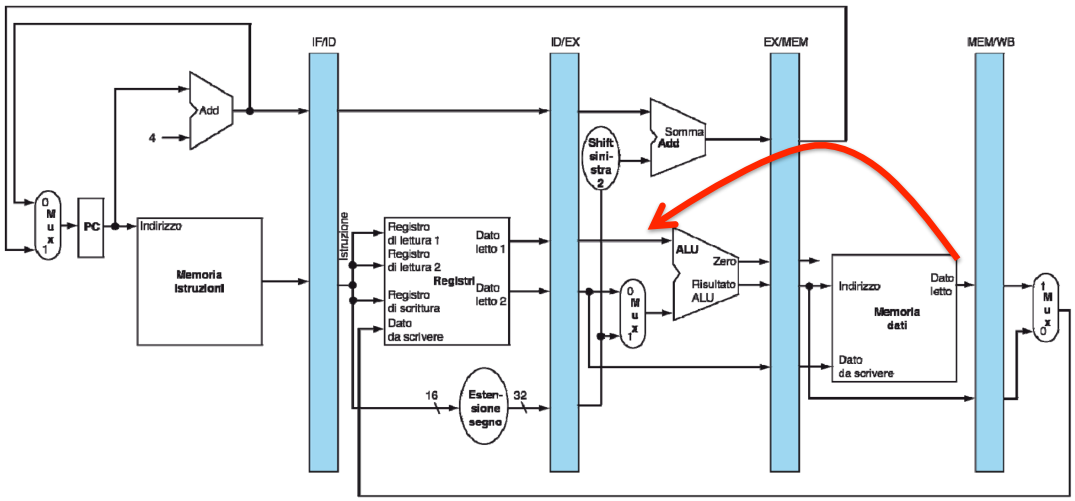
IF ID EX MEM WB

SUB $8, $6, $7 ADD $5, **$1**, $7 **bubble** LW **$1**, 45($2)



ciclo successivo

IF	ID	EX	MEM	WB
	SUB $8, $6, $7	ADD $5, **$1**, $7	**bubble**	LW **$1**, 45($2)



- se MEM/WB.IR[rt] == ID/EX.IR[rs] allora manda MEM/WB.LMD a Top ALU Input

dipendenze RAW da istr Load

Possibili casi

Situazione	Esempio di codice	Azione

Nessuna dipendenza	LW **$1**, 45($2)
	ADD $5, $6, $7
	SUB $8, $6, $7
	OR $9, $6, $7
		Non occorre fare nulla

Dipendenza che **richiede uno stallo**	LW **$1**, 45($2)
	ADD $5, **$1**, $7
	SUB $8, $6, $7
	OR $9, $6, $7
		Opportuni comparatori rilevano l'uso di $1 in ADD ed evitano il rilascio di ADD (quindi stallo)

Dipendenza risolvibile con un forwarding	LW **$1**, 45($2)
	ADD $5, $6, $7
	SUB $8, **$1**, $7
	OR $9, $6, $7
		Opportuni comparatori rilevano l'uso di $1 in SUB e inoltrano il risultato della load alla ALU in tempo per la fase EX di SUB

Dipendenza con accessi in ordine	LW **$1**, 45($2)
	ADD $5, $6, $7
	SUB $8, $6, $7
	OR $9, **$1**, $7
		Non occorre fare nulla perché la lettura di $1 in OR avviene dopo la scrittura del dato caricato

Pipeline (MIPS)

Condizioni per riconoscere le dipendenze da una LOAD che richiedono uno stallo

Opcode (ID/EX)	Opcode (IF/ID)	Matching operand fields
Load	R-R ALU	ID/EX.IR[rt] == IF/ID.IR[rs]
Load	R-R ALU	ID/EX.IR[rt] == IF/ID.IR[rt]
Load	Load, Store, Imm ALU, branch	ID/EX.IR[rt] == IF/ID.IR[rs]

Load in fase EX

Istruzione successiva in fase ID

registro di scrittura di Load

registro di lettura di Istruzione successiva

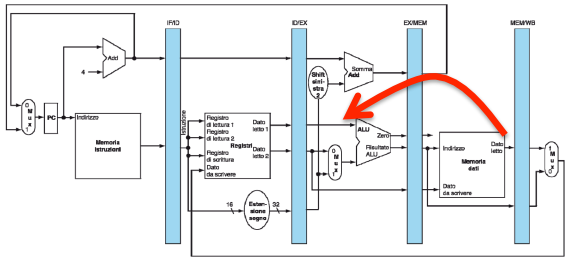
condizioni verificate da Hazard Detection Unit

Pipeline (MIPS)

Condizioni per decidere come effettuare il forwarding

Tutti i **dati** su cui effettuare il forwarding

- **provengono**:
  - dalla memoria dati
  - dall'output della ALU
  - quindi stanno in registro MEM/WB opp EX/MEM
- e sono **diretti** verso:
  - l'input della ALU
  - l'input della memoria dati
  - quindi verso registri ID/EX e EX/MEM



il **campo destinazione di IR** (cioe' quale registro viene ***scritto*** dall'istruzione) in EX/MEM e MEM/WB

Quindi occorre confrontare:
se sono uguali va fatta la propagazione
(dall'Unità di Propagazione)

con i **campi sorgente di IR** (cioè quale registro viene ***letto***) in ID/EX e EX/MEM

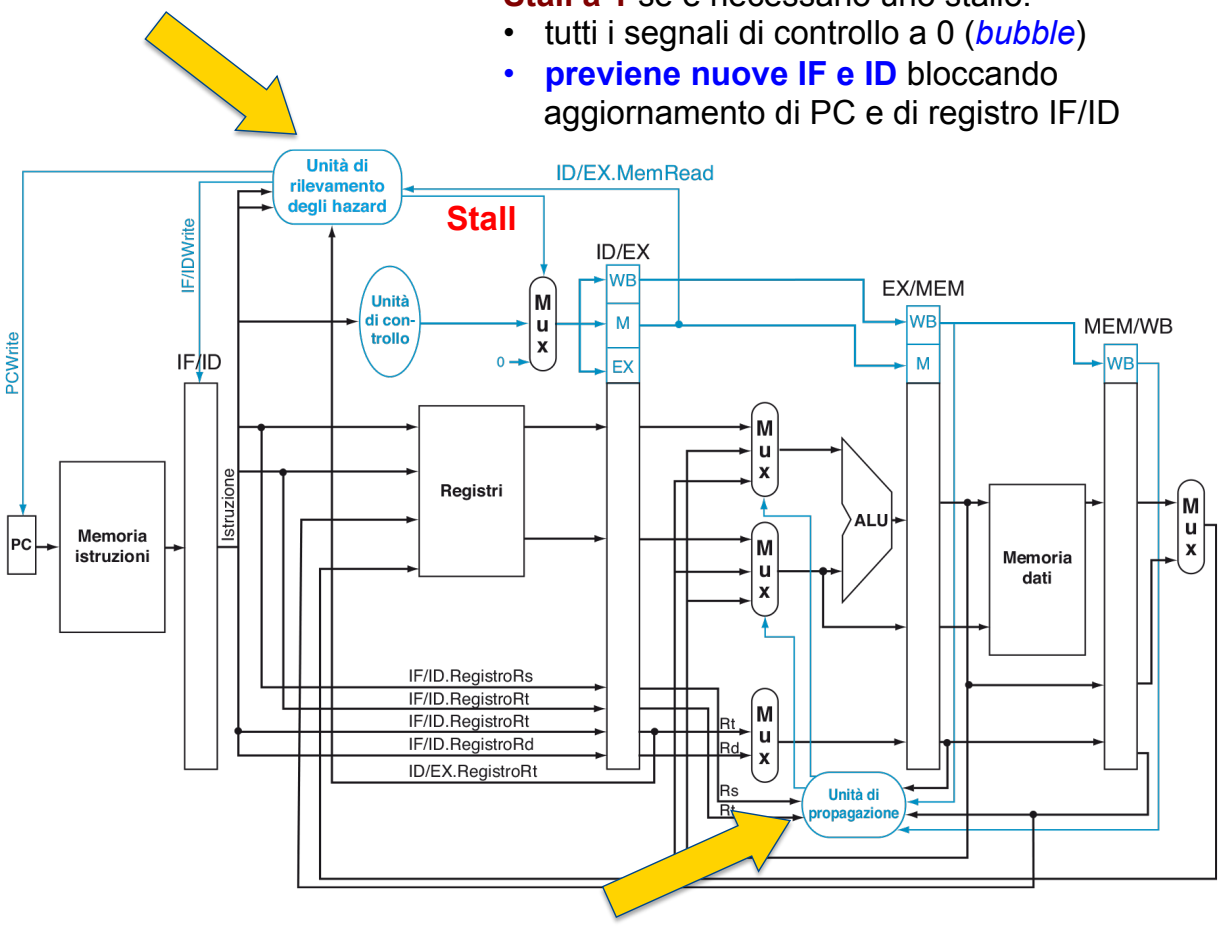
condizioni per la sola propagazione all'input di ALU

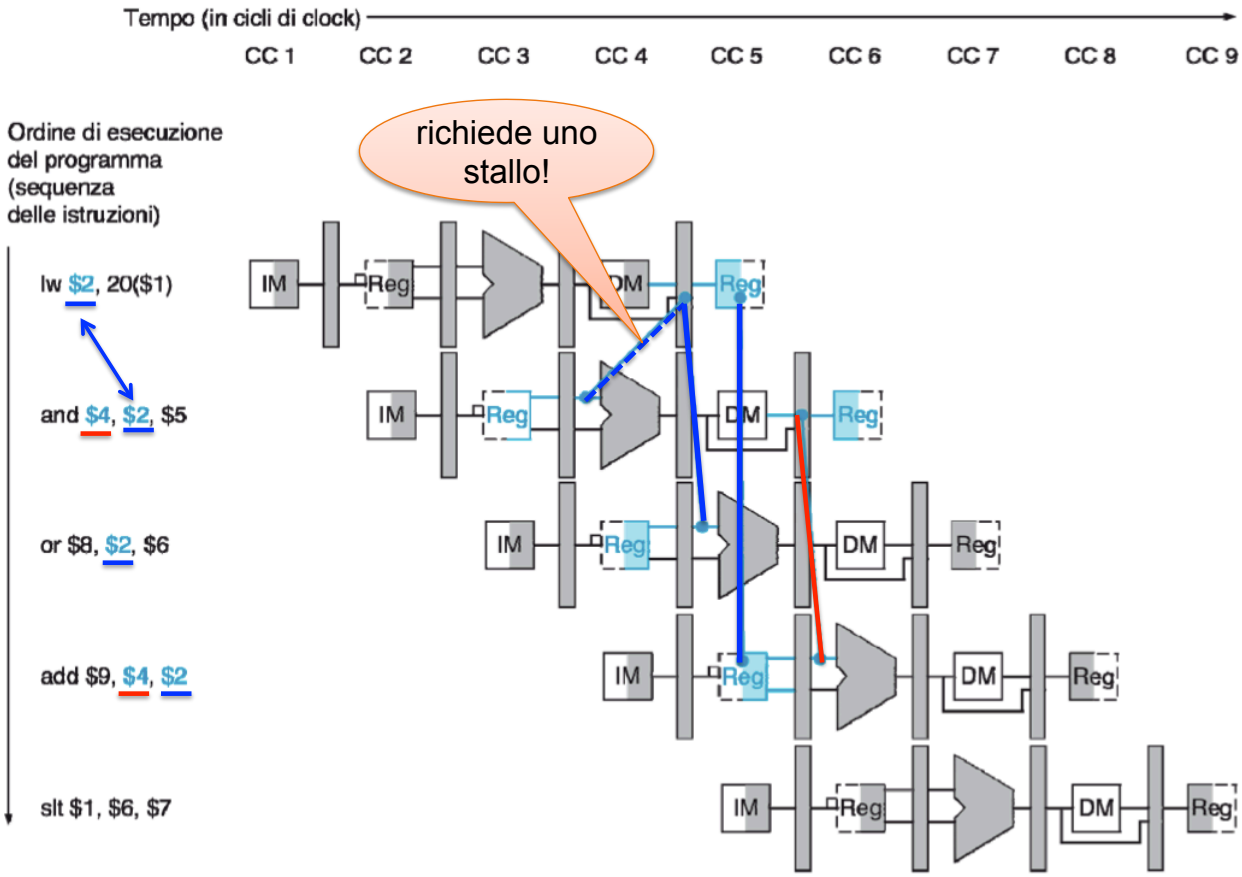
Pipeline (MIPS)

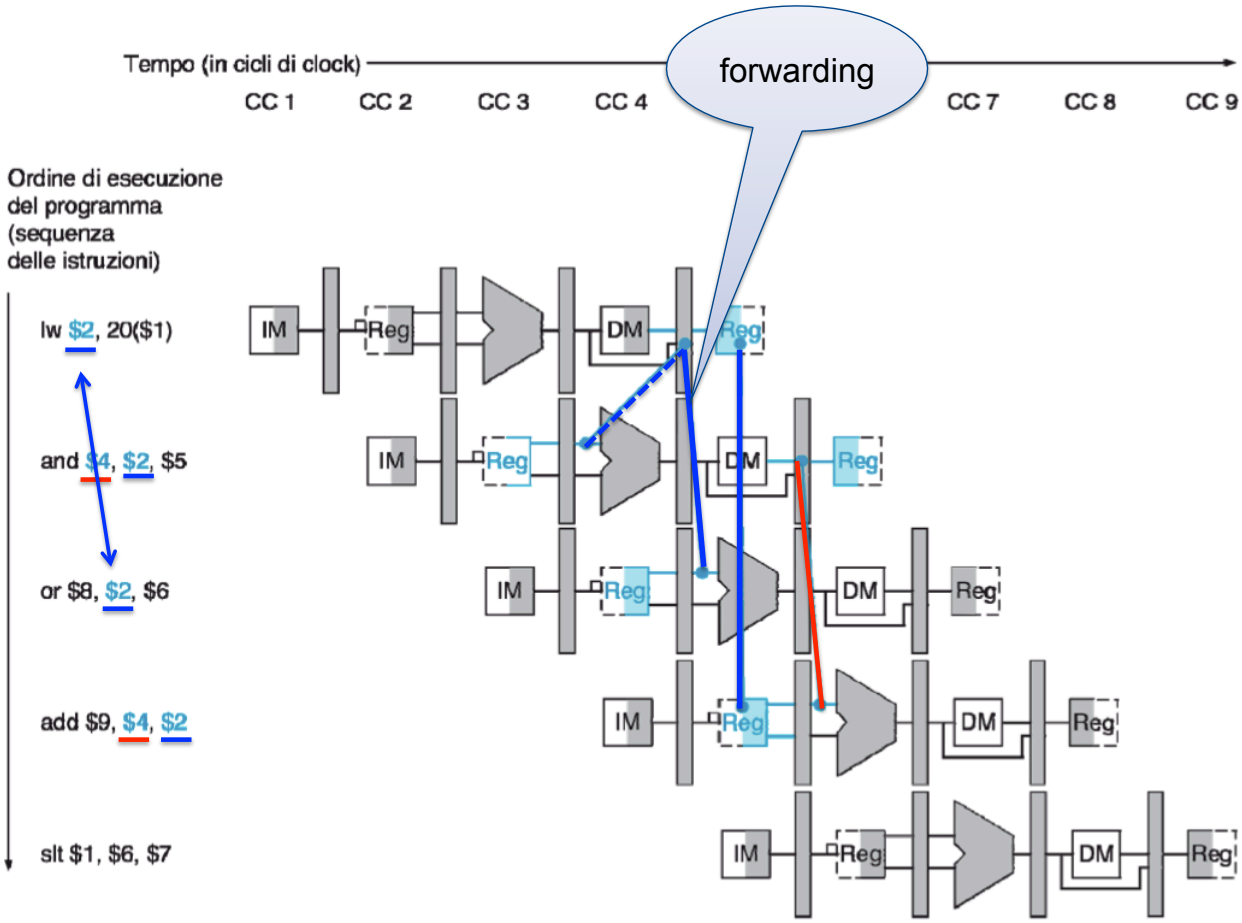
Pipeline register containing source instruction	Opcode of source instruction	Pipeline register containing destination instruction	Opcode of destination instruction	Destination of the forwarded result	Comparison (if equal then forward)
EX/MEM	Register-register ALU	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	EX/MEM.IR[rd] == ID/EX.IR[rs]
EX/MEM	Register-register ALU	ID/EX	Register-register ALU	Bottom ALU input	EX/MEM.IR[rd] == ID/EX.IR[rt]
MEM/WB	Register-register ALU	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	MEM/WB.IR[rd] == ID/EX.IR[rs]
MEM/WB	Register-register ALU	ID/EX	Register-register ALU	Bottom ALU input	MEM/WB.IR[rd] == ID/EX.IR[rt]
EX/MEM	ALU immediate	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	EX/MEM.IR[rt] == ID/EX.IR[rs]
EX/MEM	ALU immediate	ID/EX	Register-register ALU	Bottom ALU input	EX/MEM.IR[rt] == ID/EX.IR[rt]
MEM/WB	ALU immediate	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	MEM/WB.IR[rt] == ID/EX.IR[rs]
MEM/WB	ALU immediate	ID/EX	Register-register ALU	Bottom ALU input	MEM/WB.IR[rt] == ID/EX.IR[rt]
MEM/WB	Load	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	MEM/WB.IR[rt] == ID/EX.IR[rs]
MEM/WB	Load	ID/EX	Register-register ALU	Bottom ALU input	MEM/WB.IR[rt] == ID/EX.IR[rt]

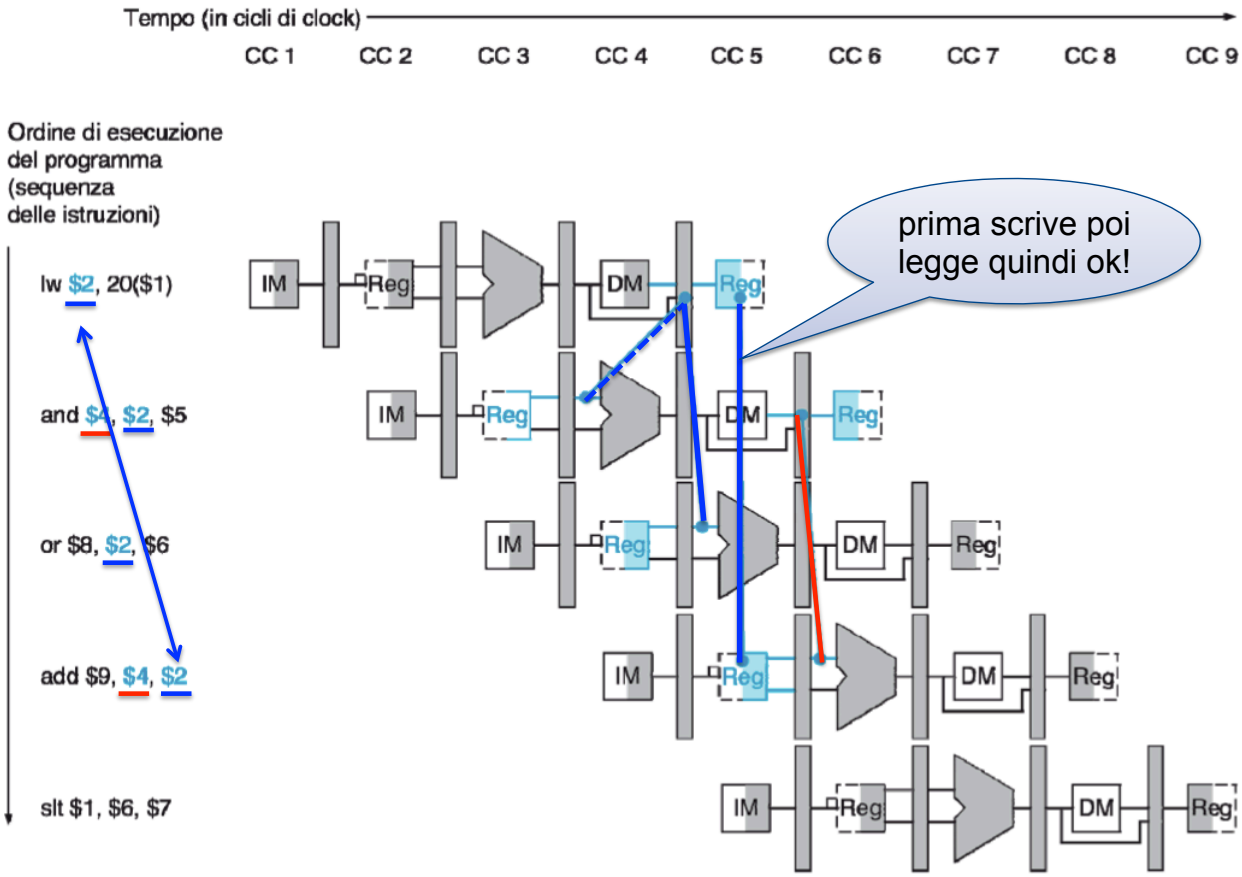
Stall a 1 se è necessario uno stallo:

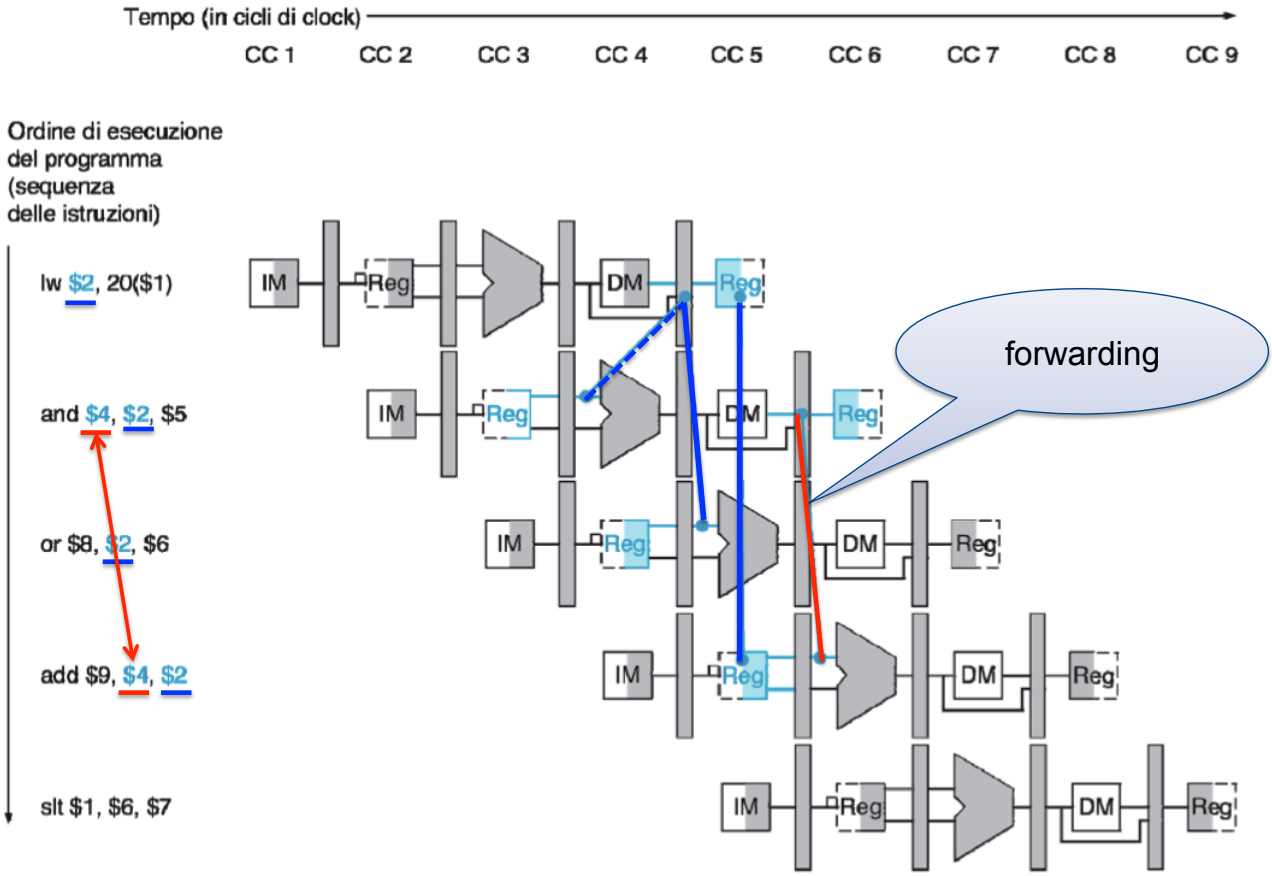
- tutti i segnali di controllo a 0 (*bubble*)
- **previene nuove IF e ID** bloccando aggiornamento di PC e di registro IF/ID

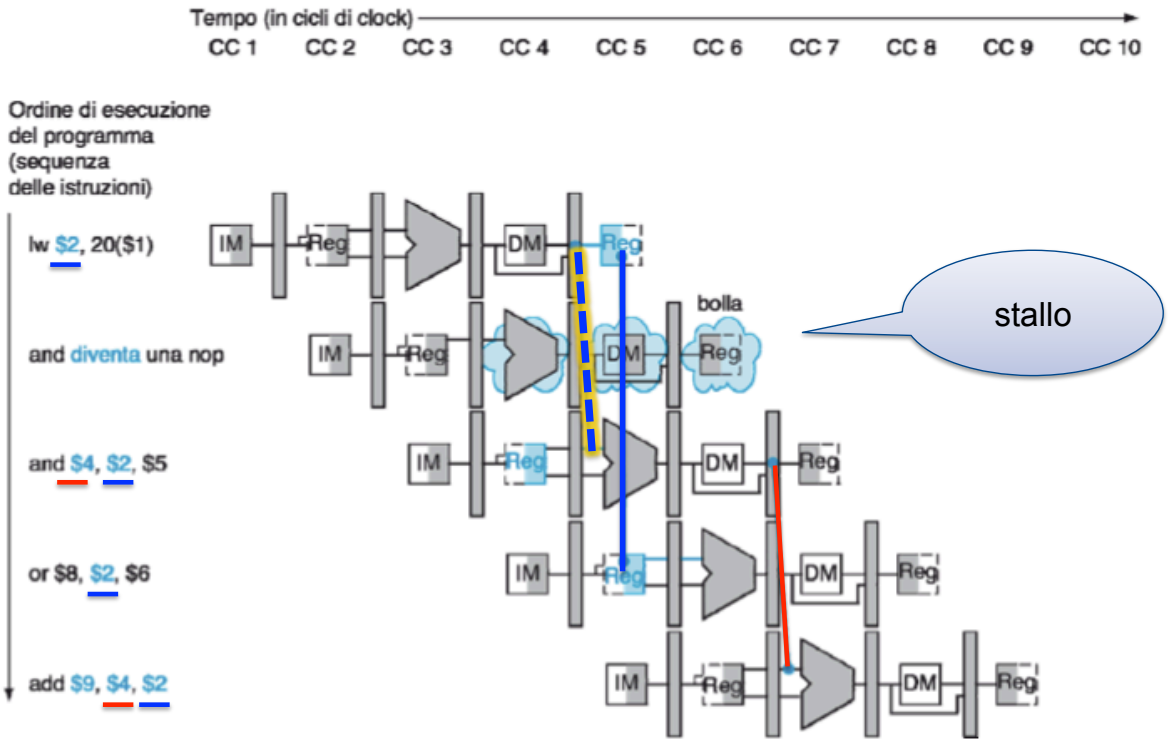



Tempo (in cicli di clock)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Ordine di esecuzione del programma (sequenza delle istruzioni)
richiede uno stallo!
lw $2, 20($1)
and $4, $2, $5
or $8, $2, $6
add $9, $4, $2
slt $1, $6, $7
IM
Reg
DM


Tempo (in cicli di clock)
forwarding
CC 1
CC 2
CC 3
CC 4
CC 7
CC 8
CC 9
Ordine di esecuzione del programma (sequenza delle istruzioni)
lw $2, 20($1)
and $4, $2, $5
or $8, $2, $6
add $9, $4, $2
slt $1, $6, $7
IM
Reg
DM

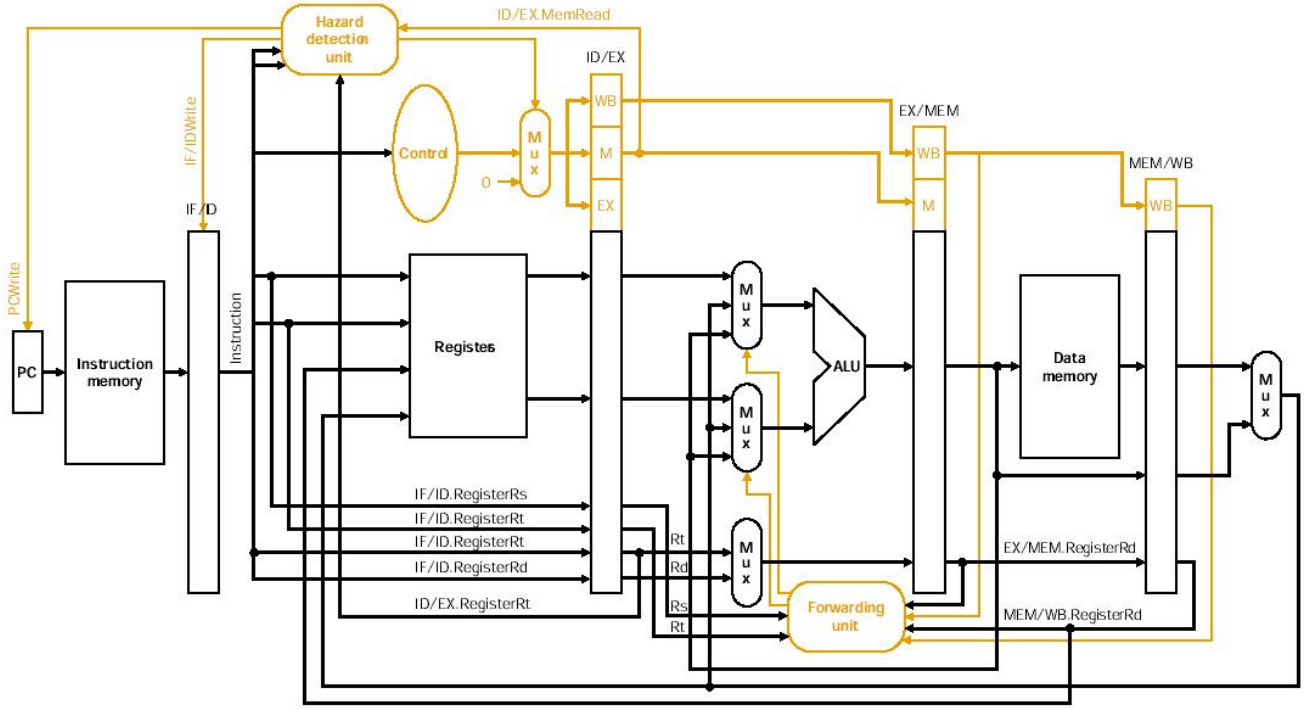

Tempo (in cicli di clock)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Ordine di esecuzione del programma (sequenza delle istruzioni)
lw $2, 20($1)
and $4, $2, $5
or $8, $2, $6
add $9, $4, $2
slt $1, $6, $7
IM
Reg
DM
prima scrive poi legge quindi ok!

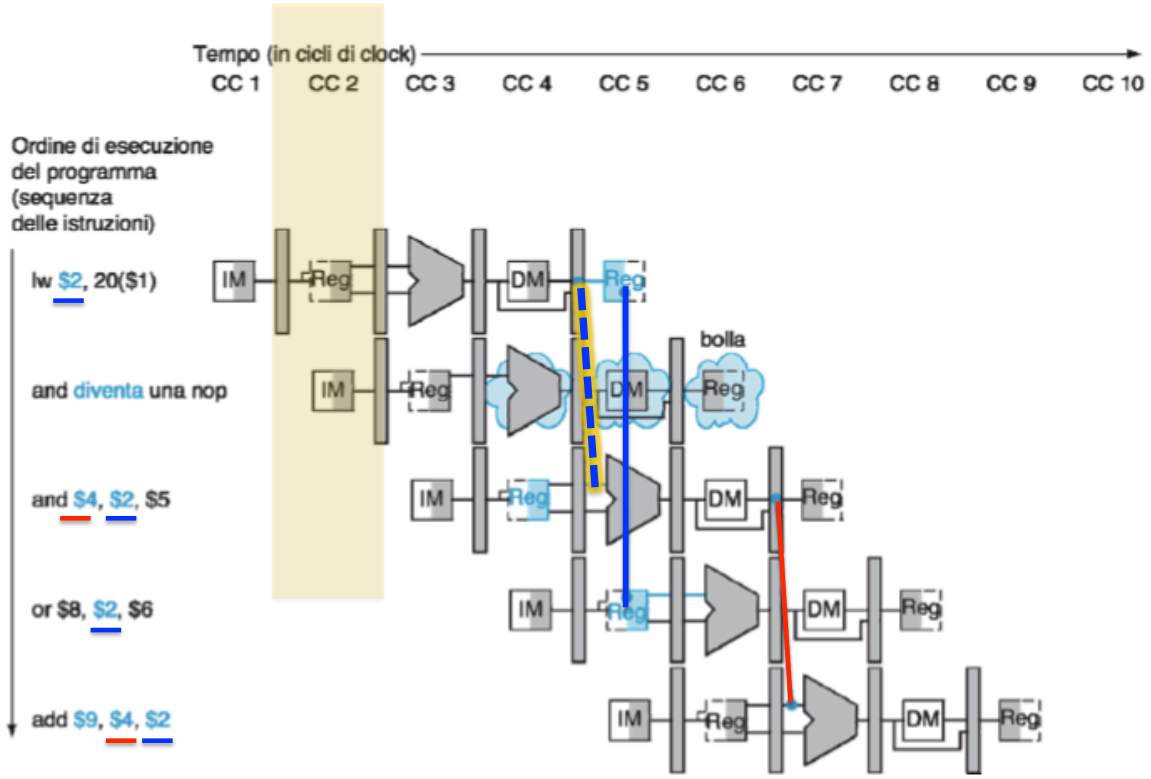

Tempo (in cicli di clock)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Ordine di esecuzione del programma (sequenza delle istruzioni)
lw $2, 20($1)
and $4, $2, $5
or $8, $2, $6
add $9, $4, $2
slt $1, $6, $7
IM
Reg
DM
forwarding



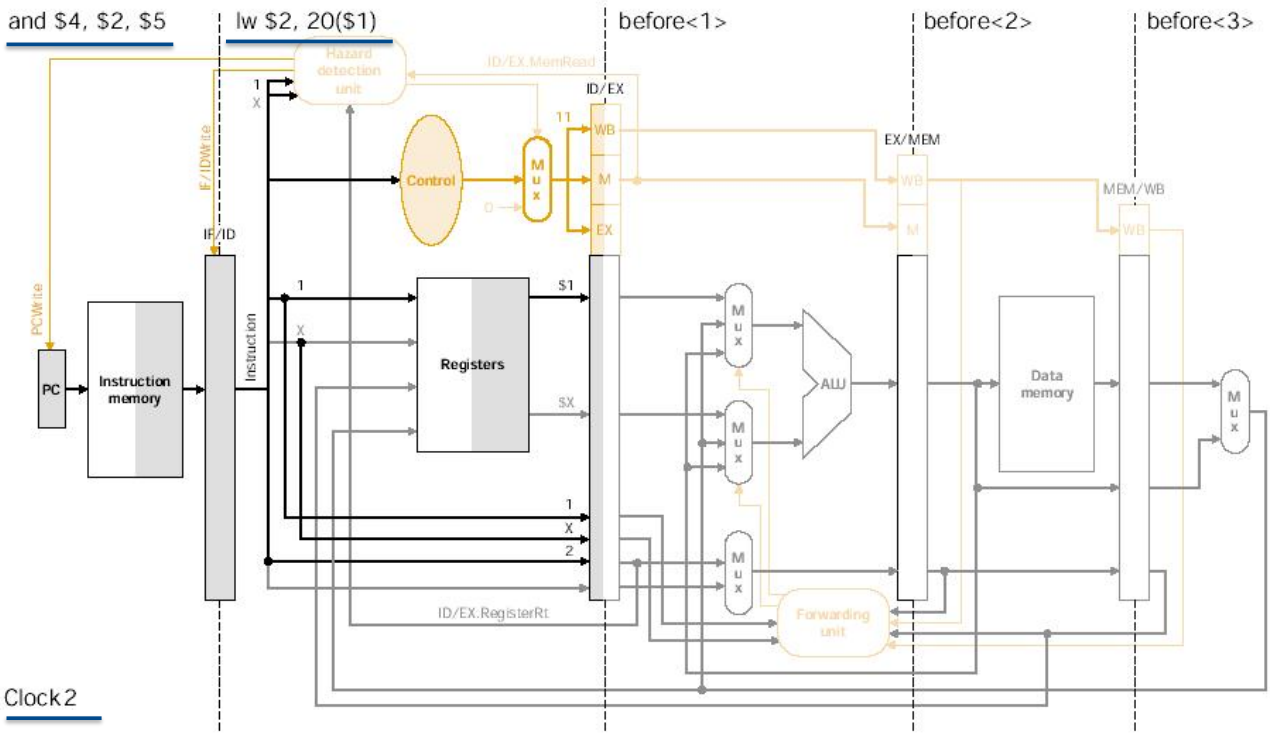
Pipeline (MIPS)

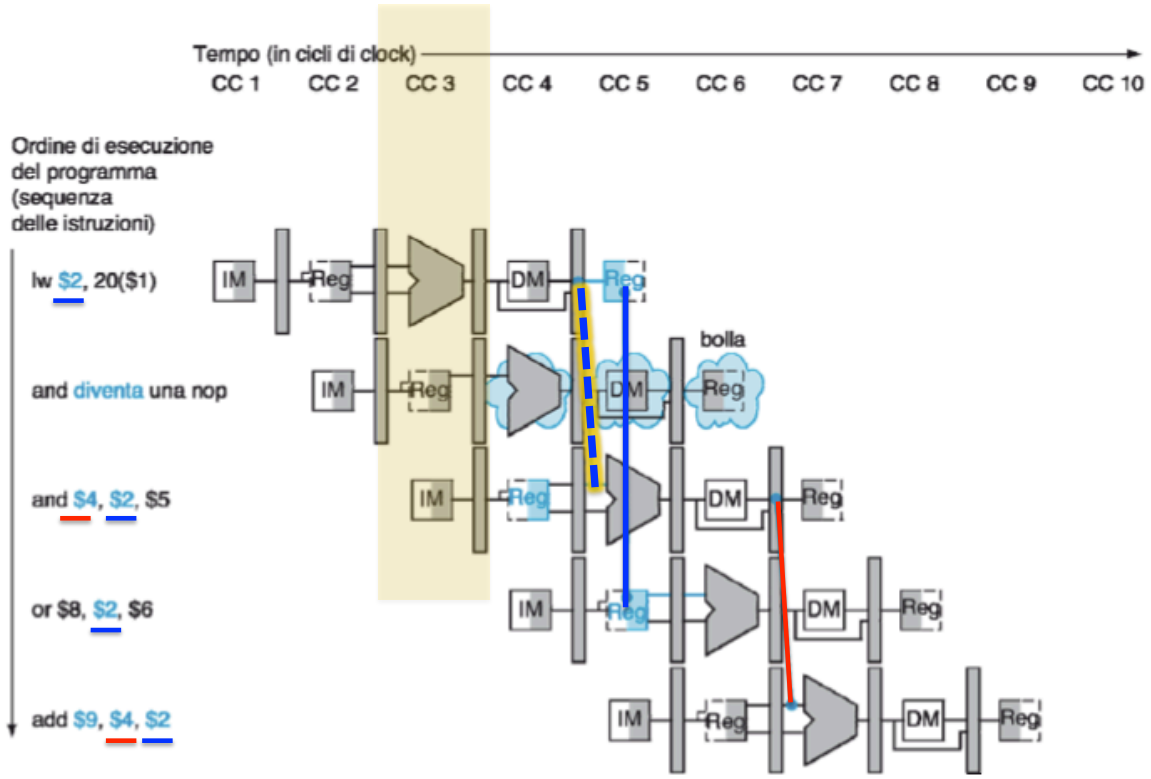
Introduzione hardware aggiuntivo per gestire il data forwarding



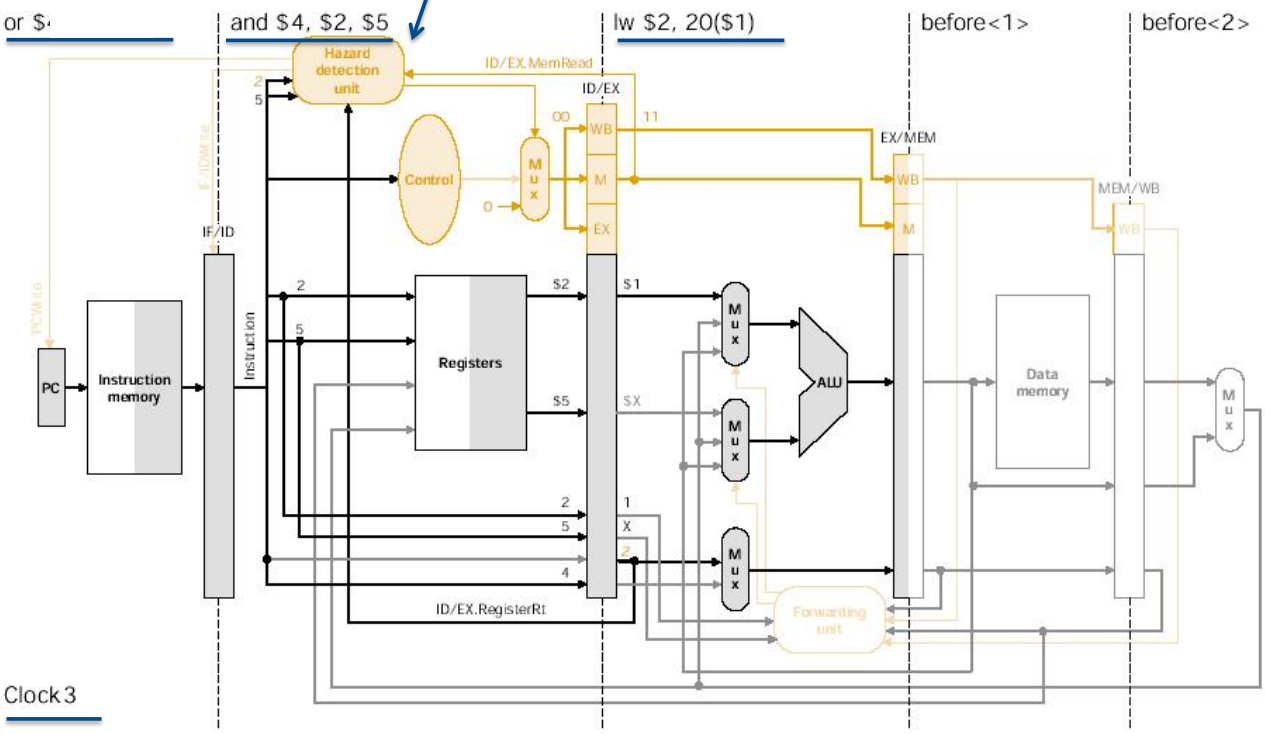


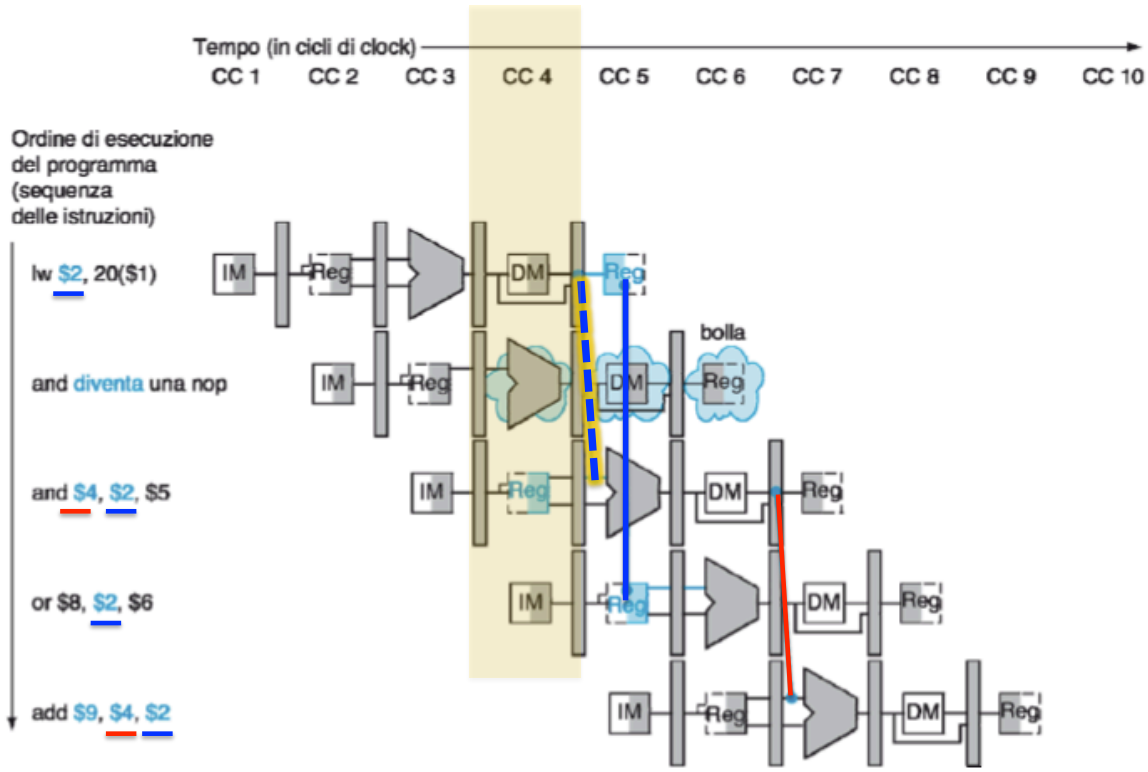
Pipeline (MIPS)





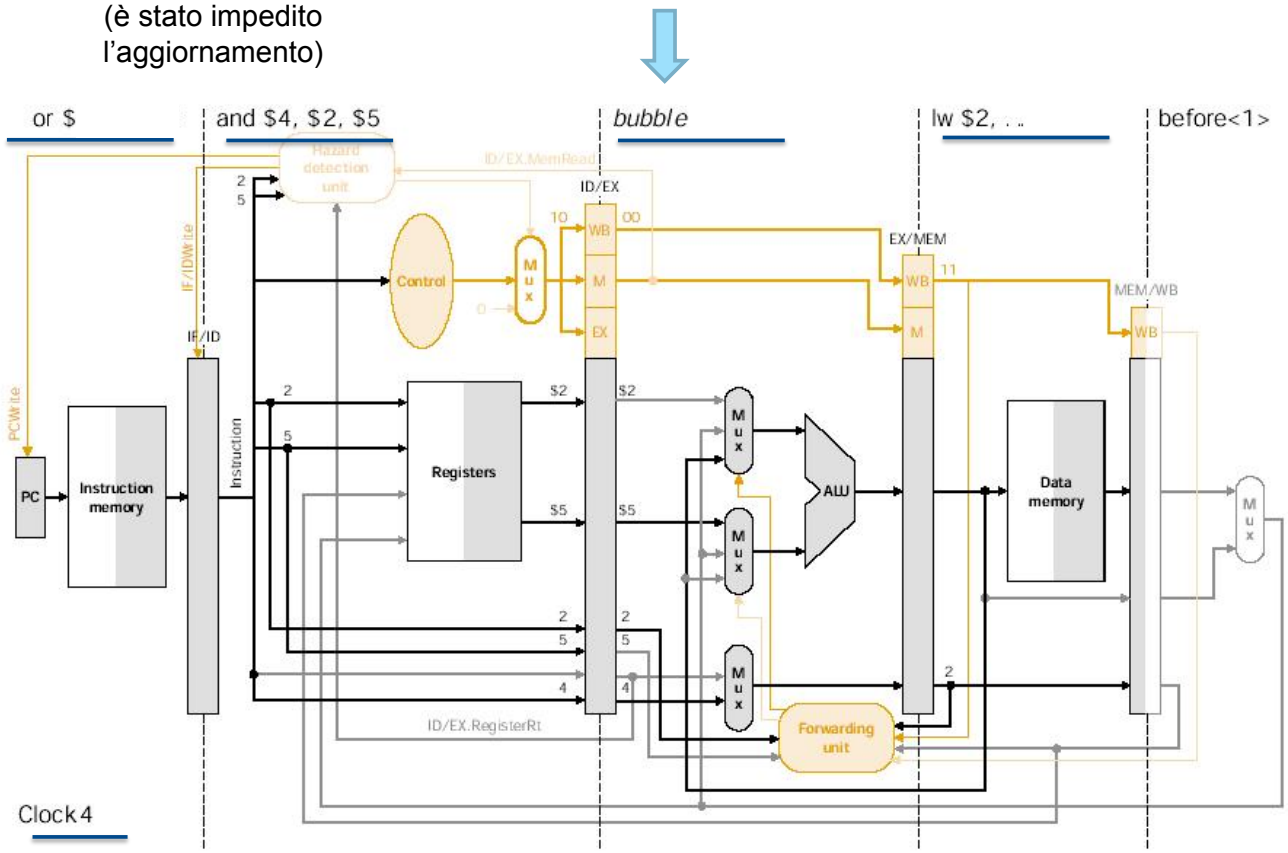
IF/ID.IR[rs]==ID/EX.IR[rt] quindi **imposta Stall a 1**

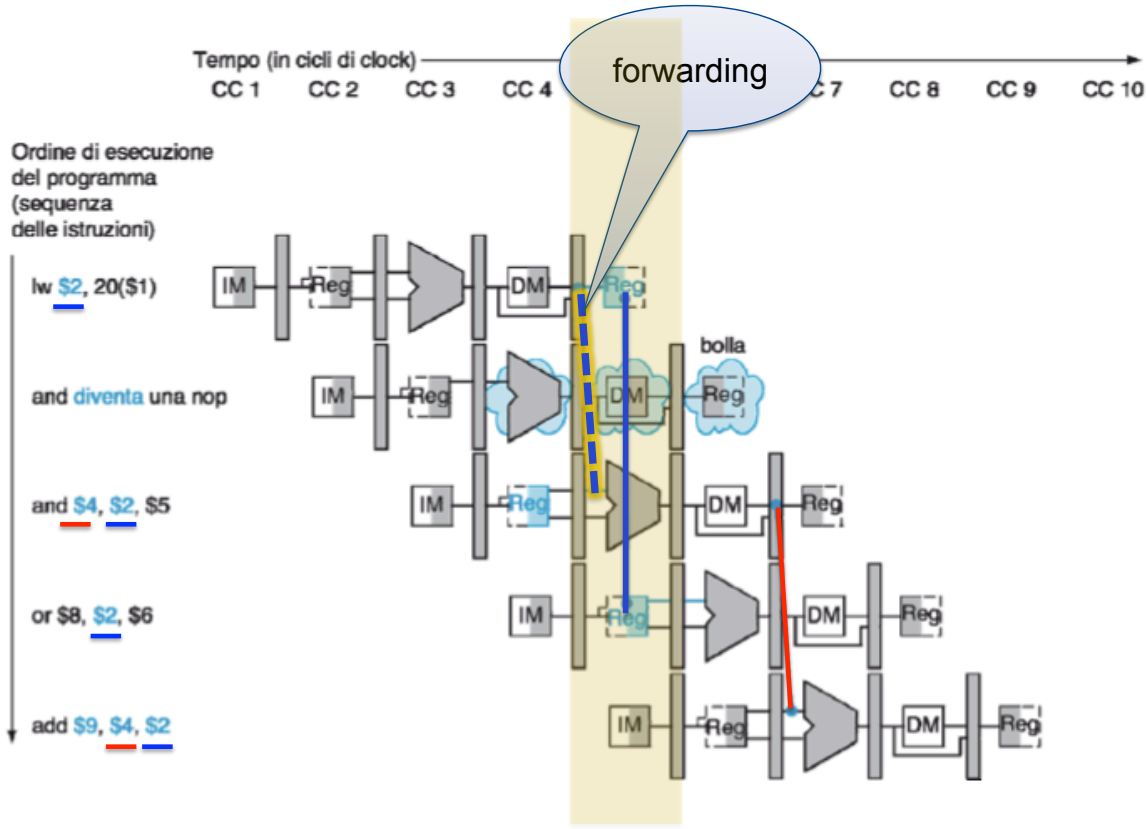


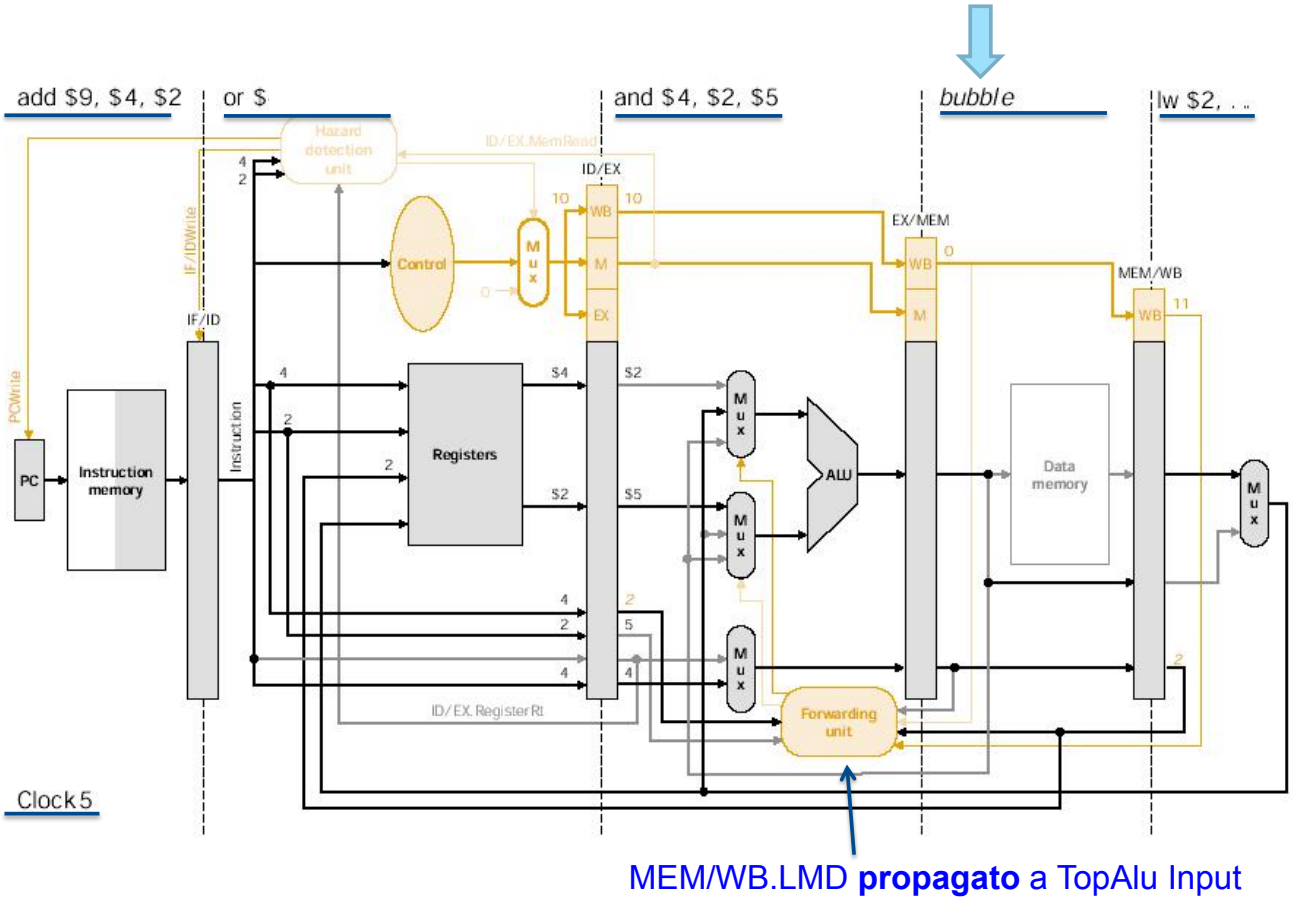


IF e ID sono ripetute

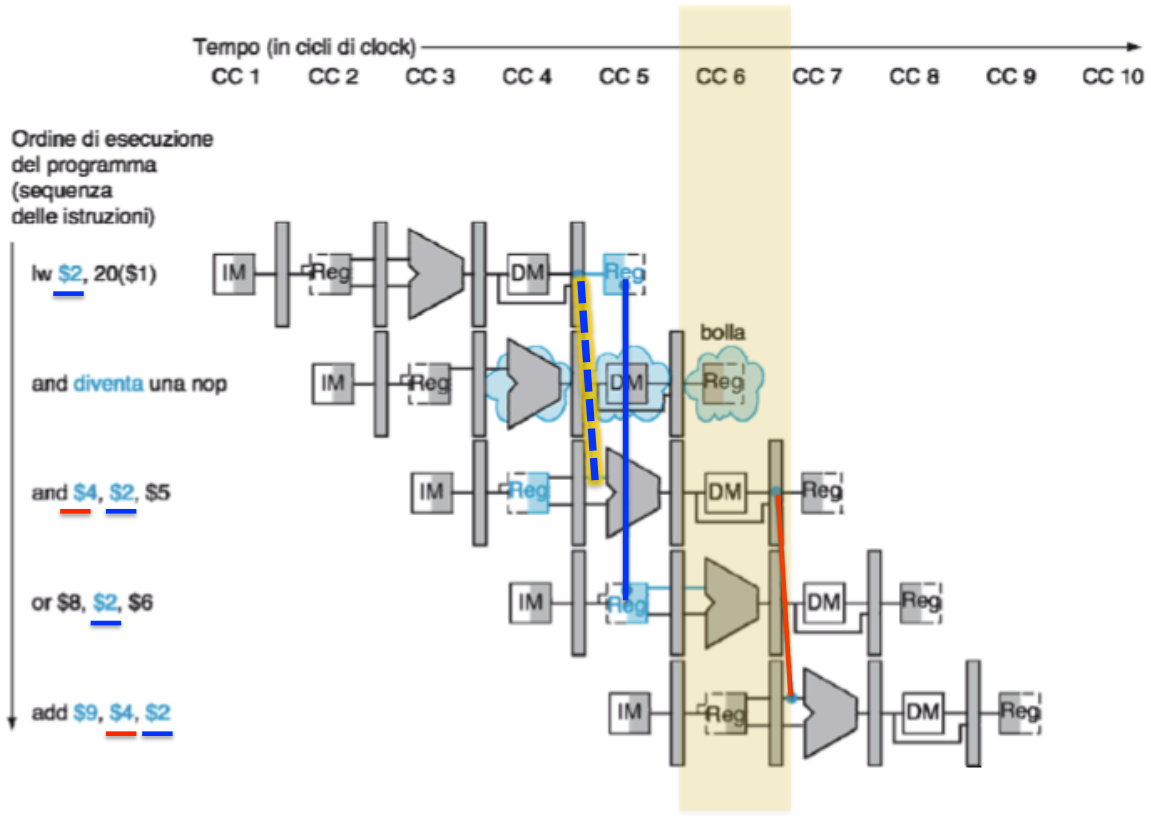
(è stato impedito l'aggiornamento)



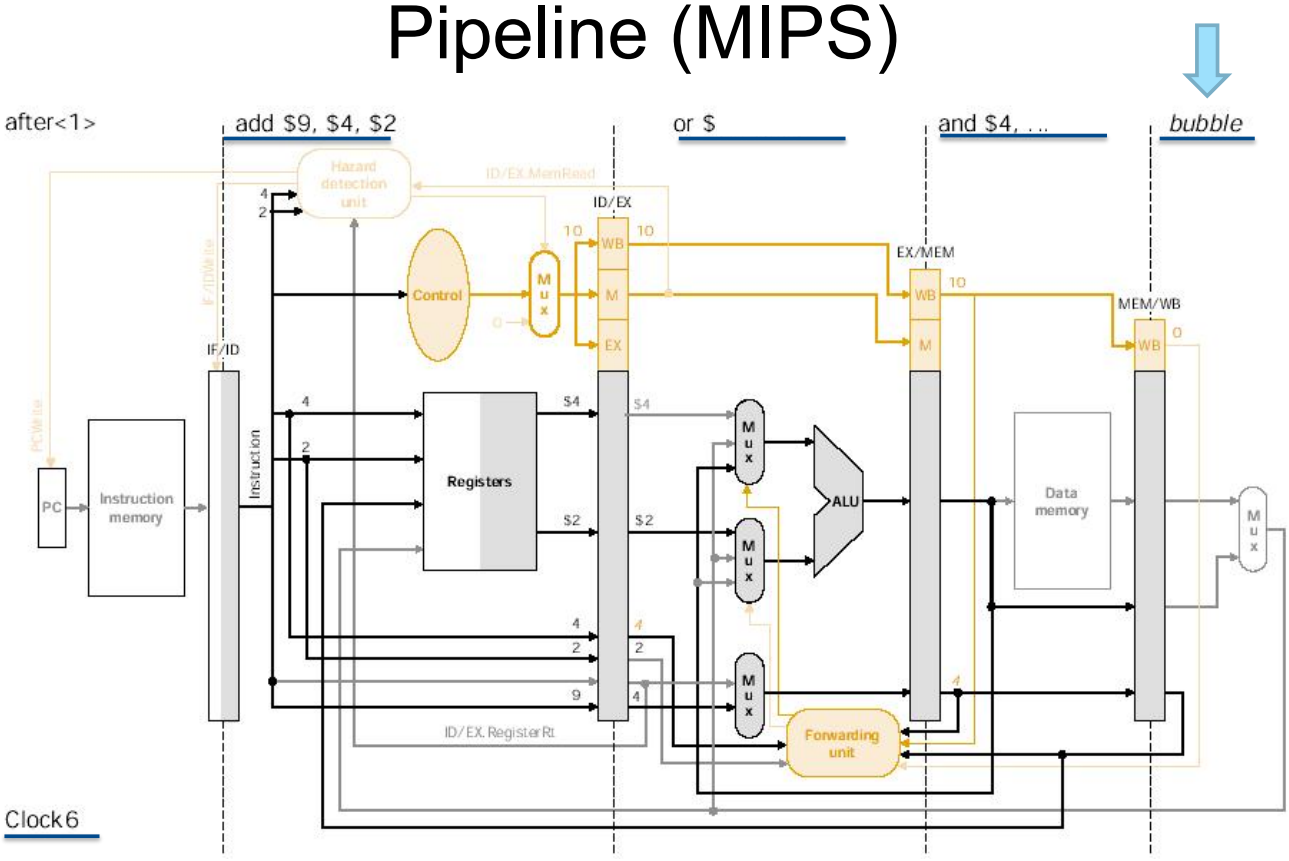


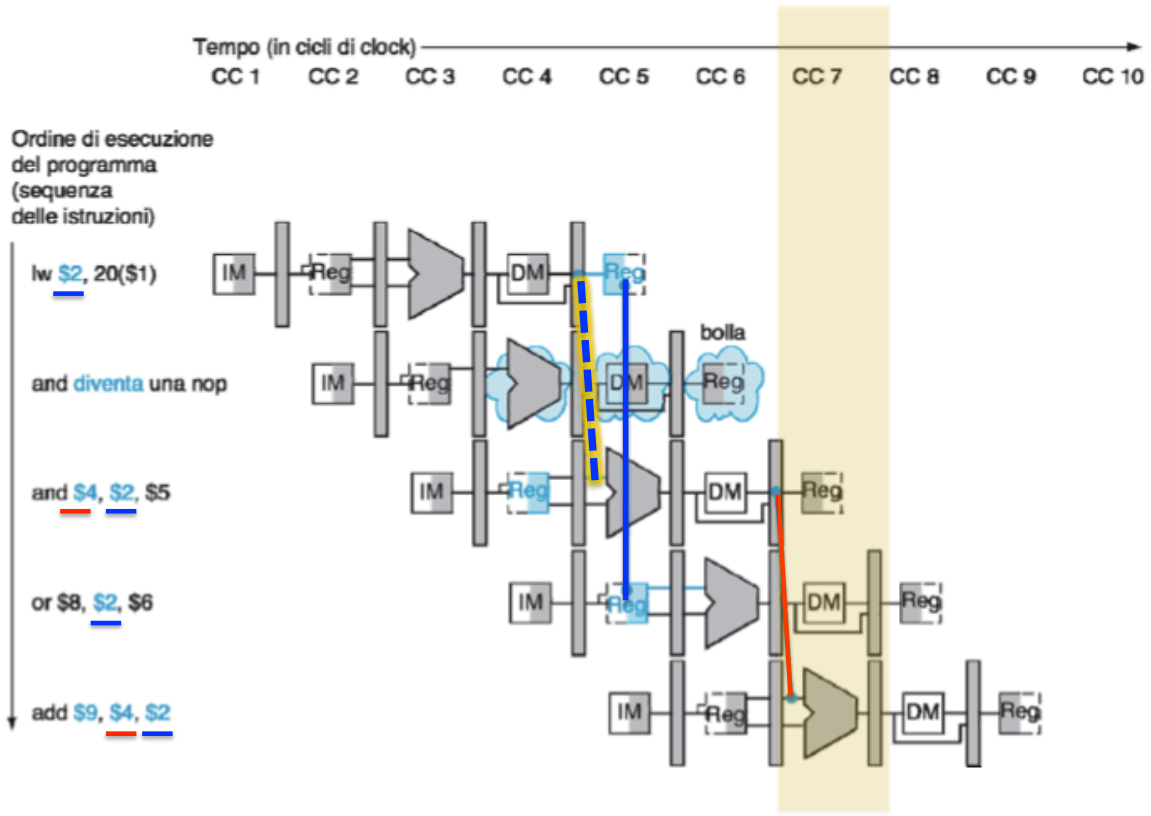


MEM/WB.LMD **propagato** a TopAlu Input

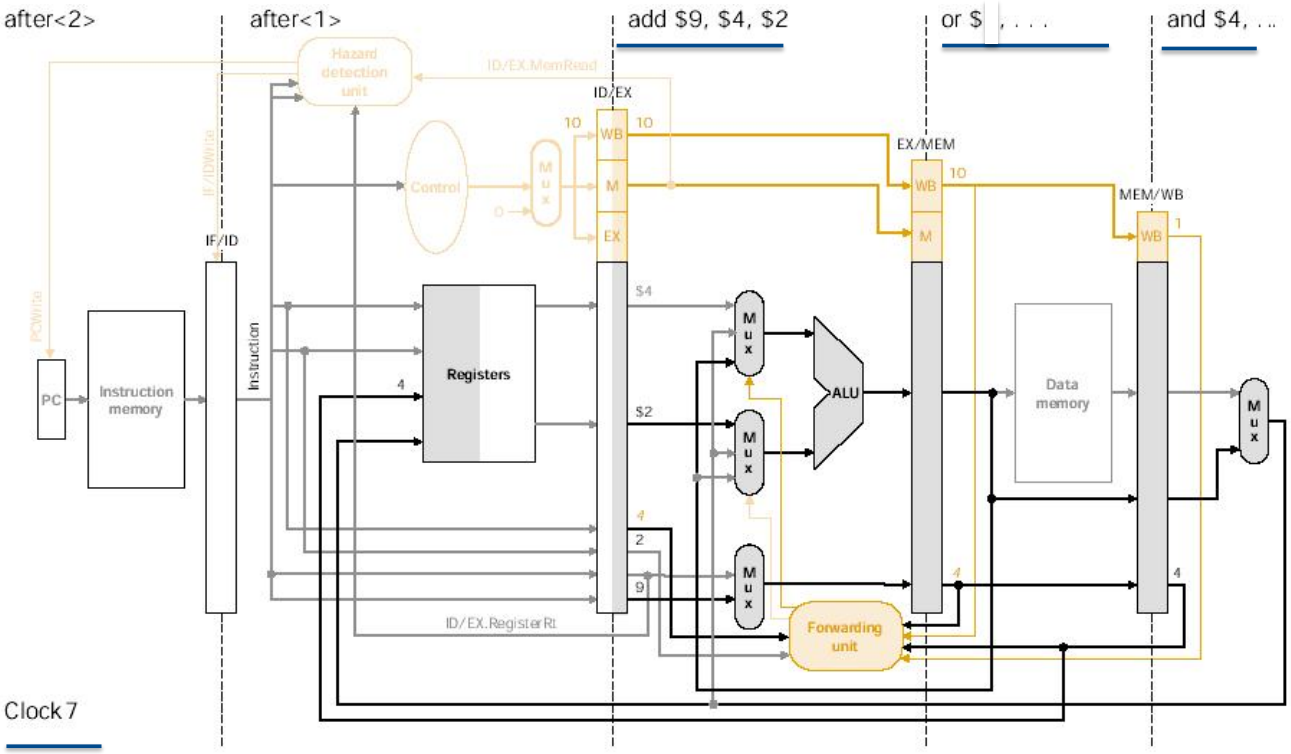


Pipeline (MIPS)

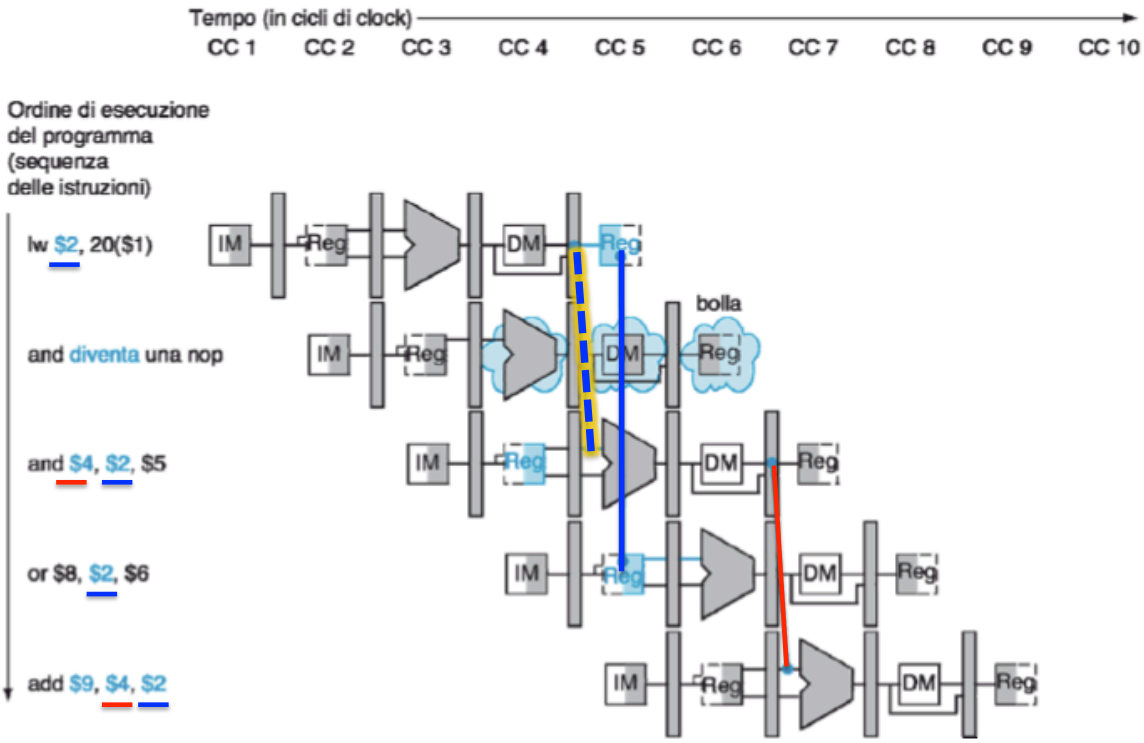




Pipeline (MIPS)



MEM/WB.AluOutput **propagato** a TopAlu Input


Tempo (in cicli di clock)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
CC 10
Ordine di esecuzione del programma (sequenza delle istruzioni)
lw $2, 20($1)
and diventa una nop
and $4, $2, $5
or $8, $2, $6
add $9, $4, $2
bolla
IM
Reg
DM