

Famiglia RISC-V

- Studiamo i processori RISC-V come esempio di architettura RISC
- Sviluppato alla UC Berkeley dal 2010
 - Per molti aspetti ispirato a MIPS, una architettura sperimentale sviluppata a Stanford negli anni '80 e poi sviluppata commercialmente
- Riferimenti: *Hennessy & Patterson “Struttura e progetto dei calcolatori: Progettare con RISC-V”*, in Biblioteca
 - Versione inglese: *Hennessy & Patterson “COMPUTER ORGANIZATION AND DESIGN RISC-V EDITION: the hardware-software interface” 2nd edition*
- Architettura **molto regolare**
- architettura progettata per un'implementazione **efficiente della pipeline**
 - MIPS = *microprocessor without interlocked pipeline stages*
- Nota: MIPS è un'azienda che ora sviluppa CPU basate sull'architettura RISC-V

RISC-V (a 32 bit)

Istruzioni:

- Tutte le istruzioni di **dimensione 32 bit**
- tutte le **operazioni sui dati** sono da registro a registro
 - le istruzioni che manipolano i dati usano i valori dei registri
- le **operazioni sulla memoria**:
 - solo *load* e *store*, per trasferire dati tra memoria e registri
 - nessuna operazione memoria-memoria
- quindi **tutte le istruzioni operano su registri**,
 - es **add x1, x2, x3**

Registri:

- 32 registri di 32 bit
- si indicano con x1, x2, x3.... **x0** contiene sempre 0

RISC-V

Dati:

- Registri possono essere caricati con byte, mezze parole, e parole
- i registri sono a 32 bit, quindi un dato “più corto” può essere “allungato” riempiendo i bit rimanenti con 0 o estendendo il segno (cioè replicandolo)

Modi di indirizzamento:

- Immediato es. **addi x2, x2, 004** ← 12 bit
- Displacement es. **sw x1, 00c(x2)**
- Altre modalità derivabili:
 - Indiretta registro (displacement a 0) es. **sw x2, 000(x3)**
 - Assoluta (registro 0 come registro base) es. **lw x1, 0c4(x0)**

RISC-V: Formato Istruzioni

Istruzione	Formato	funz7	rs2	rs1	funz3	rd	codop
add (somma)	R	0000000	reg	reg	000	reg	0110011
sub (sottrazione)	R	0100000	reg	reg	000	reg	0110011
Istruzione	Formato	immediato	rs1	funz3	rd	codop	
addi (somma immediata)	I	costante	reg	000	reg	0010011	
lw (load parola)	I	indirizzo	reg	010	reg	0000011	
Istruzione	Formato	immediato	rs2	rs1	funz3	immediato	codop
sw (store parola)	S	indirizzo	reg	reg	010	indirizzo	0100011

Figura 2.5 Codifica delle istruzioni RISC-V. In questa tabella “reg” indica il numero di un registro ed è compreso tra 0 e 31, e “indirizzo” è un indirizzo su 12 bit o una costante. I campi funz3 e funz7 agiscono come codici operativi aggiuntivi.

RISC-V: Formato Istruzioni

istruzioni aritmetiche e logiche							
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit	
load, operazioni immediate, shift	funz7	rs2	rs1	funz3	rd	codop	Esempio
sub (sottrazione)	0000000	00011	00010	000	00001	0110011	add x1, x2, x3
	01001	12 bit	011	5 bit	3 bit	5 bit	sub x1, x2, x3
Istruzioni tipo-I	immediato		rs1	funz3	rd	codop	Esempio
store	001111101000		00010	000	00001	0010011	addi x1,x2,1000
lw (load word)	7 bit	101	5 bit	3 bit	5 bit	7 bit	lw x1, 1000 (x2)
Istruzioni tipo-S	immediato	rs2	rs1	funz3	immediato	codop	Esempio
sw (store parola)	0011111	00001	00010	010	01000	0100011	sw x1, 1000 (x2)

Figura 2.6 L'architettura RISC-V descritta fino al Paragrafo 2.5. I tre formati delle istruzioni RISC-V visti finora sono R, I e S. Il formato R ha due registri sorgenti per gli operandi e un registro destinazione. Il formato I sostituisce uno dei due registri sorgente con la parte meno significativa del campo *immediato* su 12 bit. Il formato S ha due operandi sorgente e un campo immediato su 12 bit, ma non ha un registro destinazione. Il campo immediato delle istruzioni di tipo S viene suddiviso in due parti: i bit 11–5 sono contenuti nel campo immediato più a sinistra e i bit 4–0 nel secondo campo da destra.

RISC-V: Formati principali

- 32 bit per tutte le istruzioni, 6 diversi formati (di cui 4 principali):

Nome (dimensione del campo)	Campi					Commenti	
	7 bit	5 bit	5 bit	3 bit	5 bit		
Tipo R	funz7	rs2	rs1	funz3	rd	codop	Istruzioni aritmetiche
Tipo I	Immediato[11:0]		rs1	funz3	rd	codop	Istruzioni di caricamento dalla memoria e aritmetica con costanti
Tipo S	immed[11:5]	rs2	rs1	funz3	immed[4:0]	codop	Istruzioni di trasferimento alla memoria (store)
Tipo U	immediato[31:12]				rd	codop	Formato caricamento stringhe di bit più significativi

Figura 2.19 Quattro dei formati delle istruzioni RISC-V. La Figura 4.16 rivelerà i due formati RISC-V mancanti: SB per i salti condizionati e UJ per i salti incondizionati. I formati SB e UJ prevedono gli stessi campi con la stessa ampiezza dei formati S e U ma che hanno alcuni bit dell'indirizzo spostati. Il motivo dietro questa codifica diventa chiaro dopo avere capito il funzionamento dell'hardware. Come vedremo nel Capitolo 4, i formati SB e UJ semplificano l'hardware ma producono un lavoro maggiore agli assemblatori (e agli autori!).

Dimensione fissa di opcode e di riferimenti a registri

- semplifica Instruction Decode e Fetch di Operandi
- campi hanno una semantica specifica e si trovano sempre nella stessa posizione
- per cui servono + formati rispetto ad esempio a MIPS

Istruzioni – Formato R

Istruzioni aritmetiche e logiche: 6 campi a dimensione fissa

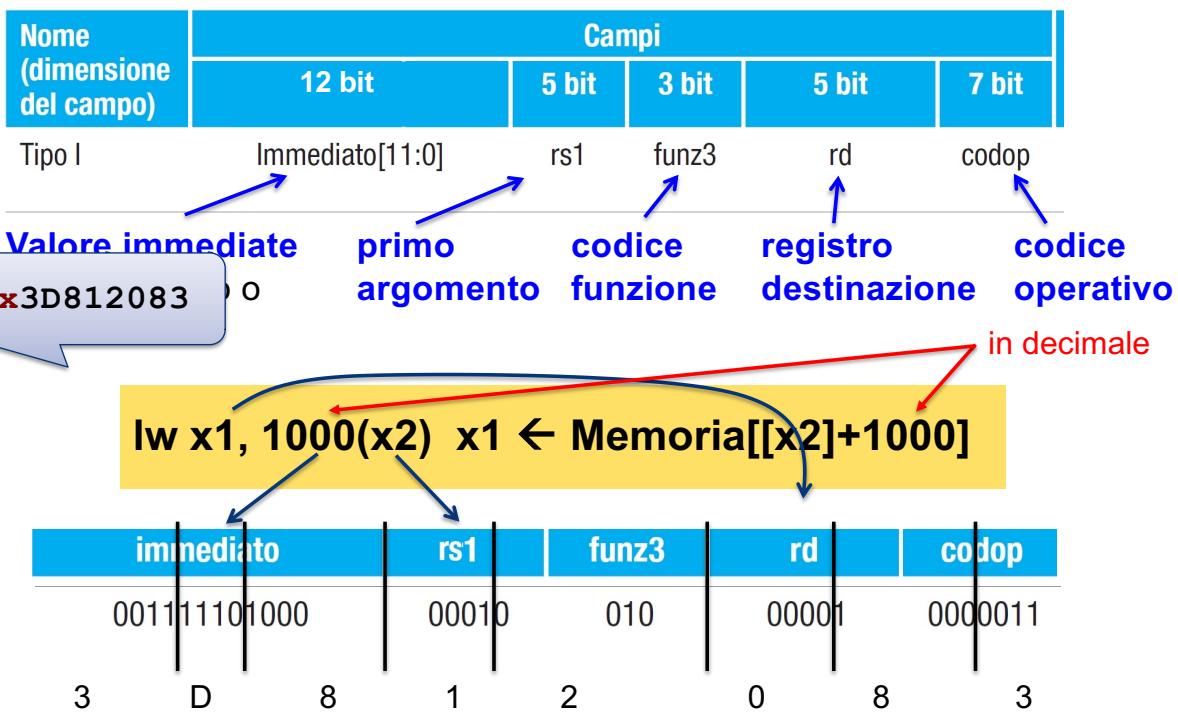
Nome (dimensione del campo)	Campi					
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit
Tipo R	funz7	rs2	rs1	funz3	rd	codop
codice funzione identifica la variante operativa (e.g. somma o sottrazione)		registro che contiene secondo e primo argomento <i>5 bit bastano per indicare quale dei 32 registri</i>		codice funzione identifica la variante di operazione		codice operativo identifica l'operazione aritmetica
					registro destinazione riceve il risultato	

Istruzioni – Formato R

Nome (dimensione del campo)	Campi					
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit
Tipo R	funz7	rs2	rs1	funz3	rd	codop
codice 0x003100B3		secondo e primo argomento		codice funzione	registro destinazione	codice operativo
add x1, x2, x3 x1 ← [x2] + [x3]						
	funz7 0000000	rs2 00011	rs1 00010	funz3 000	rd 00001	codop 0110011
	0	0	3	1	0	3

Istruzioni – Formato I

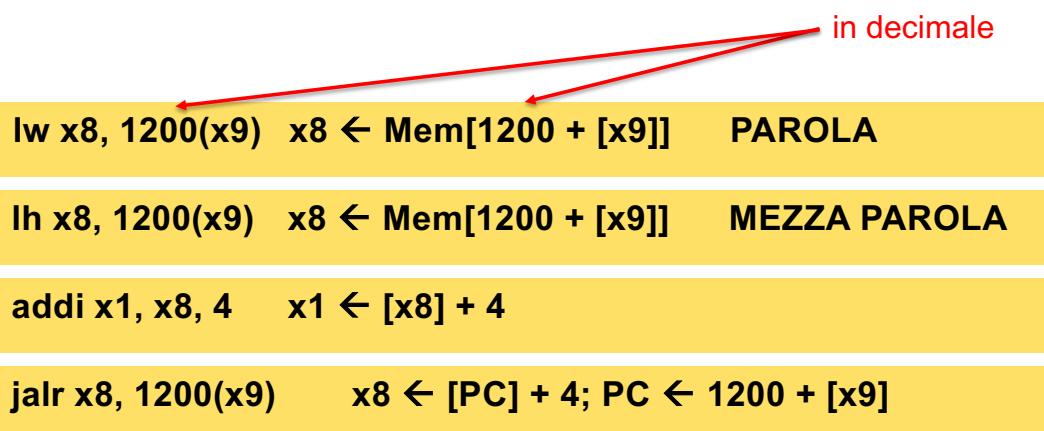
Istruzioni di caricamento dalla memoria e aritmetica con costanti



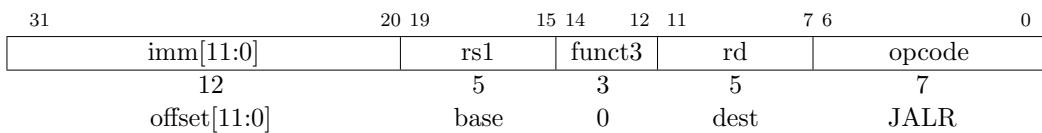
Istruzioni – Formato I

Istruzioni di caricamento dalla memoria e aritmetica con costanti

Istruzioni tipo-I	immediato	rs1	funz3	rd	codop	Esempio
addi (somma immediata)	001111101000	00010	000	00001	0010011	addi x1,x2,1000
lw (load parola)	001111101000	00010	010	00001	0000011	lw x1, 1000 (x2)



Istruzioni – Formato I

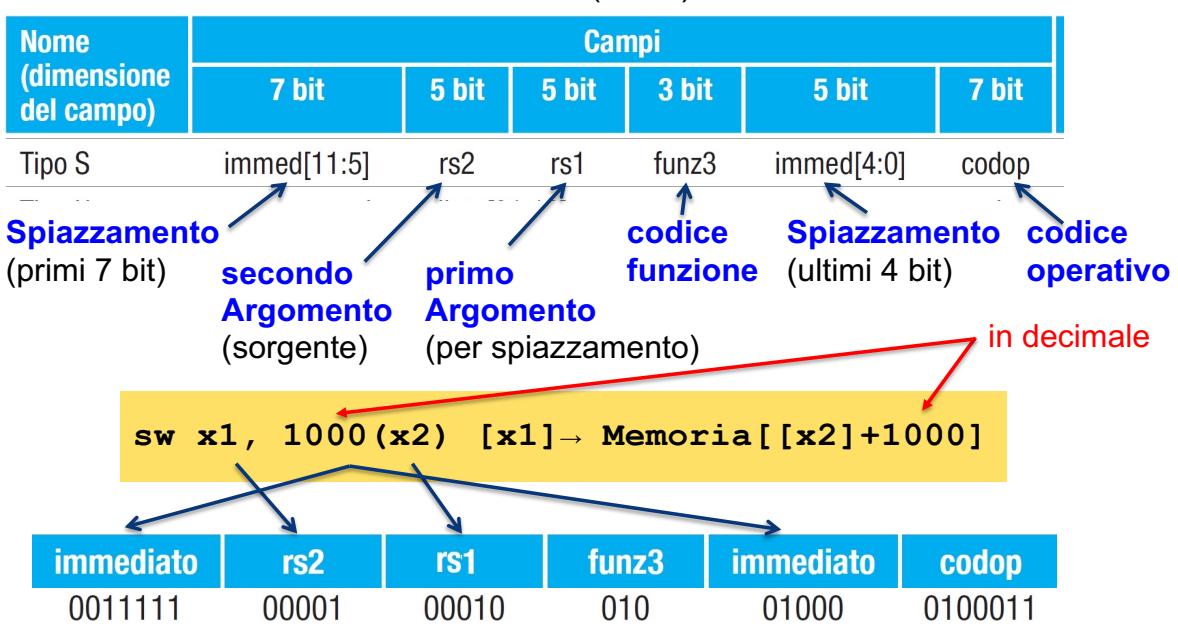


Salto incondizionato:

- L'istruzione di salto JALR (**jump and link register**) utilizza la codifica di tipo I
- L'indirizzo di destinazione si ottiene aggiungendo il valore immediato con segno a 12 bit al registro **rs1**, quindi impostando a zero il bit meno significativo del risultato
- L'indirizzo dell'istruzione successiva al salto (PC + 4) viene scritto nel registro **rd**
- Il registro **x0** può essere utilizzato come destinazione se il risultato non è richiesto

Istruzioni – Formato S

Istruzioni di caricamento in memoria (store)



Istruzioni – Formato S sottoformato B

31	30	25 24	20 19	15 14	12 11	8	7	6	0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]		opcode	
1	6	5	5	3	4	1		7	
offset[12,10:5]		src2	src1	BEQ/BNE		offset[11,4:1]		BRANCH	
offset[12,10:5]		src2	src1	BLT[U]		offset[11,4:1]		BRANCH	
offset[12,10:5]		src2	src1	BGE[U]		offset[11,4:1]		BRANCH	

Salto condizionale:

- Tutte le istruzioni di salto condizionale utilizzano il sottoformato di tipo B
- Il B-immediato a 12 bit codifica gli offset con segno in multipli di 2 e viene aggiunto al PC corrente per fornire l'indirizzo di destinazione (intervallo da -4096 a 4094 rispetto al PC)
- Le istruzioni di salto confrontano due registri
 - BEQ e BNE saltano se i registri **rs1** e **rs2** sono rispettivamente uguali o diversi
 - BLT e BLTU saltano se **rs1** è minore di **rs2**, utilizzando rispettivamente il confronto con segno e senza segno
 - BGE e BGEU saltano se **rs1** è maggiore o uguale a **rs2**, utilizzando rispettivamente il confronto con segno e senza segno

Istruzioni – Formato U

Istruzioni di caricamento in memoria (store)

Nome (dimensione del campo)	Campi		
	20 bit	5 bit	7 bit
Tipo U	immediato[31:12]	rd	codop
Dato immediate (20 bit)	registro destinazione	codice operativo	
31	12 11	7 6	0
imm[31:12]	rd	opcode	
20	5	7	
U-immediate[31:12]	dest	LUI	
U-immediate[31:12]	dest	AUIPC	

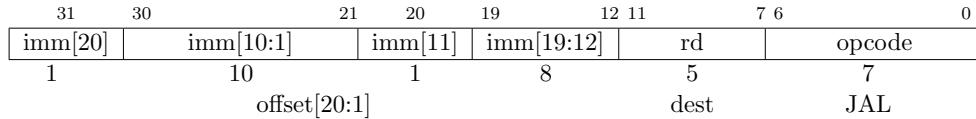
LUI (load upper immediate)

- viene utilizzato per creare costanti a 32 bit
- inserisce il valore immediato U nei primi 20 bit del registro di destinazione **rd**, riempiendo i 12 bit più bassi con zeri

AUIPC (add upper immediate to pc)

- viene utilizzato per creare indirizzi relativi al PC
- forma un offset di 32 bit dall'immediato U di 20 bit, riempiendo i 12 bit più bassi con zeri, aggiunge questo offset al PC, quindi inserisce il risultato nel registro **rd**

Istruzioni – Formato U sottoformato J



Salto incondizionato:

- L'istruzione **jump and link** (JAL) utilizza il formato di tipo UJ, dove l'immediato UJ codifica un offset con segno in multipli di 2 byte
- L'offset viene esteso con segno e aggiunto al PC per formare l'indirizzo di destinazione del salto (intervallo di ± 1 MB rispetto al PC)
- JAL memorizza l'indirizzo dell'istruzione successiva al salto (PC + 4) nel registro **rd**
- La convenzione di chiamata del software standard utilizza **x1** come registro dell'indirizzo di ritorno e **x5** come registro di collegamento alternativo

Istruzioni RISC-V

Operandi RISC-V				
Nome	Esempio	Commenti		
32 registri	x0-x31	Accesso veloce ai dati. Nel RISC-V gli operandi devono essere contenuti nei registri per potere eseguire delle operazioni. Il registro x0 contiene sempre il valore 0		
2 ¹⁰ parole di memoria	Memoria[0], Memoria[4], ..., Memoria[4 294 967 292]	Alla memoria si accede solamente attraverso istruzioni di trasferimento dati. Il RISC-V utilizza l'indirizzamento al byte, perciò due variabili successive hanno indirizzi in memoria a distanza 4. La memoria consente di memorizzare strutture dati, vettori, o il contenuto dei registri		
Linguaggio assembler RISC-V				
Tipo di istruzioni	Istruzioni	Esempio	Significato	Commenti
Aritmetiche	Somma	add x5, x6, x7	x5 = x6 + x7	Operandi in tre registri
	Sottrazione	sub x5, x6, x7	x5 = x6 - x7	Operandi in tre registri
	Somma immediata	addi x5, x6, 20	x5 = x6 + 20	Utilizzata per sommare delle costanti
Trasferimento dati	Lettura parola	lw x5, 40(x6)	x5 = Memoria[x6 + 40]	Spostamento di una parola da memoria a registro
	Lettura parola, senza segno	lwu x5, 40(x6)	x5 = Memoria[x6+40]	Spostamento di una parola da memoria a registro senza segno
	Memorizzazione parola	sw x5, 40(x6)	Memoria[x6+40] = x5	Spostamento di una parola da registro a memoria
	Lettura mezza parola	lh x5, 40(x6)	x5 = Memoria[x6+40]	Spostamento di una mezza parola da memoria a registro
	Lettura mezza parola, senza segno	lhu x5, 40(x6)	x5 = Memoria[x6+40]	Spostamento di una mezza parola senza segno da memoria a registro
	Memorizzazione mezza parola	sh x5, 40(x6)	Memoria[x6+40] = x5	Spostamento di una mezza parola da registro a memoria
	Lettura byte	lb x5, 40(x6)	x5 = Memoria[x6+40]	Spostamento di un byte da memoria a registro
	Lettura byte, senza segno	lbu x5, 40(x6)	x5 = Memoria[x6+40]	Spostamento di un byte senza segno da memoria a registro
	Memorizzazione byte	sb x5, 40(x6)	Memoria[x6+40] = x5	Spostamento di un byte da registro a memoria
	Lettura di una parola e blocco	lr.d x5, (x6)	x5 = Memoria[x6]	Caricamento di una parola come prima fase di un'operazione atomica sulla memoria
Logiche	Memorizzazione condizionata di una parola	sc.d x7, x5, (x6)	Memoria[x6] = x5; x7 = 0/1	Memorizzazione di una parola come seconda fase di un'operazione atomica sulla memoria
	Caricamento costante nella mezza parola superiore	lui x5, 0x12345	x5 = 0x12345000	Caricamento di una costante su 20 bit nei 12 bit più significativi di una parola
	And	and x5, x6, x7	x5 = x6 & x7	Operandi in tre registri; AND bit a bit
	Or inclusivo	or x5, x6, x8	x5 = x6 x8	Operandi in tre registri; OR bit a bit
	Or esclusivo (Xor)	xor x5, x6, x9	x5 = x6 ^ x9	Operandi in tre registri; XOR bit a bit
	And immediato	andi x5, x6, 20	x5 = x6 & 20	AND bit a bit tra un operando in registro e una costante

Figura 2.1 Il linguaggio assembler del RISC-V esaminato in questo capitolo. Si possono trovare queste informazioni anche nella prima colonna della scheda tecnica riassuntiva del RISC-V in fondo al libro. (continua)

Tipo di istruzioni	Istruzioni	Esempio	Significato	Commenti
Logiche (continua)	Or esclusivo immediato	ori x5, x6, 20	x5 = x6 20	OR bit a bit tra un operando in registro e una costante
	Xor immediato	xori x5, x6, 20	x5 = x6 ^ 20	XOR bit a bit tra un operando in registro e una costante
Scorrimento (shift)	Scorrimento logico a sinistra	sll x5, x6, x7	x5 = x6 << x7	Scorrimento a sinistra mediante registro
	Scorrimento logico a destra	srl x5, x6, x7	x5 = x6 >> x7	Scorrimento a destra mediante registro
	Scorrimento aritmetico a destra	sra x5, x6, x7	x5 = x6 >> x7	Scorrimento a destra aritmetico mediante registro
	Scorrimento logico a sinistra immediato	slli x5, x6, 3	x5 = x6 << 3	Scorrimento a sinistra mediante costante
	Scorrimento logico a destra immediato	srali x5, x6, 3	x5 = x6 >> 3	Scorrimento a destra mediante costante
	Scorrimento aritmetico a destra immediato	srai x5, x6, 3	x5 = x6 >> 3	Scorrimento a destra aritmetico mediante costante
Salti condizionati	Salta se uguale	beq x5, x6, 100	Se (x5 == x6) vai a PC+100	Test di uguaglianza; salto relativo al PC
	Salta se non è uguale	bne x5, x6, 100	Se (x5 != x6) vai a PC+100	Test di disuguaglianza; salto relativo al PC
	Salta se minore di	blt x5, x6, 100	Se (x5 < x6) vai a PC+100	Comparazione di minoranza; salto relativo al PC
	Salta se maggiore o uguale di	bge x5, x6, 100	Se (x5 >= x6) vai a PC+100	Comparazione di maggioranza o uguaglianza; salto relativo al PC
	Salta se minore di, senza segno	bltu x5, x6, 100	Se (x5 < x6) vai a PC+100	Comparazione di minoranza senza segno; salto relativo al PC
	Salta se maggiore o uguale di, senza segno	bgeu x5, x6, 100	Se (x5 >= x6) vai a PC+100	Comparazione di maggioranza o uguaglianza senza segno; salto relativo al PC
Salti incondizionati	Salta e collega	jal x1, 100	x1 = PC+4; vai a PC+100	Chiamata a procedura con indirizzamento relativo al PC
	Salta e collega mediante registro	jalr x1, 100(x5)	x1 = PC+4; vai a x5+100	Ritorno da procedura; chiamata indiretta

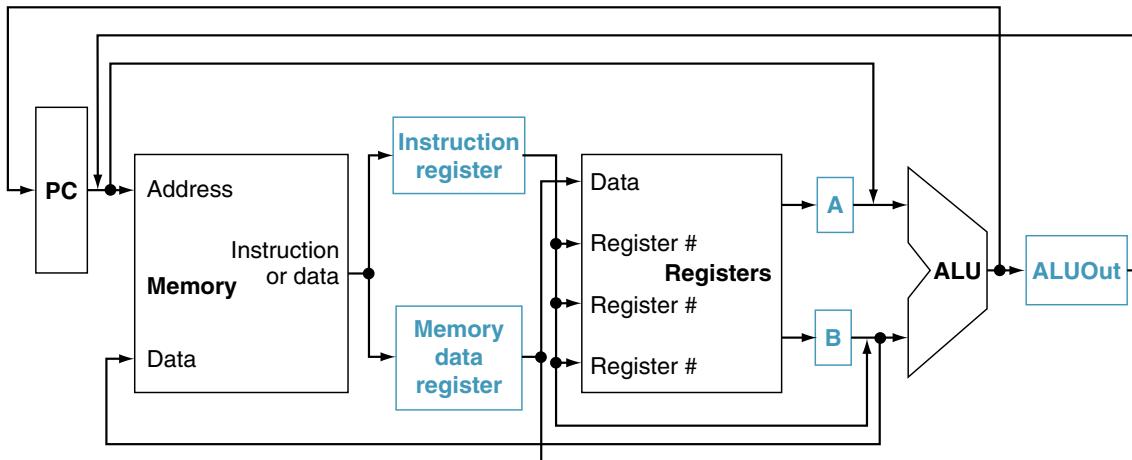
Figura 2.1 Il linguaggio assembler del RISC-V esaminato in questo capitolo. (continua)

Ciclo esecutivo di un'istruzione RISC-V

- IF** Instruction Fetch
- ID** Instruction Decode / register fetch
- EX** Execution / address calculation
tutte le istruzioni usano la ALU:
 - operazioni logico-aritmetiche
 - load/store e jump per calcolare l'indirizzo
 - salti condizionati per calcolare la condizione
- MEM** Memory access / branch completion
- WB** Write Back: scrittura del risultato nei registri

Nome (dimensione del campo)	Campi					
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit
Tipo R	funz7	rs2	rs1	funz3	rd	codop
Tipo I	Immediato[11:0]		rs1	funz3	rd	codop
Tipo S	immed[11:5]	rs2	rs1	funz3	immed[4:0]	codop
Tipo U	immediato[31:12]				rd	codop

Datapath



Elementi chiave del datapath:

- un'unità di memoria condivisa, una singola ALU condivisa tra le istruzioni e le connessioni tra queste unità condivise
- L'uso di unità funzionali condivise richiede l'aggiunta o l'ampliamento di multiplexor nonché di nuovi registri temporanei che trattengono i dati tra i cicli di clock della stessa istruzione
- I registri aggiuntivi sono il registro delle istruzioni (IR), il registro dei dati di memoria (MDR), A, B e ALUOut

Ciclo esecutivo di un'istruzione RISC-V

1. IF Instruction Fetch

- $IR \leftarrow Mem[PC]$ (*IR = instruction register, PC = program counter*)
- $PC, NPC \leftarrow PC+4$ (*NPC è un registro temporaneo*) 1 word = 4 byte

2. ID Instruction Decode / register fetch

- **Formato R** $A \leftarrow Regs[rs1]; B \leftarrow Regs[rs2]$
- **Formato I** $A \leftarrow Regs[rs1]; Imm \leftarrow$ campo immediato di IR
- **Formato U** $Imm \leftarrow$ campo immediato di IR;
 - il campo immediato di IR va esteso a 32 bit, estendendo il segno
 - A, B, Imm: registri temporanei

Nome (dimensione del campo)	Campi					
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit
Tipo R	funz7	rs2	rs1	funz3	rd	codop
Tipo I	Immediato[11:0]		rs1	funz3	rd	codop
Tipo S	immed[11:5]	rs2	rs1	funz3	immed[4:0]	codop
Tipo U		immediato[31:12]			rd	codop

Ciclo esecutivo di un'istruzione RISC-V

3. EX Execute / address calculation

- **Riferimento a memoria**
 - **Iw x8, 1200(x9)** Formato I A = $\text{Regs}[rs1] = [x9]$; Imm = 1200 (esteso)
 - $\text{ALUOutput} \leftarrow A + \text{Imm}$ //address calculation
- **Istruzione ALU registro-registro**
 - **add x8, x18, x8** Formato R A = $\text{Regs}[rs1] = [x18]$; B = $\text{Regs}[rs2] = [x8]$
 - $\text{ALUOutput} \leftarrow A \text{ funct } B$
- **Istruzione ALU registro-immediato**
 - **addi x8, x18, 4** Formato I A = $\text{Regs}[rs1] = [x18]$; Imm = 4 (esteso)
 - $\text{ALUOutput} \leftarrow A \text{ op } \text{Imm}$
- **Salto**
 - **jal x0 45054** Formato U
 - **beq x1, x2, 16** Formato I
(prossima slide)

Nome (dimensione del campo)	Campi					
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit
Tipo R	funz7	rs2	rs1	funz3	rd	codop
Tipo I	Immediato[11:0]		rs1	funz3	rd	codop
Tipo S	immed[11:5]	rs2	rs1	funz3	immed[4:0]	codop
Tipo U		immediato[31:12]			rd	codop

Ciclo esecutivo di un'istruzione RISC-V

3. EX Execute / address calculation

- **Salto incondizionato**
 - **jal x5 45054** Formato UJ Imm = 45054
 - il vecchio PC+4 viene salvato in **x5**
 - il numero di parole va trasformato in numero di byte: moltiplicando per 4 cioè **due shift a sinistra <<2**
 - viene poi sommato al PC
 - $\text{Target} \leftarrow [\text{PC}] + \text{Imm} \ll 2$
- **Salto condizionato**
 - **beq x1, x2, 16** Formato I A = $\text{Regs}[rs1] = [x1]$; B = $\text{Regs}[rs2] = [x2]$; Imm = 16
 - l'indirizzo nell'istruzione si riferisce al numero di **parole** di cui far avanzare il PC (NPC), ma gli indirizzi RISC-V sono al **byte**
 - deve **valutare la condizione e calcolare l'indirizzo** di salto
 - $\text{Cond} \leftarrow (\text{A} == \text{B})$ (equiv. $(\text{A}-\text{B}) == 0$) la ALU fornisce in uscita un segnale che indica se il risultato è 0
 - $\text{Target} \leftarrow [\text{NPC}] + (\text{Imm} \ll 2)$

Nome (dimensione del campo)	Campi					
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit
Tipo R	funz7	rs2	rs1	funz3	rd	codop
Tipo I	Immediato[11:0]		rs1	funz3	rd	codop
Tipo S	immed[11:5]	rs2	rs1	funz3	immed[4:0]	codop
Tipo U		immediato[31:12]			rd	codop

Ciclo esecutivo di un'istruzione RISC-V

4. MEM Memory access / branch completion

- Riferimento a memoria**

- **Iw x8, 1200(x9)** A = [x9]; B = [x8]; Imm = 1200; ALUOutput = A+Imm
- **LMD \leftarrow Mem[ALUOutput]** (Load Memory Data register)
- **sw x8, 1200(x9)** A = [x9]; B = [x8]; Imm = 1200; ALUOutput = A+Imm
- **Mem[ALUOutput] \leftarrow [B]**

Nome (Dimensione campo)	Campo							Commenti
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit		
Tipo R	funz7	rs2	rs1	funz3	rd	codop		Formato istruzioni aritmetiche
Tipo I	imm[11:0]		rs1	funz3	rd	codop		Load e aritmetiche con costante
Tipo S	imm[11:5]	rs2	rs1	funz3	imm[4:0]	codop		Store
Tipo SB	imm[12,10:5]	rs2	rs1	funz3	imm[4:1,11]	codop		Salvi cond
Tipo UJ	imm[20, 10:1, 11, 19:12]				rd	codop		Formato salti incond
Tipo U	imm[31:12]				rd	codop		Formato upper immed

- Salto**

- **incondizionato PC \leftarrow [PC] + [Target]**
- **condizionato if (Cond) PC \leftarrow [PC] + [Target]**

Ciclo esecutivo di un'istruzione RISC-V

5. WB Write back: scrittura nei registri

- Riferimento a memoria (solo load)**

- **Iw x8, 1200(x9)** A = [x9]; B = Regs[rs1] = [x8]; LMD \leftarrow Mem[ALUOutput]
- **x8 \leftarrow [LMD]**

- Istruzione ALU registro-registro**

- **add x8, x18, x8** Formato R A = Regs[rs1] = [x18]; B = Regs[rs2] = [x8]
- **x8 \leftarrow [ALUOutput]**

- Istruzione ALU registro-immediato**

- **addi x8, x18, 4** Formato I A = Regs[rs1] = [x18]; B = Regs[rd] = [x8]; Imm = 4
- **x8 \leftarrow [ALUOutput]**

Nome (Dimensione campo)	Campo							Commenti
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit		
Tipo R	funz7	rs2	rs1	funz3	rd	codop		Formato istruzioni aritmetiche
Tipo I	imm[11:0]		rs1	funz3	rd	codop		Load e aritmetiche con costante
Tipo S	imm[11:5]	rs2	rs1	funz3	imm[4:0]	codop		Store
Tipo SB	imm[12,10:5]	rs2	rs1	funz3	imm[4:1,11]	codop		Salvi cond
Tipo UJ	imm[20, 10:1, 11, 19:12]				rd	codop		Formato salti incond
Tipo U	imm[31:12]				rd	codop		Formato upper immed

Ciclo esecutivo di un'istruzione RISC-V

IF

- $IR \leftarrow \text{Mem}[PC]$
- $PC, NPC \leftarrow [PC]+4$

add x8, x18, x8

Nome (Dimensione campo)	Campo						Commenti
Tipo R	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit	
	funz7	rs2	rs1	funz3	rd	codop	Formato istruzioni aritmetiche

ID

- $A \leftarrow \text{Regs}[rs1] = [x18]$
- $B \leftarrow \text{Regs}[rs2] = [x8]$

EX

- $\text{ALUOutput} \leftarrow [A] \text{ funct } [B]$

MEM

- nessuna operazione

WB

- $\text{Regs}[rd] = x8 \leftarrow [\text{ALUOutput}]$

Ciclo esecutivo di un'istruzione RISC-V

lw x8, 1200(x9)

Nome (Dimensione campo)	Campo						Commenti
Tipo I	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit	
	imm[11:0]	rs1	funz3	rd	codop		Load e aritmetiche con costante

IF

- $IR \leftarrow \text{Mem}[PC]$
- $PC, NPC \leftarrow [PC]+4$

ID

- $A \leftarrow \text{Regs}[rs1] = [x9]$
- $B \leftarrow \text{Regs}[rs2] = [x8]$
- $\text{Imm} \leftarrow \text{campo immediato di IR} = 1200 \text{ (esteso)}$

EX

- $\text{ALUOutput} \leftarrow [A] + [\text{Imm}]$

MEM

- $\text{LMD} \leftarrow \text{Mem}[\text{ALUOutput}]$

WB

- $\text{Regs}[rd] = x8 \leftarrow [\text{LMD}]$

Ciclo esecutivo di un'istruzione RISC-V

IF

- $IR \leftarrow \text{Mem}[PC]$
- $PC, NPC \leftarrow [PC]+4$

ID

- $A \leftarrow \text{Regs}[rs1] = [x9]$
- $B \leftarrow \text{Regs}[rs2] = [x8]$
- Imm \leftarrow campo immediato di IR = 1200 (esteso)

sw x8, 1200(x9)

Nome (Dimensione campo)	Campo						Commenti
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit	
Tipo S	imm[11:5]	rs2	rs1	funz3	imm[4:0]	codop	Store

EX

- $ALUOutput \leftarrow [A] + [Imm]$

MEM

- $\text{Mem}[ALUOutput] \leftarrow [B]$

WB

- nessuna operazione

Ciclo esecutivo di un'istruzione RISC-V

beq x1,x2,16

Nome (Dimensione campo)	Campo						Commenti
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit	
Tipo SB	imm[12,10:5]	rs2	rs1	funz3	imm[4:1,11]	codop	Salti cond

...non è nella fase MEM di beq, ma nella fase IF che avviene, al ciclo successivo a quello di EX di beq

IF

- $IR \leftarrow \text{Mem}[[PC]$
- $PC, NPC \leftarrow [PC]+4$

ID

- $A \leftarrow \text{Regs}[rs1] = [x1]$
- $B \leftarrow \text{Regs}[rs2] = [x2]$
- Imm \leftarrow campo immediato di IR = 16 (esteso)

EXE

- Cond $\leftarrow ([A] - [B]) == 0$
- Target $\leftarrow [NPC] + ([Imm] << 2)$

MEM

- if (Cond) $PC \leftarrow [\text{Target}]$

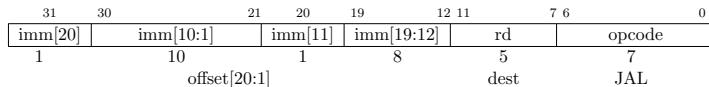
WB

- nessuna operazione

Ciclo esecutivo di un'istruzione RISC-V

x0 non può essere scritto:
indica risultato non richiesto

jal x0 45054



Formato UJ

...non è nella fase MEM di jal,
ma nella fase IF che avviene, al
ciclo successivo a quello di EX
di jal

IF

- $IR \leftarrow \text{Mem}[PC]$
- $PC, NPC \leftarrow [PC]+4$

ID

- $\text{Imm} \leftarrow \text{campo immediato di } IR = 45054 \text{ (esteso)}$

EXE

- $\text{Target} \leftarrow [NPC] + ([\text{Imm}] \ll 2)$

MEM

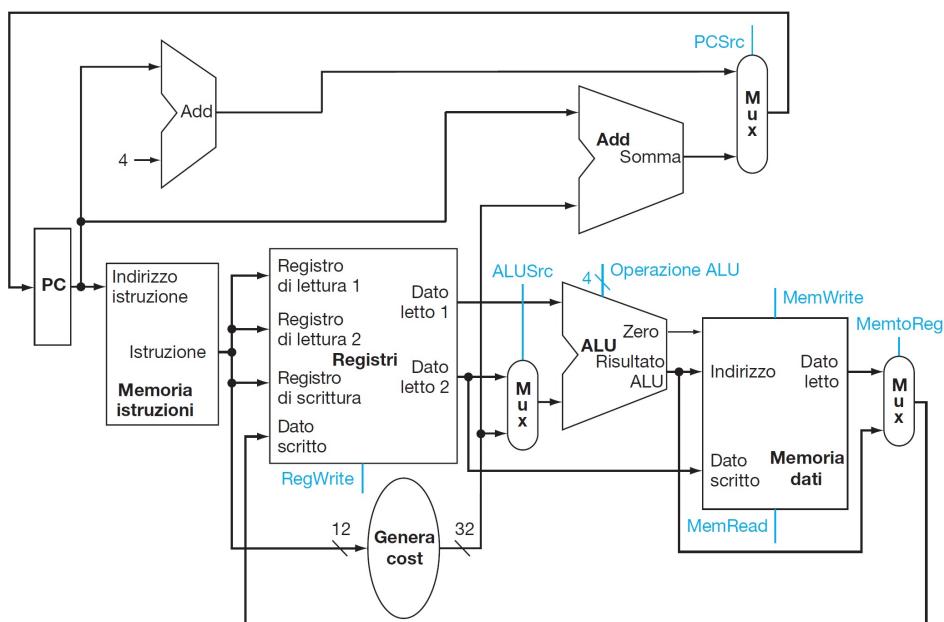
- $PC \leftarrow [\text{Target}]$

WB

- $\text{Regs}[rd] = x0 \leftarrow [NPC]$

Schema di implementazione di RISC-V:

- diverse unità funzionali (es. banco di registri, ALU, memoria...)
- loro connessioni
- mancano l'unità di controllo e le linee di controllo



- Unità di elaborazione del nucleo RISC-V per le istruzioni appartenenti ai diversi tipi
- Questo datapath può eseguire le istruzioni di base (registro load-store, operazioni ALU e salti condizionati) in un singolo ciclo di clock
- Per integrare i salti condizionali è necessario solo un multiplexer aggiuntivo

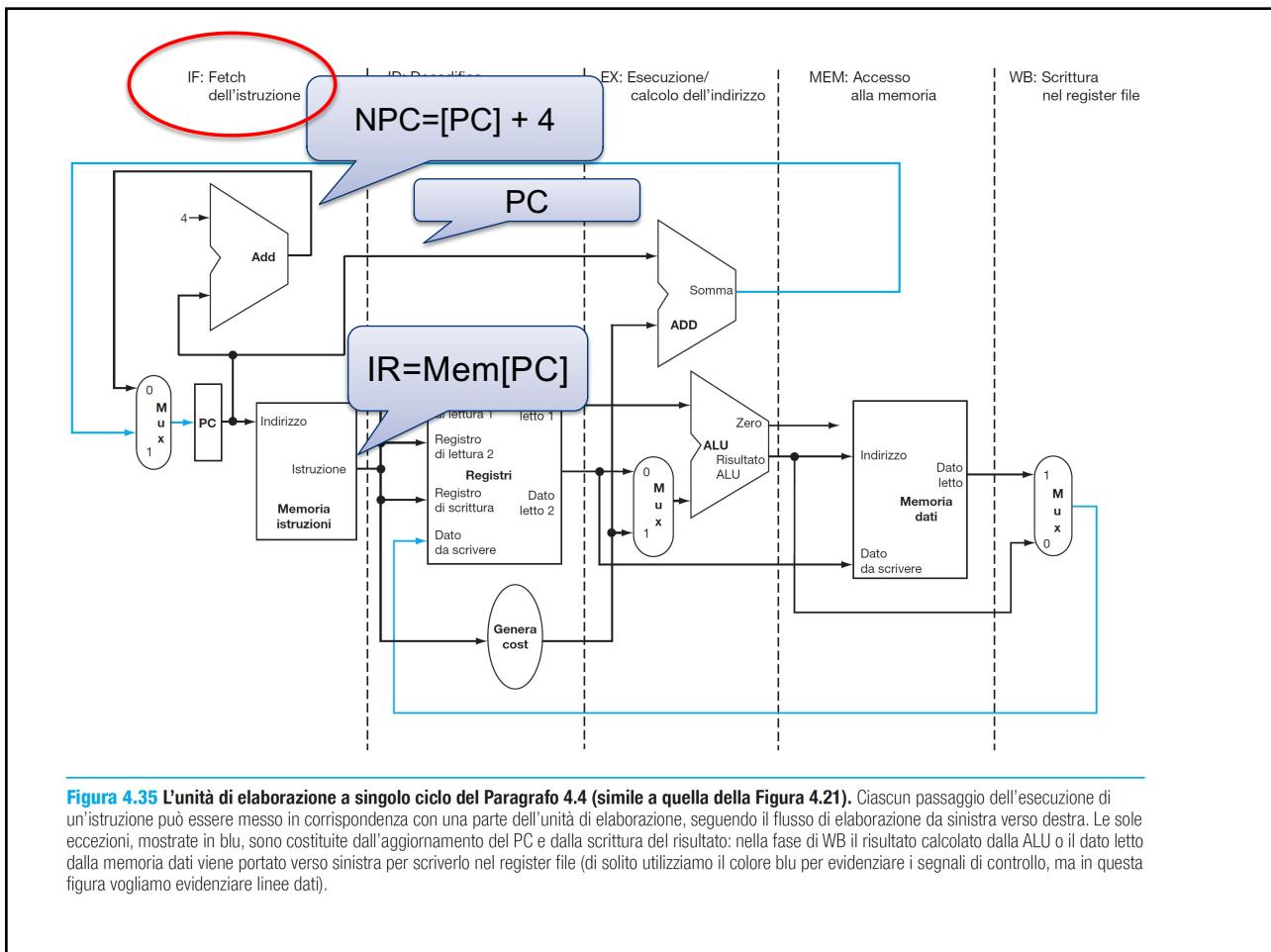


Figura 4.35 L'unità di elaborazione a singolo ciclo del Paragrafo 4.4 (simile a quella della Figura 4.21). Ciascun passaggio dell'esecuzione di un'istruzione può essere messo in corrispondenza con una parte dell'unità di elaborazione, seguendo il flusso di elaborazione da sinistra verso destra. Le sole eccezioni, mostrate in blu, sono costituite dall'aggiornamento del PC e dalla scrittura del risultato: nella fase di WB il risultato calcolato dalla ALU o il dato letto dalla memoria dati viene portato verso sinistra per scriverlo nel register file (di solito utilizziamo il colore blu per evidenziare i segnali di controllo, ma in questa figura vogliamo evidenziare linee dati).

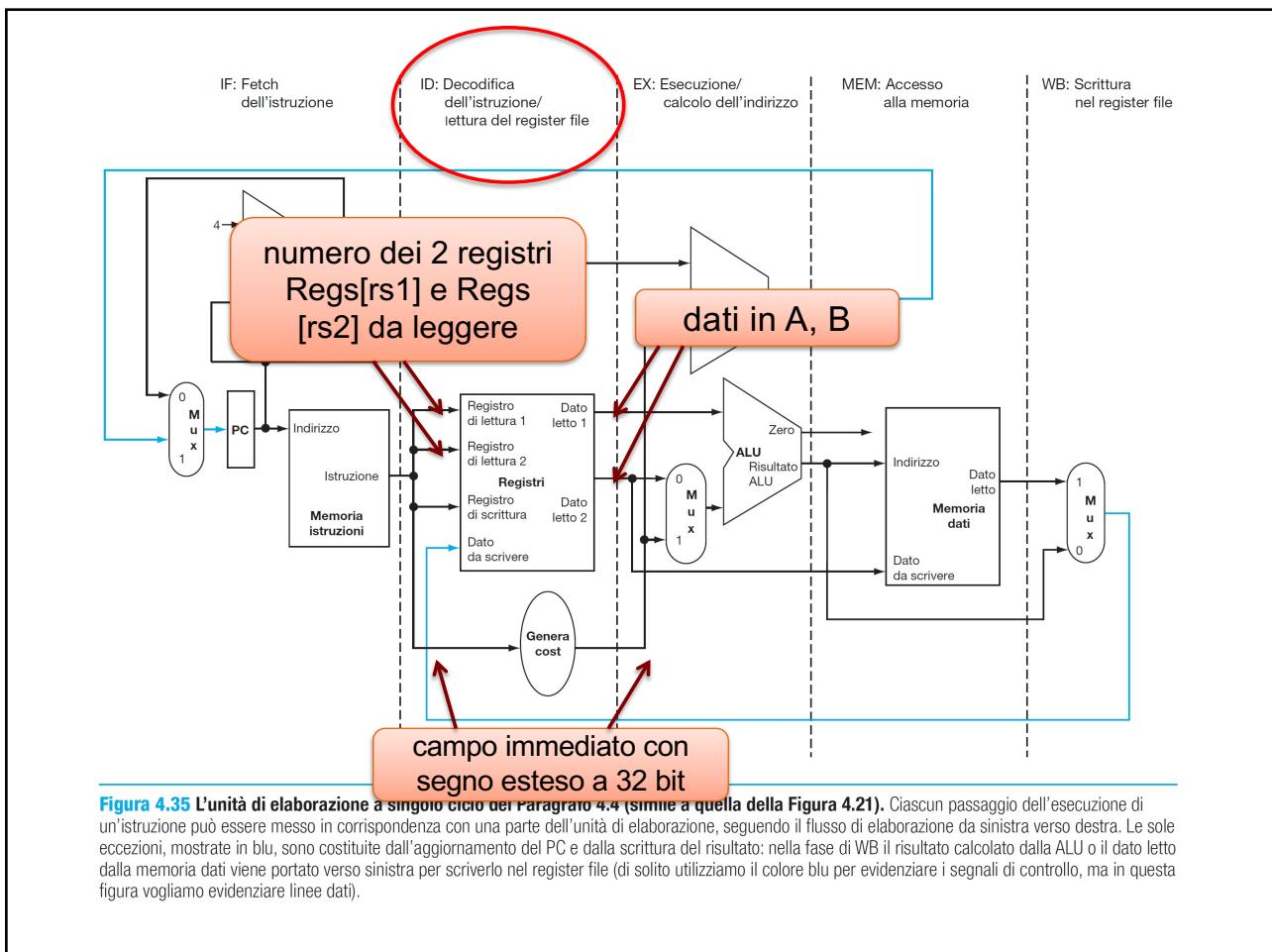


Figura 4.35 L'unità di elaborazione a singolo ciclo del Paragrafo 4.4 (simile a quella della Figura 4.21). Ciascun passaggio dell'esecuzione di un'istruzione può essere messo in corrispondenza con una parte dell'unità di elaborazione, seguendo il flusso di elaborazione da sinistra verso destra. Le sole eccezioni, mostrate in blu, sono costituite dall'aggiornamento del PC e dalla scrittura del risultato: nella fase di WB il risultato calcolato dalla ALU o il dato letto dalla memoria dati viene portato verso sinistra per scriverlo nel register file (di solito utilizziamo il colore blu per evidenziare i segnali di controllo, ma in questa figura vogliamo evidenziare linee dati).

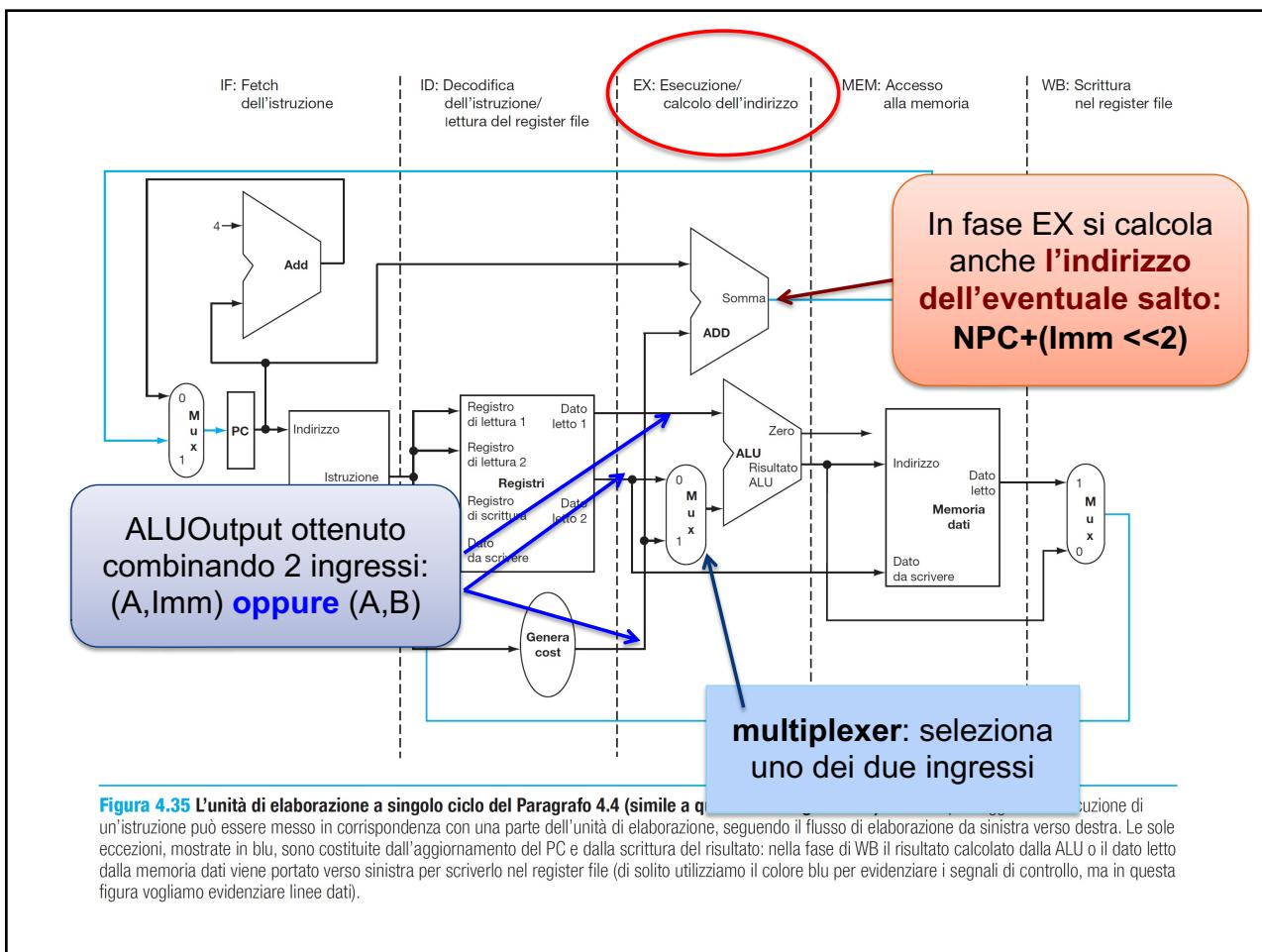


Figura 4.35 L'unità di elaborazione a singolo ciclo del Paragrafo 4.4 (simile a quella di Figura 4.34). Il diagramma illustra il flusso di elaborazione di un'istruzione, suddiviso in quattro fasi principali: IF (Fetch dell'istruzione), ID (Decodifica dell'istruzione/lettura del register file), EX (Esecuzione/calcolo dell'indirizzo) e WB (Scrittura nel register file). La fase EX è evidenziata con un cerchio rosso. Un'annotazione arancione specifica che nell'EX si calcola l'indirizzo di salto: $NPC+(Imm <<2)$. Altri elementi evidenziati in blu sono il PC (aggiornato nell'IF), i registri (letti nell'ID e scritti nell'EX/WB), e i multiplexer (selezionatori) che gestiscono i dati d'ingresso per l'ALU. I dati vengono letti da registri (lettura 1 e lettura 2) e scritti nei registri (scrittura 1 e scrittura 2). L'ALU esegue operazioni di somma (ADD) o confronto (Zero). I risultati vengono memorizzati in registri o inviati alla memoria (MEM) per la scrittura.

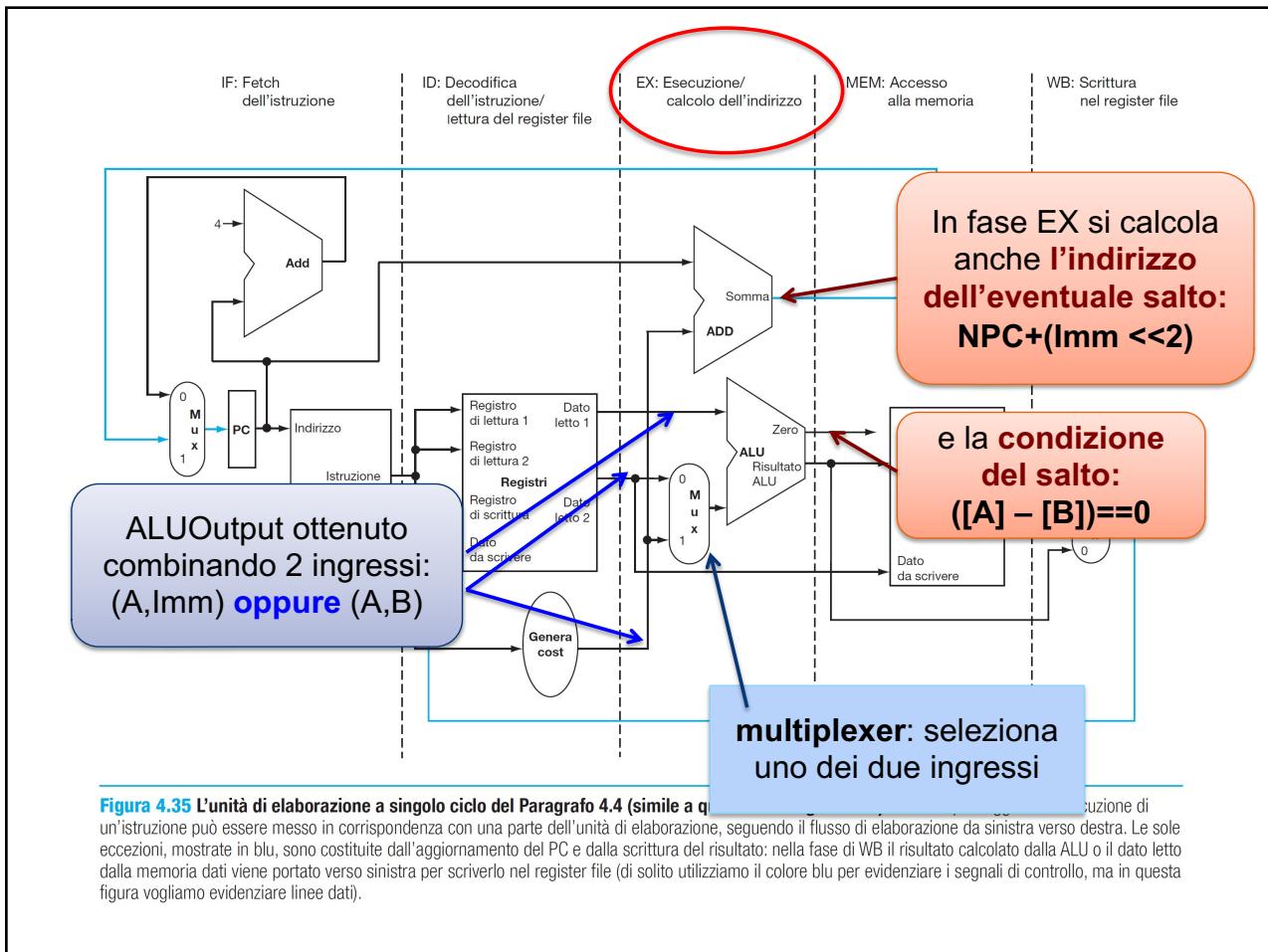
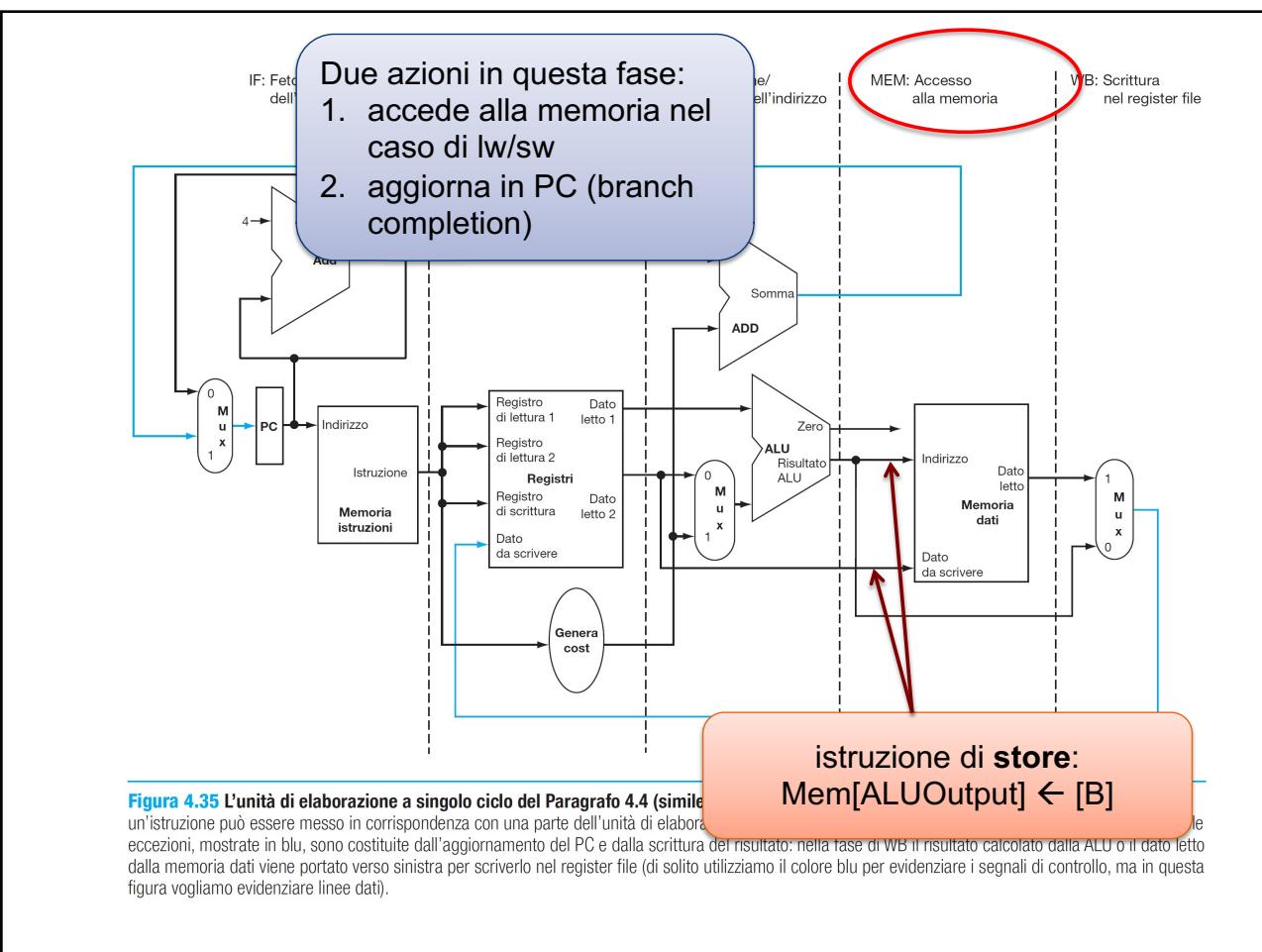
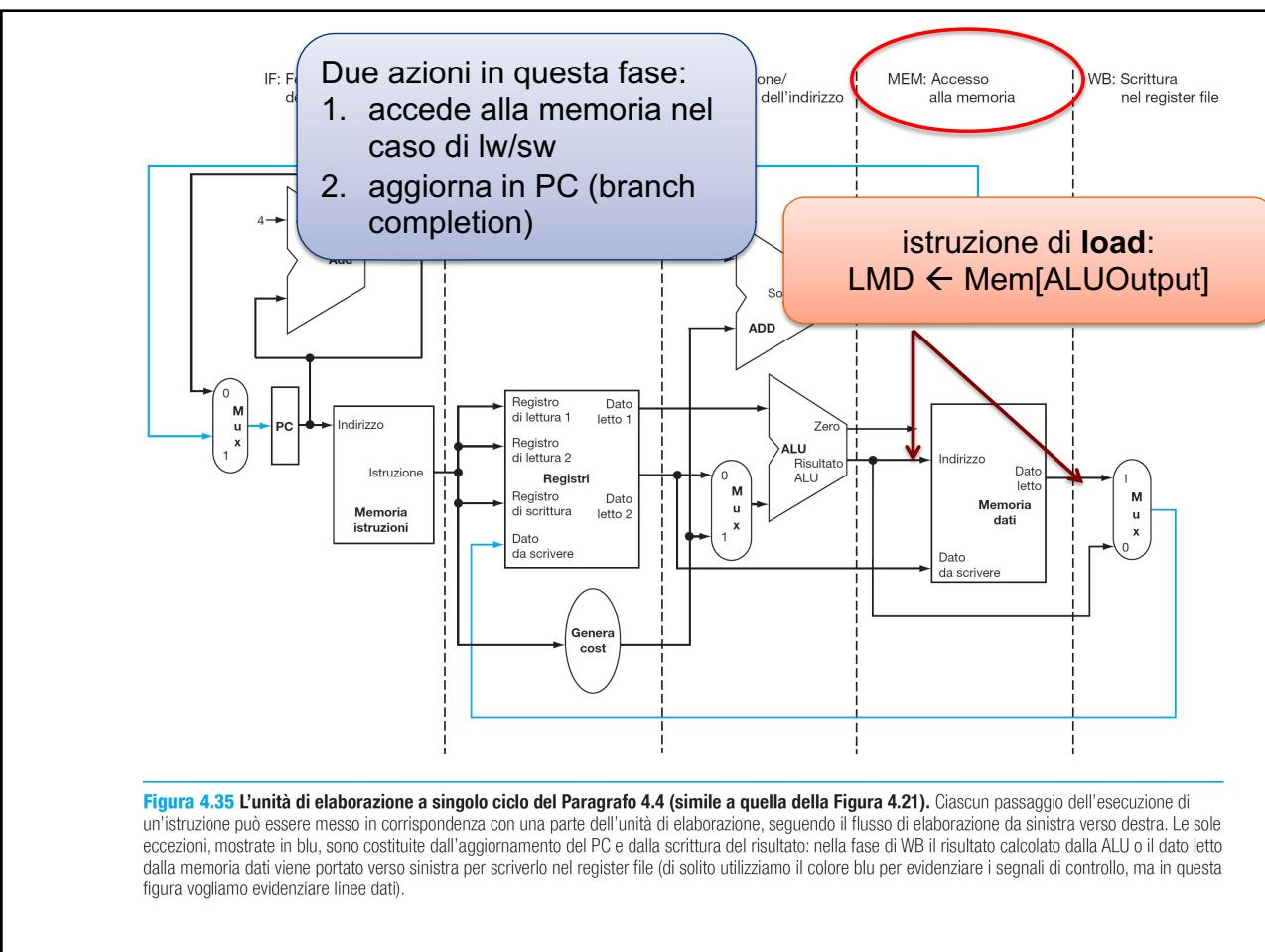
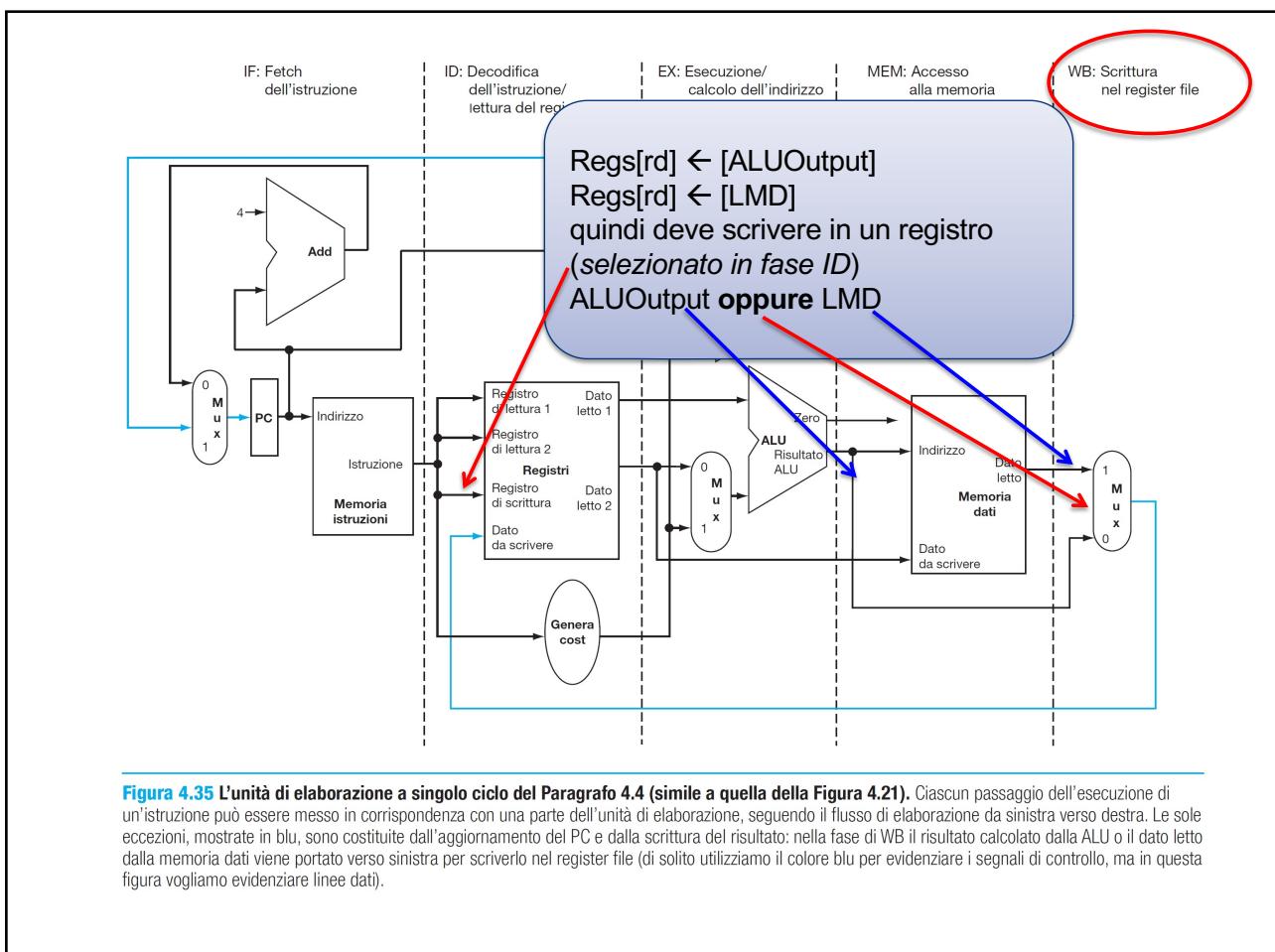
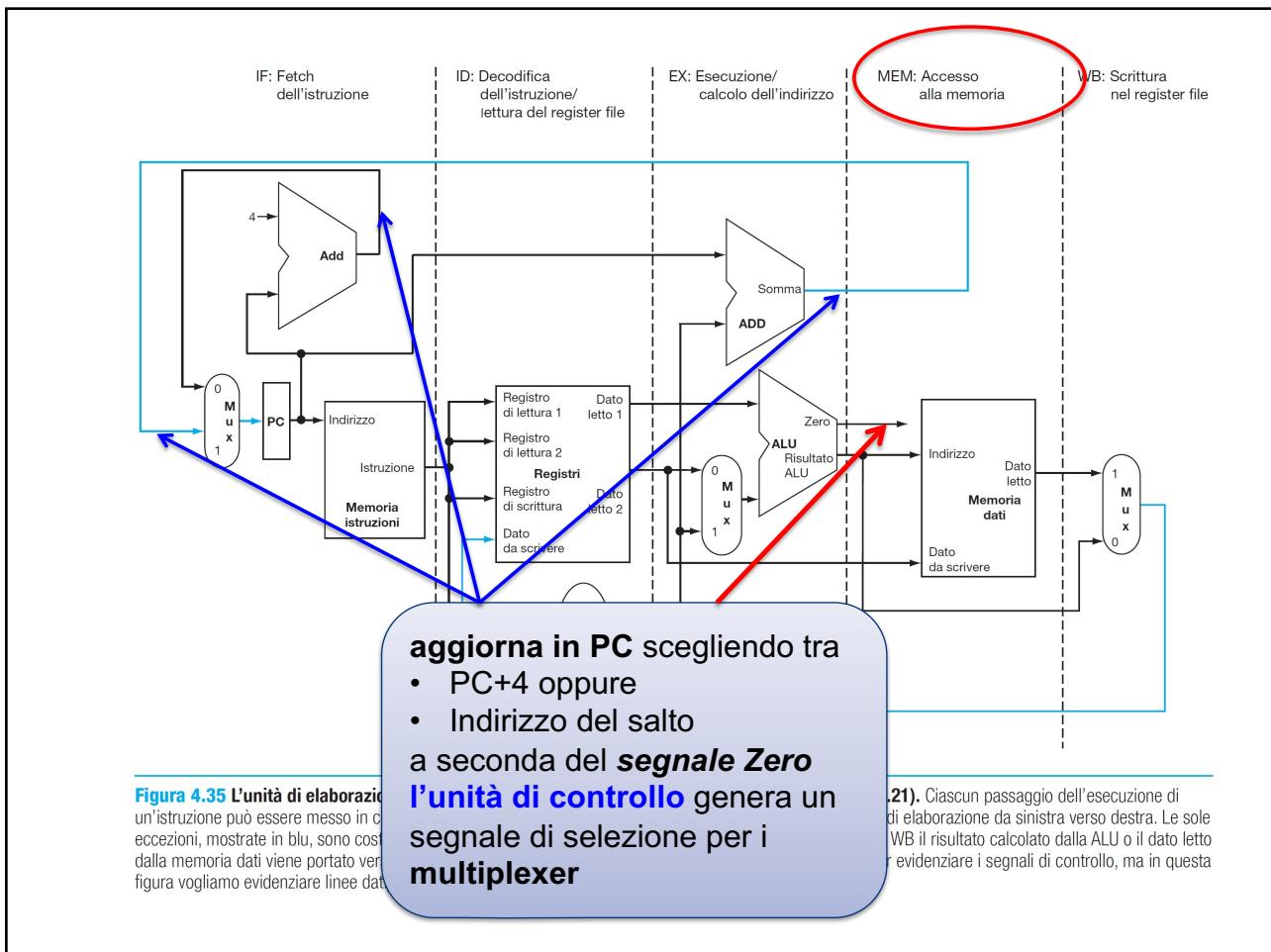
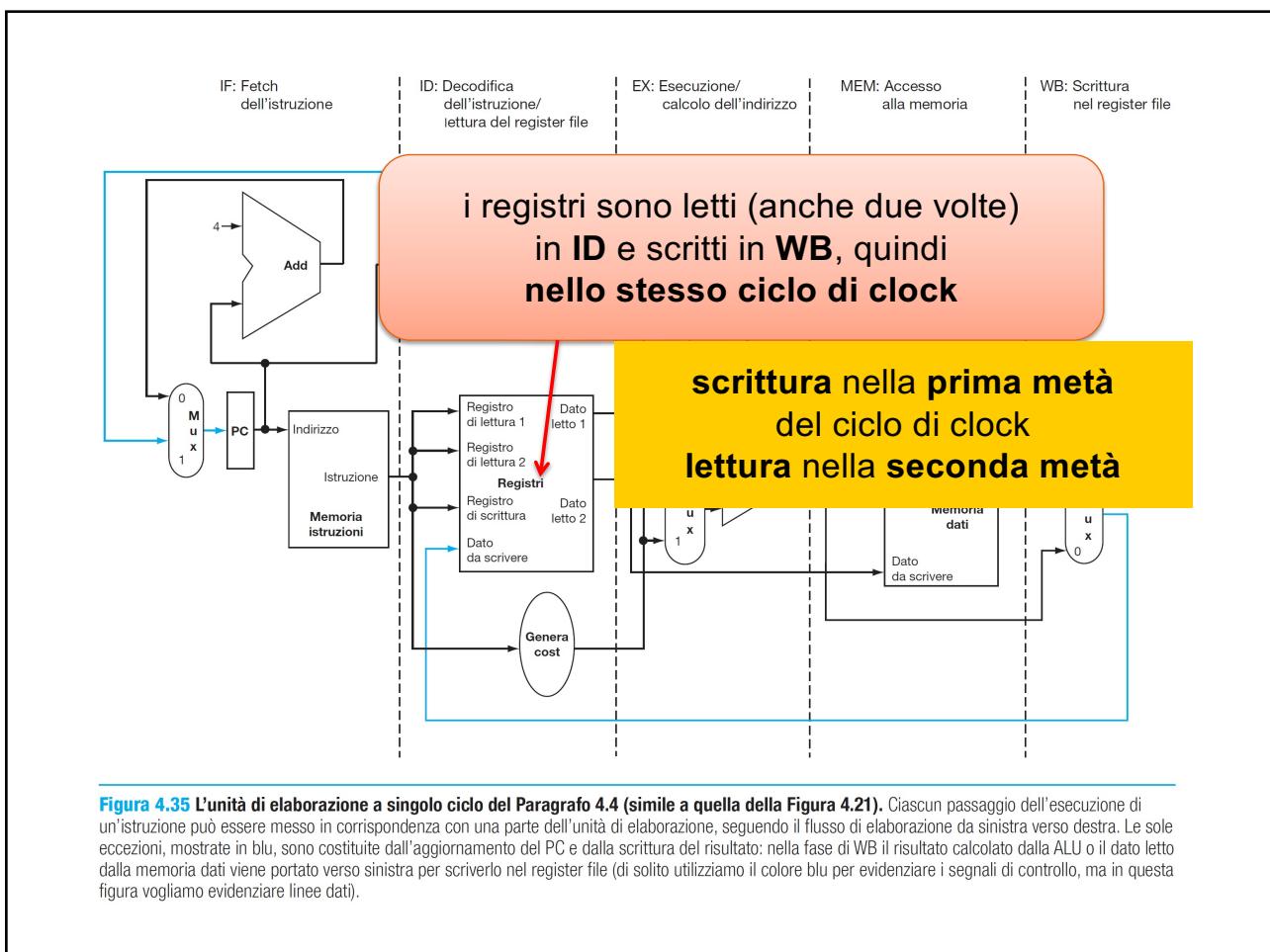
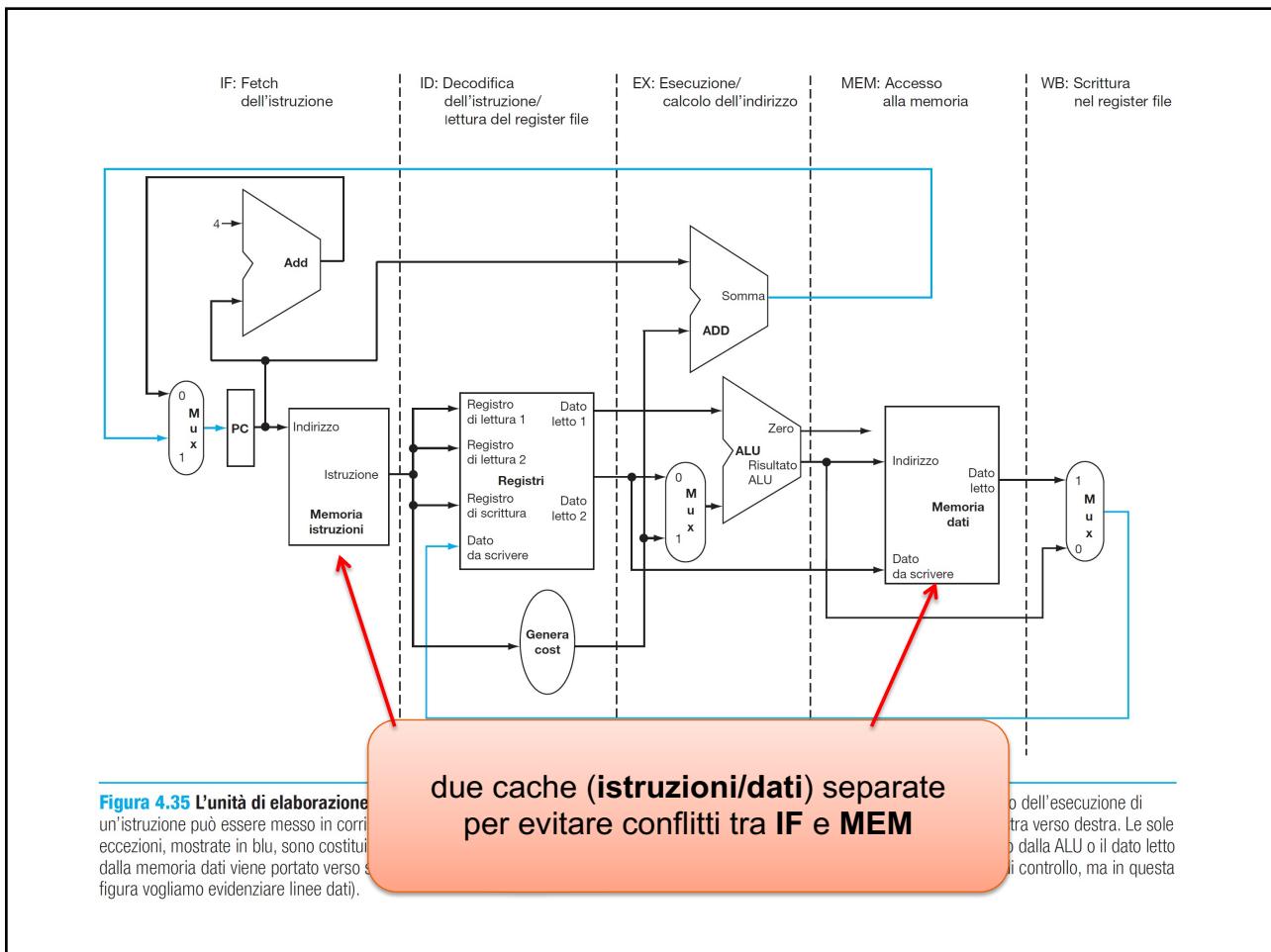


Figura 4.35 L'unità di elaborazione a singolo ciclo del Paragrafo 4.4 (simile a quella di Figura 4.34). Il diagramma illustra il flusso di elaborazione di un'istruzione, suddiviso in quattro fasi principali: IF (Fetch dell'istruzione), ID (Decodifica dell'istruzione/lettura del register file), EX (Esecuzione/calcolo dell'indirizzo) e WB (Scrittura nel register file). La fase EX è evidenziata con un cerchio rosso. Due annotazioni arancioni specificano: 1) che nell'EX si calcola l'indirizzo di salto: $NPC+(Imm <<2)$; 2) che si calcola la condizione di salto: $([A] - [B]) == 0$. Altri elementi evidenziati in blu sono il PC (aggiornato nell'IF), i registri (letti nell'ID e scritti nell'EX/WB), e i multiplexer (selezionatori) che gestiscono i dati d'ingresso per l'ALU. I dati vengono letti da registri (lettura 1 e lettura 2) e scritti nei registri (scrittura 1 e scrittura 2). L'ALU esegue operazioni di somma (ADD) o confronto (Zero). I risultati vengono memorizzati in registri o inviati alla memoria (MEM) per la scrittura.



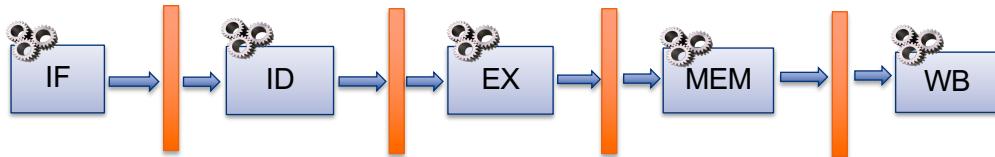




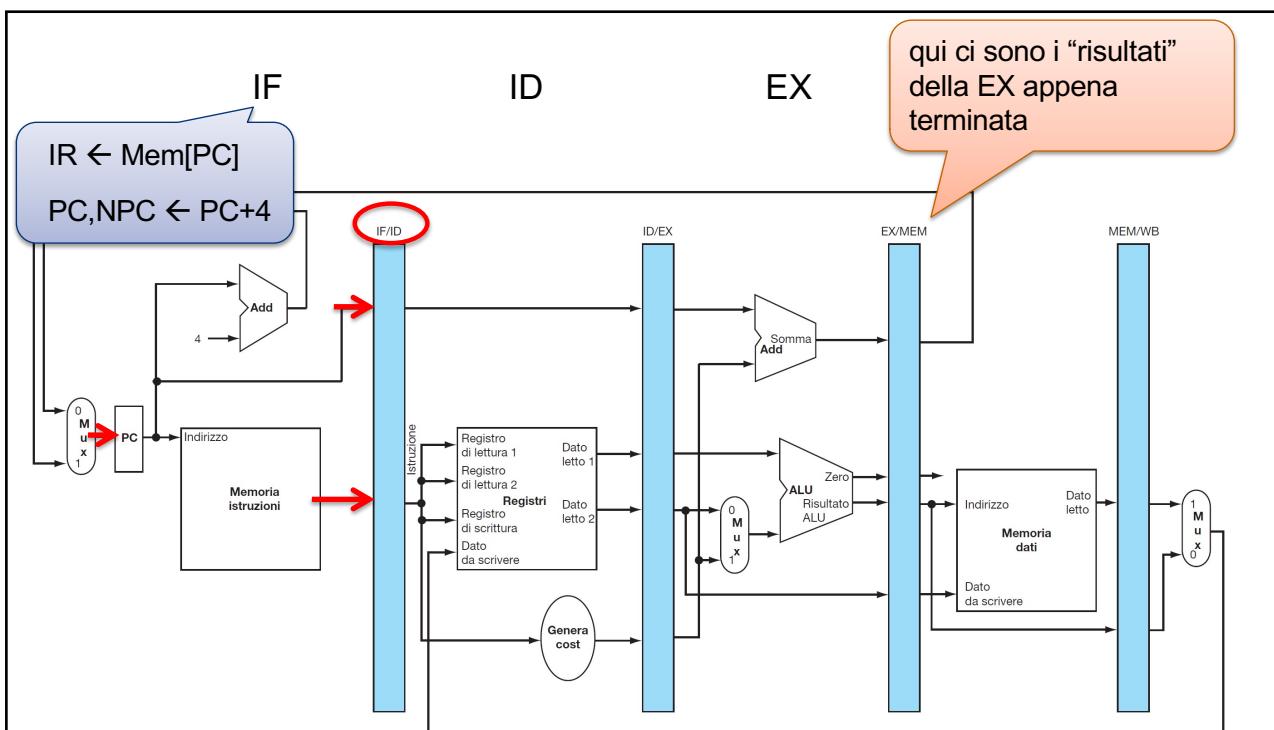
Pipeline



- **1 ciclo di clock per fase/stadio**
- 1 ciclo di clock per IF, ID, EX, MEM, WB
 - 1 istruzione in 5 cicli di clock
 - 1 ciclo di clock completa 5 fasi di 5 istruzioni diverse
 - **ogni ciclo di clock termina un'istruzione (a regime)**

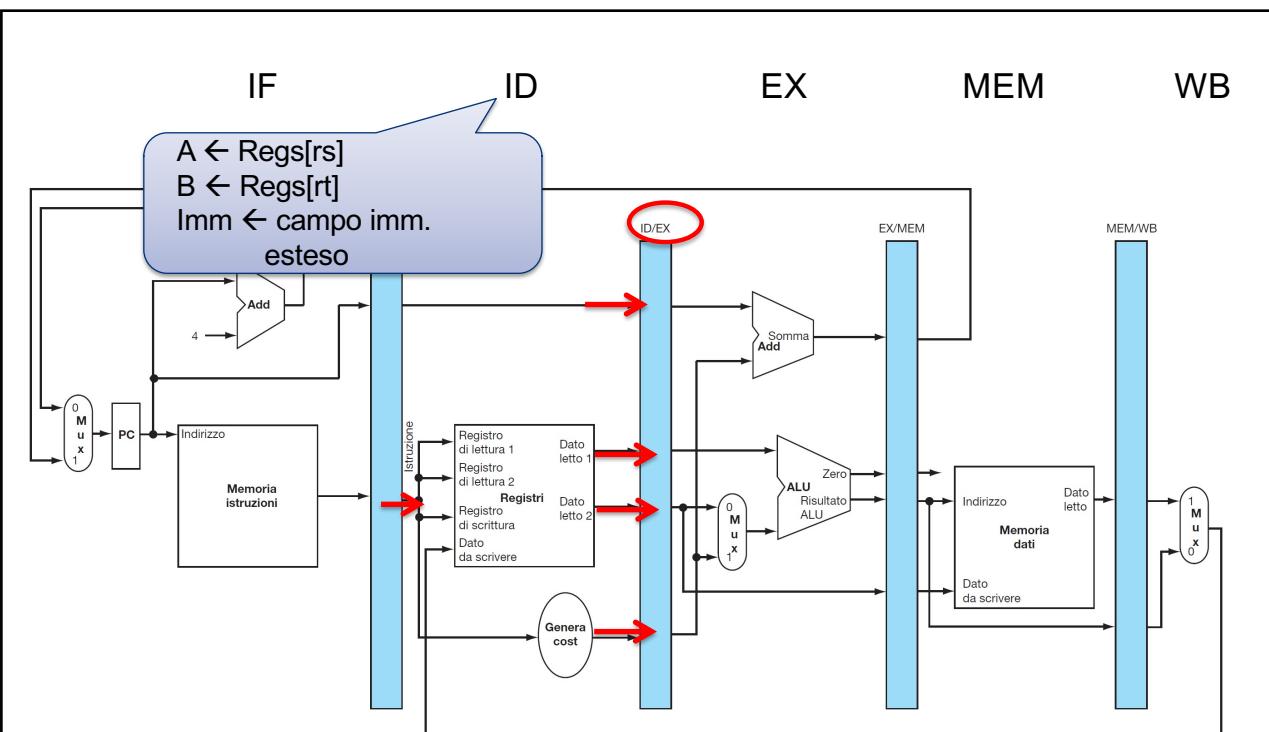


- **pipeline registers (pipeline latches)**
- i dati utili a fasi (anche non immediatamente) successive sono memorizzati nel registro successivo
- i registri memorizzano sia dati che segnali di controllo (utili in seguito)
- in ogni istante registri diversi contengono dati relativi ad istruzioni diverse



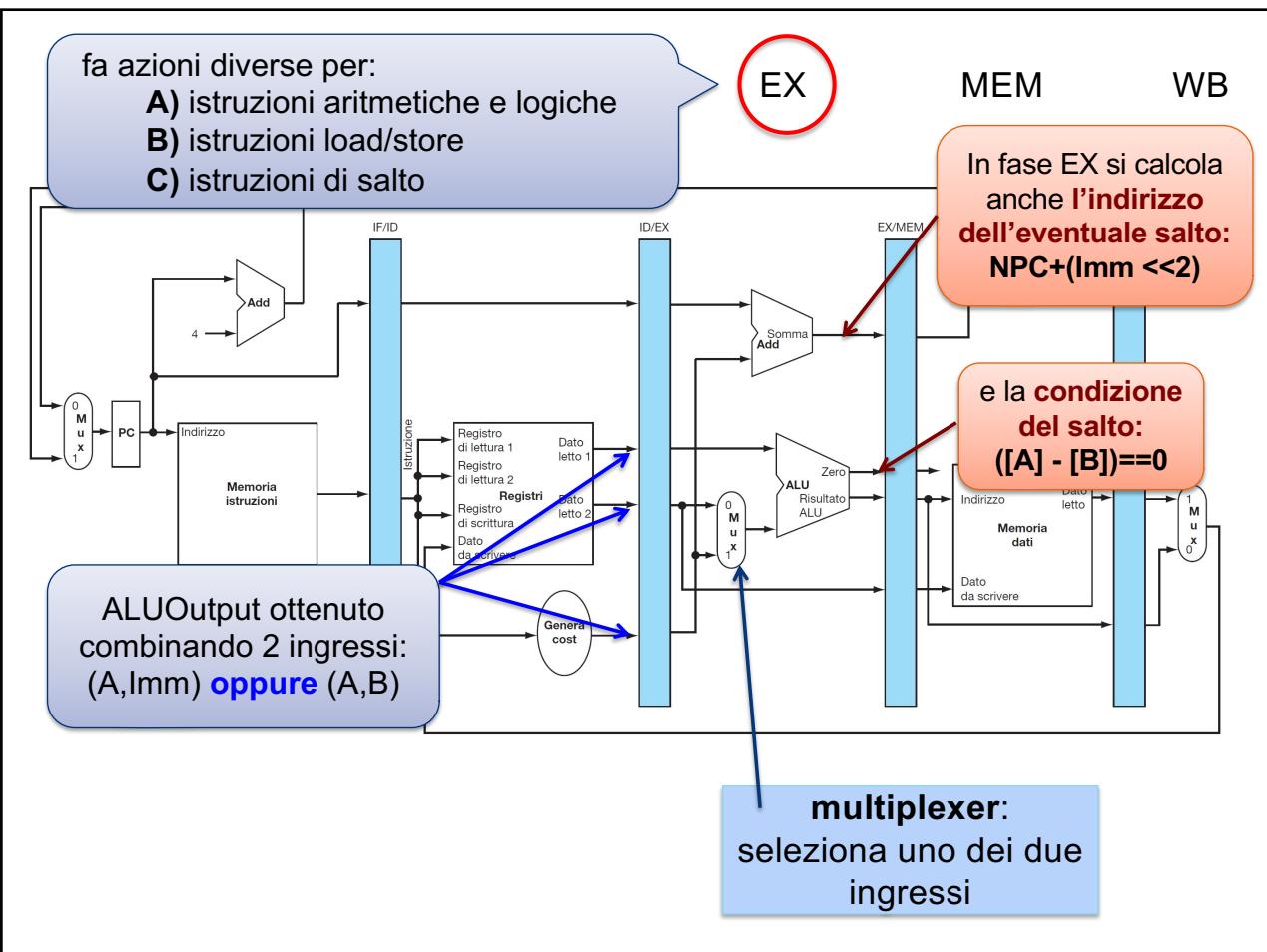
Banco registri IF/ID:

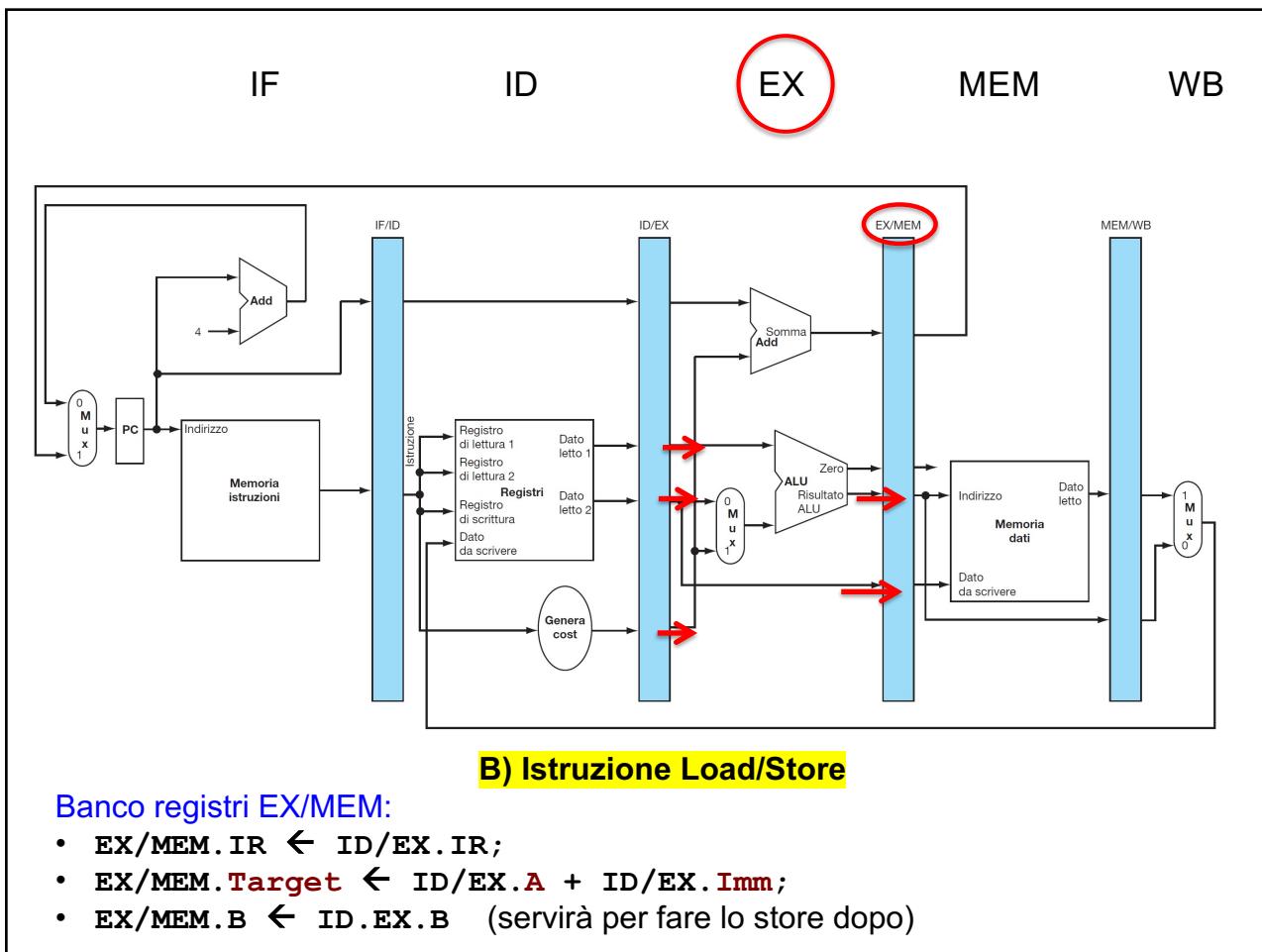
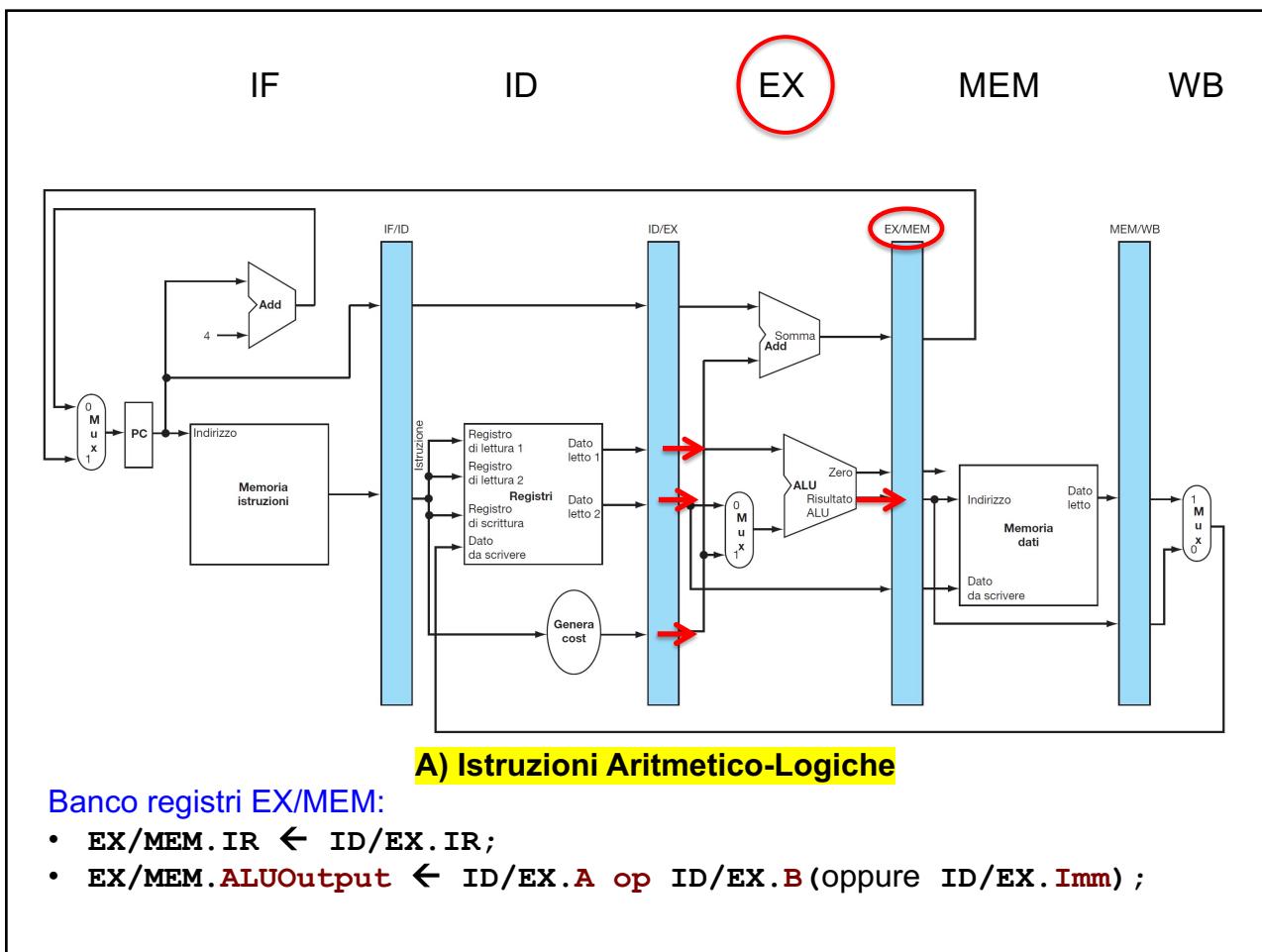
- **IF/ID.IR \leftarrow Mem[PC]**
- **IF/ID.NPC, PC \leftarrow if ((EX/MEM.IR==branch) && EX/MEM.Cond)
then {EX/MEM.Target} else {PC+4}**

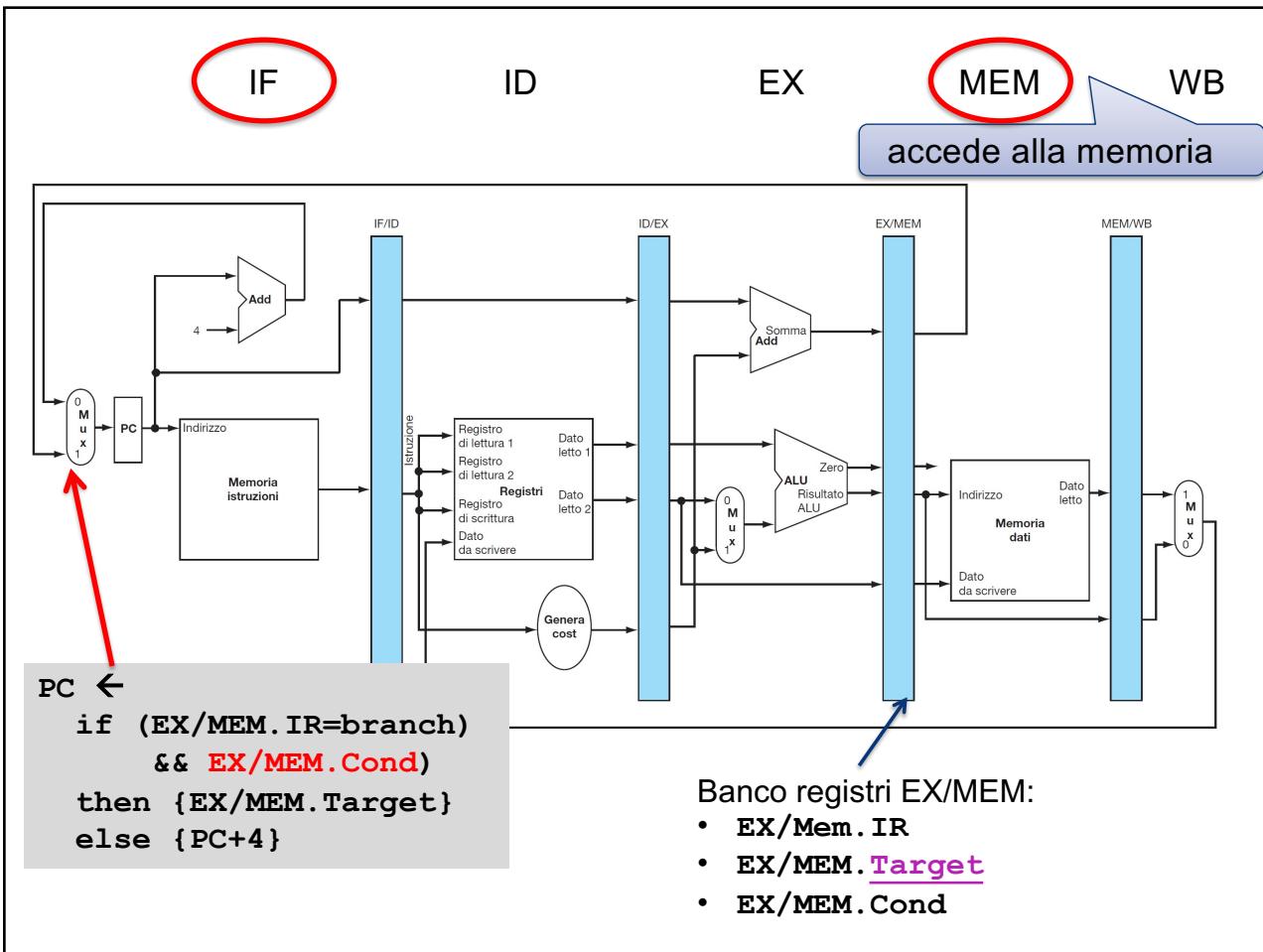
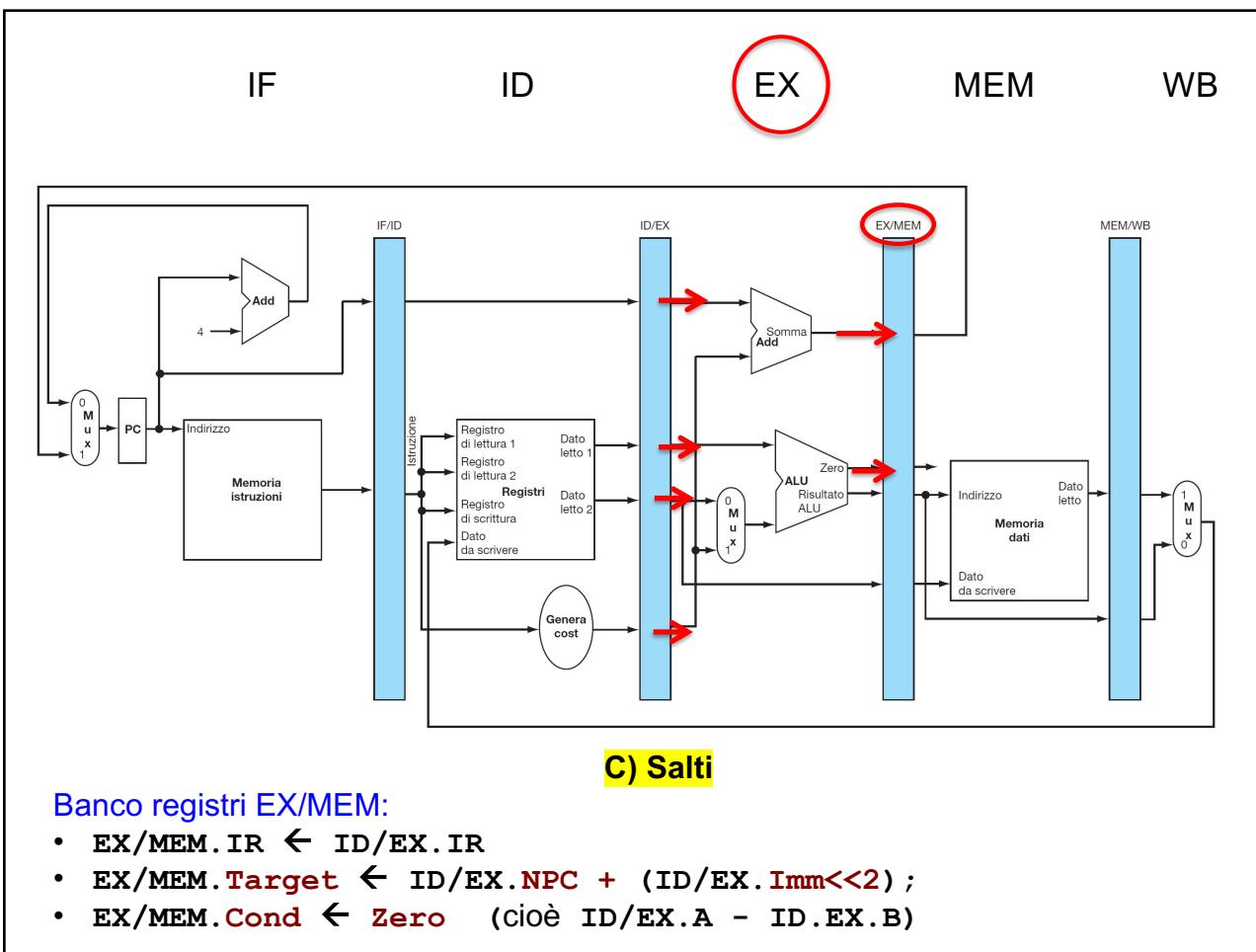


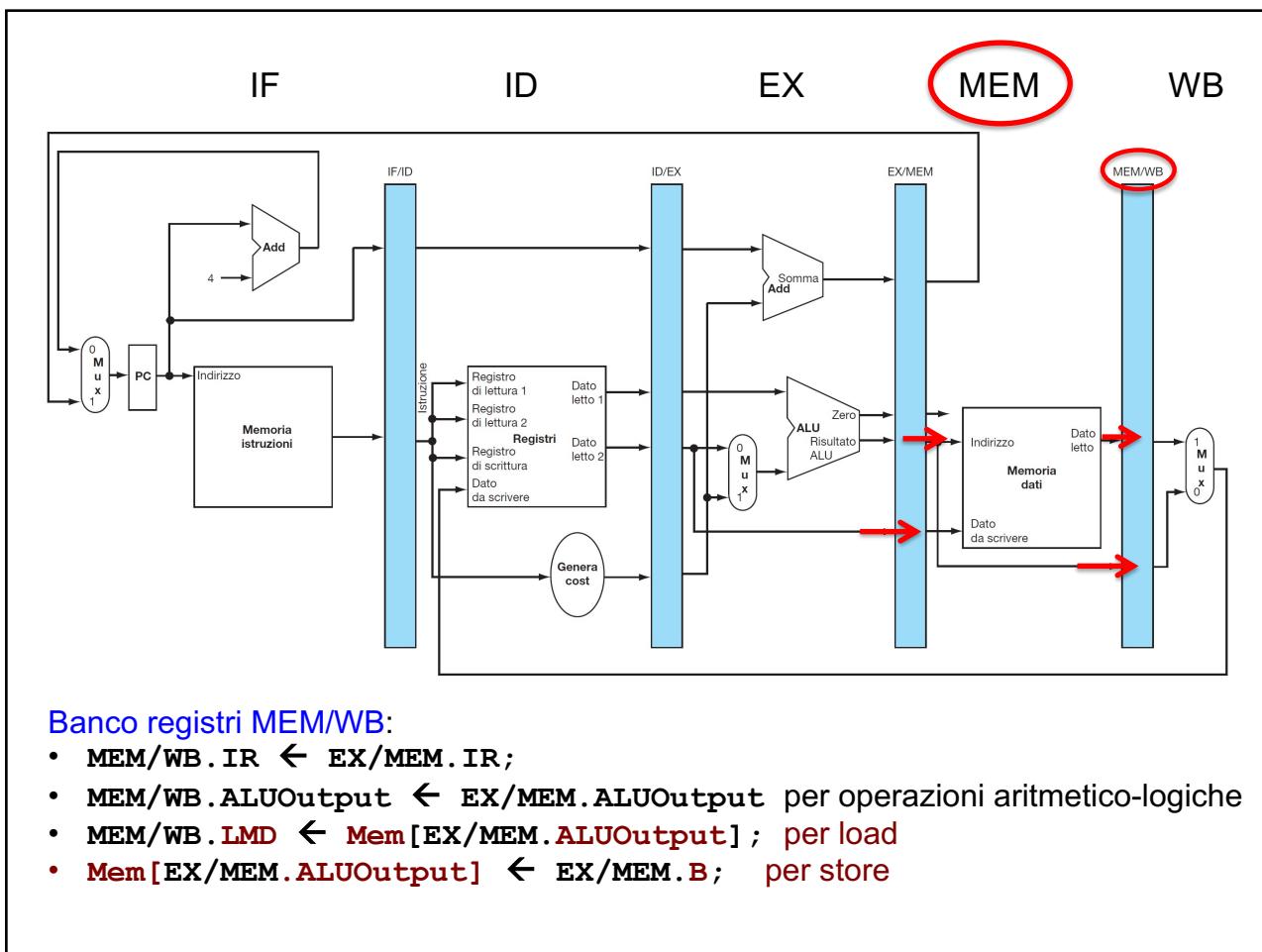
Banco registri ID/EX:

- $\text{ID/EX.IR} \leftarrow \text{IF.ID.IR};$
- $\text{ID/EX.A} \leftarrow \text{Regs[IF.ID.IR[rs1]]}; \text{ID/EX.B} \leftarrow \text{Regs[IF.ID.IR[rs2]]};$
- $\text{ID/EX.NPC} \leftarrow \text{IF.ID.NPC};$
- $\text{ID/EX.Imm} \leftarrow \text{sign-extend (IF.ID.IR[Immediate field])};$



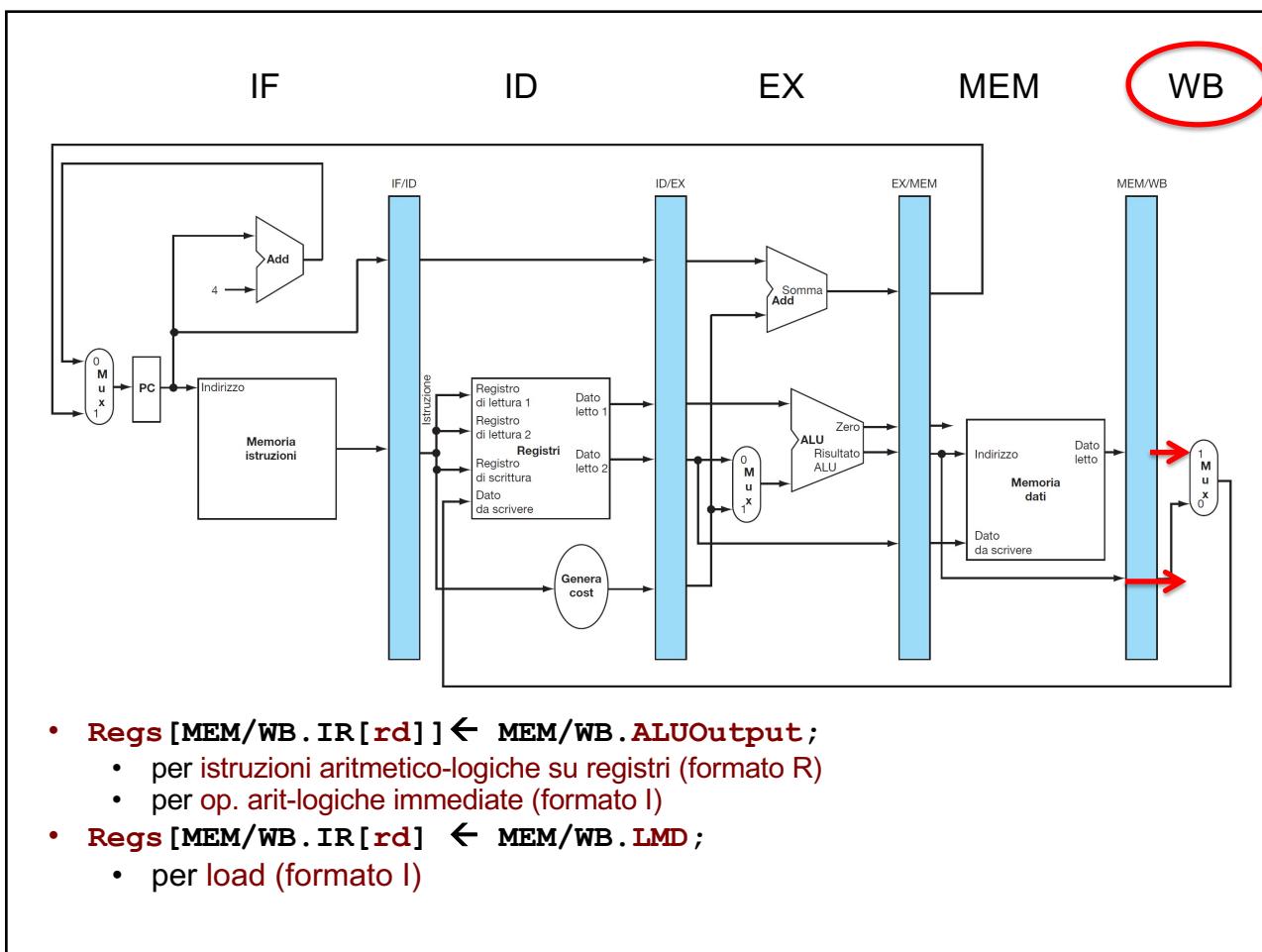






Banco registri MEM/WB:

- **MEM/WB.IR** \leftarrow EX/MEM.IR;
- **MEM/WB.ALUOutput** \leftarrow EX/MEM.ALUOutput per operazioni aritmetico-logiche
- **MEM/WB.LMD** \leftarrow Mem[EX/MEM.ALUOutput]; per load
- **Mem[EX/MEM.ALUOutput]** \leftarrow EX/MEM.B; per store



- **Regs[MEM/WB.IR[rd]]** \leftarrow MEM/WB.ALUOutput;
 - per istruzioni aritmetico-logiche su registri (formato R)
 - per op. arit-logiche immediate (formato I)
- **Regs[MEM/WB.IR[rd]]** \leftarrow MEM/WB.LMD;
 - per load (formato I)

Segnali di controllo

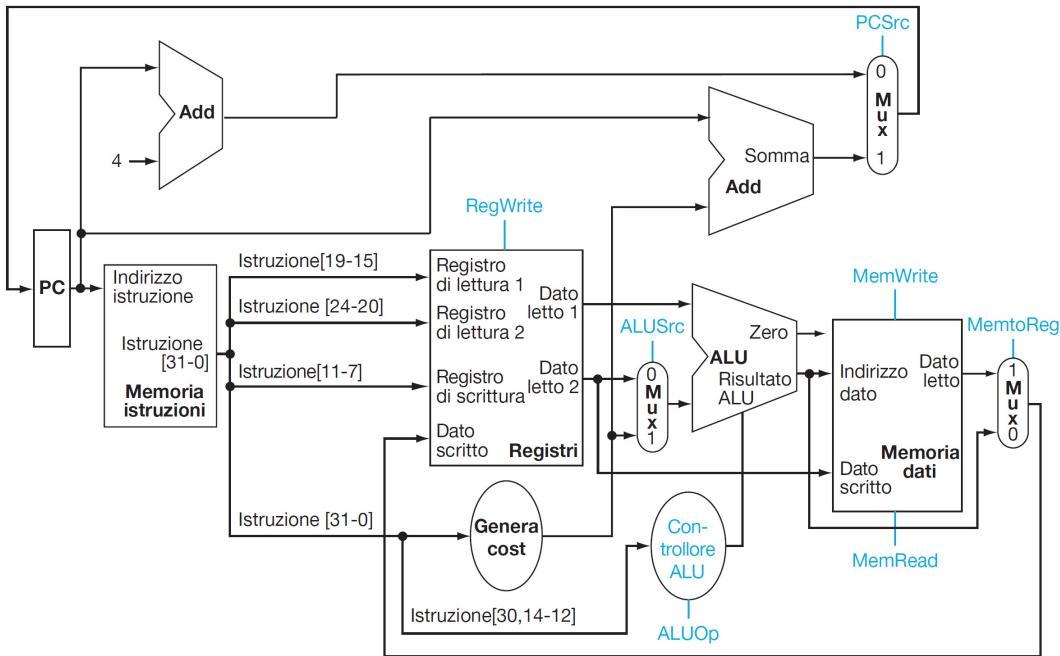


Figura 4.19 L'unità di elaborazione della Figura 4.11 con tutti i multiplexer e i segnali di controllo necessari. I segnali di controllo sono mostrati in blu. È stato introdotto anche il blocco che rappresenta il controllore della ALU. Il PC non richiede un segnale di controllo della scrittura esplicito, dal momento che viene scritto al termine di ogni ciclo di clock; la logica di controllo dei salti determina se nel PC debba essere scritto il valore precedente del PC incrementato di 4 oppure l'indirizzo di destinazione del salto.

Segnali di controllo

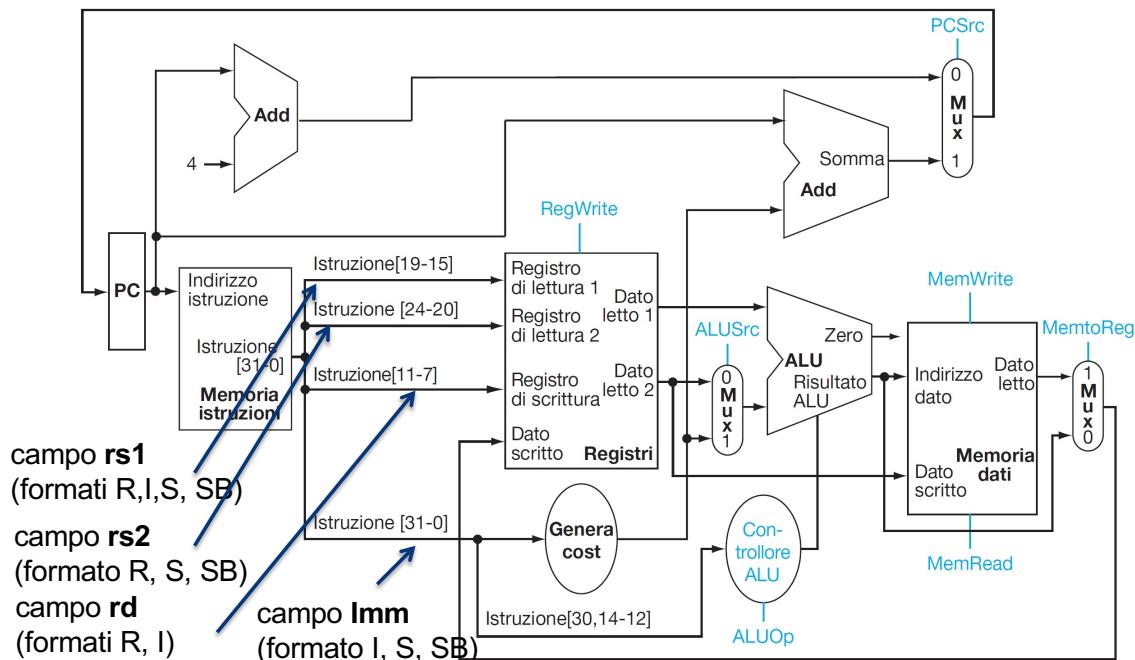


Figura 4.19 L'unità di elaborazione della Figura 4.11 con tutti i multiplexer e i segnali di controllo necessari. I segnali di controllo sono mostrati in blu. È stato introdotto anche il blocco che rappresenta il controllore della ALU. Il PC non richiede un segnale di controllo della scrittura esplicito, dal momento che viene scritto al termine di ogni ciclo di clock; la logica di controllo dei salti determina se nel PC debba essere scritto il valore precedente del PC incrementato di 4 oppure l'indirizzo di destinazione del salto.

Segnali di controllo

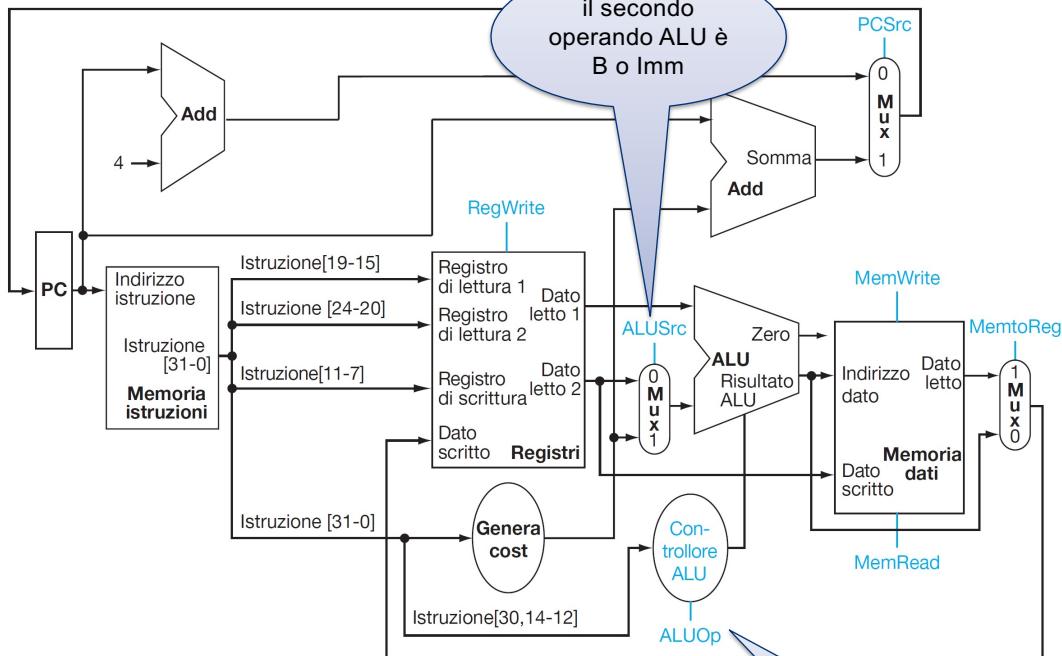


Figura 4.19 L'unità di elaborazione della Figura 4.11 con tutti i multiplexer e i segnali di controllo in blu. È stato introdotto anche il blocco che rappresenta il controllore della ALU. Il PC non richiede un segnale di controllo perché non viene incrementato ogni ciclo di clock; invece, il suo contenuto viene scritto nel momento che viene scritto al termine di ogni ciclo di clock; la logica di controllo dei salti determina se nel PC viene incrementato di 4 oppure l'indirizzo di destinazione del salto.

che op deve
fare la ALU

Segnali di controllo

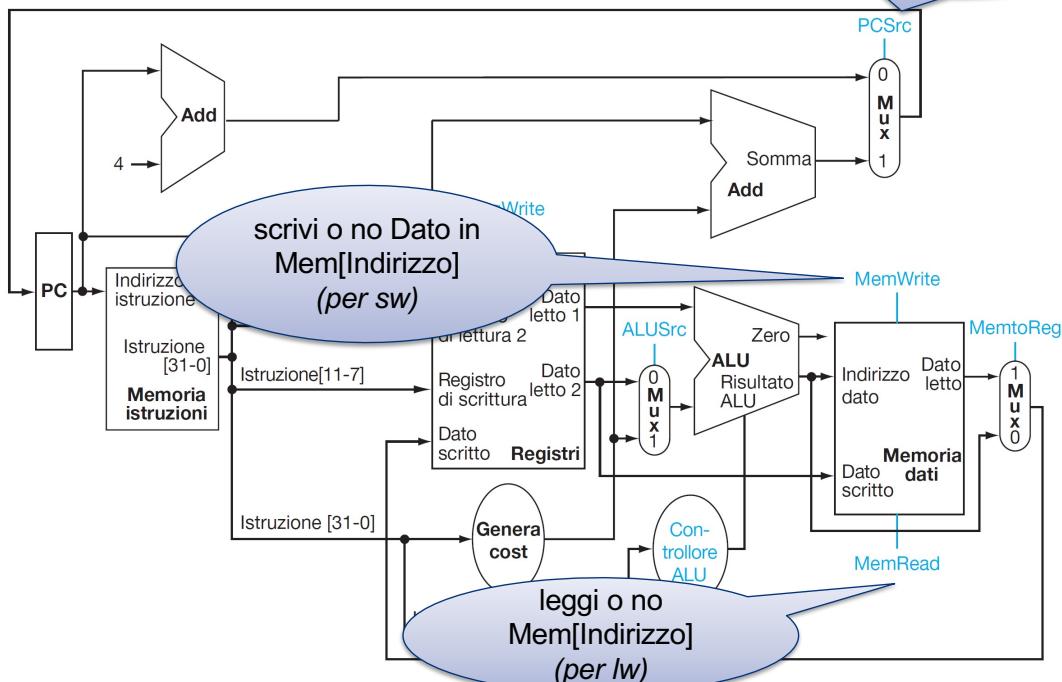


Figura 4.19 L'unità di elaborazione della Figura 4.11 con tutti i multiplexer e i segnali di controllo necessari. I segnali di controllo sono mostrati in blu. È stato introdotto anche il blocco che rappresenta il controllore della ALU. Il PC non richiede un segnale di controllo della scrittura esplicito, dal momento che viene scritto al termine di ogni ciclo di clock; la logica di controllo dei salti determina se nel PC debba essere scritto il valore precedente del PC incrementato di 4 oppure l'indirizzo di destinazione del salto.

Segnali di controllo

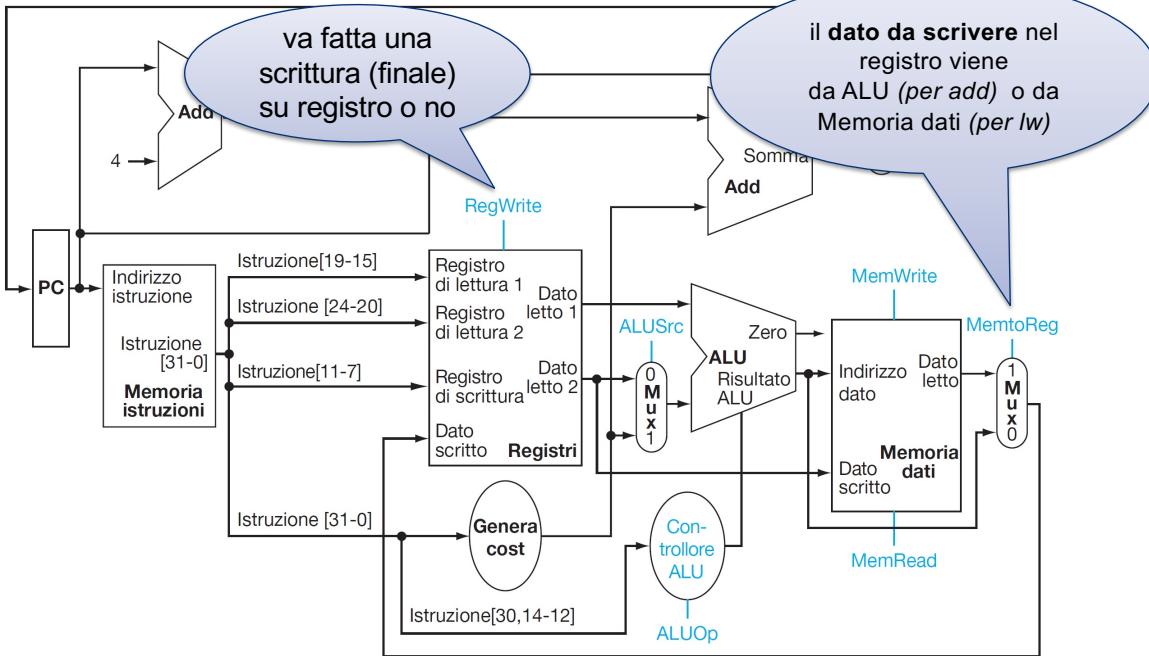


Figura 4.19 L'unità di elaborazione della Figura 4.11 con tutti i multiplexer e i segnali di controllo necessari. I segnali di controllo sono mostrati in blu. È stato introdotto anche il blocco che rappresenta il controllore della ALU. Il PC non richiede un segnale di controllo della scrittura esplicito, dal momento che viene scritto al termine di ogni ciclo di clock; la logica di controllo dei salti determina se nel PC debba essere scritto il valore precedente del PC incrementato di 4 oppure l'indirizzo di destinazione del salto.

Segnali di controllo

Nome del segnale	Effetto quando non asserito	Effetto quando asserito
RegWrite	Nullo	Il dato viene scritto nel register file nel registro individuato dal numero del registro di scrittura
ALUSrc	Il secondo operando della ALU proviene dalla seconda uscita del register file (Dato letto 2)	Il secondo operando della ALU proviene dall'estensione del segno dei 12 bit del campo immediato dell'istruzione
PCSrc	Nel PC viene scritta l'uscita del sommatore che calcola il valore di PC + 4	Nel PC viene scritta l'uscita del sommatore che calcola l'indirizzo di salto
MemRead	Nullo	Il dato della memoria nella posizione puntata dall'indirizzo viene inviato in uscita sulla linea "Dato letto"
MemWrite	Nullo	Il contenuto della memoria nella posizione puntata dall'indirizzo viene sostituito con il dato presente sulla linea "Dato scritto"
MemtoReg	Il dato inviato al register file per la scrittura proviene dalla Memoria Dati	Il dato inviato al register file per la scrittura proviene dalla Memoria Dati

Figura 4.50 Copia della Figura 4.20. Viene riportato il funzionamento dei sette segnali di controllo. Quelli della ALU (ALUOp) sono stati definiti nella seconda colonna della Figura 4.49. Quando il segnale di controllo a 1 bit di un multiplexer a due vie viene asserito, il multiplexer seleziona l'ingresso etichettato con 1; in caso contrario, cioè se il segnale di controllo è non asserito, il multiplexer seleziona l'ingresso 0. Si noti che nella Figura 4.48 PCSrc viene controllato dall'uscita di una porta AND. Se il segnale di Branch e il segnale di Zero in uscita dalla ALU assumono entrambi il valore 1, PCSrc viene impostato a 1, altrimenti viene impostato a 0. La logica di controllo impone il segnale di Branch a 1 solamente in corrispondenza delle istruzioni beq; in tutti gli altri casi, quindi, PCSrc viene impostato a 0.

Esempi di segnali di controllo

Istruzione	Segnali di controllo dello stadio di esecuzione/calcolo dell'indirizzo		Segnali di controllo dello stadio di accesso alla memoria dati			Segnali di controllo dello stadio di scrittura	
	ALUOp	ALUSrc	Branch	MemRead	MemWrite	RegWrite	MemtoReg
Tipo R	10	0	0	0	0	1	0
lw	00	1	0	1	0	1	1
sw	00	1	0	0	1	0	X
beq	01	0	1	0	0	0	X

Figura 4.51 I sette segnali di controllo per gli ultimi tre stadi della pipeline. Si noti che due dei sette segnali di controllo vengono utilizzati nella fase EX, mentre i restanti cinque vengono passati al registro di pipeline EX/MEM, esteso per memorizzare questi segnali. Tre segnali di controllo vengono utilizzati nella fase di MEM, mentre gli ultimi due vengono passati al registro MEM/WB per essere utilizzati nella fase di WB.

Formati trattati fino ad ora

Istruzioni tipo-R	funz7	rs2	rs1	funz3	rd	codop	Esempio
add (somma)	0000000	00011	00010	000	00001	0110011	add x1, x2, x3
sub (sottrazione)	0100000	00011	00010	000	00001	0110011	sub x1, x2, x3
Istruzioni tipo-I	immediato	rs1	funz3	rd	codop	Esempio	
addi (somma immediata)	001111101000	00010	000	00001	0010011	addi x1,x2,1000	
lw (load parola)	001111101000	00010	010	00001	0000011	lw x1, 1000 (x2)	
Istruzioni tipo-S	immediato	rs2	rs1	funz3	immediato	codop	Esempio
sw (store parola)	0011111	00001	00010	010	01000	0100011	sw x1, 1000 (x2)

Figura 2.6 L'architettura RISC-V descritta fino al Paragrafo 2.5. I tre formati delle istruzioni RISC-V visti finora sono R, I e S. Il formato R ha due registri sorgenti per gli operandi e un registro destinazione. Il formato I sostituisce uno dei due registri sorgente con la parte meno significativa del campo *immediato* su 12 bit. Il formato S ha due operandi sorgente e un campo immediato su 12 bit, ma non ha un registro destinazione. Il campo immediato delle istruzioni di tipo S viene suddiviso in due parti: i bit 11–5 sono contenuti nel campo immediato più a sinistra e i bit 4–0 nel secondo campo da destra.

Unità di controllo

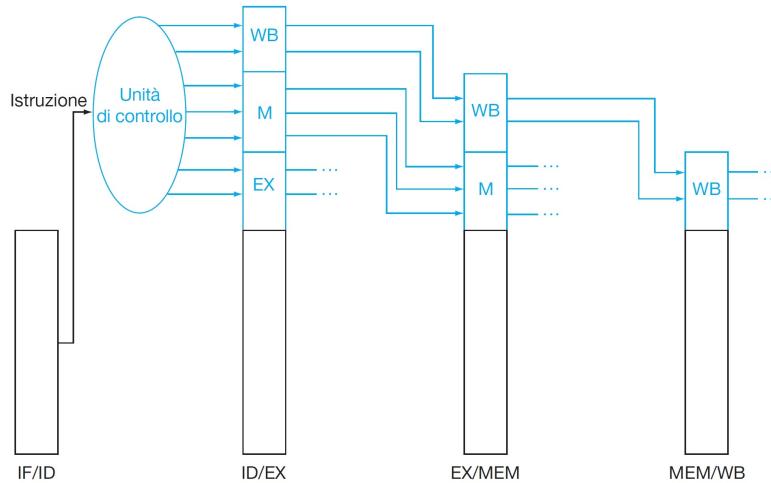


Figura 4.52 I sette segnali di controllo per gli ultimi tre stadi. Si noti che due dei sette segnali di controllo sono utilizzati nello stadio EX, mentre i rimanenti cinque vengono trasportati al registro di pipeline EX/MEM, esteso in modo da poterli contenere. Di questi cinque segnali di controllo, tre vengono impiegati nello stadio MEM, mentre gli altri due vengono trasportati al registro MEM/WB per essere poi utilizzati nello stadio WB.

- 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

PCSrc

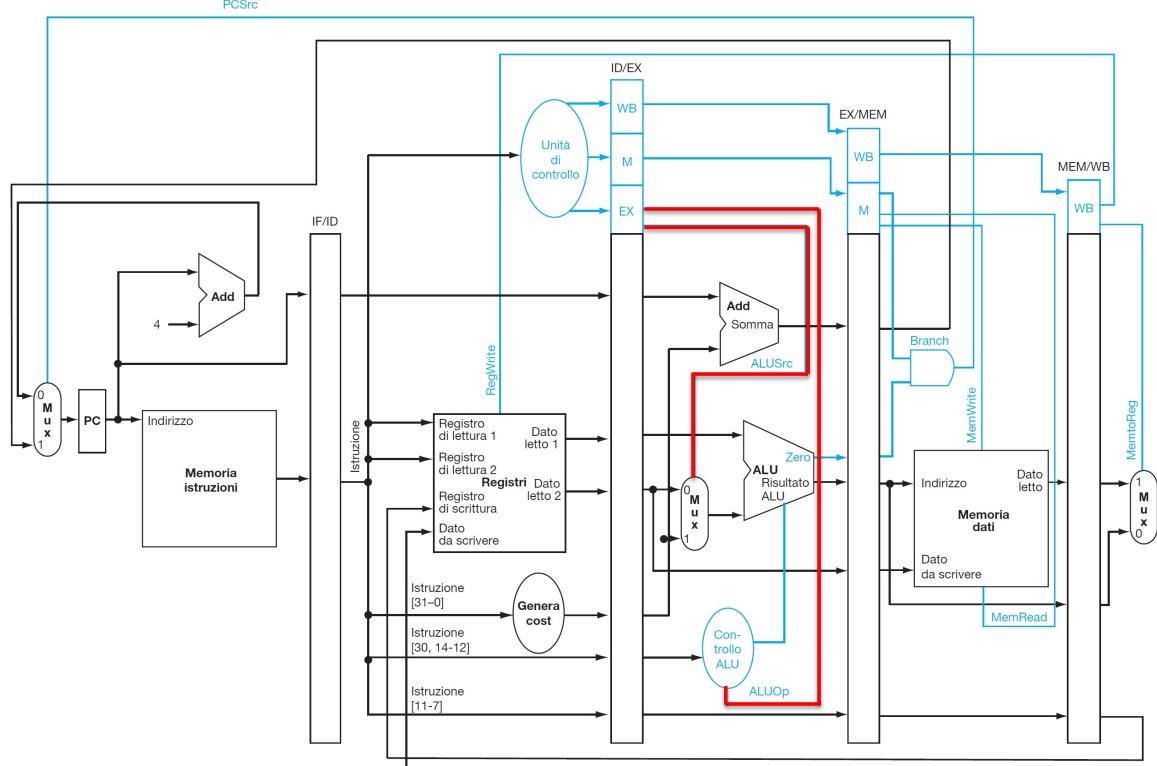


Figura 4.53 L'unità di elaborazione con pipeline della Figura 4.48 completa dei segnali di controllo uscenti dalla porzione dei registri di pipeline dedicata a memorizzare le informazioni sul controllo. I valori dei segnali di controllo degli ultimi tre stadi sono generati durante lo stadio di decodifica dell'istruzione e vengono salvati nel registro di pipeline ID/EX. In ciascuno degli stadi successivi sono impiegati alcuni di questi segnali di controllo, mentre i rimanenti vengono inviati allo stadio successivo della pipeline.

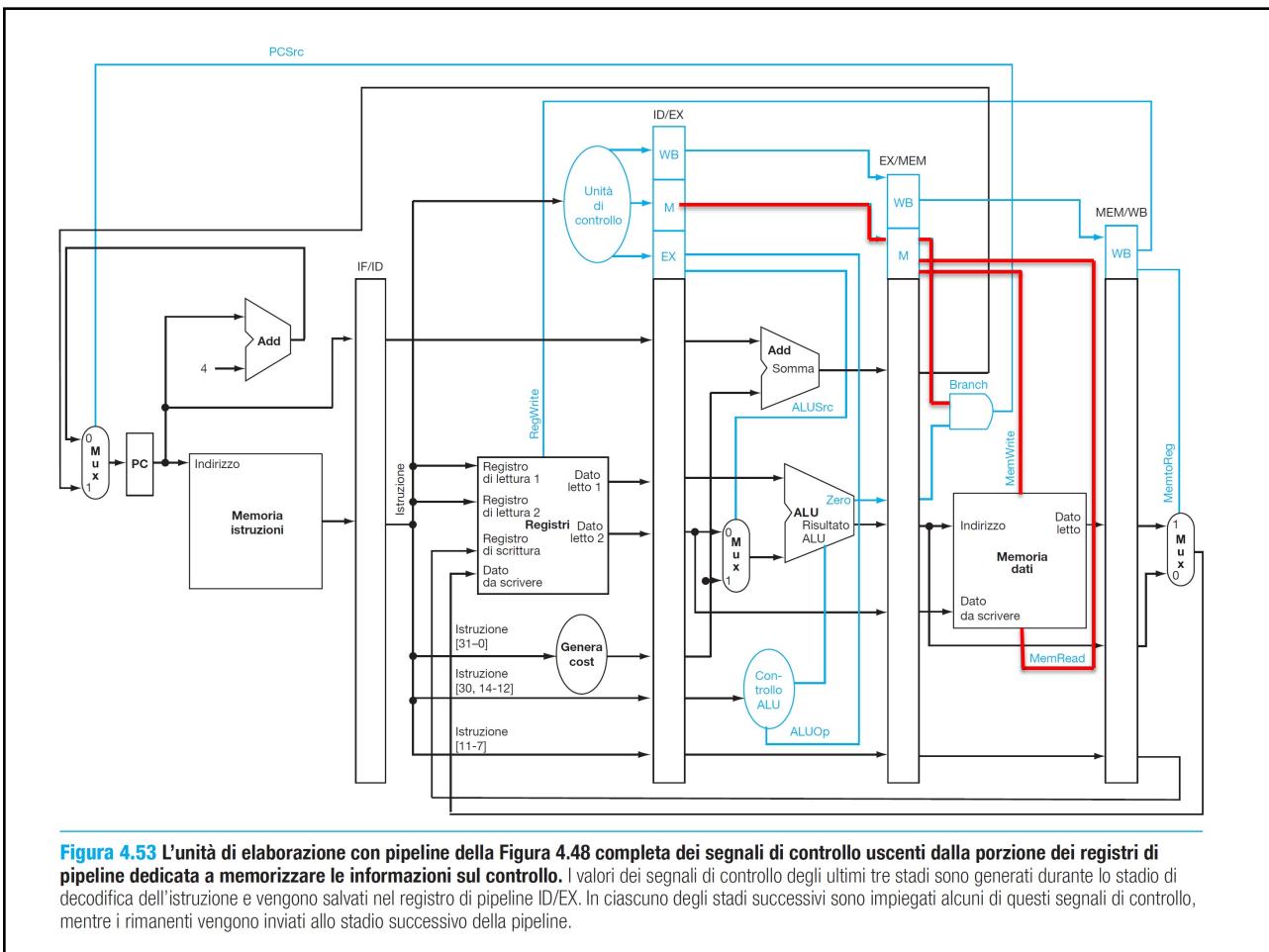
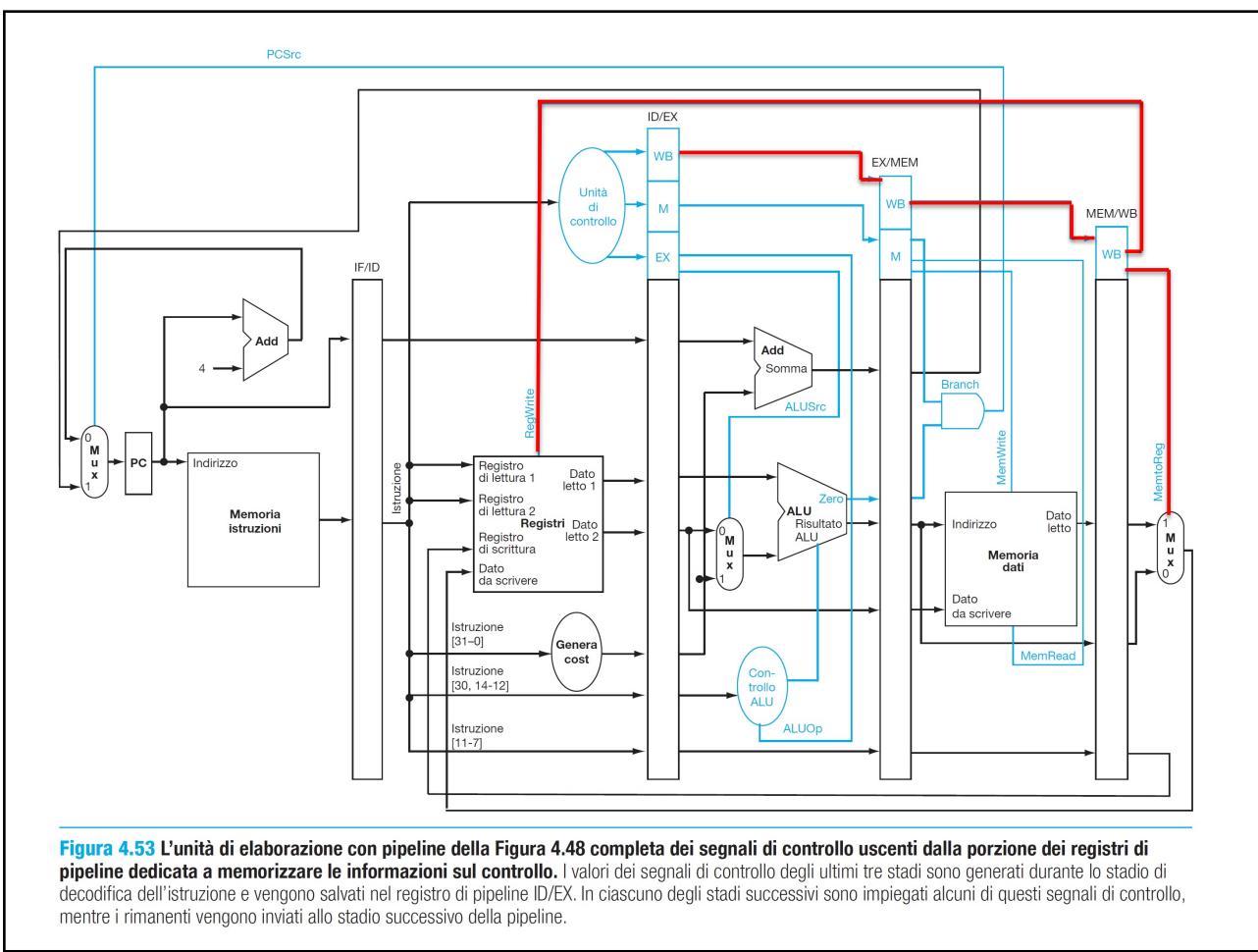


Figura 4.53 L'unità di elaborazione con pipeline della Figura 4.48 completa dei segnali di controllo uscenti dalla porzione dei registri di pipeline dedicata a memorizzare le informazioni sul controllo. I valori dei segnali di controllo degli ultimi tre stadi sono generati durante lo stadio di decodifica dell'istruzione e vengono salvati nel registro di pipeline ID/EX. In ciascuno degli stadi successivi sono impiegati alcuni di questi segnali di controllo, mentre i rimanenti vengono inviati allo stadio successivo della pipeline.



Pipeline e dipendenze dai dati

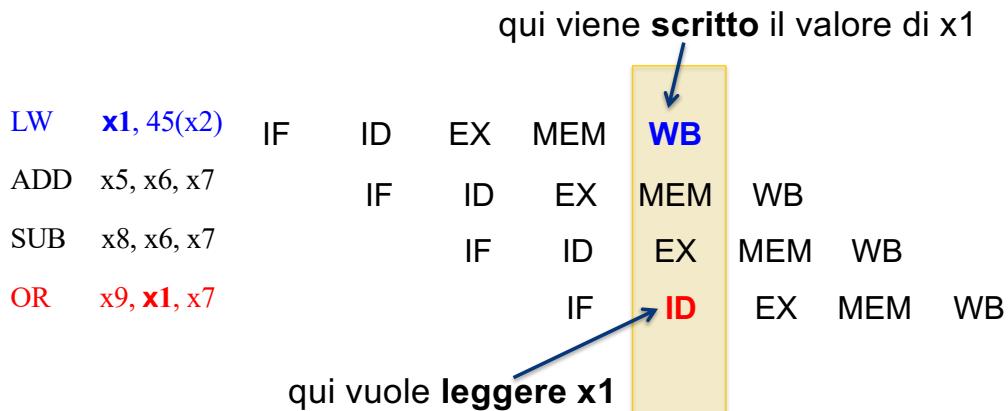
- Nella Pipeline RISC-V è possibile individuare **tutte le dipendenze dai dati nella fase ID**
 - Se si rileva una dipendenza dai dati per una istruzione, questa **va in stallo prima di essere rilasciata (issued)**, cioè quando una passa dalla fase ID a quella EX)
 - Inoltre, sempre nella fase ID, è possibile determinare che tipo di **data forwarding** adottare per evitare lo stallo ed anche predisporre gli opportuni segnali di controllo
 - Per semplicità, considereremo solo data forwarding per un'operazione che deve essere eseguita nella fase EX (operazione ALU o calcolo indirizzo)
- 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 istruzione Load

Possibili casi

Situazione	Esempio di codice	Azione
Nessuna dipendenza	LW x1, 45(x2) ADD x5, x6, x7 SUB x8, x6, x7 OR x9, x6, x7	Non occorre fare nulla
	LW x1, 45(x2) ADD x5, x1, x7 SUB x8, x6, x7 OR x9, x6, x7	
	LW x1, 45(x2) ADD x5, x6, x7 SUB x8, x1, x7 OR x9, x6, x7	
	LW x1, 45(x2) ADD x5, x6, x7 SUB x8, x6, x7 OR x9, x1, x7	

Dipendenza RAW



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

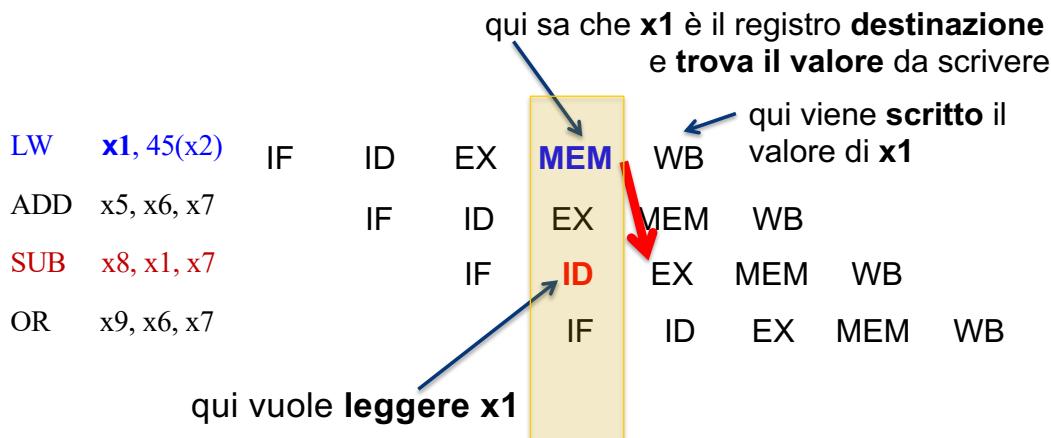
nessun problema

Dipendenze RAW da istruzione Load

Possibili casi

Situazione	Esempio di codice	Azione
Nessuna dipendenza	LW x1, 45(x2) ADD x5, x6, x7 SUB x8, x6, x7 OR x9, x6, x7	Non occorre fare nulla
	LW x1, 45(x2) ADD x5, x1, x7 SUB x8, x6, x7 OR x9, x6, x7	
	LW x1, 45(x2) ADD x5, x6, x7 SUB x8, x1, x7 OR x9, x6, x7	
Dipendenza con accessi in ordine	LW x1, 45(x2) ADD x5, x6, x7 SUB x8, x6, x7 OR x9, x1, x7	Non occorre fare nulla perché la lettura di x1 in OR avviene dopo la scrittura del dato caricato

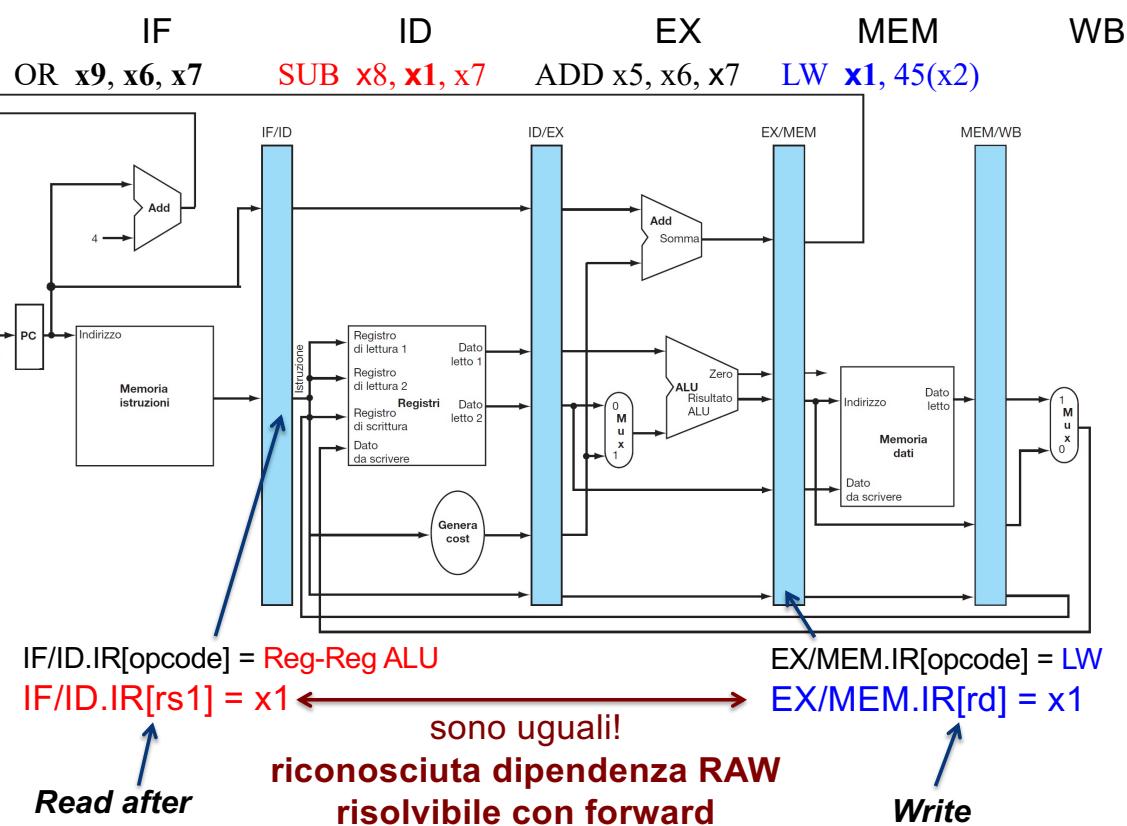
Dipendenza RAW



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

il data forwarding

(il contenuto della memoria è inviato alla ALU
prima della scrittura nel registro x1)
evita lo stallo

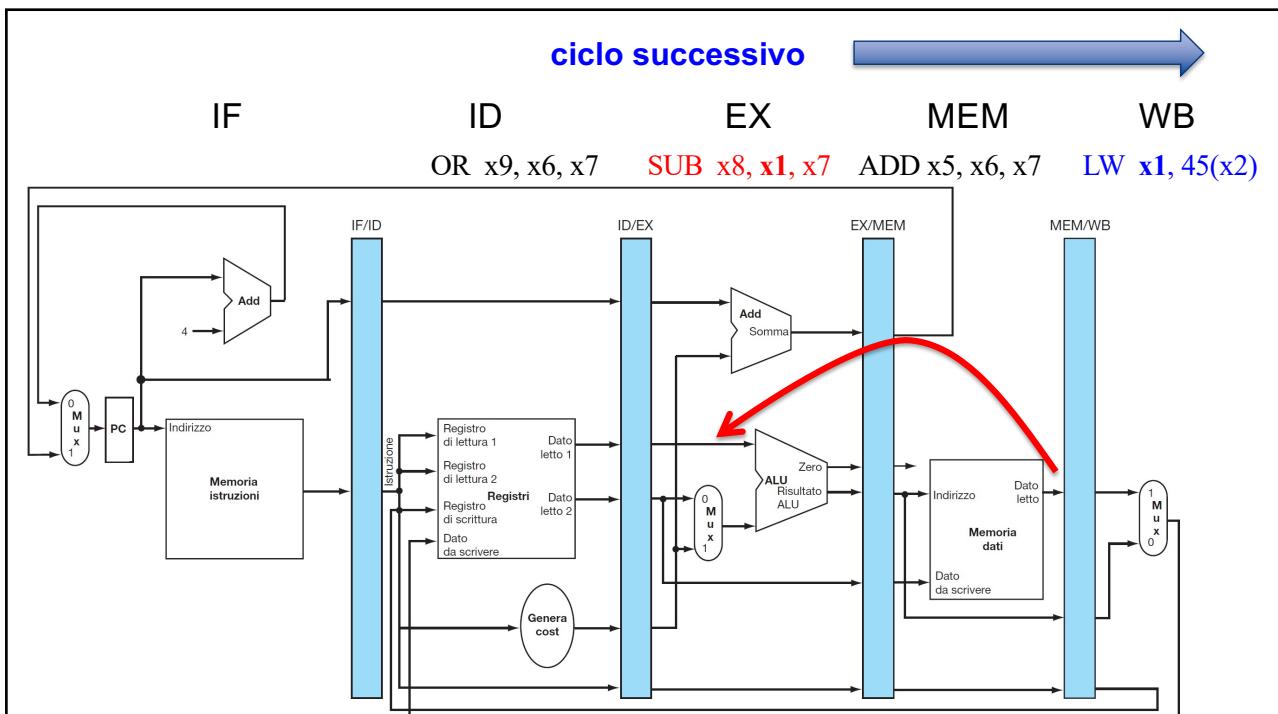


Condizioni per riconoscere la dipendenza

- In realtà, invece di controllare il codice operativo, si controlla il segnale RegWrite dell'istruzione più avanzata nella pipeline
- RegWrite è a 1 per **operazioni di tipo R e per load**

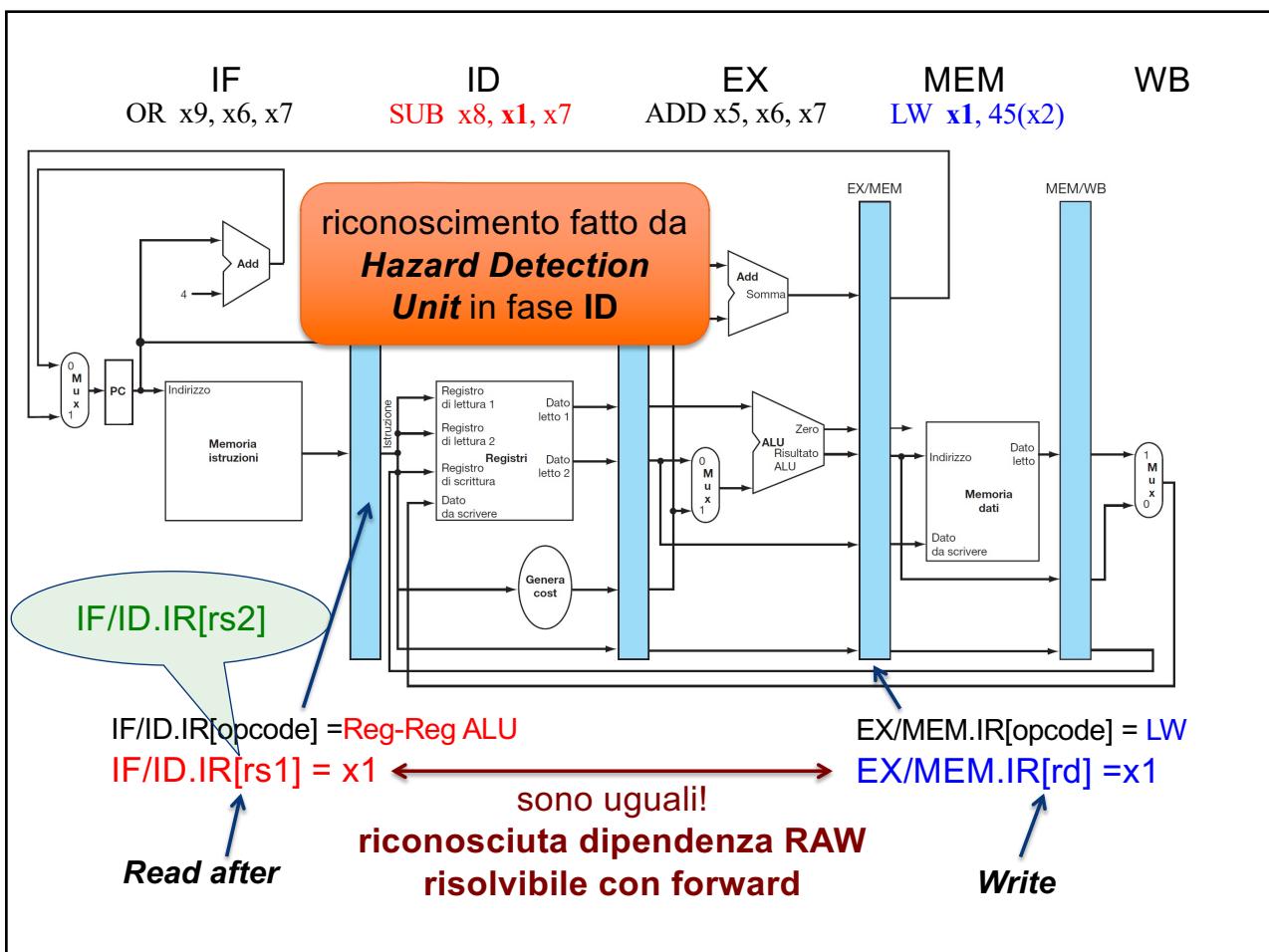
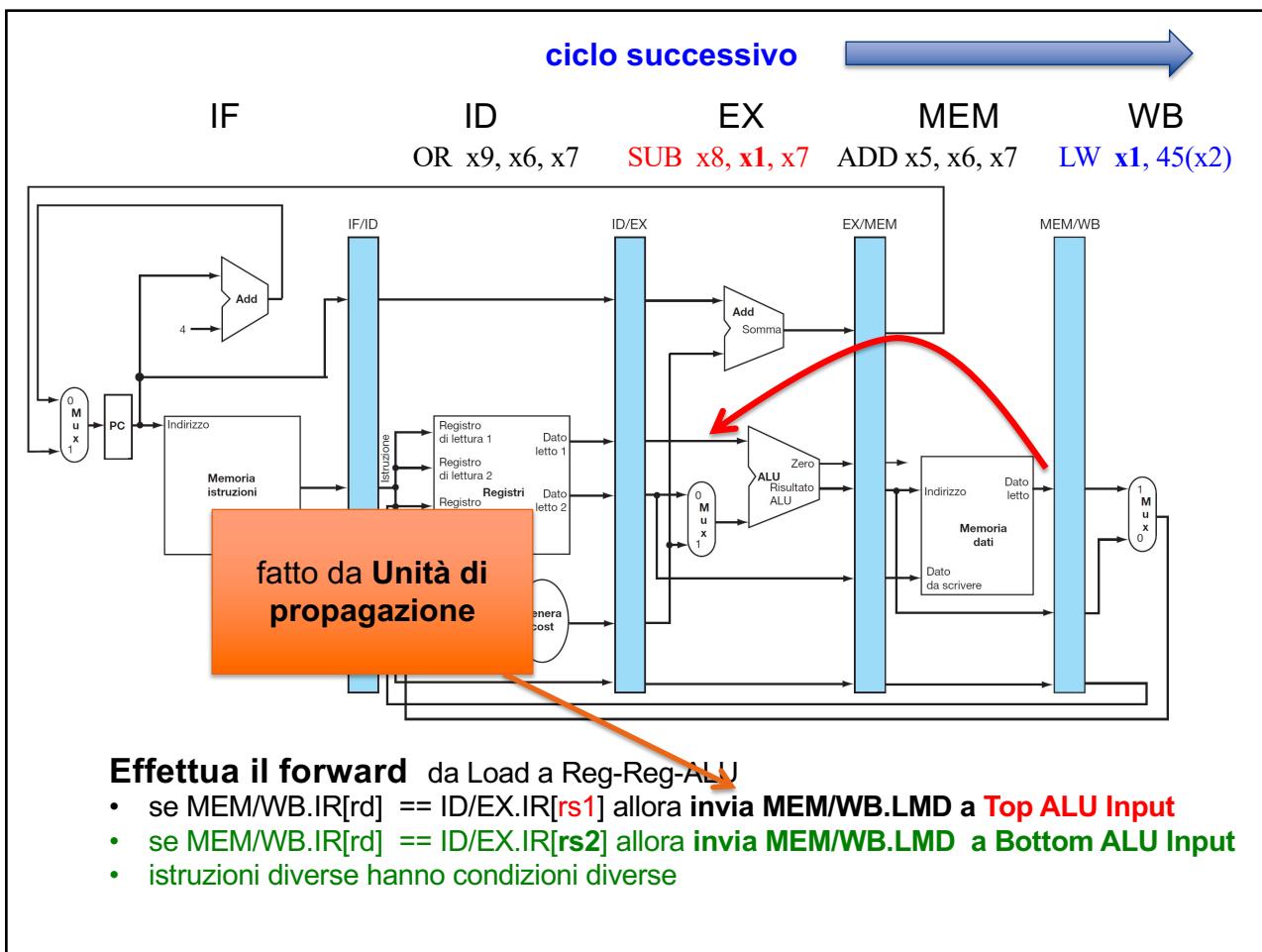
Istruzione	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
Tipo R	0	0	1	0	0	0	1	0
lw	1	1	1	1	0	0	0	0
sw	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1

Figura 4.22 Il valore assunto dalle linee di controllo è completamente determinato dal campo del codice operativo dell'istruzione. La prima riga della tabella corrisponde alle istruzioni in formato R (add, sub, and e or), per le quali i campi registro sorgente sono rs1 e rs2 e il campo registro destinazione è rd; questo stabilisce come impostare il segnale ALUSrc. Inoltre, le istruzioni di tipo R scrivono un registro (RegWrite = 1), ma non leggono né scrivono nella memoria dati. Quando il segnale di controllo Branch è impostato a 0, il PC viene aggiornato incondizionatamente a PC + 4; altrimenti, il contenuto del PC viene sostituito con l'indirizzo della destinazione del salto, ma solo quando anche l'uscita Zero della ALU è alta. Il campo ALUOp per le istruzioni di tipo R è impostato a 10 per indicare che i segnali di controllo della ALU dovranno essere generati a partire dai campi funz. La seconda e la terza riga di questa tabella riportano l'impostazione dei segnali di controllo per lw e sw, per le quali ALUSrc e ALUOp assumono il valore richiesto perché venga calcolato l'indirizzo in memoria, mentre MemRead e MemWrite determinano il tipo di accesso alla memoria. Infine, in una load, RegWrite viene impostato in modo tale che il dato letto dalla memoria venga scritto nel registro rd. Nell'istruzione di salto condizionato il campo ALUOp indica la sottrazione (01), utilizzata per controllare l'uguaglianza degli operandi. Si noti che il segnale MemtoReg risulta irrilevante quando il segnale RegWrite è uguale a 0: dal momento che l'istruzione non scrive nel register file, il valore del dato presente sulla porta di scrittura non viene utilizzato; quindi il valore di MemtoReg nelle ultime due righe viene sostituito con una X per indicare che è indifferente. La scelta di impostare questi segnali di controllo con un valore indifferente deve essere fatta dal progettista, poiché dipende dalla conoscenza del funzionamento dell'unità di elaborazione.



Effettua il forward da Load a Reg-Reg-ALU

- se MEM/WB.IR[rd] == ID/EX.IR[rs1] allora invia MEM/WB.LMD a Top ALU Input

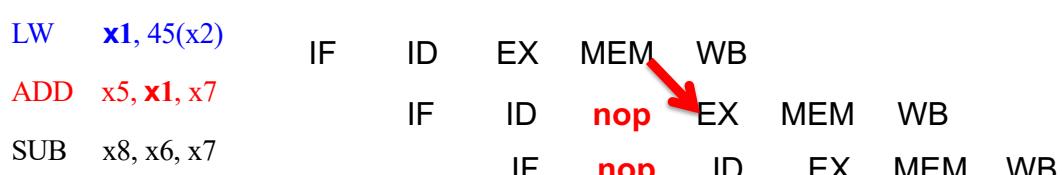
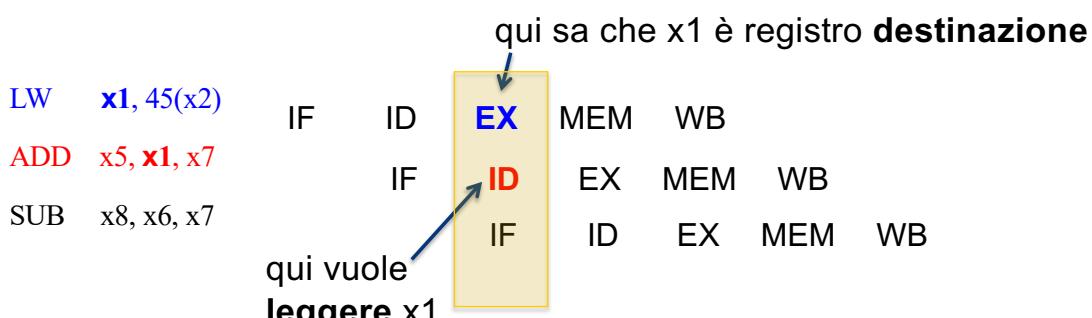


Dipendenze RAW da istruzione Load

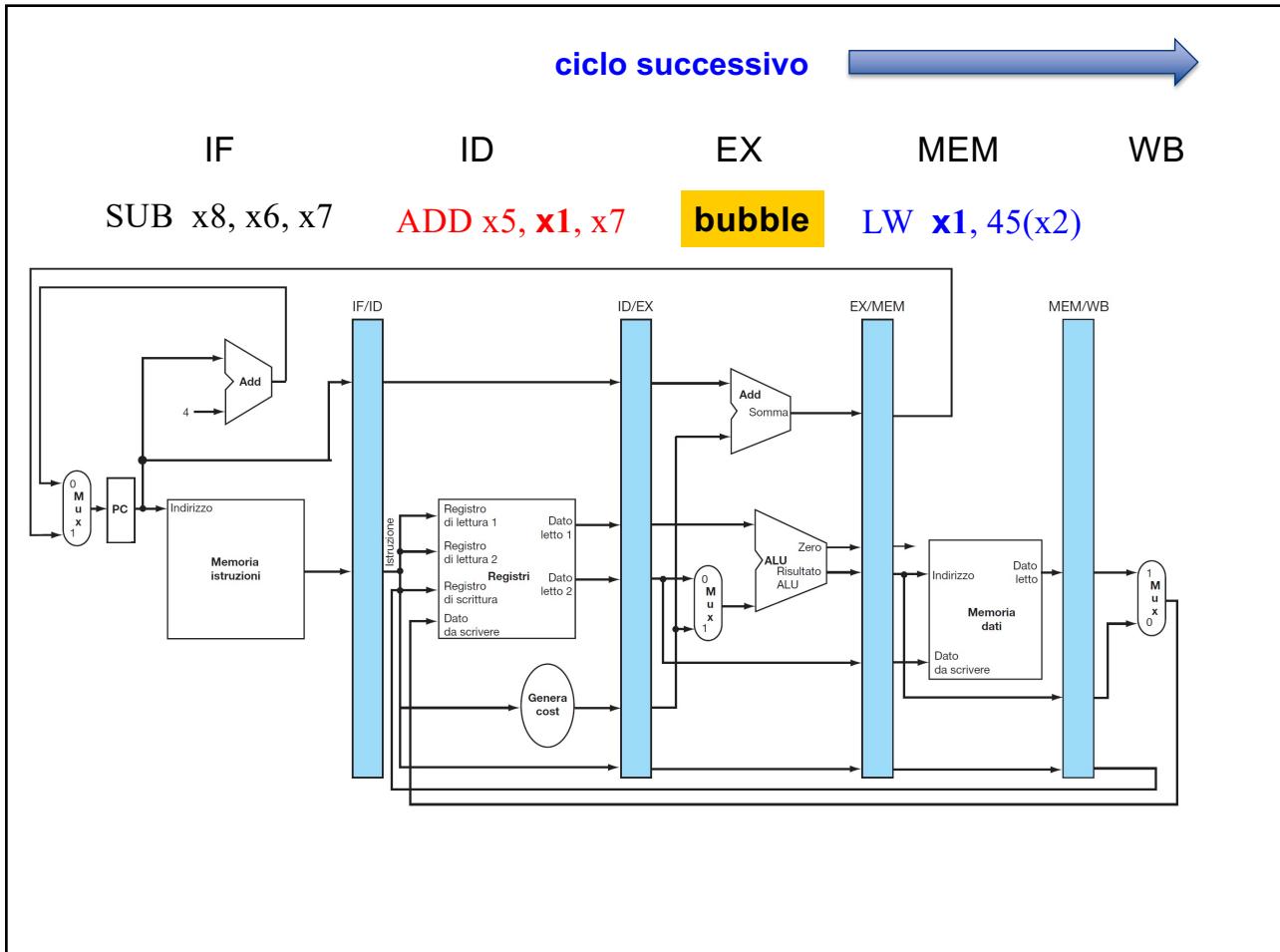
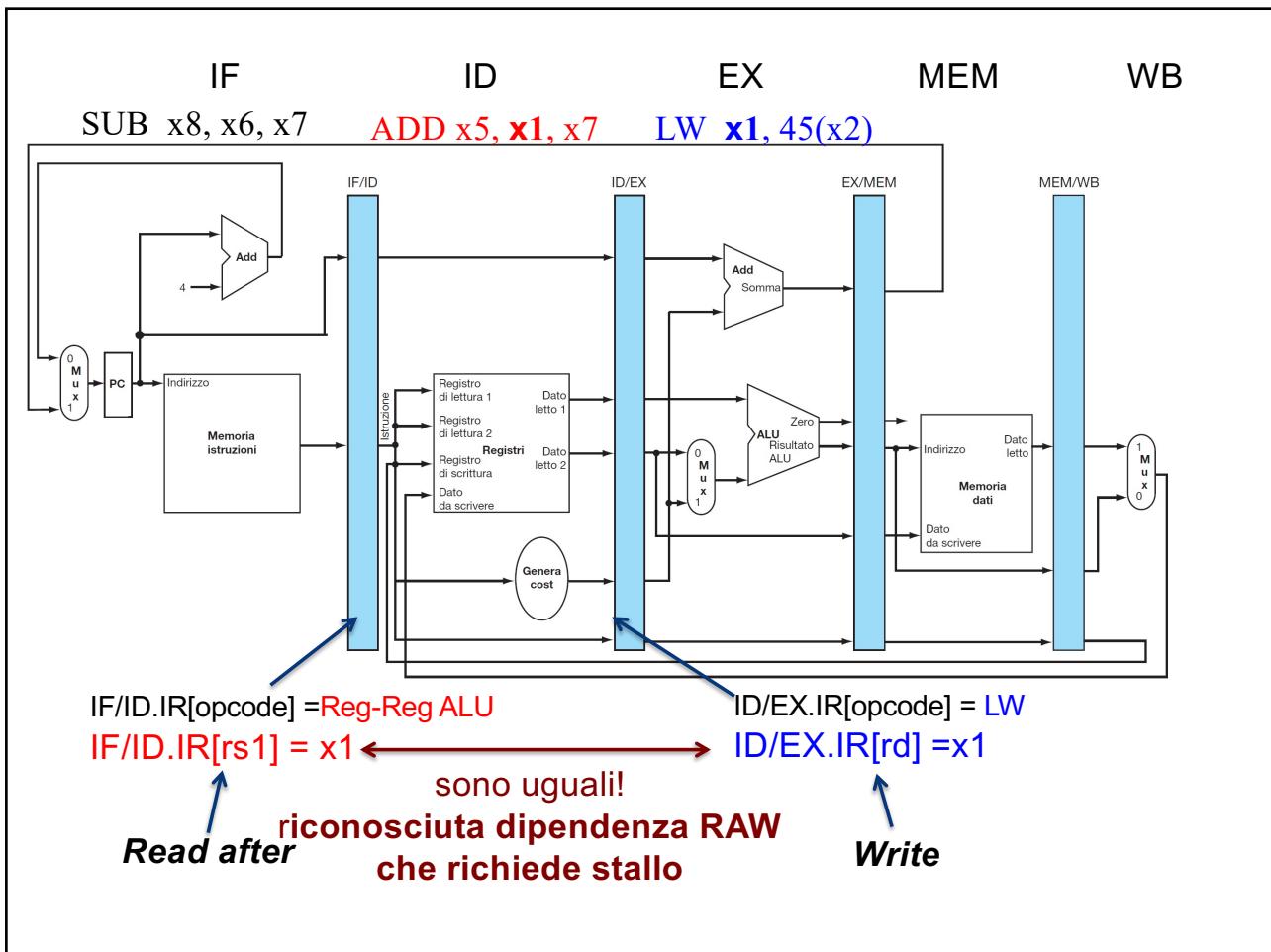
Possibili casi

Situazione	Esempio di codice	Azione
Nessuna dipendenza	LW x1, 45(x2) ADD x5, x6, x7 SUB x8, x6, x7 OR x9, x6, x7	Non occorre fare nulla
	LW x1, 45(x2) ADD x5, x1, x7 SUB x8, x6, x7 OR x9, x6, x7	
Dipendenza risolvibile con un forwarding	LW x1, 45(x2) ADD x5, x6, x7 SUB x8, x1, x7 OR x9, x6, x7	Opportuni comparatori rilevano l'uso di x1 in SUB e inoltrano il risultato della load alla ALU in tempo per la fase EX di SUB
Dipendenza con accessi in ordine	LW x1, 45(x2) ADD x5, x6, x7 SUB x8, x6, x7 OR x9, x1, x7	Non occorre fare nulla perché la lettura di x1 in OR avviene dopo la scrittura del dato caricato

Dipendenza RAW



1 ciclo di stallo e poi data forwarding



ciclo successivo



IF

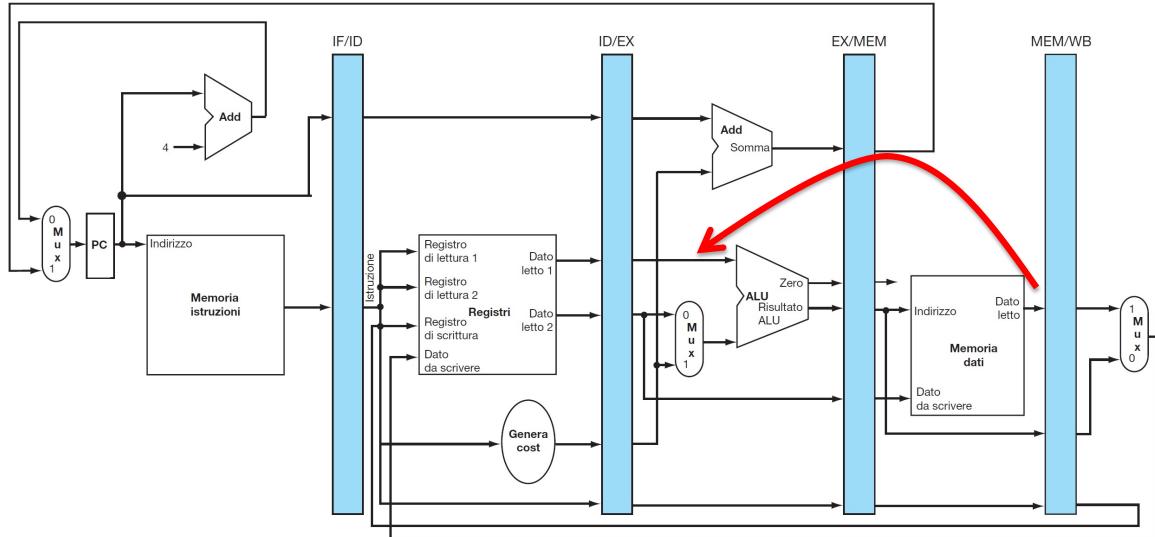
ID

EX

MEM

WB

SUB x8, x6, x7 ADD x5, x1, x7 bubble LW x1, 45(x2)

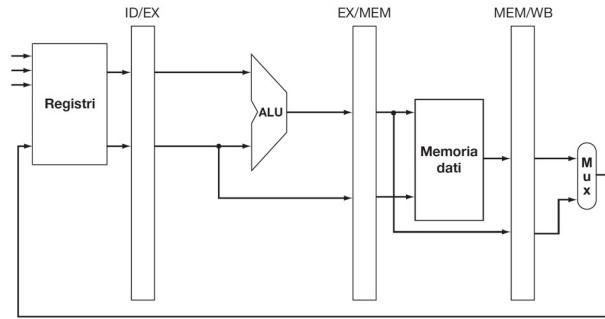


- se MEM/WB.IR[rd] == ID/EX.IR[rs1] allora manda MEM/WB.LMD a Top ALU Input

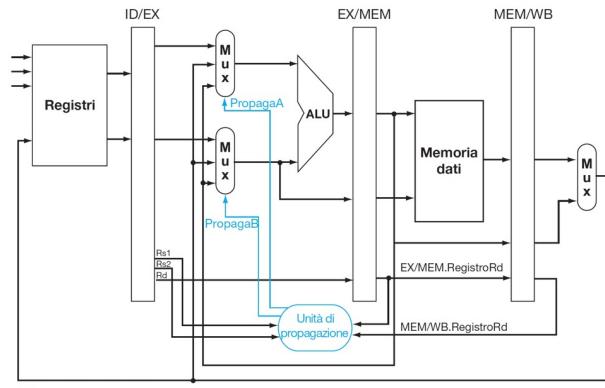
Dipendenze RAW da istruzione Load

Possibili casi

Situazione	Esempio di codice	Azione
Nessuna dipendenza	LW x1, 45(x2) ADD x5, x6, x7 SUB x8, x6, x7 OR x9, x6, x7	Non occorre fare nulla
Dipendenza che richiede uno stallo	LW x1, 45(x2) ADD x5, x1, x7 SUB x8, x6, x7 OR x9, x6, x7	Opportuni comparatori rilevano l'uso di x1 in ADD ed evitano il rilascio di ADD (quindi stallo)
Dipendenza risolvibile con un forwarding	LW x1, 45(x2) ADD x5, x6, x7 SUB x8, x1, x7 OR x9, x6, x7	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 x1, 45(x2) ADD x5, x6, x7 SUB x8, x6, x7 OR x9, x1, x7	Non occorre fare nulla perché la lettura di \$1 in OR avviene dopo la scrittura del dato caricato



(a) Nessuna propagazione



(b) Con propagazione

Figura 4.56 Nella parte superiore della figura sono mostrate la ALU e i registri di pipeline prima di aggiungere l'hardware che esegue la propagazione. Nella parte inferiore della figura sono stati inseriti dei multiplexer per la selezione dei cammini di propagazione e viene mostrata l'unità di propagazione. L'hardware aggiunto è evidenziato in blu. La figura è semplificata, in quanto non contiene alcuni dettagli dell'unità di elaborazione completa, come il circuito per l'estensione del segnale.

Pipeline RISC-V

istr.
precedent
e in fase
EX

Condizioni per riconoscere le dipendenze

$$1a. \text{EX/EM.RegisterRd} = \text{ID/EX.RegisterRs1}$$

$$1b. \text{EX/EM.RegisterRd} = \text{ID/EX.RegisterRs2}$$

$$2a. \text{MEM/WB.RegisterRd} = \text{ID/EX.RegisterRs1}$$

$$2b. \text{MEM/WB.RegisterRd} = \text{ID/EX.RegisterRs2}$$

istr.
precedent
e in fase
MEM

registro di Istruzione
scrittura
istr. successiva
in fase ID
precedente

registro di lettura di
Istruzione
successiva

condizioni verificate da **Hazard Detection Unit**

Pipeline RISC-V

Condizioni per riconoscere le dipendenze da un hazard nello stadio EX

```
if (EX/MEM.RegWrite  
and (EX/MEM.RegisterRd ≠ 0)  
and (EX/MEM.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 10  
  
if (EX/MEM.RegWrite  
and (EX/MEM.RegisterRd ≠ 0)  
and (EX/MEM.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 10
```

condizioni verificate da **Hazard Detection Unit**

Pipeline RISC-V

Condizioni per riconoscere le dipendenze da un hazard nello stadio MEM

```
if (MEM/WB.RegWrite  
and (MEM/WB.RegisterRd ≠ 0)  
and (MEM/WB.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 01  
  
if (MEM/WB.RegWrite  
and (MEM/WB.RegisterRd ≠ 0)  
and (MEM/WB.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 01
```

condizioni verificate da **Hazard Detection Unit**

Pipeline RISC-V

Condizioni per riconoscere le dipendenze da un hazard nello stadio MEM

- Una complicazione è il potenziale rischio di dipendenza dati tra il risultato dell'istruzione nella fase WB, il risultato dell'istruzione nella fase MEM e l'operando sorgente dell'istruzione nella fase ALU. Ad esempio, quando si somma un vettore di numeri in un singolo registro, una sequenza di istruzioni leggerà e scriverà tutte nello stesso registro:

add x1, x1, x2

add x1, x1, x3

add x1, x1, x4

...

- In questo caso, il risultato dovrebbe essere inoltrato dalla fase MEM perché il risultato nella fase MEM è il risultato più recente. Pertanto, il controllo del pericolo MEM sarebbe (con le aggiunte evidenziate):

```
if (MEM/WB.RegWrite  
and (MEM/WB.RegisterRd ≠ 0)  
and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)  
       and (EX/MEM.RegisterRd = ID/EX.RegisterRs1))  
and (MEM/WB.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 01  
  
if (MEM/WB.RegWrite  
and (MEM/WB.RegisterRd ≠ 0)  
and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)  
       and (EX/MEM.RegisterRd = ID/EX.RegisterRs2))  
and (MEM/WB.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 01
```

Pipeline RISC-V: hardware propagazione istruzioni R / I

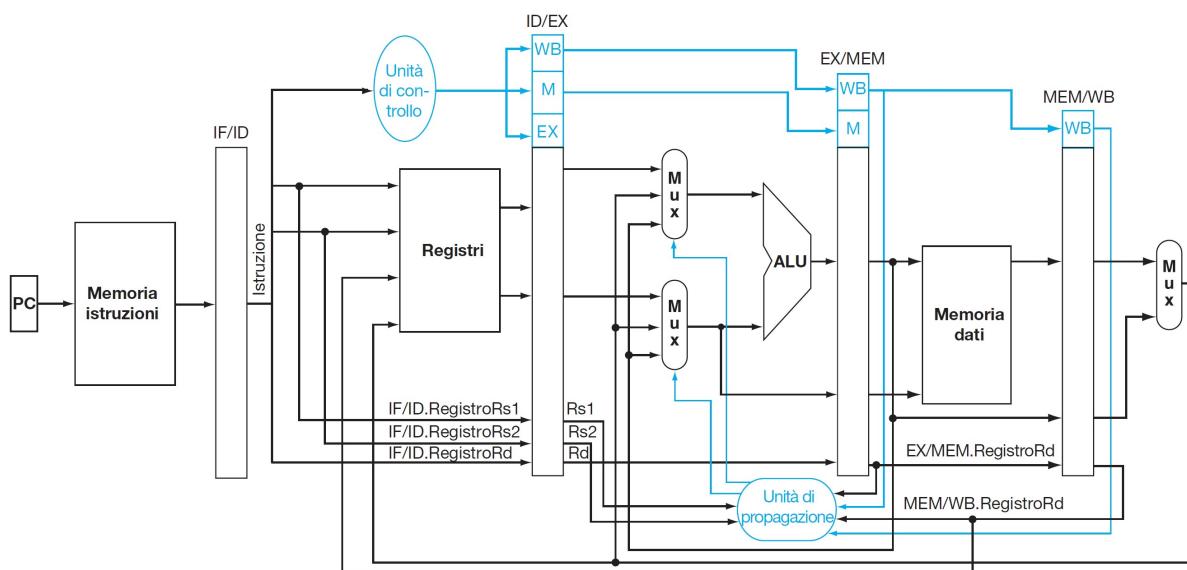


Figura 4.58 L'unità di elaborazione dopo le modifiche apportate per risolvere gli hazard tramite la propagazione. Se si confronta questo schema con l'unità di elaborazione della Figura 4.53, si nota che le aggiunte sono costituite dai multiplexer sugli ingressi della ALU. La figura rappresenta uno schema stilizzato, in cui mancano alcuni dettagli relativi all'implementazione completa che contiene anche la logica dei salti e dell'estensione del segno.

Pipeline RISC-V: hardware propagazione store

Consideriamo operazioni di copia di memoria

lw x6, 16(x3)

sw x6, 416(x3)

...

- Il valore di x6 letto da *lw* è disponibile dopo la sua fase MEM
- SW richiede tale valore al più tardi nella sua fase EX
- Si possono utilizzare gli stessi circuiti di forward, dopo aver inserito uno stall
- Dopo lo stall avremo
 $MEM/WB.RegisterRd = ID/EX.RegisterRs2$
- **Nota:** senza data forwarding, servirebbero 2 stalli perchè sw legge il contenuto di x6 nella fase ID
- **Nota2:** per non dover inserire lo stall, avremmo bisogno di ulteriori circuiti di forward più complessi (non li consideriamo)

Pipeline RISC-V: hardware propagazione istruzioni R / I / S

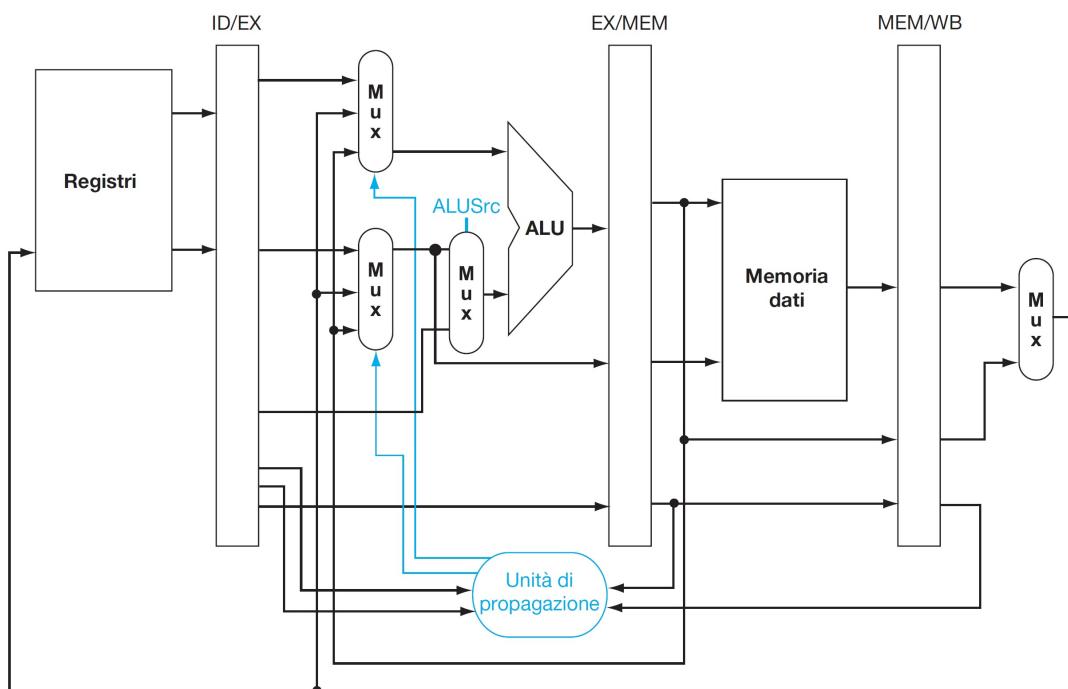


Figura 4.59 Rappresentazione più dettagliata dell'unità di elaborazione della Figura 4.56; essa contiene un multiplexer 2:1, che è stato aggiunto per potere eventualmente selezionare come secondo operando della ALU il campo immediato dotato di segno.

Pipeline RISC-V

Condizioni per decidere come effettuare il forwarding

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

- provengono:**

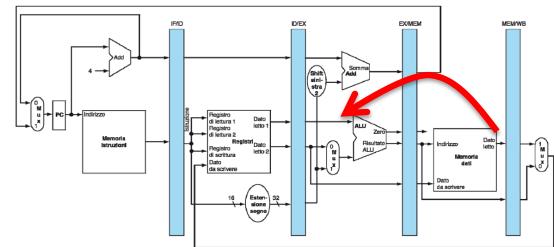
- dalla memoria dati
- dall'output della ALU
- quindi sono in registri MEM/WB opp EX/MEM

- e sono **diretti** verso:

- l'input della ALU
- (l'input della memoria dati)
- quindi verso registri ID/EX (e EX/MEM)

Quindi occorre confrontare:

se sono uguali va fatta la propagazione
(dall'Unità di Propagazione)



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

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

controlli per la sola propagazione all'input di ALU

Pipeline RISC-V

Controllo multiplexer	Sorgente	Spiegazione
PropagaA = 00	ID/EX	Il primo operando della ALU proviene dal register file
PropagaA = 10	EX/MEM	Il primo operando della ALU viene propagato dal risultato della ALU nel ciclo di clock precedente
PropagaA = 01	MEM/WB	Il primo operando della ALU viene propagato dalla memoria dati o da un precedente risultato della ALU
PropagaB = 00	ID/EX	Il secondo operando della ALU proviene dal register file
PropagaB = 10	EX/MEM	Il secondo operando della ALU viene propagato dal risultato della ALU nel ciclo di clock precedente
PropagaB = 01	MEM/WB	Il secondo operando della ALU è propagato dalla memoria dati o da un precedente risultato della ALU

Figura 4.57 Il valore assunto dai segnali di controllo dei multiplexer di propagazione della Figura 4.56. Il campo immediato dotato di segno che costituisce un ulteriore ingresso della ALU viene descritto nella sezione *Approfondimento* al termine di questo paragrafo.

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

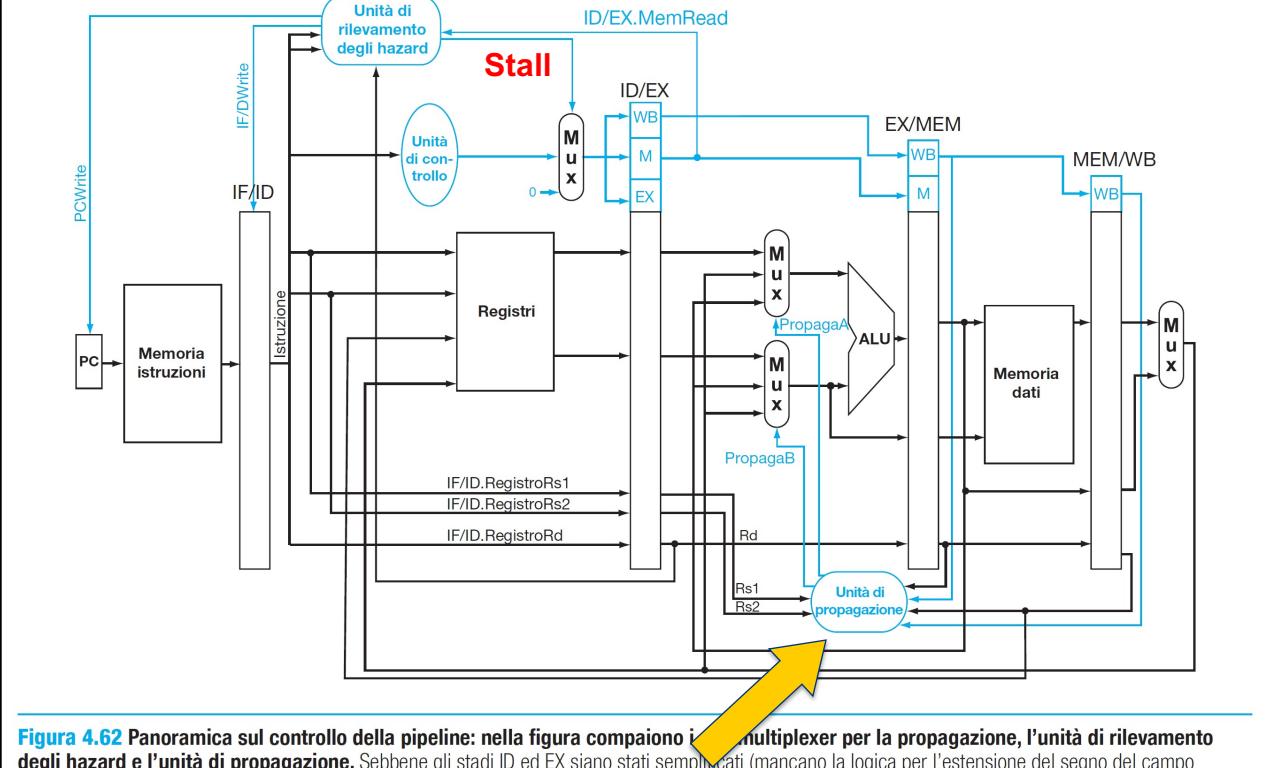


Figura 4.62 Panoramica sul controllo della pipeline: nella figura compaiono i multiplexer per la propagazione, l'unità di rilevamento degli hazard e l'unità di propagazione. Sebbene gli stadi ID ed EX siano stati semplificati (mancano la logica per l'estensione del segno del campo

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

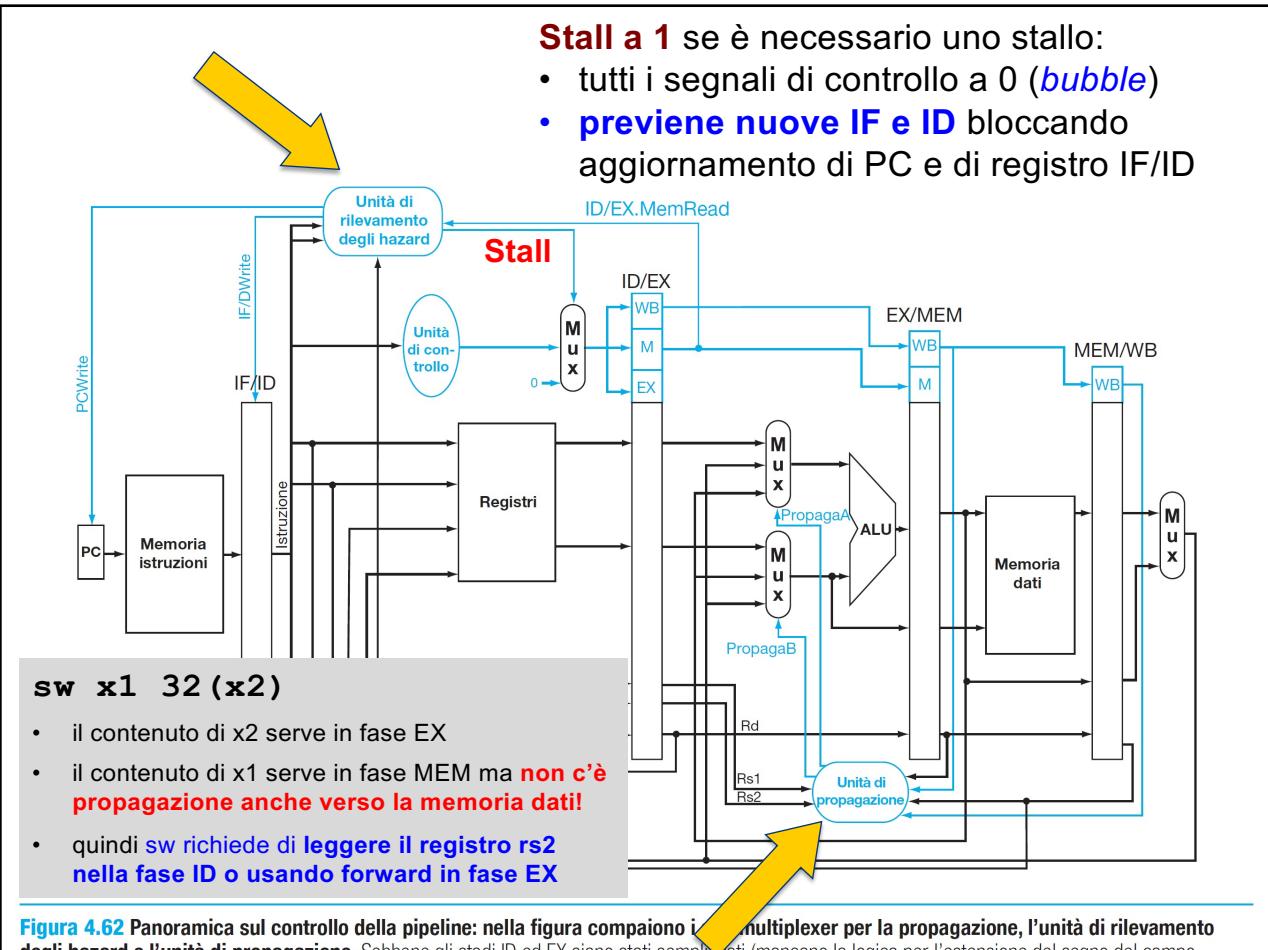
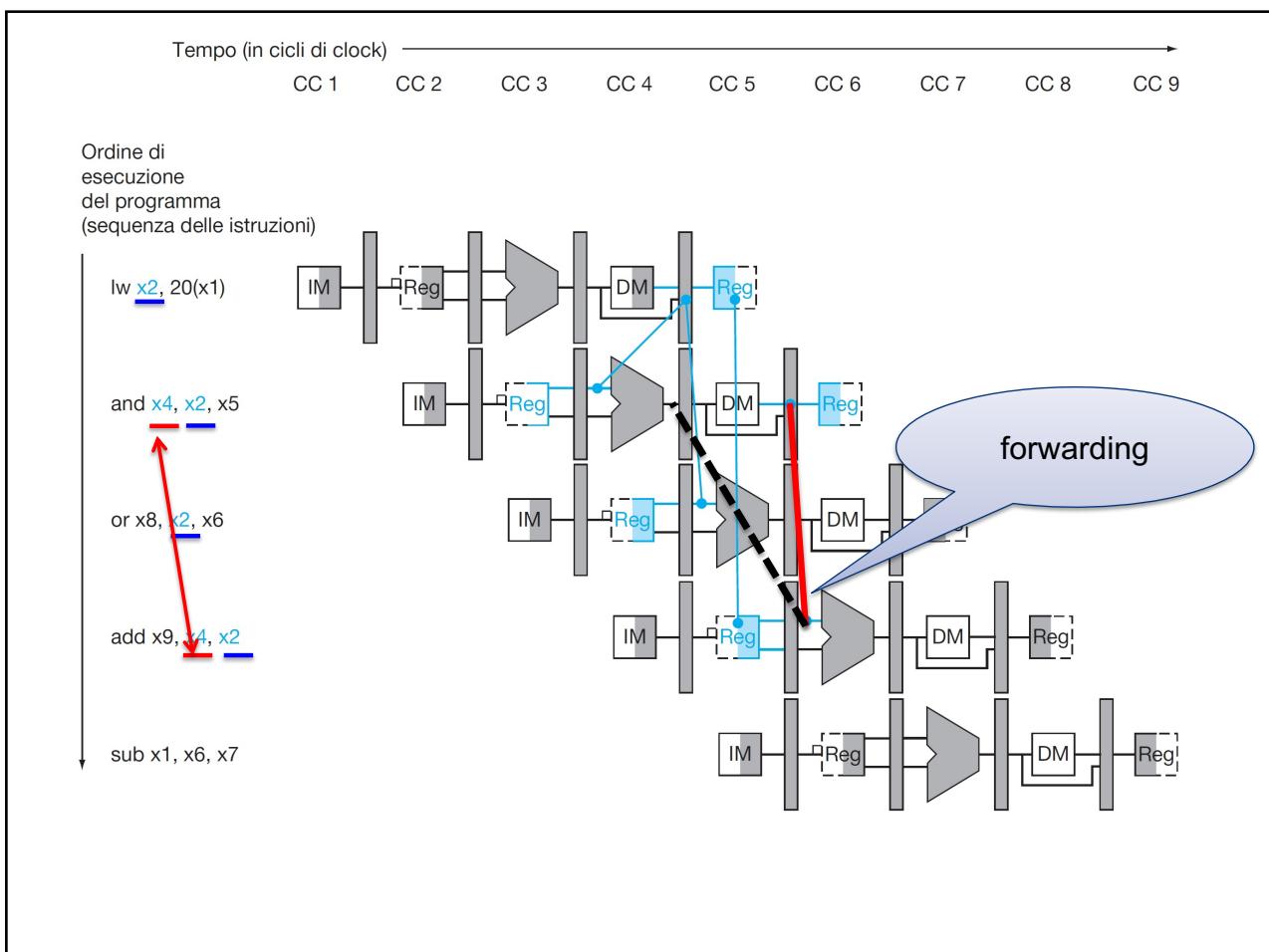
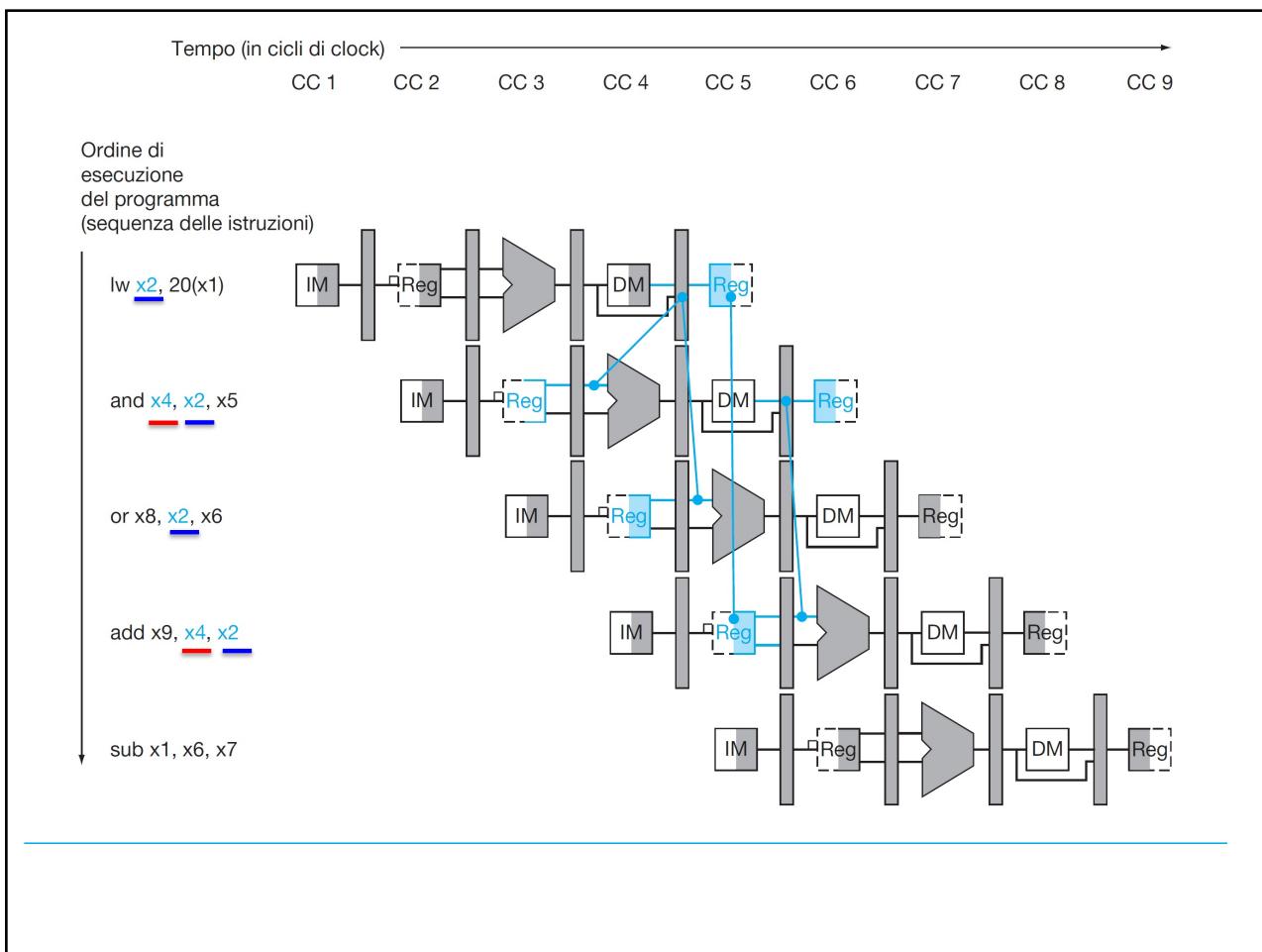
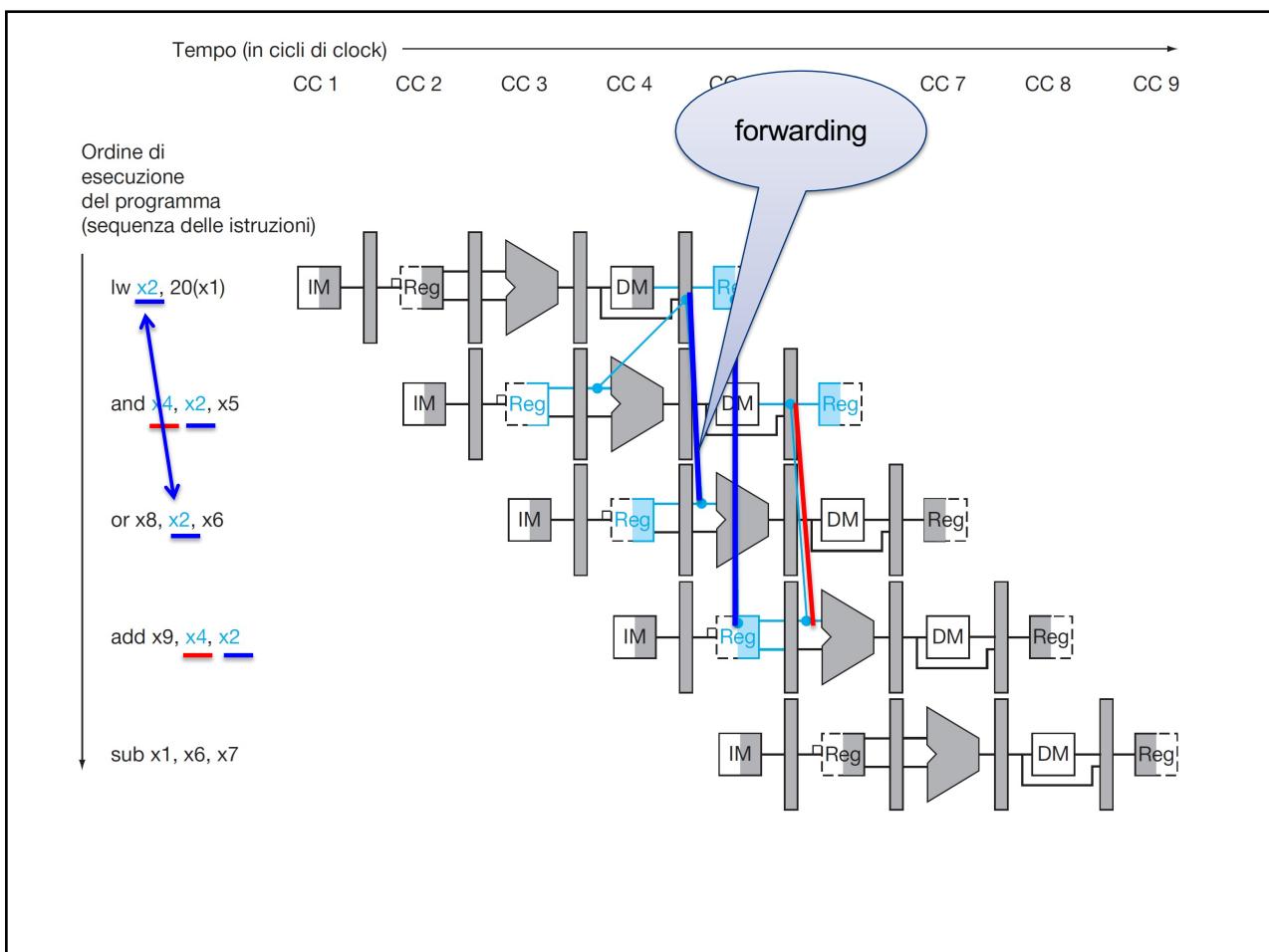
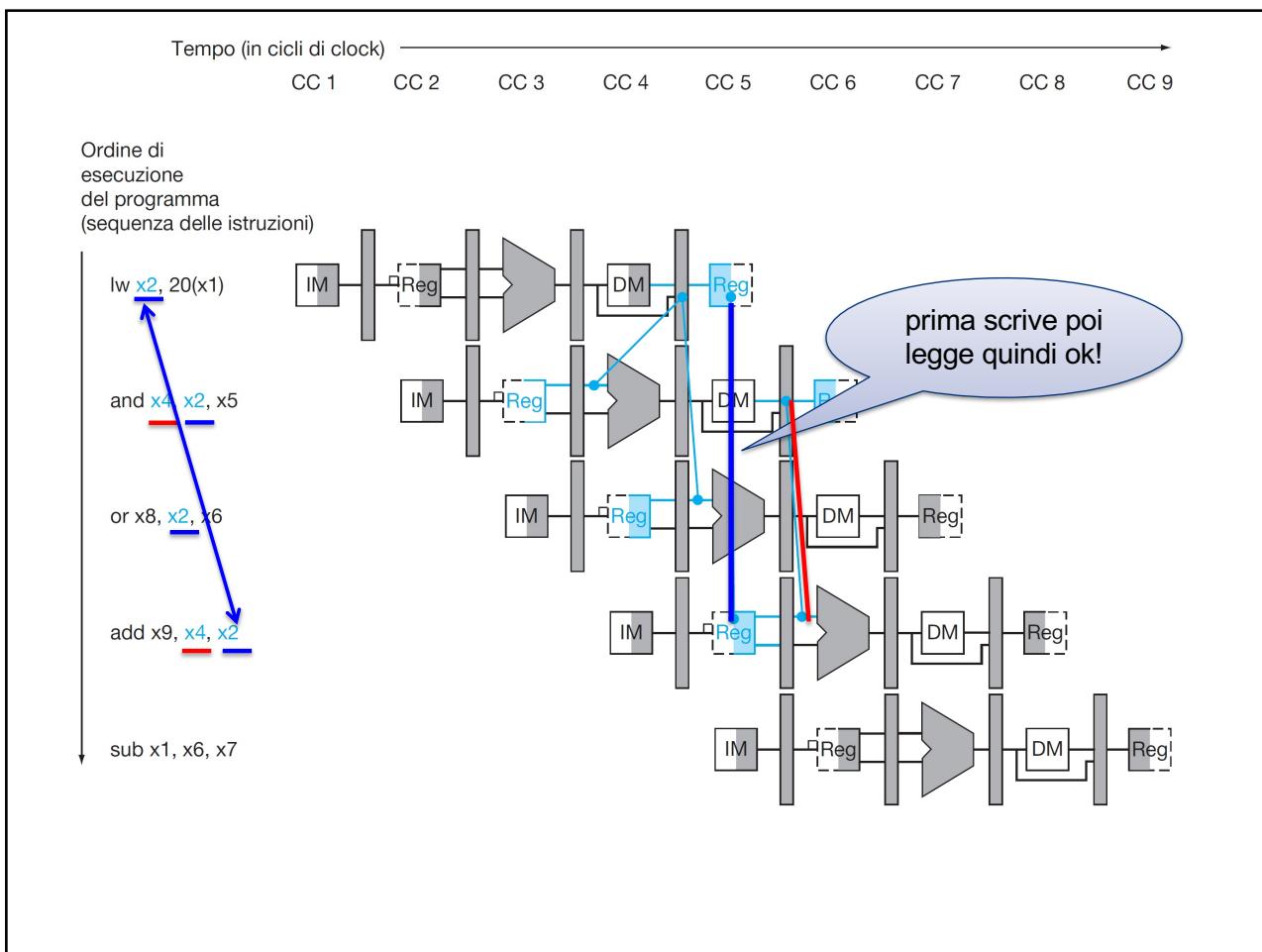
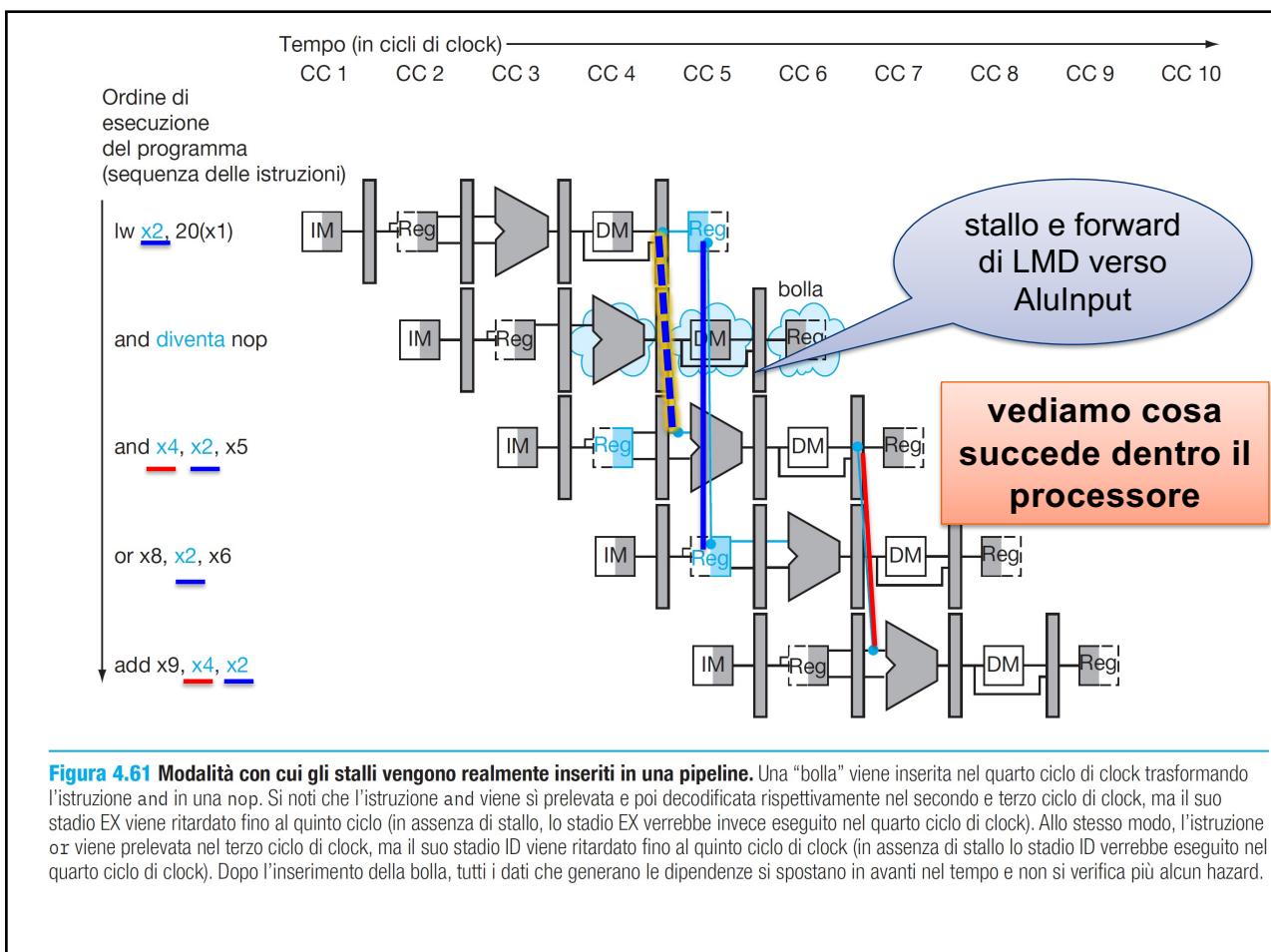
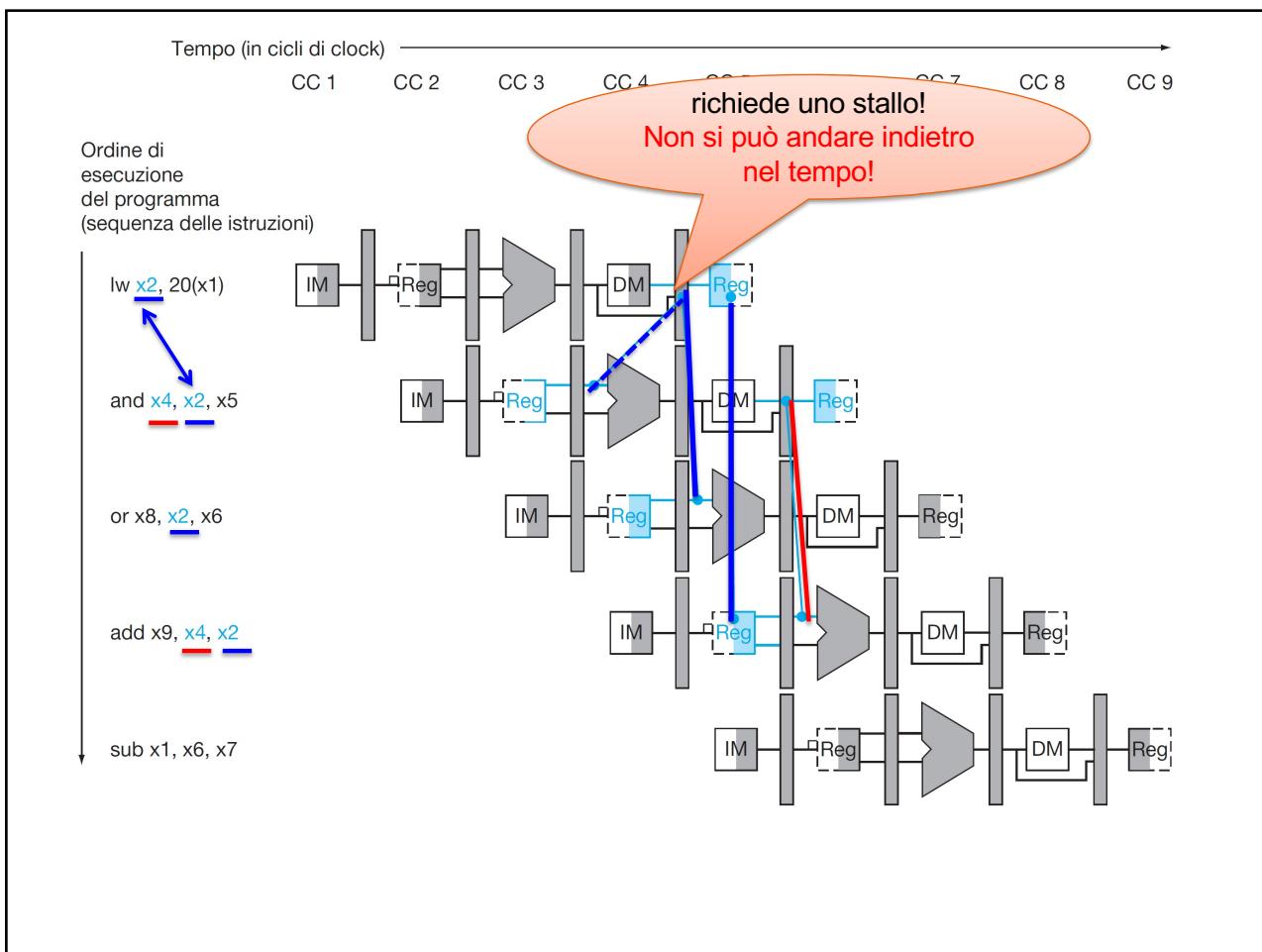


Figura 4.62 Panoramica sul controllo della pipeline: nella figura compaiono i multiplexer per la propagazione, l'unità di rilevamento degli hazard e l'unità di propagazione. Sebbene gli stadi ID ed EX siano stati semplificati (mancano la logica per l'estensione del segno del campo







Pipeline RISC-V: esempio stallo

and x4, x2, x5 | lw x2, 20(x1) | before<1> | before<2> | before<3>.

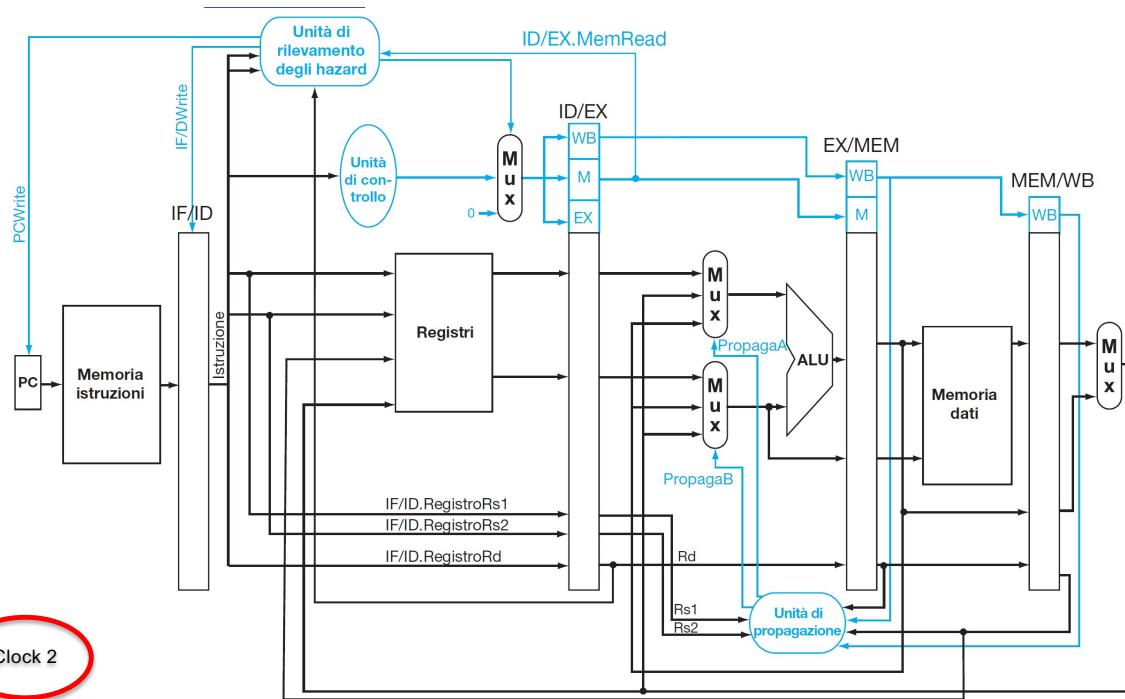


Figura 4.62 Panoramica sul controllo della pipeline: nella figura compaiono i due multiplexer per la propagazione, l'unità di rilevamento degli hazard e l'unità di propagazione. Sembra gli stadi ID ed EX siano stati semplificati (mancano la logica per l'estensione del campo del campo

IF/ID.IR[rs1] == ID/EX.IR[rd] = x2 quindi **imposta Stall a 1** e segnali per Unità di Forward

or x8, x2, x6 | and x4, x2, x5 | lw x2, 20(x1) | before<1> | before<2>

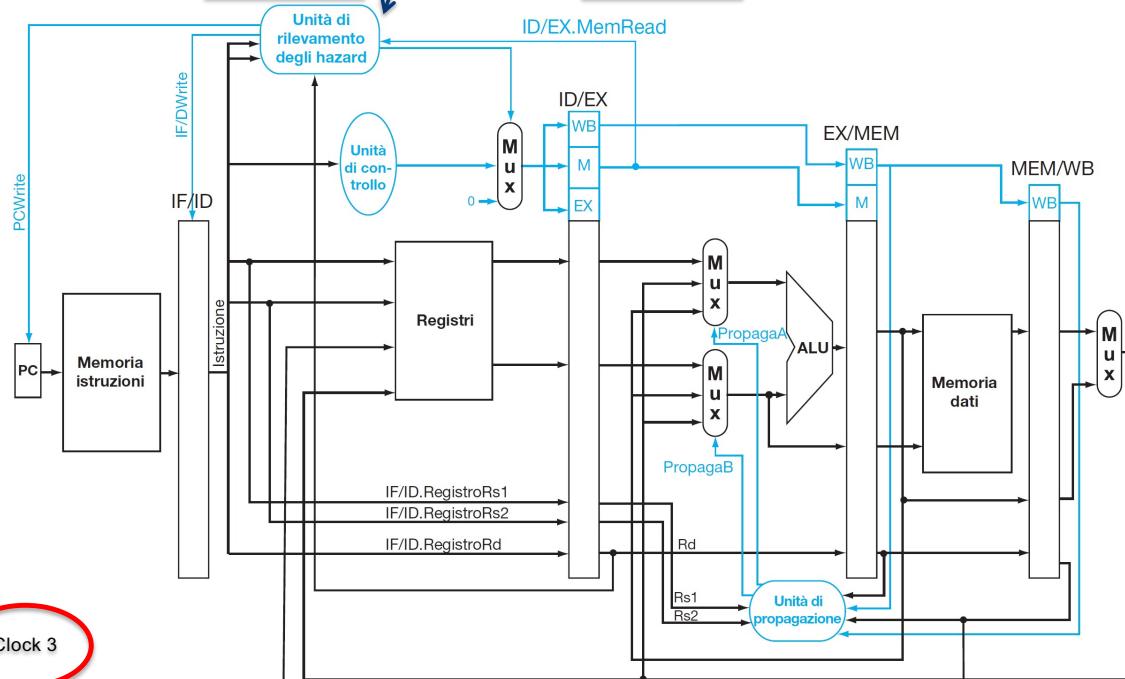


Figura 4.62 Panoramica sul controllo della pipeline: nella figura compaiono i due multiplexer per la propagazione, l'unità di rilevamento degli hazard e l'unità di propagazione. Sembra gli stadi ID ed EX siano stati semplificati (mancano la logica per l'estensione del campo del campo

IF e ID sono ripetute (è stato impedito l'aggiornamento)

or x8, x2, x6 | and x4, x2, x5

| bubble

|. lw x2, 20(x1) | before<1>

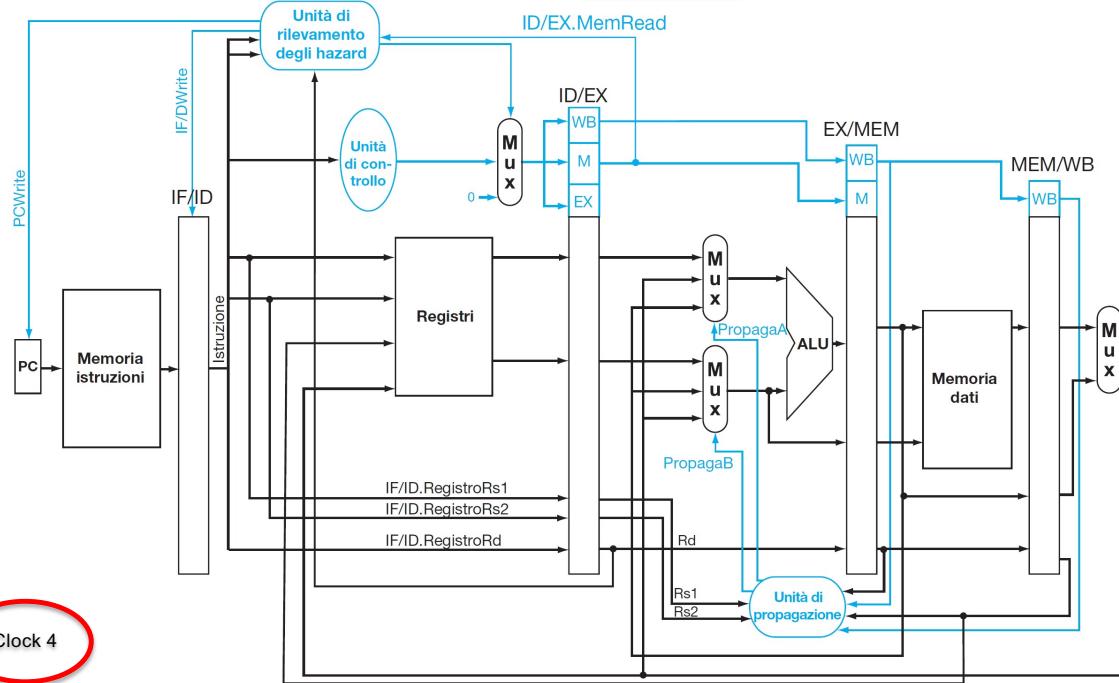


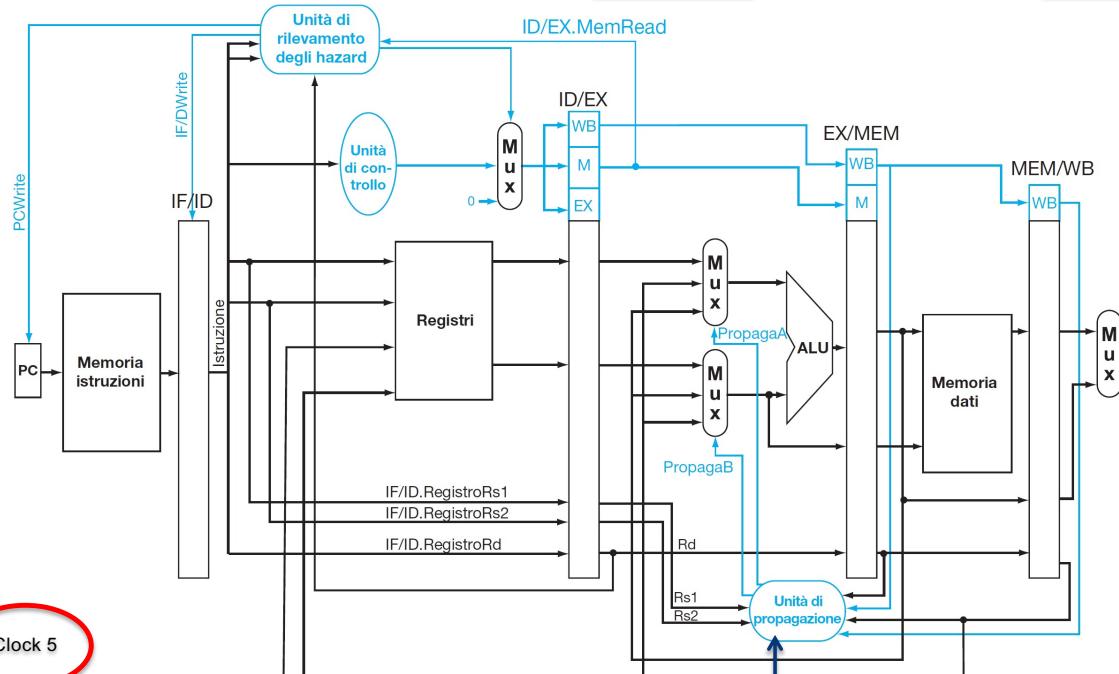
Figura 4.62 Panoramica sul controllo della pipeline: nella figura compaiono i due multiplexer per la propagazione, l'unità di rilevamento degli hazard e l'unità di propagazione. Sembra gli stadi ID ed EX siano stati semplificati (mancano la logica per l'estensione del campo del campo

add x9, x4, x2 | or x8, x2, x6

| and x4, x2, x5

| bubble

|. lw x2, 20(x1)



MEM/WB.LMD propagato a TopAlu Input

anche RAW su x2 ma nessun problema (no forward):
prima lw scrive poi or legge

