



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Architettura degli Elaboratori

Lez. 9 – CU e nuove istruzioni

Prof. Andrea Sterbini – sterbini@di.uniroma1.it



Argomenti

► Argomenti della lezione

- Soluzione esercizio per casa
- Modifiche alla CPU MIPS a 1 colpo di clock
 - Come aggiungere altre istruzioni:
 - l'istruzione **J** (Jump)
 - Unità funzionali necessarie
 - Datapath e modifiche all'unità di controllo
 - L'istruzione **jal** (Jump and Link)
 - L'istruzione **jr** (Jump to Register)
 - L'istruzione **addi** (add immediate)

Soluzione

- ▶ Se i codici dei 4 tipi di istruzioni sono:

istruzione	codice decimale	in binario
di tipo R	0	000000
lw	35	100011
sw	43	101011
beq	4	000100

- ▶ ... e dobbiamo produrre i segnali dell'unità di controllo

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

Soluzione

- ▶ Se i codici dei 4 tipi di istruzioni sono:

istruzione	codice decimale	in binario
di tipo R	0	000000
lw	35	100011
sw	43	101011
beq	4	000100

- ▶ ... e dobbiamo produrre i segnali dell'unità di controllo

Istruzione	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
000000	1	0	0	1	0 (X)	0	0	1	0
100011	0	1	1	1	1	0	0	0	0
101011	X	1	X	0	0 (X)	1	0	0	0
000100	X	0	X	0	0 (X)	0	1	0	1

Soluzione

- ▶ Se i codici dei 4 tipi di istruzioni sono:

istruzione	codice decimale	in binario
di tipo R	0	000000
lw	35	100011
sw	43	101011
beq	4	000100

- ▶ ... e dobbiamo produrre i segnali dell'unità di controllo

Istruzione	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
000000	1	0	0	1	0 (X)	0	0	1	0
100011	0	1	1	1	1	0	0	0	0
101011	X	1	X	0	0 (X)	1	0	0	0
000100	X	0	X	0	0 (X)	0	1	0	1

- ▶ Da cui possiamo produrre la PLA oppure le funzioni booleane necessarie

Soluzione

- ▶ Se i codici dei 4 tipi di istruzioni sono:

istruzione	codice decimale	in binario
di tipo R	0	000000
lw	35	100011
sw	43	101011
beq	4	000100

- ▶ ... e dobbiamo produrre i segnali dell'unità di controllo

Istruzione	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
000000	1	0	0	1	0 (X)	0	0	1	0
100011	0	1	1	1	1	0	0	0	0
101011	X	1	X	0	0 (X)	1	0	0	0
000100	X	0	X	0	0 (X)	0	1	0	1

- ▶ Da cui possiamo produrre la PLA oppure le funzioni booleane necessarie
- ▶ Per esempio

Soluzione

- ▶ Se i codici dei 4 tipi di istruzioni sono:

istruzione	codice decimale	in binario
di tipo R	0	000000
lw	35	100011
sw	43	101011
beq	4	000100

- ▶ ... e dobbiamo produrre i segnali dell'unità di controllo

Istruzione	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
000000	1	0	0	1	0 (X)	0	0	1	0
100011	0	1	1	1	1	0	0	0	0
101011	X	1	X	0	0 (X)	1	0	0	0
000100	X	0	X	0	0 (X)	0	1	0	1

- ▶ Per esempio

- ▶ **ALUSrc = Opcode_0**

- ▶ **Branch = Opcode_2**

- ▶ **MemWrite = Opcode_3**

Aggiungere una nuova istruzione

- ▶ Supponiamo di voler aggiungere la nuova istruzione, **J** (Jump), dobbiamo:
 - Definire la sua **codifica**
 - Definire **cosa fa**
 - Individuare le **unità funzionali necessarie** (e se sono già presenti)
 - Individuare i **flussi delle informazione** necessarie
 - Individuare i **segnali di controllo** necessari
 - Calcolare il **tempo necessario** per la nuova istruzione e se modifica il tempo totale

Aggiungere una nuova istruzione

- ▶ Supponiamo di voler aggiungere la nuova istruzione, **J** (Jump), dobbiamo:
 - Definire la sua **codifica**
 - Definire **cosa fa**
 - Individuare le **unità funzionali necessarie** (e se sono già presenti)
 - Individuare i **flussi delle informazione** necessarie
 - Individuare i **segnali di controllo** necessari
 - Calcolare il **tempo necessario** per la nuova istruzione e se modifica il tempo totale
- ▶ Supponiamo che abbia la **codifica** seguente (formato J)

Campo	000010	indirizzo
Posizione dei bit	31-26	25:0

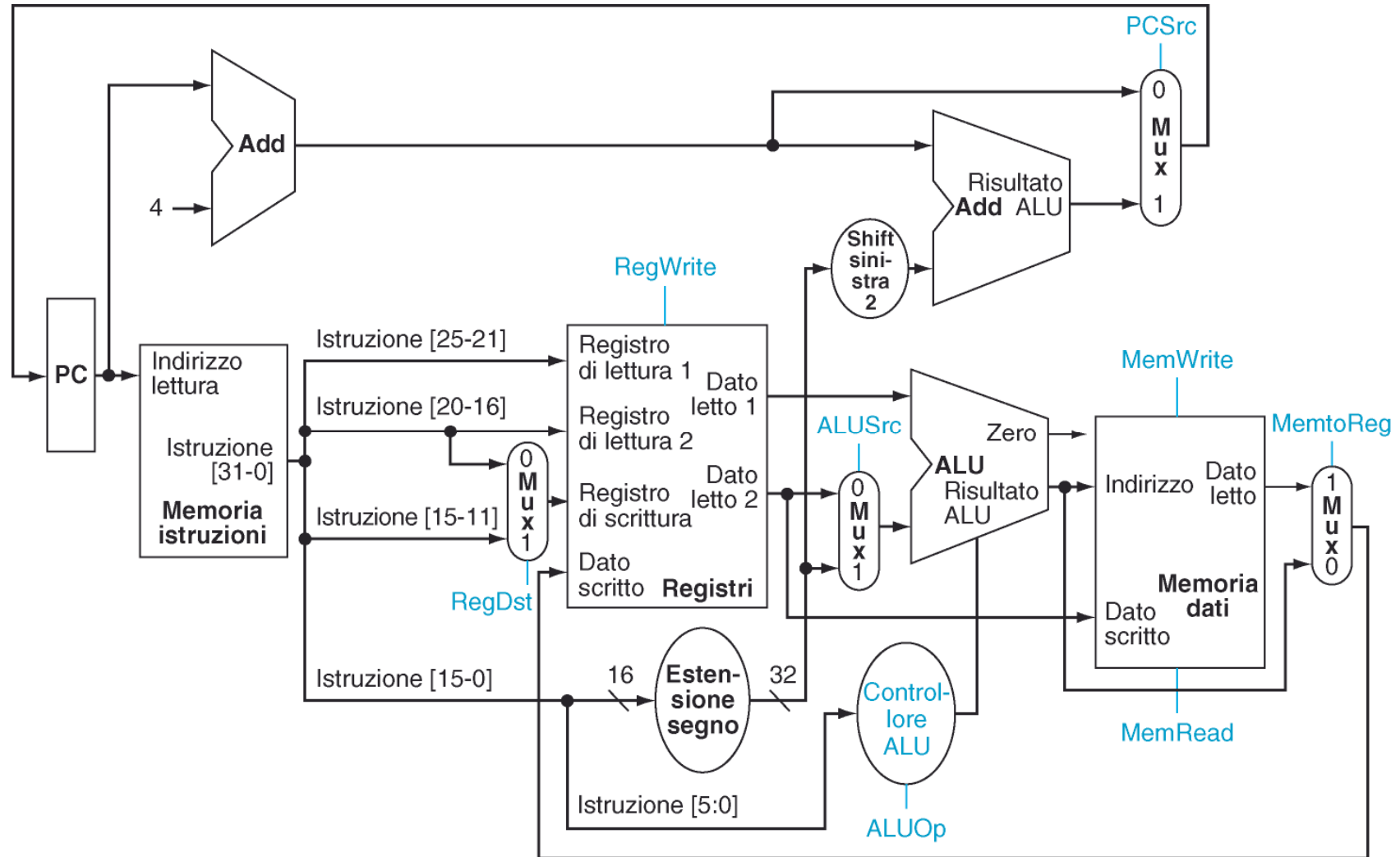
Aggiungere una nuova istruzione

- ▶ Supponiamo di voler aggiungere la nuova istruzione, **J** (Jump), dobbiamo:
 - Definire la sua **codifica**
 - Definire **cosa fa**
 - Individuare le **unità funzionali necessarie** (e se sono già presenti)
 - Individuare i **flussi delle informazione** necessarie
 - Individuare i **segnali di controllo** necessari
 - Calcolare il **tempo necessario** per la nuova istruzione e se modifica il tempo totale
- ▶ Supponiamo che abbia la **codifica** seguente (formato J)

Campo	000010	indirizzo
Posizione dei bit	31-26	25:0

- ▶ ... e che il campo da 26 bit in essa contenuto sia l'**istruzione di destinazione** del salto:
 - è un **indirizzo assoluto** (invece che uno relativo al PC come per i branch)
 - indica l'**istruzione di destinazione** (va moltiplicato per 4 perché le istr. sono «allineate»)
 - i 4 bit «mancanti» verranno presi dal PC+4 (ovvero si rimane nello stesso blocco di 256G)
 - (per i salti tra blocchi diversi sarà necessario introdurre l'istruzione **jr**)

CPU (senza Jump)



Aggiungere il Jump

Cosa fa: $PC \leftarrow (\text{shift left di 2 bit di Istruzione}[25..0]) \text{ OR } (PC+4)[31..28]$

Aggiungere il Jump

Cosa fa: $PC \leftarrow (\text{shift left di 2 bit di Istruzione}[25..0]) \text{ OR } (PC+4)[31..28]$

Unità funzionali:

Aggiungere il Jump

Cosa fa: $PC \leftarrow (\text{shift left di 2 bit di Istruzione}[25..0]) \text{ OR } (PC+4)[31..28]$

Unità funzionali:

PC + 4	(già presente)
shift left di 2	(da aggiungere)
OR dei 28 bit ottenuti con i 4 del PC+4	(si ottiene dalle connessioni)
MUX per selezionare il nuovo PC	(da aggiungere)

Aggiungere il Jump

Cosa fa: $PC \leftarrow (\text{shift left di 2 bit di Istruzione}[25..0]) \text{ OR } (PC+4)[31..28]$

Unità funzionali:

PC + 4	(già presente)
shift left di 2	(da aggiungere)
OR dei 28 bit ottenuti con i 4 del PC+4	(si ottiene dalle connessioni)
MUX per selezionare il nuovo PC	(da aggiungere)

Flusso dei dati:

Aggiungere il Jump

Cosa fa: $PC \leftarrow (\text{shift left di 2 bit di Istruzione}[25..0]) \text{ OR } (PC+4)[31..28]$

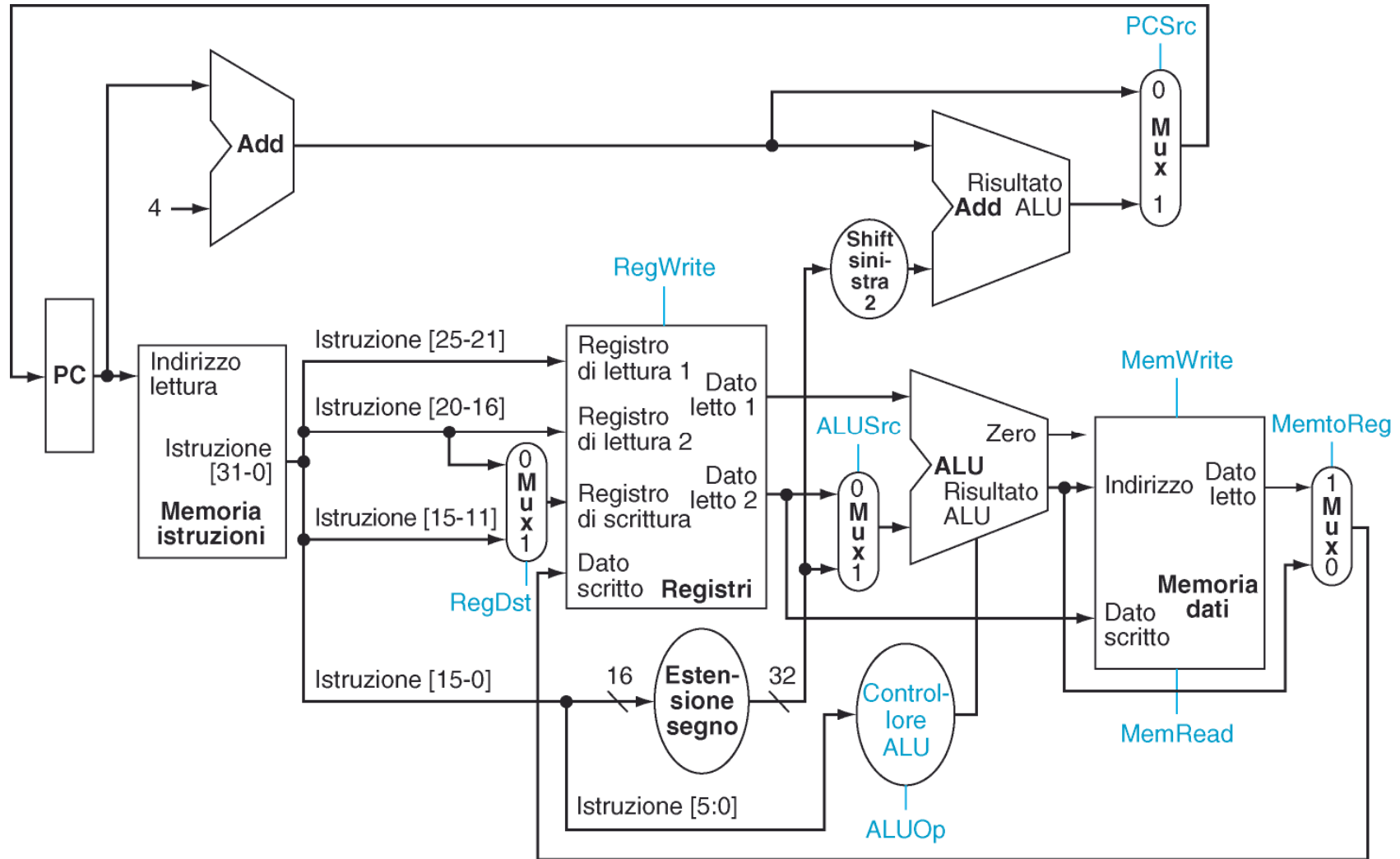
Unità funzionali:

PC + 4	(già presente)
shift left di 2	(da aggiungere)
OR dei 28 bit ottenuti con i 4 del PC+4	(si ottiene dalle connessioni)
MUX per selezionare il nuovo PC	(da aggiungere)

Flusso dei dati:

Destinazione ---> **SL2** ---> **(OR i 4 bit di PC+4)** ---> **MUX** ---> **PC**

Aggiungere il Jump



Aggiungere il Jump

Cosa fa: $PC \leftarrow (\text{shift left di 2 bit di Istruzione}[25..0]) \text{ OR } (PC+4)[31..28]$

Unità funzionali:

PC + 4	(già presente)
shift left di 2	(da aggiungere)
OR dei 28 bit ottenuti con i 4 del PC+4	(si ottiene dalle connessioni)
MUX per selezionare il nuovo PC	(da aggiungere)

Flusso dei dati:

Destinazione \rightarrow **SL2** \rightarrow **(OR i 4 bit di PC+4)** \rightarrow **MUX** \rightarrow **PC**

Aggiungere il Jump

Cosa fa: $PC \leftarrow (\text{shift left di 2 bit di Istruzione}[25..0]) \text{ OR } (PC+4)[31..28]$

Unità funzionali:

PC + 4	(già presente)
shift left di 2	(da aggiungere)
OR dei 28 bit ottenuti con i 4 del PC+4	(si ottiene dalle connessioni)
MUX per selezionare il nuovo PC	(da aggiungere)

Flusso dei dati:

Destinazione \rightarrow **SL2** \rightarrow **(OR i 4 bit di PC+4)** \rightarrow **MUX** \rightarrow **PC**

Segnali di controllo:

Aggiungere il Jump

Cosa fa: $PC \leftarrow (\text{shift left di 2 bit di Istruzione}[25..0]) \text{ OR } (PC+4)[31..28]$

Unità funzionali:

PC + 4	(già presente)
shift left di 2	(da aggiungere)
OR dei 28 bit ottenuti con i 4 del PC+4	(si ottiene dalle connessioni)
MUX per selezionare il nuovo PC	(da aggiungere)

Flusso dei dati:

Destinazione ---> **SL2** ---> **(OR i 4 bit di PC+4)** ---> **MUX** ---> **PC**

Segnali di controllo:

Jump = 1 che seleziona la nuova destinazione sul MUX

RegWrite=0 e **MemWrite=0** per evitare modifiche a registri e MEM

Aggiungere il Jump

Cosa fa: $PC \leftarrow (\text{shift left di 2 bit di Istruzione}[25..0]) \text{ OR } (PC+4)[31..28]$

Unità funzionali:

PC + 4	(già presente)
shift left di 2	(da aggiungere)
OR dei 28 bit ottenuti con i 4 del PC+4	(si ottiene dalle connessioni)
MUX per selezionare il nuovo PC	(da aggiungere)

Flusso dei dati:

Destinazione ---> **SL2** ---> **(OR i 4 bit di PC+4)** ---> **MUX** ---> **PC**

Segnali di controllo:

Jump = 1 che seleziona la nuova destinazione sul MUX

RegWrite=0 e **MemWrite=0** per evitare modifiche a registri e MEM

Tempo necessario:

Aggiungere il Jump

Cosa fa: $PC \leftarrow (\text{shift left di 2 bit di Istruzione}[25..0]) \text{ OR } (PC+4)[31..28]$

Unità funzionali:

PC + 4	(già presente)
shift left di 2	(da aggiungere)
OR dei 28 bit ottenuti con i 4 del PC+4	(si ottiene dalle connessioni)
MUX per selezionare il nuovo PC	(da aggiungere)

Flusso dei dati:

Destinazione ---> **SL2** ---> **(OR i 4 bit di PC+4)** ---> **MUX** ---> **PC**

Segnali di controllo:

Jump = 1 che seleziona la nuova destinazione sul MUX

RegWrite=0 e **MemWrite=0** per evitare modifiche a registri e MEM

Tempo necessario:

Fetch

Aggiungere il Jump

Cosa fa: $PC \leftarrow (\text{shift left di 2 bit di Istruzione}[25..0]) \text{ OR } (PC+4)[31..28]$

Unità funzionali:

PC + 4	(già presente)
shift left di 2	(da aggiungere)
OR dei 28 bit ottenuti con i 4 del PC+4	(si ottiene dalle connessioni)
MUX per selezionare il nuovo PC	(da aggiungere)

Flusso dei dati:

Destinazione ---> **SL2** ---> **(OR i 4 bit di PC+4)** ---> **MUX** ---> **PC**

Segnali di controllo:

Jump = 1 che seleziona la nuova destinazione sul MUX

RegWrite=0 e **MemWrite=0** per evitare modifiche a registri e MEM

Tempo necessario:

Fetch

e in parallelo il tempo dell'adder che calcola **PC+4** (quindi il massimo tra i due tempi)

Aggiungere il Jump

Cosa fa: $PC \leftarrow (\text{shift left di 2 bit di Istruzione}[25..0]) \text{ OR } (PC+4)[31..28]$

Unità funzionali:

PC + 4	(già presente)
shift left di 2	(da aggiungere)
OR dei 28 bit ottenuti con i 4 del PC+4	(si ottiene dalle connessioni)
MUX per selezionare il nuovo PC	(da aggiungere)

Flusso dei dati:

Destinazione ---> **SL2** ---> **(OR i 4 bit di PC+4)** ---> **MUX** ---> **PC**

Segnali di controllo:

Jump = 1 che seleziona la nuova destinazione sul MUX

RegWrite=0 e **MemWrite=0** per evitare modifiche a registri e MEM

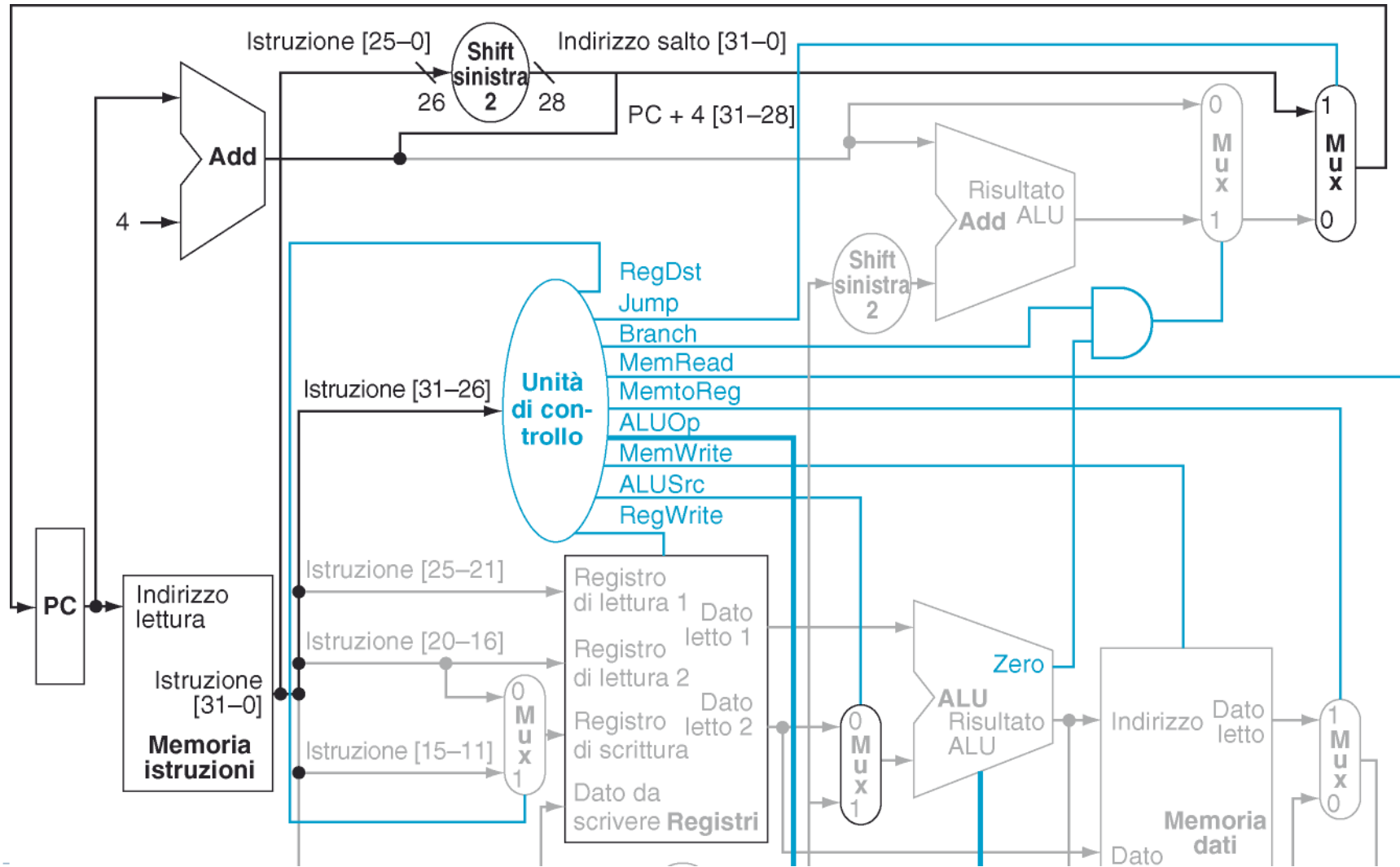
Tempo necessario:

Fetch

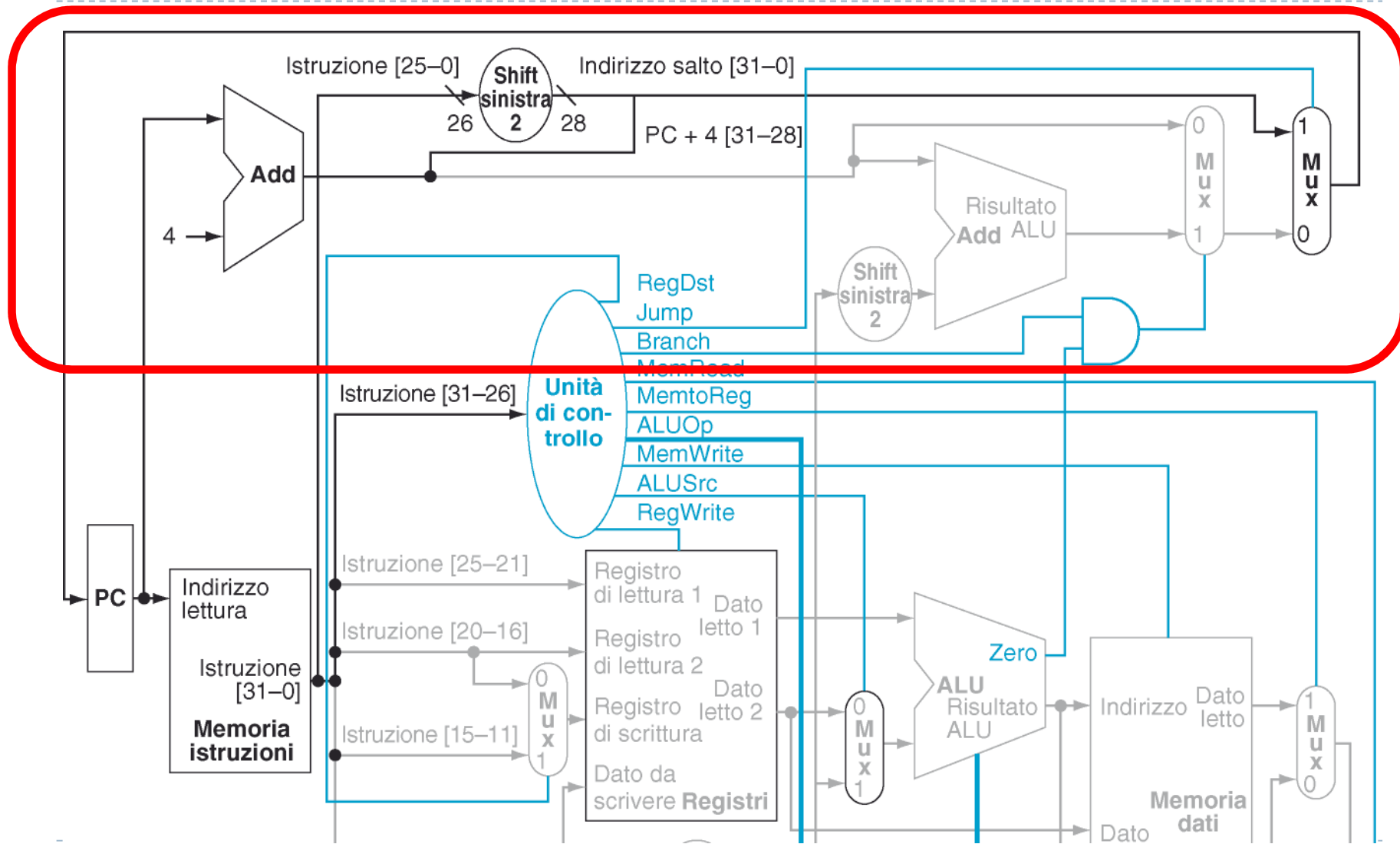
e in parallelo il tempo dell'adder che calcola **PC+4** (quindi il massimo tra i due tempi)

- ▶ **NOTA:** l'hardware necessario al calcolo della destinazione del salto è sempre presente e calcola la destinazione anche se l'istruzione non è un Jump. Solo se la CU riconosce che è un Jump il valore calcolato viene immesso nel PC per passare (al colpo di clock successivo) alla destinazione del salto.
- ▶ **Obiettivo:** una CPU più veloce possibile a costo di usare più componenti hardware

Modifiche per il Jump



Modifiche per il Jump



JAL (Jump and Link)

Come è codificata:

di **tipo J**

(come Jump)

JAL (Jump and Link)

Come è codificata: di **tipo J** (come Jump)

Cosa fa: **PC ← SL2(indirizzo) OR (PC+4)[31..28]** come Jump

\$ra ← PC+4

JAL (Jump and Link)

Come è codificata: di **tipo J** (come Jump)

Cosa fa: **PC ← SL2(indirizzo) OR (PC+4)[31..28]** come Jump

\$ra ← PC+4

Unità funzionali: **le stesse del Jump**

più **MUX** per selezionare il valore di PC+4 come valore di destinazione

più **MUX** per selezionare il numero del registro **\$ra** come destinazione

JAL (Jump and Link)

Come è codificata: di **tipo J** (come Jump)

Cosa fa: **$PC \leftarrow SL2(\text{indirizzo}) \text{ OR } (PC+4)[31..28]$** come Jump
 $\$ra \leftarrow PC+4$

Unità funzionali: **le stesse del Jump**

più **MUX** per selezionare il valore di PC+4 come valore di destinazione

più **MUX** per selezionare il numero del registro **\$ra** come destinazione

Flusso dei dati: **lo stesso del Jump**, inoltre

PC+4 → **MUX** → **Registri (dato da memorizzare)**

31 → **MUX** → **Registri(#registro destinazione)**

JAL (Jump and Link)

Come è codificata: di **tipo J** (come Jump)

Cosa fa: **PC** \leftarrow **SL2(indirizzo) OR (PC+4)[31..28]** come Jump
\$ra \leftarrow **PC+4**

Unità funzionali: **le stesse del Jump**

più **MUX** per selezionare il valore di PC+4 come valore di destinazione

più **MUX** per selezionare il numero del registro **\$ra** come destinazione

Flusso dei dati: **lo stesso del Jump**, inoltre

PC+4 \rightarrow **MUX** \rightarrow **Registri (dato da memorizzare)**

31 \rightarrow **MUX** \rightarrow **Registri(#registro destinazione)**

Segnali di controllo: il segnale **Jump** deve essere asserito

la **CU** deve produrre un segnale **Link** per attivare i due nuovi MUX

JAL (Jump and Link)

Come è codificata: di **tipo J** (come Jump)

Cosa fa: **PC \leftarrow SL2(indirizzo) OR (PC+4)[31..28]** come Jump
\$ra \leftarrow PC+4

Unità funzionali: **le stesse del Jump**

più **MUX** per selezionare il valore di PC+4 come valore di destinazione

più **MUX** per selezionare il numero del registro **\$ra** come destinazione

Flusso dei dati: **lo stesso del Jump**, inoltre

PC+4 \rightarrow MUX \rightarrow Registri (dato da memorizzare)

31 \rightarrow MUX \rightarrow Registri(#registro destinazione)

Segnali di controllo: il segnale **Jump** deve essere asserito

la **CU** deve produrre un segnale **Link** per attivare i due nuovi MUX

- ▶ Tempo necessario: il WB deve avvenire dopo che sono finiti sia il Fetch (per leggere l'istruzione) sia il calcolo di PC+4 (che va memorizzato in \$ra) per cui possono presentarsi due casi

JAL (Jump and Link)

Come è codificata: di **tipo J** (come Jump)

Cosa fa: **$PC \leftarrow SL2(\text{indirizzo}) \text{ OR } (PC+4)[31..28]$** come Jump
 $\$ra \leftarrow PC+4$

Unità funzionali: **le stesse del Jump**

più **MUX** per selezionare il valore di PC+4 come valore di destinazione

più **MUX** per selezionare il numero del registro **\$ra** come destinazione

Flusso dei dati: **lo stesso del Jump**, inoltre

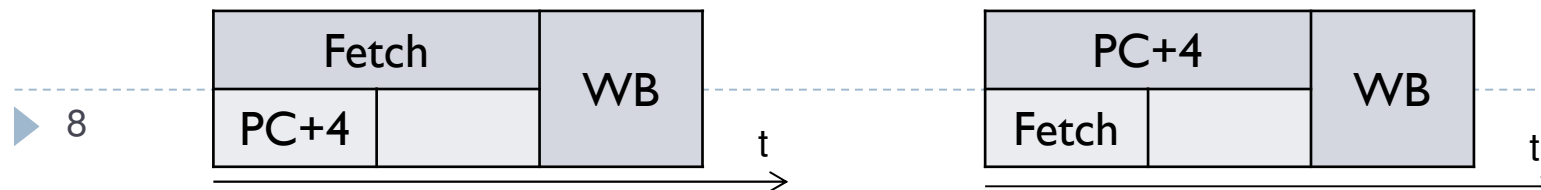
PC+4 → **MUX** → **Registri (dato da memorizzare)**

31 → **MUX** → **Registri(#registro destinazione)**

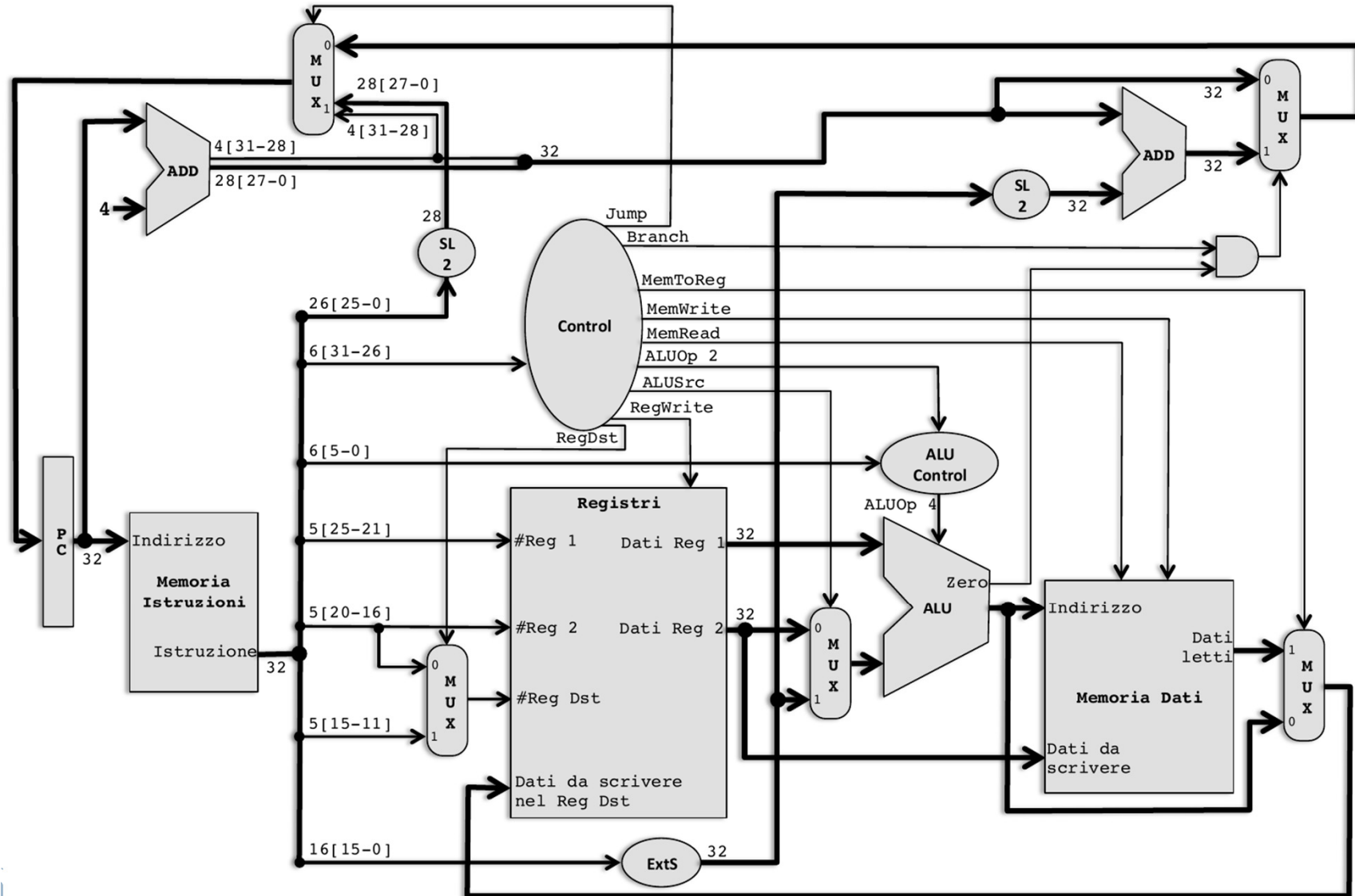
Segnali di controllo: il segnale **Jump** deve essere asserito

la **CU** deve produrre un segnale **Link** per attivare i due nuovi MUX

- ▶ Tempo necessario: il WB deve avvenire dopo che sono finiti sia il Fetch (per leggere l'istruzione) sia il calcolo di PC+4 (che va memorizzato in \$ra) per cui possono presentarsi due casi



JAL



Istruzione addi/la

Assembly: **addi** **rt, rs, costante** **(add immediate)** **di tipo I**
Cosa fa: Somma la parte immediata al registro **rs** e mette il risultato in **rt**

Istruzione addi/la

Assembly: addi	rt, rs, costante	(add immediate)	di tipo I
Cosa fa:	Somma la parte immediata al registro rs e mette il risultato in rt		
Unità funzionali:	ALU per la somma		(presente)
	MUX che seleziona la parte immediata come secondo arg		(presente)
	Estensione del segno della parte immediata		(presente)

Istruzione addi/la

Assembly: addi	rt, rs, costante	(add immediate)	di tipo I
Cosa fa:	Somma la parte immediata al registro rs e mette il risultato in rt		
Unità funzionali:	ALU per la somma		(presente)
	MUX che seleziona la parte immediata come secondo arg		(presente)
	Estensione del segno della parte immediata		(presente)
Flusso dei dati:	Registri[rs] → ALU		
	Immediata → EstSegno → ALU		
	ALU → Registri[rt]		

Istruzione addi/la

Assembly: addi	rt, rs, costante	(add immediate)	di tipo I
Cosa fa:	Somma la parte immediata al registro rs e mette il risultato in rt		
Unità funzionali:	ALU per la somma		(presente)
	MUX che seleziona la parte immediata come secondo arg		(presente)
	Estensione del segno della parte immediata		(presente)
Flusso dei dati:	Registri[rs] → ALU		
	Immediata → EstSegno → ALU		
	ALU → Registri[rt]		
Si comporta come la	lw rt, costante(rs)	<u>che memorizza l'indirizzo invece che il dato</u>	

Istruzione addi/la

Assembly: addi	rt, rs, costante	(add immediate)	di tipo I
Cosa fa:	Somma la parte immediata al registro rs e mette il risultato in rt		
Unità funzionali:	ALU per la somma		(presente)
	MUX che seleziona la parte immediata come secondo arg		(presente)
	Estensione del segno della parte immediata		(presente)
Flusso dei dati:	Registri[rs] → ALU		
	Immediata → EstSegno → ALU		
	ALU → Registri[rt]		
Si comporta come la	lw rt, costante(rs)	<u>che memorizza l'indirizzo invece che il dato</u>	
ovvero come la istruzione	la rt, costante(rs)	(load address). La CU produce i segnali:	

Istruzione addi/la

Assembly: **addi** **rt, rs, costante** (add immediate) di tipo I
 Cosa fa: Somma la parte immediata al registro **rs** e mette il risultato in **rt**
 Unità funzionali: **ALU** per la somma (presente)
MUX che seleziona la parte immediata come secondo arg (presente)
Estensione del segno della parte immediata (presente)

Flusso dei dati: Registri[rs] → ALU
 Immediate → EstSegno → ALU
 ALU → Registri[rt]

Si comporta come la **lw rt, costante(rs)** che memorizza l'indirizzo invece che il dato
 ovvero come la istruzione **la rt, costante(rs)** (load address). La CU produce i segnali:

Istruzione	Reg Dst	ALU Src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
addi \$rd, \$rs, costante	0	1	0	1	X	0	0	0	0	0

Istruzione addi/la

Assembly: **addi** **rt, rs, costante** **(add immediate)** **di tipo I**

Cosa fa: Somma la parte immediata al registro **rs** e mette il risultato in **rt**

Unità funzionali: **ALU** per la somma (presente)

MUX che seleziona la parte immediata come secondo arg (presente)

Estensione del segno della parte immediata (presente)

Flusso dei dati: Registri[rs] → ALU

 Immediate → EstSegno → ALU

 ALU → Registri[rt]

Si comporta come la **lw rt, costante(rs)** che memorizza l'indirizzo invece che il dato

ovvero come la istruzione **la rt, costante(rs)** (load address). La CU produce i segnali:

Istruzione	Reg Dst	ALU Src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU OpI	ALU Op0
addi \$rd, \$rs, costante	0	1	0	1	X	0	0	0	0	0

Istruzione addi/la

Assembly: **addi rt, rs, costante** (add immediate) di tipo I

Cosa fa: Somma la parte immediata al registro **rs** e mette il risultato in **rt**

Unità funzionali: **ALU** per la somma (presente)
MUX che seleziona la parte immediata come secondo arg (presente)
Estensione del segno della parte immediata (presente)

Flusso dei dati: Registri[rs] → ALU
 Immediate → EstSegno → ALU
 ALU → Registri[rt]

Si comporta come la **lw rt, costante(rs)** che memorizza l'indirizzo invece che il dato
 ovvero come la istruzione **la rt, costante(rs)** (load address). La CU produce i segnali:

Istruzione	Reg Dst	ALU Src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
addi \$rd, \$rs, costante	0	1	0	1	X	0	0	0	0	0

Istruzione addi/la

Assembly: **addi** **rt, rs, costante** (**add immediate**) **di tipo I**

Cosa fa: Somma la parte immediata al registro **rs** e mette il risultato in **rt**

Unità funzionali: **ALU** per la somma (presente)

MUX che seleziona la parte immediata come secondo arg (presente)

Estensione del segno della parte immediata (presente)

Flusso dei dati: Registri[rs] → ALU

 Immediate → EstSegno → ALU

 ALU → Registri[rt]

Si comporta come la **lw rt, costante(rs)** che memorizza l'indirizzo invece che il dato

ovvero come la istruzione **la rt, costante(rs)** (load address). La CU produce i segnali:

Istruzione	Reg Dst	ALU Src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU OpI	ALU Op0
addi \$rd, \$rs, costante	0	1	0	1	X	0	0	0	0	0

Tempo necessario: **come una istruzione di tipo R**

Istruzione jr (Jump to Register)

L'istruzione **jr rs** è di formato **R**

Cosa fa: trasferisce in **PC** il contenuto del registro **rs**

Unità funzionali:

MUX per selezionare il PC dal blocco registri

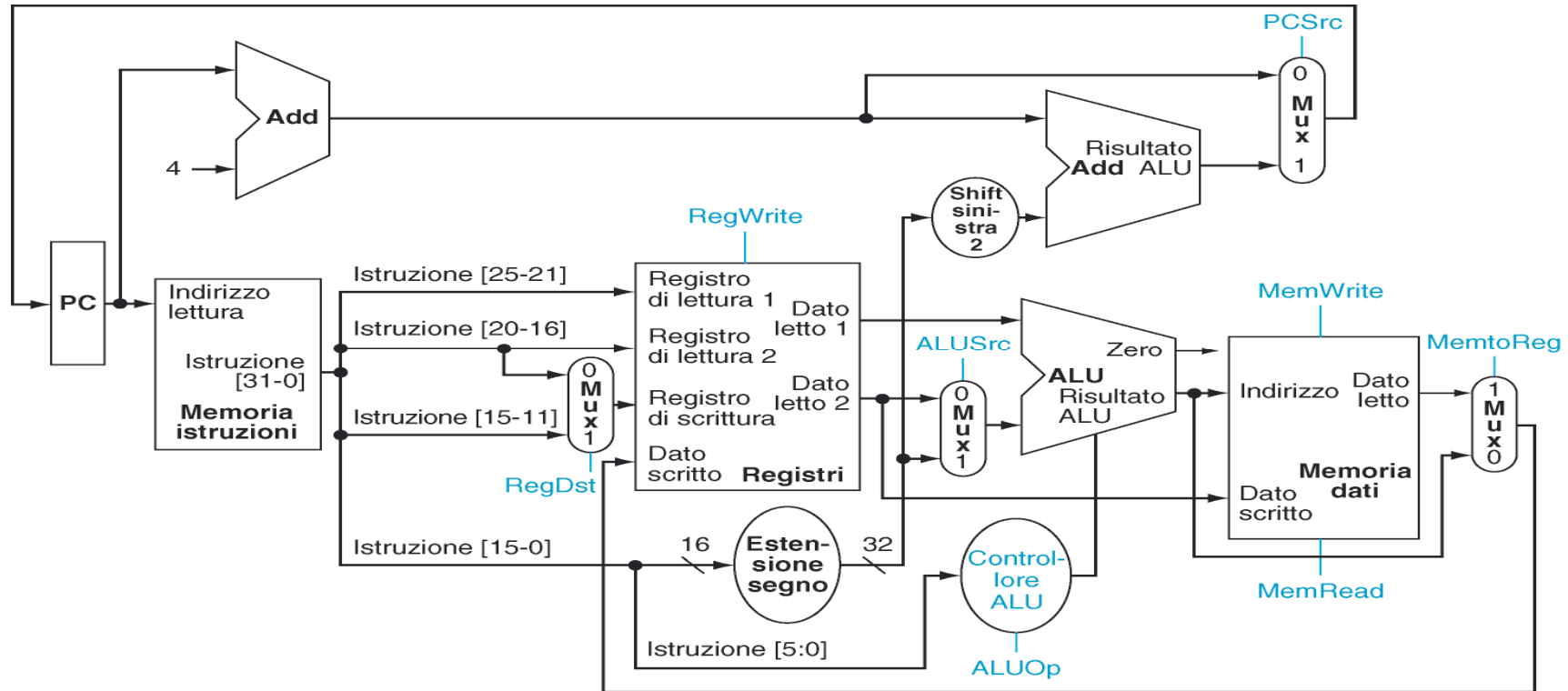
Flusso dei dati: **Registri[rs] → PC**

Segnali di controllo:

Jr che abilita il MUX per inserire in PC il valore del registro

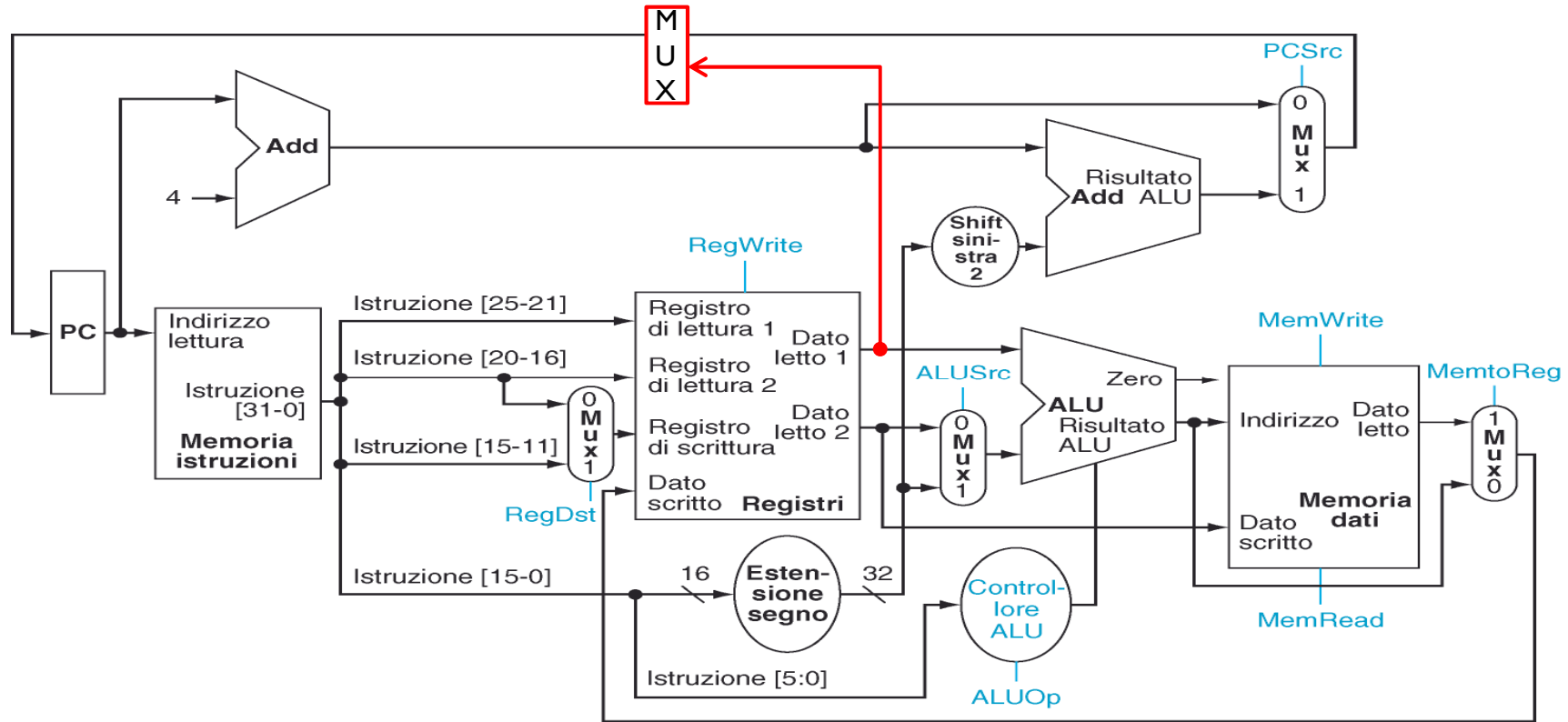
Tempo necessario:
Fetch+Reg

Istruzione jr (Jump to Register)



Tempo necessario:
Fetch+Reg

Istruzione jr (Jump to Register)



Tempo necessario:
Fetch+Reg

Esercizio

Aggiungere alla CPU l'istruzione

jrr rs (Jump Relative to Register)

di tipo R, che salta all'indirizzo (relativo al PC) contenuto nel registro **rs**

Ovvero che esegue come prossima istruzione quella che si trova all'indirizzo **PC+4+Registri[rs]**

1. Modificate lo schema per realizzare l'istruzione
2. Indicate tutti i segnali di controllo che la CU deve generare
3. Calcolate il tempo di esecuzione della istruzione assumendo che:

Accesso a memorie = 66ns, accesso ai registri = 33ns, ALU e sommatore = 100ns