SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Architettura degli Elaboratori
Lez. 8 – CPU MIPS a 1 colpo di clock

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

Argomenti

Progetto della CPU MIPS a 1 colpo di clock

- Istruzioni da implementare
- Unità funzionali necessarie
- Datapath e unità di controllo
- Tempo di esecuzione delle istruzioni

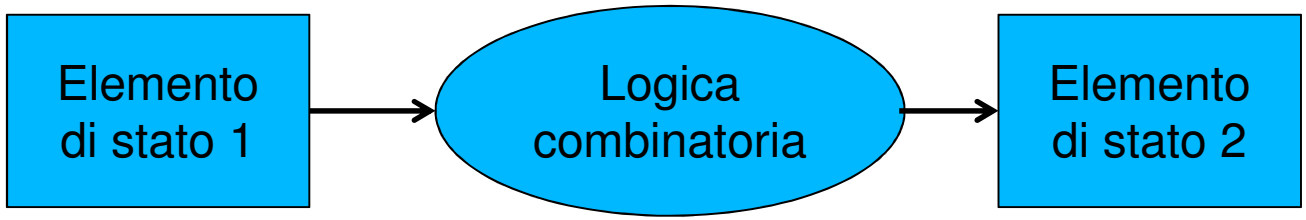
Argomenti

Progetto della CPU MIPS a 1 colpo di clock

- Istruzioni da implementare
- Unità funzionali necessarie
- Datapath e unità di controllo
- Tempo di esecuzione delle istruzioni

CPU = macchina sequenziale ovvero

Stato + circuito combinatorio



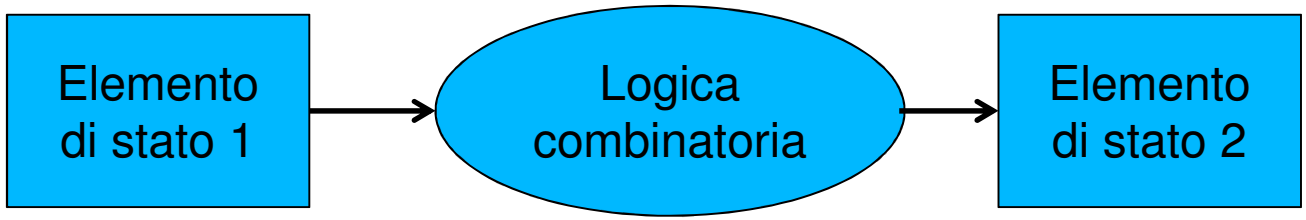
Argomenti

Progetto della CPU MIPS a 1 colpo di clock

- Istruzioni da implementare
- Unità funzionali necessarie
- Datapath e unità di controllo
- Tempo di esecuzione delle istruzioni

CPU = macchina sequenziale ovvero

Stato + circuito combinatorio



Argomenti

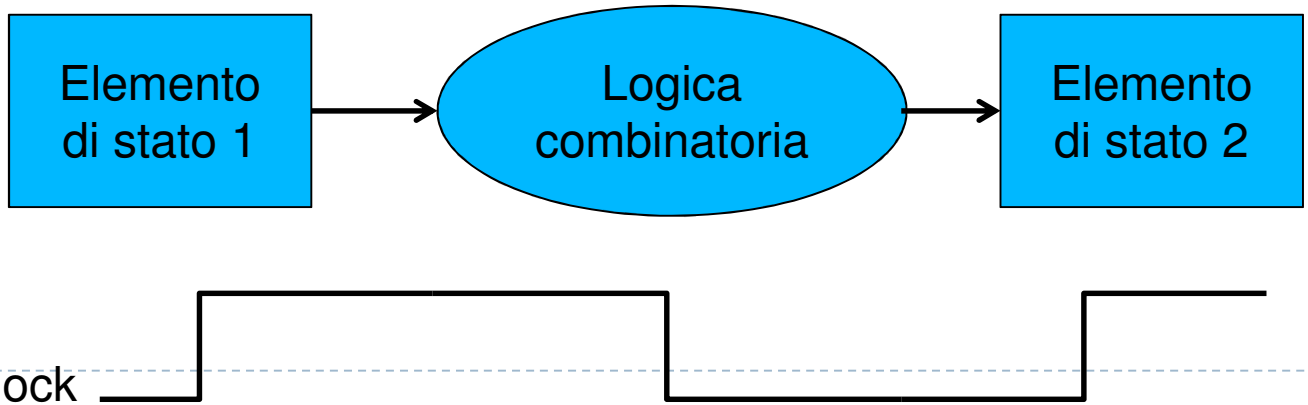
Progetto della CPU MIPS a 1 colpo di clock

- Istruzioni da implementare
- Unità funzionali necessarie
- Datapath e unità di controllo
- Tempo di esecuzione delle istruzioni

Tempo di esecuzione delle istruzioni = durata dell'istruzione più lenta

CPU = macchina sequenziale ovvero

Stato + circuito combinatorio



Progettare la CPU MIPS

- **Come prima fase progettiamo una CPU MIPS semplice non ottimizzata (senza pipeline)**
- **Per progettare la CPU dobbiamo:**
- Definire come viene elaborata una istruzione **(fasi di esecuzione)**
- Scegliere le **istruzioni da realizzare**
- Scegliere le **unità funzionali necessarie**
- **Collegare** le unità funzionali
- **Costruire la CU (Control Unit)** che controlla il funzionamento della CPU
- Calcolare il massimo tempo di esecuzione delle istruzioni (che ci dà il **periodo di clock**)

Progettare la CPU MIPS

- **Come prima fase progettiamo una CPU MIPS semplice non ottimizzata (senza pipeline)**
- **Per progettare la CPU dobbiamo:**
- Definire come viene elaborata una istruzione **(fasi di esecuzione)**
- Scegliere le **istruzioni da realizzare**
- Scegliere le **unità funzionali necessarie**
- **Collegare** le unità funzionali
- **Costruire la CU (Control Unit)** che controlla il funzionamento della CPU
- Calcolare il massimo tempo di esecuzione delle istruzioni (che ci dà il **periodo di clock**)
- **Le fasi di esecuzione di una istruzione sono:**
- **Fetch** **caricamento** di una istruzione dalla memoria alla CU
- **Decodifica** **decodifica** della istruzione e **lettura argomenti** dai registri
- **Esecuzione** **esecuzione** (attivazione delle unità funzionali necessarie)
- **Memoria** accesso alla **memoria**
- **Write Back** scrittura dei **risultati nei registri**

Progettare la CPU MIPS

- **Come prima fase progettiamo una CPU MIPS semplice non ottimizzata (senza pipeline)**
- **Per progettare la CPU dobbiamo:**
  - Definire come viene elaborata una istruzione **(fasi di esecuzione)**
  - Scegliere le **istruzioni da realizzare**
  - Scegliere le **unità funzionali necessarie**
  - **Collegare** le unità funzionali
  - **Costruire la CU (Control Unit)** che controlla il funzionamento della CPU
  - Calcolare il massimo tempo di esecuzione delle istruzioni (che ci dà il **periodo di clock**)
- **Le fasi di esecuzione di una istruzione sono:**
  - **Fetch** **caricamento** di una istruzione dalla memoria alla CU
  - **Decodifica** **decodifica** della istruzione e **lettura argomenti** dai registri
  - **Esecuzione** **esecuzione** (attivazione delle unità funzionali necessarie)
  - **Memoria** accesso alla **memoria**
  - **Write Back** scrittura dei **risultati nei registri**
- **Altre operazioni necessarie**
  - **aggiornamento del PC** **(normale / salti condizionati / salti non condizionati)**

Progettare la CPU (segue)

- **Istruzioni da realizzare**
- accesso alla memoria: **lw, sw** **(di tipo I)**
- salti condizionati: **beq** **(di tipo I)**
- op. aritmetico-logiche: **add, sub, sll, slt,…** **(di tipo R)**
- salti non condizionati **j, jal** **(di tipo J)**
- op. con costanti **li, addi, subi, …** **(di tipo I)**
- Il formato delle istruzioni MIPS è

Progettare la CPU (segue)

- **Istruzioni da realizzare**
  - accesso alla memoria: **lw, sw** **(di tipo I)**
  - salti condizionati: **beq** **(di tipo I)**
  - op. aritmetico-logiche: **add, sub, sll, slt,…** **(di tipo R)**
  - salti non condizionati **j, jal** **(di tipo J)**
  - op. con costanti **li, addi, subi, …** **(di tipo I)**
- Il formato delle istruzioni MIPS è

Nome	Campi						Note
Dimensione del campo	6 bit	5 bit	5 bit	5 bit	5 bit	6 bit	Tutte le istruzioni MIPS sono a 32 bit
Formato R	op	rs	rt	rd	shamt	funct	Formato delle funzioni aritmetiche
Formato I	op	rs	rt	Indirizzo relativo / costante			Formato delle istruzioni di salto condizionato ed immediate
Formato J	op	Indirizzo di destinazione (26 bit)					Formato delle istruzioni di salto incondizionato

Unità funzionali necessarie

- **PC** registro che contiene l'indirizzo della istruzione
- **memoria istruzioni** contiene le istruzioni
- **adder** per calcolare il PC (successivo o salto)
- **registri** contengono gli argomenti delle istruzioni
- **ALU** fa le operazioni aritmetico-logiche, confronti, calcola gli indirizzi per gli accessi in memoria
- **memoria dati** da cui leggere/in cui scrivere i dati (load/store)

Unità funzionali necessarie

- **PC** registro che contiene l'indirizzo della istruzione
- **memoria istruzioni** contiene le istruzioni
- **adder** per calcolare il PC (successivo o salto)
- **registri** contengono gli argomenti delle istruzioni
- **ALU** fa le operazioni aritmetico-logiche, confronti, calcola gli indirizzi per gli accessi in memoria
- **memoria dati** da cui leggere/in cui scrivere i dati (load/store)

- Queste unità sono collegate da diversi **datapath,** ovvero dalle interconnessioni che definiscono il flusso delle informazioni nella CPU
- Se una unità funzionale può ricevere dati da **più sorgenti** è necessario inserire un **MUX** per selezionare la sorgente necessaria

Unità funzionali necessarie

- **PC** registro che contiene l'indirizzo della istruzione
- **memoria istruzioni** contiene le istruzioni
- **adder** per calcolare il PC (successivo o salto)
- **registri** contengono gli argomenti delle istruzioni
- **ALU** fa le operazioni aritmetico-logiche, confronti, calcola gli indirizzi per gli accessi in memoria
- **memoria dati** da cui leggere/in cui scrivere i dati (load/store)

- Queste unità sono collegate da diversi **datapath,** ovvero dalle interconnessioni che definiscono il flusso delle informazioni nella CPU
- Se una unità funzionale può ricevere dati da **più sorgenti** è necessario inserire un **MUX** per selezionare la sorgente necessaria

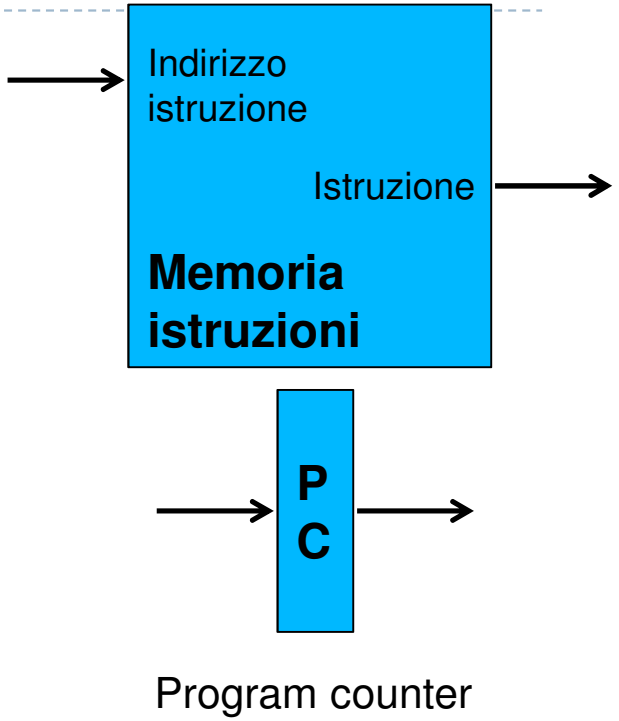
- **Le unità funzionali sono attivate e coordinate dai segnali prodotti dalla Control Unit (a seconda dell'istruzione corrente)**

Memoria istruzioni, PC e Adder

- **Memoria istruzioni:**
- Riceve un indirizzo a 32 bit
- Fornisce in uscita l'istruzione (da 32 bit) che si trova a quell'indirizzo

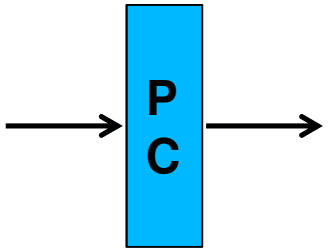
Memoria istruzioni, PC e Adder

- **Memoria istruzioni:**
  - Riceve un indirizzo a 32 bit
  - Fornisce in uscita l'istruzione (da 32 bit) che si trova a quell'indirizzo
- **Program Counter:**
  - Registro che contiene l'**indirizzo** della istruzione corrente

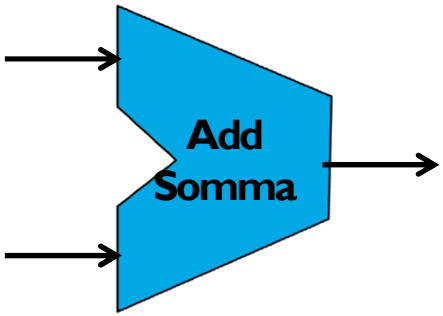


Memoria istruzioni, PC e Adder

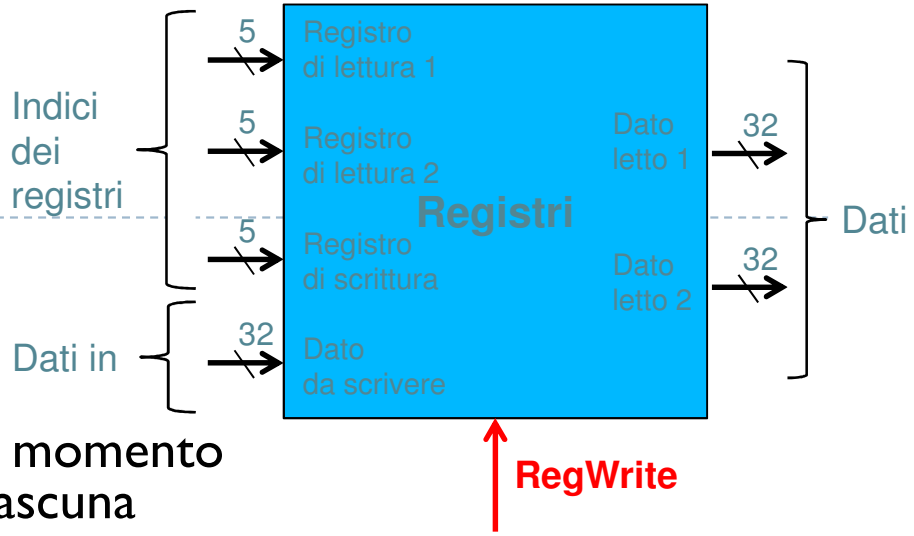
- **Memoria istruzioni:**
  - Riceve un indirizzo a 32 bit
  - Fornisce in uscita l'istruzione (da 32 bit) che si trova a quell'indirizzo
- **Program Counter:**
  - Registro che contiene l'**indirizzo** della istruzione corrente
- **Sommatore:**
  - Necessario per calcolare il nuovo PC e le destinazioni dei salti relativi
  - Riceve due valori a 32 bit e ne fornisce in uscita la somma



Program counter



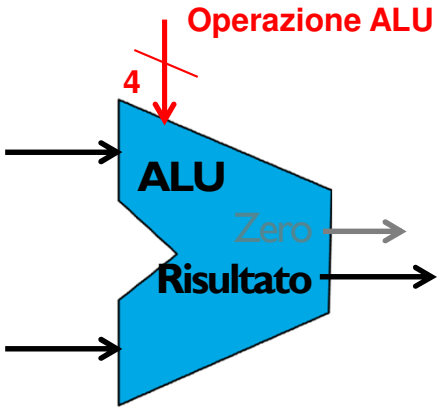
Registri e ALU



- **Il blocco dei registri:**
- contiene **32 registri** a 32 bit, indirizzabili con 5 bit (2^5 = 32)
- fornisce **due argomenti** nello stesso momento su due porte dati di uscita da 32 bit ciascuna
- può memorizzare un dato in un registro e contemporaneamente darlo in uscita
- ha 3 porte a 5 bit per indicare quali 2 registri leggere e quale registro scrivere, tre porte dati (a 32 bit), una in ingresso per il valore da memorizzare e 2 di uscita per i valori letti
- il segnale **RegWrite** abilita (se 1) la scrittura nel registro di scrittura (destinazione)

Registri e ALU

- **Il blocco dei registri:**
- contiene **32 registri** a 32 bit, indirizzabili con 5 bit (2^5 = 32)
- fornisce **due argomenti** nello stesso momento su due porte dati di uscita da 32 bit ciascuna
- può memorizzare un dato in un registro e contemporaneamente darlo in uscita
- ha 3 porte a 5 bit per indicare quali 2 registri leggere e quale registro scrivere, tre porte dati (a 32 bit), una in ingresso per il valore da memorizzare e 2 di uscita per i valori letti
- il segnale **RegWrite** abilita (se 1) la scrittura nel registro di scrittura (destinazione)



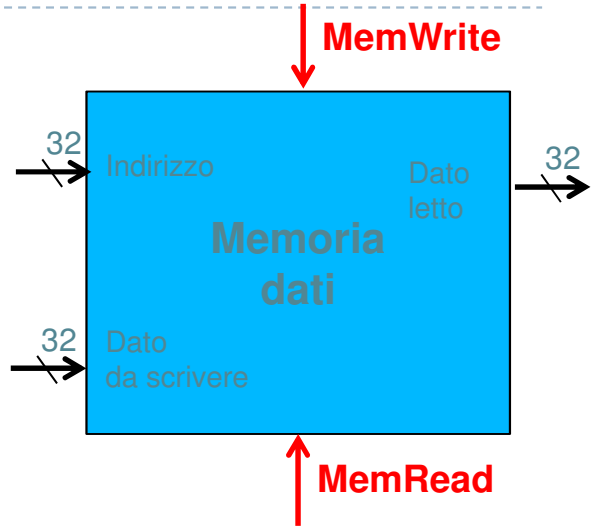
- **L' ALU:** riceve due valori interi a 32 bit

e svolge la operazione indicata dai segnali **Op. ALU**

- Oltre al risultato da 32 bit produce un segnale «**Zero**» asserito se il risultato è zero

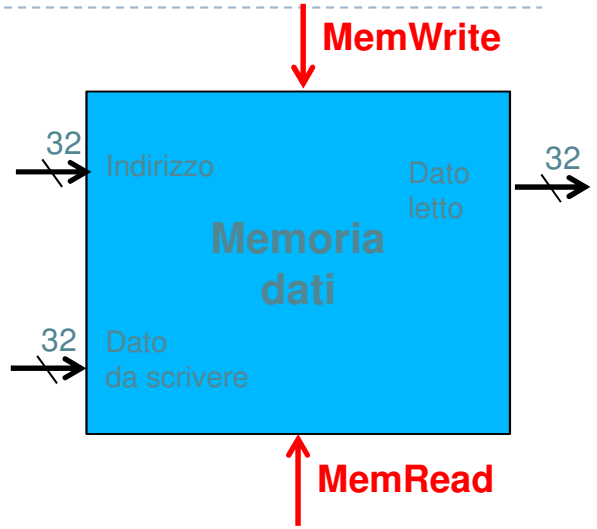
Memoria ed Estensione del segno

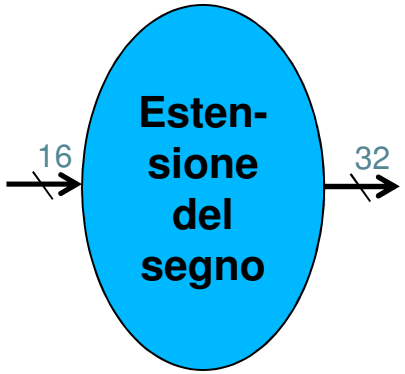
- **L'unità di memoria**
- riceve un **indirizzo** (da 32 bit) che indica quale word della memoria va letta/scritta
- riceve il segnale **MemRead** che abilita la lettura dall'indirizzo (se lw)
- riceve un dato da 32 bit da scrivere in memoria a quell'indirizzo (se sw)
- riceve il segnale di controllo **MemWrite** che abilita (1) la scrittura del dato all'indirizzo
- e fornisce su una porta di uscita da 32 bit il lato letto (se **MemRead** = 1)



Memoria ed Estensione del segno

- **L'unità di memoria**
  - riceve un **indirizzo** (da 32 bit) che indica quale word della memoria va letta/scritta
  - riceve il segnale **MemRead** che abilita la lettura dall'indirizzo (se lw)
  - riceve un dato da 32 bit da scrivere in memoria a quell'indirizzo (se sw)
  - riceve il segnale di controllo **MemWrite** che abilita (1) la scrittura del dato all'indirizzo
  - e fornisce su una porta di uscita da 32 bit il lato letto (se **MemRead** = 1)
- **L'unità di estensione del segno** serve a trasformare un intero relativo (in CA2) da 16 a 32 bit
  - Ovvero copia il bit del segno nei 16 bit più significativi della parola

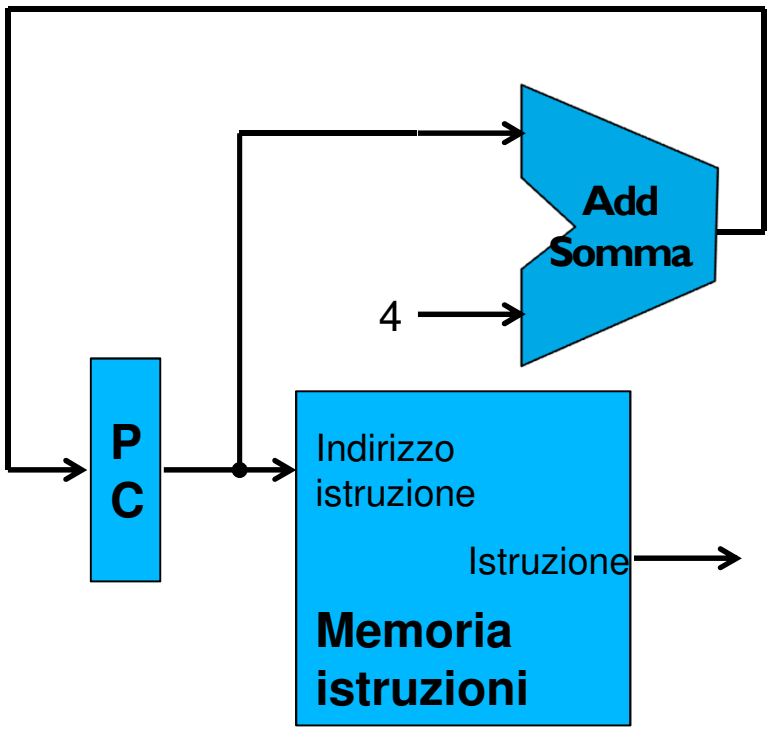




Fetch della istruzione/aggiornamento PC

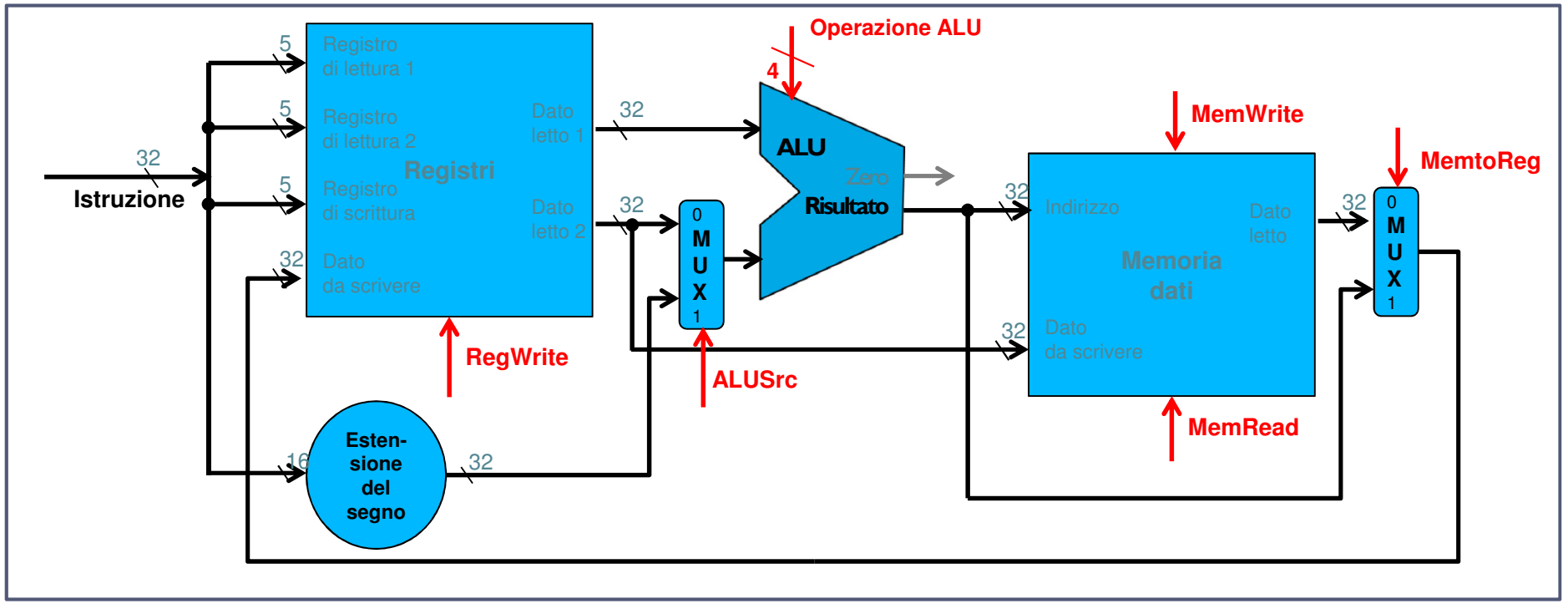
- **PC = indirizzo della istruzione**
- **Viene letta l'istruzione**
- **Il PC è incrementato di 4 (una word)**
- **e il valore viene rimesso nel PC**

- **NOTE**
- **tutte connessioni da 32 bit**
- **<u>mentre</u> viene letta la istruzione viene già calcolato il nuovo PC**



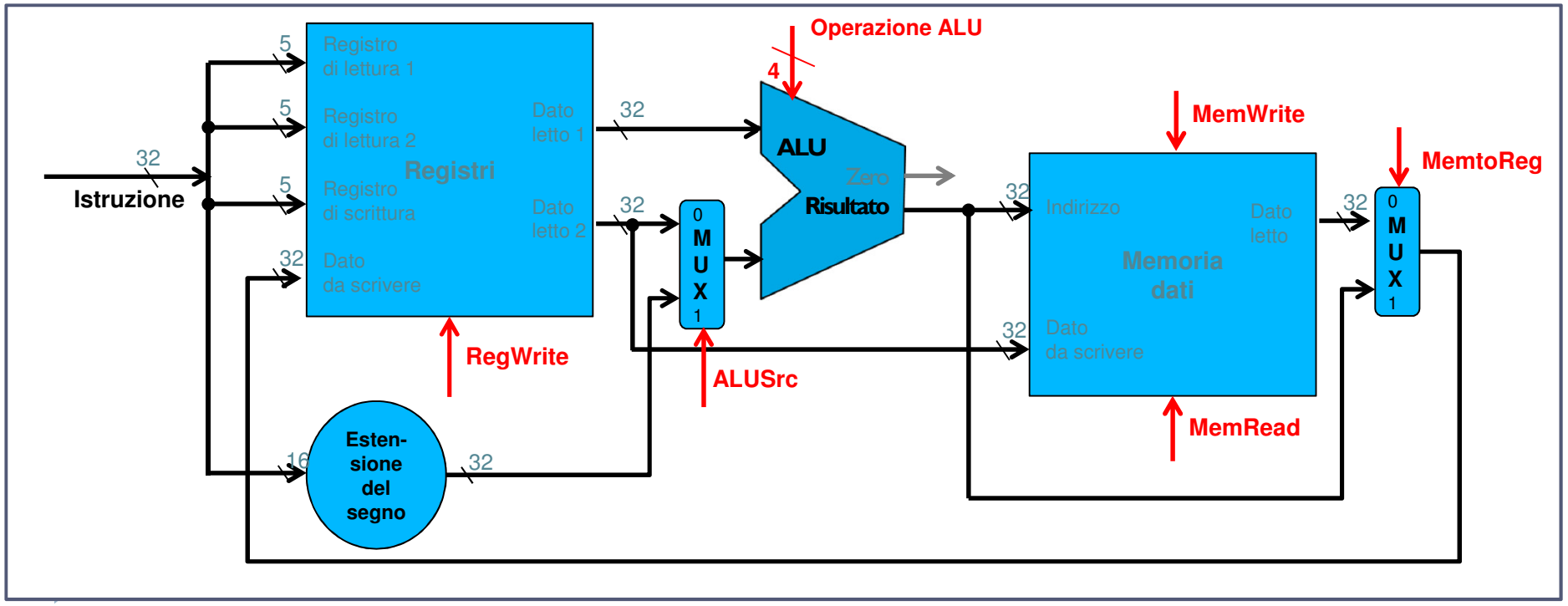
Operazioni ALU

- La **decodifica** della istruzione è estremamente facile perché i formati **I** ed **R** sono quasi uguali
- Il **secondo argomento** della istruzione può essere un registro oppure il campo immediato della istruzione stessa (in questo caso il valore viene esteso nel segno) a seconda del segnale di controllo **ALUSrc** che seleziona la porta corrispondente del MUX



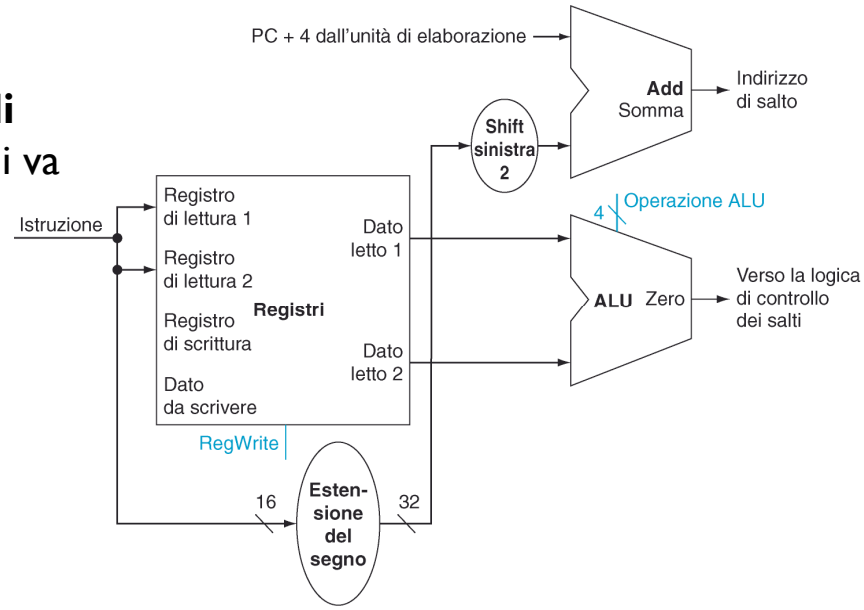
Accesso alla Memoria

- Per calcolare l'indirizzo di accesso alla memoria si usa la stessa ALU, sommando il dato del primo registro (registro base) al campo immediato della istruzione
- Il risultato del'ALU o della lw viene selezionato da **MemtoReg** che comanda il MUX a destra



Salti condizionati (beq)

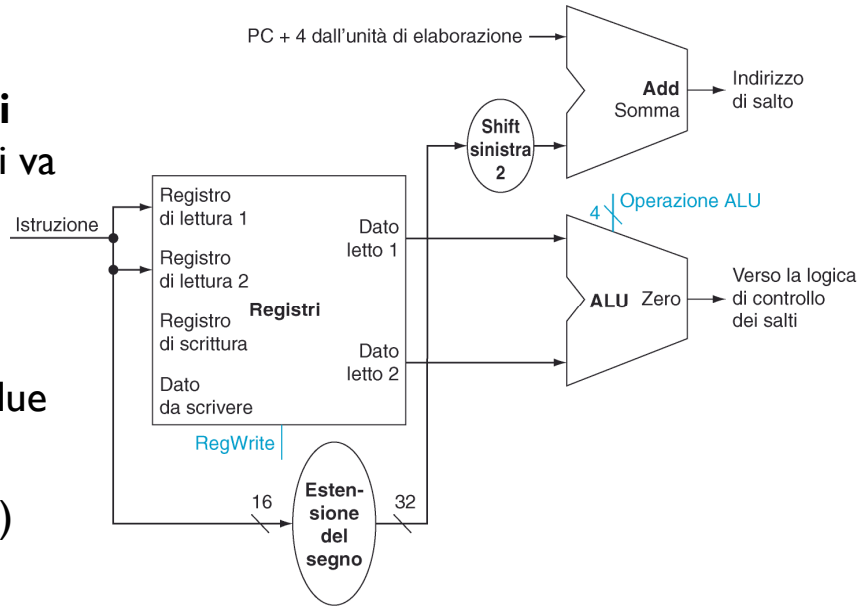
- L'**ALU** viene usata come **comparatore** (svolgendo una **sottrazione**)
- Il segnale **Zero** indica se va fatto il salto
- La destinazione dei salti è un **numero relativo di istruzioni** rispetto alla istruzione seguente, quindi va estesa nel segno e moltiplicata per 4 prima di sommarla al PC + 4 con un sommatore



Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Salti condizionati (beq)

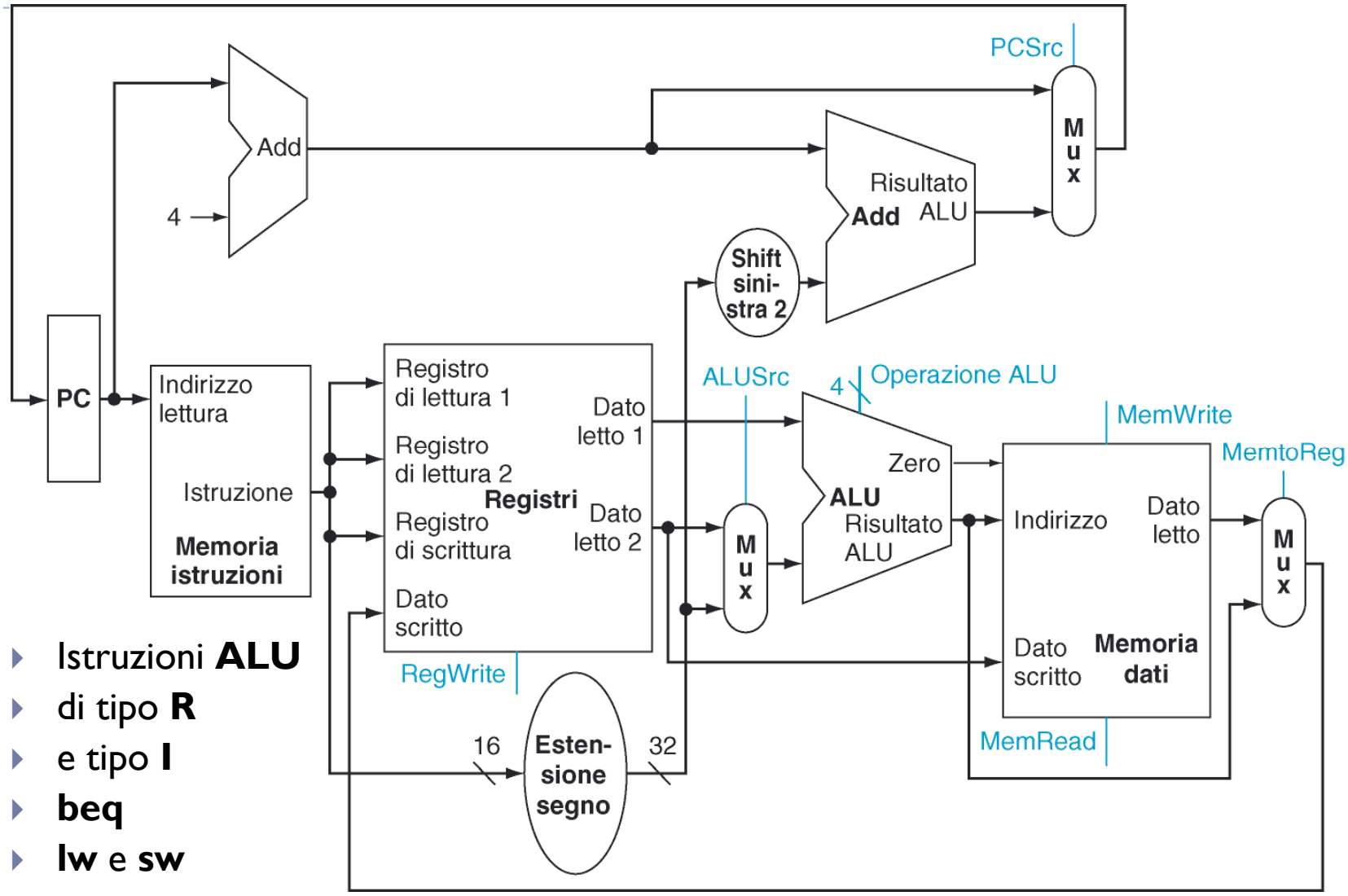
- L'**ALU** viene usata come **comparatore** (svolgendo una **sottrazione**)
- Il segnale **Zero** indica se va fatto il salto
- La destinazione dei salti è un **numero relativo di istruzioni** rispetto alla istruzione seguente, quindi va estesa nel segno e moltiplicata per 4 prima di sommarla al PC + 4 con un sommatore



- Notate che il nuovo valore del PC può venire da due diverse sorgenti:
- PC+4 (istruzione seguente)
- oppure uscita dell'adder (salto relativo)
- Quindi è necessario un **MUX** di selezione

Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

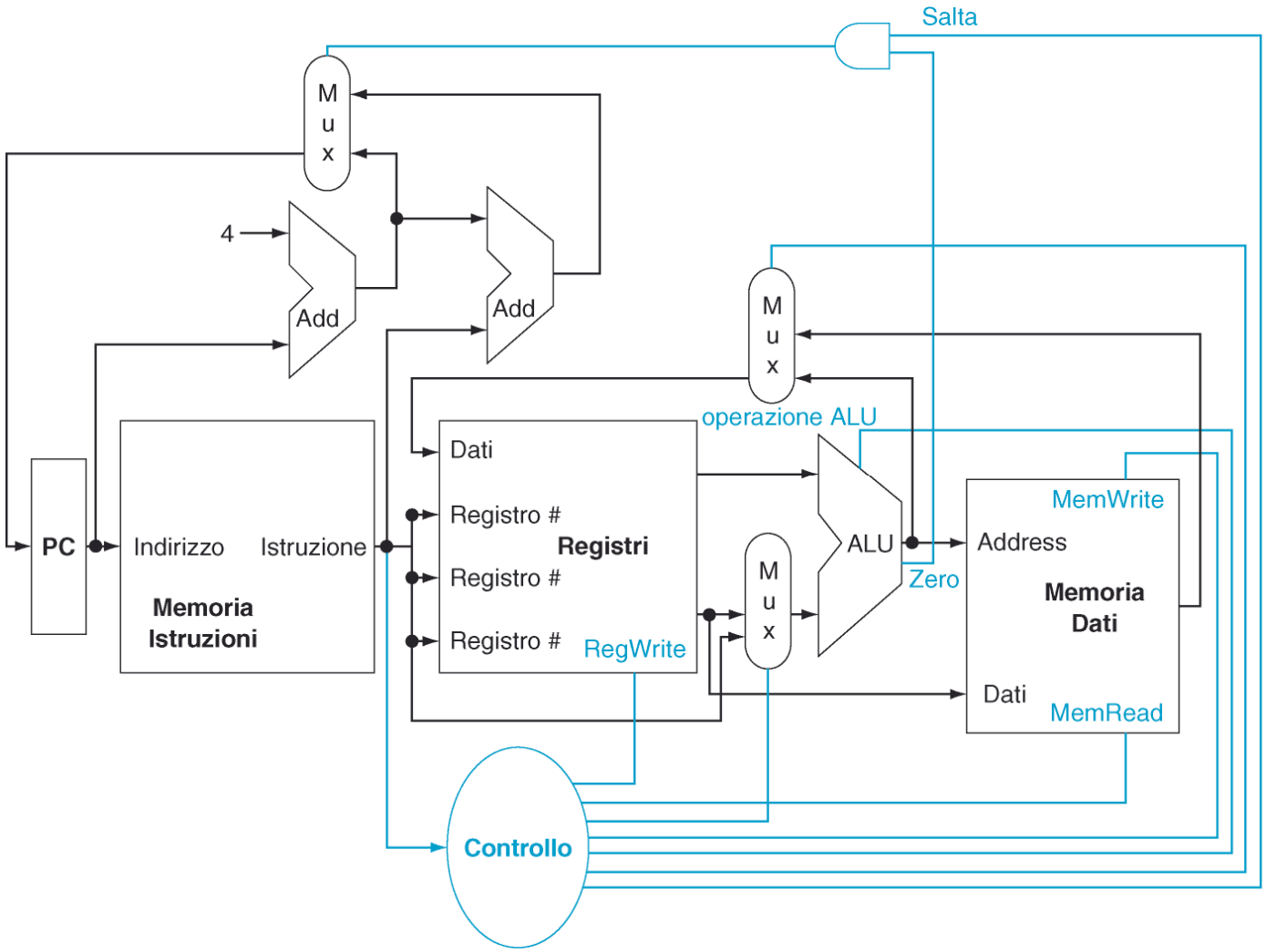
Mettiamo tutto assieme ...



- Istruzioni **ALU**
- di tipo **R**
- e tipo **I**
- **beq**
- **lw** e **sw**

Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

… con Control Unit e logica di salto



Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Dettaglio sulle istruzioni di tipo R e lw

Campo	0	rs	rt	rd	shamt	funct
Posizione dei bit	31-26	25-21	20-16	15-11	10-6	5-0

a. Istruzioni di tipo R

Campo	35 or 43	rs	rt	indirizzo
Posizione dei bit	31-26	25-21	20-16	15-0

b. Istruzioni di load e store

Campo	4	rs	rt	indirizzo
Posizione dei bit	31-26	25-21	20-16	15-0

c. Istruzioni di salto condizionato

Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Dettaglio sulle istruzioni di tipo R e lw

- Notate che **l'indice del registro destinazione** si trova in campi diversi dell'istruzione nelle istruzioni di **tipo R** e nella **lw** (che è l'unica istruzione di tipo I che modifica un registro)

Campo	0	rs	rt	rd	shamt	funct
Posizione dei bit	31-26	25-21	20-16	15-11	10-6	5-0

a. Istruzioni di tipo R

Campo	35 or 43	rs	rt	indirizzo
Posizione dei bit	31-26	25-21	20-16	15-0

b. Istruzioni di load e store

Campo	4	rs	rt	indirizzo
Posizione dei bit	31-26	25-21	20-16	15-0

c. Istruzioni di salto condizionato

Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Dettaglio sulle istruzioni di tipo R e lw

- Notate che **l'indice del registro destinazione** si trova in campi diversi dell'istruzione nelle istruzioni di **tipo R** e nella **lw** (che è l'unica istruzione di tipo I che modifica un registro)

Campo	0	rs	rt	rd	shamt	funct
Posizione dei bit	31-26	25-21	20-16	15-11	10-6	5-0

a. Istruzioni di tipo R

Campo	35 or 43	rs	rt	indirizzo
Posizione dei bit	31-26	25-21	20-16	15-0

b. Istruzioni di load e store

Campo	4	rs	rt	indirizzo
Posizione dei bit	31-26	25-21	20-16	15-0

c. Istruzioni di salto condizionato

- **È quindi necessario un MUX per selezionare il campo corretto della istruzione**

Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Dettaglio sulle istruzioni di tipo R e lw

- Notate che **l'indice del registro destinazione** si trova in campi diversi dell'istruzione nelle istruzioni di **tipo R** e nella **lw** (che è l'unica istruzione di tipo I che modifica un registro)

Campo	0	rs	rt	rd	shamt	funct
Posizione dei bit	31-26	25-21	20-16	15-11	10-6	5-0

a. Istruzioni di tipo R

Campo	35 or 43	rs	rt	indirizzo
Posizione dei bit	31-26	25-21	20-16	15-0

b. Istruzioni di load e store

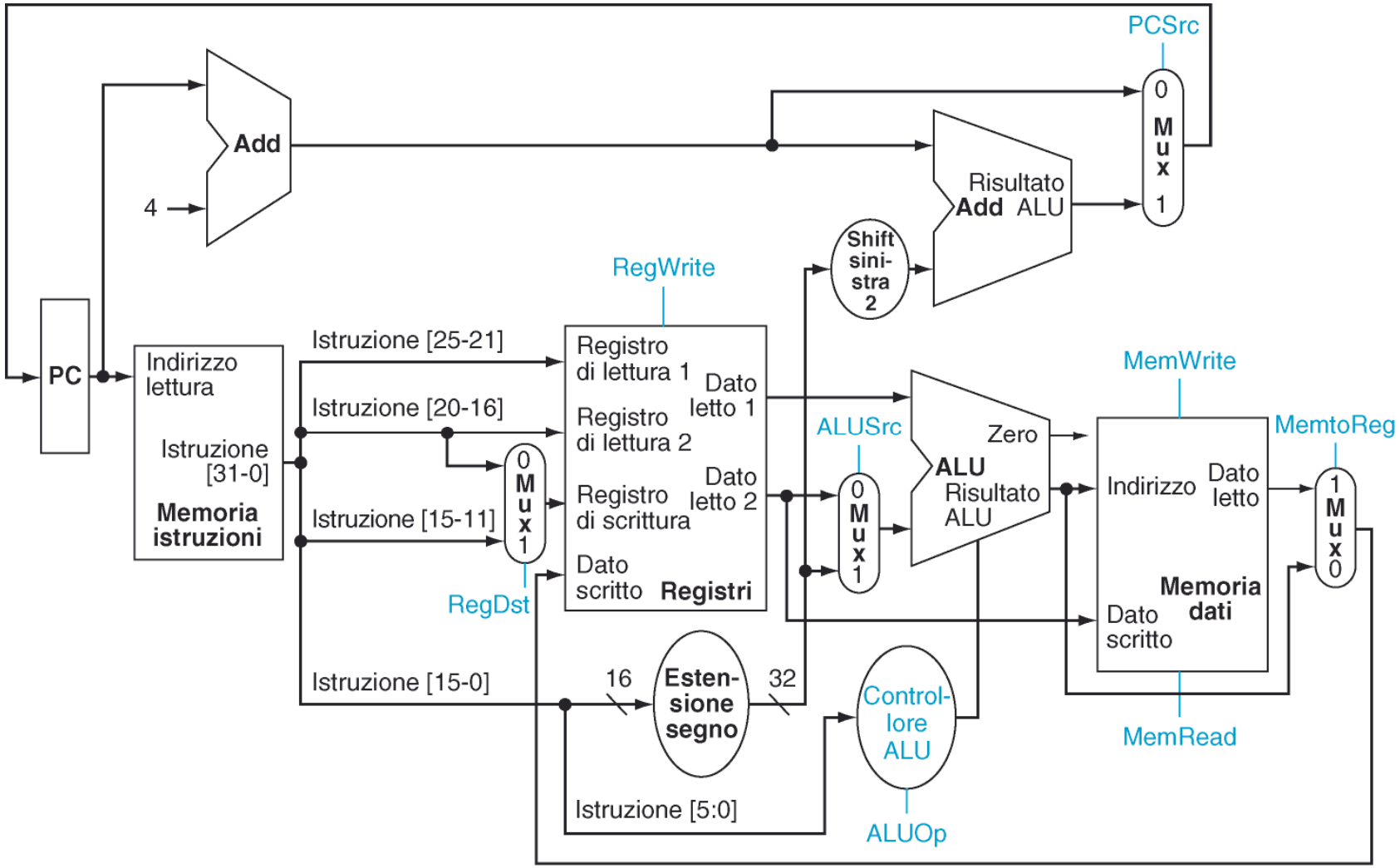
Campo	4	rs	rt	indirizzo
Posizione dei bit	31-26	25-21	20-16	15-0

c. Istruzioni di salto condizionato

- **È quindi necessario un MUX per selezionare il campo corretto della istruzione**
- **INOLTRE:** tutte le istruzioni R hanno lo stesso codice (0) e sono distinte dal campo **funct**

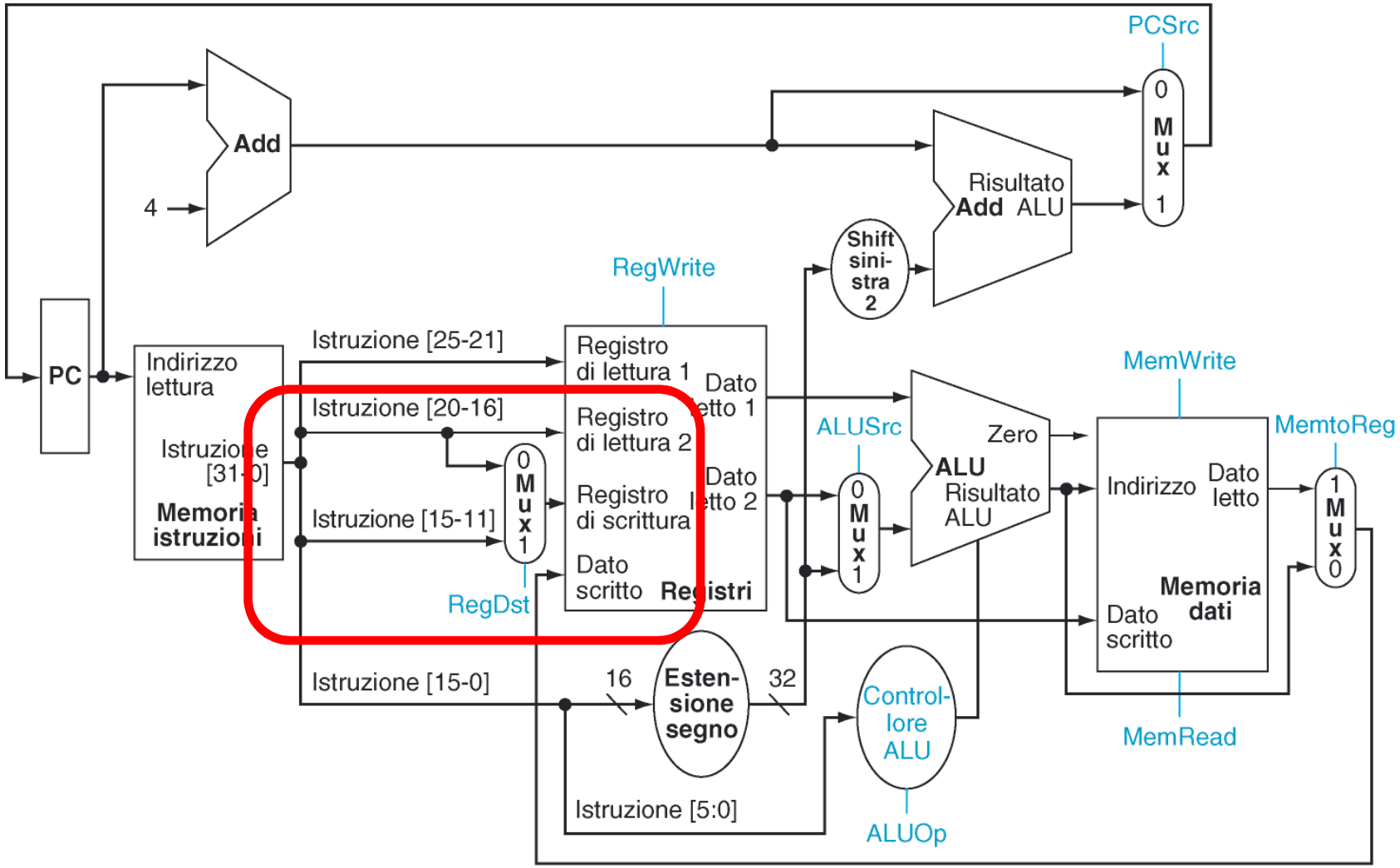
Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Datapath completo (senza CU)



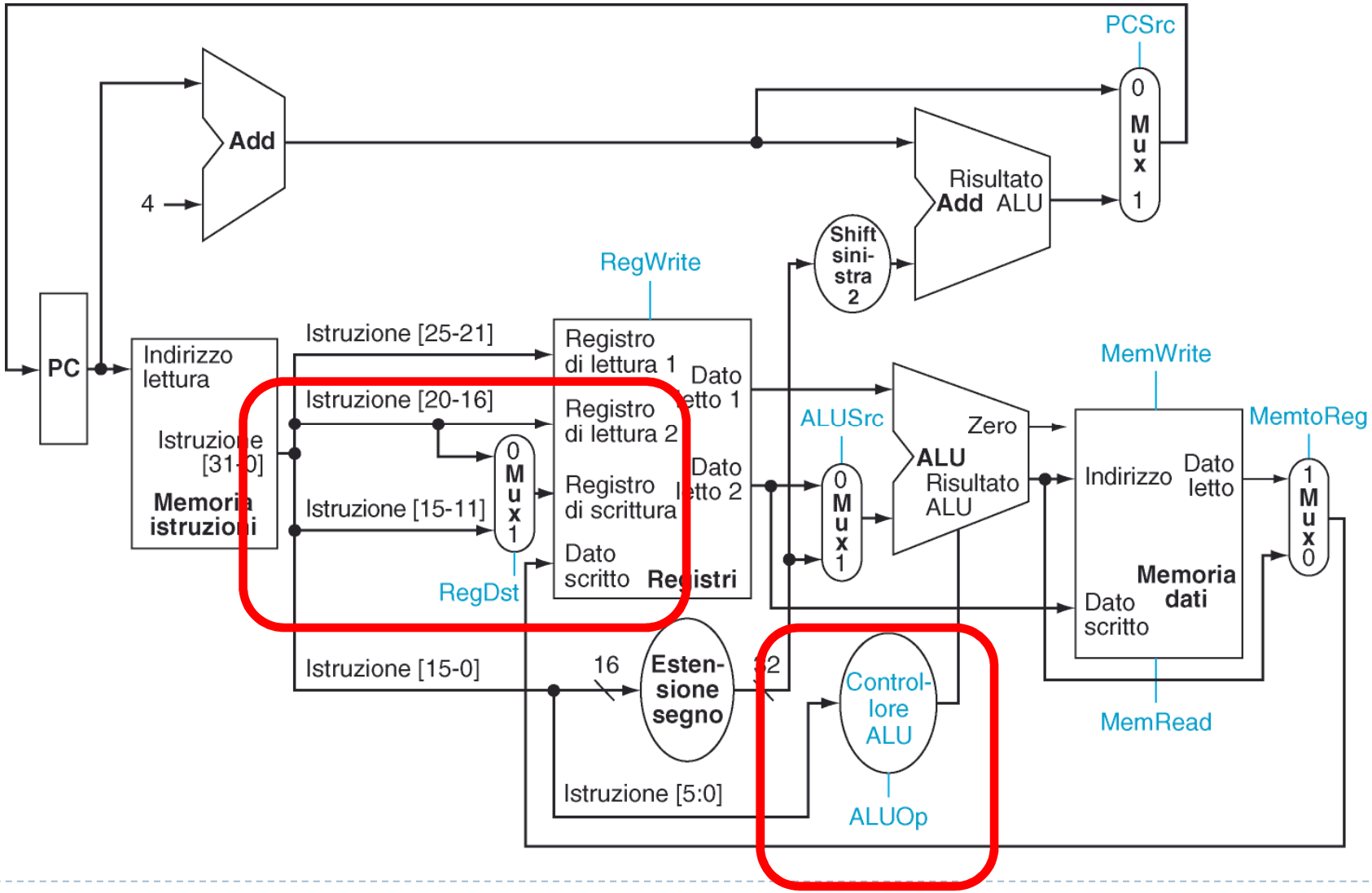
Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Datapath completo (senza CU)



Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Datapath completo (senza CU)



Patterson et al., STRUTTURA E PROGETTO DEI CALCOLATORI, 3/E, Zanichelli editore S.p.A. Copyright © 2010

Segnali di controllo

Nome del segnale	Effetto se NON asserito (0)	Effetto se asserito (1)
RegDst	Il numero del registro di scrittura proviene dal campo **rt** (bit 20-16)	Il numero del registro di scrittura proviene dal campo **rd** (bit 15-11)
RegWrite	Nulla (i registri non sono modificati)	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 dalla estensione del segno dei 16 bit meno significativi dell'istruzione (parte immediata)
PCSrc	Nel PC viene scritta l'uscita del sommatore che calcola il valore PC+4	Nel PC viene scritta l'uscita del sommatore che calcola l'indirizzo del salto relativo (beq)
MemRead	Nulla	Il dato della memoria nella posizione puntata dall'indirizzo è inviato all'uscita «dato letto»
MemWrite	Nulla (la memoria non viene modificata)	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 ALU	Il dato inviato al register file per la scrittura proviene dalla Memoria Dati

I segnali da generare

- L'ALU deve seguire 3 tipi di comportamento:
  - Se l'istruzione è di **tipo R** eseguire l'operazione indicata dal campo **funct** dell'istruzione
  - Se l'istruzione accede alla memoria (**lw, sw**) svolgere la **somma** che calcola l'indirizzo
  - Se l'istruzione è un **beq** deve svolgere una **differenza**
- Per codificare 3 comportamenti bastano 2 segnali dalla CU: **ALUOp1** ed **ALUOp0**

- **Quindi i segnali che la CU deve produrre per i diversi tipi di istruzione devono essere:**

Istruzione	Reg Dst	ALU Src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	ALU Op 1	ALU Op2
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

Tempi di esecuzione

- Se conosciamo il tempo necessario a produrre i risultati delle diverse unità funzionali possiamo calcolare il tempo totale di ciascuna istruzione. Supponiamo che i tempi siano:
- **accesso alla memoria** (dati o istruzione) = 100 ns (Fetch e MEM)
- **ALU e sommatori** = 150 ns (EX e PC)
- **accesso ai registri** (in lettura o scrittura) = 50 ns (ID e WB)
- tutte le altre componenti = 0 ns
- Allora i tempi di esecuzione delle istruzioni saranno

Istruzione	Istruction Fetch	Istruction Decode	Execution	MEM	Write Back	Totale
di tipo R	100	50	150		50	**350**
lw	100	50	150	100	50	**450**
sw	100	50	150	100		**400**
beq	100	50	150			**300**

- **NOTA:** le due operazioni di somma per calcolare PC +4 (150ns) e salti condizionati (altri 150ns) sono svolte in parallelo al Fetch, Decode ed Execution e non allungano i tempi

Tempi di esecuzione

- Se conosciamo il tempo necessario a produrre i risultati delle diverse unità funzionali possiamo calcolare il tempo totale di ciascuna istruzione. Supponiamo che i tempi siano:
- **accesso alla memoria** (dati o istruzione) = 100 ns (Fetch e MEM)
- **ALU e sommatori** = 150 ns (EX e PC)
- **accesso ai registri** (in lettura o scrittura) = 50 ns (ID e WB)
- tutte le altre componenti = 0 ns
- Allora i tempi di esecuzione delle istruzioni saranno

Istruzione	Istruction Fetch	Istruction Decode	Execution	MEM	Write Back	Totale
di tipo R	100	50	150		50	**350**
lw	100	50	150	100	50	**450**
sw	100	50	150	100		**400**
beq	100	50	150			**300**

- **NOTA:** le due operazioni di somma per calcolare PC +4 (150ns) e salti condizionati (altri 150ns) sono svolte in parallelo al Fetch, Decode ed Execution e non allungano i tempi

Esercizio

- 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 come visto prima

- Progettate la tabella di verità della **CU** (solo per le righe necessarie):
- che ha come ingresso i 6 bit del codice operativo della istruzione in binario
- e come uscita i 9 bit dei segnali di controllo da produrre:
  - **(RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, ALUOp1, ALUOp0)**