POLITECNICO
MILANO 1863

Architettura dei calcolatori e sistemi operativi

Il processore
Capitolo 4 P&H

4 . 11. 2015

Sommario

Instruction Set di riferimento per il processore

Esecuzione delle istruzioni

Struttura del processore

Caratteristiche principali dell'architettura MIPS

Architettura RISC (Reduced Instruction Set Computer):
esegue soltanto istruzioni semplici in un ciclo base ridotto e ha solo 3 formati istruzione diversi (R, I e J)

Architettura LOAD/STORE: gli operandi dell'*ALU* possono provenire soltanto dai registri di uso generale presenti nella *CPU* e **non** possono provenire direttamente dalla memoria.
Sono necessarie apposite istruzioni di:

- *caricamento (load)* dei dati da memoria ai registri;
- *memorizzazione (store)* dei dati dai registri alla memoria.

Architettura pipeline: tecnica per migliorare le prestazioni basata sulla sovrapposizione dell'esecuzione di più istruzioni appartenenti ad un flusso di esecuzione sequenziale.

Processore e set istruzioni di riferimento

Analizzeremo un'implementazione semplificata dal processore MIPS cui è associato il corrispondente set istruzioni

L'***insieme ridotto delle istruzioni*** (istruzioni di riferimento per la realizzazione del processore) fanno parte delle seguenti categorie

- Istruzioni aritmetico-logiche
- Istruzioni di trasferimento da/verso la memoria *(load/store)*
- Istruzioni di salto (condizionato e incondizionato)

Coprono i ***tre formati istruzioni*** (R, I e J) del processore MIPS

Istruzioni di riferimento

Istruzioni ***aritmetico-logiche*** sia a soli registri che immediate, esempi:

```
add $s1, $s2, $s3          # $s1 ← $s2 + $s3
addi $s1, $s1, 4           # $s1 ← $s1 + 4
```

Si potrebbe considerare anche la

```
slt $s1, $s2, $s3          # $s2 < $s3 $s1=1 altrimenti $s1=0
```

Istruzioni di ***trasferimento da/verso la memoria*** *(load/store)*

```
lw $s1, offset ($s2)       # $s1 ← M[$s2+offset]
sw $s1, offset ($s2)       # M[$s2+offset] ← $s1
```

Istruzioni di ***salto condizionato*** (conditional branch): **`beq`** (*branch on equal*).

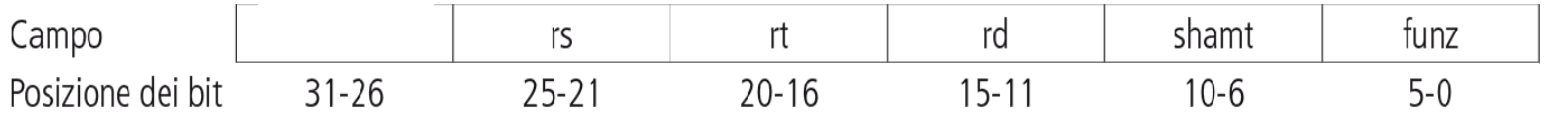
```
beq $s1, $s2, L1   # go to L1 if ($s1 == $s2)
(bne $s1, $s2, L1  # go to L1 if ($s1 != $s2))
```

Istruzioni di ***salto incondizionato*** (unconditional jump): **`j`** *(jump)*

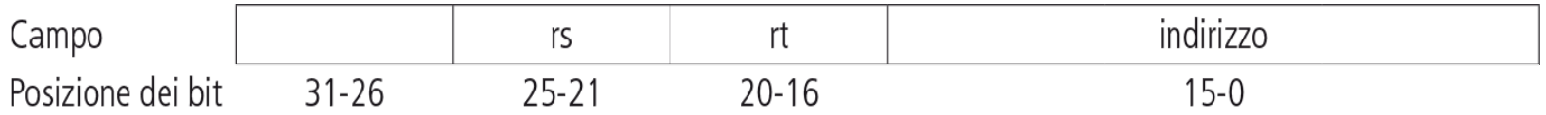
```
j   L1      # go to L1
```

Formato istruzioni e dimensioni dei campi

I diversi formati **(R, I, J)** sono riconosciuti tramite il valore del primo campo ***codice operativo* (*opcode*) di 6 bit** che indica al processore come trattare i rimanenti bit dell'istruzione



Istruzioni di tipo **R**

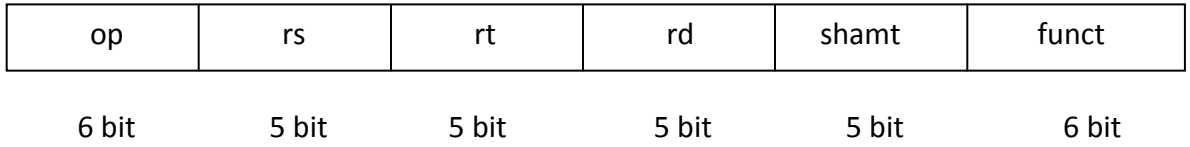


Istruzioni di tipo **I**

Istruzioni di tipo **J**

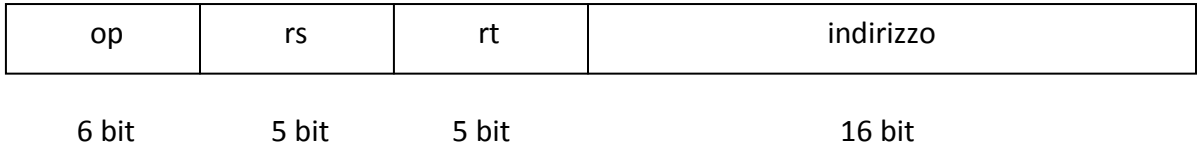
Formato istruzioni e significato dei campi (1)

tipo R: aritmetico-logiche a 3 registri (codice operativo = 0)



rs: primo registro sorgente
rt: secondo registro sorgente
rd: registro destinazione
shamt: shift amount (scorrimento)
funct: indica l'operazione specifica

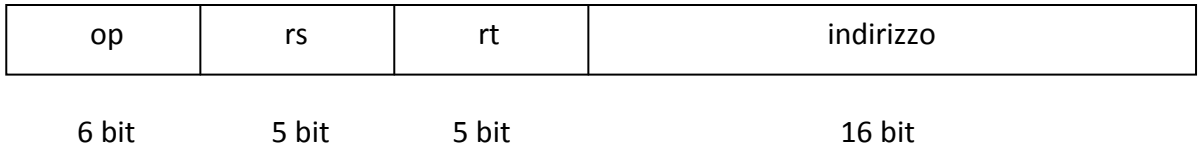
tipo I: aritmetico-logiche con immediati



rs: registro sorgente
rt: registro destinazione
indirizzo: valore dell'operando ***immediato***

Formato istruzioni e significato dei campi (2)

tipo I: load/store

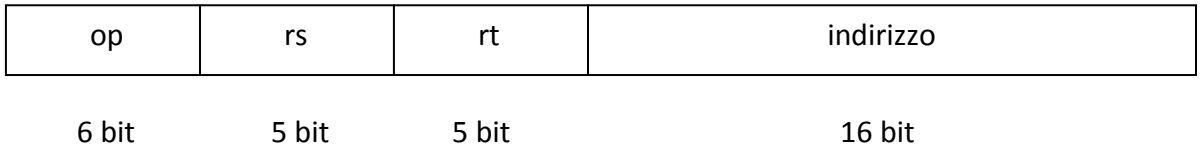


rs: registro base

rt: registro destinazione se load, sorgente se store

indirizzo: ***spiazzamento*** da sommare al registro base per ottenere l'indirizzo effettivo di memoria

I: salti condizionati beq/bne



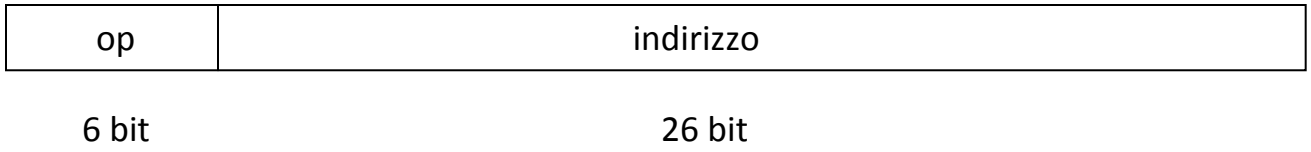
rs: primo registro sorgente

rt: secondo registro sorgente

indirizzo: ***spiazzamento di parola*** da sommare a (PC + 4) per ottenere l'indirizzo effettivo della destinazione di salto se la condizione è verificata

Formato istruzioni e significato dei campi (3)

tipo J: salto incondizionato



indirizzo (composto da **26-bit**): è una parte (26 bit su 32) dell'indirizzo **assoluto** di destinazione del salto

I 26-bit del campo **indirizzo** rappresentano un indirizzo di parola **(word address)**

Questa istruzione verrà considerata più avanti

Esecuzione delle istruzioni

Passi svolti durante l'esecuzione delle istruzioni aritmetico-logiche *di tipo R*

Istruzioni aritmetico-logiche: **op $x,$y,$z**

Prelievo Istruz. & Increm. PC	Lettura Registri Sorgente $y e $z	Op. ALU sui Dati Letti ($y op $z)	Scrittura nel Reg. Destinazione $x

Un'istruzione aritmetico-logica **(tipo R),** ad esempio **`add $x, $y, $z`** viene eseguita in **4** passi:

- Prelievo istruzione dalla memoria istruzioni e incremento del *PC*.
- Lettura dei 2 registri sorgente (**$y** e **$z**) dal banco dei registri, utilizzando i bit [25-21] e [20 -16] per selezionare i registri
- Operazione dell'ALU sui dati letti dal banco dei registri, utilizzando il campo ***function*** per realizzare la funzione aritmetico-logica.
- Scrittura del risultato dell'ALU nel banco dei registri utilizzando i bit [15-11] dell'istruzione per selezionare il registro destinazione **($x)**

Passi svolti durante l'esecuzione delle istruzioni di *load*

Istruzioni di *load*: `lw $x,offset($y)`

Prelievo Istruz. & Increm. PC	Lettura Registro Base `$y`	Op. ALU (`$y+offset`)	Prelievo Dato `M($y+offset)`	Scrittura nel Reg. Destinazione `$x`

Un'istruzione di load (tipo I), ad esempio **`lw $x,offset($y)`** viene eseguita in 5 passi:

- Prelievo istruzione dalla memoria istruzioni e incremento del PC
- Lettura del registro base ($y) dal banco dei registri, bit [25-21]
- Operazione dell'ALU per calcolare la somma del valore letto dal registro base e dei 16 bit meno significativi dell'istruzione estesi in segno (campo *offset*)
- Prelievo del dato nella memoria dati utilizzando come indirizzo di lettura il risultato dell'ALU
- Scrittura del dato proveniente dalla memoria nel banco dei registri; il registro destinazione ($x) è indicato dai bit [20-16] dell'istruzione.

Passi svolti durante l'esecuzione delle istruzioni di *store*

Istruzioni di *store*: `sw $x,offset($y)`

Prelievo Istruz. & Increm. PC	Lettura Registri Base $y & Sorg. $x	Op. ALU ($y+offset)	Scrittura Dato M($y+offset)

Un'istruzione di store (tipo I), ad esempio **`sw $x,offset($y)`** viene eseguita in 4 passi:

- Prelievo istruzione dalla memoria istruzioni e incremento del PC
- Lettura del registro base ($y), bit [25-21], e del registro sorgente ($x) dal banco dei registri; il registro sorgente è indicato dai bit [20-16] dell'istruzione
- Operazione dell'ALU per calcolare la somma del valore letto dal registro base e dei 16 bit meno significativi dell'istruzione estesi in segno (*offset*)
- Scrittura del dato proveniente dal registro sorgente ($x) nella memoria dati utilizzando come indirizzo di scrittura il risultato dell'ALU

Passi svolti durante l'esecuzione dell' istruzione *addi*

Istruzione di *somma con immediato*: **`addi $x,$y, 22`**

Prelievo Istruz. & Increm. PC	Lettura Registro sorgente `$y`	Op. ALU (`$y+val.imm.`)	Scrittura nel Reg. destinazione `$x`

Questa istruzione viene eseguita in 4 passi:

- Prelievo istruzione dalla memoria istruzioni e incremento del PC
- Lettura del registro sorgente ($y) dal banco dei registri, bit [25-21]
- Operazione dell'ALU per calcolare la somma del valore letto dal registro sorgente e dei 16 bit meno significativi dell'istruzione estesi in segno (campo *immediato*)
- Scrittura del risultato dell'ALU nel banco dei registri; il registro destinazione ($x) è indicato dai bit [20-16] dell'istruzione.

Passi svolti durante l'esecuzione delle istruzioni di salto condizionato

Istruzioni di salto condizionato: `beq $x,$y,offset`

Prelievo Istruz. & Increm. PC	Lettura Registri Sorgente $x e $y	Op. ALU ($x-$y) & (PC+4+offset)	Scrittura nel PC

Un'istruzione di salto condizionato (tipo I), ad esempio **`beq $x, $y, offset`** viene eseguita in 4 passi:

- Prelievo istruzione dalla memoria istruzioni e incremento del PC
- Lettura dei 2 registri sorgente ($x e $y) dal banco dei registri, bit [25-21] e [20 -16]
- Operazione dell'ALU per effettuare la sottrazione tra i valori letti dal banco dei registri. Il valore (PC+4) viene sommato ai 16 bit meno significativi dell'istruzione estesi in segno (offset); il risultato è l'indirizzo di destinazione del salto (Branch Target Address)
- L'uscita Zero dell'ALU viene utilizzata per decidere quale valore debba essere memorizzato nel PC: (PC+4) oppure (PC+4+offset)

Passi svolti durante l'esecuzione delle istruzioni

Istruzioni aritmetico-logiche: `op $x,$y,$z`

Prelievo Istruz. & Increm. PC	Lettura Registri Sorgente `$y` e `$z`	Op. ALU sui Dati Letti (`$y op $z`)	Scrittura nel Reg. Destinazione `$x`

Istruzioni di caricamento *(load)*: `lw $x,offset($y)`

Prelievo Istruz. & Increm. PC	Lettura Registro Base `$y`	Op. ALU (`$y+offset`)	Prelievo Dato `M($y+offset)`	Scrittura nel Reg. Destinazione `$x`

Istruzioni di memorizzazione *(store)*: `sw $x,offset($y)`

Prelievo Istruz. & Increm. PC	Lettura Registri Base `$y` & Sorg. `$x`	Op. ALU (`$y+offset`)	Scrittura Dato `M($y+offset)`

Istruzioni di salto condizionato: `beq $x,$y,offset`

Prelievo Istruz. & Increm. PC	Lettura Registri Sorgente `$x` e `$y`	Op. ALU (`$x-$y`) & (`PC+4+offset`)	Scrittura nel PC

Esecuzione delle istruzioni

Per ogni tipo di istruzione i primi 2 passi da eseguire sono identici:

- Inviare il contenuto del *Program Counter (PC)* ad una memoria che contiene il codice per prelevare l'istruzione *(Memoria Istruzioni).*
- Leggere uno o due registri dal banco dei registri utilizzando i campi dell'istruzione per selezionare i registri ai quali accedere

Dopo questi 2 passi, le azioni necessarie per concludere l'esecuzione dell'istruzione dipendono dal tipo di istruzione (codice operativo), sebbene tutte le istruzioni utilizzino l'*ALU* dopo la lettura dei registri:

- Le istruzioni aritmetico-logiche per l'esecuzione dell'operazione
- Le istruzioni di riferimento a memoria per il calcolo dell'indirizzo effettivo;
- I salti condizionati per valutare l'esito dei confronti.

Esecuzione delle istruzioni (cont.)

Dopo aver utilizzato l'ALU, le azioni richieste per completare le varie istruzioni si differenziano ulteriormente:

- Le istruzioni aritmetico-logiche devono scrivere nel registro destinazione il risultato dell'ALU

- Le istruzioni di *load* richiedono l'accesso in lettura alla *Memoria Dati* ed eseguono il caricamento del dato letto nel registro destinazione

- Le istruzioni di *store* richiedono l'accesso in scrittura alla *Memoria Dati* ed eseguono la memorizzazione del dato proveniente dal registro sorgente

- Le istruzioni di salto condizionato, possono cambiare l'indirizzo dell'istruzione successiva in base al risultato del confronto.

Unità funzionali richieste durante l'esecuzione in alcune istruzioni

Tipo di istruzione	Unità funzionali utilizzate				
Tipo R	Memoria istruzioni	Banco registri	ALU	Banco registri	
Caricamento parola	Memoria istruzioni	Banco registri	ALU	Memoria dati	Banco registri
Memorizzazione parola	Memoria istruzioni	Banco registri	ALU	Memoria dati	
Salto condizionato	Memoria istruzioni	Banco registri	ALU		
Salto	Memoria istruzioni				

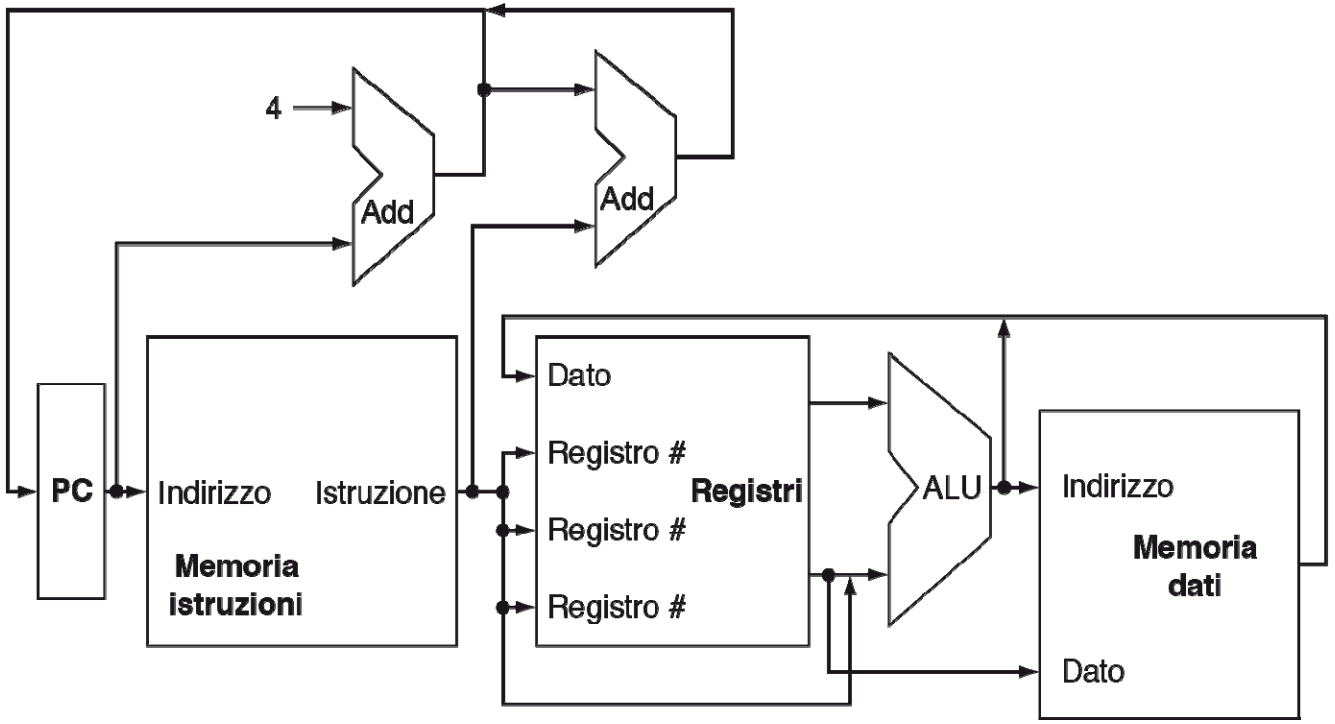
Tempo di esecuzione delle istruzioni

Tipo di istruzione	Lettura dell'istruzione	Lettura dei registri	Operazione con la ALU	Accesso ai dati in memoria	Scrittura del register file	Tempo totale
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
Formato R (`add`, `sub`, `and`, `or`, `slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Salto condizionato (`beq`)	200 ps	100 ps	200 ps			500 ps

- Nota: se il processore deve effettuare ogni istruzione in un solo ciclo di clock, la durata del ciclo non deve essere inferiore a *800 ps*, perché si deve tenere conto dell'istruzione più lenta

Architettura del processore

Struttura base del processore MIPS



- *Memoria Istruzioni* (*MI*) (usata solo in lettura) separata dalla *Memoria Dati* (*MD*)
- I 32 registri del processore sono organizzati *Register File* - **RF** con *due* porte di lettura e *una* porta in scrittura.
  - I campi dell'istruzione indicano direttamente i registri che debbono essere utilizzati come operandi dell'istruzione e vengono perciò collegati agli ingressi del Register File

Struttura base del processore MIPS (cont.)

Il *Register File* ha 4 ingressi e 2 uscite, che realizzano due porte di lettura e una porta di scrittura:

- 3 ingressi sono collegati ai campi dell'istruzione che specificano i registri sorgente o base e il registro destinazione (2 per porte in lettura e 1 per porta in scrittura)
- 1 ingresso è per i dati che possono essere scritti nel registro destinazione (per la porta di scrittura)
- 2 uscite del banco dei registri riportano il contenuto dei 2 registri letti (per le porte di lettura)

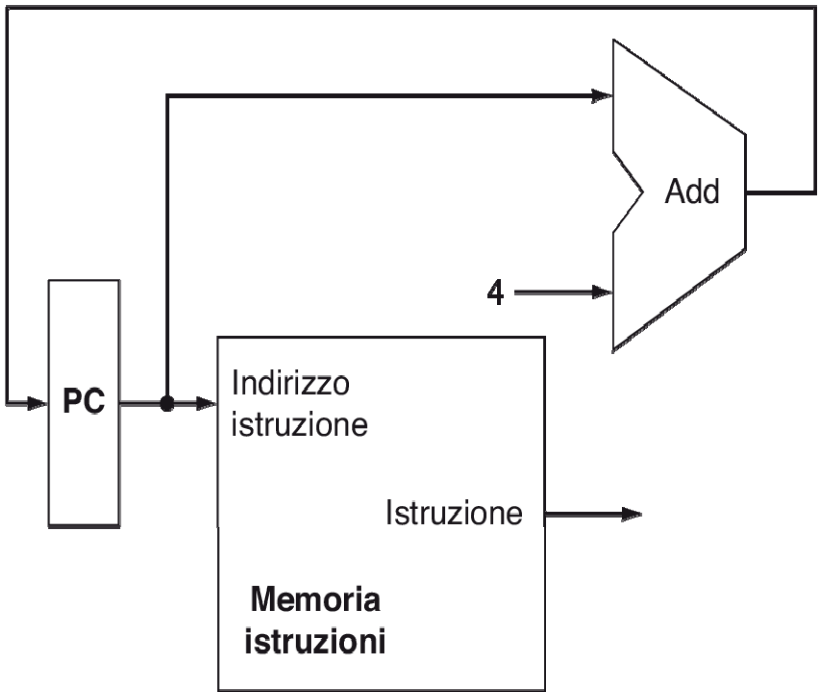
Gli operandi dell'***ALU*** sono utilizzati per:

- Calcolare un risultato aritmetico (per un'istruzione aritmetico-logica);
- Calcolare un indirizzo di memoria (per load/store);
- Effettuare un confronto (per un salto condizionato).

Il risultato dell'ALU è utilizzato per:

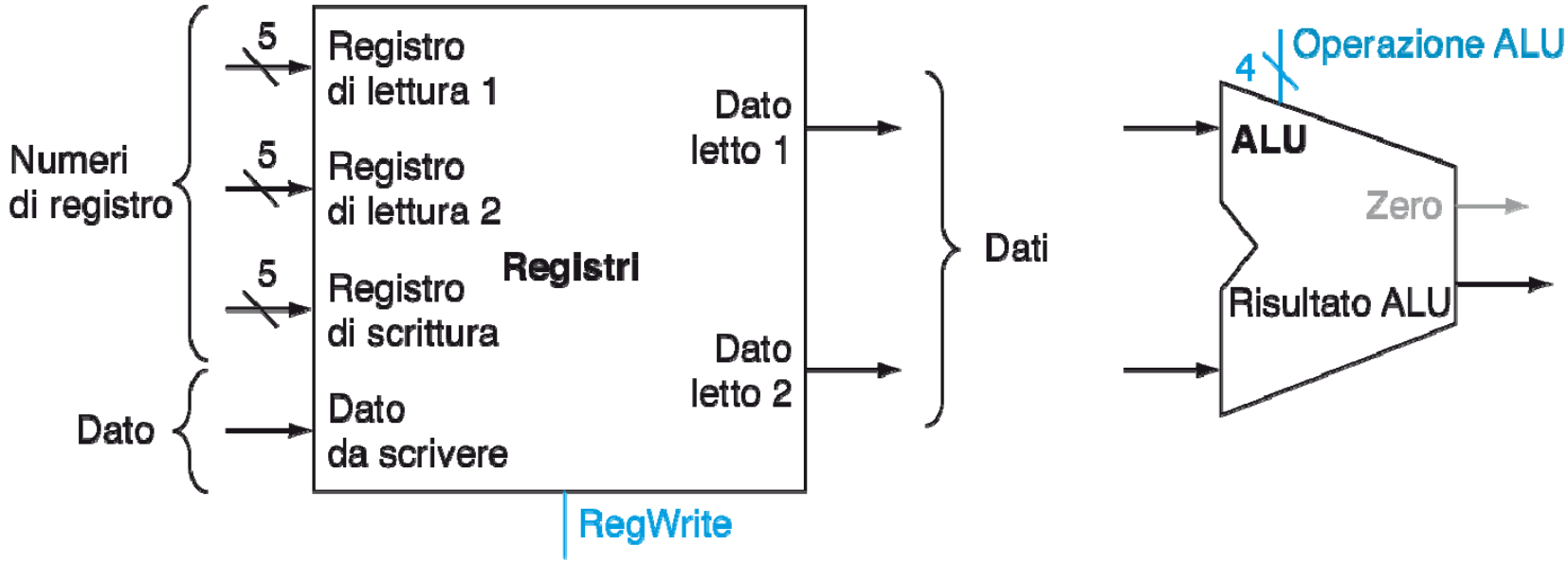
- Scrittura in un registro destinazione (se l'istruzione è aritmetico-logica)
- Essere usato come indirizzo di lettura/scrittura della Memoria Dati (se l'istruzione è load/store)
- Calcolare l'indirizzo della prossima istruzione, secondo un'apposita logica di controllo (se l'istruzione è di salto condizionato).

Esecuzione della fase di caricamento *(fetch)* delle istruzioni



- *Memoria Istruzioni*
- *Program Counter (PC)* per memorizzare l'indirizzo dell'istruzione corrente;
- Un *Sommatore* (*Add_seq*) per incrementare il *PC* in modo da poter indirizzare l'istruzione successiva *(PC + 4).*

Esecuzione delle istruzioni aritmetico-logiche



a. Registri

b. ALU

- *Register File*
- *ALU* a 32 bit che riceve 2 ingressi da 32 bit e che restituisce un risultato da 32 bit.

Esecuzione delle istruzioni aritmetico-logiche (cont.)

Le istruzioni aritmetico-logiche hanno come operandi **3 registri:** per ogni istruzione si devono leggere due parole di dati dal banco di registri e se ne deve scrivere una

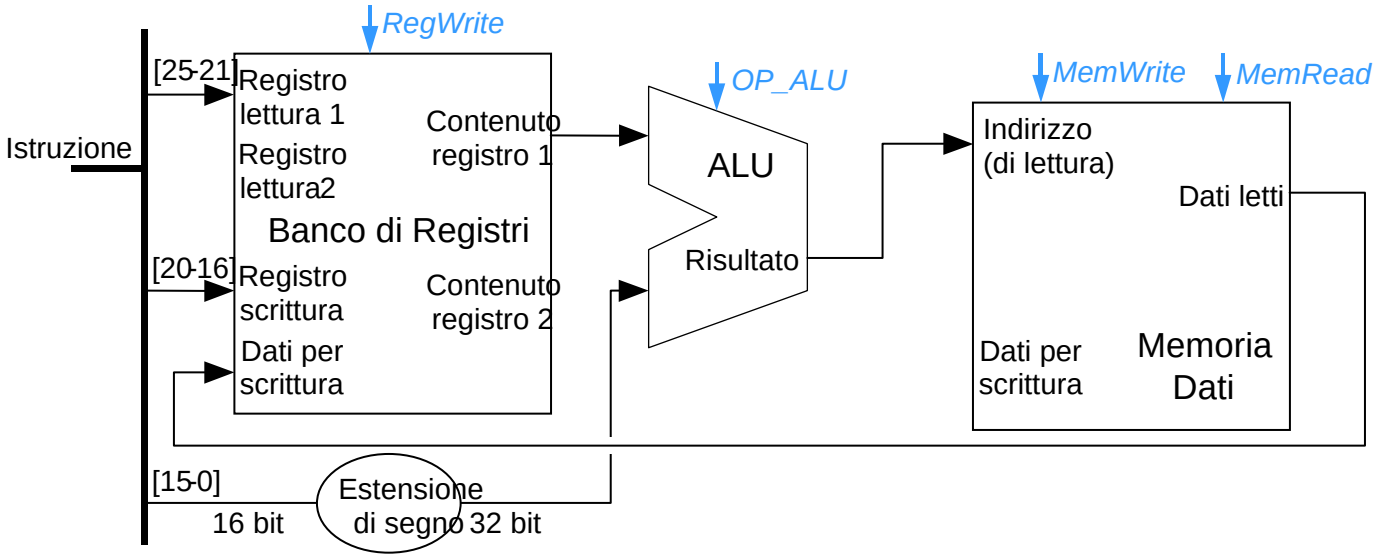
Per ogni parola letta dai registri sorgente sono necessari un ingresso *(5 bit)* al banco di registri, per specificare il numero del registro che si vuole leggere, ed un'uscita dal banco *(32 bit)* per il valore letto dal registro

Per scrivere una parola nel registro destinazione sono necessari due ingressi: un ingresso *(5 bit)* per specificare il numero del registro in cui si vuole scrivere ed un ingresso *(32 bit)* per il dato da scrivere

- Il banco di registri fornisce sempre in uscita il contenuto dei registri di lettura, mentre le scritture sono controllate da un apposito segnale di controllo della scrittura *(RegWrite)*

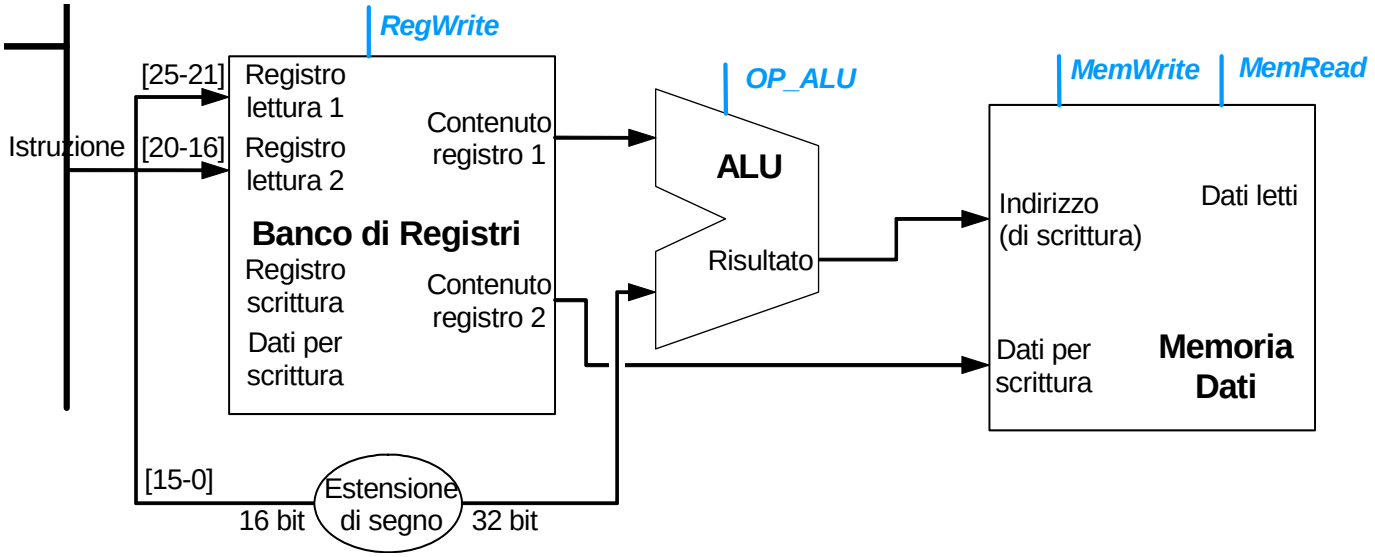
Il segnale di controllo *OP_ALU* provvede a specificare all'ALU il tipo di operazione.

Esecuzione delle istruzioni di *load*



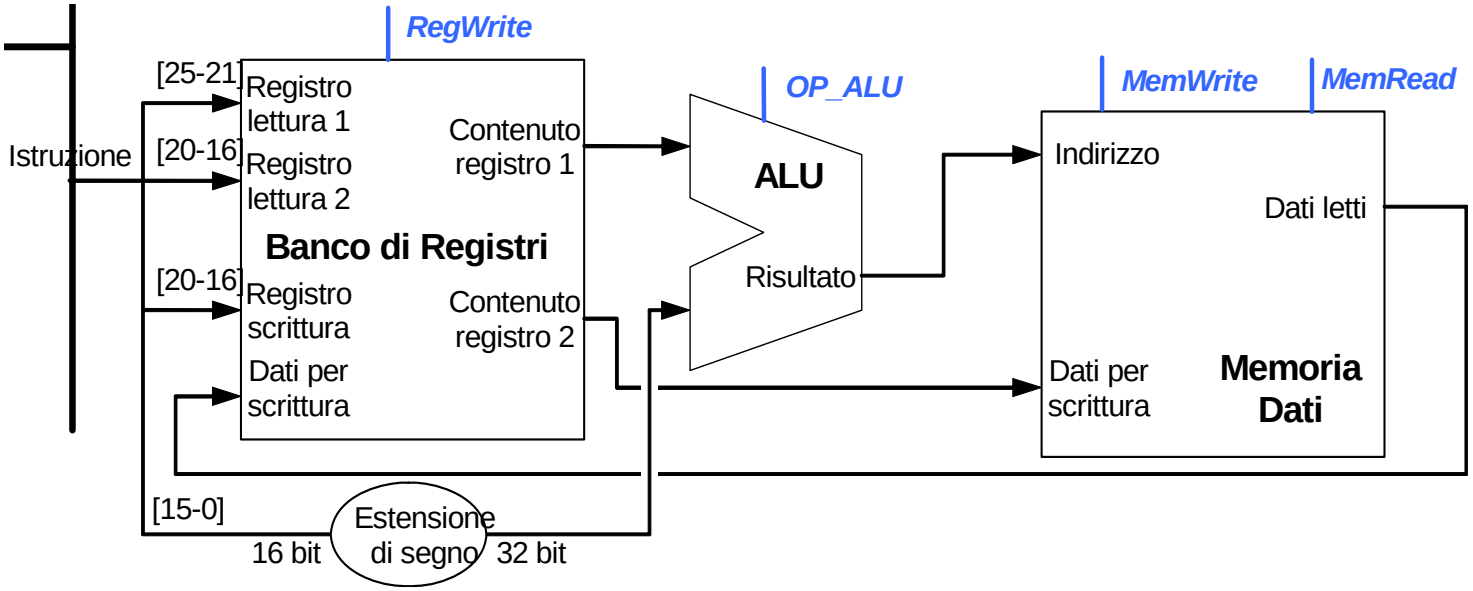
- *Register File*
- ALU a 32 bit per calcolare l'indirizzo
- *Memoria Dati* da cui leggere;
- Unità per estendere con il segno corretto il valore dello spiazzamento *(offset)* dai 16 bit contenuti nell'istruzione ad un valore con segno a 32 bit.

Esecuzione delle istruzioni di *store*



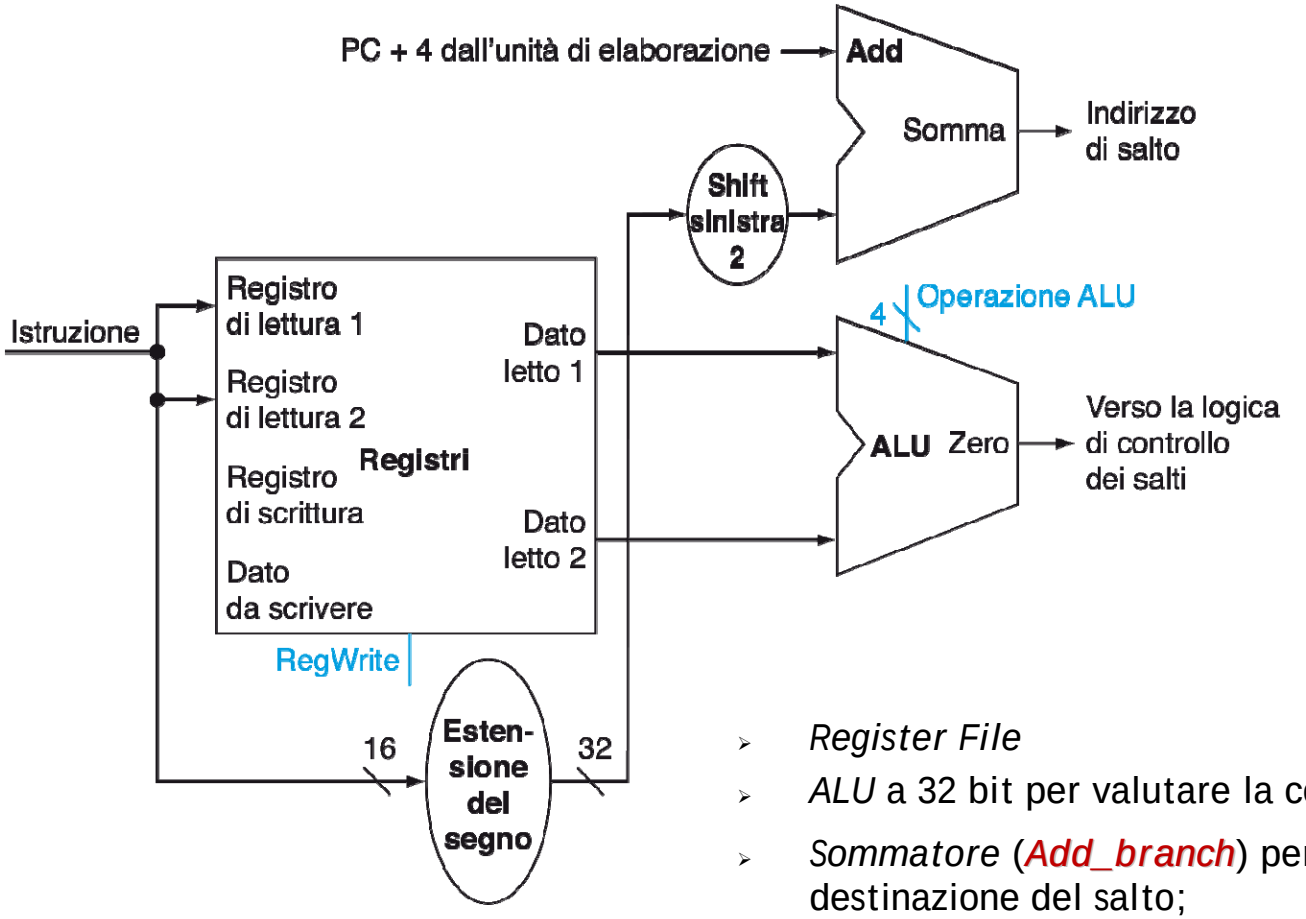
- *Register File*
- *ALU* a 32 bit per calcolare l'indirizzo
- Una *Memoria Dati*
- Unità per estendere con il segno corretto il valore dello spiazzamento *(offset)* dai 16 bit contenuti nell'istruzione ad un valore con segno a 32 bit

Esecuzione delle istruzioni di *load/store*



- Il valore dello spiazzamento *(offset),* dopo l'estensione di segno, è utilizzato come secondo operando dell'*ALU*;
- Ogni operazione è costituita da lettura del RF, calcolo nell'*ALU* di un indirizzo di lettura/scrittura per accedere alla *Memoria Dati,* lettura/scrittura della *Memoria Dati* e, nel caso di *load* , scrittura del *RF*.
- Durante la *load*, il valore letto dalla *Memoria Dati* deve essere scritto nel registro destinazione del *RF*. Durante la *store*, il valore da scrivere nella *Memoria Dati* deve essere letto dal registro sorgente del *RF*.

Esecuzione delle istruzioni di salto condizionato



- *Register File*
- *ALU* a 32 bit per valutare la condizione di salto
- *Sommatore* (*Add_branch*) per calcolare l'indirizzo di destinazione del salto;
- Unità per l'*estensione del segno* e uno **shifter** a sinistra di 2 bit;
- Logica di controllo per decidere, in base al valore dell'uscita *Zero* dell'*ALU*, il nuovo valore del *PC*.

Esecuzione delle istruzioni di salto condizionato (cont.)

La base per il calcolo dell'indirizzo di destinazione di un salto condizionato è l'indirizzo dell'istruzione che segue quella di salto (PC + 4).

Poiché ogni istruzione occupa 4 byte, prima di effettuare la somma dello spiazzamento *(offset)* con il contenuto di *(PC+4),* bisogna **moltiplicare per 4** il valore contenuto nell'istruzione
⇒ **scorrimento a sinistra di 2 bit.**

L'operazione di salto condizionato è costituita da due operazioni:

- Calcolo dell'indirizzo di destinazione del salto attraverso un sommatore che effettua la somma tra il *(PC+4)* e il valore contenuto nell'istruzione dopo avere esteso il segno e fatto scorrere a sinistra di 2 bit;
- Confronto nell'ALU del contenuto dei registri operandi letti dal *RF.*
  Se il segnale di uscita Zero dell'ALU è asserito ⇒ la condizione di salto è verificata e l'indirizzo di destinazione del salto diventa il nuovo *PC*.
  Se invece la condizione non è verificata ⇒ il *PC* incrementato sostituisce il PC attuale.

Realizzazione del processore completo

Esaminati gli elementi richiesti da ogni tipo di operazione, è possibile combinarli in un'unica unità di elaborazione.

Si assume che tutte le istruzioni siano eseguite in un solo ciclo di clock

- Nessuna risorsa può essere utilizzata più di una volta per istruzione
- Qualsiasi risorsa di cui si ha bisogno più di una volta deve essere duplicata

Occorre quindi una *Memoria Istruzioni* **distinta** dalla *Memoria Dati* (memorie cache)

Realizzazione del processore completo (cont.)

Alcune unità funzionali potrebbero essere **duplicate** nel momento in cui si combinano le varie unità di calcolo definite nella precedente sezione, mentre altre unità possono essere ***condivise*** da ***differenti flussi di istruzioni***

Per condividere un elemento tra due diversi tipi di istruzione, si deve introdurre un ***multiplexer*** di dati per permettere connessioni multiple all'ingresso di un elemento e selezionare uno tra i vari ingressi in base alla configurazione delle linee di controllo.

Esecuzione delle istruzioni aritmetico-logiche e *load/store*

Da risolvere:

1. Il secondo ingresso dell'ALU è il contenuto di un registro (istruzione di tipo R) oppure la metà meno significativa dell'istruzione (istruzione di *load/store o aritmetiche immediate*)

   ⇒ **MUX al secondo ingresso dell'ALU (*Mux A*)**

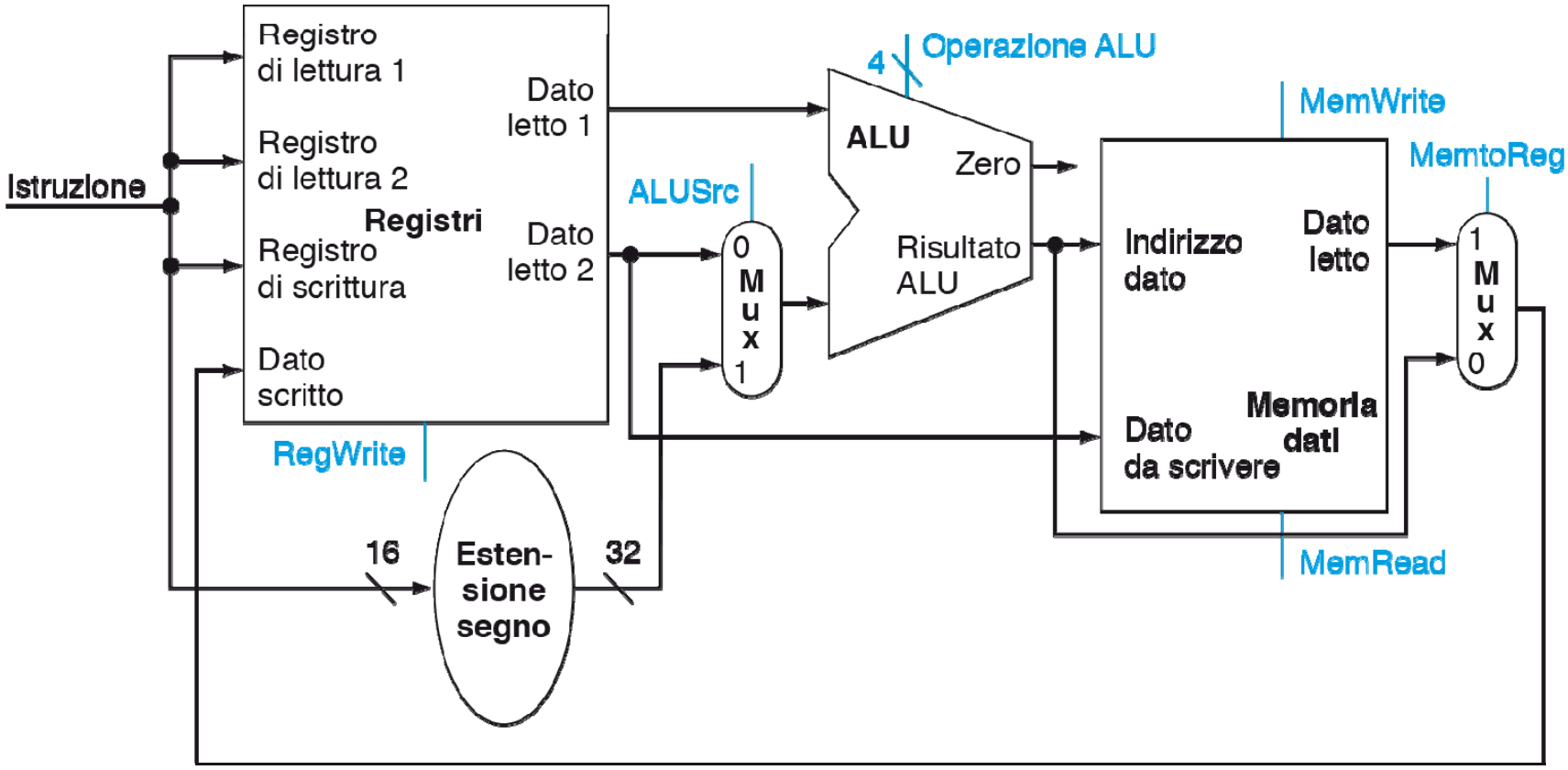
2. Il valore scritto nel registro destinazione proviene dal risultato dell'ALU (istruzione tipo R o aritmetica immediata) oppure dalla Memoria Dati (istruzione di *load*)

   ⇒ **MUX all'ingresso dei Dati per Scrittura del RF (*Mux C*)**

3. Il numero del registro in cui si vuole scrivere il risultato è indicato da diversi campi (i bit [15-11] per le istruzioni di tipo R e bit [20-16] per istruzioni di *load e aritmetiche immediate*)

   ⇒ **MUX all'ingresso del Registro Scrittura del RF (*Mux D*)**

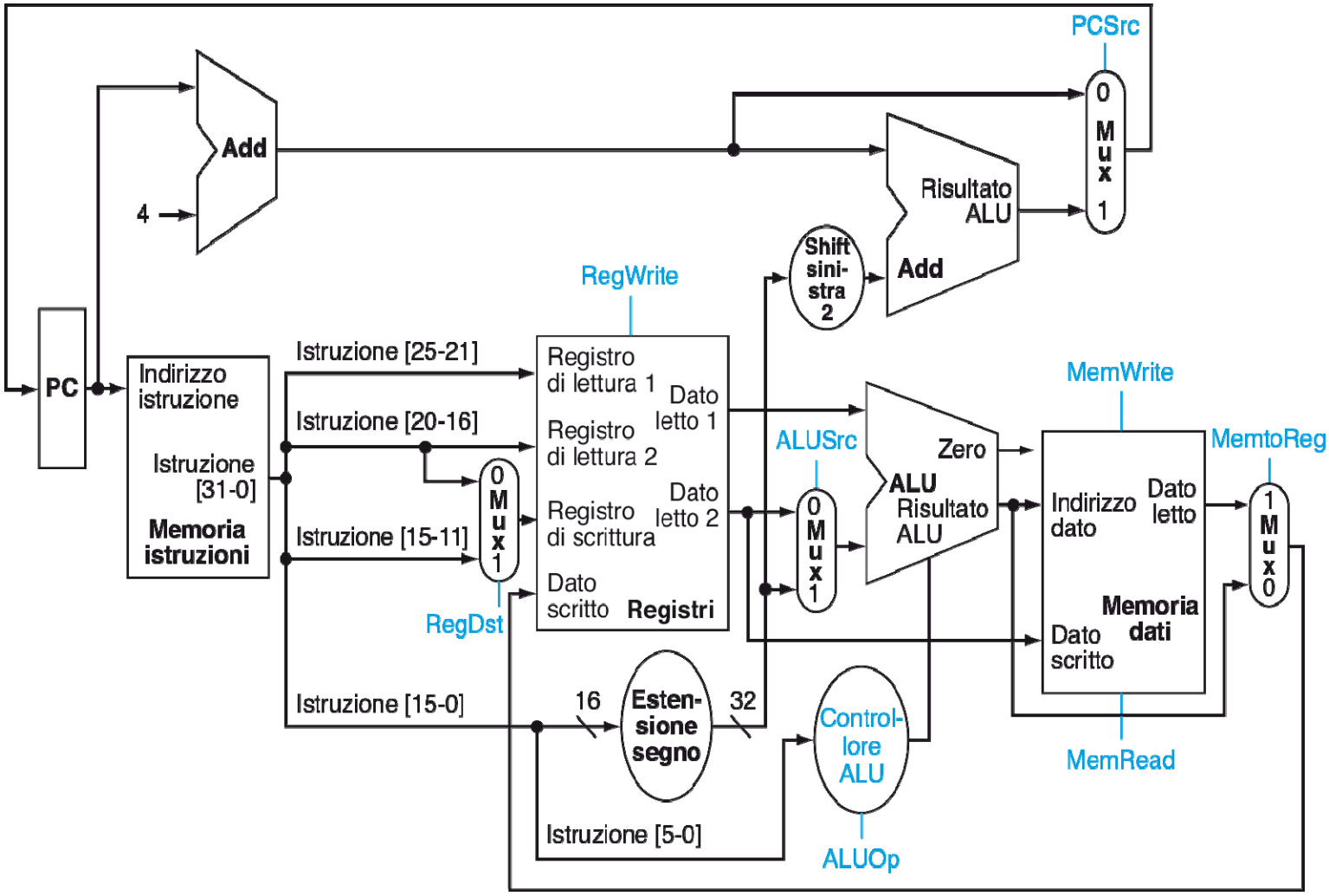
Esecuzione delle istruzioni aritmetico-logiche e *load/store* (cont.)



Non ancora risolto il «numero» del registro di scrittura

- istruzione aritmetica di tipo R, bit [15-11]
- istruzione di load, bit [20 -16]

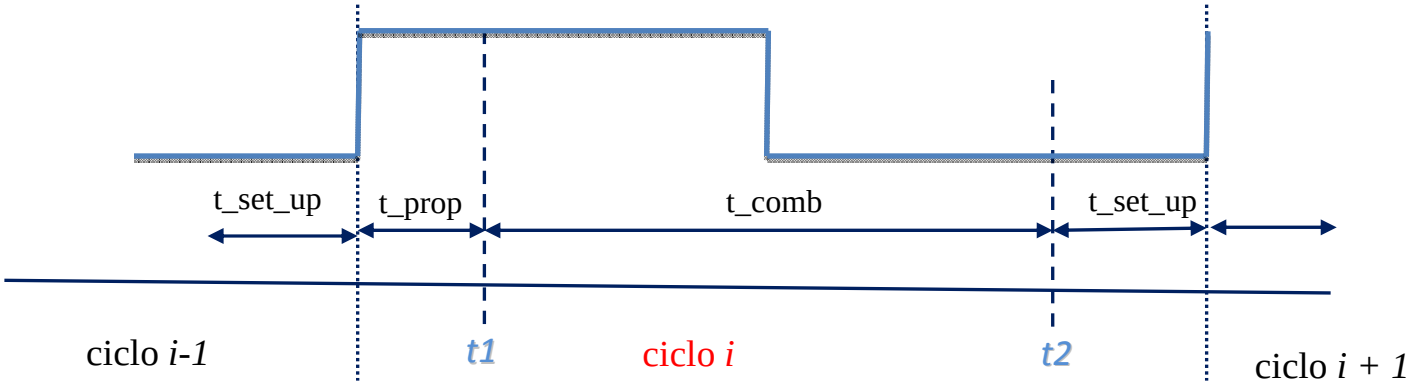
Struttura della CPU

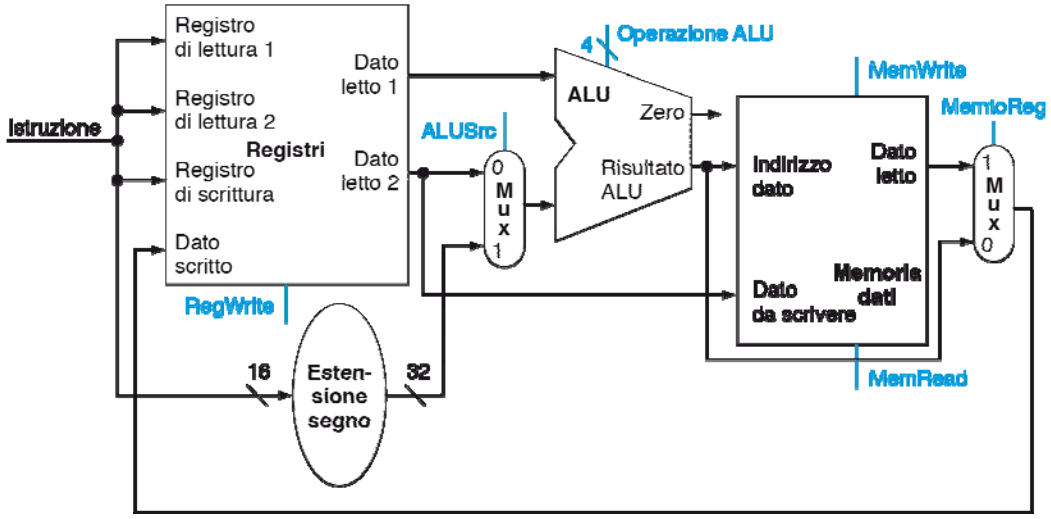


Nota sulla temporizzazione

Nelle istruzioni aritmetiche (R e I) un registro sorgente può anche essere destinazione!!!!

T_clock = T_prop + T_set_up + T_comb dove ***T_comb*** è sostanzialmente il tempo di operazione dell'ALU

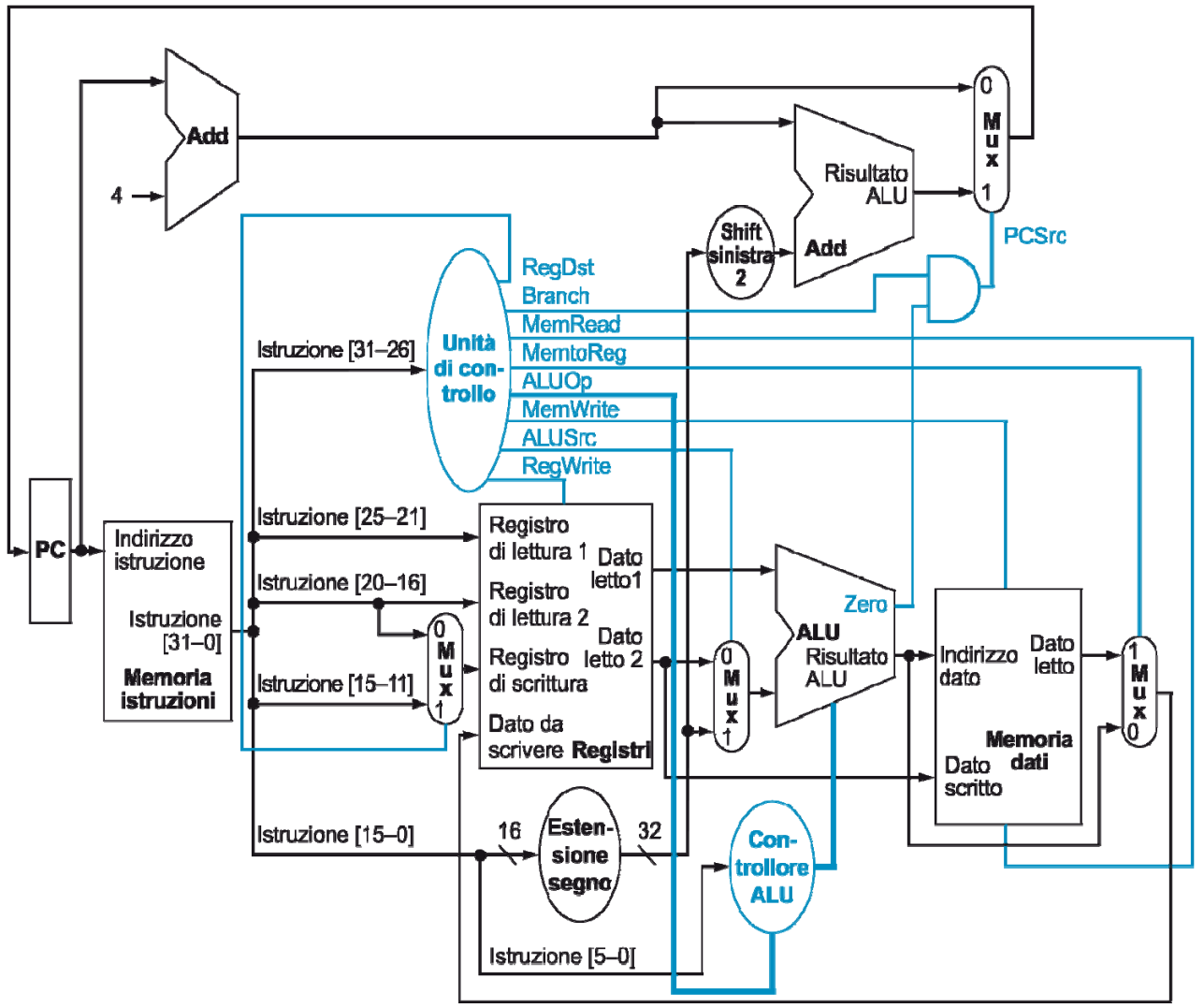




I segnali di controllo

Nome del segnale	Effetto quando non asserito	Effetto quando asserito
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	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 16 bit meno significativi 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	Nulla	Il dato della memoria nella posizione puntata dall'indirizzo viene inviato in uscita sulla linea «dato letto»
MemWrite	Nulla	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

Struttura della CPU con unità di controllo



Codice operativo e segnali di controllo

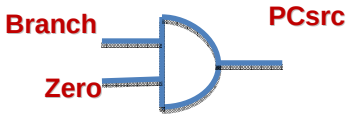
Istruzione	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
Tipo R	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Osservazione:
Nell'architettura di processore considerata le configurazioni delle ***4 linee di controllo dell'ALU*** vengono generate dal **controllore dell'ALU** che riceve come ingressi:
•i segnali ***ALUOp*** che dipendono dall' OP_CODE dell'istruzione
•i ***6 bit meno significativi dell'istruzione*** che sono rilevanti solo nel caso di ***istruzioni di tipo R*** (funct)

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

Questa implementazione non consente di specificare in modo completo le linee di controllo dell'ALU nel caso di istruzioni artmetico-logiche immediate

….. e per eseguire anche la bne?



La logica di controllo per l'istruzione di salto condizionato

PCsrc = Zero and Branch

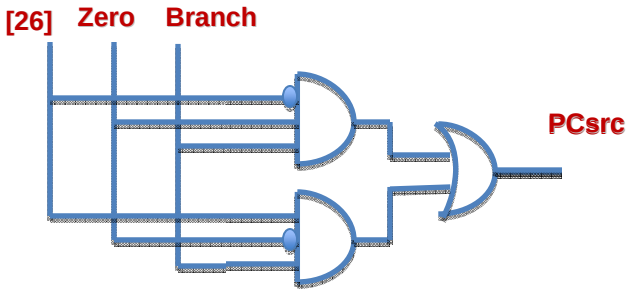
seleziona per la **beq** l'indirizzo della prossima istruzione da eseguire (PCsrc=1 salta, PCsrc=0, in sequenza)

- per **beq** si salta se il bit Zero = 1 e cioè se i due registri selezionati hanno valori identici
- per **bne** si salta se il bit Zero = 0 e cioè se i due registri selezionati **non** hanno valori identici

Dobbiamo modificare la logica di controllo (in modo banale!!!) che genera il segnale **PCsrc** tenendo conto dei bit di codice operativo [31-26] per distinguere **beq** da **bne**

- beq – Op_code = 4, in binario 000100
- bne – Op_code = 5, in binario 000101
- quindi differiscono per il bit [26]

PCsrc = (notOp_code [26] and Zero and Branch) or (Op_code [26] and notZero and Branch)



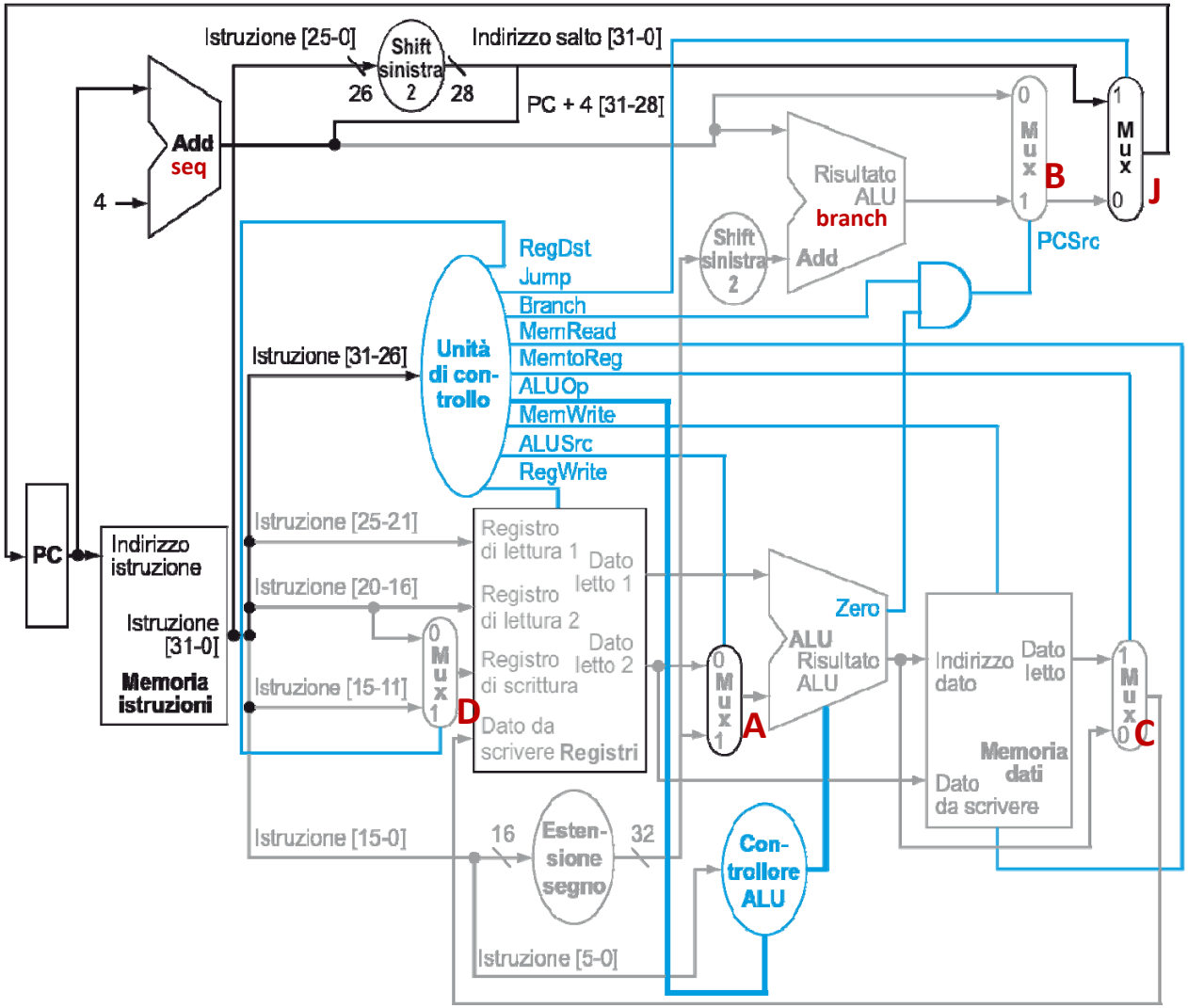
Implementazione delle istruzioni salto incondizionato (jump)

	6 bit	5 bit	5 bit	5 bit	5 bit	6 bit	Commenti
Formato J	op [31-26]	campo indirizzo [25-0]					istruzioni di salto

Calcolo dell'indirizzo di destinazione del salto:

4 bit	26 bit	2 bit
PC+4 [31-28]	campo indirizzo [25-0]	00

Estensione CPU e unità di controllo per istruzioni di jump



Implementazione della CPU a singolo ciclo

Si avvia all'esecuzione un'istruzione per ogni ciclo di clock e ogni istruzione deve essere iniziata e completata in un solo ciclo di clock

Il ciclo di clock deve avere la stessa lunghezza per ogni istruzione *(CPI = 1)*

La lunghezza del ciclo di clock è determinata dal percorso più lungo (percorso critico o *critical path*): nell'esempio l'istruzione di load che utilizza 5 unità funzionali in serie e richiedere *T = 800 ps (f = 1250 MHz)*

L'implementazione su singolo ciclo non è molto veloce perché le altre istruzioni potrebbero essere implementate con un ciclo di clock più breve: nell'esempio le istruzioni tipo R richiedono *T = 600 ps,* le istruzioni di store *T = 700 ps,* istruzioni di salto condizionato *T = 500 ps* e istruzioni di salto incondizionato può essere considerato pari a 200 ps