Progetto CPU a singolo ciclo

Proff. A. Borghese, F. Pedersini

Dipartimento di Informatica
Università degli Studi di Milano

Sommario

- La CPU
- Sintesi di una CPU per le istruzioni di tipo R
- Sintesi di una CPU per le istruzioni di tipo I (memoria)
- Sintesi di una CPU per le istruzioni di tipo I (salti)
- CPU che gestisce istruzioni: lw/sw e branch

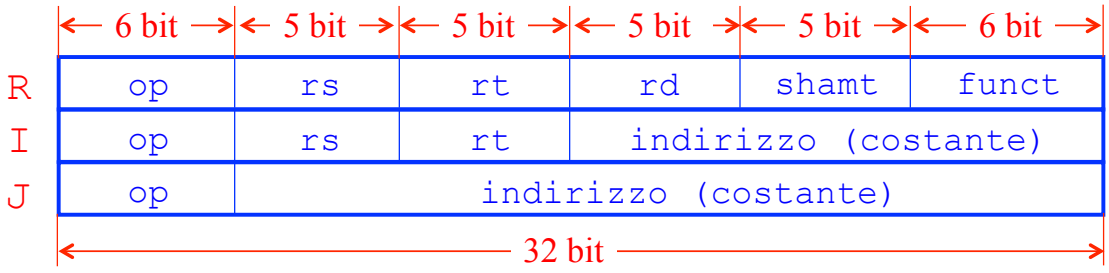
Obiettivo

- Obiettivo: costruzione di una CPU completa
  - In grado di eseguire:
  - Istruzioni logico-matematiche (e.g. **`add`**, **`sub`**, **`and`**....).
  - Accesso alla memoria in lettura (**`lw`**) o scrittura (**`sw`**).
  - Istruzioni di salto condizionato (**`branch`**) o incondizionato (**`jump`**)

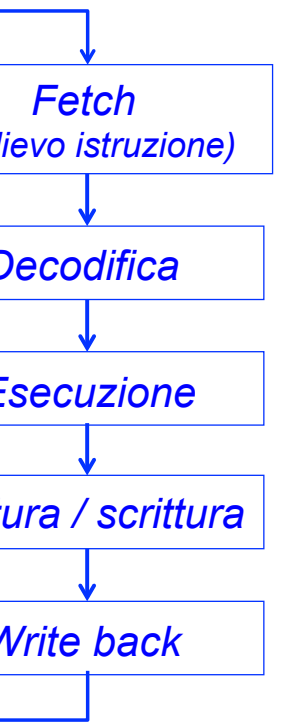
- Per la sintesi dei circuiti è necessario definire:
  - I formati di istruzione: **tipo R, tipo I, tipo J**
  - Ciclo esecuzione istruzioni: Fetch, Decodifica, Esecuzione

Riepilogo: Formato istruzioni

- **3 diversi formati: R, I, J**
- Formato riconosciuto dalla CPU tramite il valore del primo campo:
  ***codice operativo* (*opcode*) – (6 bit più significativi)**
  - che indica alla macchina come trattare i rimanenti bit dell'istruzione.



Ciclo di esecuzione di istruzioni MIPS



1. **FETCH**: preleva l'istruzione dalla memoria e la inserisce in IR
2. **DECODIFICA**: capisce di che tipo di istruzione si tratta
   - usa l'istruzione per decidere cosa fare
   - recdupera gli operandi, contenuti nei registri del Register File
3. **ESECUZIONE**: da qui le istruzioni si differenziano
   - **Calcolo**: utilizzo dell'ALU dopo aver letto i registri:
     - per calcolare l'indirizzo in memoria
     - per eseguire un'operazione logico-aritmetica
     - per effettuare un test.
   - **Accesso** alla memoria
   - **Scrittura** del risultato nel register file

Rapporto UC – Dati

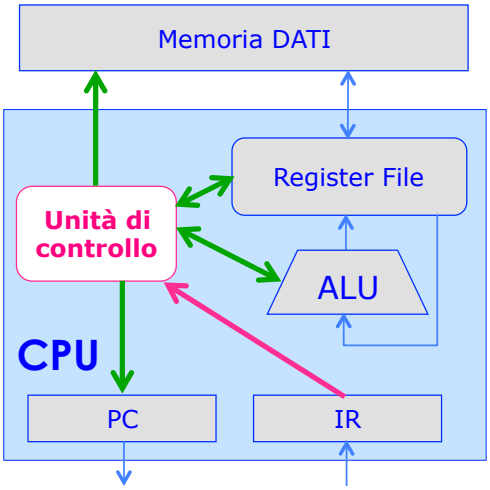
Approccio alla progettazione alla CPU: **Controllore – Data-path**

- **Fasi comuni:** (nel ciclo di esecuzione)
  Fetch, Decodifica (generazione dei segnali di controllo)
- **Fasi specifiche:**
  Esecuzione, Accesso memoria, WriteBack

L'unità di controllo

Controllore: Unità di controllo (UC)
coordina i flussi di informazione (è il "cervello" della CPU):

- selezionando l'operazione opportuna delle ALU.
- abilitando le vie di comunicazione opportune a seconda dell'istruzione in corso di esecuzione.



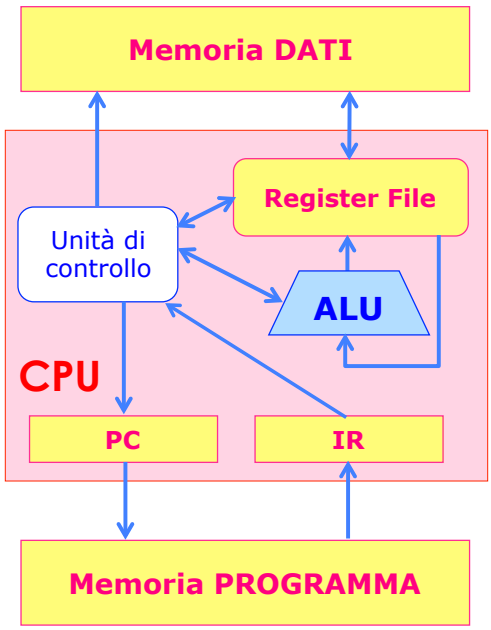
I componenti di un'architettura – Datapath

CPU:

- **Banco di registri (Register File)** : posso **leggere 2 registri** e **scriverne uno** contemporaneamente.
- Registro **Program counter (PC)** Contiene l'indirizzo dell'istruzione da eseguire.
- Registro **Instruction Register (IR)** Contiene l'istruzione in corso di esecuzione.
- **Arithmetic Logic Unit (ALU):** Unità per l'esecuzione delle operazioni aritmetico-logiche.
- **Unità di controllo (UC):** controlla il flusso e determina le operazioni di ciascun blocco.

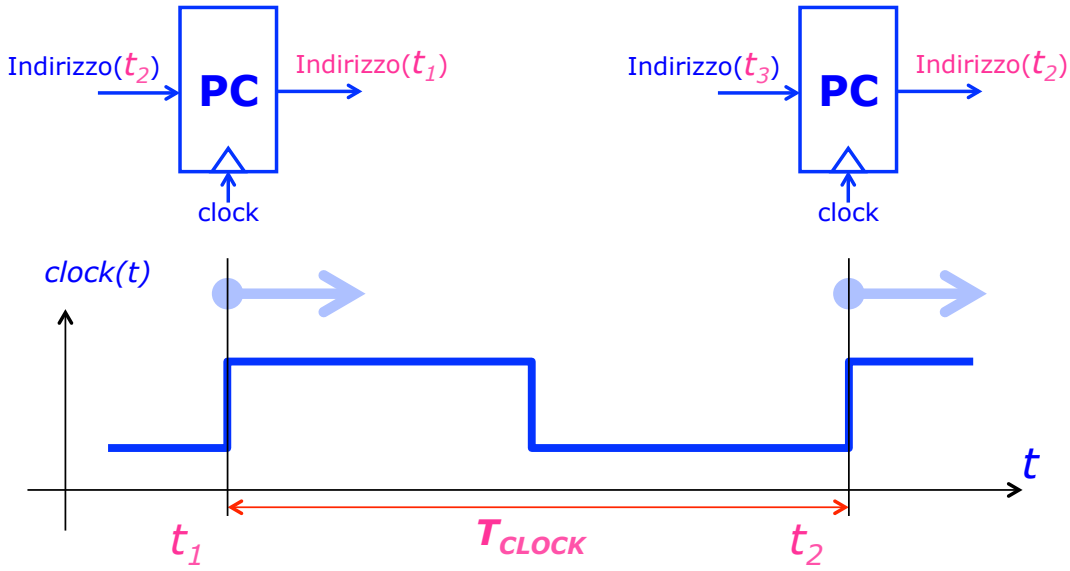
2 MEMORIE:

- Memoria Programma (contiene le istruzioni)
- Memoria Dati (per contenere dati e risultati)



CPU singolo ciclo: temporizzazione

CPU **singolo ciclo** = **1 istruzione/ciclo** di clock



Sommario

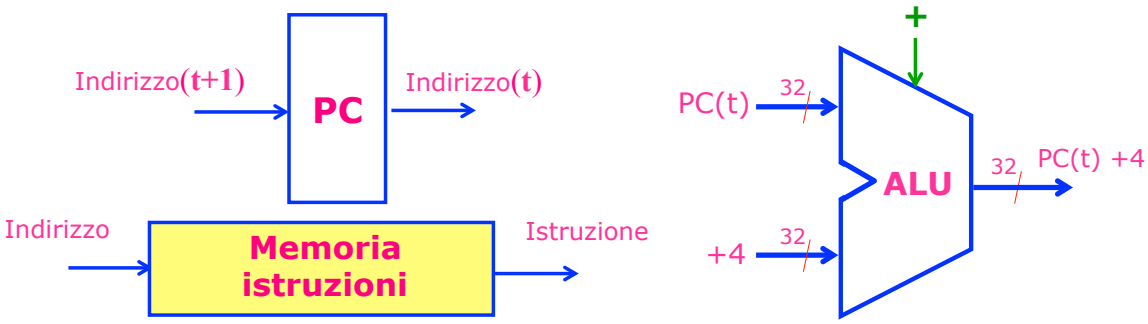
- La CPU
- Sintesi di una CPU: fase di fetch
- Sintesi di una CPU per le istruzioni di tipo R
- Sintesi di una CPU per le istruzioni di tipo I (memoria)
- Sintesi di una CPU per le istruzioni di tipo I (salti)
- CPU che gestisce istruzioni: lw/sw e branch

Fase di FETCH (prelievo dell'istruzione)

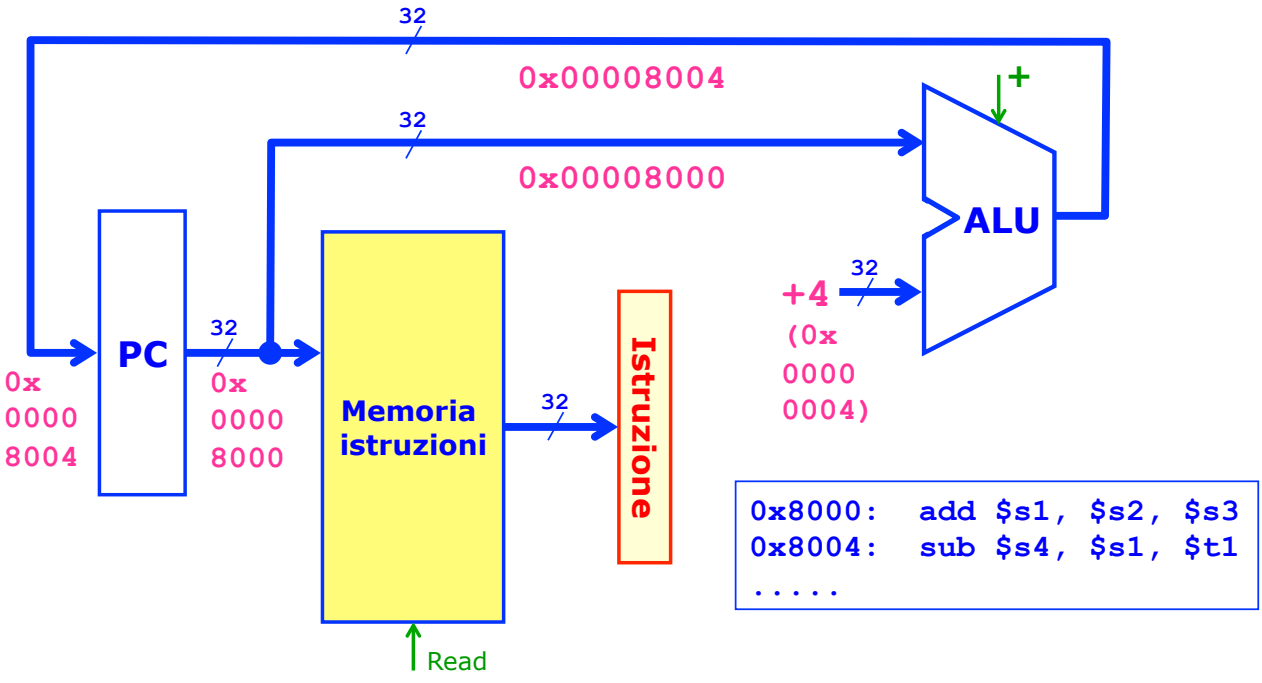
- Memorizzare l'indirizzo dell'istruzione nel Program Counter.
- Leggere l'istruzione dalla memoria di programma.
- Aggiornare l'indirizzo in modo che in PC sia contenuto l'indirizzo dell'istruzione successiva:

$$PC \leftarrow PC + 4$$

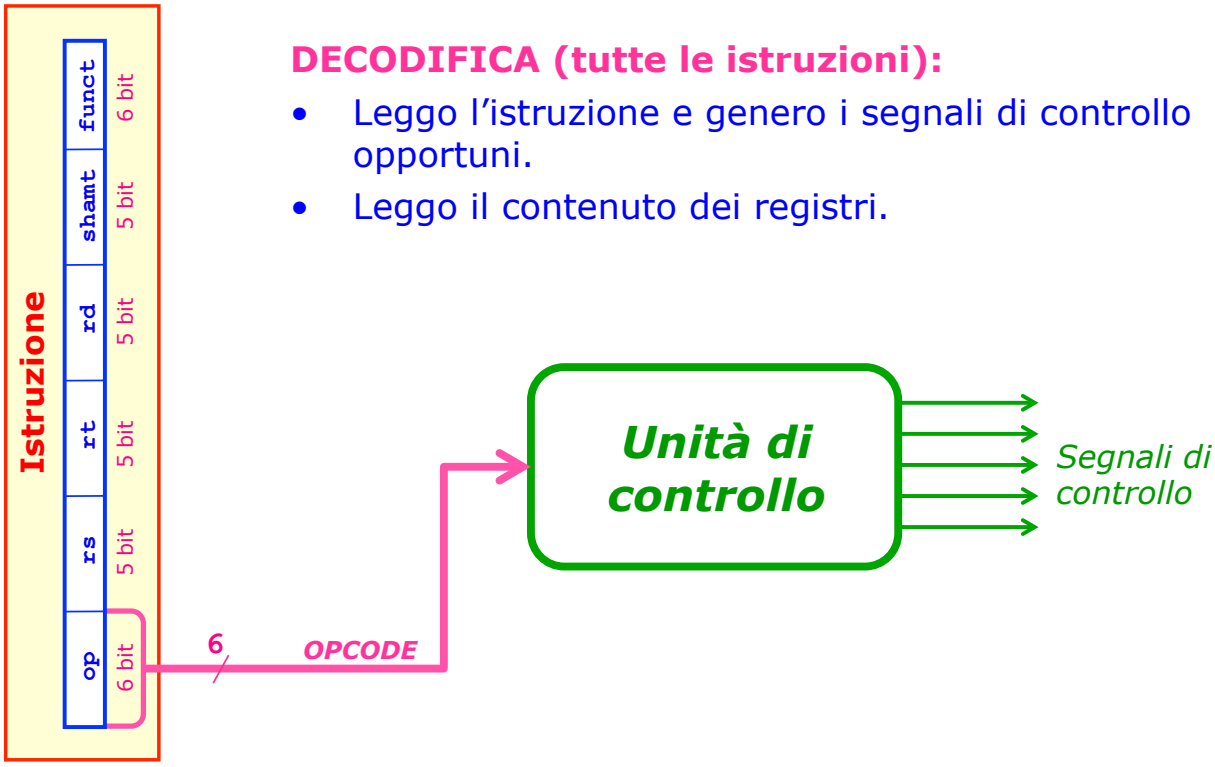
- Dispositivi coinvolti:



Datapath: fase di FETCH (tutte le istruzioni)



Fase di decodifica



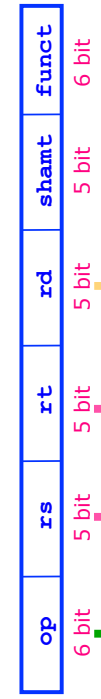
DECODIFICA (tutte le istruzioni):

- Leggo l'istruzione e genero i segnali di controllo opportuni.
- Leggo il contenuto dei registri.

Sommario

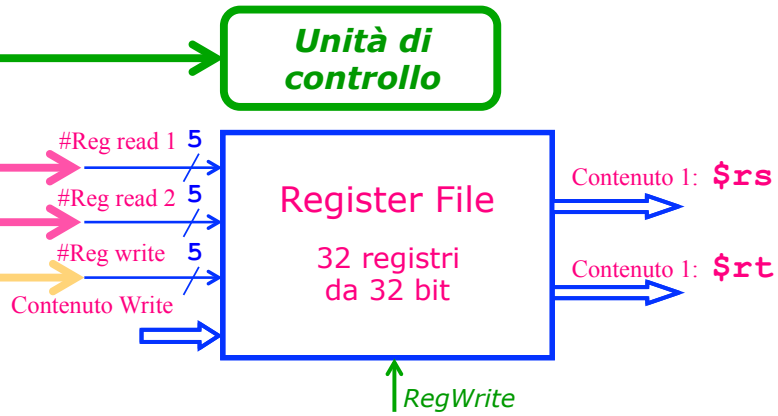
- La CPU
- Sintesi di una CPU: fase di fetch
- Sintesi CPU per le istruzioni A/L di tipo R
- Sintesi CPU per le istruzioni di tipo I – trasf. memoria
- Sintesi CPU per le istruzioni di tipo I – salti
- Sintesi CPU complessiva

Lettura dei registri (istr. tipo R)



A/L tipo R:

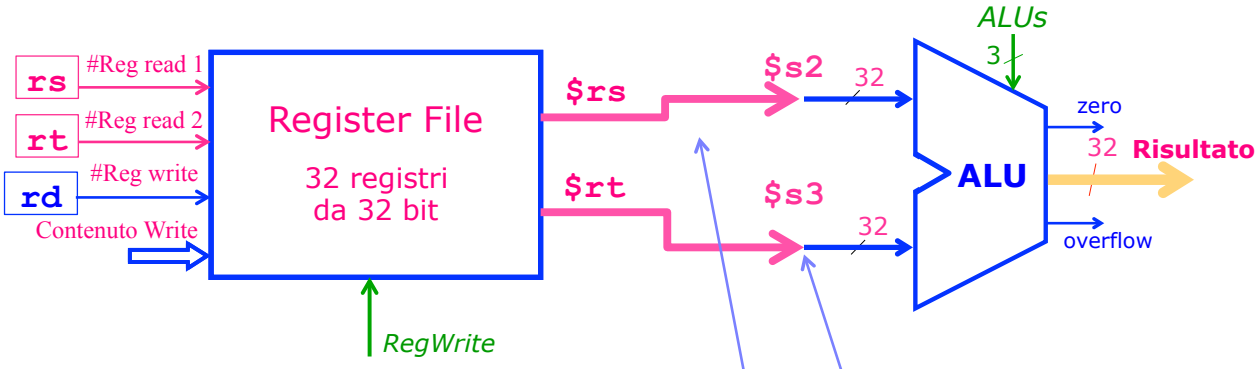
1) **FETCH**: leggo l'istruzione.
2) **Decodifica**: leggo OpCode e contenuto dei registri.




```
8000    add $s1, $s2, $s3
8004    sub $s4, $s1, $t1
.....
```


Fase di **Esecuzione** (tipo R)

- I dati vengono letti dal Register File e immessi nella ALU
  - Necessità della doppia porta di uscita del Register File
- La ALU viene comandata in modo opportuno (ALU_S)

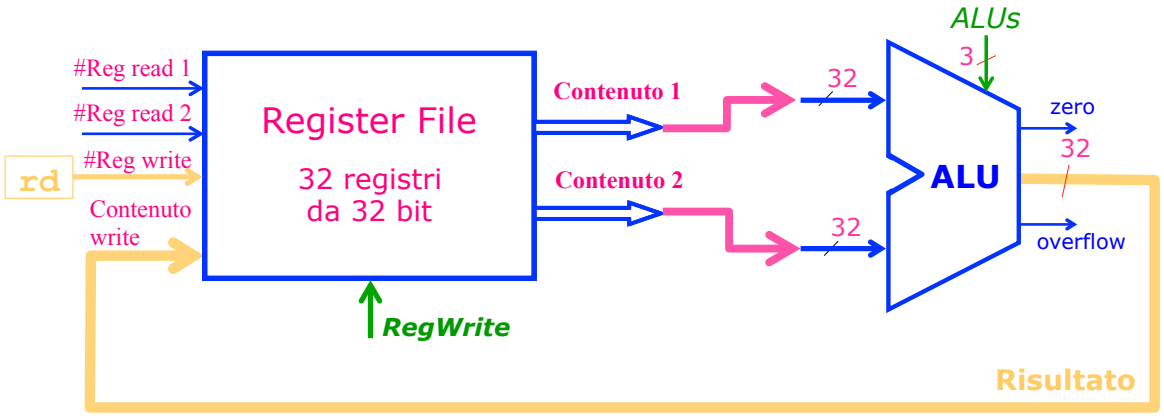



```
8000:    add $s1, $s2, $s3
.....
```

Fase di **Write back** (tipo R)

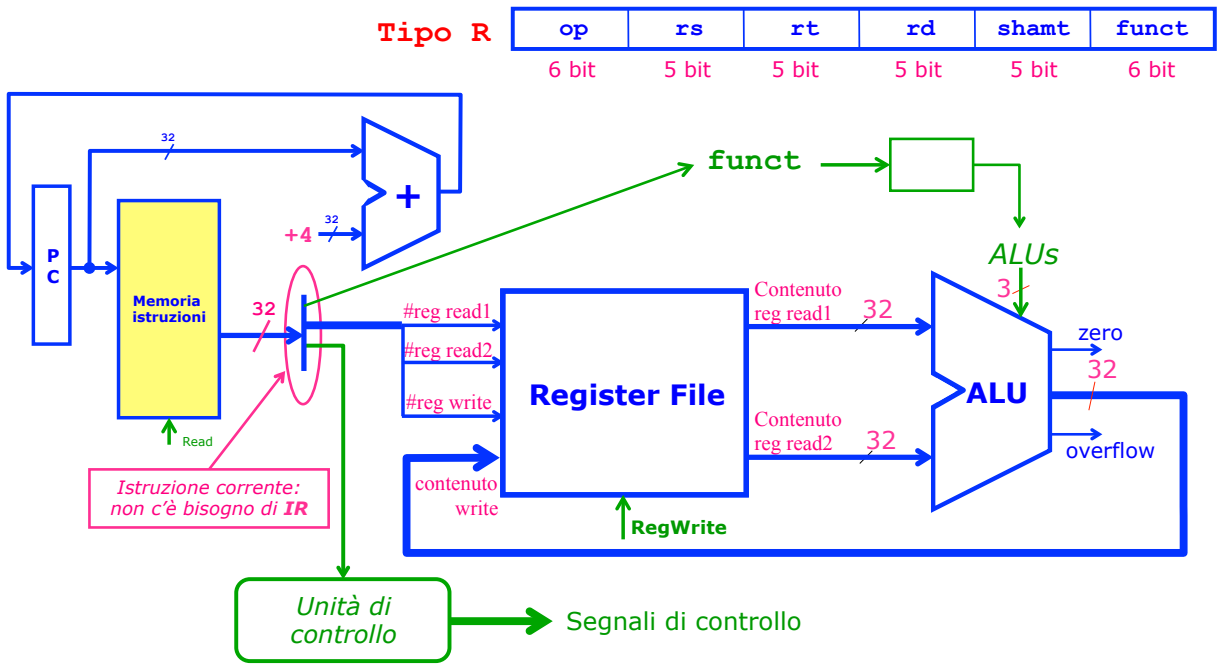
Write-back:

- La ALU ha eseguito l'operazione (ALU_S)
- Il risultato viene memorizzato nel register file
  - Comandi: Write (**W**), #reg_write



CPU per l'esecuzione di istruzione A/L tipo R

Esecuzione: istruzione aritmetico/logica tipo R:



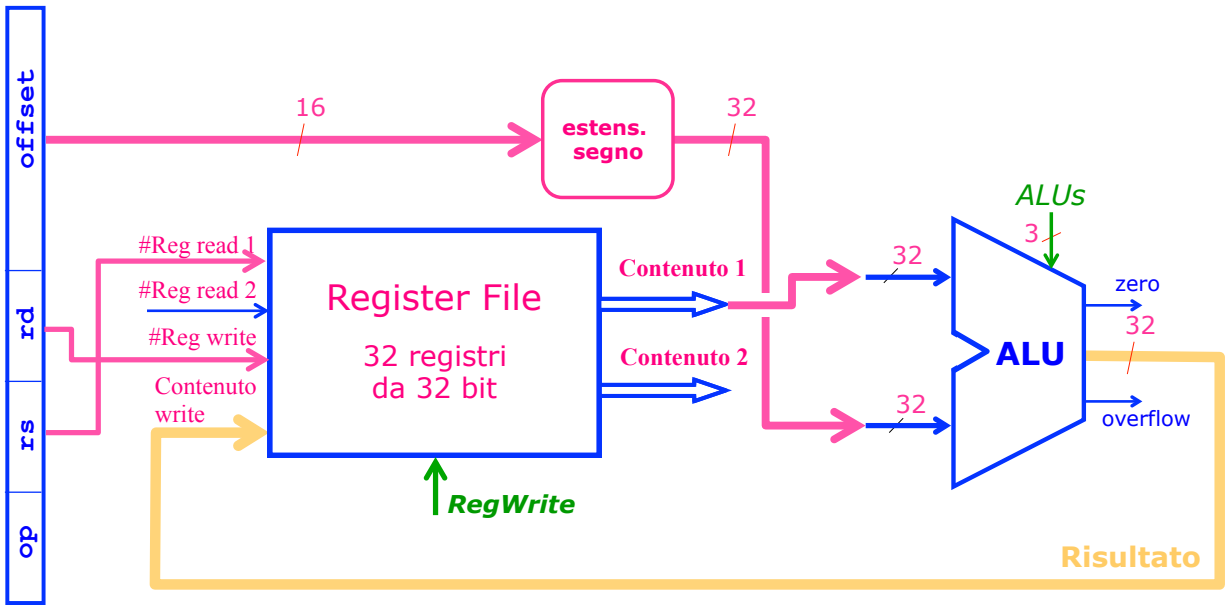
Sommario

- La CPU
- Sintesi di una CPU: fase di fetch
- Sintesi CPU per le istruzioni A/L di **tipo I**
- Sintesi CPU per le istruzioni di tipo I – trasf. memoria
- Sintesi CPU per le istruzioni di tipo I – salti
- Sintesi CPU complessiva

Fase di *Esecuzione / write back (A/L tipo I)*

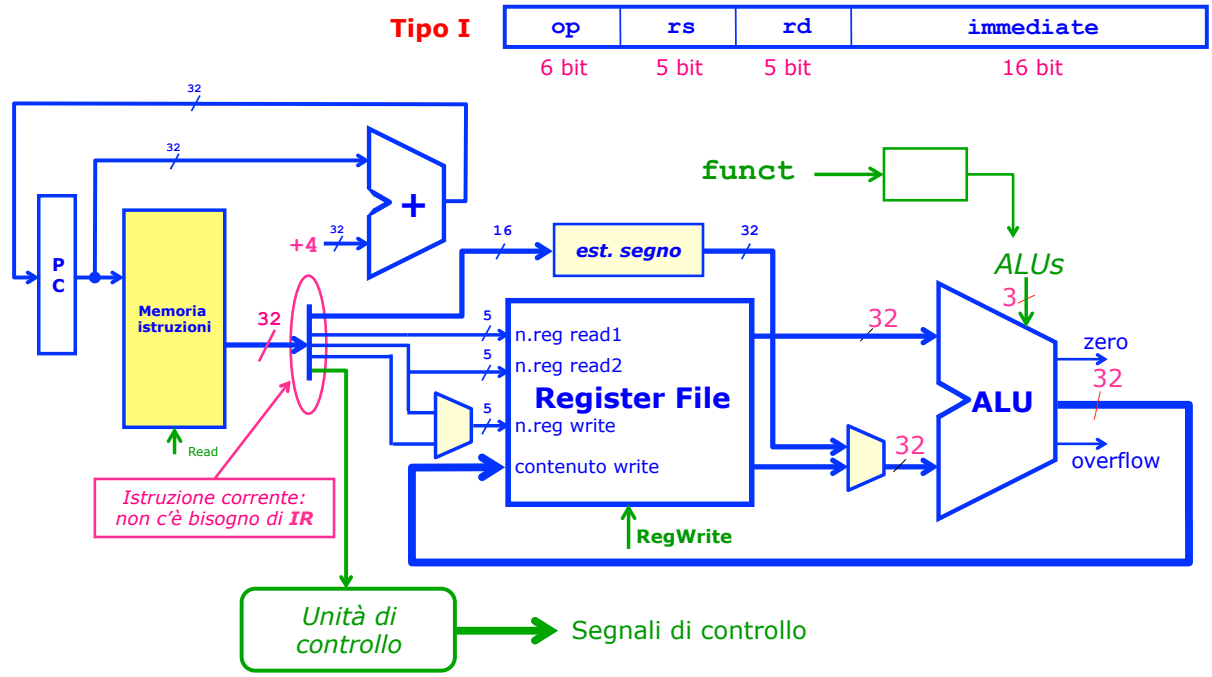
A/L tipo I:

- Cambiano: – l'origine del **secondo operando**
  – l'origine del **campo di selezione di `rd`** (bit: 16-20)



CPU per l'esecuzione di istruzione A/L tipo R

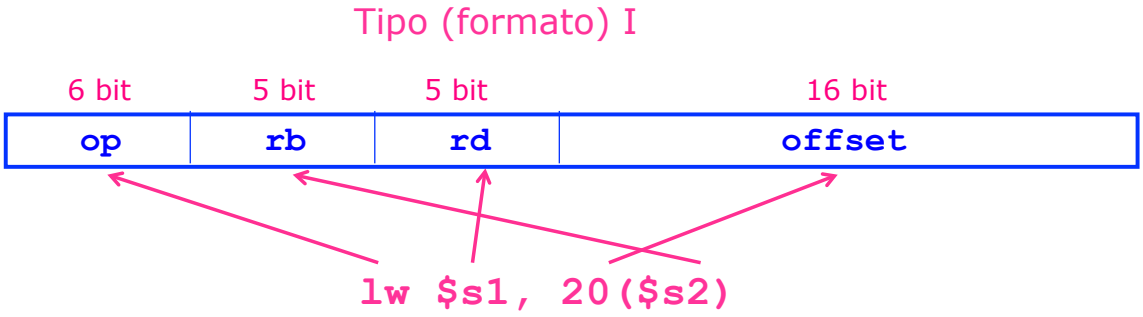
*Esecuzione: istruzione **aritmetico/logica, tipo R + tipo I**:*



Sommario

- La CPU
- Sintesi di una CPU: fase di fetch
- Sintesi CPU per le istruzioni A/L di tipo R
- Sintesi CPU per le istruzioni di tipo I – trasf. memoria
- Sintesi CPU per le istruzioni di tipo I – salti
- Sintesi CPU complessiva

Istruzioni di tipo I – memoria: `lw/sw`

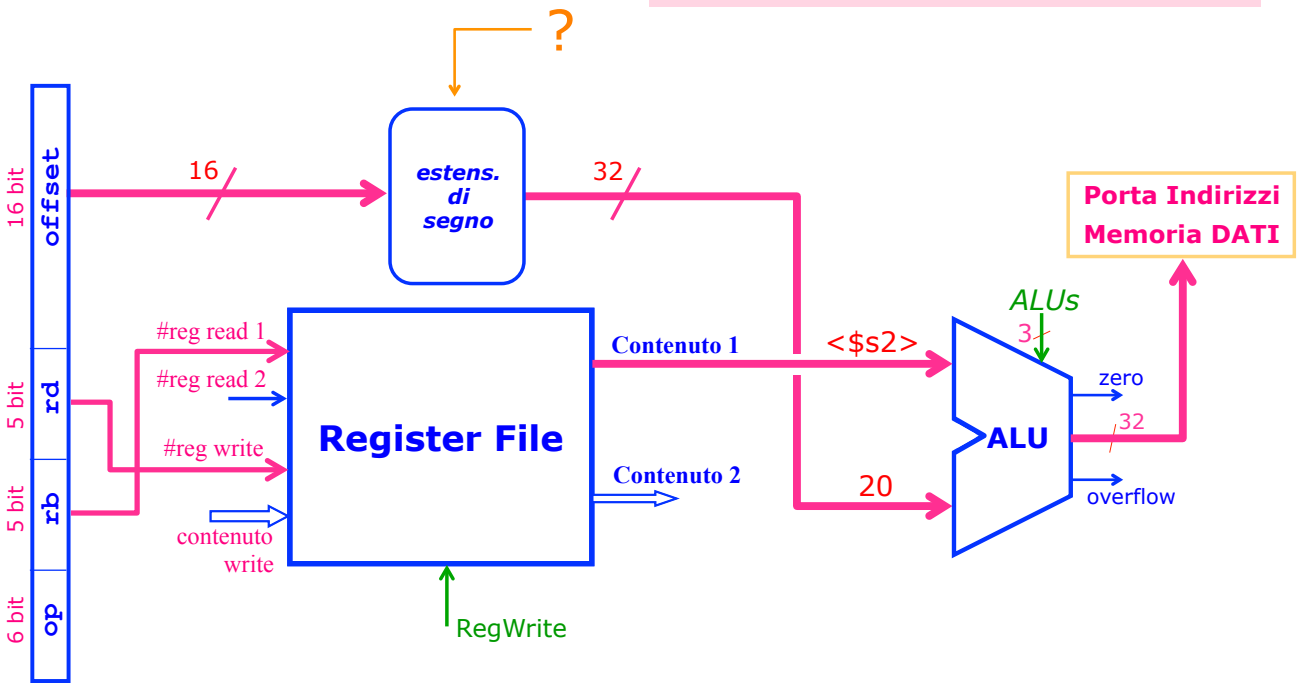


L'indirizzo di memoria deve essere:

`rb ($s2)`	0100 1000 0011 0001 1011 1011 1011 1011 +
`+ offset`	0000 0000 0001 0100 +
Indirizzo:	0100 1000 0011 0001 1011 1011 1100 1111

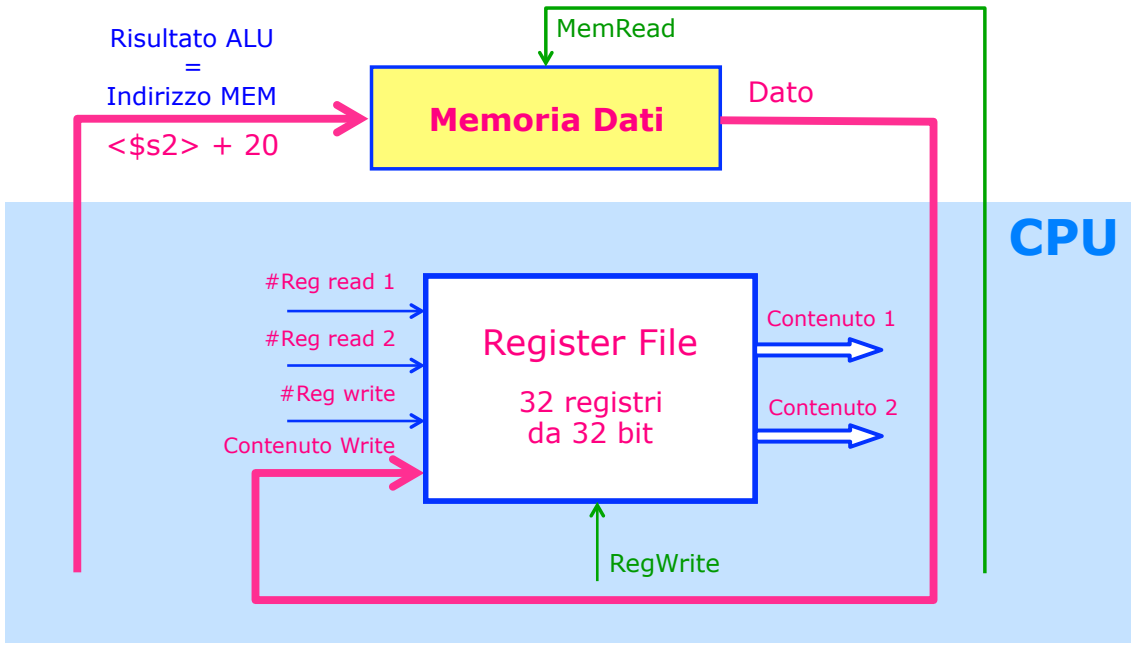
Fase di esecuzione (tipo I: `lw`)

`8000: lw $s1, 20($s2)`



Lettura della memoria: write-back


```
8000:   lw $s1, 20($s2)
```

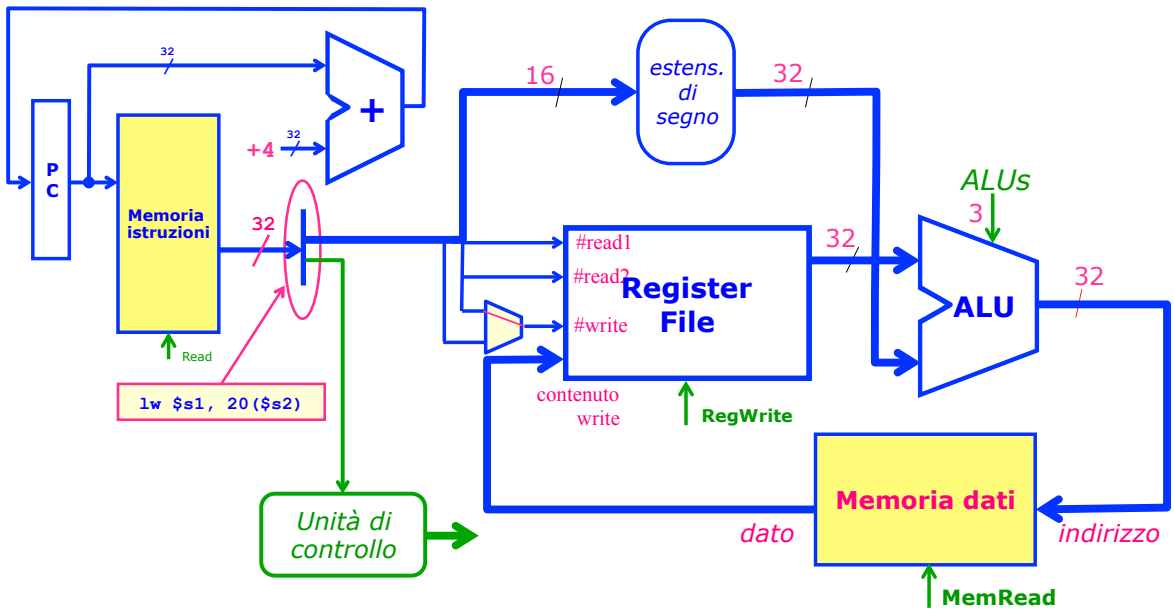


CPU per l'esecuzione di una lw

Datapath completo

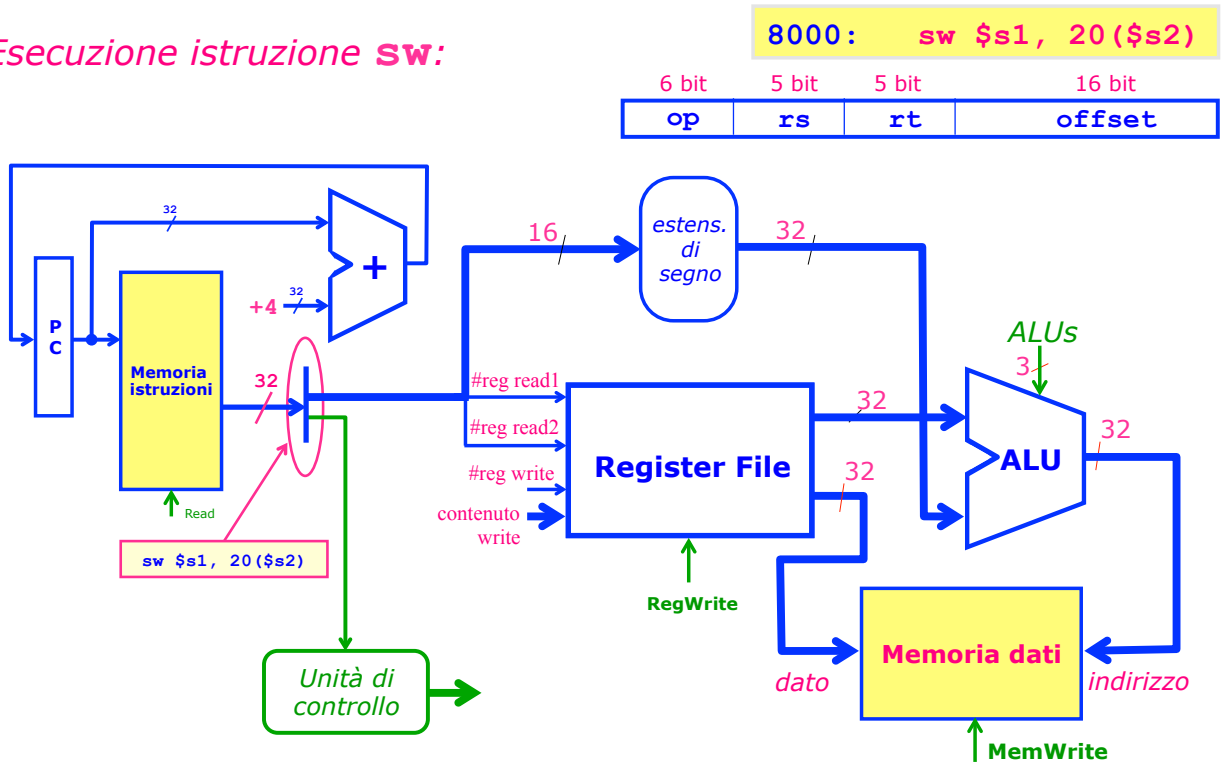
istruzione **lw:**

```
8000:   lw $s1, 20($s2)
```



CPU per l'esecuzione di una `sw`

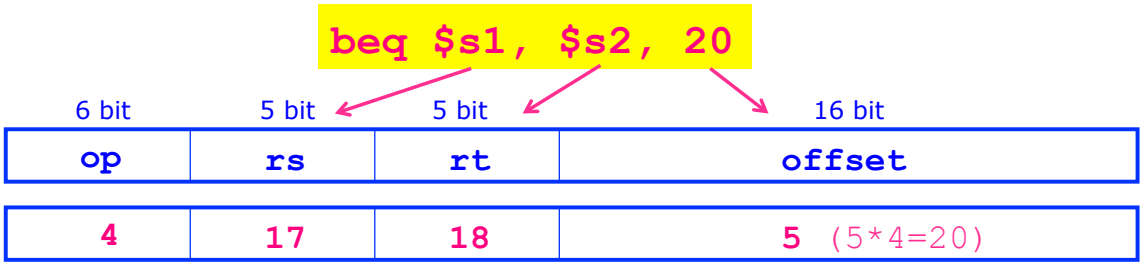
Esecuzione istruzione **`sw`**:



Sommario

- La CPU
- Sintesi di una CPU: fase di fetch
- Sintesi CPU per le istruzioni A/L di tipo R
- Sintesi CPU per le istruzioni di tipo I – trasf. memoria
- Sintesi CPU per le istruzioni di tipo I – salti (branch)
- Sintesi CPU complessiva

Istruzioni di tipo I: branch



1. Calcolo dell'indirizzo di salto:
- calcolo dell' offset in byte: `offset*4` `20 = 5*4:` 101_2 → 10100_2
- somma dell'offset in byte al contenuto del Program Counter:

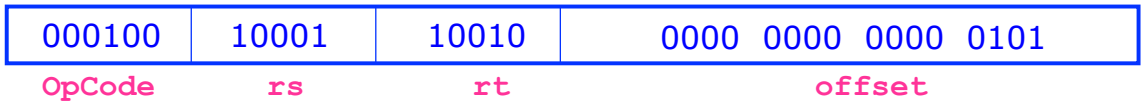
```
Program Counter:   0100 1000 0011 0001   1011 1011 1011 0100 +
Offset*4:          0000 0000 0000 0000   0000 0000 0001 0100 =

Indirizzo salto:   0100 1000 0011 0001   1011 1011 1100 1000
```

2. BRANCH:
- valutare l'uguaglianza e
- saltare se è vera.

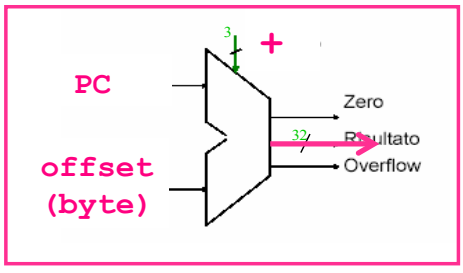
Fase di esecuzione della beq

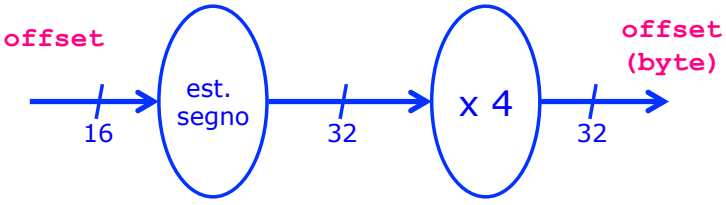
beq $s1, $s2, 20

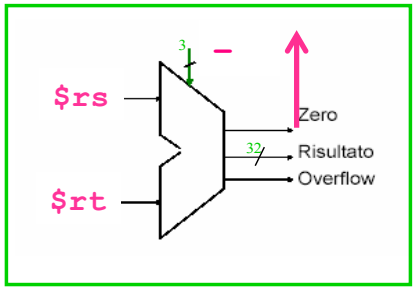


Operazioni:

1) Estensione dell'offset su 32 bit.
2) Moltiplicazione per 4 dell'offset.
3) Somma del PC con l'estensione del segno.
4) Controllo se il contenuto dei registri è uguale.



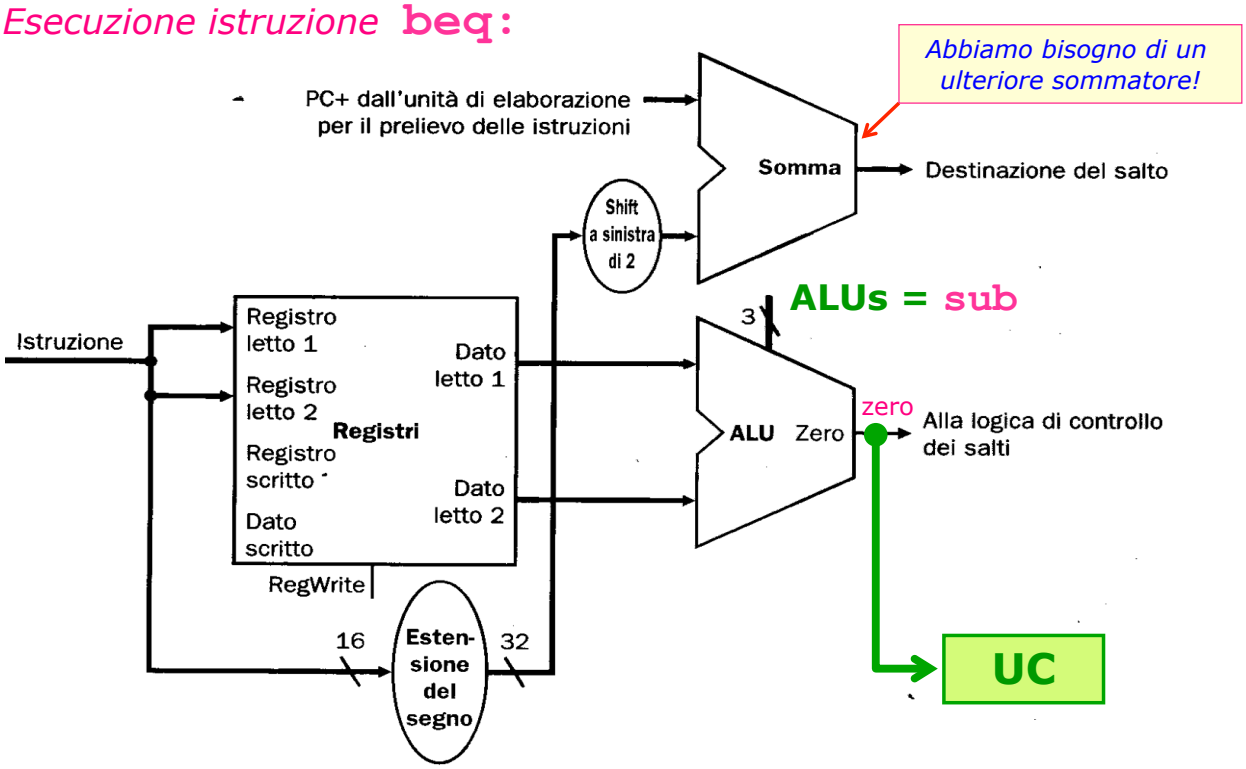




→ **ci servono 2 ALU !!!**

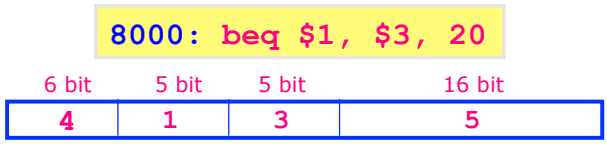
Circuito di esecuzione: beq

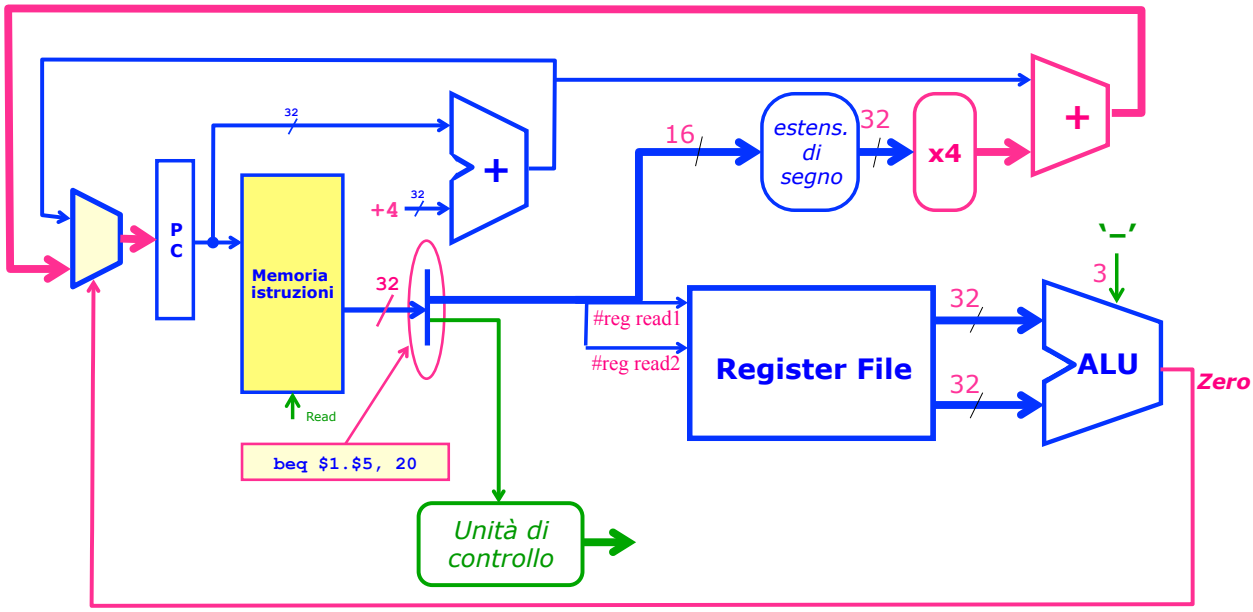
Esecuzione istruzione beq:



CPU per l'esecuzione di una lw

Esecuzione istruzione beq:





Sommario

- La CPU
- Sintesi di una CPU: fase di fetch
- Sintesi CPU per le istruzioni A/L di tipo R
- Sintesi CPU per le istruzioni di tipo I – trasf. memoria
- Sintesi CPU per le istruzioni di tipo I – salti
- Sintesi della CPU complessiva

ISA MIPS – tipi di istruzione: R, I (`lw/sw,branch`)

- I tre formati delle istruzioni sono:

31-26	25-21	20-16	15-11	10-6	5-0
op	rs	rt	rd	shamt	funct

31-26	25-21	20-16	15-0
35/43	rs	rt	offset

31-26	25-21	20-16	15-0
4	rs	rt	indirizzo

- Campo op sempre contenuto nei primi 6 bit (in seguito, Op[5-0])
- Registri da leggere sempre rs e rt, di posizione 25-21 e 20-16
- Registro base per lettura/scrittura sempre rs, di posizione 25-21
- Offset sempre in posizione 15-0
- Registro destinazione rd (15-11) per istruzioni di tipo R, rt (20-16) per istruzioni lettura (necessario multiplexer)

CPU singolo ciclo: caratteristiche

- **CPU a singolo ciclo:** il ciclo di esecuzione di un'istruzione si compie in un unico ciclo di clock.

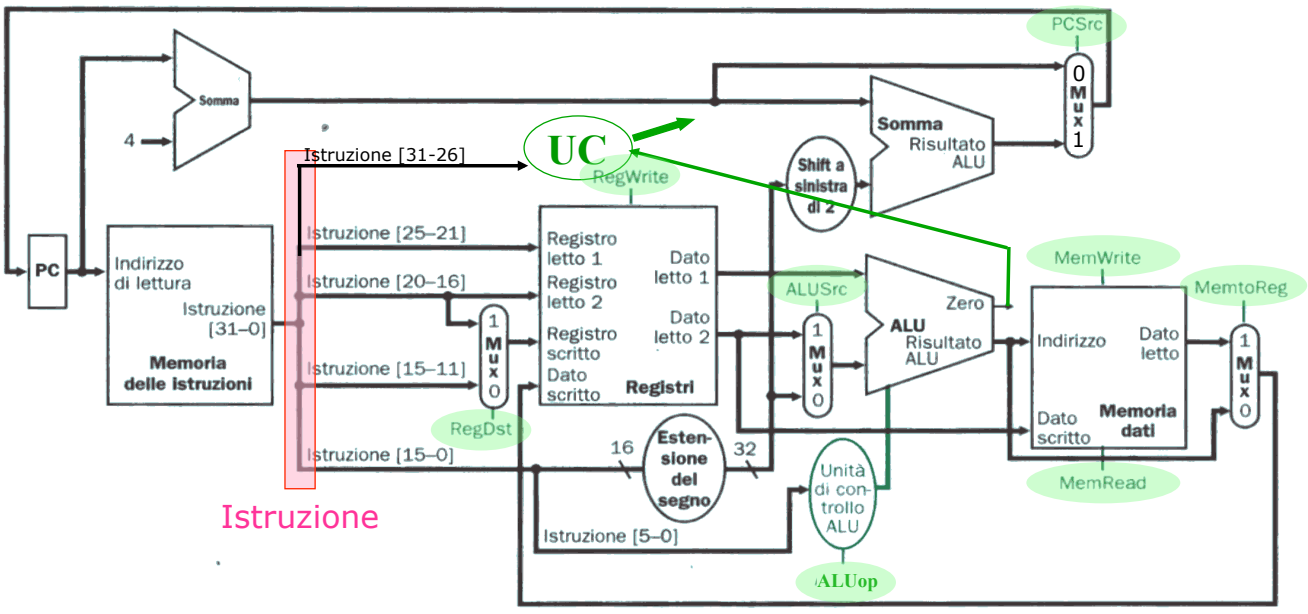
- tutte le unità funzionali coinvolte in un'istruzione vengono utilizzate contemporaneamente

- Ogni unità funzionale può essere utilizzata 1 sola volta durante un'istruzione.

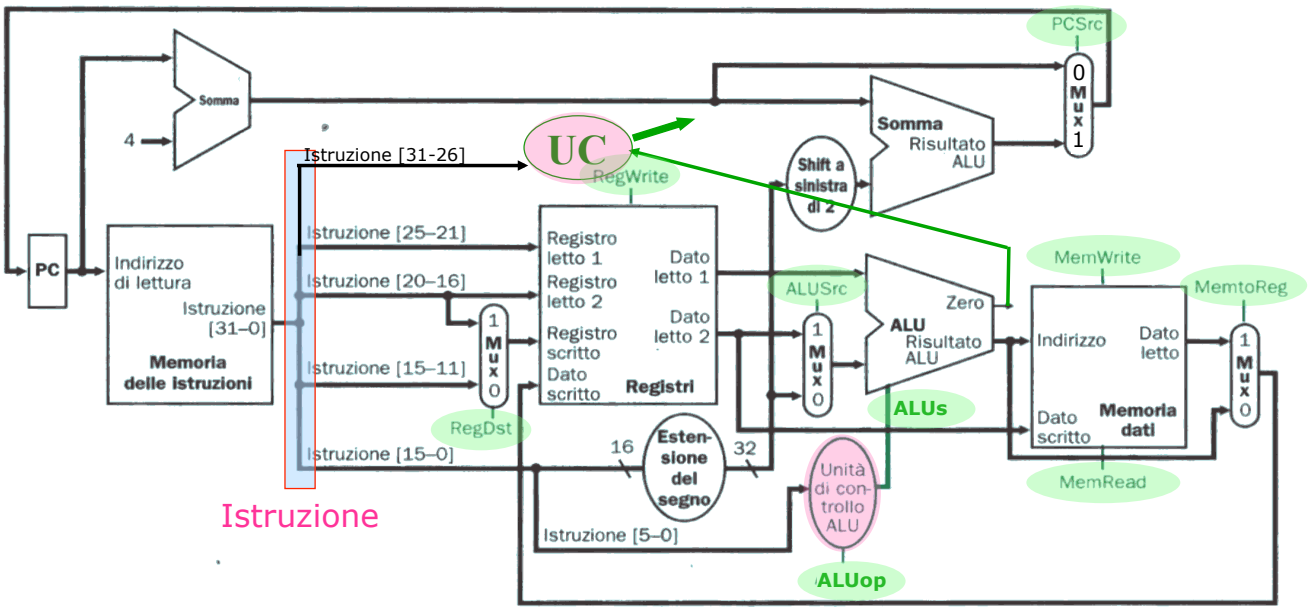
- Duplicazione Memoria: Memoria dati e memoria istruzioni.
- Triplicazione ALU: 2 sommatori + 1 general purpose.
- Introduzione di multiplexers.

Schema generale

- Esegue istruzioni di: accesso a memoria: ***lw/sw,***
  aritmetico-logiche ***tipo R, tipo I***
  branch: ***beq***



Schema generale (`lw/sw + R + beq`)



Unità di Controllo CPU: $Segnali\_Controllo = f(\texttt{Opcode})$

Controllore della ALU: $ALUs = f(\texttt{Opcode}, \texttt{funct})$

Sommario

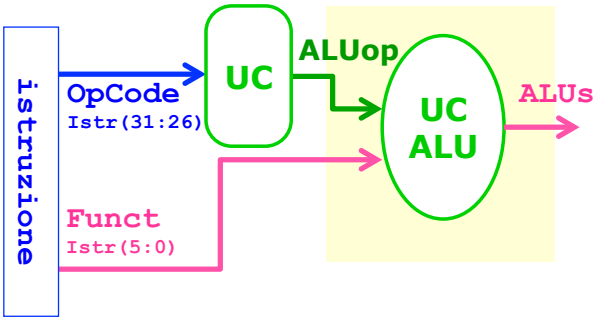
Unità di controllo:

- Controllore della ALU
- Unità di Controllo Principale

UC e ALU

Sintesi UC-ALU:

- **UC** genera **ALUop** in funzione dell'istruzione (**OpCode**)
- **UC-ALU** genera **ALUop** in funzione di:
  - **OpCode**
  - **Funct**



Quali operazioni devono essere eseguite?

Istruz.		OpCode	Operaz. ALU	ALUop
R	→	000000	dipende da funct	10
lw	→	100100	somma	00
sw	→	100110	somma	00
beq	→	000100	confronto → differenza	01

➔ 3 classi distinte (2 bit → ALUop)

Istr	ALUop
R	10
lw/sw	00
beq	01

da implementare nella UC!

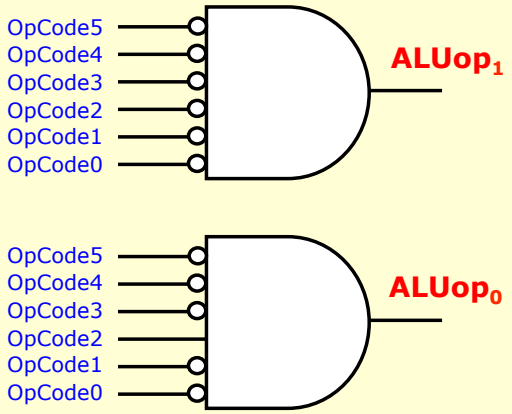
Controllo della ALU: sintesi segnale ALUop

Sintesi del circuito logico:

ALUop = *f* (**OpCode**)

Istr		OpCode						ALUop	
		5	4	3	2	1	0	1	0
lw		1	0	0	1	0	0	0	0
sw		1	0	1	1	0	0	0	0
beq		0	0	0	1	0	0	0	1
R	add	0	0	0	0	0	0	1	0
	sub	0	0	0	0	0	0		
	and	0	0	0	0	0	0		
	or	0	0	0	0	0	0		
	slt	0	0	0	0	0	0		

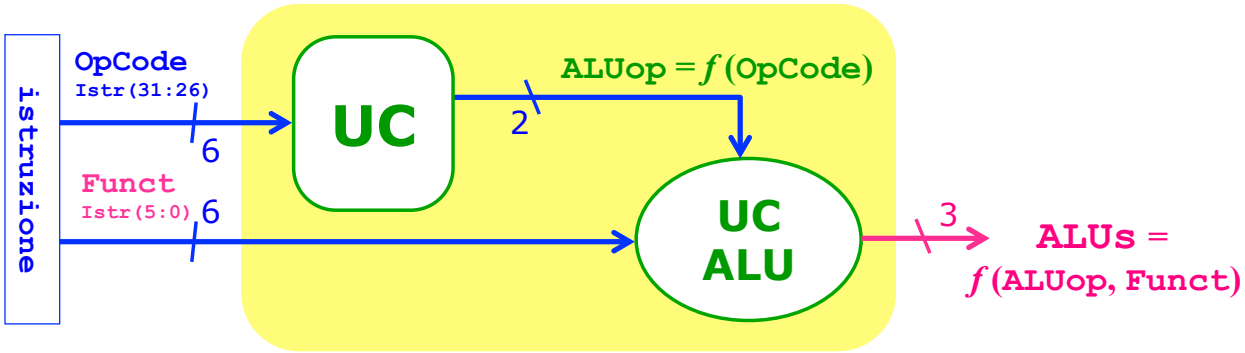
nella UC principale...



$$ALUop_1 = \overline{OpCode5} \cdot \overline{OpCode4} \cdot \overline{OpCode3} \cdot \overline{OpCode2} \cdot \overline{OpCode1} \cdot \overline{OpCode0}$$

$$ALUop_0 = \overline{OpCode5} \cdot \overline{OpCode4} \cdot \overline{OpCode3} \cdot OpCode2 \cdot \overline{OpCode1} \cdot \overline{OpCode0}$$

UC–ALU: Controllo gerarchico



3 casi distinti:

1. R
2. lw/sw
3. branch

OpCode	ALUop	ALUs
R	10	
lw/sw	00	+: 010
beq	01	–: 110

funct	ALUs
and	000
or	001
add	010
sub	110
slt	111

tipo R → **ALUs** dipende da funct

Controllo della ALU

ALUs = f (**ALUop**, **Funct**)

Istr	OpCode						ALUop (*Ao*)		Funct (f_5 – f_0)						ALUs		
	5	4	3	2	1	0	1	0	5	4	3	2	1	0	0	1	2
lw	1	0	0	1	0	0	0	0	x	x	x	x	x	x	0	1	0
sw	1	0	1	1	0	0	0	0	x	x	x	x	x	x	0	1	0
beq	0	0	0	1	0	0	0	1	x	x	x	x	x	x	1	1	0
add	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0
sub	0	0	0	0	0	0	1	0	1	0	0	0	1	0	1	1	0
and	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0
or	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	1
slt	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1

$$ALUs_0 = \overline{Ao_1} Ao_0 + Ao_1 \overline{Ao_0} \left(f_5 \overline{f_4} \overline{f_2} f_1 \overline{f_0} \right) \qquad SOP$$

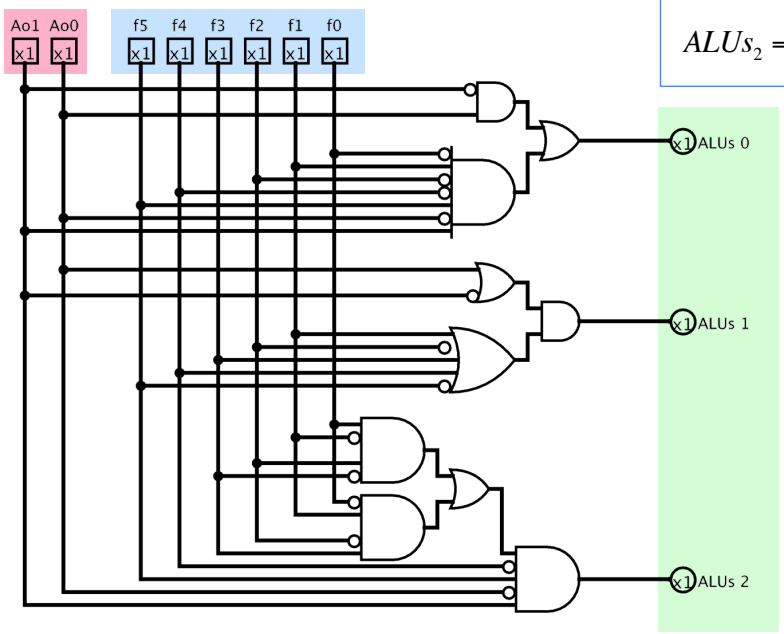
$$ALUs_1 = \left(\overline{Ao_1} + Ao_0 \right) \left(\overline{f_5} + f_4 + f_3 + \overline{f_2} + f_1 \right) \qquad POS$$

$$ALUs_2 = Ao_1 \overline{Ao_0} f_5 \overline{f_4} \left(\overline{f_3} f_2 \overline{f_1} f_0 + f_3 \overline{f_2} f_1 \overline{f_0} \right) \qquad SOP$$

Controllo della ALU

ALUs = f (**ALUop**, **Funct**)

$$ALUs_0 = \overline{Ao_1}Ao_0 + Ao_1\overline{Ao_0}\left(f_5\overline{f_4}\overline{f_2}f_1\overline{f_0}\right)$$
$$ALUs_1 = \left(\overline{Ao_1} + Ao_0\right)\left(\overline{f_5} + f_4 + f_3 + \overline{f_2} + f_1\right)$$
$$ALUs_2 = Ao_1\overline{Ao_0}f_5\overline{f_4}\left(\overline{f_3}f_2\overline{f_1}f_0 + f_3\overline{f_2}f_1\overline{f_0}\right)$$

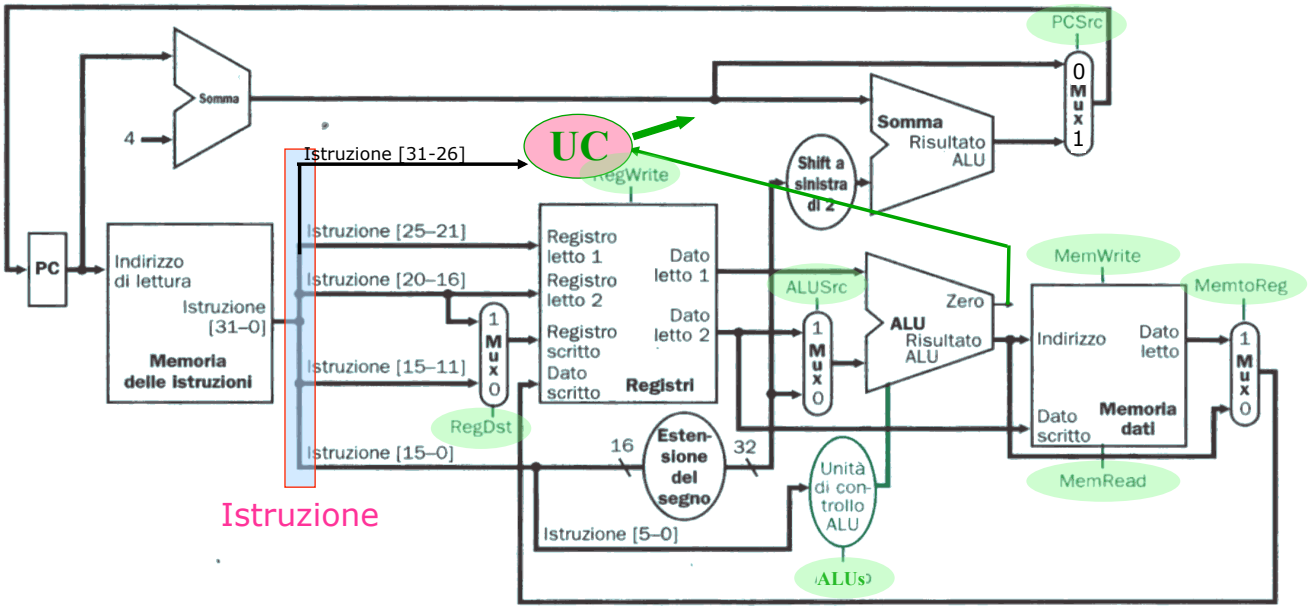


Sommario

Unità di controllo:

- Controllore della ALU
- Unità di Controllo Principale

Schema generale (`lw/sw` + R + `beq`)



Unità di Controllo CPU: $\textbf{Segnali\_Controllo} = f(\texttt{Opcode})$

Controllore della ALU: $\textbf{ALUs} = f(\texttt{Opcode}, \texttt{funct})$

Segnali di controllo CPU

Sintesi UC principale:

OpCode → ***Unità di controllo*** → Segnali di controllo

Segnali di **selezione**:

Segnale	Effetto quando vale **0**	Effetto quando vale **1**
RegDst	Il numero del registro destinazione proviene dal campo **`rd`** (bit **15-11**)	Il numero del registro destinazione proviene dal campo **`rt`** (bit **20-16**)
ALUSrc	Il secondo operando della ALU proviene dal campo **offset** (estesa a 32 bit)	Il secondo operando della ALU proviene dalla **II porta di lettura** del Register File
PCSrc	Il valore del **PC** viene sostituito dall'uscita del sommatore che calcola **PC+4**	Il valore del **PC** viene sostituito dall'uscita del sommatore che calcola l'**indirizzo di salto**
MemtoReg	Il valore inviato a ContenutoWrite del Register File proviene dalla **ALU**	Il valore inviato a ContenutoWrite del Register File proviene dalla **memoria**

Segnali di **comando**:

Segnale	Effetto quando vale **0**	Effetto quando vale **1**
RegWrite	Nessuno	Nel registro specificato su #RegWrite viene scritto il valore all'ingresso: ContenutoWrite
MemRead	Nessuno	Comando di **lettura** della memoria dati
MemWrite	Nessuno	Comando di **scrittura** della memoria dati

Segnali di controllo del Data-path

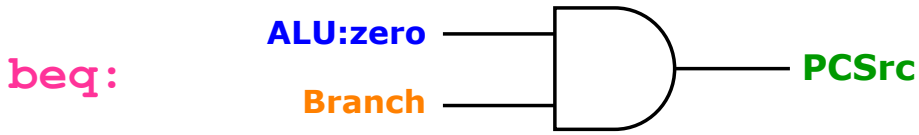
Sintesi segnali di controllo – Tabella di verità

Istr	OpCode	Reg Dst	ALU src	Mem toReg	Reg Write	Mem Read	Mem Write	ALU op	Branch
R	000000	0	1	0	1	0	0	10	0
lw	100011	1	0	1	1	1	0	00	0
sw	101011	x	0	x	0	0	1	00	0
beq	000100	x	1	x	0	0	0	01	1

Gestione del branch: (beq)

- Relazione tra **PCSrc** e **Branch**:

PCSrc = Branch **AND** *[condiz. salto verificata]*



UC principale: Segnali di controllo

Tabella di sintesi UC principale:

Ingressi: **OpCode, ALU.zero**

Uscite: segnali di controllo CPU

Ingressi			Uscite							
Istr	OpCode	ALU: Zero	Reg Dst	ALU src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	ALU op
R	000000	x	0	1	0	1	0	0	0	10
lw	100100	x	1	0	1	1	1	0	0	00
sw	101100	x	x	0	x	0	0	1	0	00
beq	000100	1	x	1	x	0	0	0	1	01

RegDst = OpCode2

MemtoReg = OpCode2

MemRead = OpCode5 · not(OpCode3)

PCsrc = **Branch** · **ALUzero**

RegWrite = not(OpCode2) + OpCode5 · not(OpCode3)

ALUsrc = not(OpCode5)

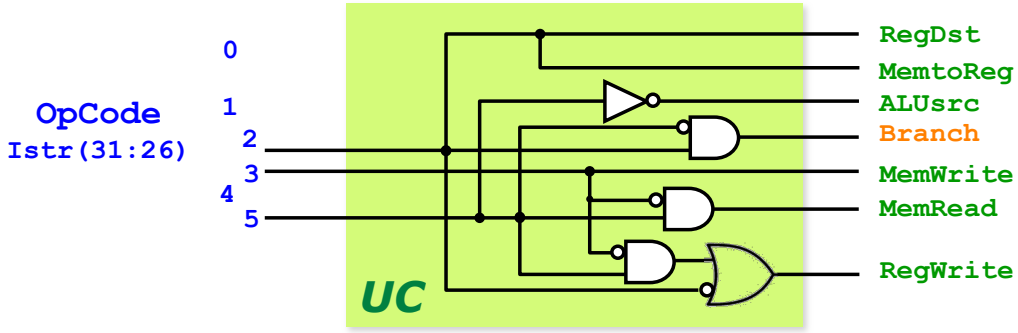
MemWrite = OpCode3

Branch = OpCode2 · not(OpCode5)

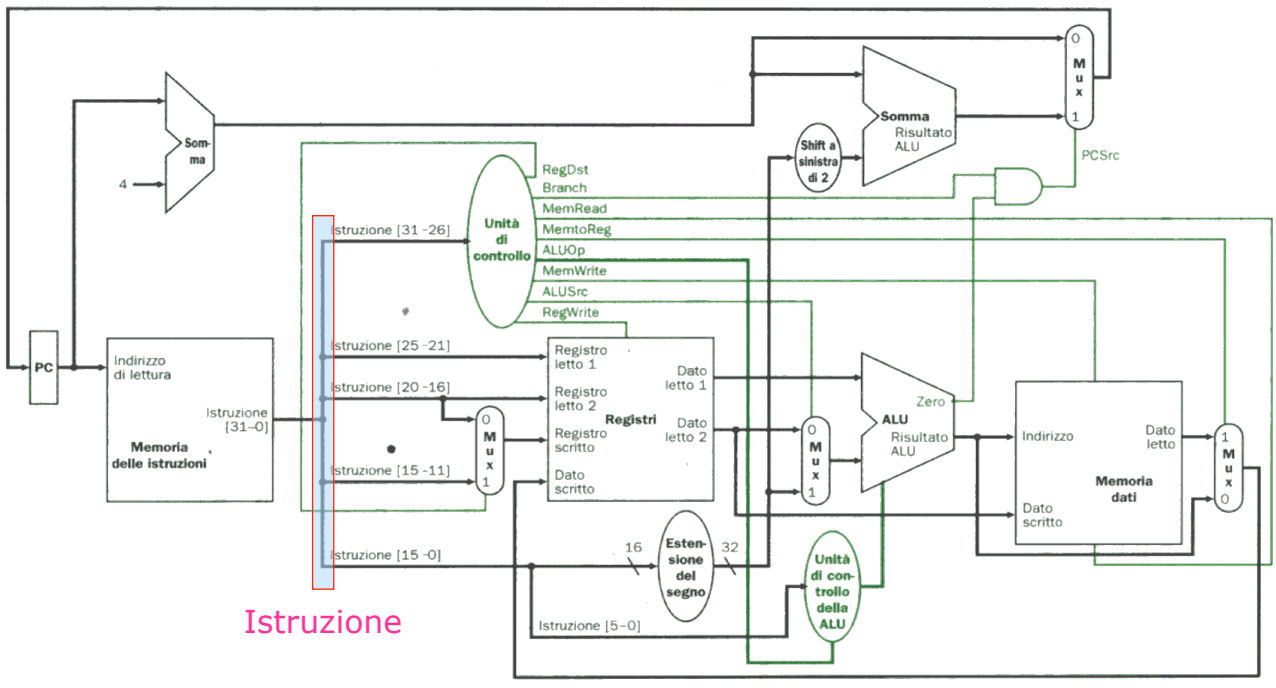
Sintesi di alcuni segnali di controllo

- **Sul nostro ISA ridotto:**

```
RegDst = OpCode2                    ALUsrc = not(OpCode5)
MemtoReg = OpCode2                  MemWrite = OpCode3
MemRead = OpCode5 ·not(OpCode3)     Branch = OpCode2 ·not(OpCode5)
PCsrc = Branch ·ALUzero
RegWrite = not(OpCode2) + OpCode5 ·not(OpCode3)
```



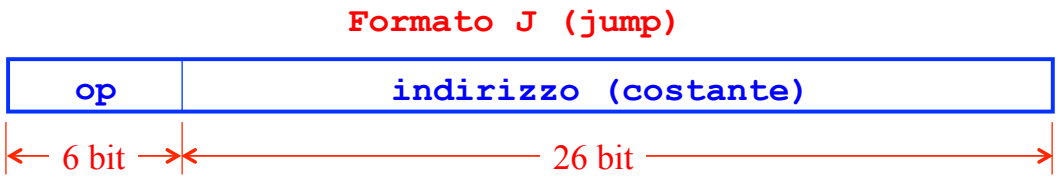
CPU Singolo Ciclo + UC



Istruzione

© Patterson, Hennessy

L'istruzione **jump** *(formato J)*



- Indirizzo di salto costruito in due passi:
  - Costruzione parte bassa indirizzo (28 bit): **addr** = indirizzo*4 = [ind.|00]
  - Determinazione dell'indirizzo di salto: PC ← **[PC(31:28) | addr]**

```
j 80040 :
```

Base (PC): 0100 1000 0011 0001 1011 1011 1011 10 11
Campo indirizzo: 1000 0110 0111 0000 0000 0001 00 **00** =

Indirizzo salto: 0100 1000 0110 0111 0000 0000 0001 00 00

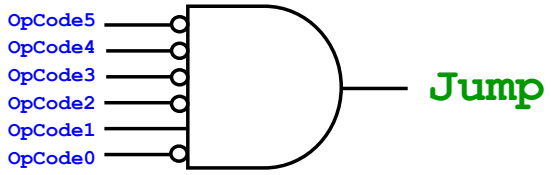
Segnali di controllo

Segnale	Effetto quando è **0**	Effetto quando è **1**
RegDst	Il numero del registro destinazione proviene dal campo **rd** (bit **15-11**)	Il numero del registro destinazione proviene dal campo **rt** (bit **20-16**)
RegWrite	Nessuno	Nel registro specificato su #RegWrite viene scritto il valore all'ingresso: ContenutoWrite
ALUSrc	Il secondo operando della ALU proviene dal campo **offset** (estesa a 32 bit)	Il secondo operando della ALU proviene dalla **II porta di lettura** del Register File
PCSrc	Il valore del PC viene sostituito dall'uscita del sommatore che calcola **PC+4**	Il valore del PC viene sostituito dall'uscita del sommatore che calcola l'**indirizzo di salto**
MemRead	Nessuno	Comando di **lettura** della memoria dati
MemWrite	Nessuno	Comando di **scrittura** della memoria dati
MemtoReg	Il valore inviato a ContenutoWrite del Register File proviene dalla **ALU**	Il valore inviato a ContenutoWrite del Register File proviene dalla **memoria**

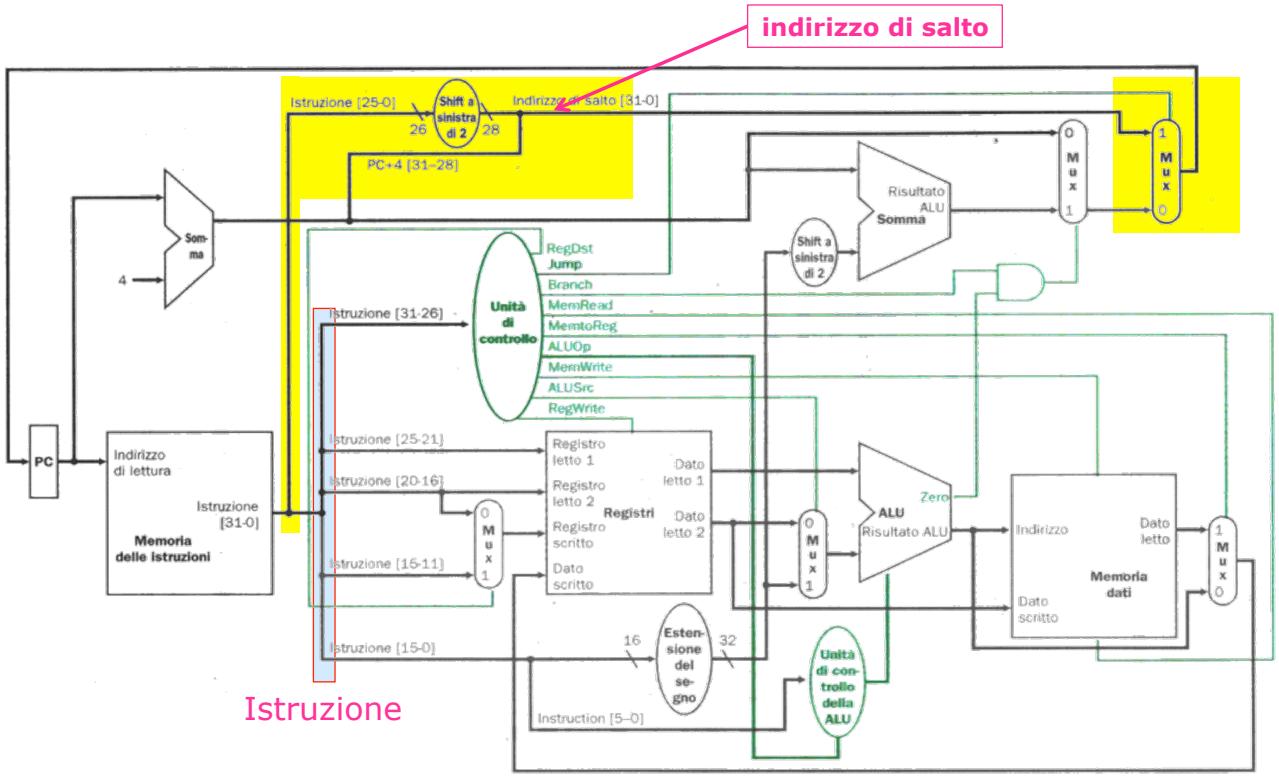
Jump	PC viene impostato a **PC+4** oppure all'indirizzo **destinazione** della **branch**	PC viene impostato al valore ottenuto dal campo **dato** della **jump**

Opcode(**j**): **2** (000010)

$$Jump = \overline{O_5} \cdot \overline{O_4} \cdot \overline{O_3} \cdot \overline{O_2} \cdot O_1 \cdot \overline{O_0}$$



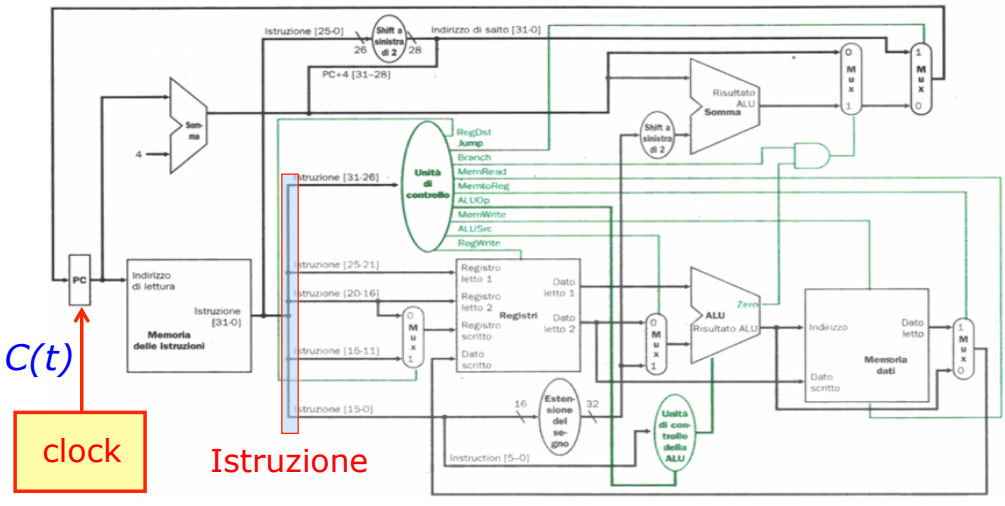
CPU + UC completa (aggiunta di jump)

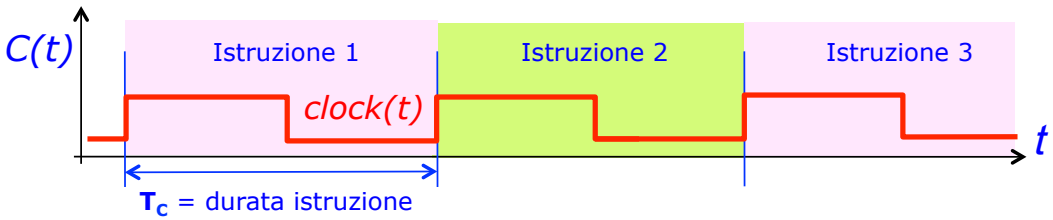


CPU a Ciclo singolo

- CPU a CICLO SINGOLO:

  Ad ogni ciclo di clock viene eseguita un'istruzione completa.

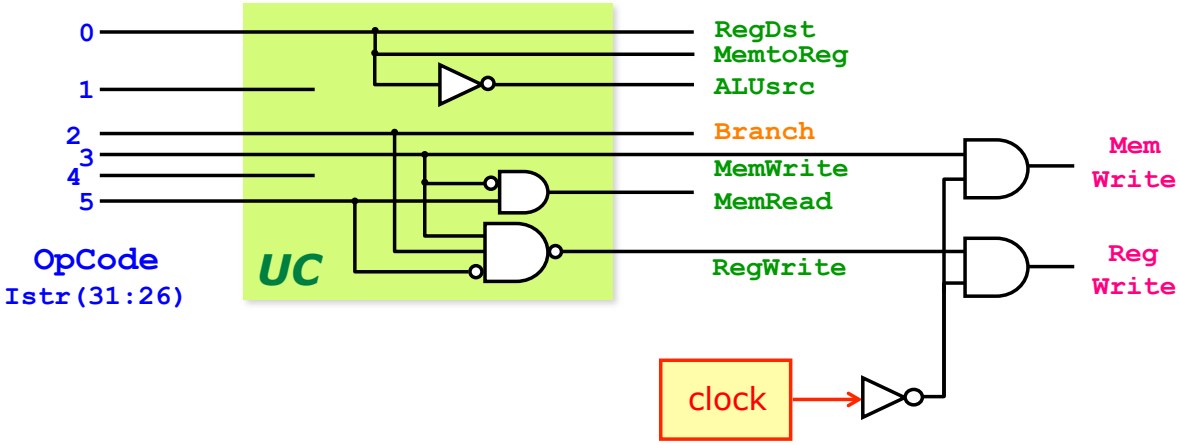




Sintesi di alcuni segnali di controllo

Problemino di sincronizzazione...

- `RegWrite` e `MemWrite` devono agire (memorizzare) con un minimo di **ritardo** rispetto all'inizio dell'istruzione (fase di fetch)
- Li sincronizzo sul fronte di discesa del clock (→ metà periodo)



Segnali di controllo del Data-path

*Esempio di **arricchimento del Set Istruzioni***

- Aggiungiamo l'istruzione: `addi (OpCode: 8)`

Istr	OpCode	Reg Dst	ALU src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	ALU op
R	000000	0	1	0	1	0	0	0	10
addi	001000	1	0	0	1	0	0	0	00
lw	100100	1	0	1	1	1	0	0	00
sw	101100	x	0	x	0	0	1	0	00
beq	000100	x	1	x	0	0	0	1	01

UC principale: Segnali di controllo

Tabella di sintesi UC principale, con aggiunta di: `addi`

Ingressi: **`OpCode, ALU.zero`**

Uscite: segnali di controllo CPU

Ingressi			Uscite							
Istr	OpCode	ALU: Zero	Reg Dst	ALU src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	ALU op
R	000000	x	0	1	0	1	0	0	0	10
addi	001000	x	1	0	0	1	0	0	0	00
lw	100100	x	1	0	1	1	1	0	0	00
sw	101100	x	x	0	x	0	0	1	0	00
beq	000100	1	x	1	x	0	0	0	1	01

I segnali modificati sono sottolineati:

RegDst = OpCode2+OpCode3

ALUsrc = not(OpCode2+OpCode3)

MemtoReg = OpCode3

MemWrite = OpCode2 ·OpCode3

MemRead = OpCode5 ·not(OpCode3)

Branch = OpCode2 ·not(OpCode5)

PCsrc = Branch ·ALUzero

RegWrite = not(OpCode2) + not(OpCode3) ·OpCode5

Esercizi di riepilogo

Si traduca il seguente frammento di codice Assembly MIPS in linguaggio macchina, formato esadecimale, calcolando prima i valori esadecimali ***N1*** e ***N2*** che permettono di saltare esattamente all'indirizzo indicato in ciascun commento.

```
0x12345678:   beq $s0, $s4, N1     # salta a: 0x12345660
              j N2                 # salta a: 0x10203040
```

---

beq: PC contiene: 0x1234567C (PC+4)
si vuole saltare a: 0x12345660

Operazione: **0x1234567C + N1** (in byte) **= 0x12345660 → N1 = –0x1C (-28_{10})**
N1 = –(0x1C) = (in complemento a 2, su 16 bit) = **0xFFE4**

Istruzione: **N1** va espresso **in words (non in bytes)** **→ N1/4 = –710 = 0xFFF9**
| 000100 | 10000 | 10100 | 1111 1111 1111 1001 | → **0x1214FFF9**

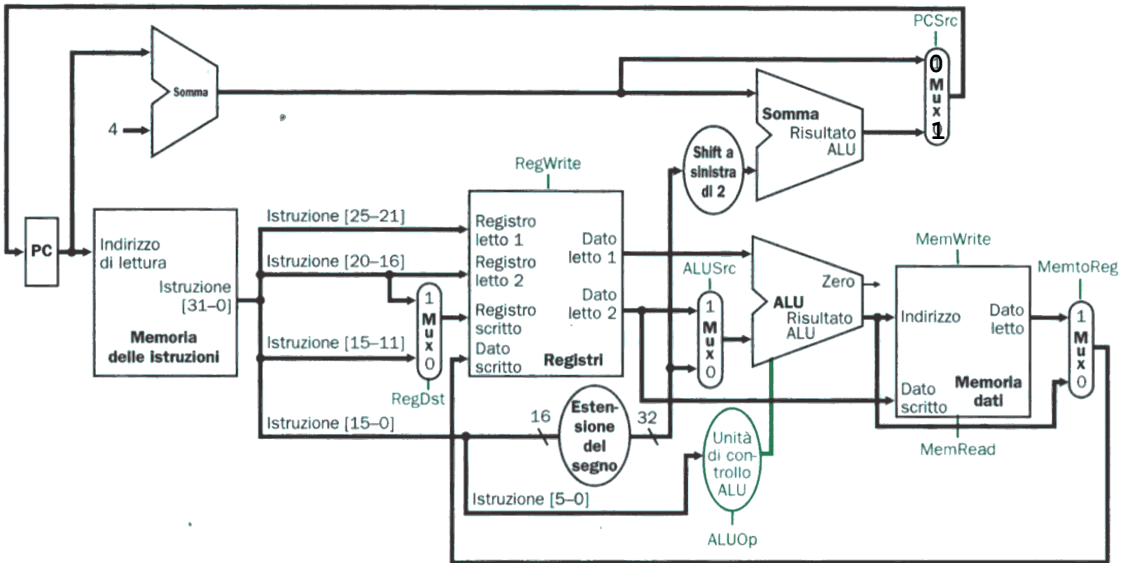
---

j: PC punta a: 0x12345680 (PC+4)
si vuole saltare a: 0x10203040 → PC(31:28)=0x1: non cambia → OK!
N2 = 0x0203040 (su 28 bit)
Istruzione: N2 va espresso in words: | 2 | N2/4 |
→ | 000010 | 00 0000 1000 0000 1100 0001 0000 | (00) → **0x08080C10**

Esercizi di riepilogo

Evidenziare, nel seguente schema di CPU, i valori all'ingresso ed all'uscita di ogni ALU ed i valori di ogni segnale di controllo, supponendo che la CPU esegua l'istruzione:

`0x300: lw $2, 16($1)`



Esercizi di riepilogo

Evidenziare, nel seguente schema di CPU, i valori all'ingresso ed all'uscita di ogni ALU ed i valori di ogni segnale di controllo, supponendo che la CPU esegua l'istruzione:

`0x300: lw $2, 16($1)`