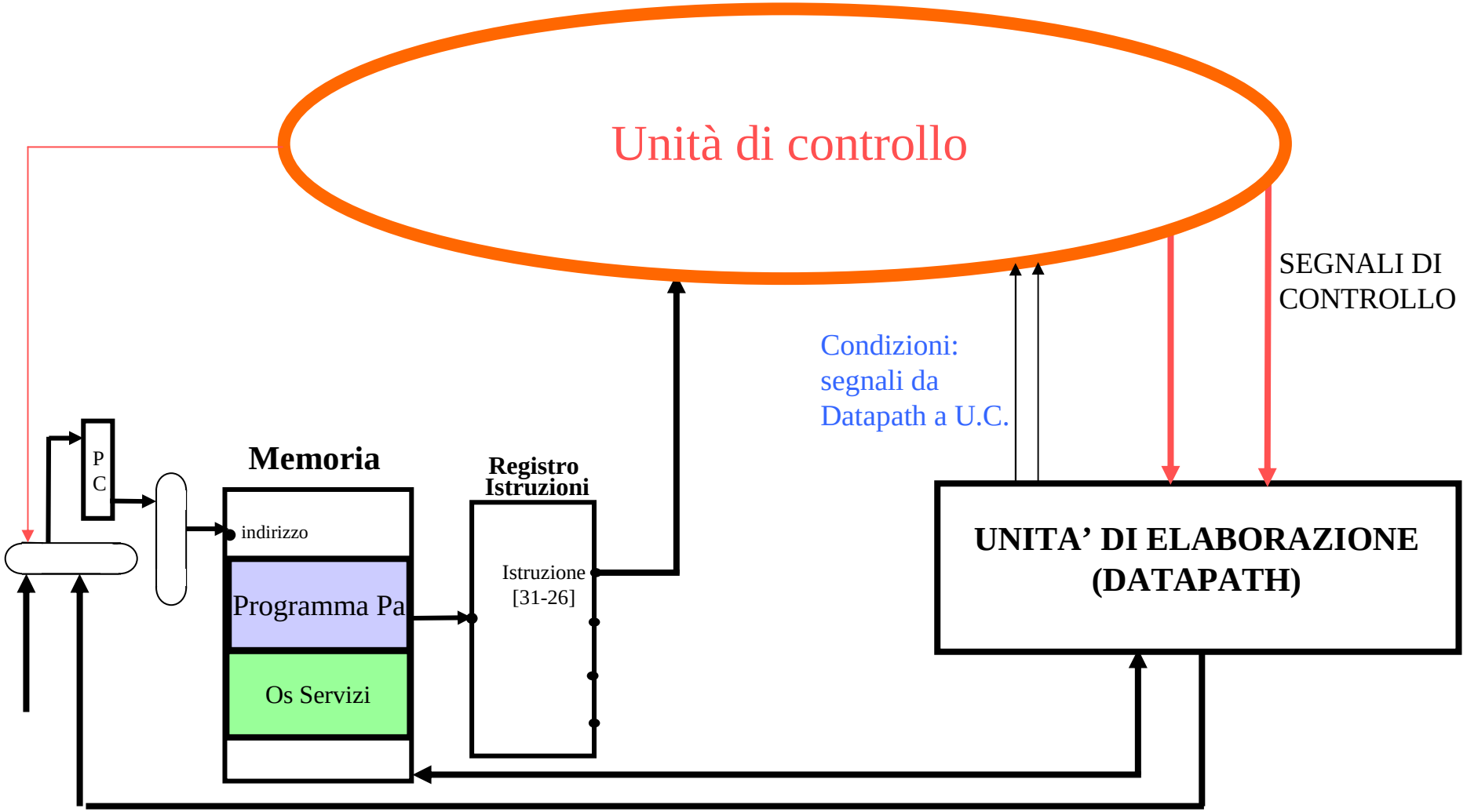
Calcolatori Elettronici A
a.a. 2008/2009

CPU a singolo ciclo

Massimiliano Giacomin

Schema del processore (e memoria)

Durante l'esecuzione di un programma applicativo Pa, i circuiti interpretano le istruzioni del programma in linguaggio macchina costituito dal < Pa (tradotto) ∘ i servizi OS>



Nella progettazione della CPU, faremo riferimento alle seguenti istruzioni:

- Istruzioni aritmetiche: add, sub, and, or, slt

  add rd, rs, rt // rd ← rs + rt

  slt rd, rs, rt // rd = 1 se rs < rt, 0 altrimenti

- Istruzioni di accesso a memoria:

  lw rt, offset(rs) // rt ← M[rs+offset]

  sw rt, offset(rs) // M[rs+offset] ← rt

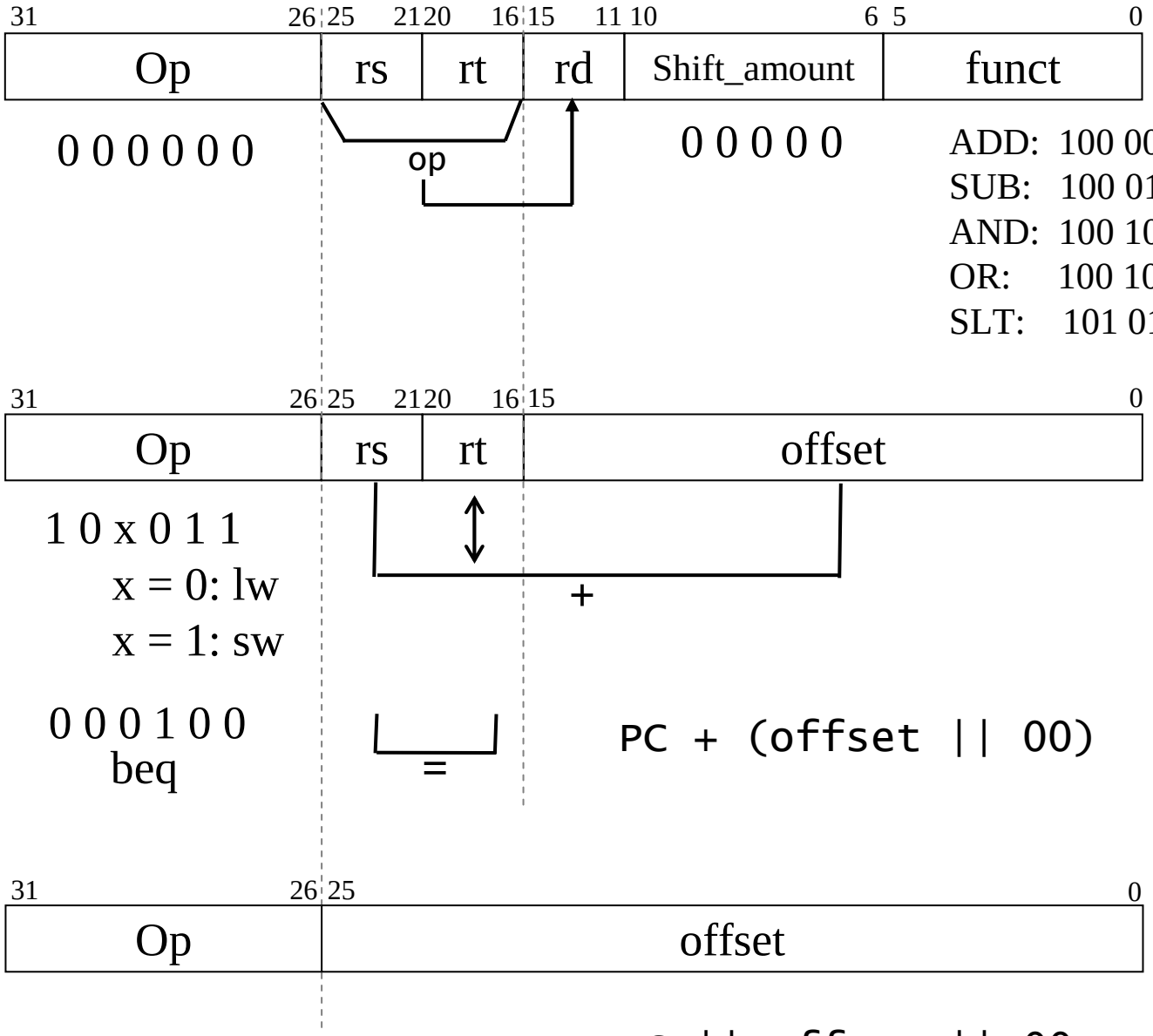
- Istruzioni di salto condizionato:

  beq rs, rt, offset // se rs=rt salta a offset *istruzioni* rispetto a PC
  (aggiornato a istruzione corrente + 4 bytes!)
  in bytes: PC + (offset || 00)

- Salto incondizionato:

  j offset // salta all'indirizzo *in istruzioni* ottenuto da:
  4 bit di PC || offset [30 bit]
  indirizzo in byte è la concatenazione di
  4 bit di PC || offset || 00 [32 bit]

Codifica delle istruzioni viste:



Aritmetiche: Tipo-R

lw, sw, beq: Tipo-I

J: Tipo-J

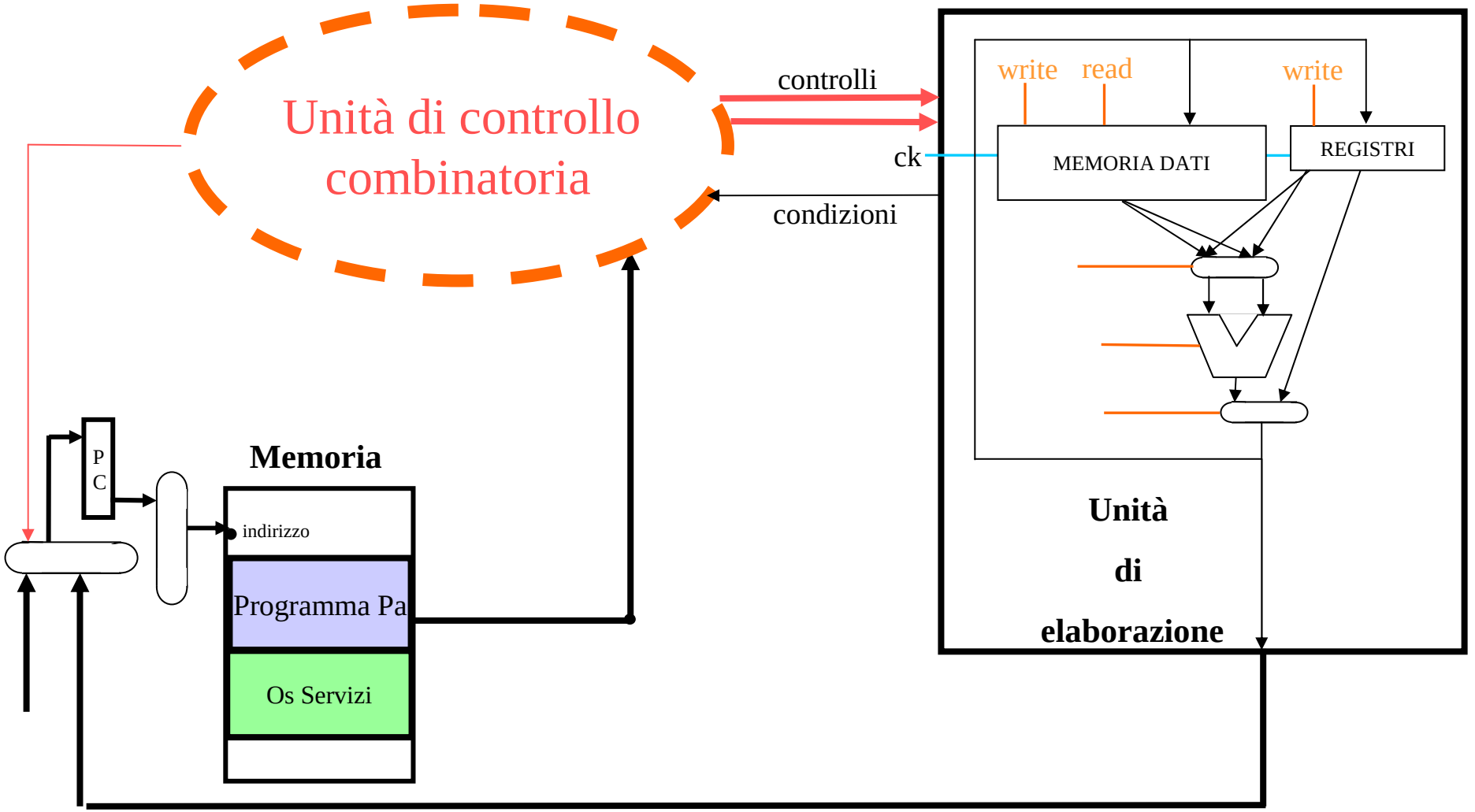
VEDREMO:

- SINGOLO CICLO
- MULTICICLO
    - A STATO ESPLICITO (FSM)
    - A CONTROLLO MICROPROGRAMMATO

IN TUTTI I CASI, PROGETTEREMO:

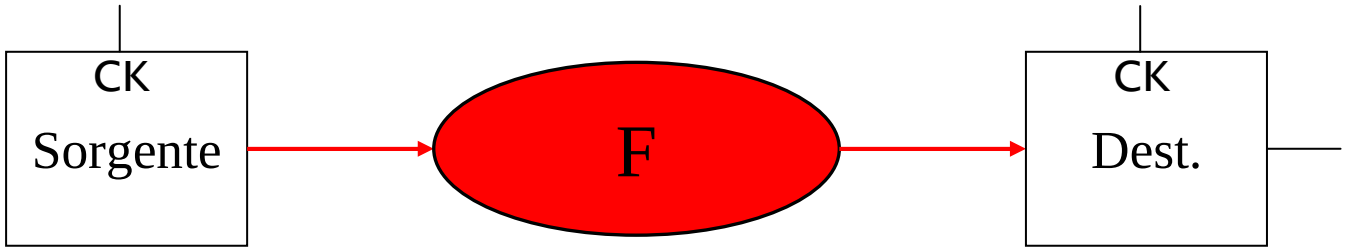
- DATAPATH
- UNITA’ DI CONTROLLO

Controllo di un processore a singolo ciclo: l'idea di base



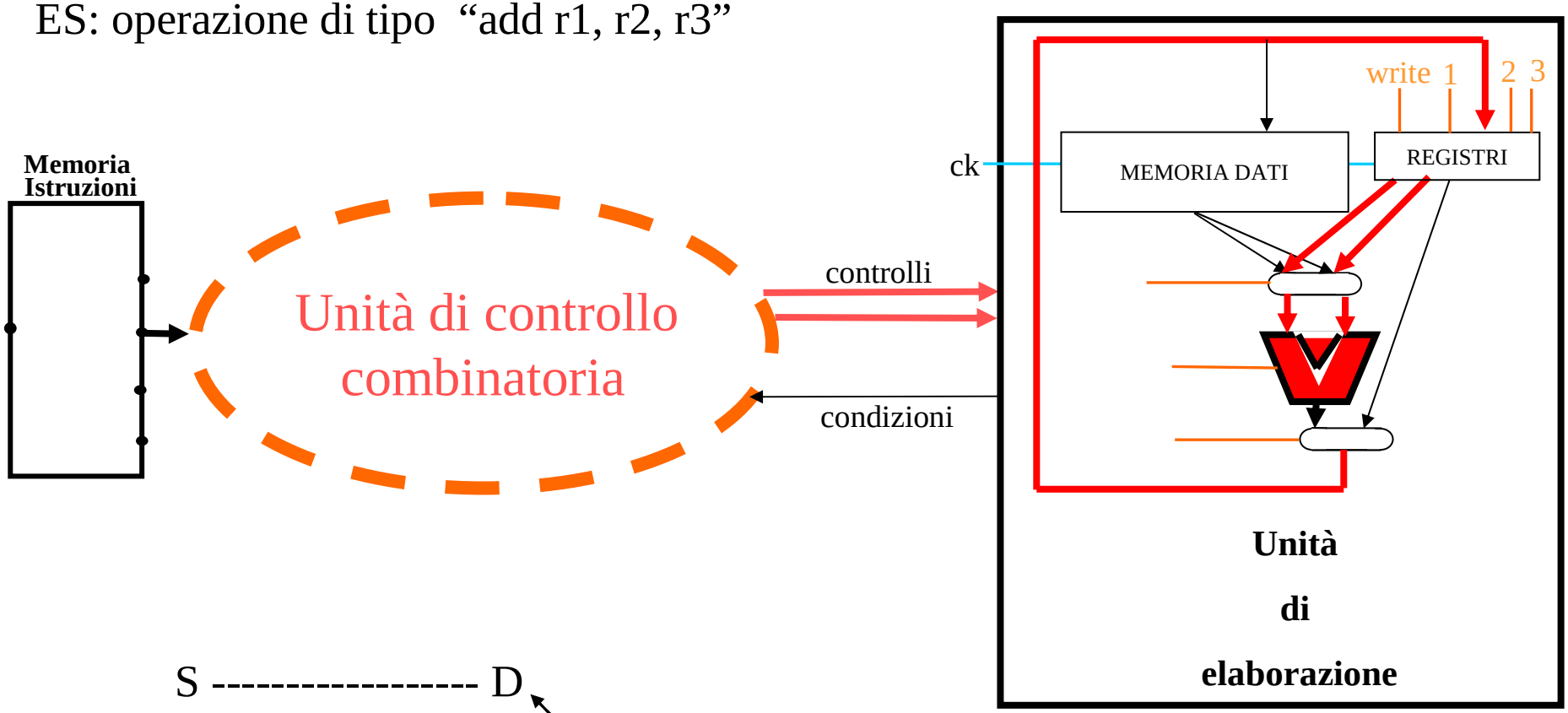
Idea di base

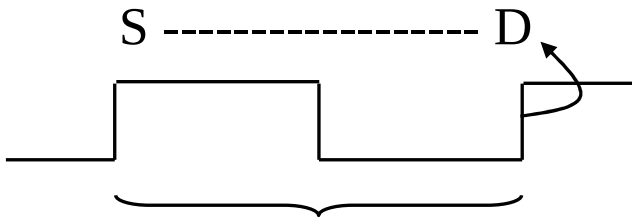
- Ad ogni ciclo di clock, la memoria istruzioni fornisce l'istruzione corrente
- L'unità di controllo è una rete combinatoria che:
  - riceve in input l'istruzione corrente
  - produce in output segnali di controllo all'unità di elaborazione:
    controllo multiplexer, read e write ad elementi di memoria, controllo ALU
- I segnali di controllo determinano, a seconda del tipo di istruzione:
  - il percorso sorgente-destinazione dei dati mediante:
    indirizzi e numeri registri + segnali di controllo ai multiplexer
  - le operazioni aritmetiche e logiche effettivamente svolte mediante:
    segnali di controllo alle ALU
  - se un elemento di memoria deve scrivere e/o leggere un dato mediante:
    segnali di tipo read/write
- Avremo quindi la determinazione di un "percorso" del tipo:



dove: - sorgente e destinazione possono coincidere
- valore sorgente disponibile "nel corso" del ciclo, destinazione scritta alla fine

ES: operazione di tipo “add r1, r2, r3”



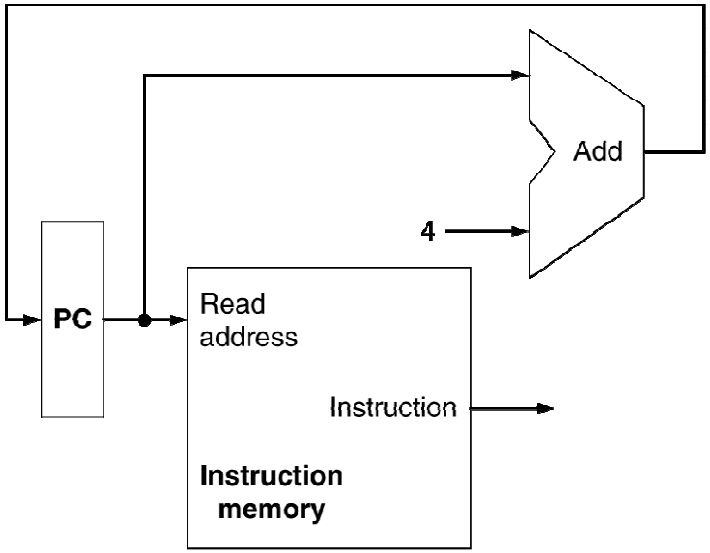


- valori dei registri disponibili (poco dopo – T_{prop} – l’inizio del periodo)
  + si “crea” percorso sorgente-destinazione (con i comandi relativi)
- valori all’ingresso della destinazione (registri) stabili entro la fine del ciclo
- questi valori sono scritti nel fronte successivo – disponibili prossimo ciclo

PROGETTAZIONE DATAPATH

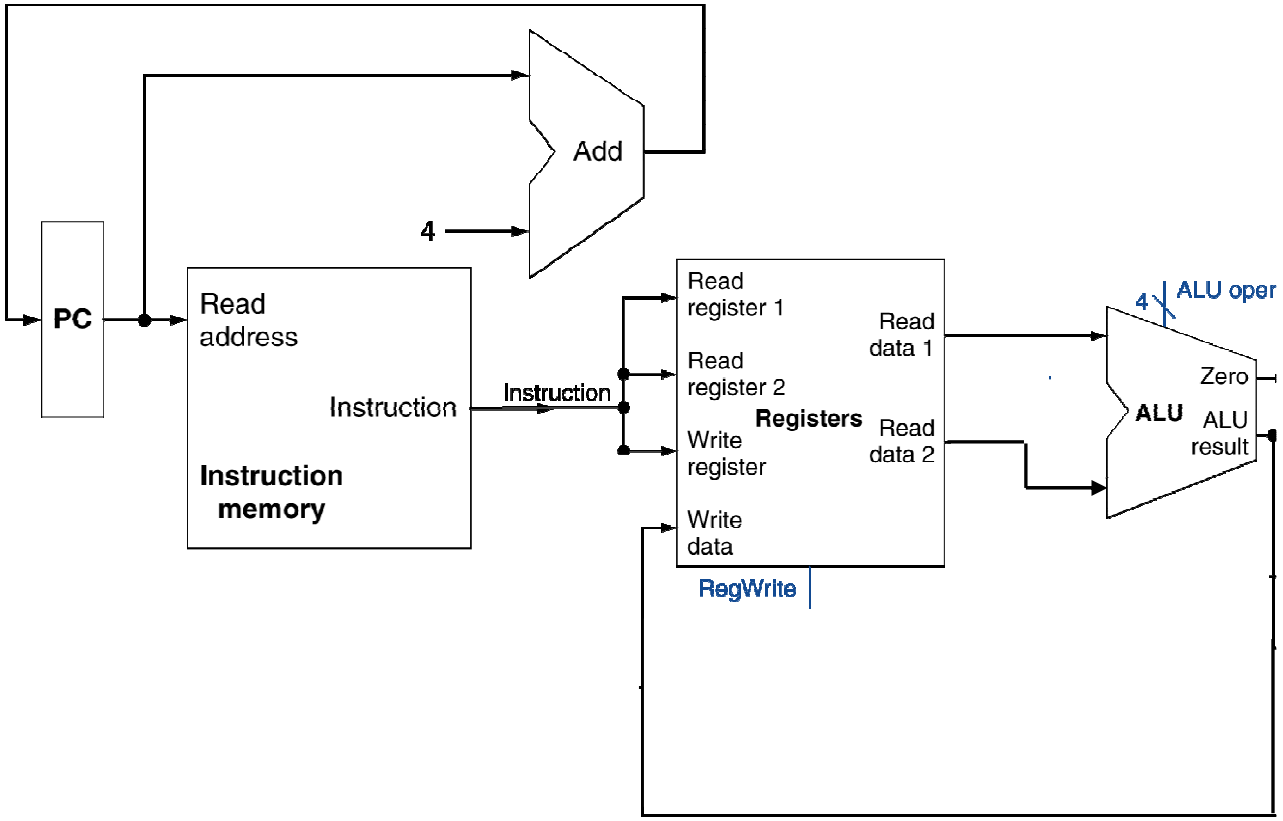
(CASO MIPS)

Consideriamo inizialmente solo la *add*



FETCH

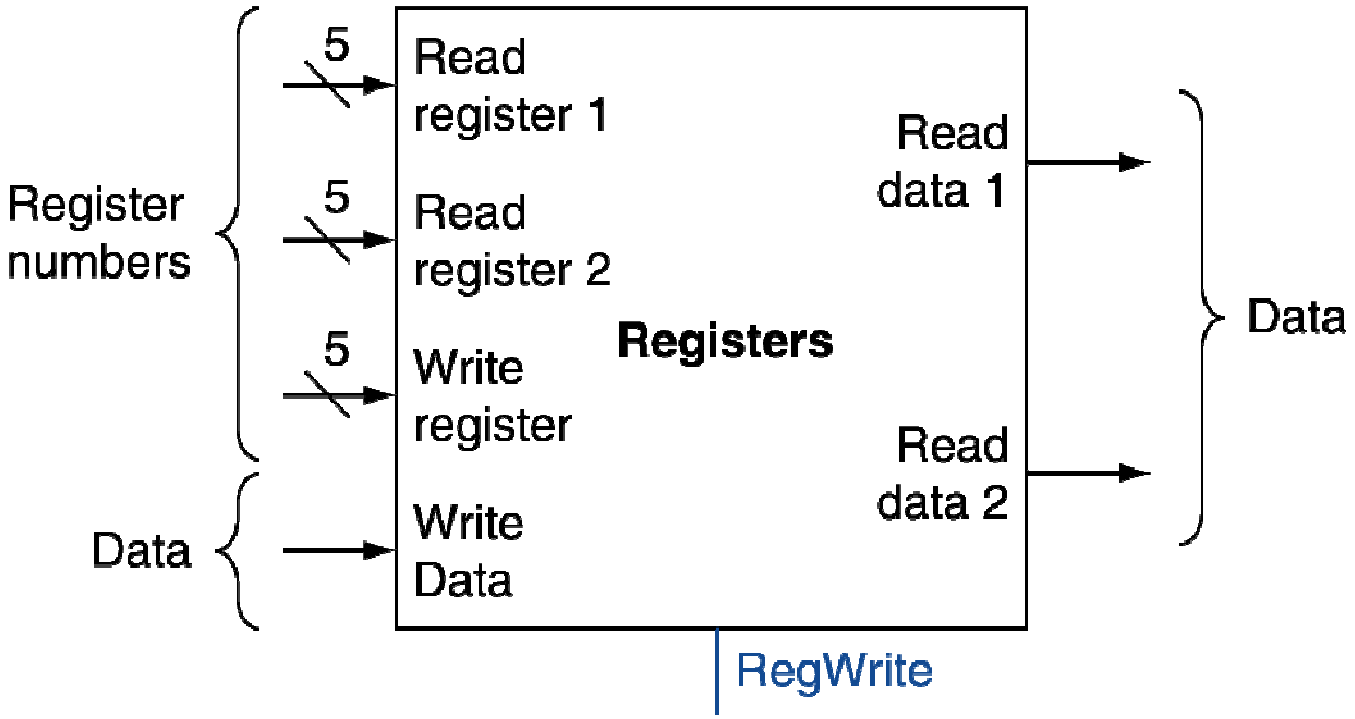
Consideriamo inizialmente solo la *add*



READ EX

WRITE

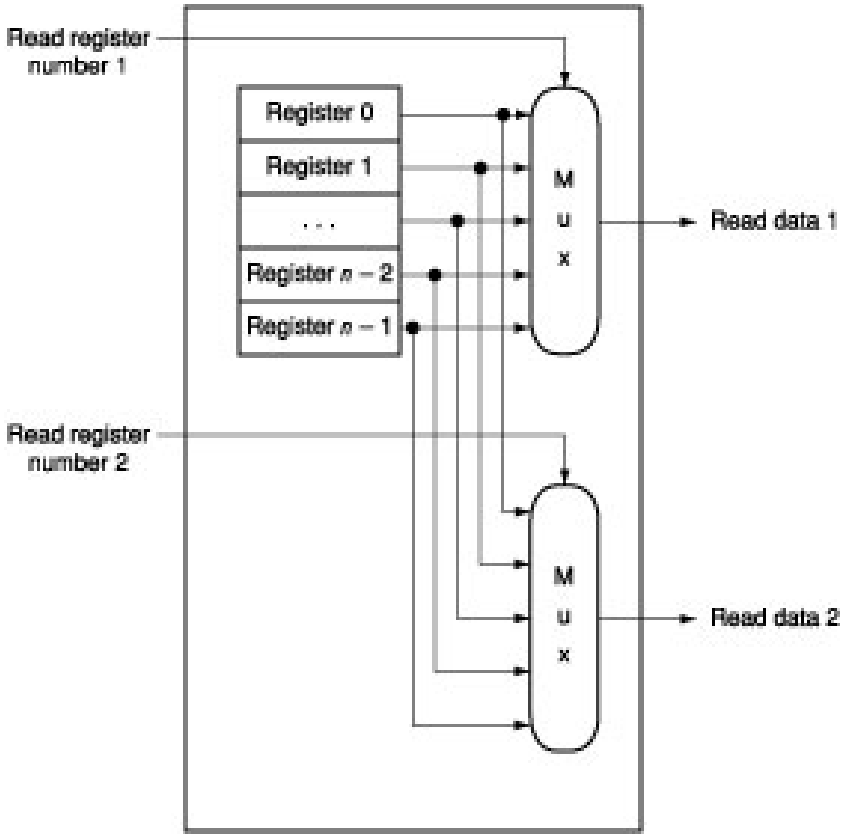
Un nuovo componente: il register file



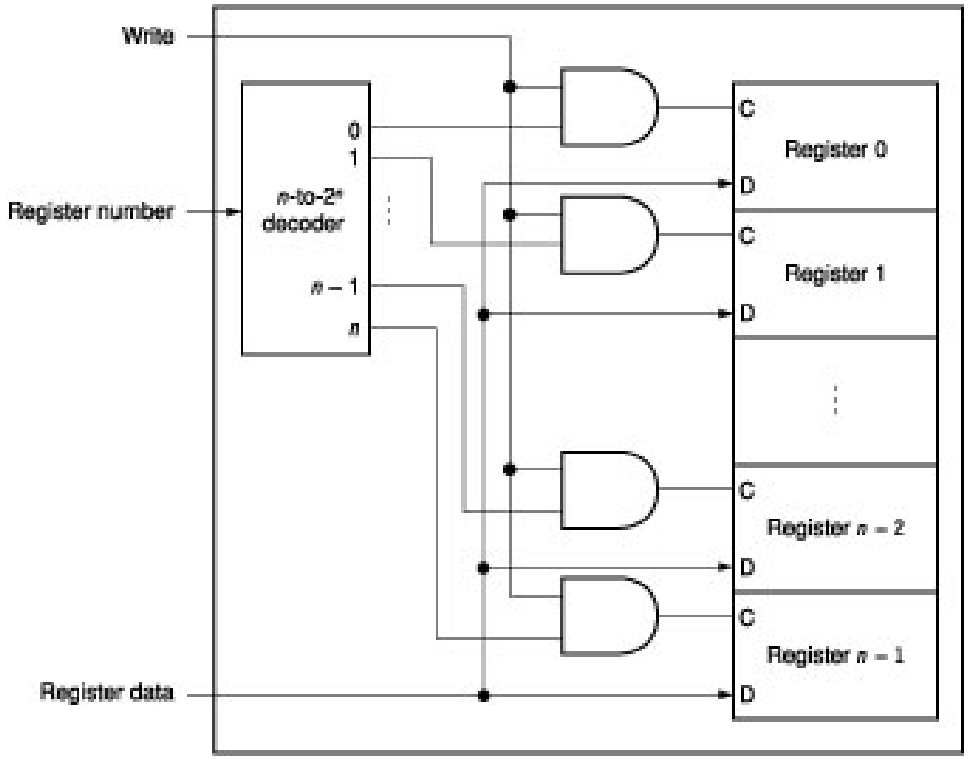
Note:

- lettura asincrona, no segnale di read
- scrittura sincrona (edge-triggered), abilitata dal segnale RegWrite

Realizzazione



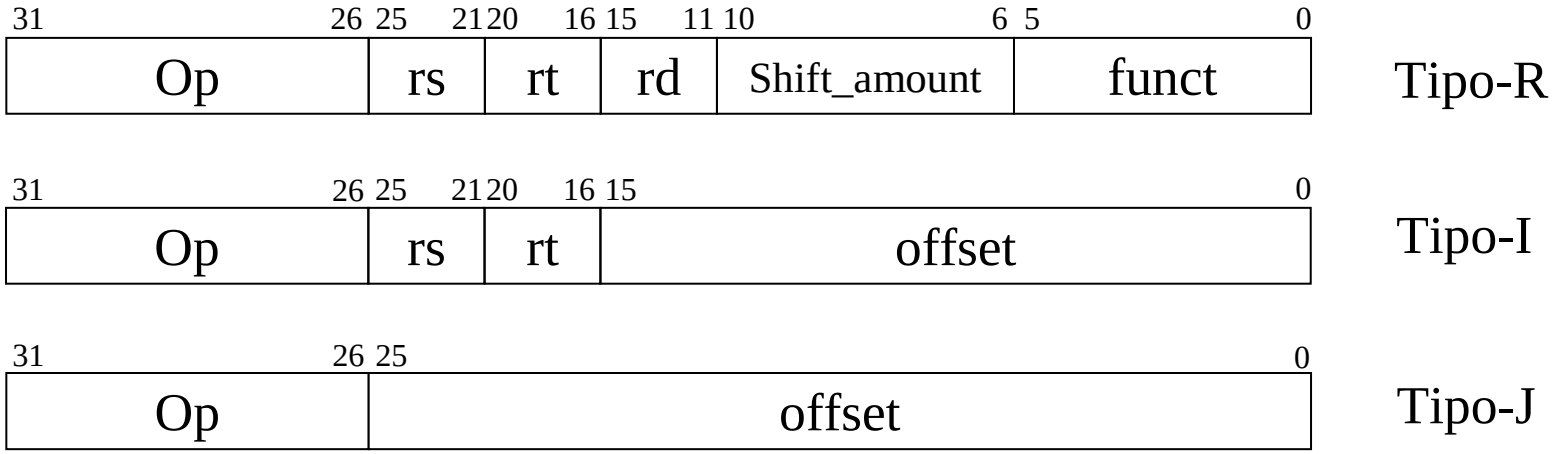
Porte di lettura



Porta di scrittura (clock, non indicato, va in AND)

Integrazione delle altre istruzioni

- Aritmetiche e logiche } Tipo-R
- lw, sw } Tipo-I
- beq
- j } Tipo-J



La progettazione è facilitata dal fatto che:

- Campo Op sempre in [31-26]
- i registri da leggere sono sempre in rs e rt
- il registro base [da sommare] per lw e sw è rs [primo ingresso della ALU]
- l'offset a 16 bit da sommare per beq, lw, sw è in [15-0]

Il registro su cui scrivere può essere rd (per operaz. di TIPO-R) o rt (per lw)

Le fasi (informalmente)

- Fetch (uguale per tutte le istruzioni)
- Lettura registri:
  - *Tipo-R*, *beq*, *sw*: *rs* e *rt*
  - *lw*: *rs*
- Uso della ALU:
  - *Tipo-R* (per eseguire l'operazione): *rs* e *rt*
  - accesso alla memoria (per calcolare l'indirizzo): *rs* e *offset*
  - *beq* (per confrontare i registri): *rs* e *rt*
- Uso della memoria
  - *lw*: in lettura (per prelevare *rt*)
  - *sw*: in scrittura (per memorizzare *rt*)
- Scrittura registri
  - *Tipo-R*: *rd*
  - *lw*: *rt*

Condivisione degli elementi funzionali

• Ogni elemento usato più di una volta nell'esecuzione di una istruzione duplicato:

- Memoria dati (lw, sw)

≠

Memoria programmi (fase di fetch di tutte le istruzioni)

- ALU (lw, sw, beq, aritmetiche)

≠

Sommatore PC + 4 (tutte le istruzioni)

≠

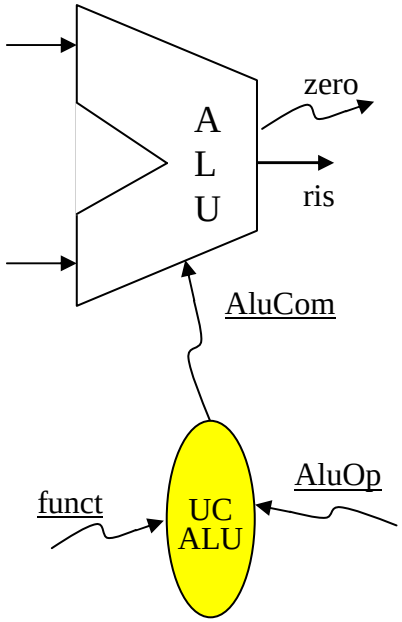
Sommatore PC+offset (beq)

• Per integrare i diversi elementi:

- ALU principale deve effettuare diverse operazioni a seconda dell'istruzione
- servono MUX per selezionare i dati su cui un elemento funzionale lavora (diversi da istruzione a istruzione)

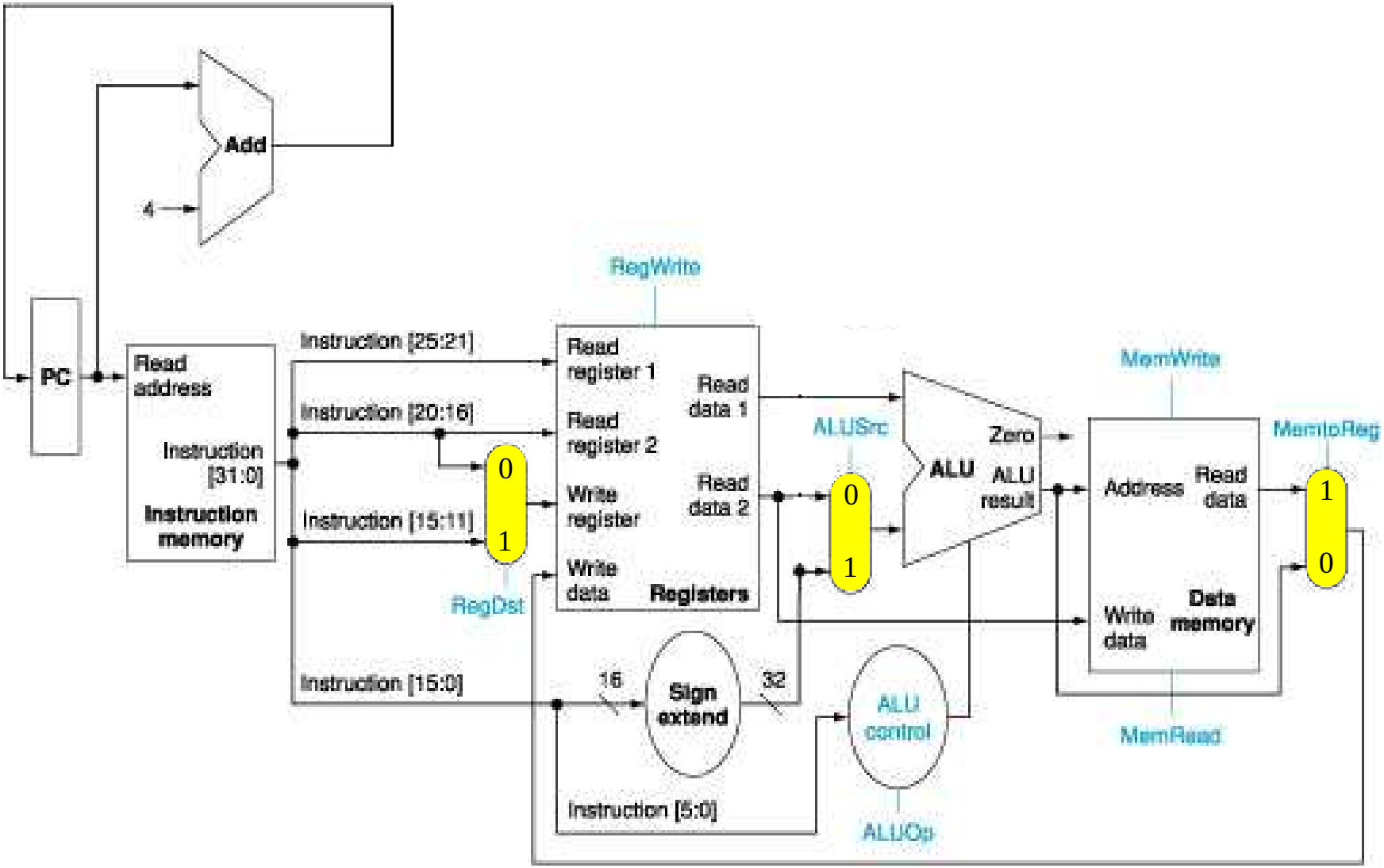
Il controllo (combinatorio) della ALU

- Unità di controllo è critica dal punto di vista delle performance
  - ⇒ più livelli di controllo:
    - ↓ dimensioni, ↑ velocità

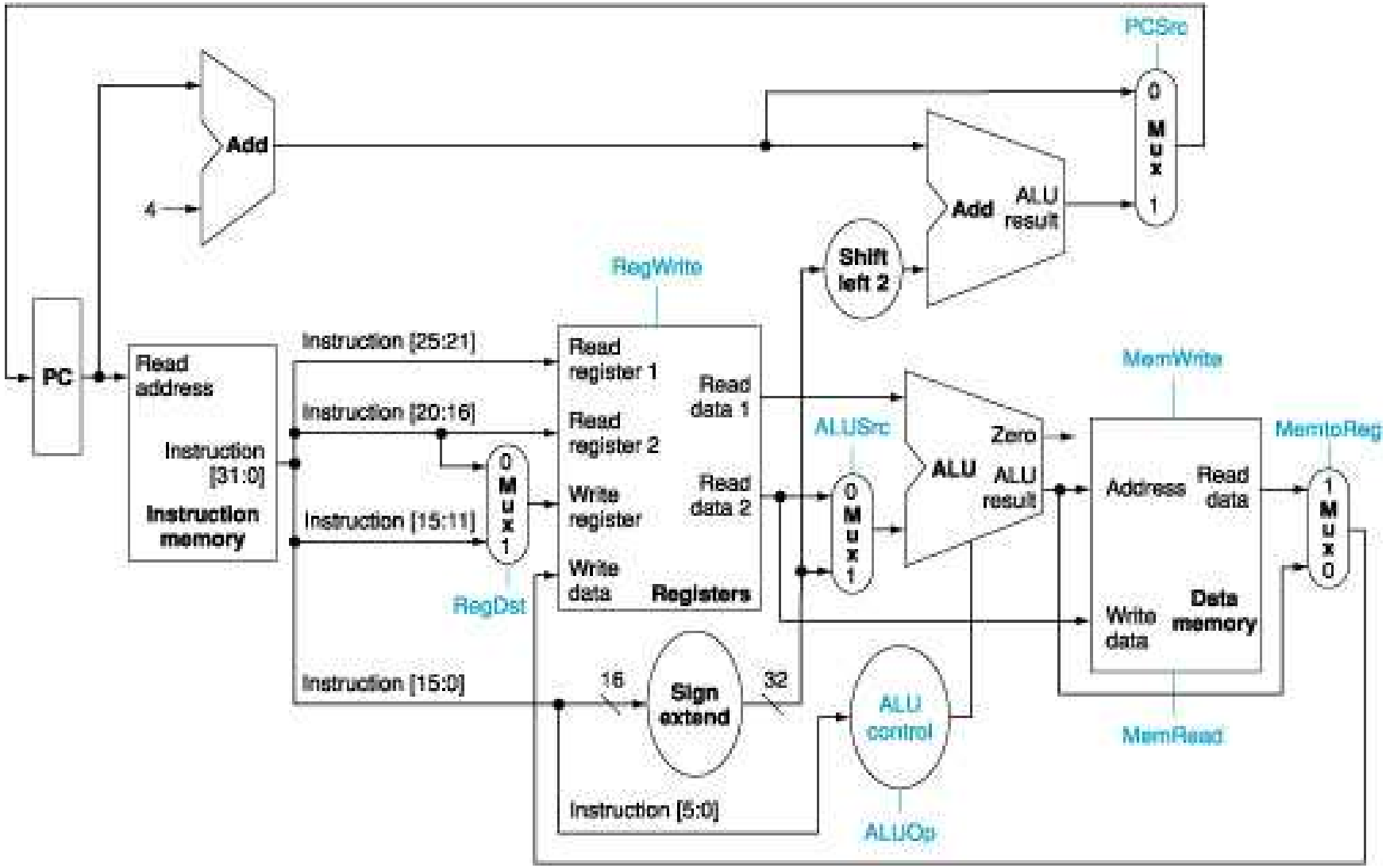


opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

Integrazione della lw e sw



Datapath per Tipo-R, lw, sw, beq



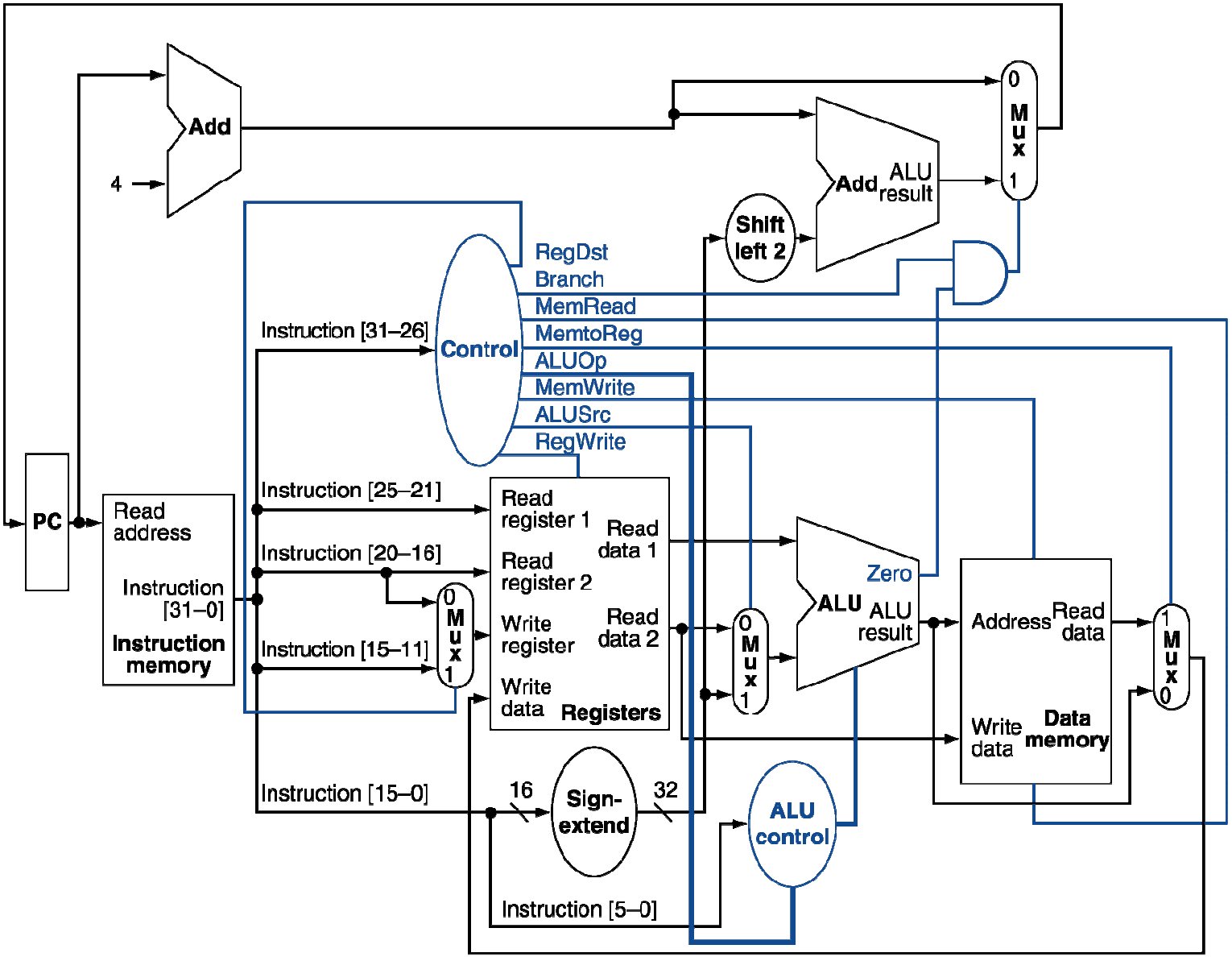
PROGETTAZIONE
UNITA’ DI CONTROLLO

(CASO MIPS)

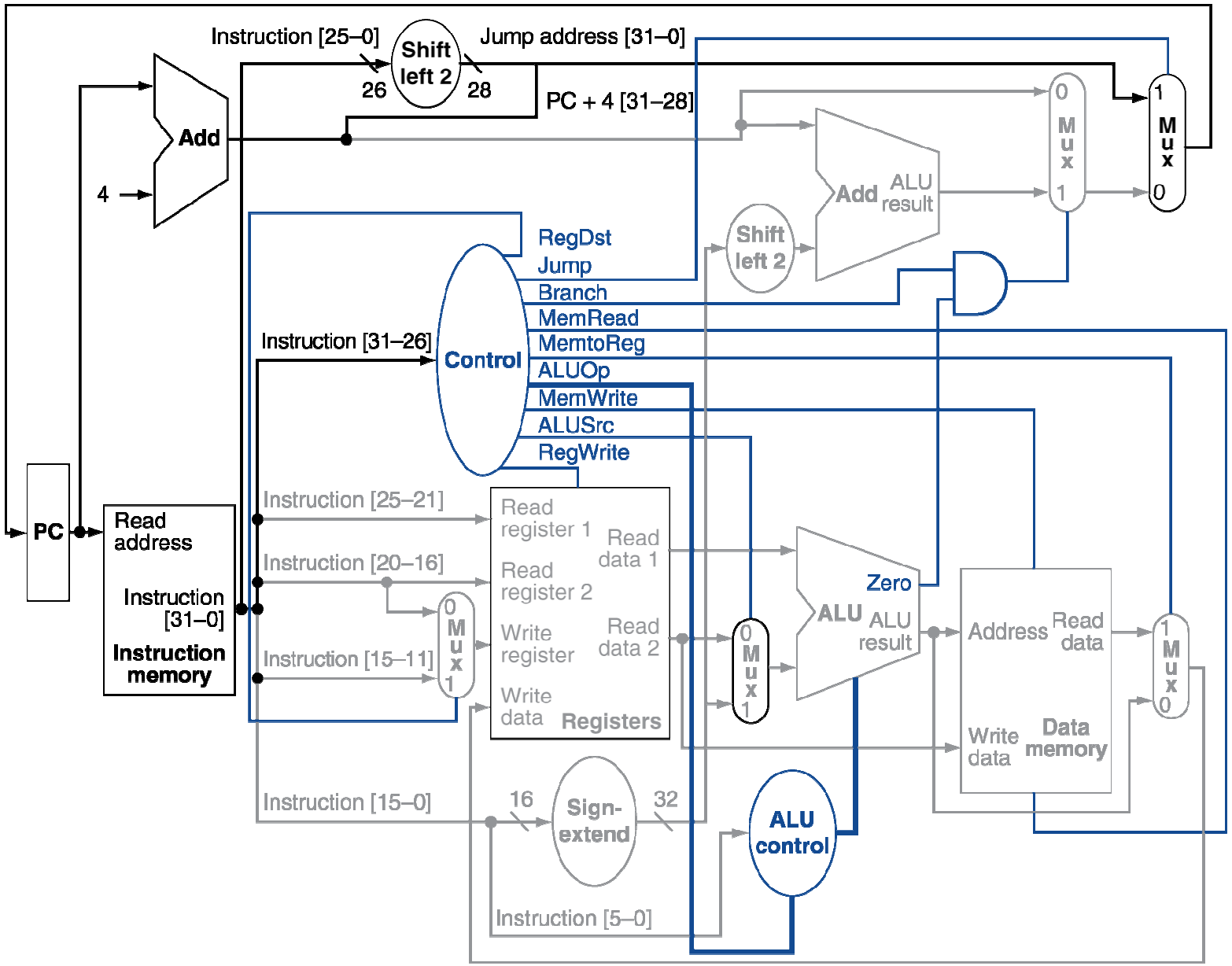
- cominciamo nel datapath a “semplificare” il controllo per beq

Datapath con il controllo

(cfr. la parte relativa a branch)



Datapath con il controllo (include j)

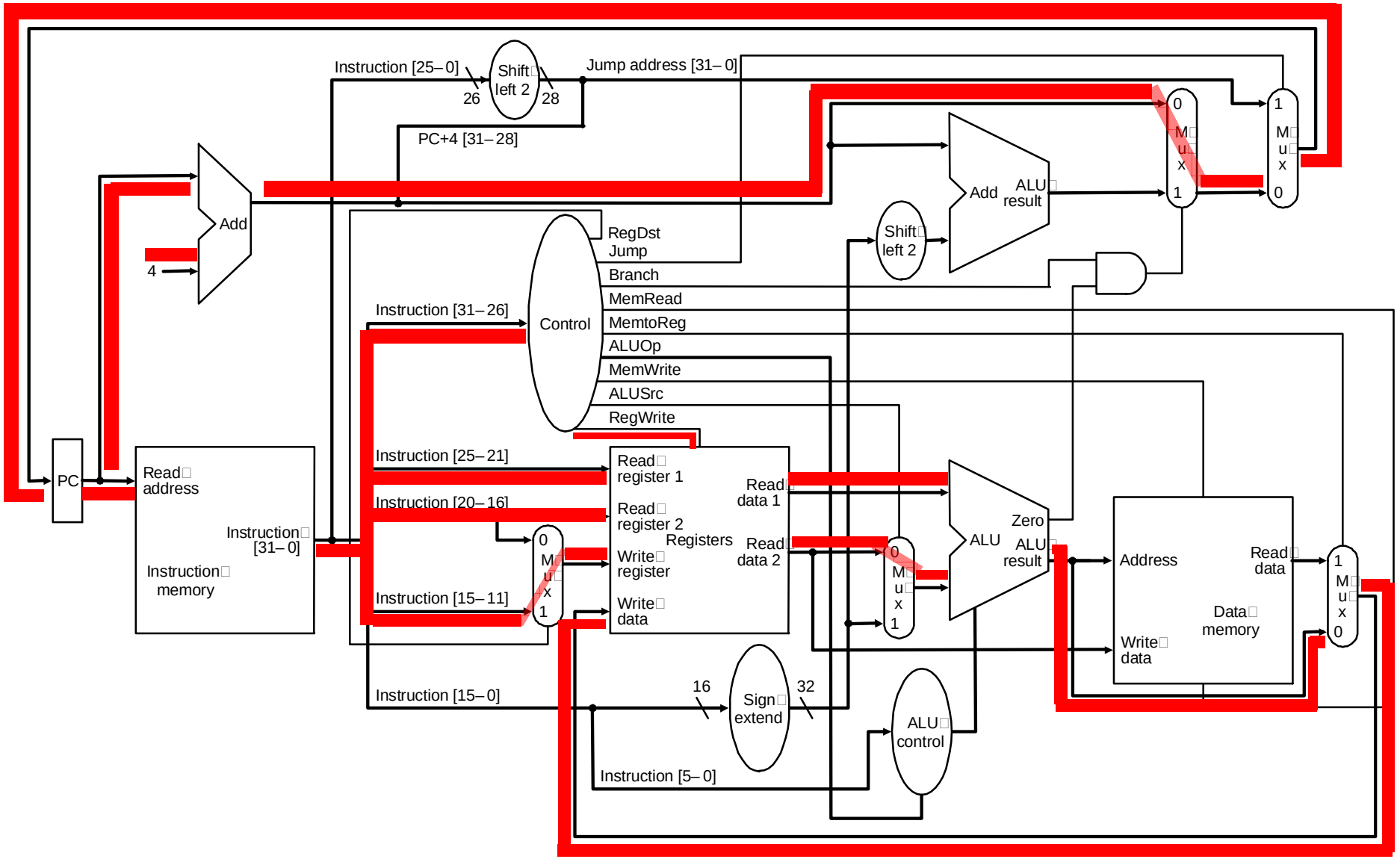


Nota sui segnali di controllo:

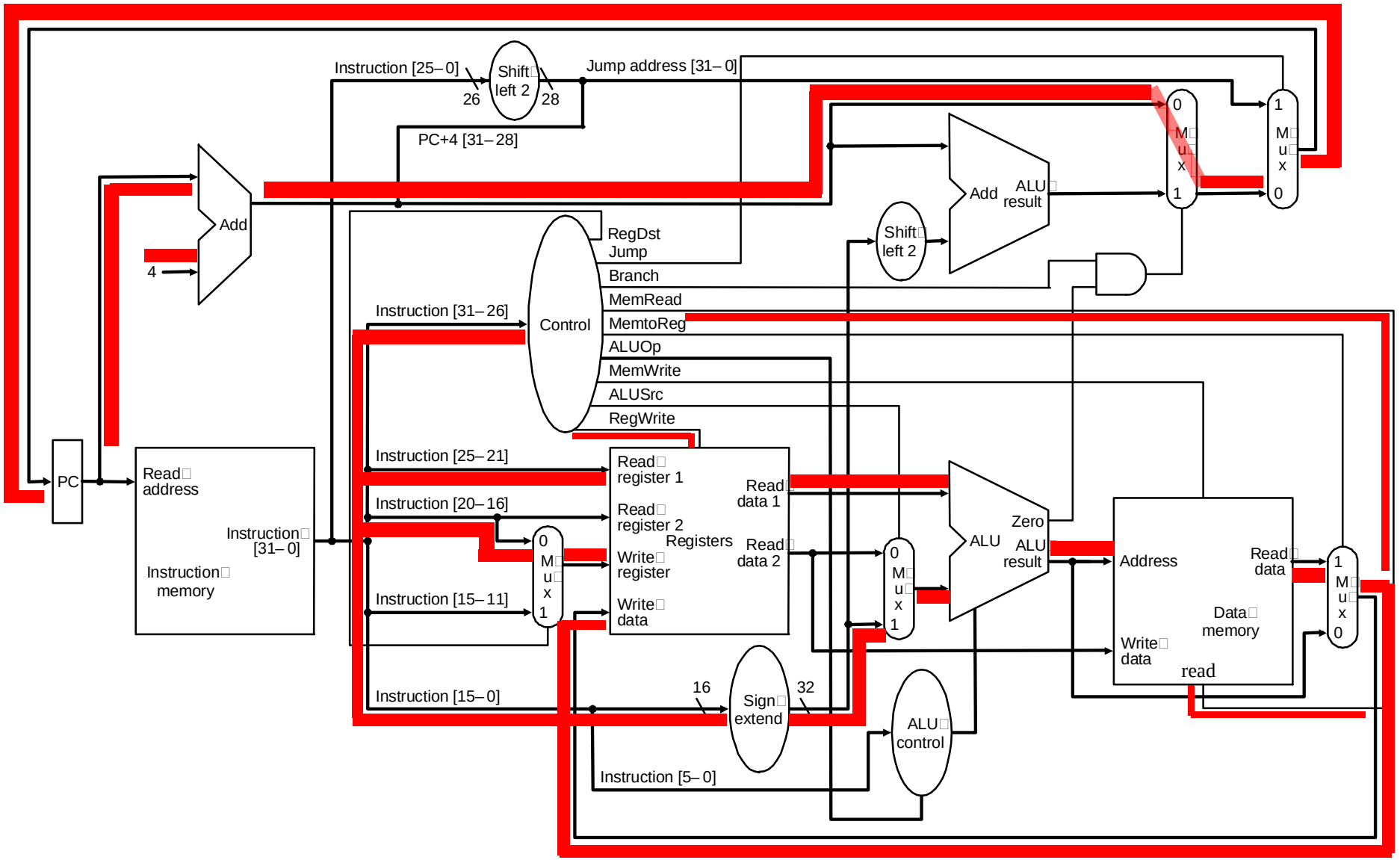
- i segnali di controllo sono determinati in modo “combinatorio” soltanto sulla base del campo Opcode
- non è in generale possibile prevedere l’ordine di arrivo dei segnali di controllo: le operazioni non sono eseguite “in sequenza”, controllo combinatorio
  (è necessario che T_{clock} sia sufficientemente lungo)

Prima di progettare l’unità di controllo, vediamo allora alcuni esempi di esecuzione delle istruzioni…

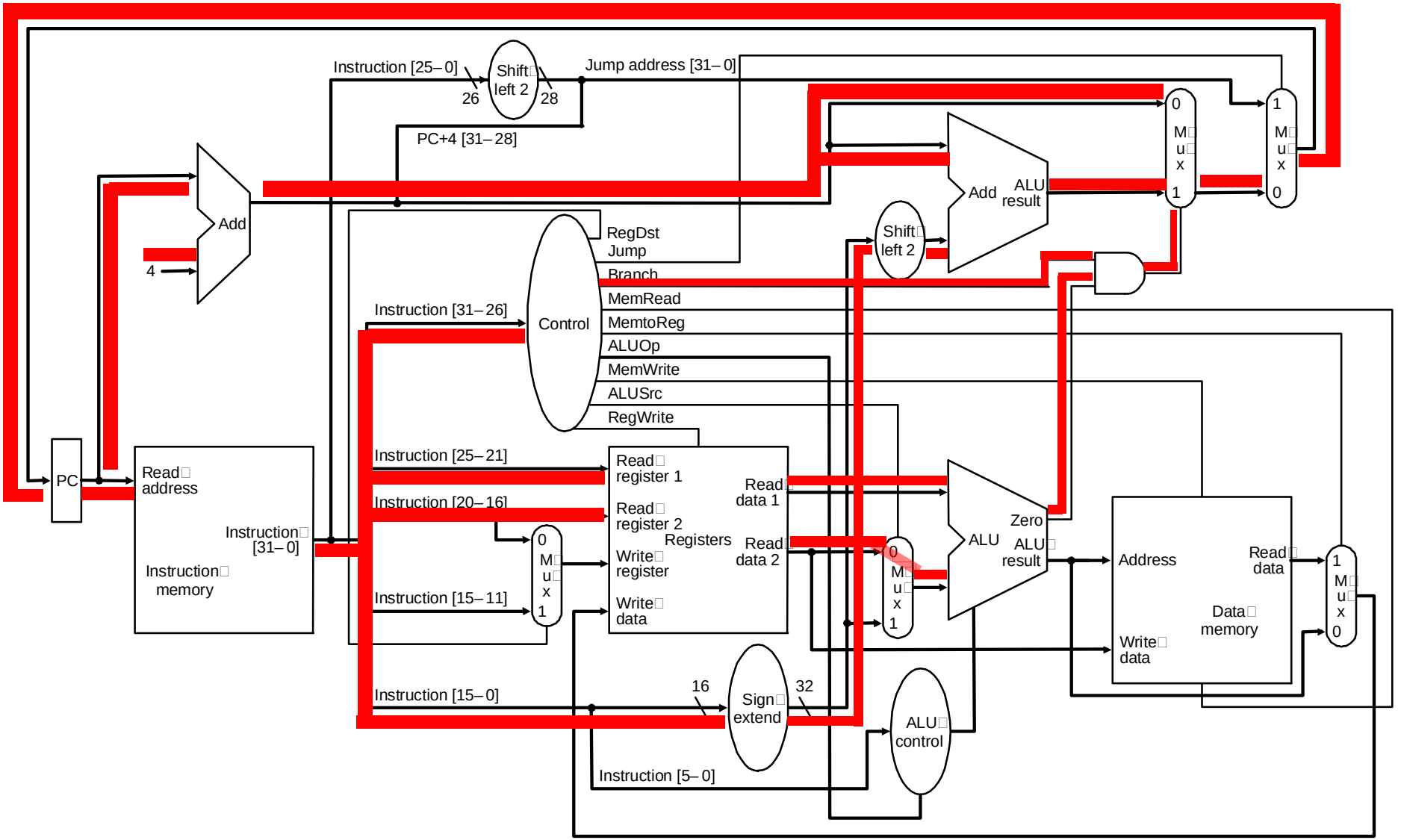
Esempio: istruzione di tipo-R



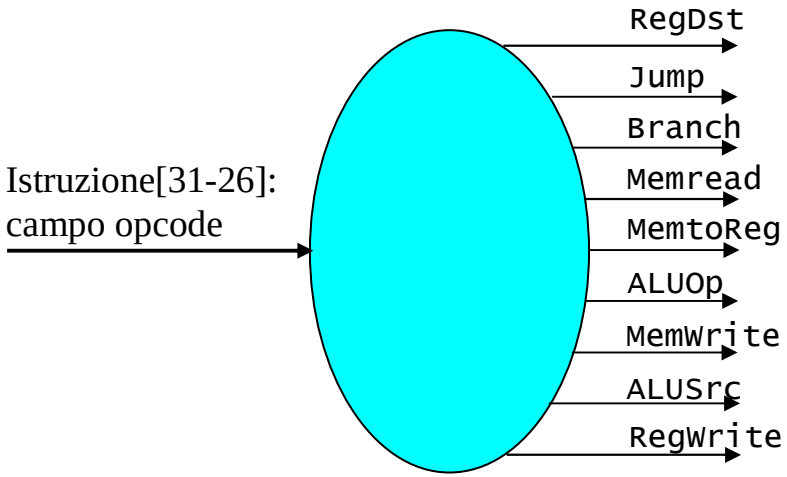
Esempio: istruzione di load



Esempio: istruzione beq



Progetto e realizzazione dell'unità di controllo principale



E' una rete combinatoria:

- sulla base di opcode (istruzione da eseguire)

deve selezionare i valori dei segnali di controllo

→ Può essere specificata da tabella di verità e implementata nei modi visti (porte logiche, PLA, ROM)

ISTRUZ	RegDst	Jump	Branch	Mem Read	Memto Reg	ALUOp	Mem Write	ALUSrc	Reg Write
Tipo-R	1	0	0	0	0	10	0	0	1
lw	0	0	0	1	1	00	0	1	1
sw	X	0	0	0	X	00	1	1	0
beq	X	0	1	0	X	01	0	0	0
j	X	1	X	0	X	XX	0	X	0

Opcode [6 bit]