Calcolatori Elettronici B
a.a. 2008/2009

RICHIAMI DI CALCOLATORI A

Massimiliano Giacomin

IL LIVELLO HARDWARE

istruzioni macchina

ISA

Livello architetturale

Organizzazione di componenti per implementare ISA

Reti logiche:
registri, ALU, MUX…

Livello logico

Modelli logici:
si parla di variabili, valori… binari!

Porte logiche:
NOT, AND, …

Livello circuitale

Modelli elettronici:
si parla di
tensioni, correnti, ecc.

Transistor

Livello del layout

Modelli fisici: si parla di dimensioni fisiche, materiali, ecc.

Due tipi di unità funzionali

- Elementi di tipo *combinatorio*:
  - valori di uscita dipendono solo da valori in ingresso
  - Es. Porte logiche, PLA
- Elementi *di memoria:*
  - capaci di memorizzare un valore
  - Es. flip-flop, registri, memoria RAM

Due tipi di reti

- **RETI COMBINATORIE**
  - Contengono solamente elementi di tipo combinatorio

    ⇒ Valori di uscita dipendono solo da valori di ingresso
- **RETI SEQUENZIALI**
  - Contengono elementi di memoria

    ⇒ Valori di uscita dipendono dalla storia del sistema (sequenza di tutti gli ingressi) – sintetizzata nel valore di stato contenuto negli elementi di memoria.

RETI COMBINATORIE

Assumiamo **n** ingressi **m** uscite

Specifica

- Si possono specificare mediante due approcci alternativi:
  - **tabelle di verità**

    (n ingressi $\Rightarrow 2^n$ righe, per ciascuna si specificano tutte le m uscite)

Es.

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

(OR esclusivo)

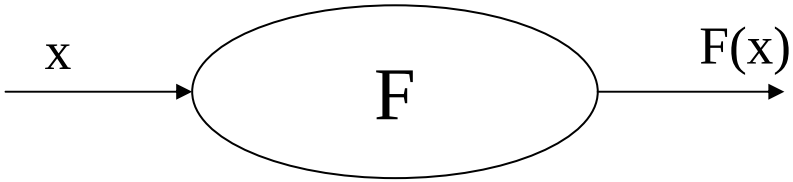
  - **equazioni logiche** (algebra booleana)

    Es. $A\overline{B}+\overline{A}B$

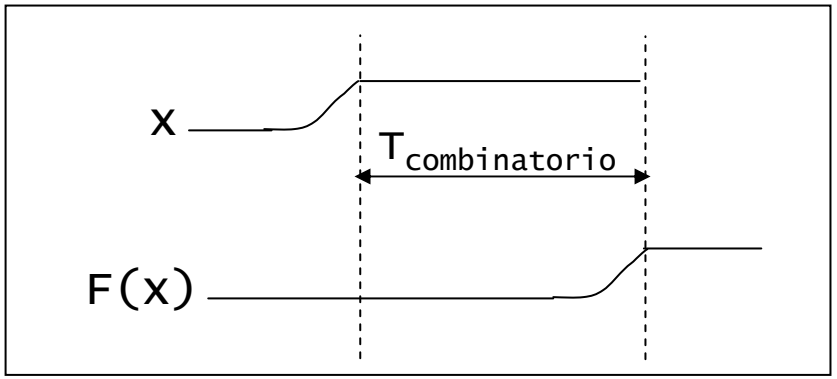
Realizzazione: Porte logiche, PLA, ROM, PLD (Programmable logic device)

- Elettronica digitale: realizzazione degli elementi sopraindicati (es. Famiglie logiche TTL, CMOS) – noi non ce ne occupiamo direttamente
- Determina parametri tecnologici che influenzano le prestazioni

Tempo di propagazione



Dal momento in cui l'ingresso è valido al momento in cui l'uscita è valida trascorre un certo intervallo temporale:



Al max dopo $T_{combinatorio}$ siamo sicuri che l'uscita è valida e stabile

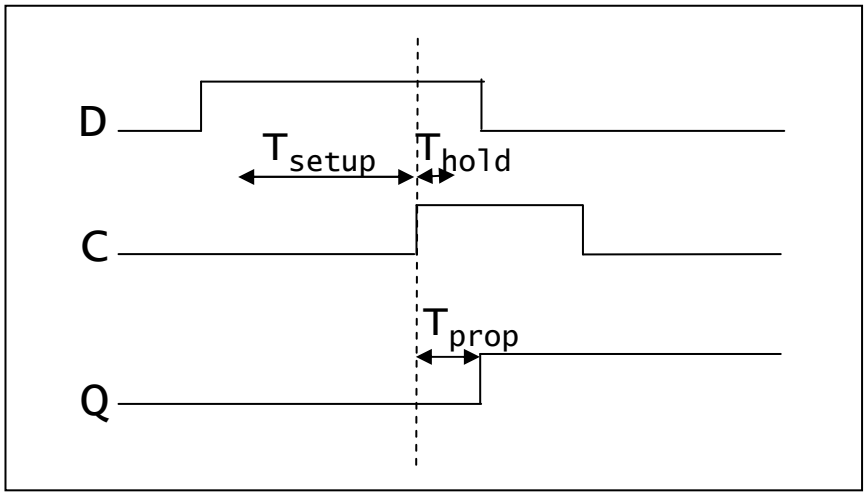
RETI SEQUENZIALI

Elementi di memoria

- **Flip-flop di tipo D**

Sensibile ai fronti: l'ingresso è memorizzato sul fronte (di salita) del clock

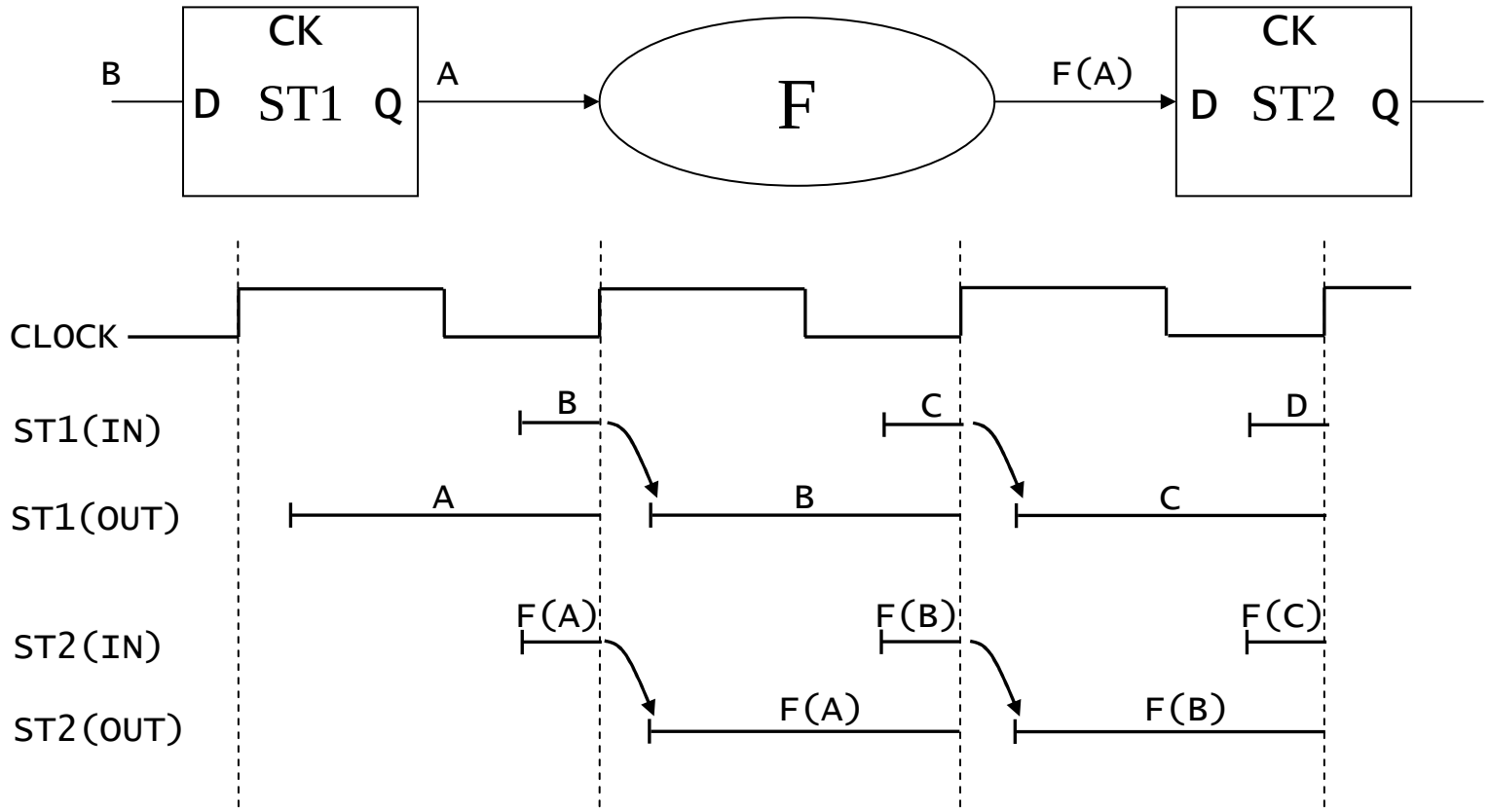
- Vincoli sull'ingresso: tempo di setup e tempo di hold
  Ritardo sull'uscita: tempo di propagazione



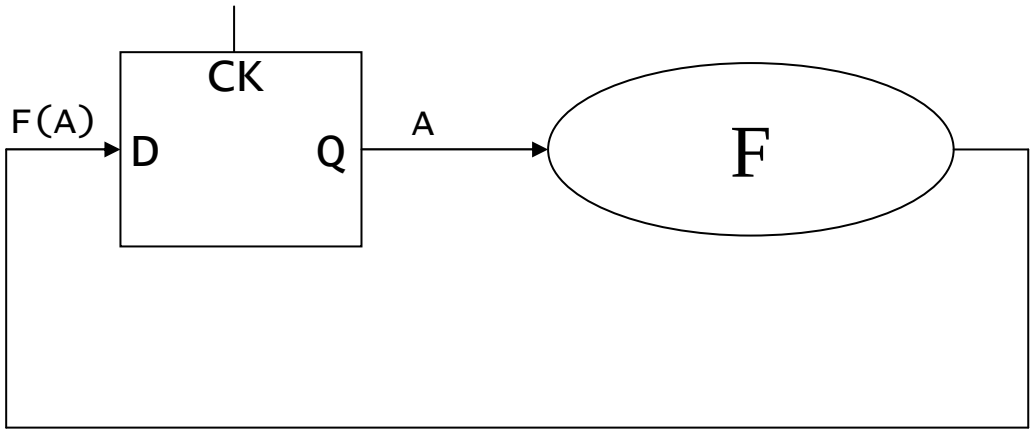
- Tutti i tempi riferiti al fronte del clock
- In generale $T_{hold} < T_{prop}$

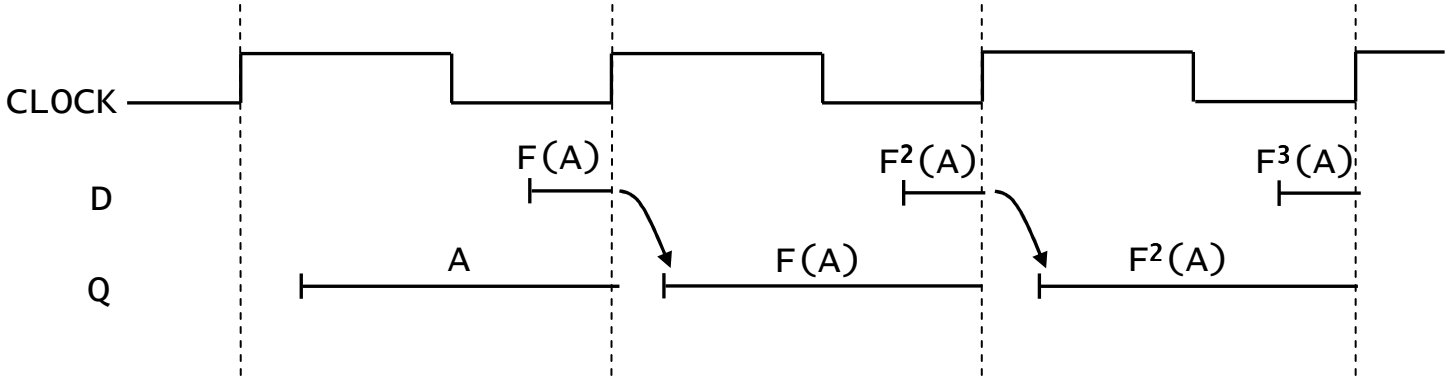
TEMPORIZZAZIONE

• Sistemi sincroni: segnale di clock comune determina aggiornamento elementi di stato

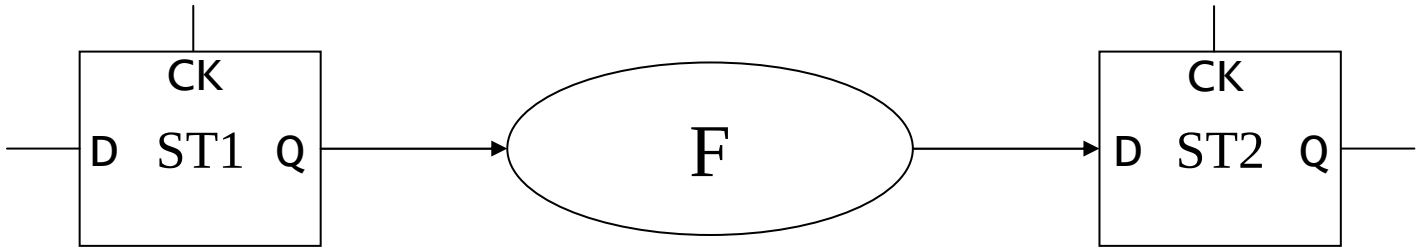


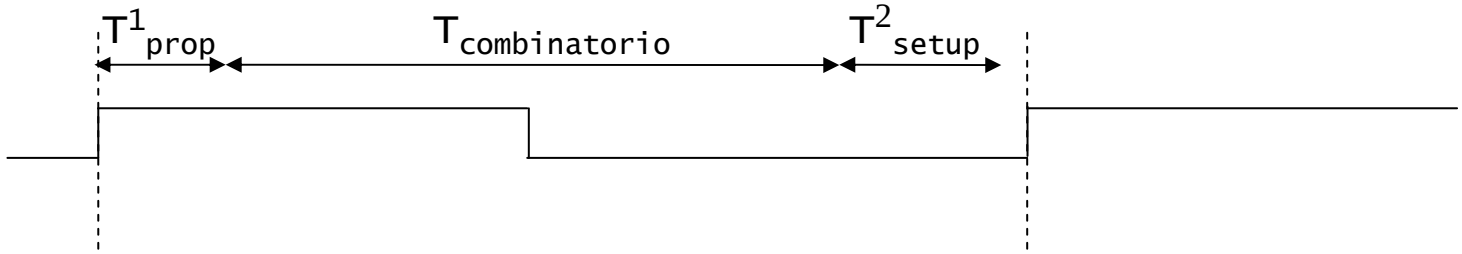
• Al fronte di clock, un elemento di stato memorizza il valore di ingresso
• Nel periodo di clock, un nuovo valore di ingresso viene propagato dalla parte combinatoria e sarà disponibile al successivo fronte


CK
F(A)
D
Q
A
F


CLOCK
D
Q
F(A)
F²(A)
F³(A)
A
F(A)
F²(A)

Temporizzazione e vincoli temporali





- Dopo $T^1_{prop} + T_{combinatorio}$, ingresso a ST2 è stabile: anticipo di almeno T^2_{setup}

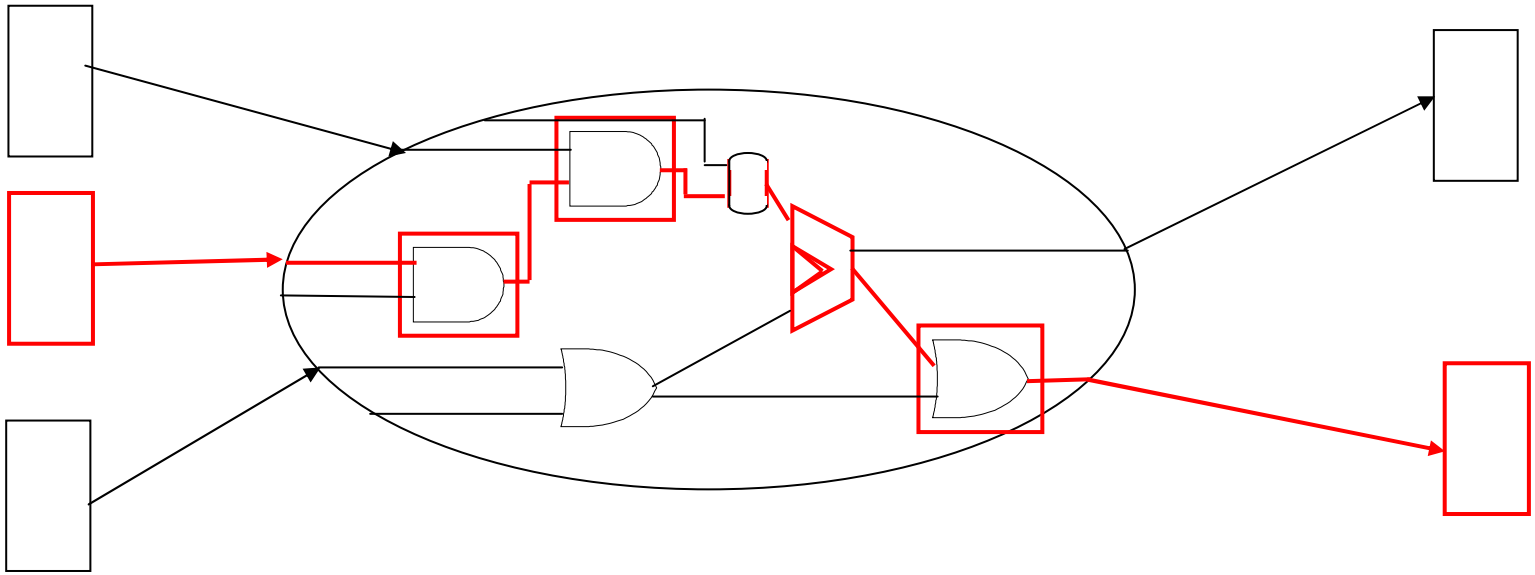
  ⇒ $T_{clock} \geq T^1_{prop} + T_{combinatorio} + T^2_{setup}$

- Vincolo per rispetto di T^2_{hold} : ingresso ST2 permane per almeno T^2_{hold} dopo il fronte

  ⇒ $T^1_{prop} + T_{combinatorio} \geq T^2_{hold}$

  [verificato automaticamente perché $T_{hold} < T_{prop}$]

ESTENDENDO QUESTE CONSIDERAZIONI AD UNA RETE COMPLESSA…



Occorre considerare il caso peggiore; in particolare il **“cammino critico”** vincola la lunghezza del periodo di clock e quindi limita la frequenza ottenibile!

$$T_{clock} \geq T_{prop} + T_{cammino\ critico} + T_{setup}$$

Nel caso peggiore!

Specifica di una rete sequenziale: macchine a stati finiti (FSM)

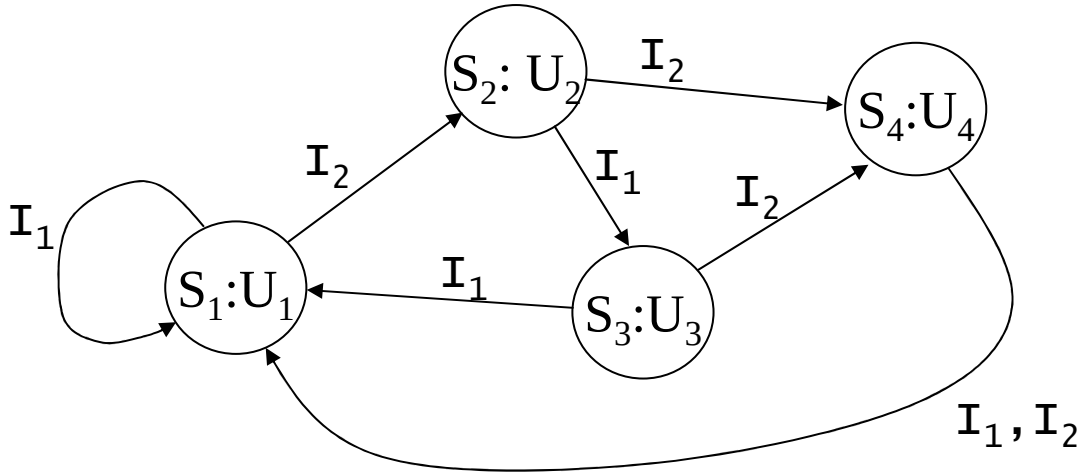
Occorre definire:

- L'insieme degli ingressi (dominio **I**) e delle uscite (dominio **U**)
- L'insieme degli stati **S**
- Dinamica (come si passa da uno stato all'altro):

  funzione **f**: **S*I $\rightarrow$ S**
- Come si generano le uscite

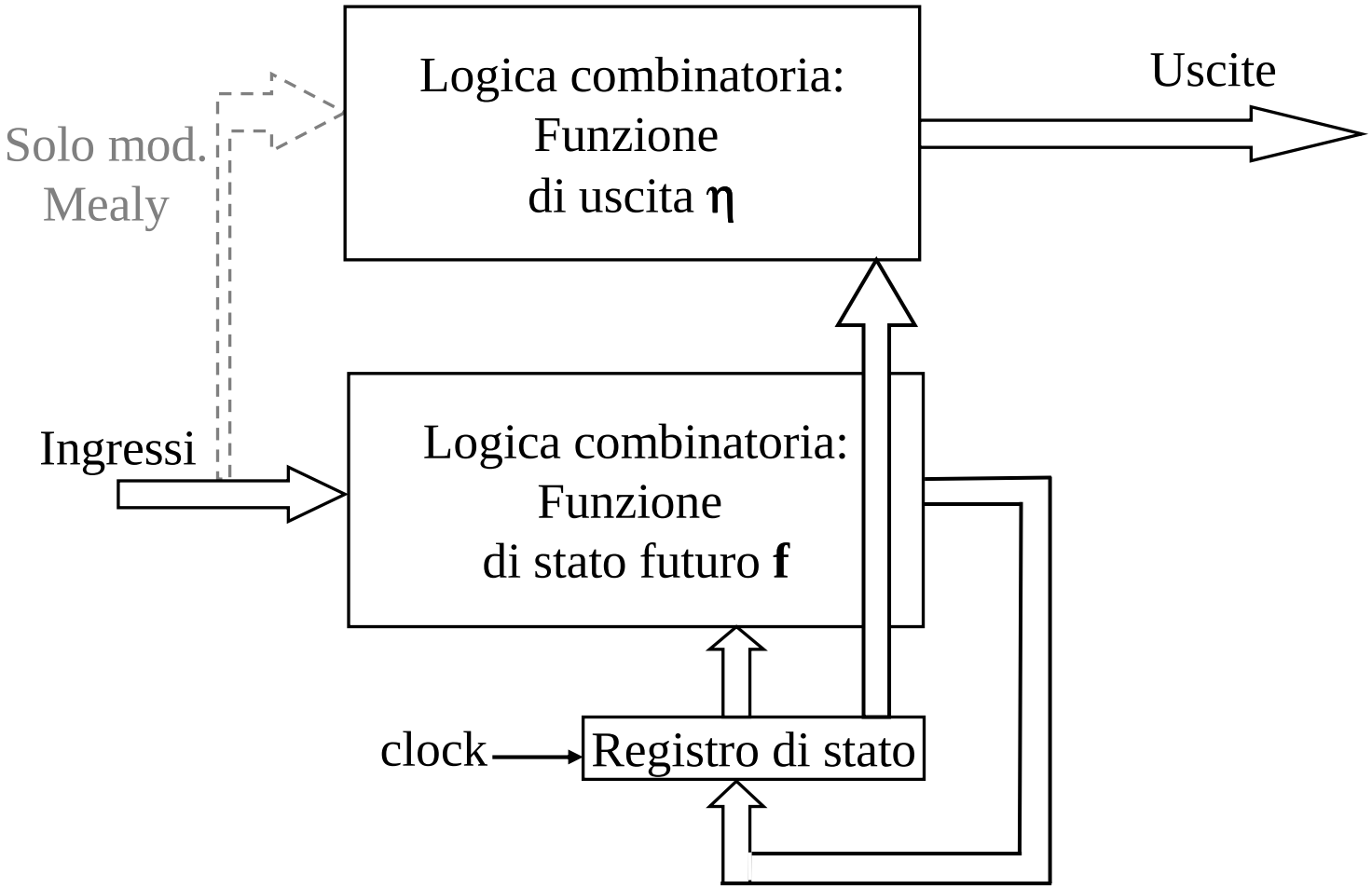
  funzione η **η: S $\rightarrow$ U** **Modello di Moore**

  η: S*I $\rightarrow$ U **Modello di Mealy**



[Modello di Moore]

Implementazione di una rete sequenziale



IL LIVELLO HARDWARE

istruzioni macchina

ISA

Livello architetturale

Organizzazione di componenti per implementare ISA

Reti logiche:
registri, ALU, MUX…

Livello logico

Modelli logici:
si parla di variabili, valori… binari!

Porte logiche:
NOT, AND, …

Livello circuitale

Modelli elettronici:
si parla di
tensioni, correnti, ecc.

Transistor

Livello del layout

Modelli fisici: si parla di dimensioni fisiche, materiali, ecc.

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

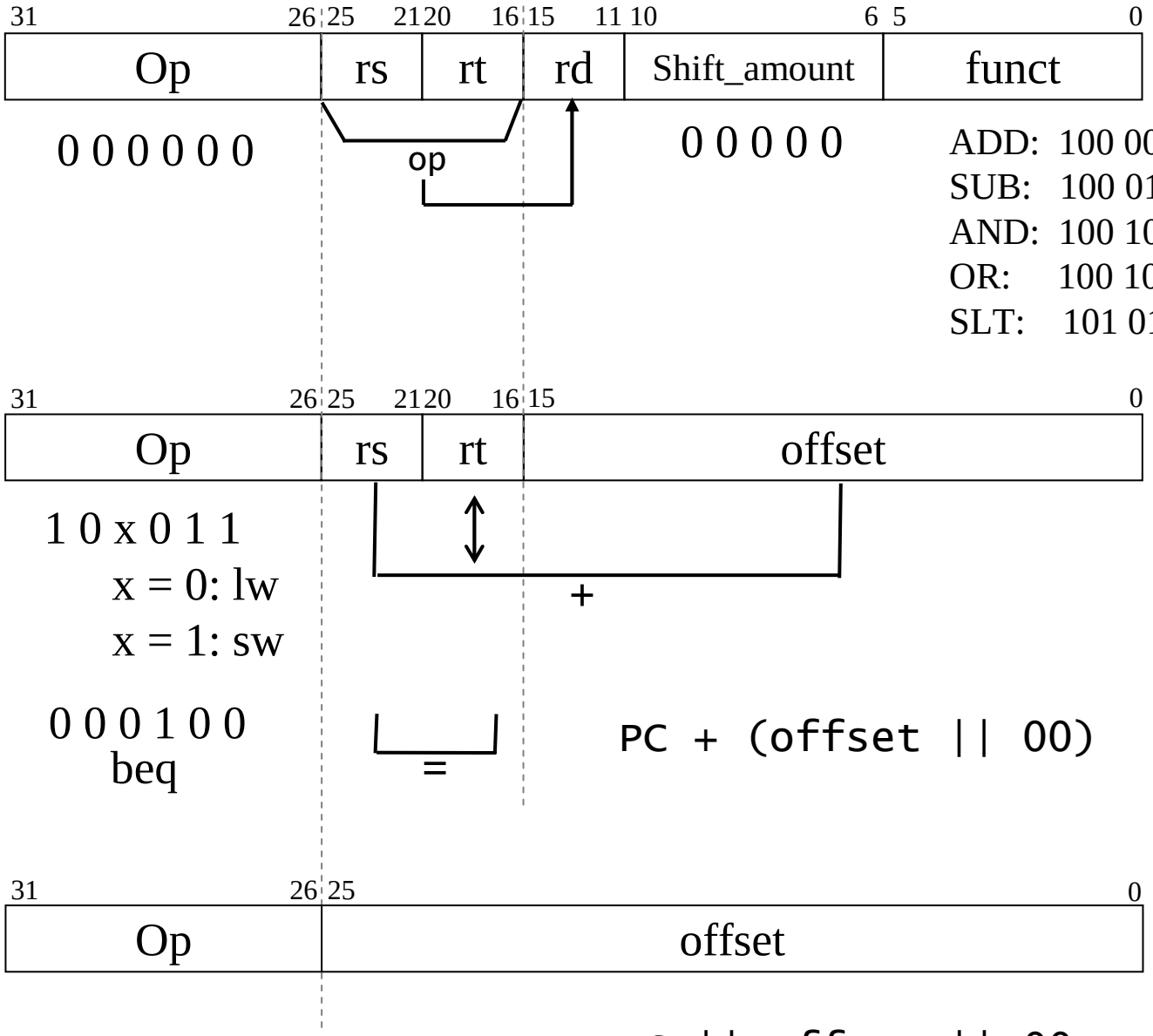
- Istruzioni di salto condizionato:

  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:

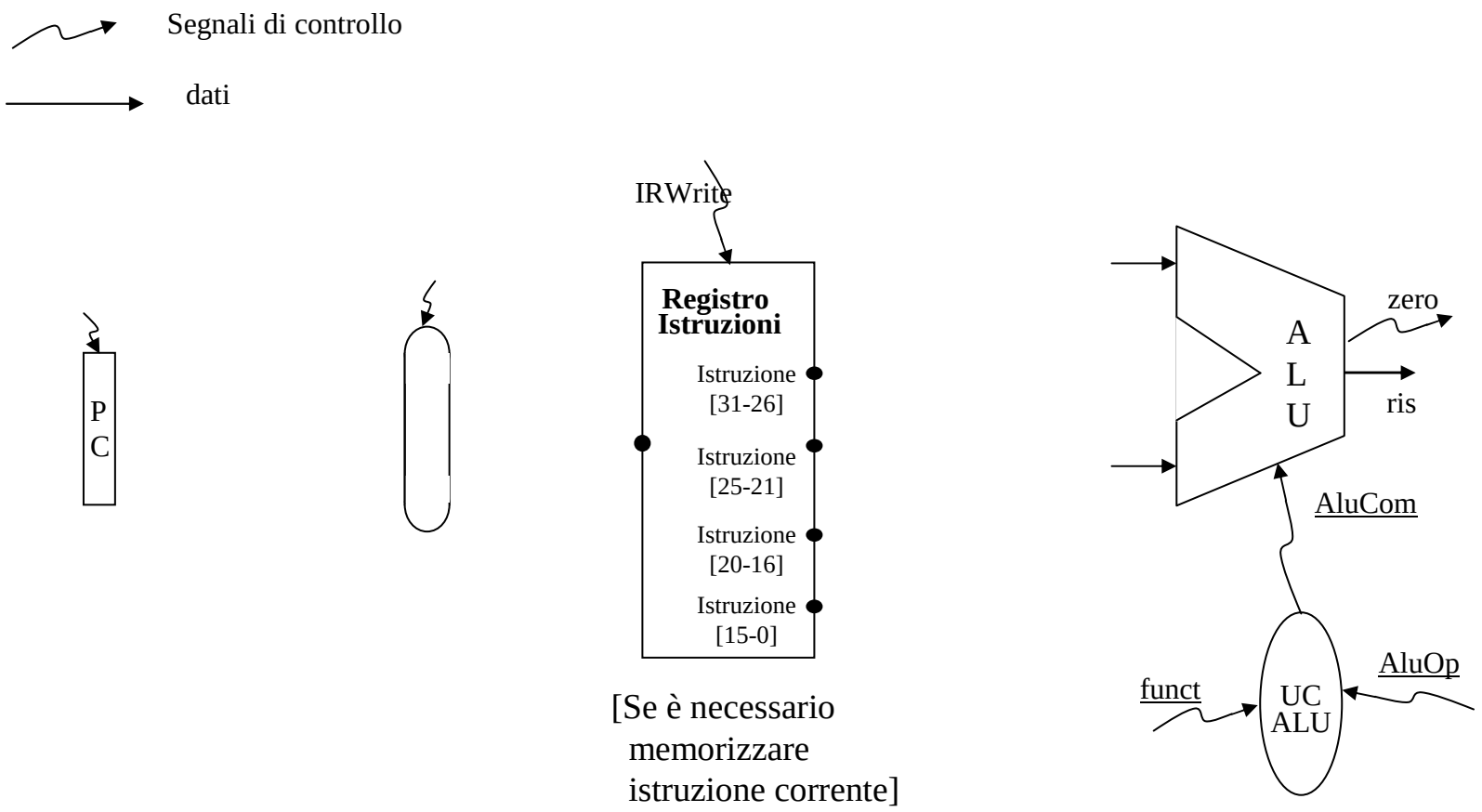


Aritmetiche: Tipo-R

lw, sw, beq: Tipo-I

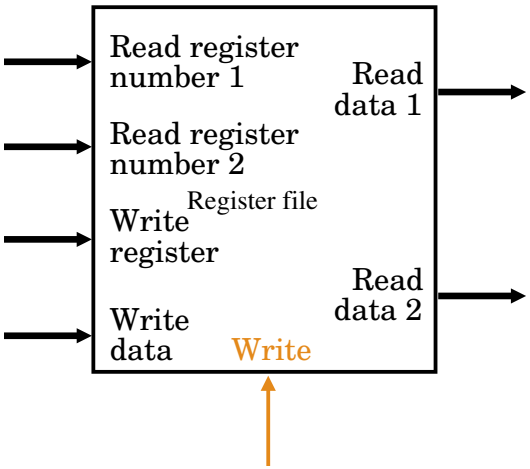
J: Tipo-J

Glossario Visuale: indica le risorse individuate in base ad una prima analisi



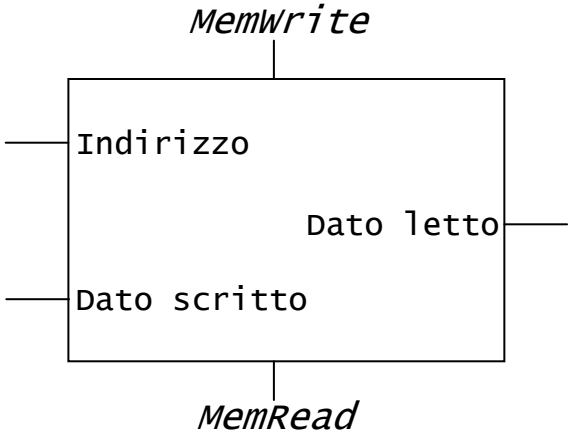
• **Registri**: capaci di memorizzare un insieme di bit
(si possono ottenere mediante *array* di Flip-flop di tipo D)

• **Register File:**



- Lettura: asincrona rispetto al clock, senza segnale di controllo *read*
- Scrittura: attiva sul fronte del clock e solo quando *write* è affermato
- Implementato con registri, multiplexer (per read port) e decodificatore (per write port)

• **Memorie** (per memorizzare quantità maggiori di dati)

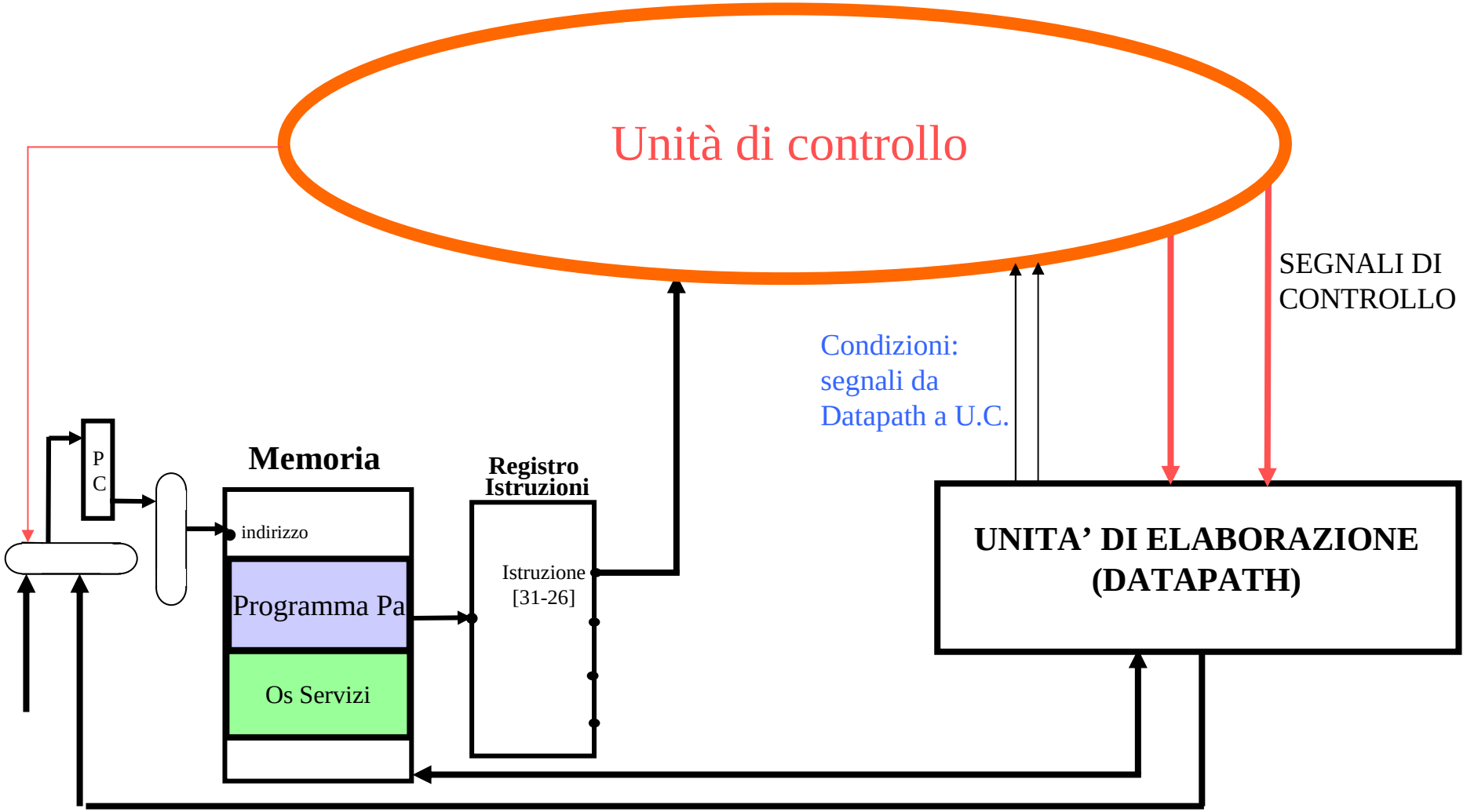


- Lettura asincrona risp. clock, scrittura attiva sul fronte del clock
- NB: forma semplificata (cfr. SRAM, DRAM, ecc.)

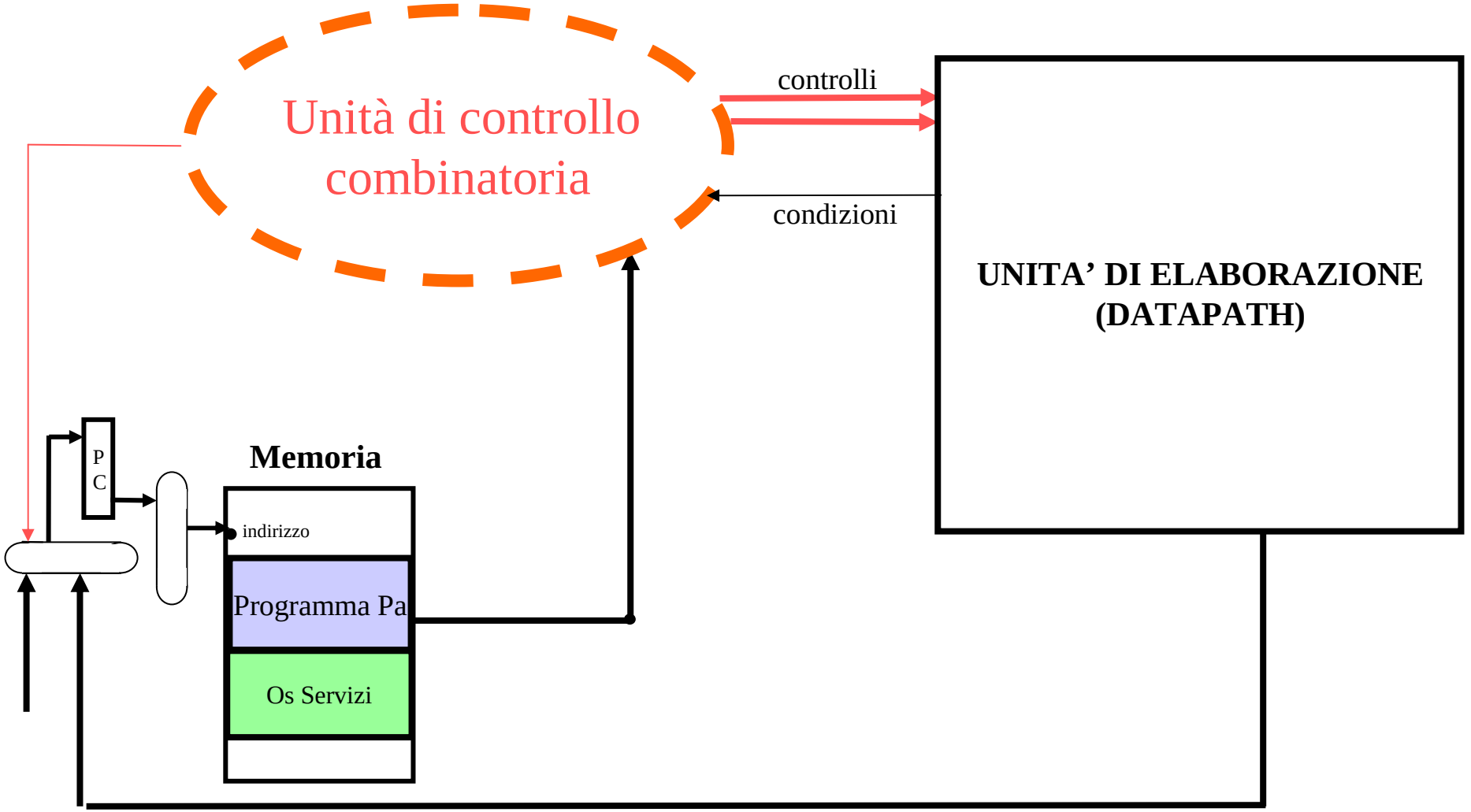
NB: il clock è presente ma non viene indicato per rendere le figure più chiare.

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>

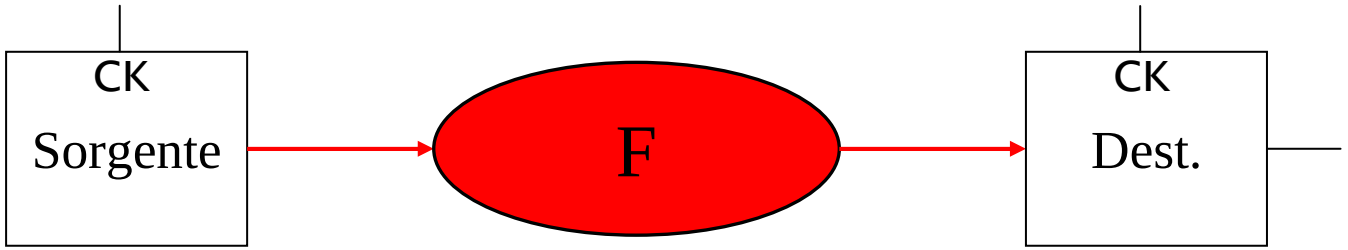


Controllo di un processore a singolo ciclo: l'idea di base



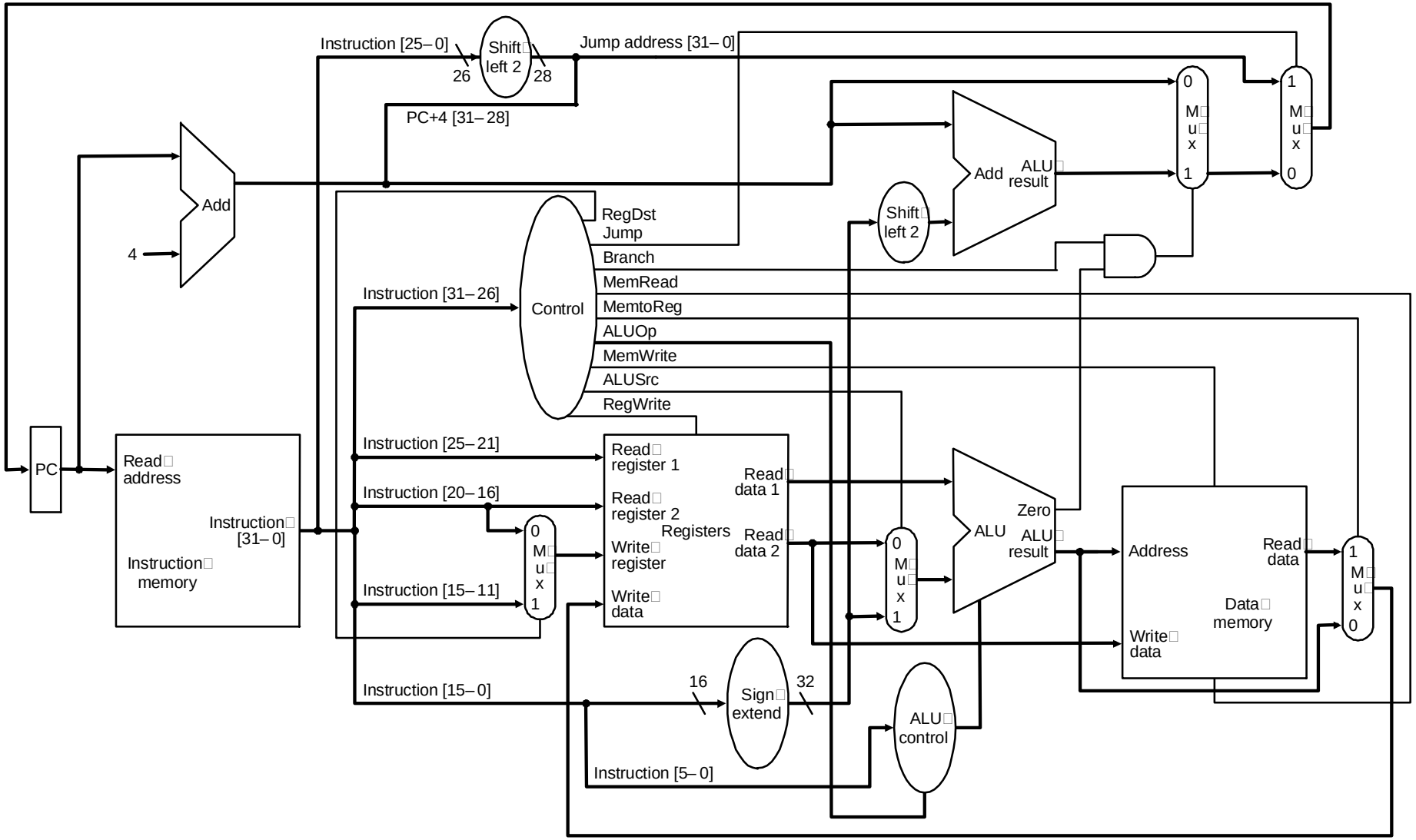
Idea di base

- 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

Il MIPS

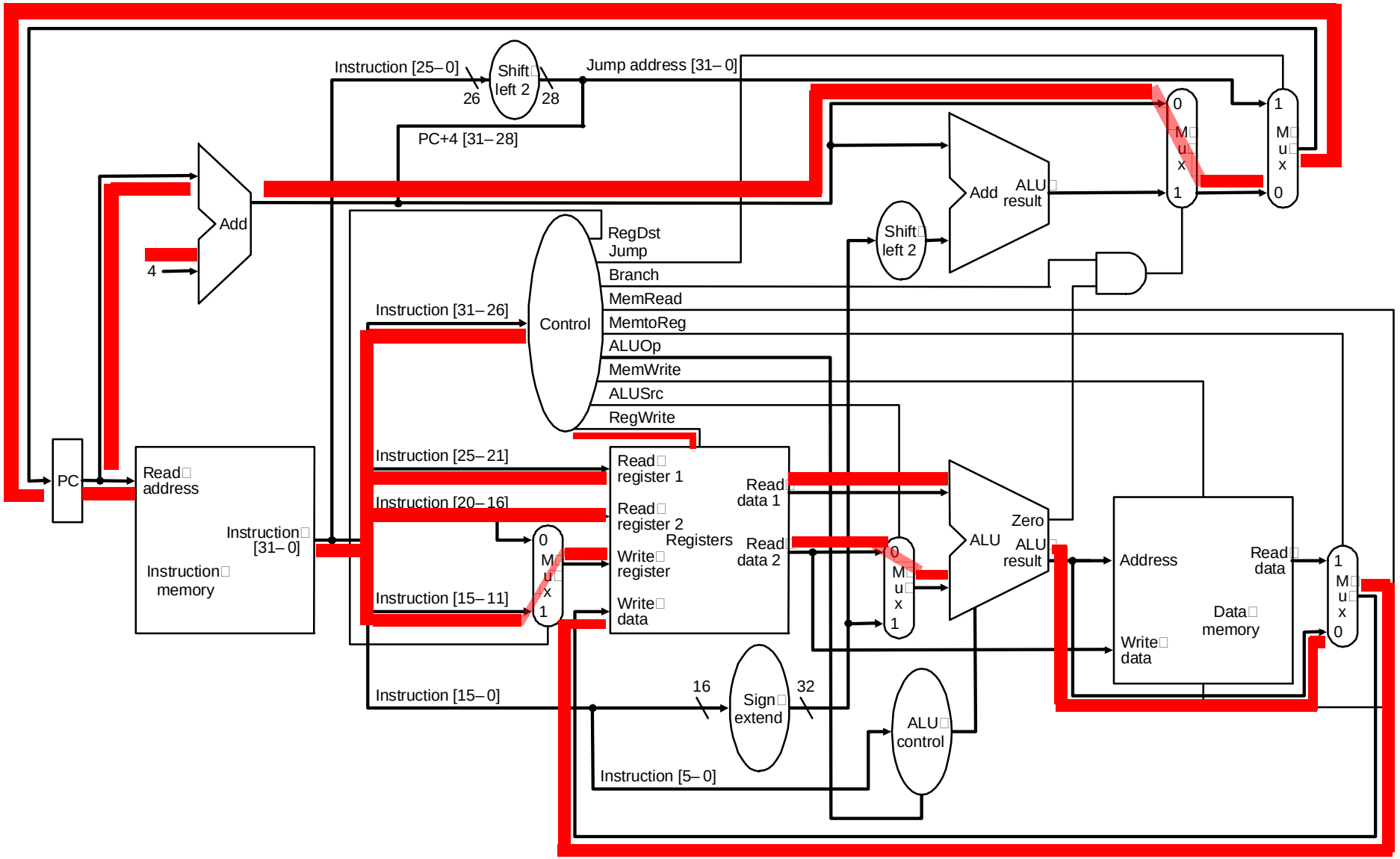

Instruction [25–0]
Shift left 2
26
28
Jump address [31–0]
PC+4 [31–28]
Add
4
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
Shift left 2
Add
ALU result
M u x
0
1
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
ALU control
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

Nota sui segnali di controllo:

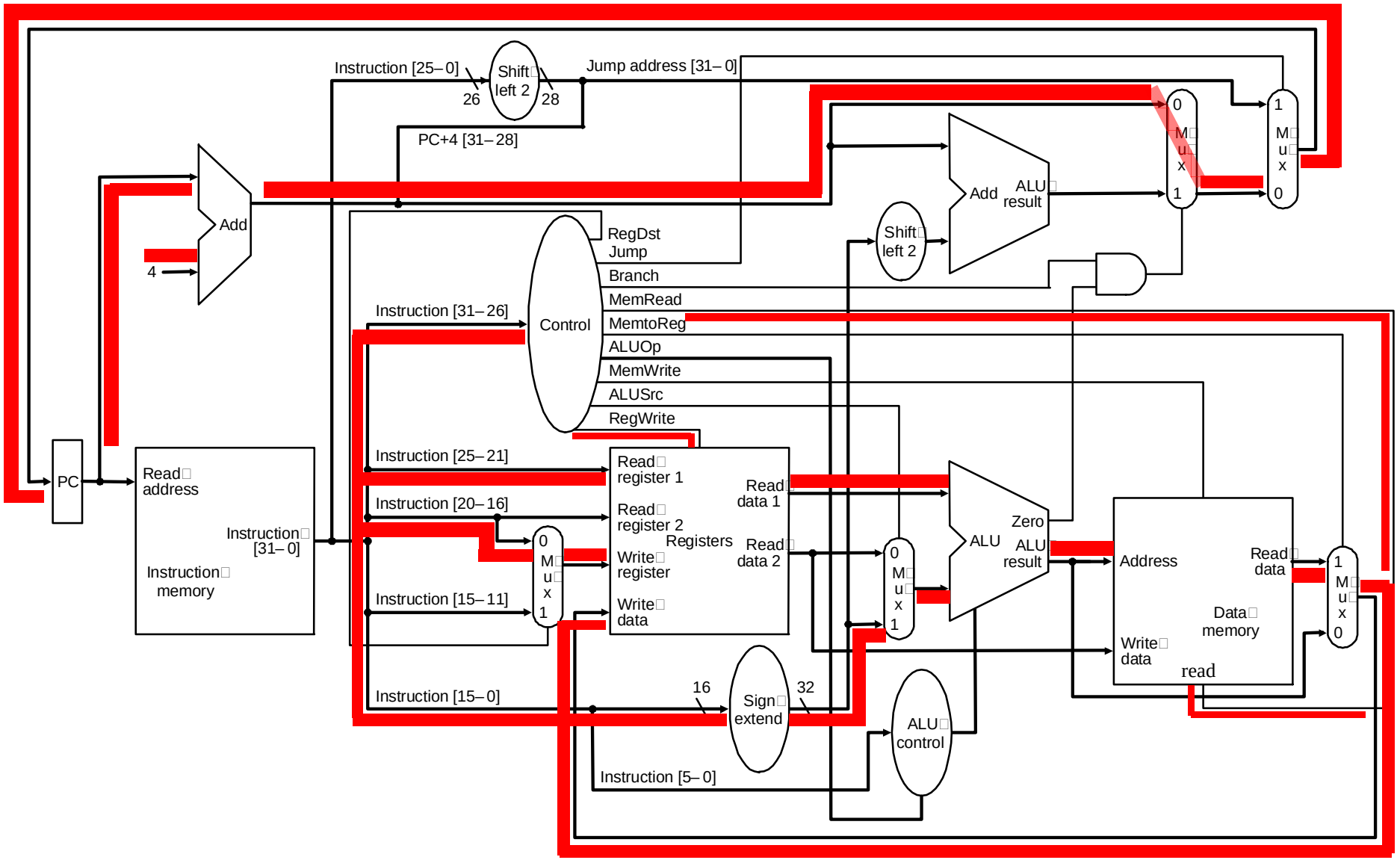
- i segnali di controllo sono determinati in modo “combinatorio” soltanto sulla base del campo Opcode
- non è in generale possibile prevedere l’ordine di arrivo dei segnali di controllo: le operazioni non sono eseguite “in sequenza”, controllo combinatorio

  (è necessario che T_{clock} sia sufficientemente lungo)

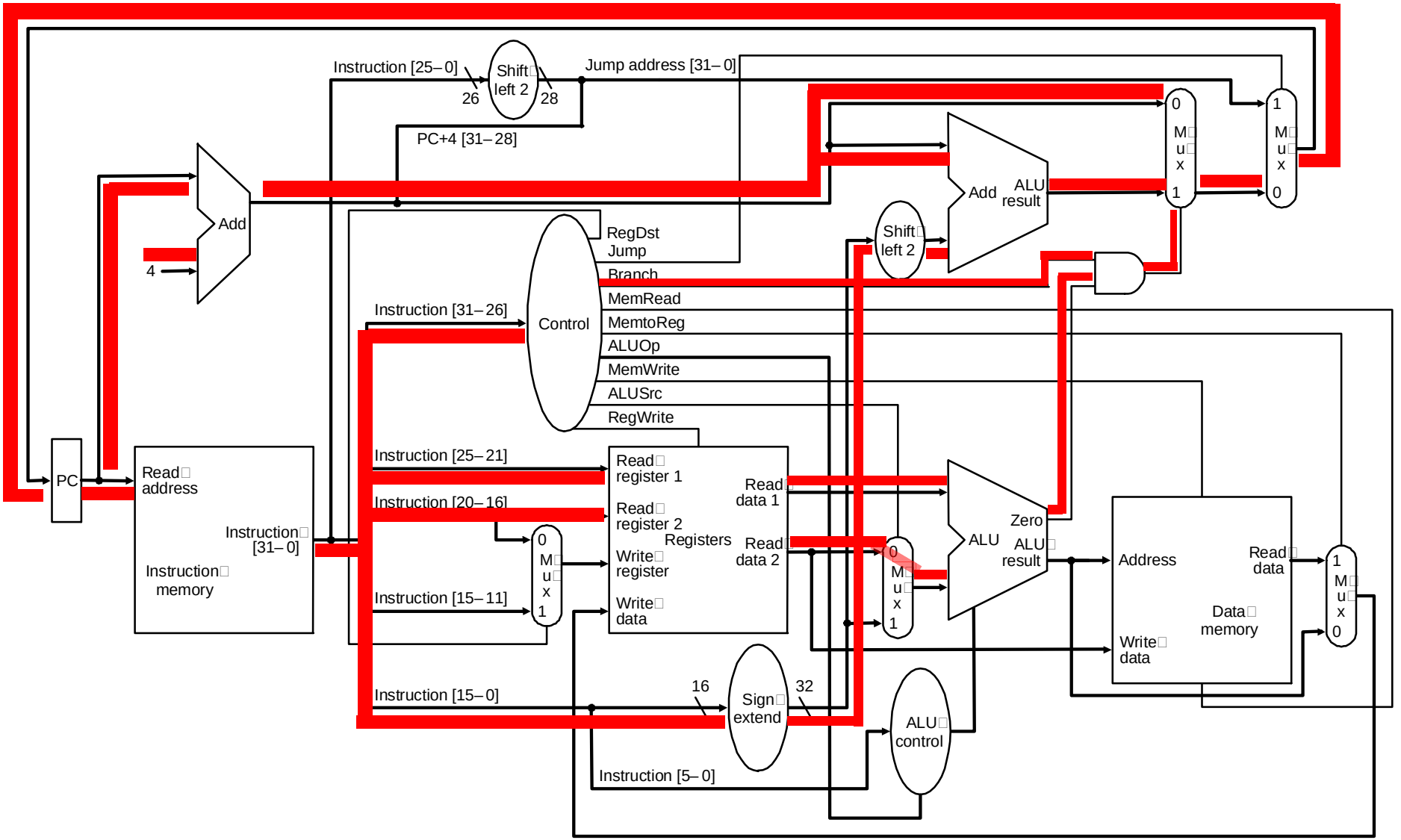
Esempio: istruzione di tipo-R



Esempio: istruzione di load



Esempio: istruzione beq



Esempi di processori che usano controllo a singolo ciclo:

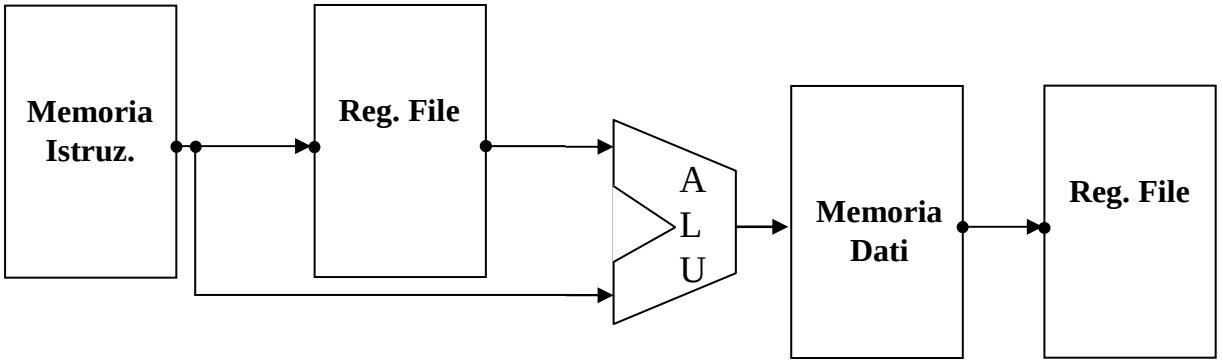
NESSUNO!

Perché?

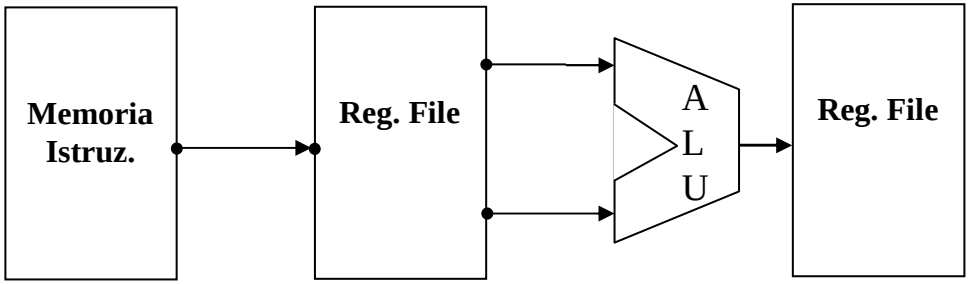
- Periodo di clock: abbastanza lungo per garantire la stabilità dei segnali attraverso il percorso più lungo ⇒ tempo di esecuz. costante per diverse istruz.

  ➡ Istruzioni “più lente” limitano le istruzioni “più veloci”!

Es. lw



Es. Tipo-R



Limitazione aggravata da

- istruzioni che utilizzano decine di unità funzionali in serie, ad esempio comprendenti calcoli in virgola mobile!

⇒ CPI = 1, ma T_{clock} alto, ovvero frequenza di clock molto bassa!

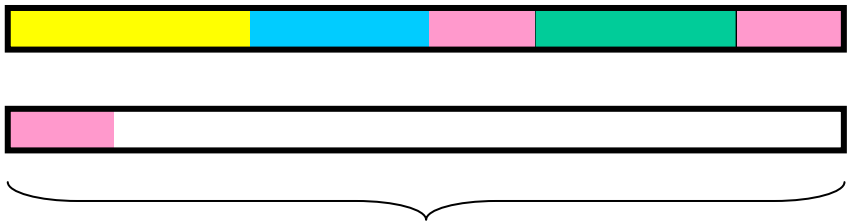
Nel complesso, il tempo di esecuzione di una istruzione è sempre pari al caso peggiore, ovvero a quello dell'istruzione più complessa e lunga

• Altro svantaggio: duplicazione dell'HW ⇒ costi elevati!

Nell'esempio del MIPS, come visto:

- occorrono due memorie diverse [dati e istruzioni]
  [NB: questo va comunque bene perché negli attuali calcolatori si usa memoria cache + pipeline]
- occorrono tre ALU, una per istruzioni aritmetiche | confronto operandi in beq | calcolo indirizzo lw e sw, una per calcolare PC+4 ed una per calcolare indirizzo di salto beq
  [la cosa si complica considerando modalità di indirizzamento complesse, ad esempio quelle con autoincremento, o istruzioni che effettuano operazioni in virgola mobile]

CONTROLLO MULTICICLO: L'IDEA DI BASE

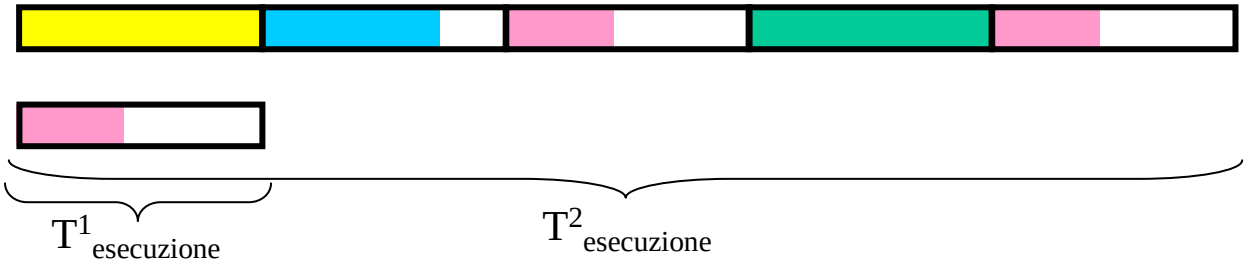


Istruzione lunga

Istruzione potenzialmente breve

$T_{istruzione}$ costante
e pari a T medio di esecuzione per istruzione

Suddividere le istruzioni in "fasi": un ciclo di clock per una fase!



NB: $T^2_{esecuzione}$ (il caso peggiore) può in generale essere maggiore del precedente!
Tuttavia, le istruzioni più lunghe sono anche meno frequenti:
principio "rendere più veloce l'evento più frequente" comporta un guadagno!

Ciascuna fase deve essere sufficientemente breve (T_{clock} breve)

⇒ Bilanciare la suddivisione in fasi,
evitare di mettere in serie più unità funzionali lente.

⇒ Faremo in modo di non mettere in serie più di una delle operazioni:
- accesso in memoria (istruzioni o dati)
- accesso al register file (due letture e una scrittura)
- operazioni della ALU

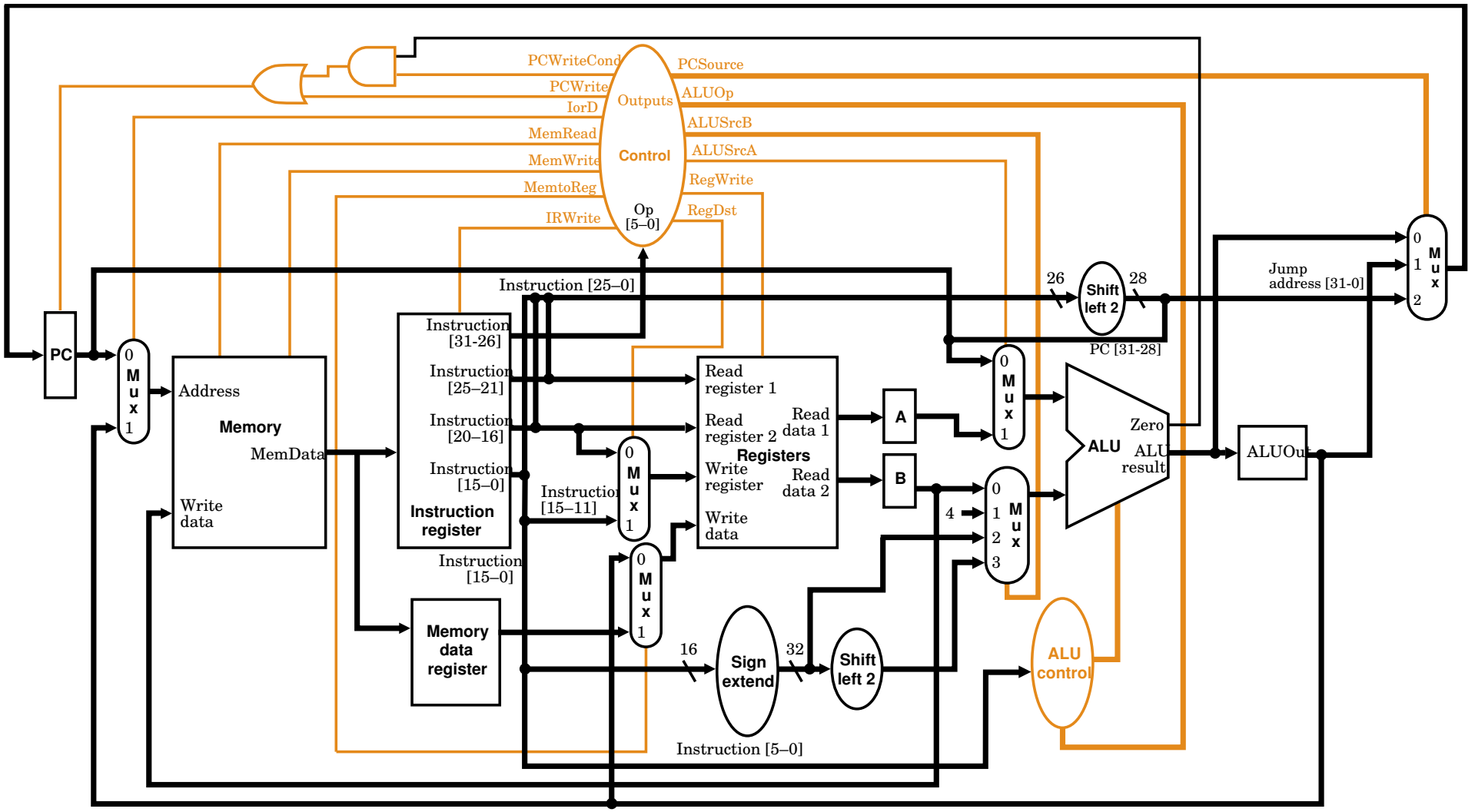
(NB: accesso/scrittura in un registro singolo considerato non oneroso)

⇒ Ciascuna di queste unità [memoria, register file, ALU] necessita di registri temporanei per memorizzarne il risultato.

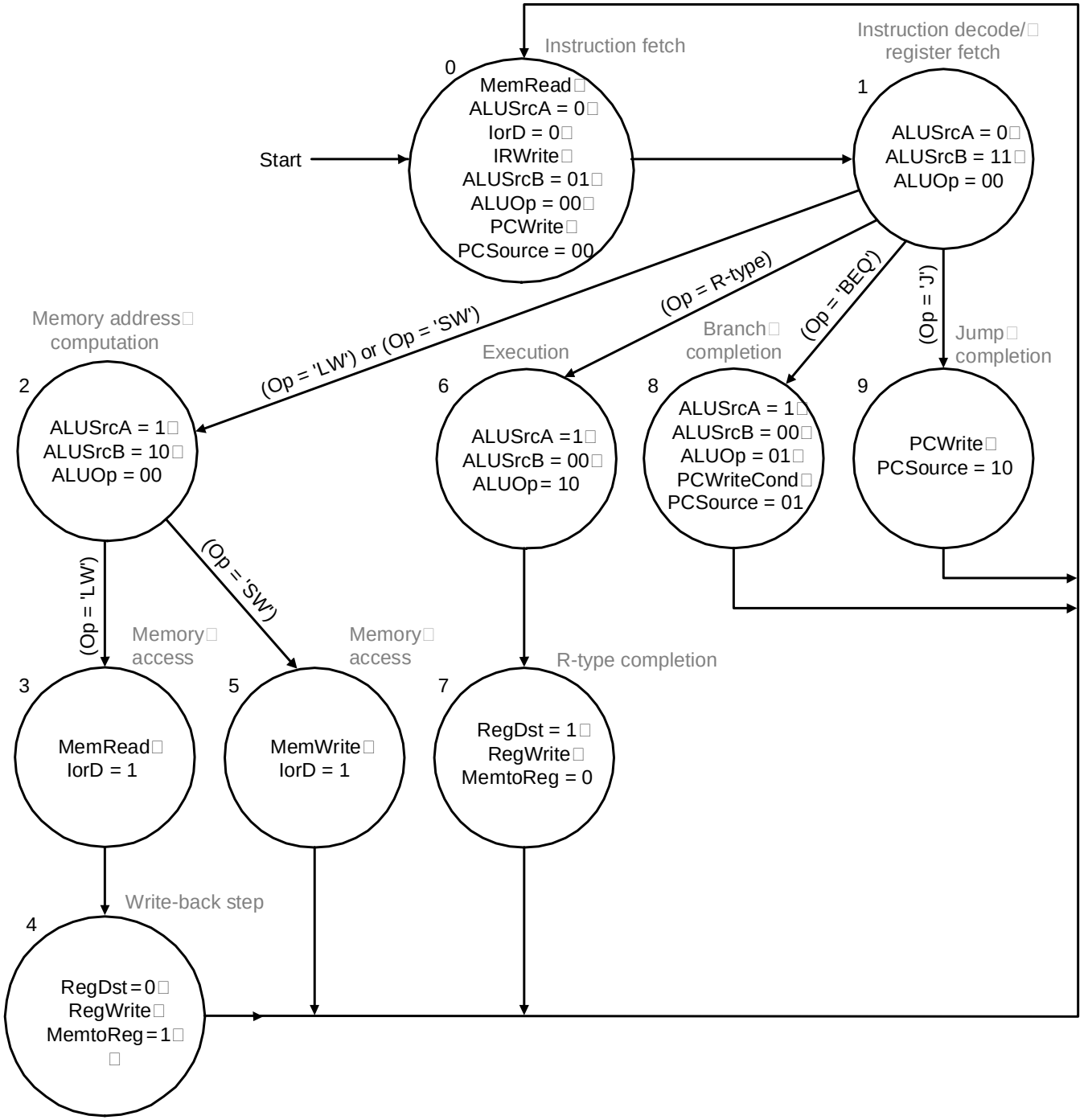
- ***registri temporanei***: salvano dati prodotti in un ciclo di clock e utilizzati dalla stessa istruzione in un ciclo di clock successivo
- ***registri visibili al prog***: dati utilizzati da istruzioni successive

Il MIPS

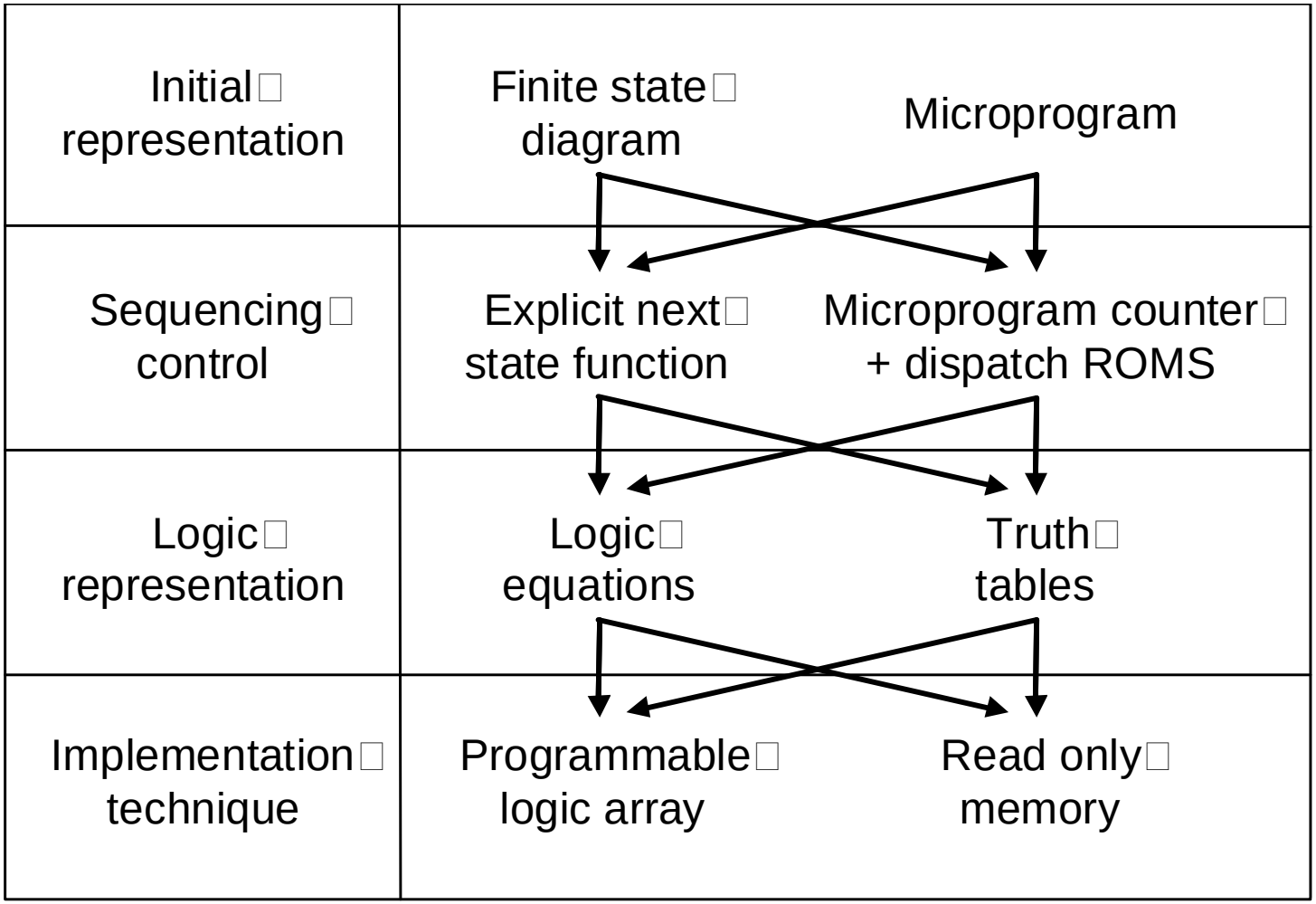
(Vedi Lezione CalcA “CPU multiciclo”, lucidi 7—20, per la specifica progressiva)



NB: l’unità di controllo si può specificare e realizzare in vari modi…


Start
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Instruction fetch
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
Instruction decode/
register fetch
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Memory address
computation
6
ALUSrcA =1
ALUSrcB = 00
ALUOp= 10
Execution
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
Branch
completion
9
PCWrite
PCSource = 10
Jump
completion
(Op = 'LW')
(Op = 'SW')
3
MemRead
IorD = 1
Memory
access
5
MemWrite
IorD = 1
Memory
access
7
RegDst = 1
RegWrite
MemtoReg = 0
R-type completion
4
RegDst = 0
RegWrite
MemtoReg = 1
Write-back step

Controllo di un processore-multiciclo: Riepilogo specifica e realizzazione



Calcolo CPI e Prestazioni nei sistemi a singolo ciclo e multiciclo

1) Calcolo prestazioni nei sistemi a singolo ciclo

$CPI = 1$

$T_{clock} = \max\{T_a + \ldots T_k\}$

ovvero la serie più lenta di "operazioni atomiche" [cammino critico]

$T_{esecuzione} = \#istruzioni * CPI * T_{clock} = \#istruzioni * T_{clock}$

2) Calcolo CPI e prestazioni nei sistemi multi-ciclo

Dato un certo carico di lavoro con frequenze relative delle istruzioni $f_1, \ldots, f_n$

$CPI = f_1*CPI_1 + f_2*CPI_2 + \ldots + f_n*CPI_n$

$T_{clock} = \max\{T_1, \ldots, T_m\}$ [operazioni atomiche eseguite <u>in un ciclo di clock</u>]

$T_{esecuzione} = \#istruzioni * CPI * T_{clock}$

3) Confronto di prestazioni tra sistemi diversi [su un carico/prog. determinato]

$$\frac{T^1_{esecuzione}}{T^2_{esecuzione}} = \frac{CPI^1 * T^1_{clock}}{CPI^2 * T^2_{clock}}$$