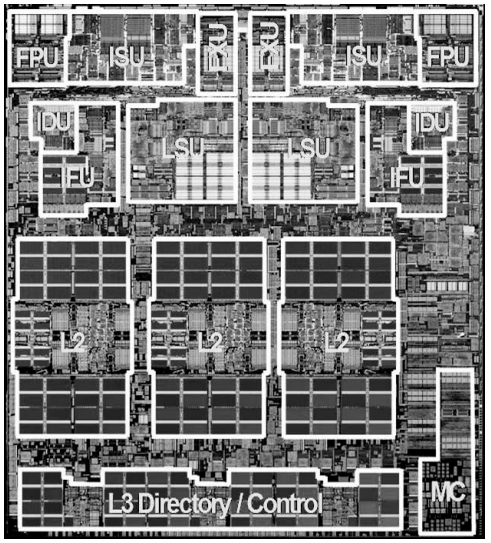
Università degli Studi di Cassino



Corso di Calcolatori Elettronici II

Data path multiciclo

Anno Accademico 2007/2008

Francesco Tortorella

Problemi dell'implementazione singolo ciclo

Arithmetic & Logical

Load

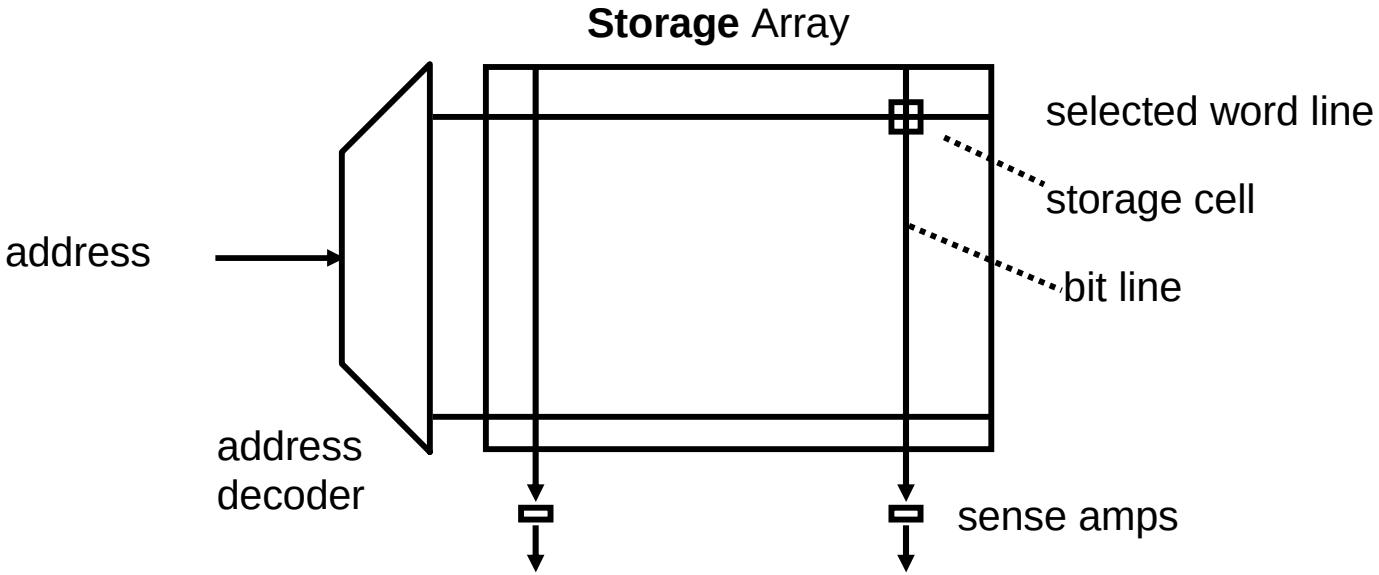
Store

Branch

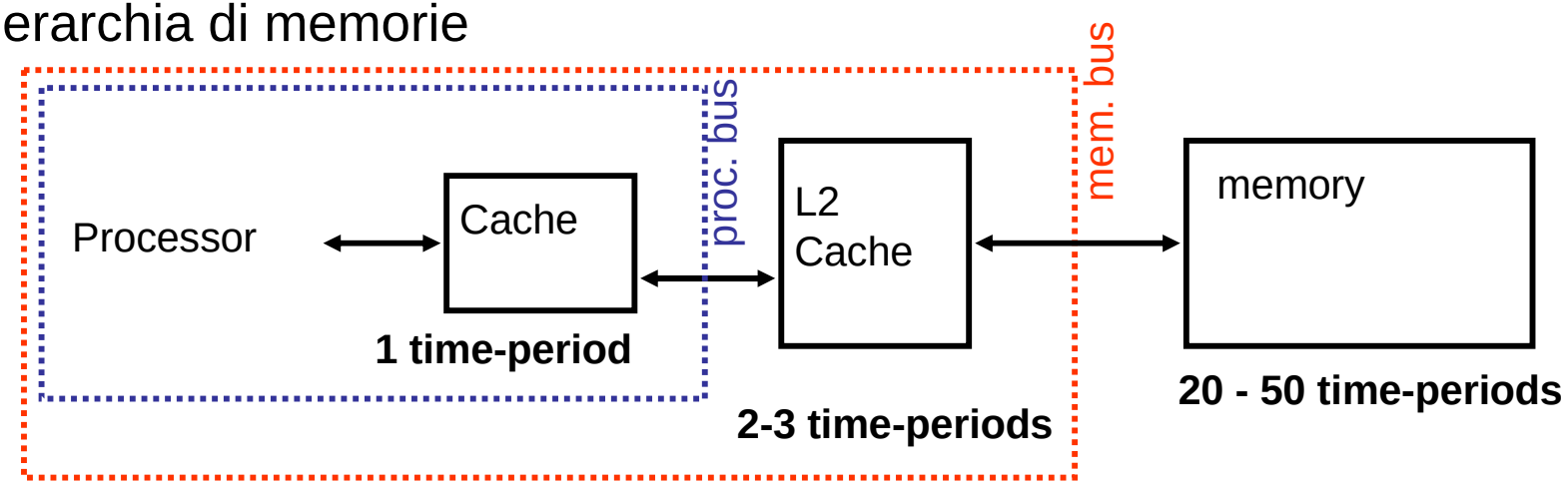
- Tempo di ciclo lungo
  - Tutte le istruzioni hanno un tempo di esecuzione uguale all'istruzione più lenta
  - Memoria reale più lenta della memoria ideale
- Risorse duplicate

Tempo di accesso delle memorie

- Limite fisico: le memorie veloci sono piccole (e viceversa)

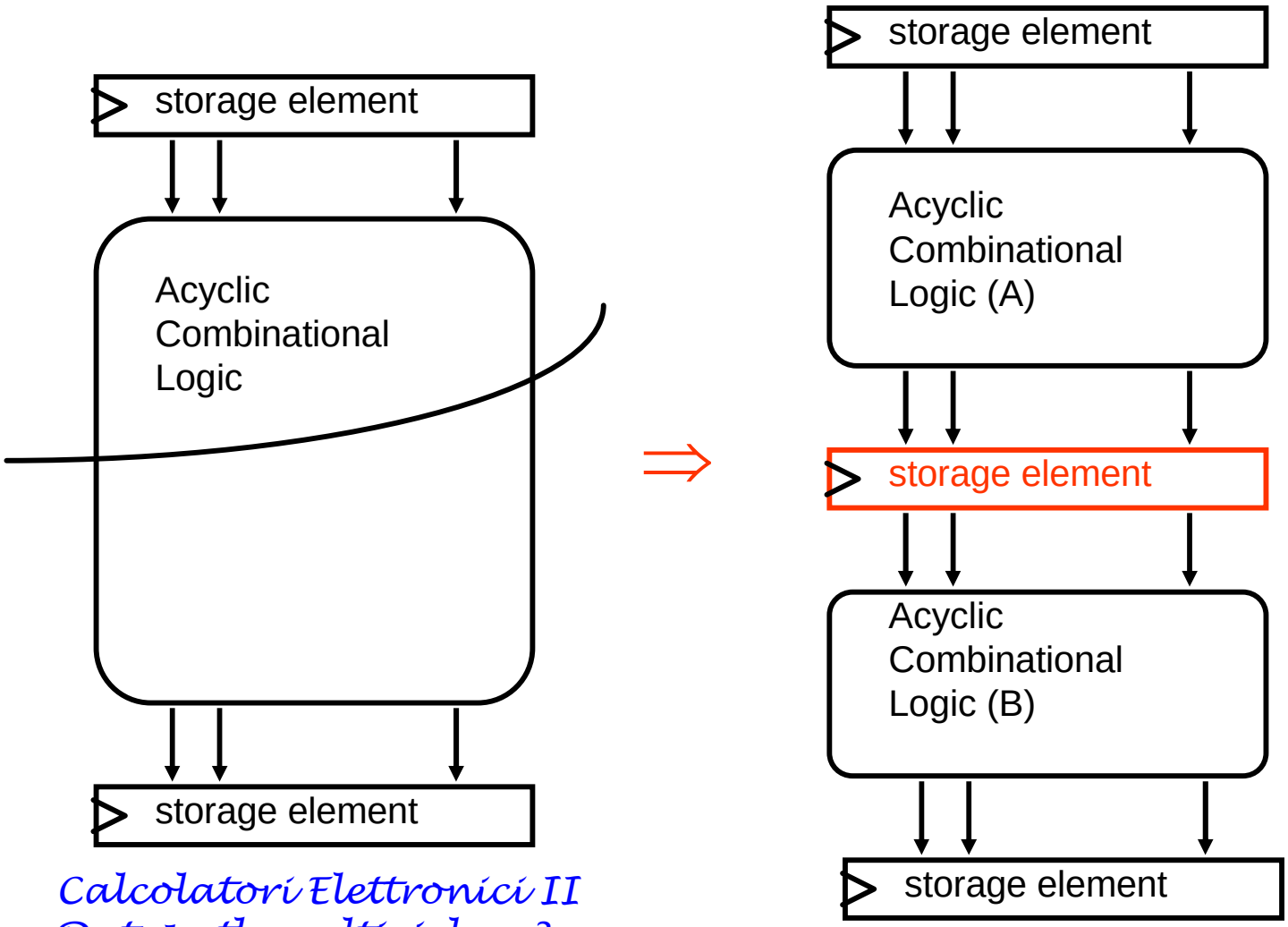


- => Gerarchia di memorie



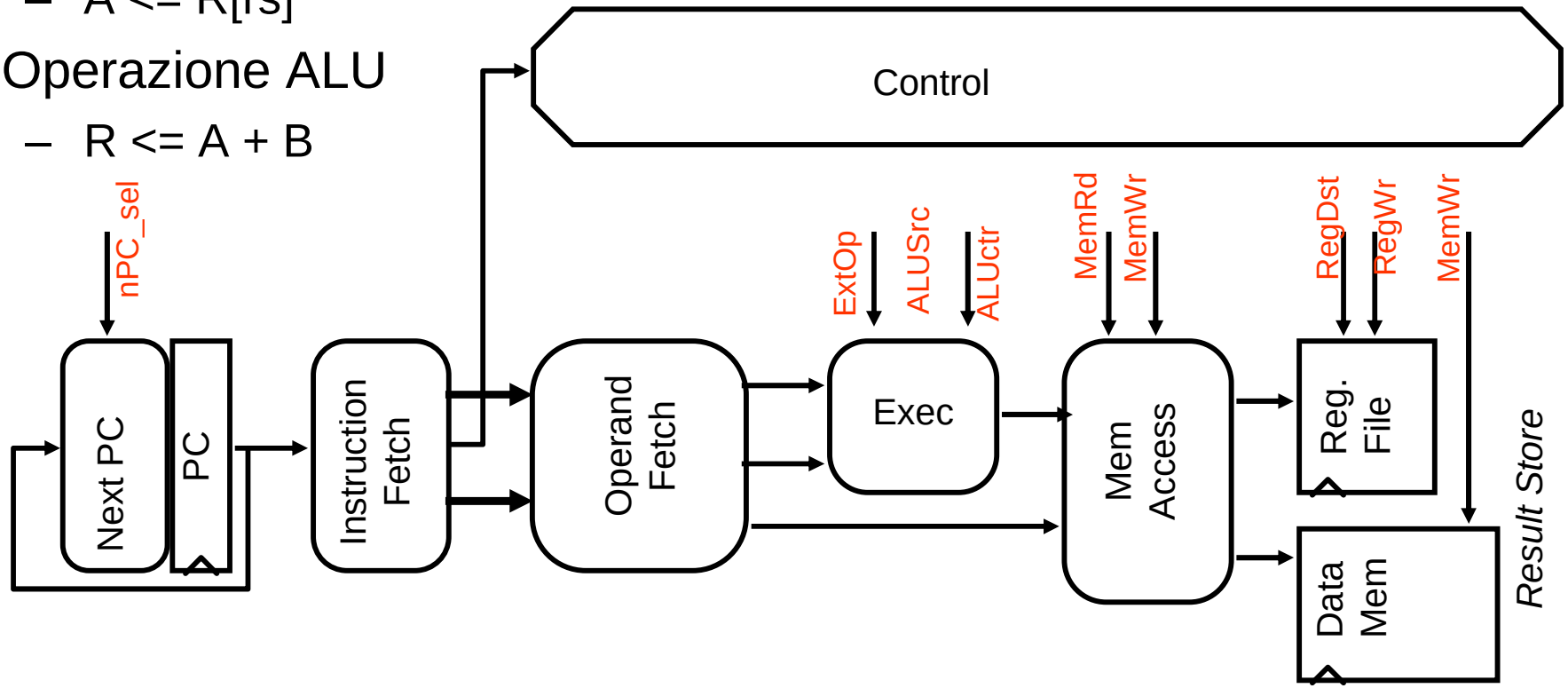
Riduzione del tempo di ciclo

- Si partiziona la parte combinatoria in più parti combinatorie più piccole, separate da registri.
- La stessa operazione viene realizzata in due cicli di breve durata invece che in uno solo molto lungo.



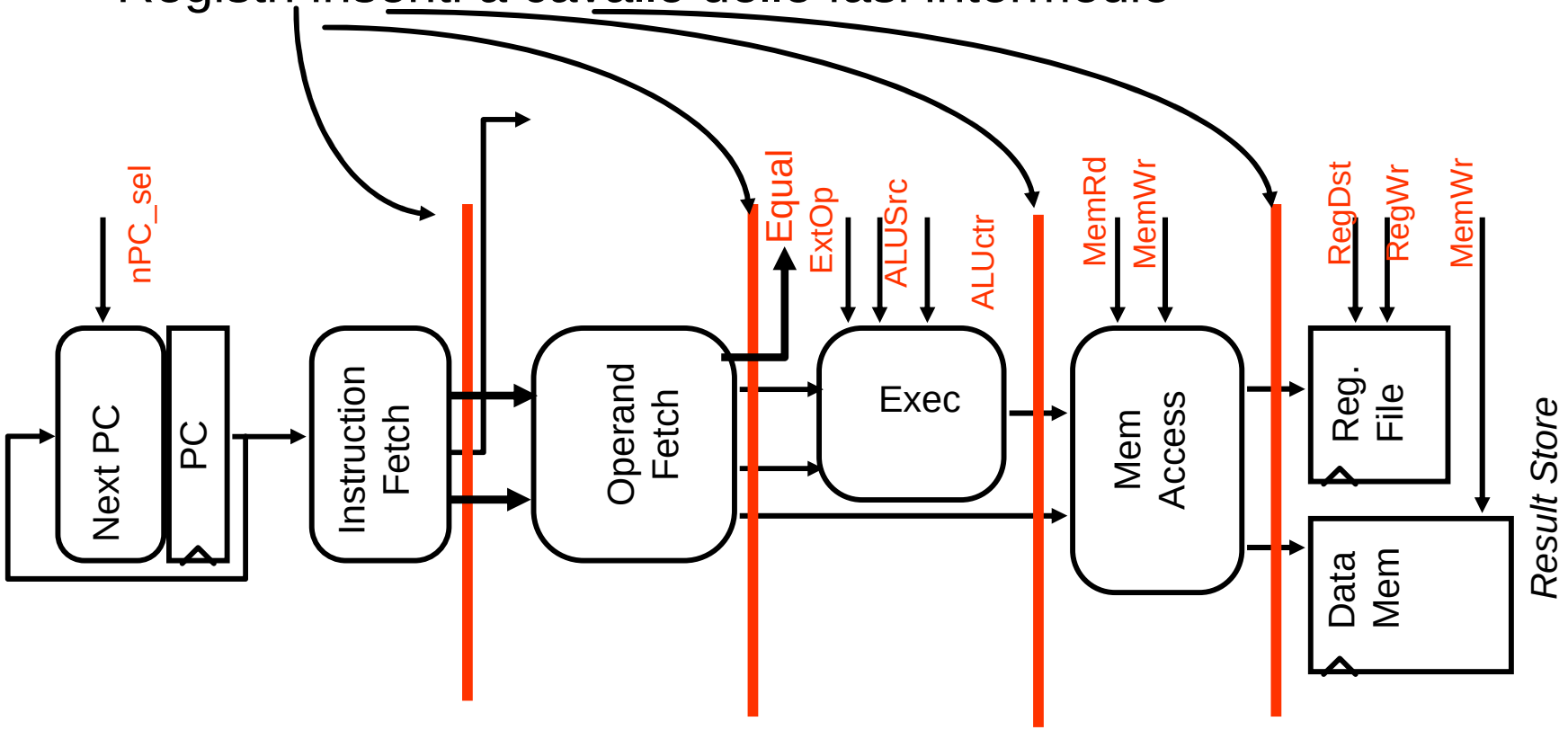
Vincoli sul tempo di ciclo

- Logica per calcolare l'indirizzo successivo
  - PC <= branch ? PC + offset : PC + 4
- Instruction Fetch
  - InstructionReg <= Mem[PC]
- Accesso ai registri
  - A <= R[rs]
- Operazione ALU
  - R <= A + B



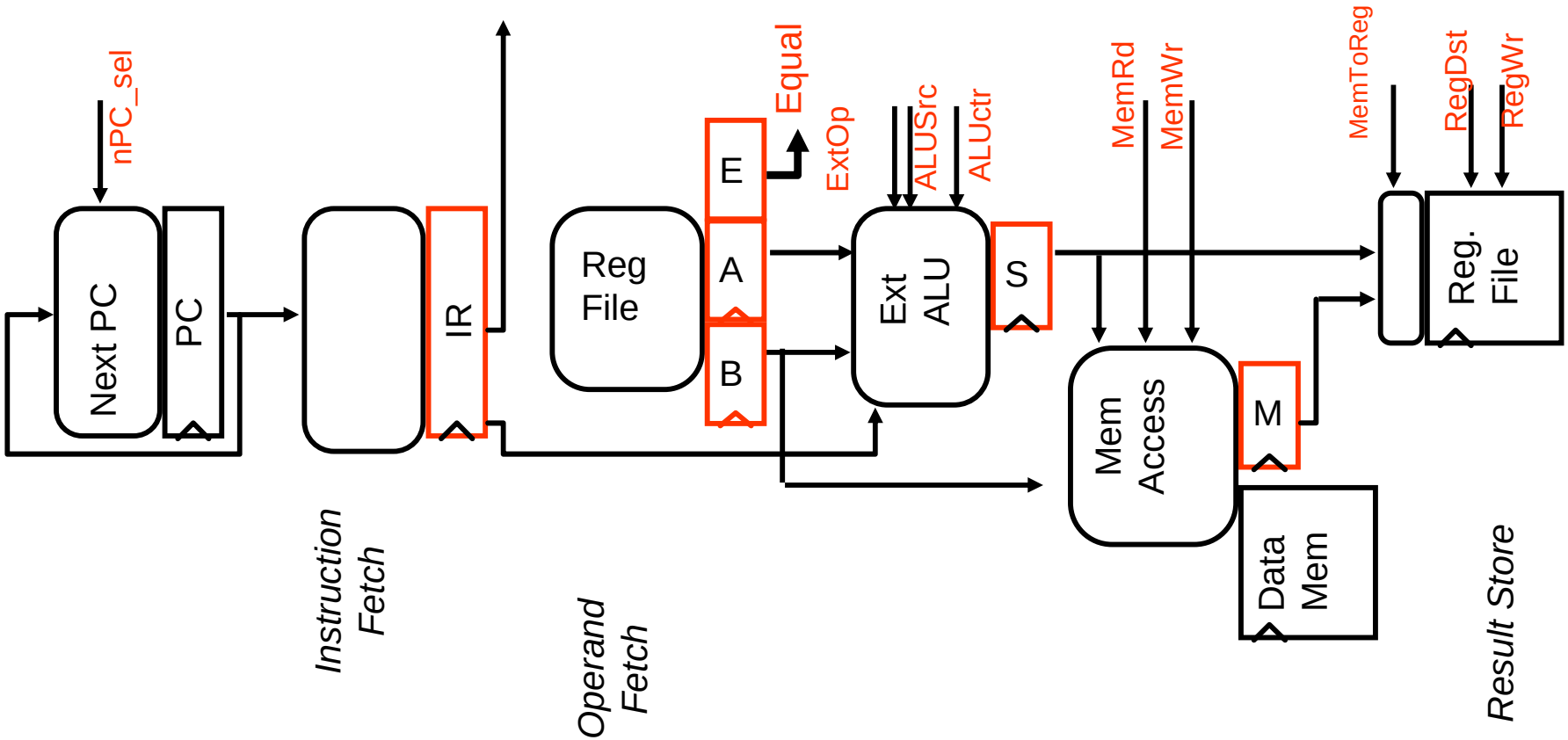
Partizionamento del datapath

- Registri inseriti a cavallo delle fasi intermedie



1. Registri inseriti in modo da bilanciare la lunghezza del ciclo di clock (parti combinatorie tra i registri con lo stesso ritardo)
2. Registri inseriti per conservare le informazioni necessarie in passi successivi dell'esecuzione dell'istruzione.

Datapath multiciclo (visione logica)

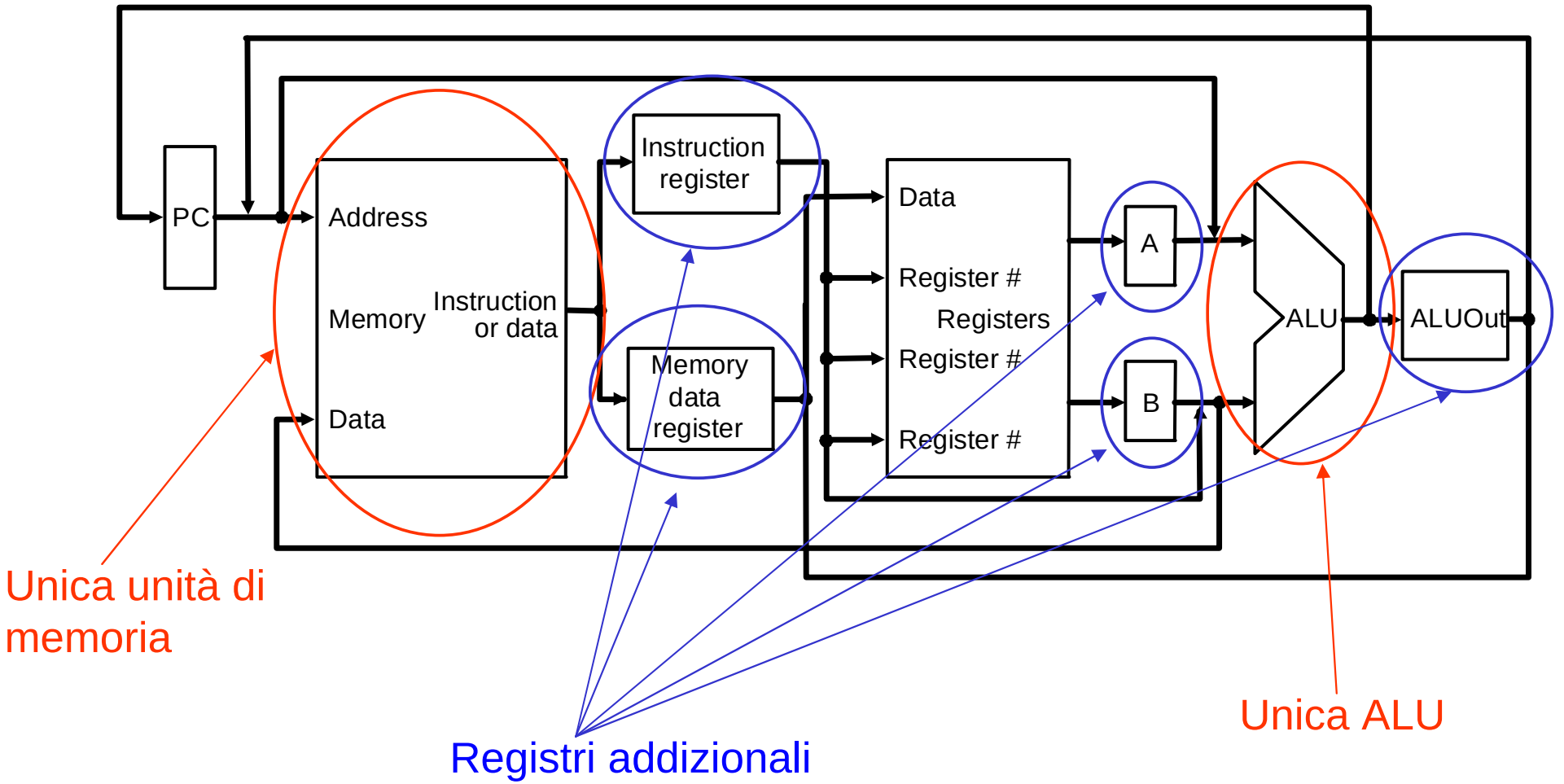


- Critical Path ?

Datapath multiciclo

- Nell'implementazione che stiamo considerando, assumiamo che in un ciclo di clock si possa realizzare al più una tra le seguenti operazioni:
  - Accesso in memoria (lett. istruzione o lett./scritt. dati)
  - Accesso al register file (2 letture o 1 scrittura)
  - Operazione ALU
- Al termine di ogni ciclo di clock, i registri intermedi contengono i dati necessari alle fasi da realizzare nei cicli seguenti.

Implementazione del datapath multiciclo (vista ad alto livello)

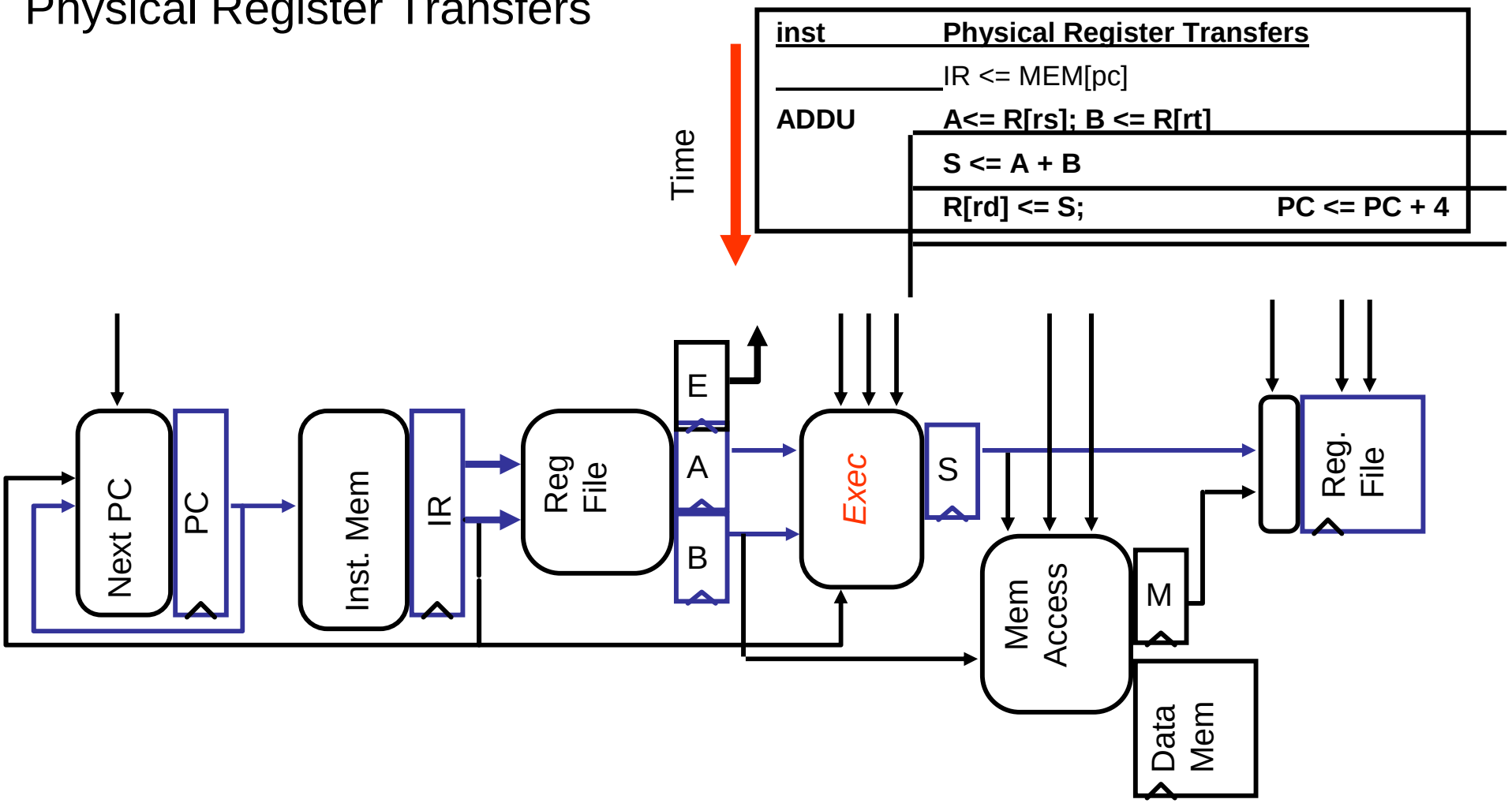


Istruzione R-rtype (add, sub, . . .)

- Logical Register Transfer

inst	Logical Register Transfers
ADDU	R[rd] <= R[rs] + R[rt]; PC <= PC + 4

- Physical Register Transfers

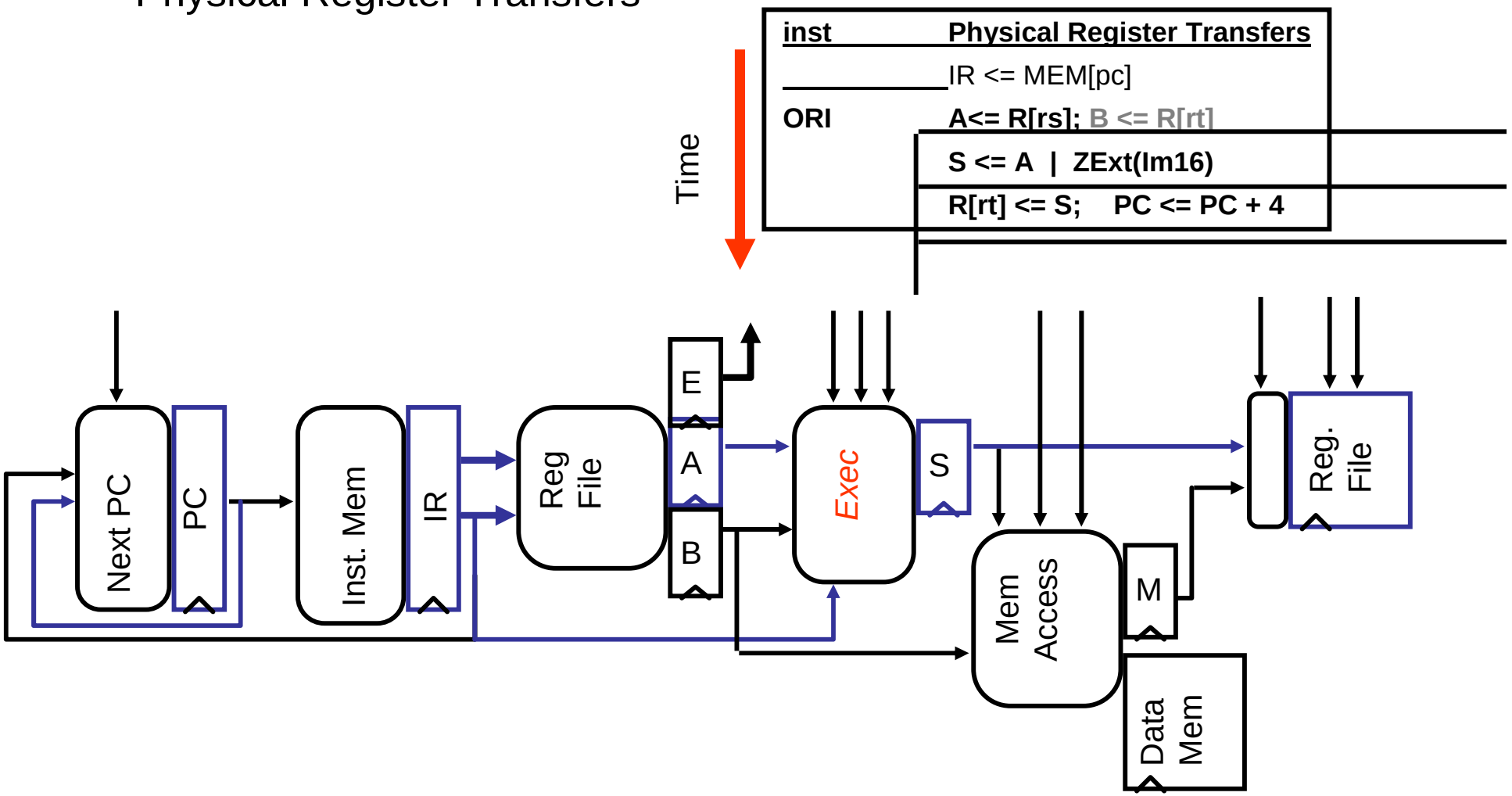


Istruzione Logica con immediato

- Logical Register Transfer

inst	Logical Register Transfers
ORI	R[rt] <= R[rs] | ZExt(Im16); PC <= PC + 4

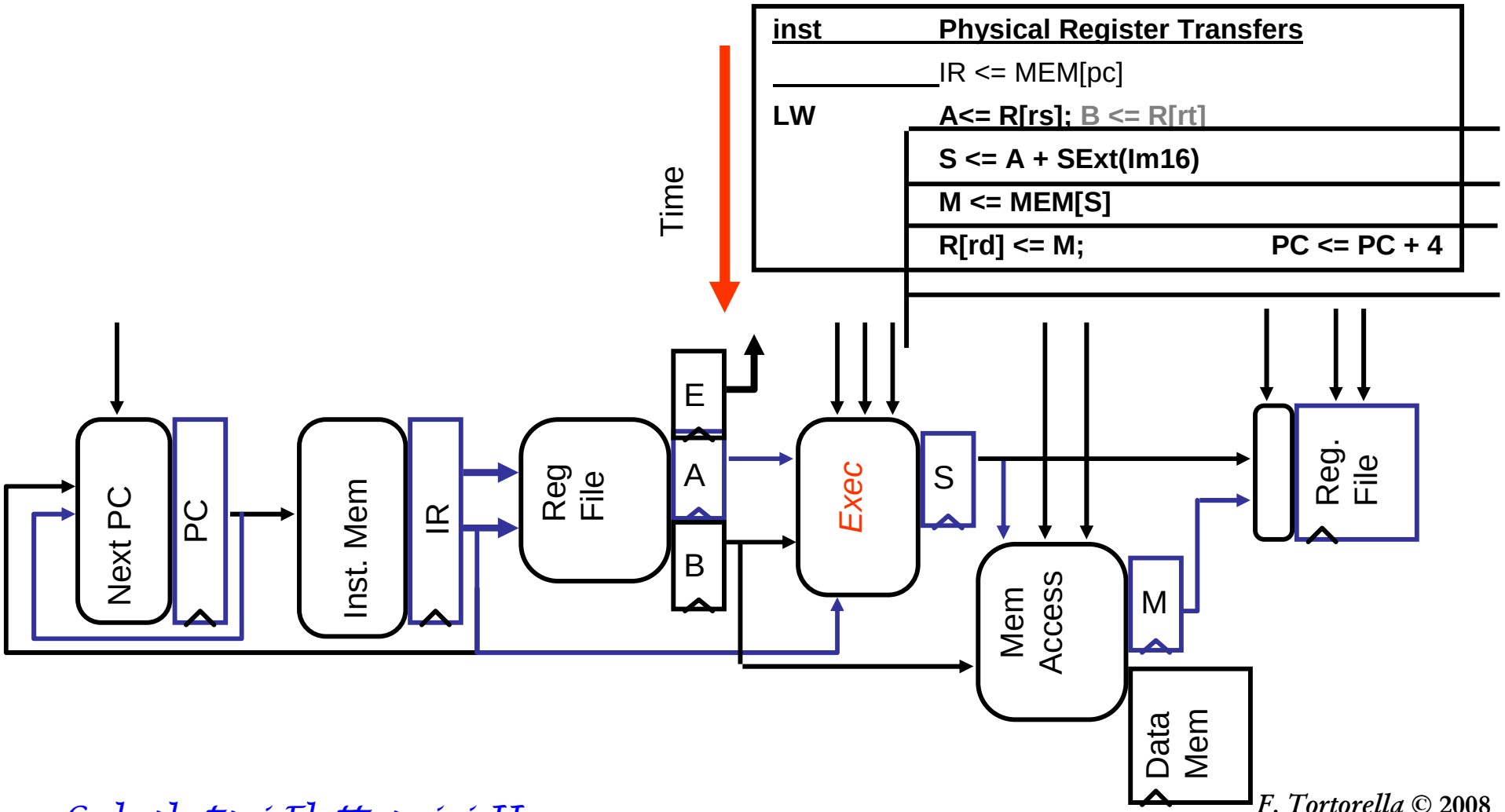
- Physical Register Transfers



Load

- Logical Register Transfer
- Physical Register Transfers

inst	Logical Register Transfers
LW	R[rt] <= MEM[R[rs] + SExt(Im16)];
	PC <= PC + 4

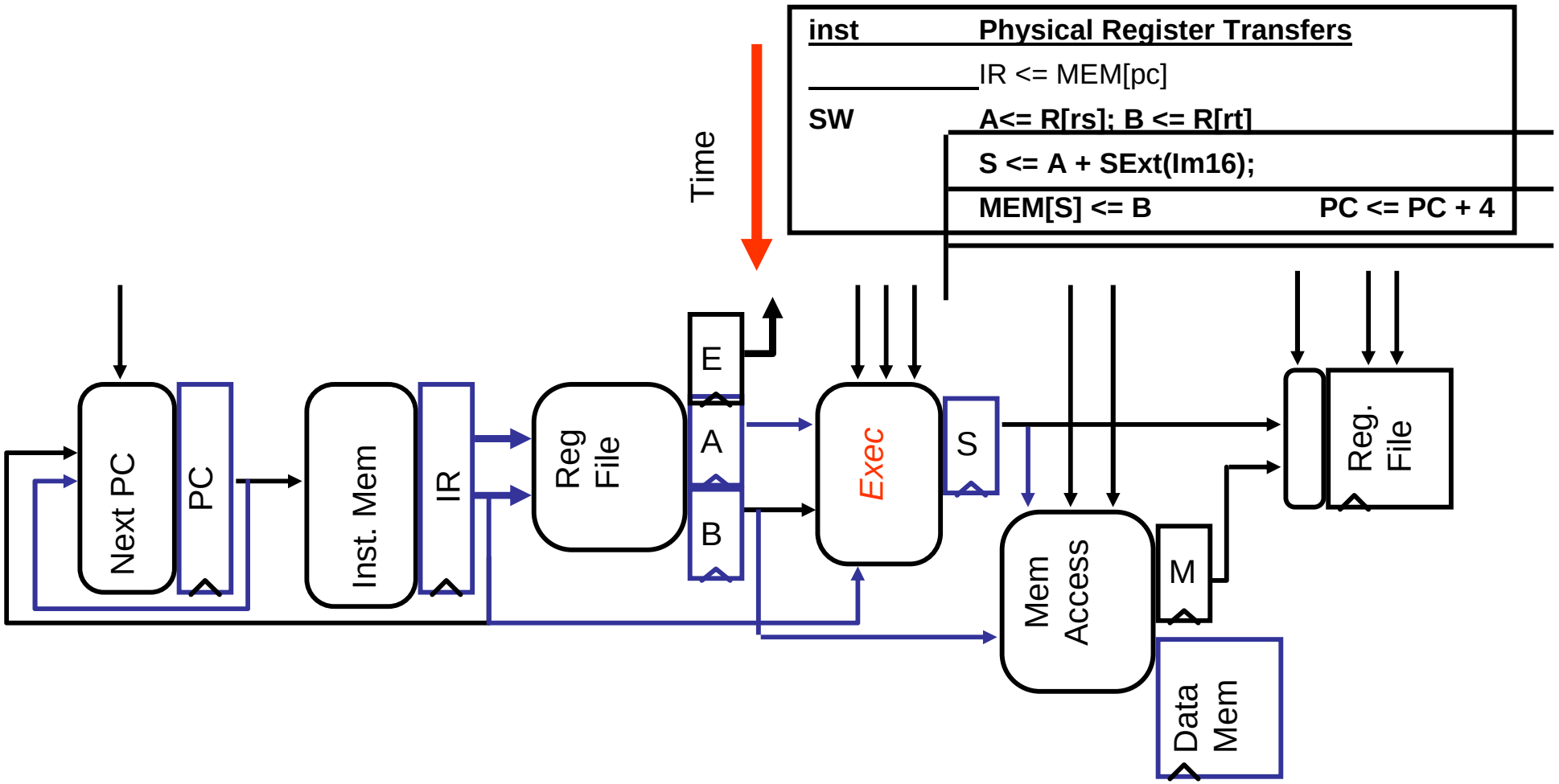


Store

- Logical Register Transfer

inst	Logical Register Transfers
SW	MEM[R[rs] + SExt(Im16)] <= R[rt];
	PC <= PC + 4

- Physical Register Transfers



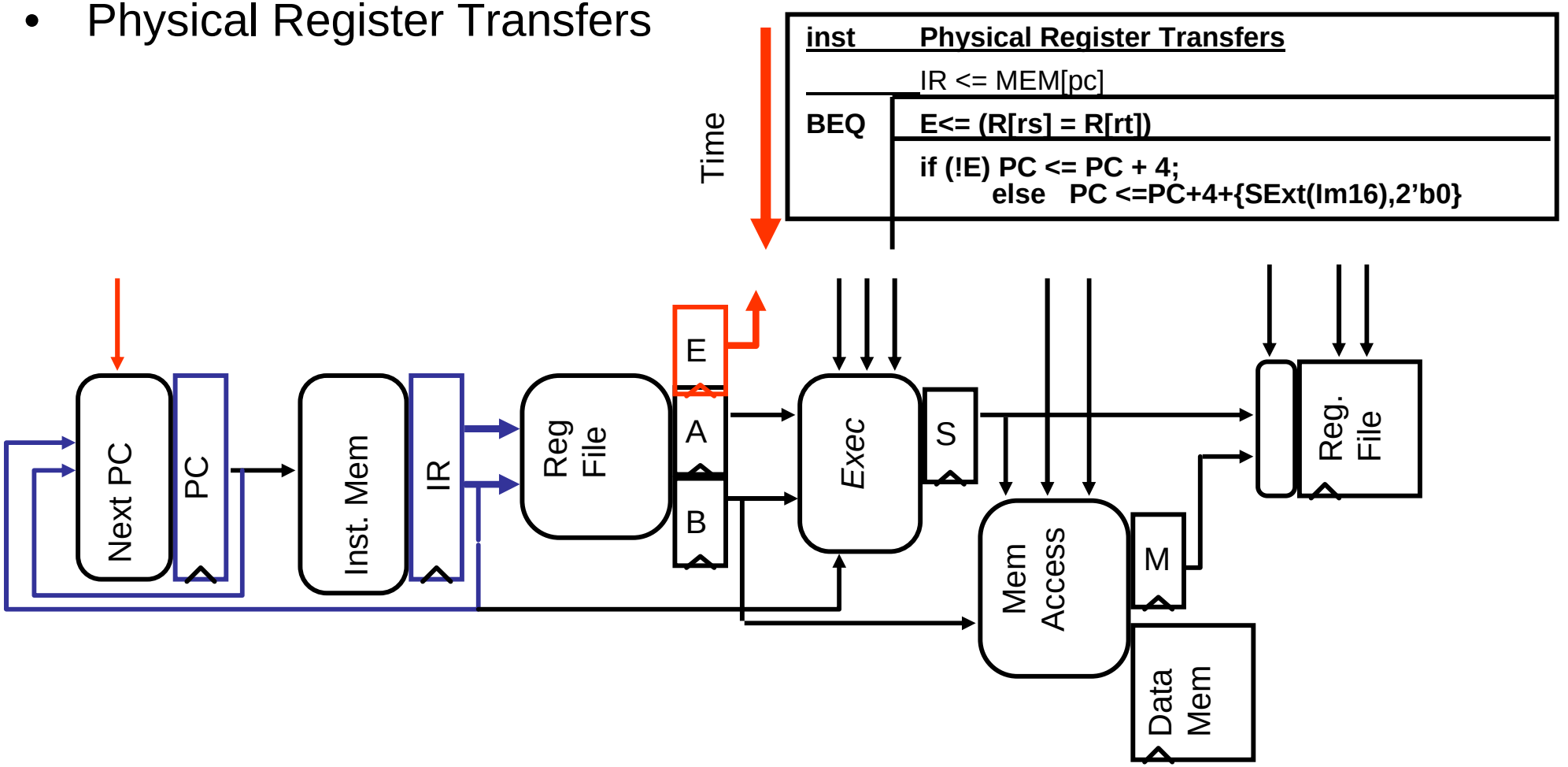
Branch

- Logical Register Transfer

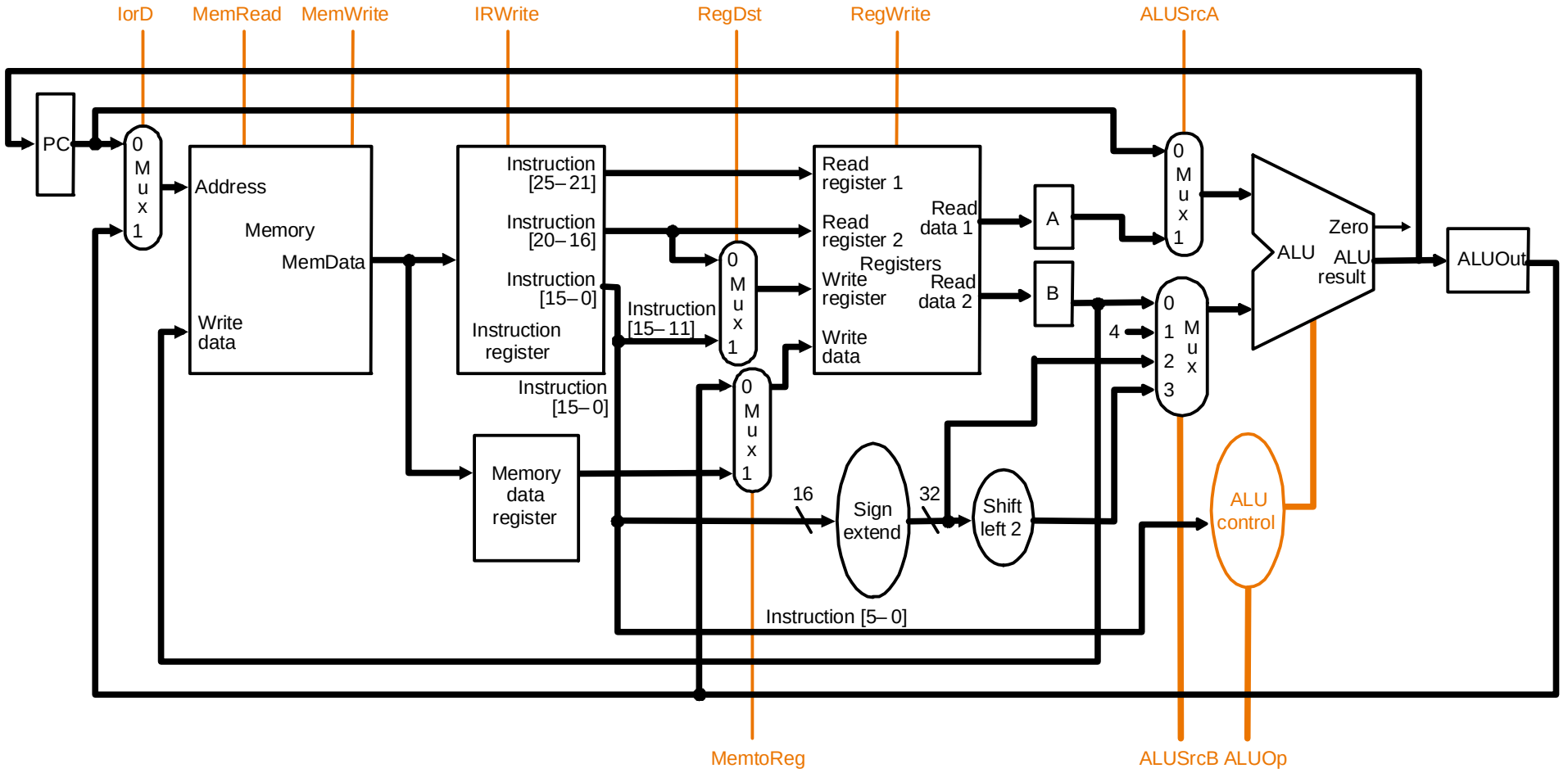
inst	Logical Register Transfers
BEQ	if R[rs] == R[rt]
	then PC <= PC + 4+{SExt(Im16), 2'b00}
	else PC <= PC + 4

- Physical Register Transfers

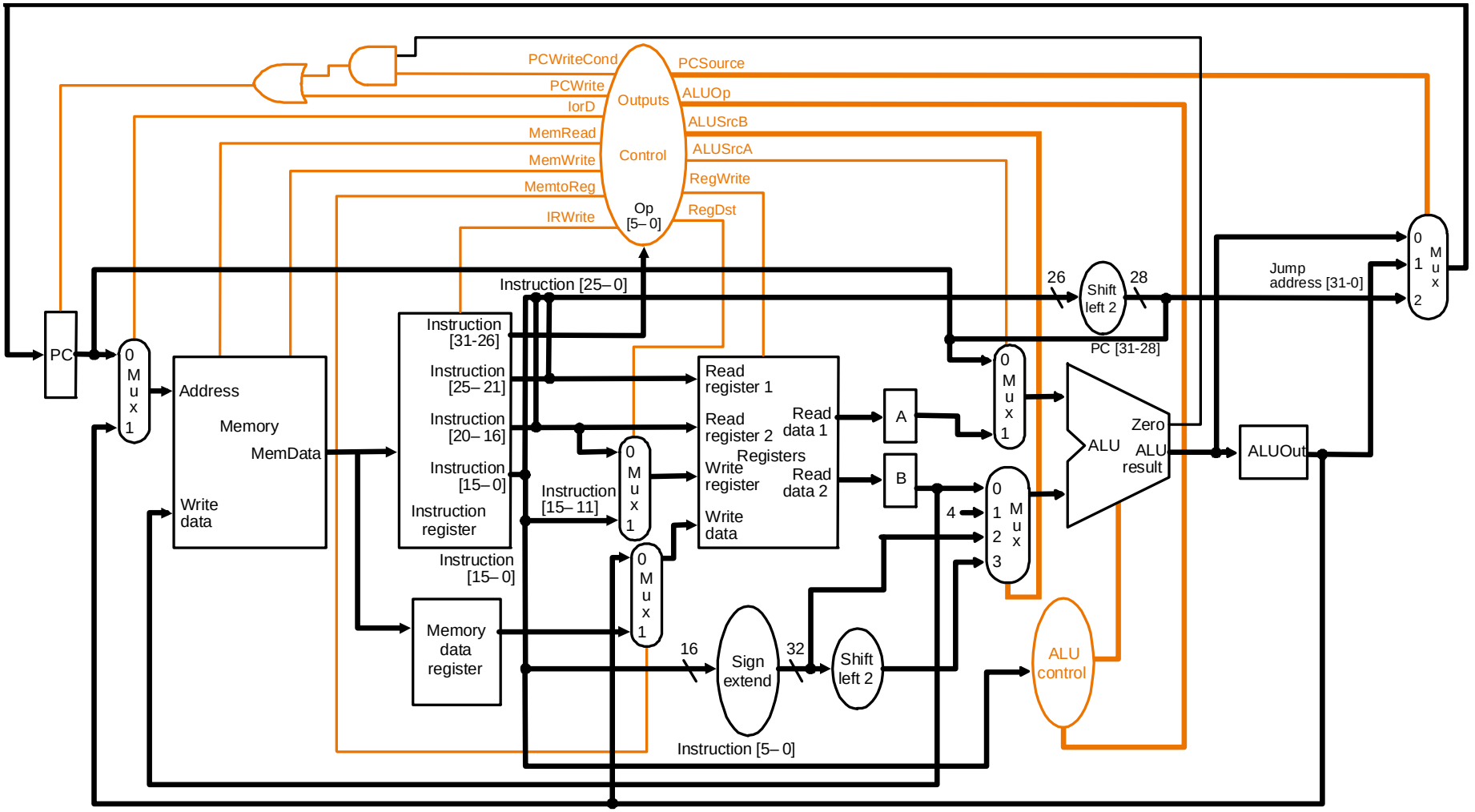
inst	Physical Register Transfers
	IR <= MEM[pc]
BEQ	E<= (R[rs] = R[rt])
	if (!E) PC <= PC + 4; else PC <=PC+4+{SExt(Im16),2'b0}



Implementazione del datapath multiciclo



Implementazione del datapath multiciclo



Esecuzione dell'istruzione in 5 passi

1. Instruction fetch
2. Instruction decode and register fetch
3. Execution / Memory address computation / Branch completion / Jump completion
4. Memory access / R-type instruction completion
5. Memory read completion

1. Instruction fetch

- `IR <= Memory[PC]`
- `PC <= PC+4`

- Quali segnali attivi ?
- Perché si incrementa il PC ?

2. Instruction decode and register fetch

- `A <= Reg[IR[25:21]]`
- `B <= Reg[IR[20:16]]`
- `ALUOut <= PC+(signext(IR[15:0])<<2)`

- Non è ancora nota l'istruzione da eseguire
- Si preparano possibili operandi (rs->A, rt->B) e si calcola l'offset dell'indirizzo destinazione di una eventuale istruzione di branch

3. Execution / Memory address computation / Branch completion / Jump completion

- Le operazioni eseguite da questo passo in poi dipendono dalla particolare istruzione
- Load/Store
  - **`ALUOut <= A+signext(IR[15:0])`**
- Istruzione Logico-Aritmetica (R-type)
  - **`ALUOut <= A op B`**
- Branch (completa)
  - **`If (A==B) PC <= ALUOut`**
  - La condizione è fornita dal flag Zero dell'ALU
- Jump (completa)
  - **`PC <= { PC[31:28] , {IR[25:0] , '00'} }`**

4. Memory access / R-type instruction completion

- Load
  - **`MDR <= Memory[ALUOut]`**
- Store (completa)
  - **`Memory[ALUOut] <= B`**
- Istruzione Logico-Aritmetica (R-type) (completa)
  - **`Reg[IR[15:11]] <= ALUOut`**

5. Memory read completion

- Load
  - **`Reg[IR[20:16]] <= MDR`**

Passi realizzati per eseguire una classe di istruzioni. Vista complessiva

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR = Memory[PC] PC = PC + 4			
Instruction decode/register fetch	A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A ==B) then PC = ALUOut	PC = PC [31-28] II (IR[25-0]<<2)
Memory access or R-type completion	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
Memory read completion		Load: Reg[IR[20-16]] = MDR		

Valutazione del data path multiciclo

- Ricordiamo la distribuzione stimata per le istruzioni:
  - Load: 25 %
  - Store: 10 %
  - Istruzioni logico-aritmetiche: 45 %
  - Branch: 15 %
  - Jump: 5 %

Valutazione del data path multiciclo

Tipo di istruzione	Numero di cicli
Logico aritmetiche	4
Load	5
Store	4
Branch	3
Jump	3

Valutiamo il numero medio di cicli per istruzione (CPI):

CPI= .45*4+.25*5+.1*4+.15*3+.05*3 =4.05

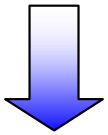
Se avessimo adottato lo stesso numero di cicli avremmo avuto un CPI=5.

Definizione del controllo

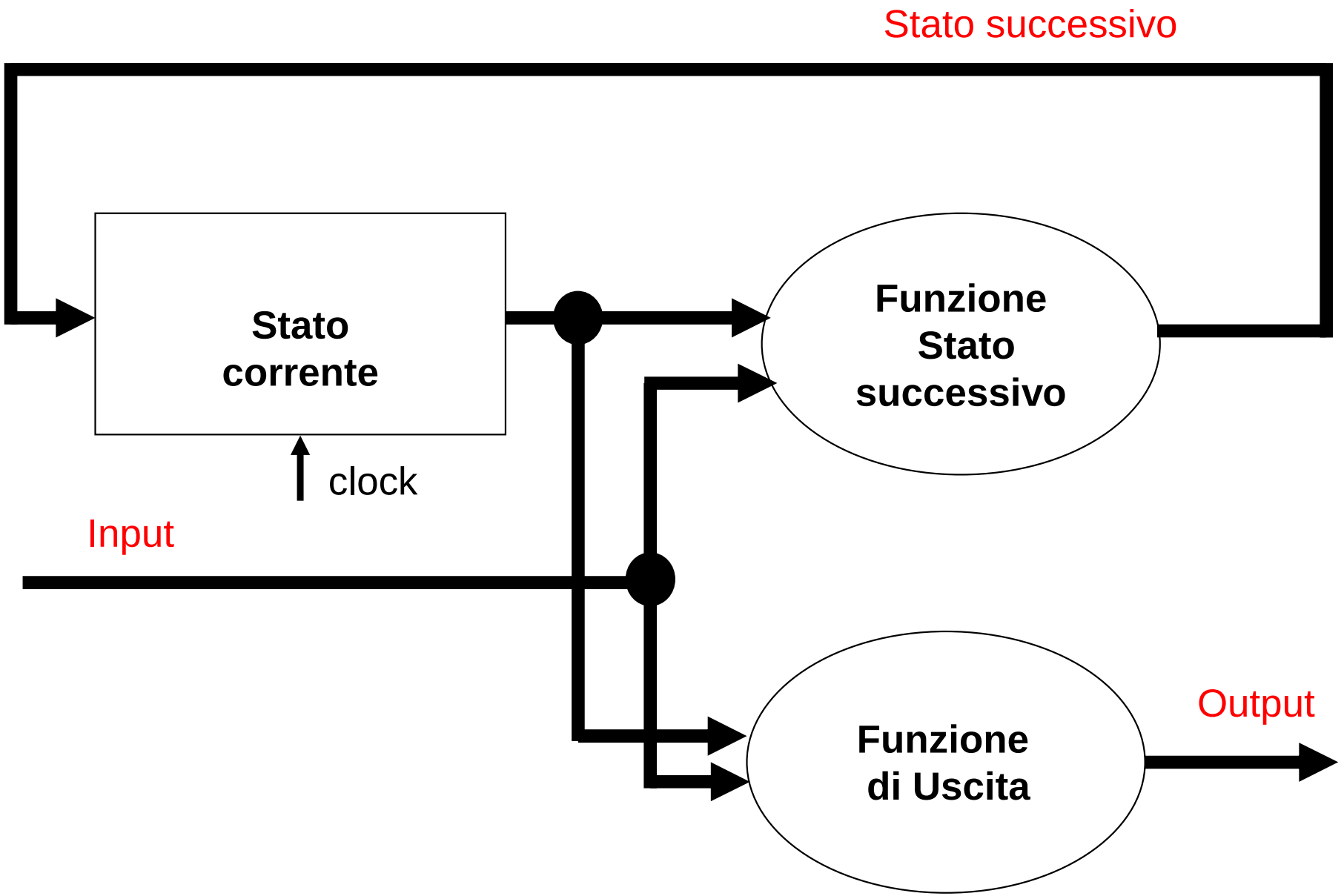
- I valori dei segnali di controllo (abilitazioni) deipendono da:
  - Istruzione in esecuzione
  - Passo corrente
- Si usano le informazioni raccolte per specificare una macchina a stati finiti (MSF) che sia capace di generare le sequenze opportune di abilitazioni.

Reti sequenziali

- una rete sequenziale è un circuito logico avente n ingressi (x_1,x_2,…,x_n) ed m uscite (y_1,y_2, …,y_m), ciascuno dei quali assume valori binari (0/1).
- ad ogni istante, il valore delle uscite dipende sia dagli ingressi presenti, sia dalla sequenza degli ingressi precedenti.
- in ogni istante, si può identificare uno *stato S* in cui la rete si trova, in funzione della sequenza degli ingressi precedenti.
- le uscite possono quindi essere definite come funzioni degli ingressi correnti e dello stato corrente.
- ugualmente, lo stato successivo può essere determinato in funzione dello stato corrente e degli ingressi correnti.



Modello di macchina a stati finiti



Esempi di macchine sequenziali

- **Registri (RS, D, JK)**
- **Registri a scorrimento**
- **Contatori**

Come si definisce lo stato ?

Quanti stati diversi può assumere la macchina ?

Come si definisce la funzione stato successivo ?

Come si definisce la funzione di Uscita ?

Registro JK

Stato: valore di Q

Num. di stati: 2

F. di stato successivo:

$J=0$ e $K=0 \Rightarrow Q'=Q$

$J=1$ e $K=0 \Rightarrow Q'=1$

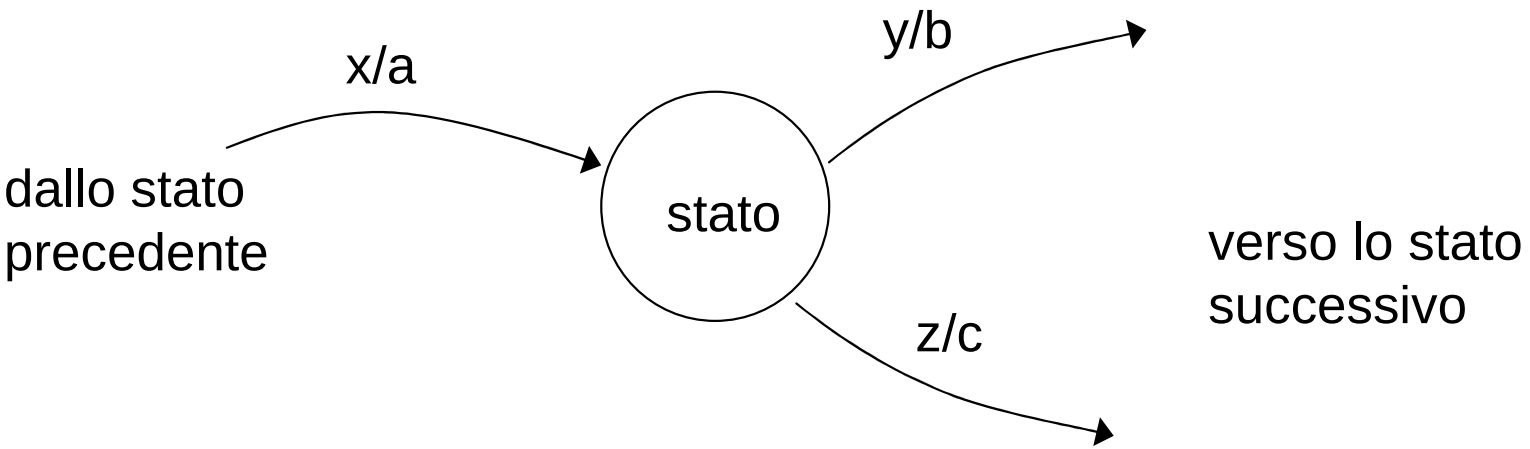
$J=0$ e $K=1 \Rightarrow Q'=0$

$J=1$ e $K=1 \Rightarrow Q'=\overline{Q}$

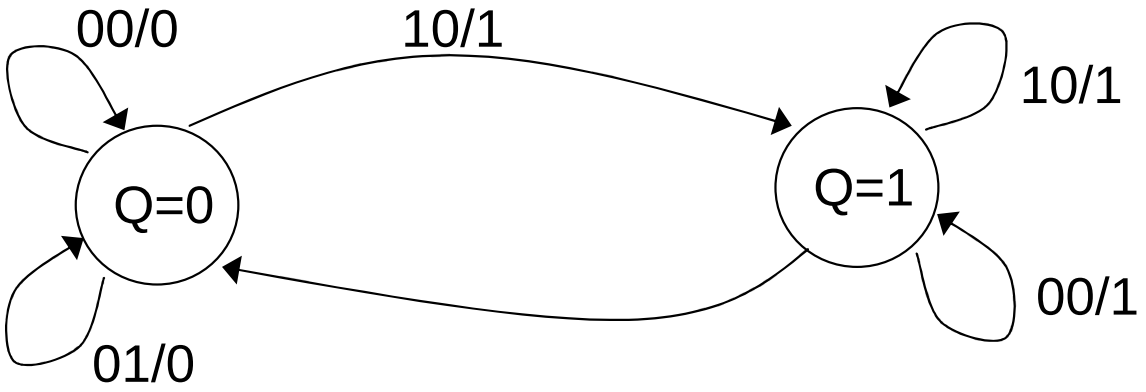
F. di uscita: valore di Q

Diagramma di stato

Strumento per descrivere graficamente una macchina sequenziale.

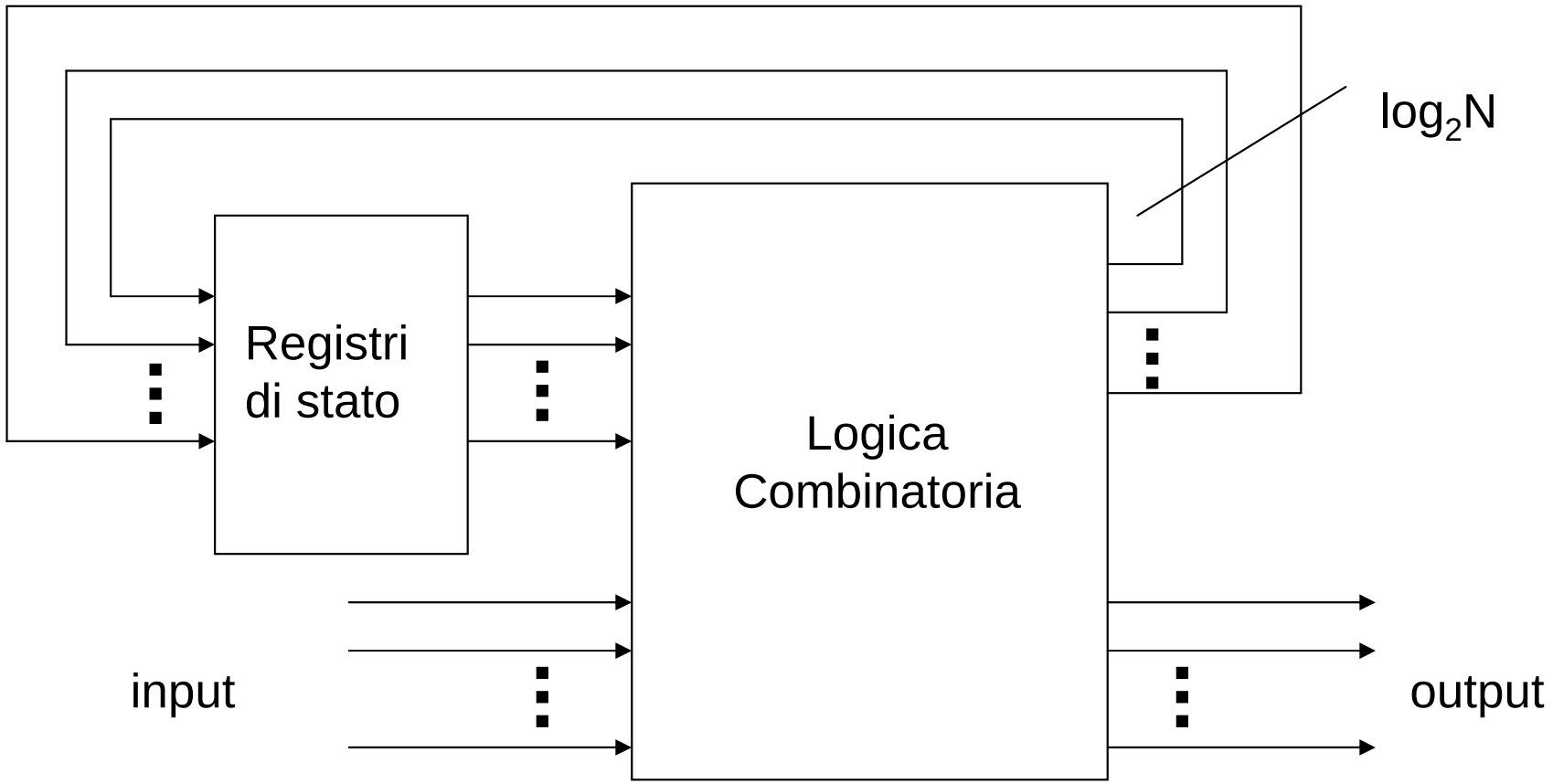


Registro JK

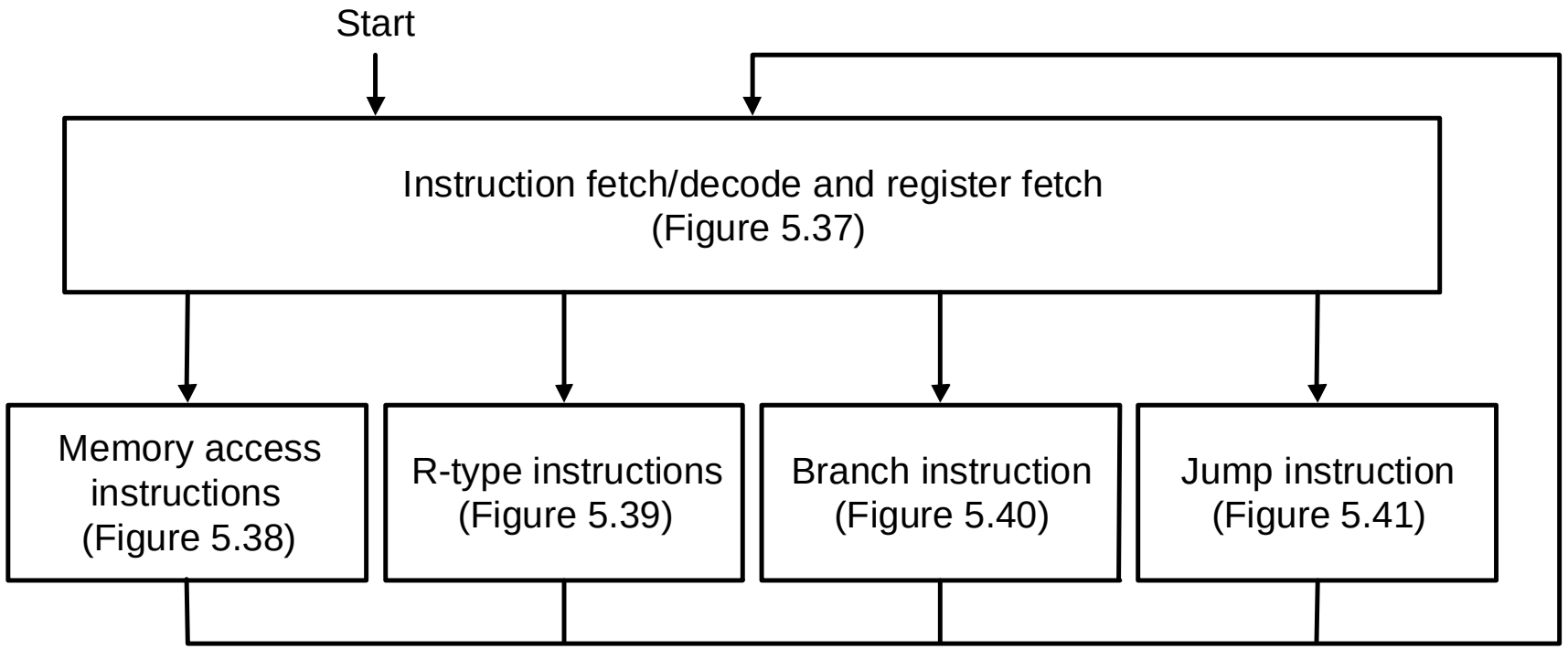


Realizzazione di una macchina a stati finiti

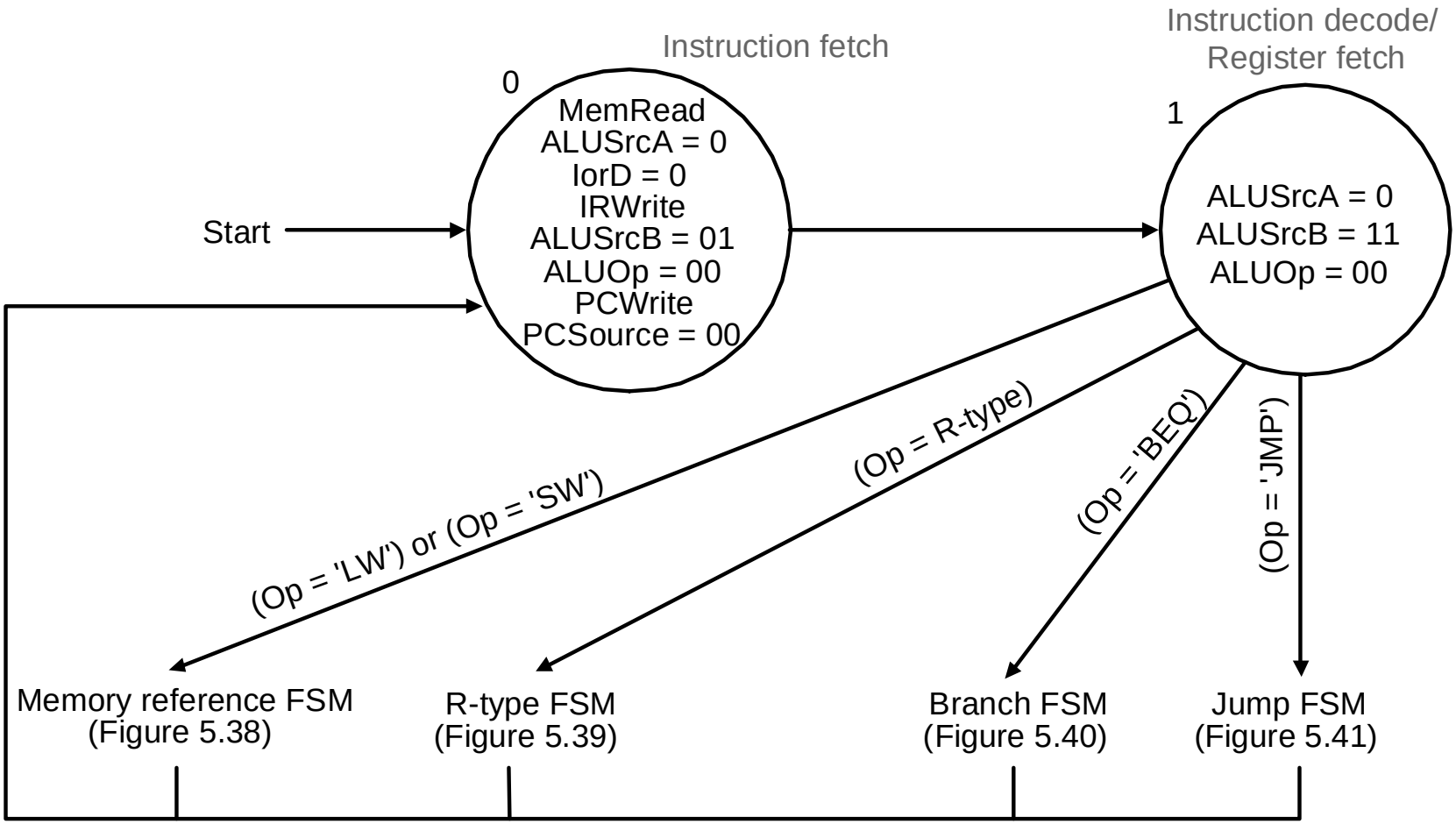
MSF ad N stati ⇨ lo stato è rappresentabile con $\log_2 N$ bit



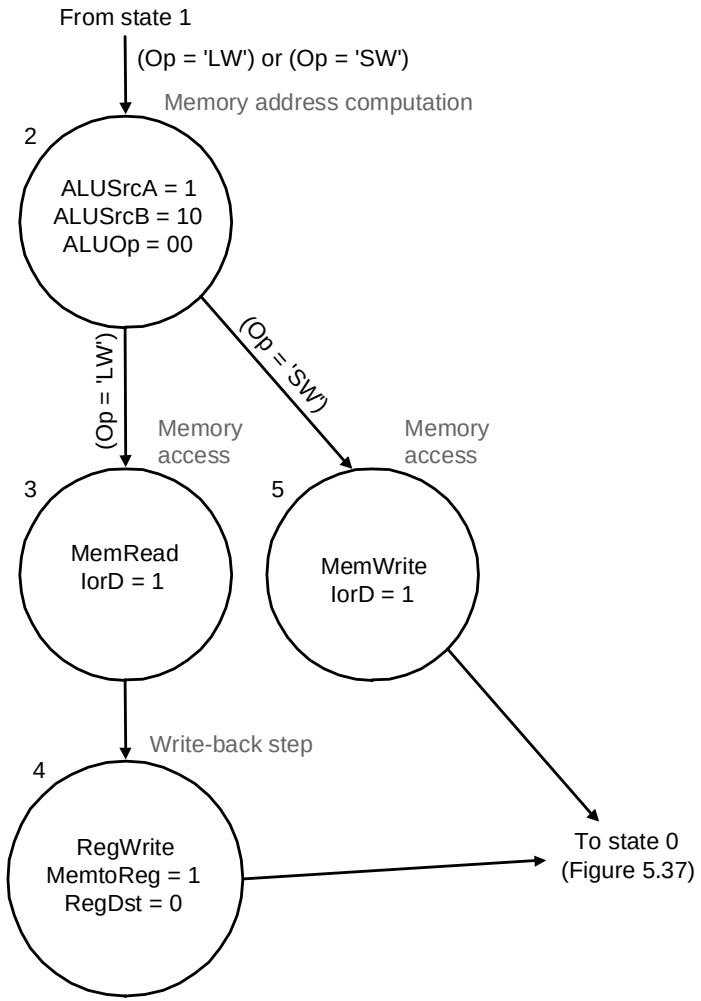
Vista ad alto livello della MSF



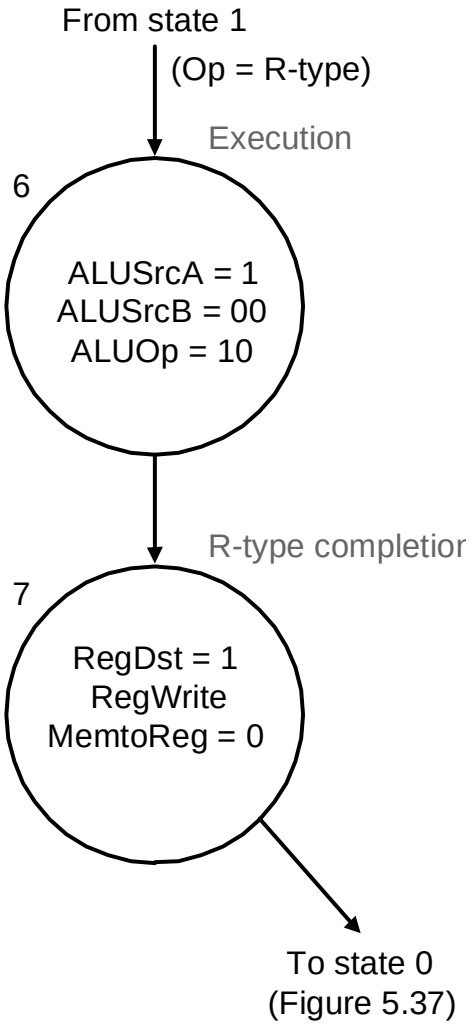
MSF: i primi due stati (comuni)



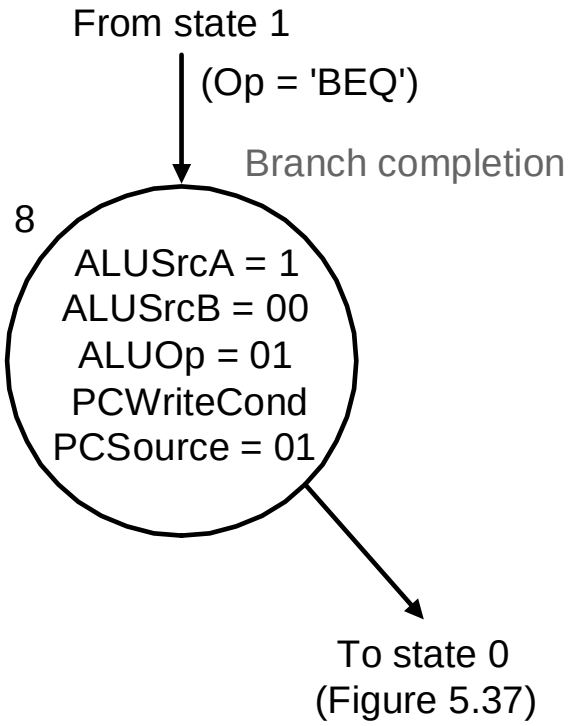
MSF: istruzioni lw/sw

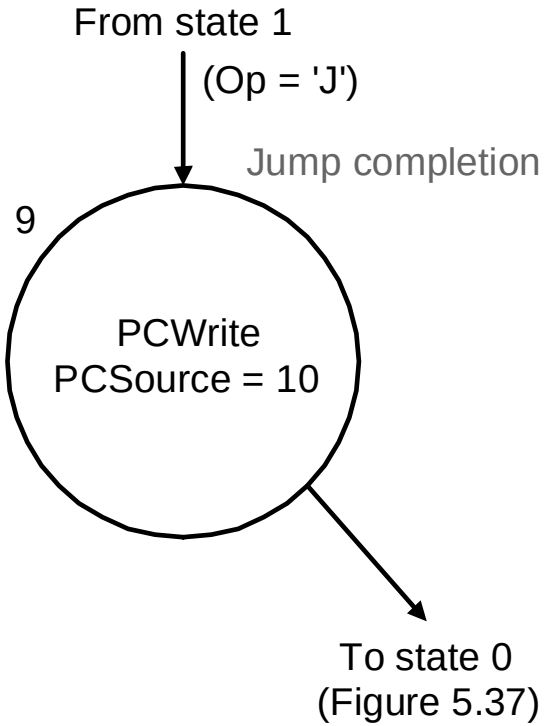


MSF: istruzioni di tipo R

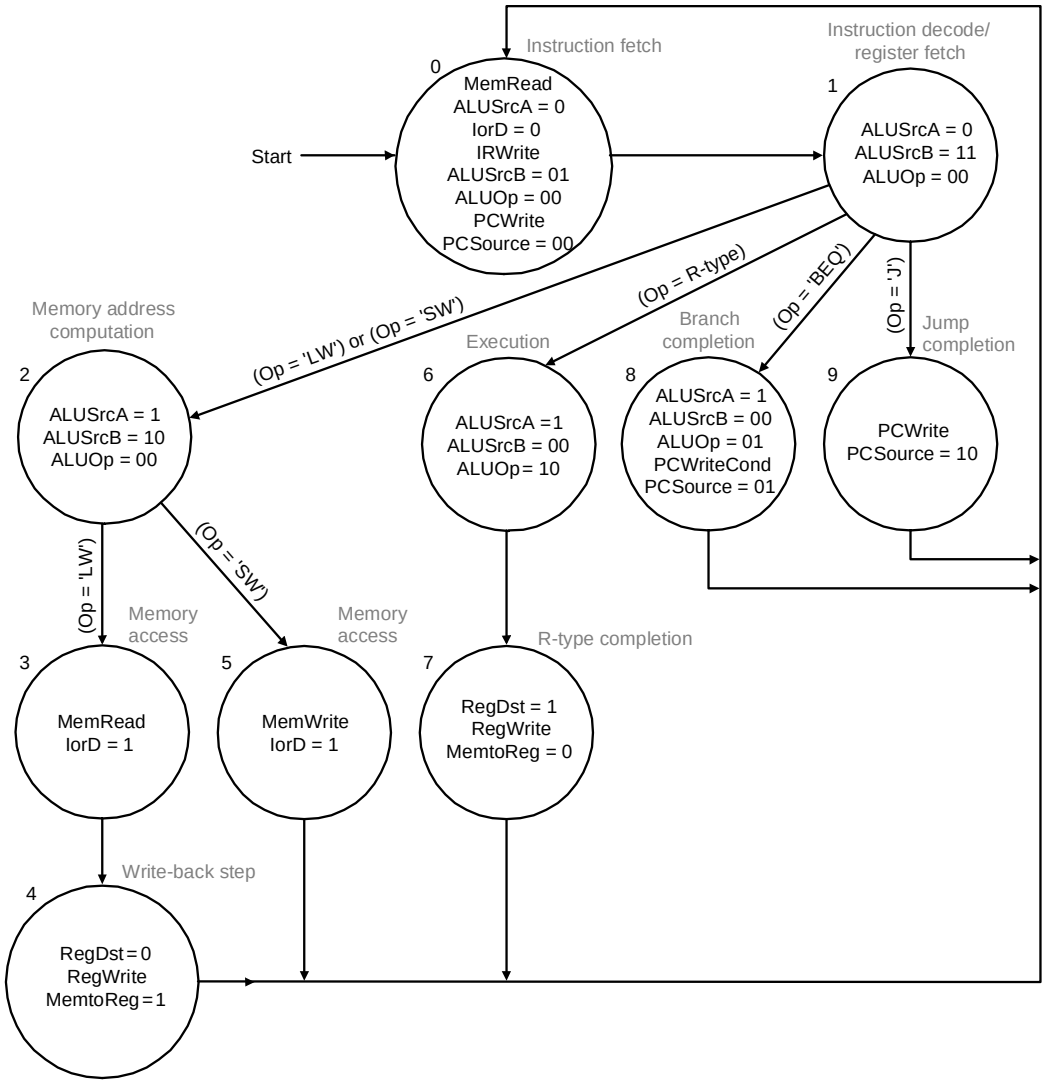


MSF: istruzioni di salto

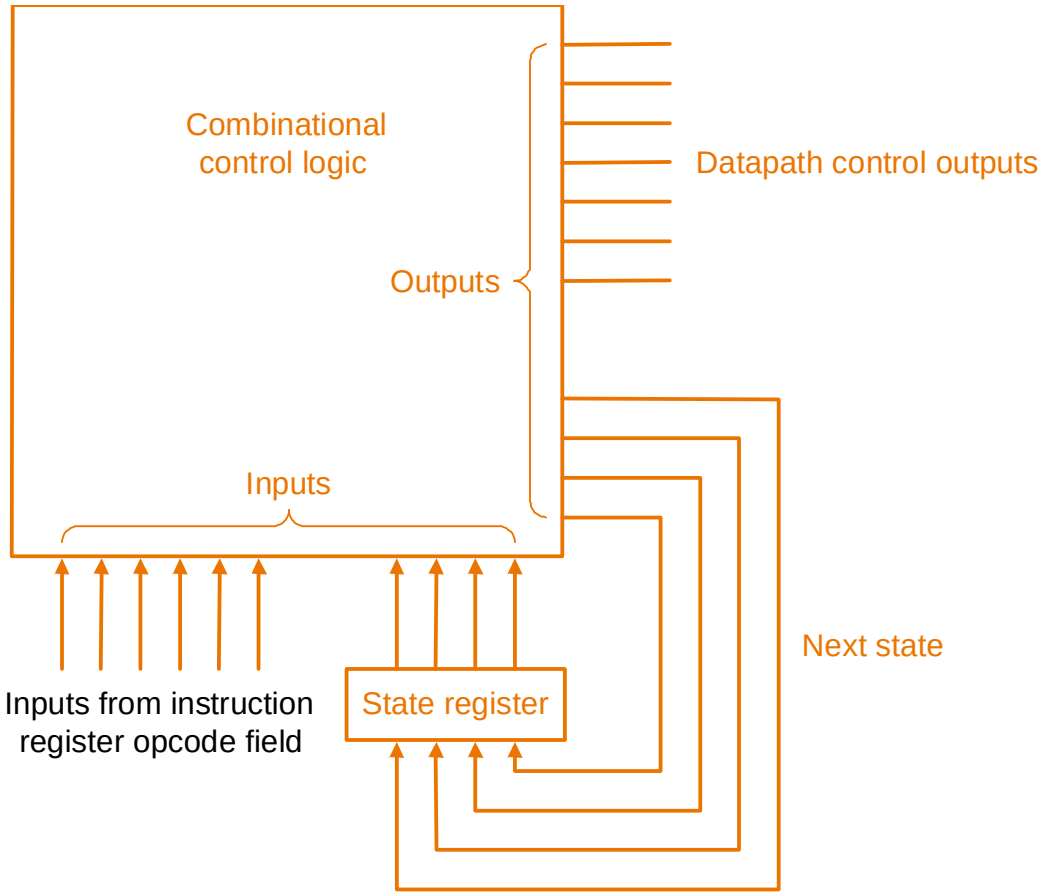




MSF: diagramma a stati finiti completo



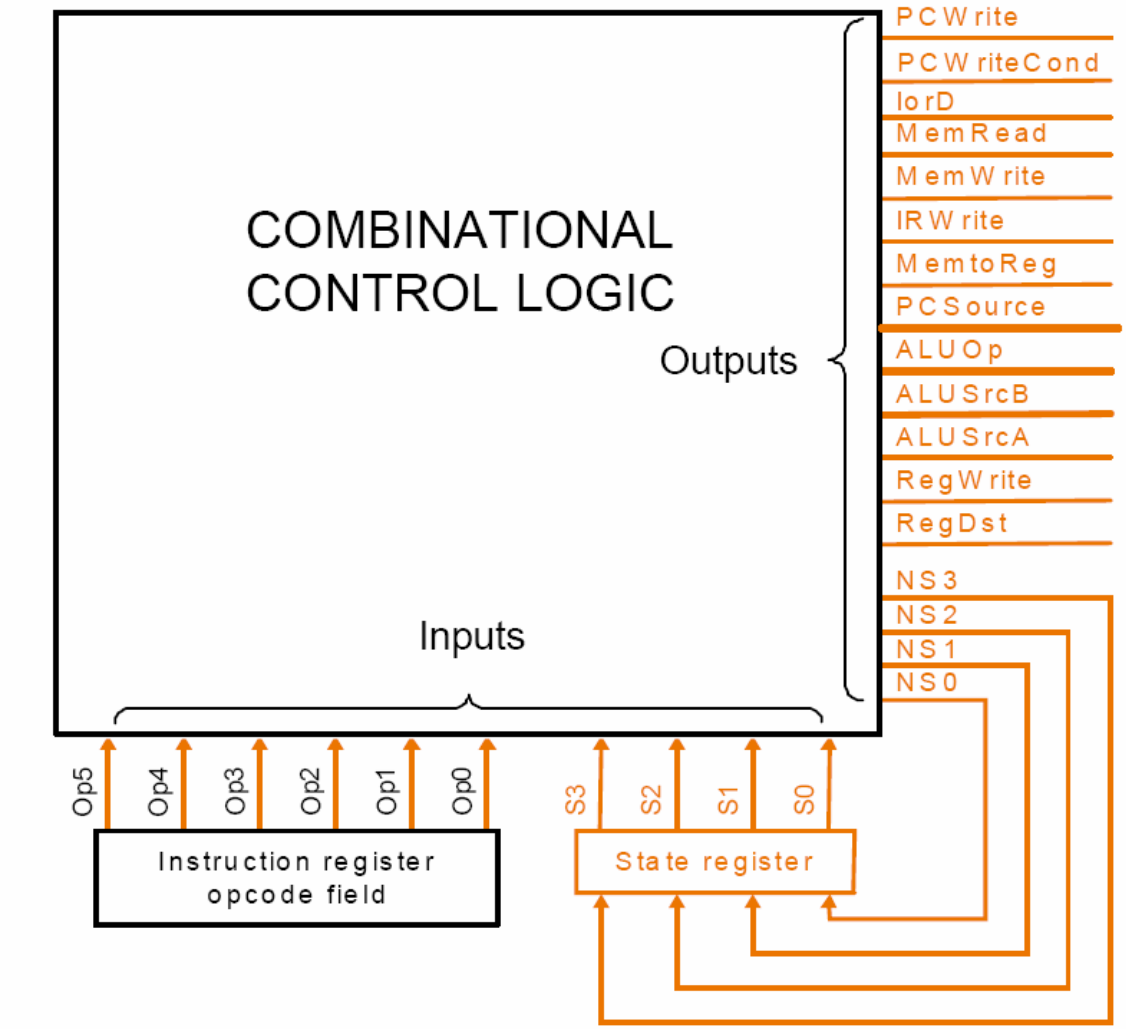
Implementazione della MSF



Implementazione della MSF

- Codifica degli stati
  - 10 stati → 4 bit per codificare lo stato
- Definizione tabella dello stato prossimo
  - Sulla base dello stato corrente e dell'ingresso (valore di op)
- Definizione delle uscite (abilitazioni)
  - Funzione dello stato corrente (macchina di Moore)

Implementazione della MSF



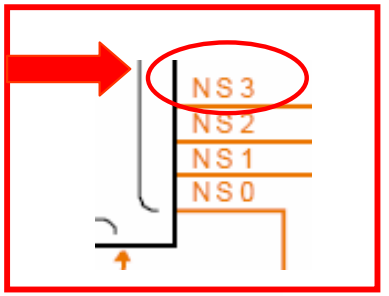
Implementazione della MSF

Output	Current states	Op
PCWrite	State0+State 9	
IorD	State3+State5	
MemRead	State0+State3	
IRWrite	State0	
…	…	
NextState0	State4+State5+State7+State8+State9	
NextState1	State0	
NextState2	State1	(Op='lw')+(Op=sw')
…	…	
NextState5	State2	(Op='sw')
…	…	

Definizione delle uscite

Definizione stato prossimo

Implementazione della MSF



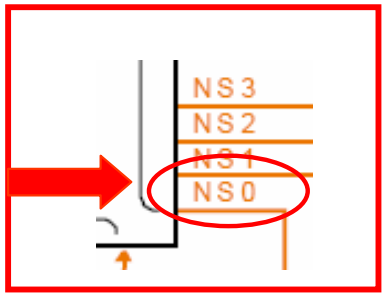
Output	Current states	Op
NextState0	State4+State5+State7+State8+State9	
NextState1	State0	
NextState2	State1	(Op='lw')+(Op=sw')
NextState3	State2	(Op='lw')
NextState4	State3	
NextState5	State2	(Op='sw')
NextState6	State1	(Op='R-type')
NextState7	State6	
NextState8	State1	(Op='beq')
NextState9	State1	(Op='jmp')

Stati che hanno $NS_3=1$

$$NS_3 = State1 \cdot (Op = 'beq') + State1 \cdot (Op = 'jmp')$$

$$NS_3 = \overline{S}_3\overline{S}_2\overline{S}_1S_0\overline{Op}_5\overline{Op}_4\overline{Op}_3Op_2\overline{Op}_1\overline{Op}_0 + \overline{S}_3\overline{S}_2\overline{S}_1S_0\overline{Op}_5\overline{Op}_4\overline{Op}_3\overline{Op}_2Op_1\overline{Op}_0$$

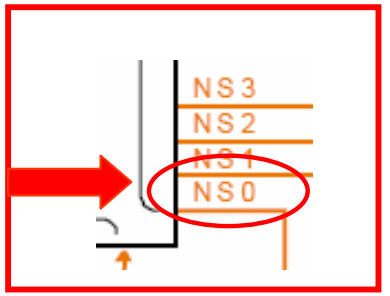
Implementazione della MSF



Output	Current states	Op
NextState0	State4+State5+State7+State8+State9	
NextState1	State0	
NextState2	State1	(Op='lw')+(Op=sw')
NextState3	State2	(Op='lw')
NextState4	State3	
NextState5	State2	(Op='sw')
NextState6	State1	(Op='R-type')
NextState7	State6	
NextState8	State1	(Op='beq')
NextState9	State1	(Op='jmp')

Stati che hanno NS_0=1

Implementazione della MSF

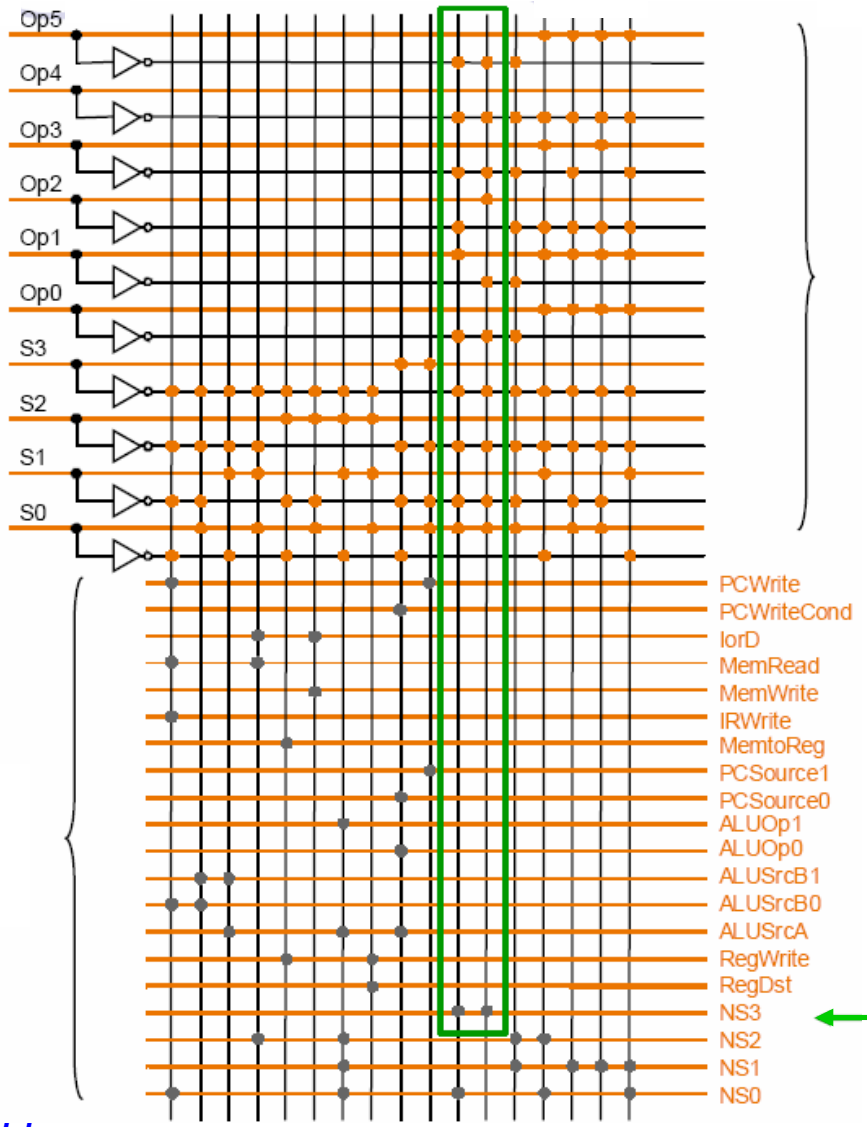


NS_0 è la somma logica di questi 5 termini

$$\begin{aligned}
\text{NextState1} &= \text{State0} = \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot \overline{S0} \\
\text{NextState3} &= \text{State2} \cdot (\text{Op}[5\text{-}0] = \text{lw}) \\
&= \overline{S3} \cdot \overline{S2} \cdot S1 \cdot \overline{S0} \cdot Op5 \cdot \overline{Op4} \cdot \overline{Op3} \cdot \overline{Op2} \cdot Op1 \cdot Op0 \\
\text{NextState5} &= \text{State 2} \cdot (\text{Op}[5\text{-}0] = \text{sw}) \\
&= \overline{S3} \cdot \overline{S2} \cdot S1 \cdot \overline{S0} \cdot Op5 \cdot \overline{Op4} \cdot Op3 \cdot \overline{Op2} \cdot Op1 \cdot Op0 \\
\text{NextState7} &= \text{State6} = \overline{S3} \cdot S2 \cdot S1 \cdot \overline{S0} \\
\text{NextState9} &= \text{State1} \cdot (\text{Op}[5\text{-}0] = \text{jmp}) \\
&= \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0 \cdot \overline{Op5} \cdot \overline{Op4} \cdot \overline{Op3} \cdot \overline{Op2} \cdot Op1 \cdot \overline{Op0}
\end{aligned}$$

Implementazione della MSF

Rete combinatoria tramite PLA



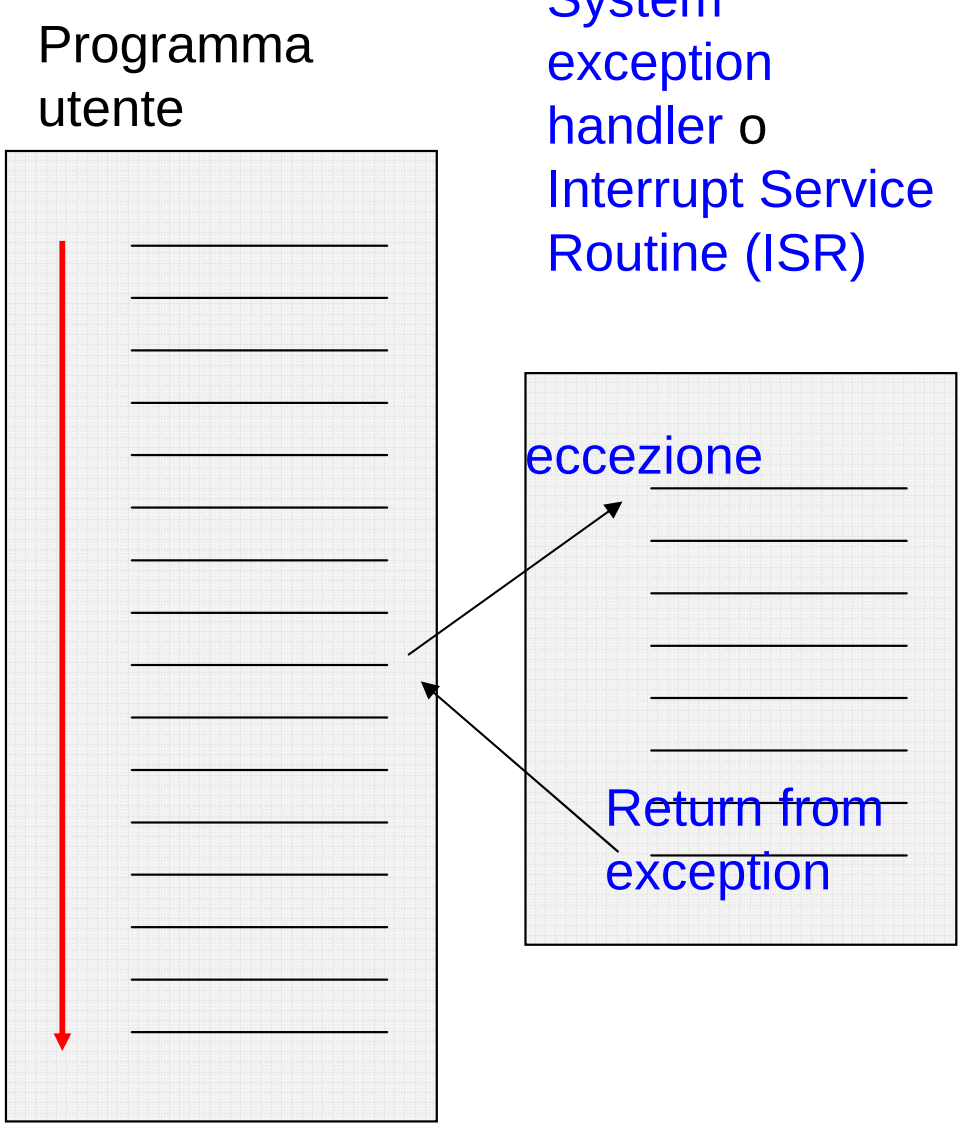
AND plane:
Calcola i mintermini

OR plane:
Calcola le somme

NS_3

Eccezioni

- Trasferimento imprevisto del controllo sulla base del verificarsi di un evento.
- Il sistema risponde con un'azione opportuna all'eccezione.
  - Necessario salvare l'indirizzo dell'istruzione in corrispondenza della quale si è generata l'eccezione
- Al termine viene restituito il controllo al programma interrotto
- Necessario salvare e poi ripristinare lo stato del programma interrotto.



Flusso di controllo normale: sequenza, branch, jump

Tipi di eccezioni

- **Interruzioni**
  - Causate da eventi esterni
  - Asincrone rispetto al programma in esecuzione
  - Possono essere gestite tra due istruzioni consecutive
  - Semplice sospensione e ripresa del programma utente

- **Eccezioni**
  - Causate da eventi interni
    - Condizioni di eccezione (overflow)
    - Errori (parity)
    - Faults (pagine non residenti in memoria)
  - Sincrone rispetto al programma in esecuzione
  - La condizione di eccezione deve essere risolta dall'handler
  - Se la condizione è risolubile, l'istruzione può essere rieseguita ed il programma continuato. In caso contrario, il programma deve essere terminato prematuramente.

Gestione delle eccezioni

- Due esigenze vanno considerate per una corretta gestione delle eccezioni:
  - Salvare lo stato del programma interrotto
    - PC, altre informazioni
  - Individuare la routine da eseguire in funzione della causa che ha generato l'eccezione.
- Salvataggio dello stato
  - Push sullo stack (Vax, Motorola 68K, 80x86)
  - Salvataggio in registri speciali (MIPS)
  - Registri ombra (Motorola 88K)

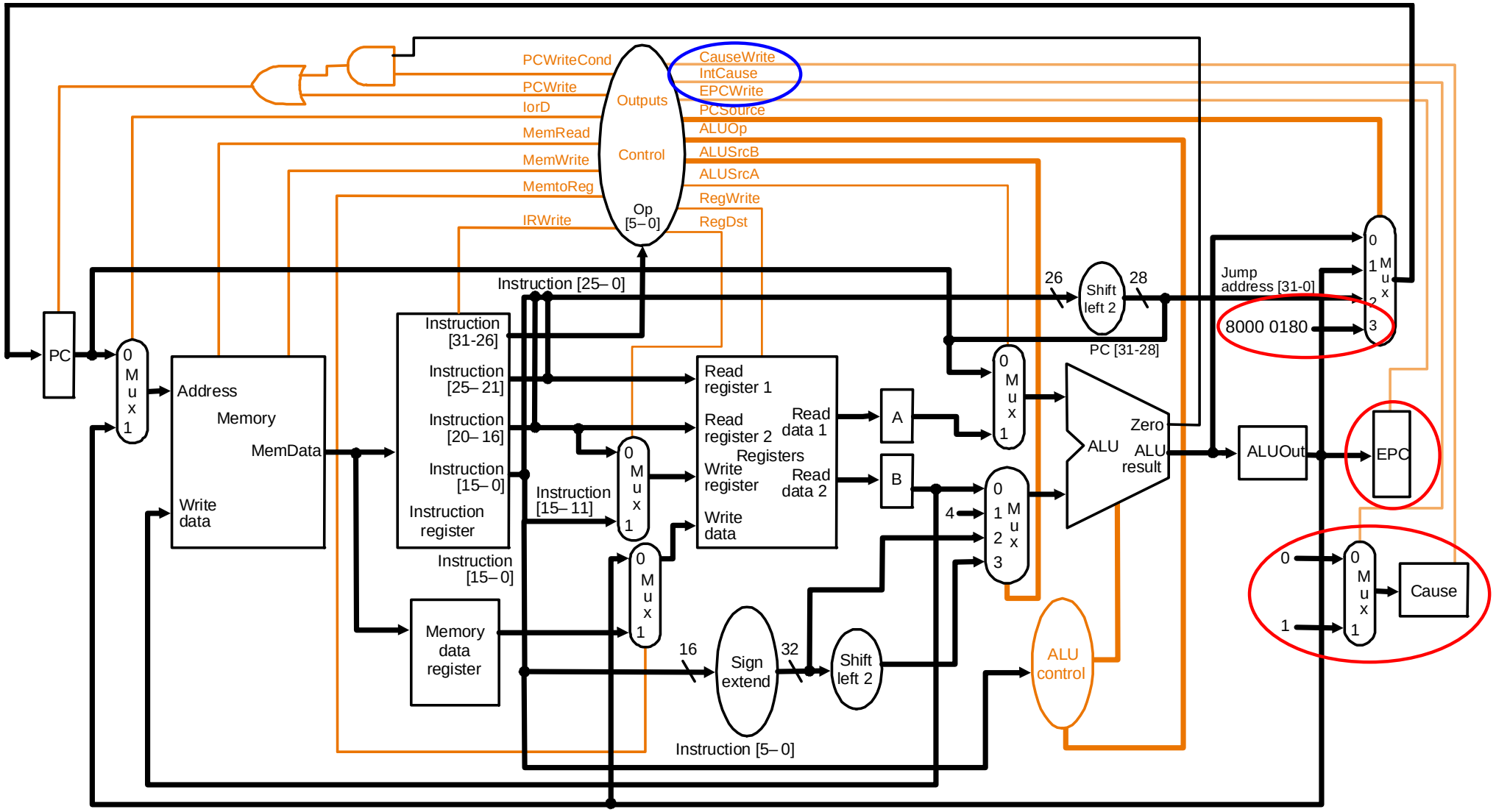
Gestione delle eccezioni: indirizzamento dell'ISR

- Diverse soluzioni:
- Interrupt vettorizzato (68K, 80x86,…)
  - Viene fornito un valore (vector) che è un indice in una tabella (vector table) che contiene gli indirizzi delle varie ISR: PC=Mem[IVT_base+vector||00]. Causa→vector
- Tabella delle ISR (Sparc, 88K,…)
  - Viene fornito uno spiazzamento all'interno di un'area di memoria che contiene tutte le ISR: PC=IT_base+offset||0000. Causa→offset
- Indirizzo fisso (MIPS)
  - Si salta ad un unico indirizzo: unica ISR che si occupa di identificare la causa e avviare la routine opportuna (effettivamente nel MIPS si considerano due entry). PC=Exc_address

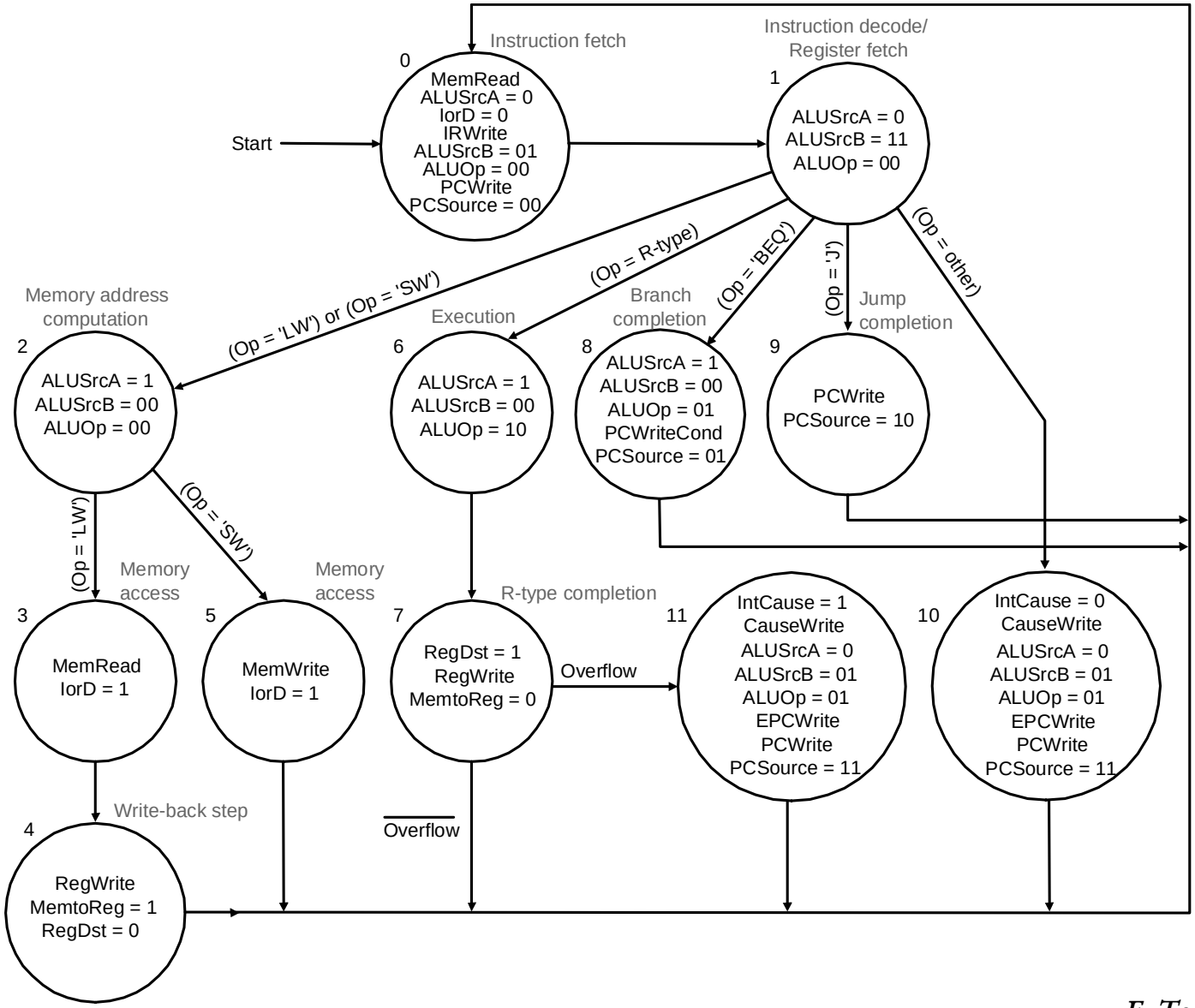
Gestione delle eccezioni: come cambia il controllo ?

- Consideriamo per semplicità due sole cause possibili di eccezione:
  - Istruzione non riconosciuta (generata dal controllo a valle della decodifica dell'istruzione)
  - Overflow (generata dall'ALU nella fase execute di un'istruzione R-type)
- Nuovi compiti da eseguire:
  - Salvare in un registro opportuno (Cause) la causa dell'eccezione
  - Salvare l'indirizzo dell'istruzione che ha causato l'eccezione (PC-4) in un registro opportuno (EPC)
  - Saltare all'entry point dell'ISR

Eccezioni:
Modifica del datapath e del controllo



Eccezioni:
Modifica della MSF



Interruzioni

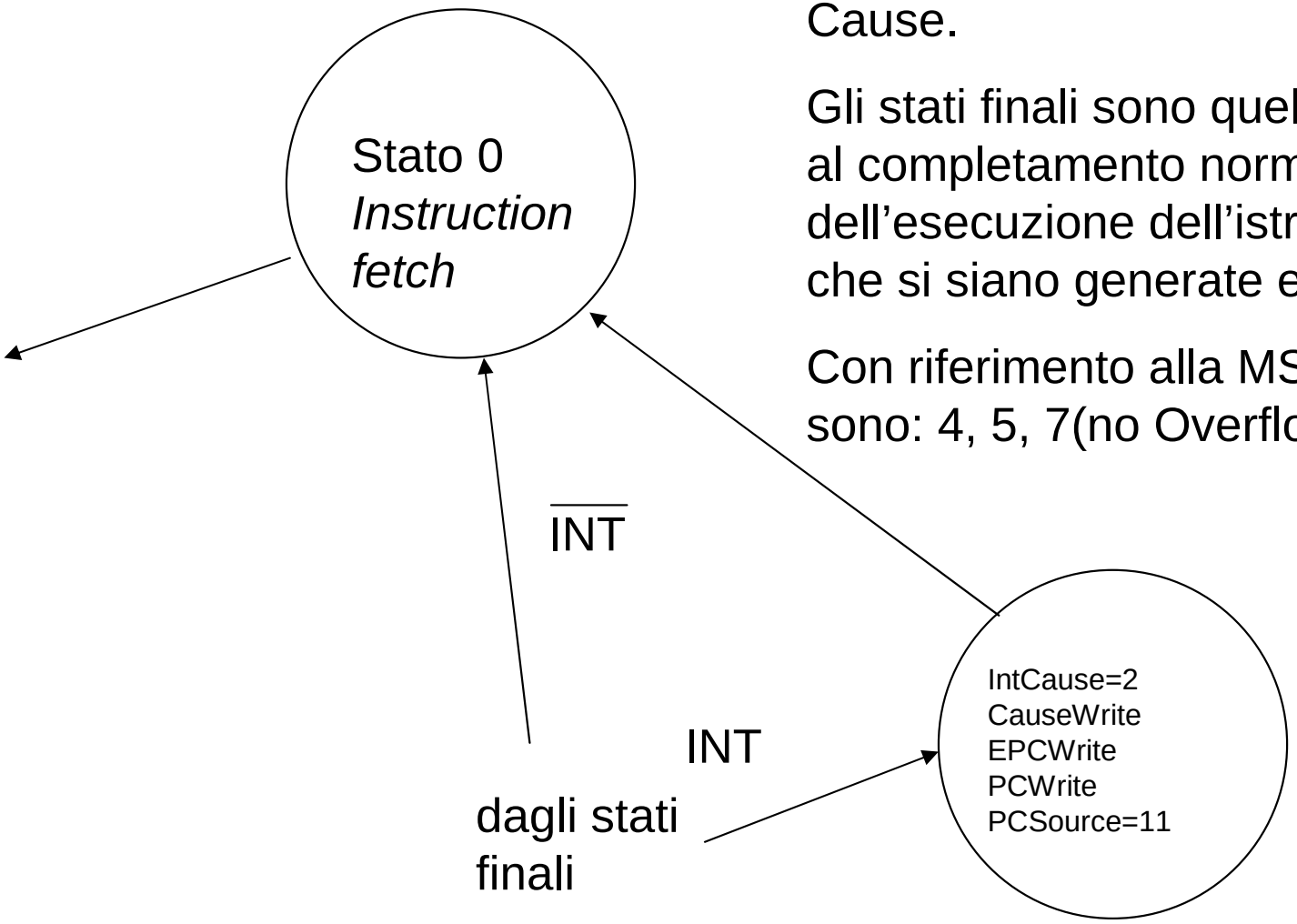
- Sono essere gestite tra due istruzioni consecutive
- Semplice sospensione e ripresa del programma utente
- Al termine dell'esecuzione dell'istruzione corrente si verifica se è attivo un segnale di interruzione
- In tal caso si esegue il cambio di contesto descritto per le eccezioni, con la differenza che nel registro Cause viene codificato l'insorgere di un'interruzione.

Interruzioni: Modifica della MSF

Assumiamo che l'interruzione sia codificata con il valore 2 nel registro Cause.

Gli stati finali sono quelli corrispondenti al completamento normale dell'esecuzione dell'istruzione (senza che si siano generate eccezioni).

Con riferimento alla MSF precedente, sono: 4, 5, 7(no Overflow), 8, 9



Interruzioni:
Modifica del datapath e del controllo