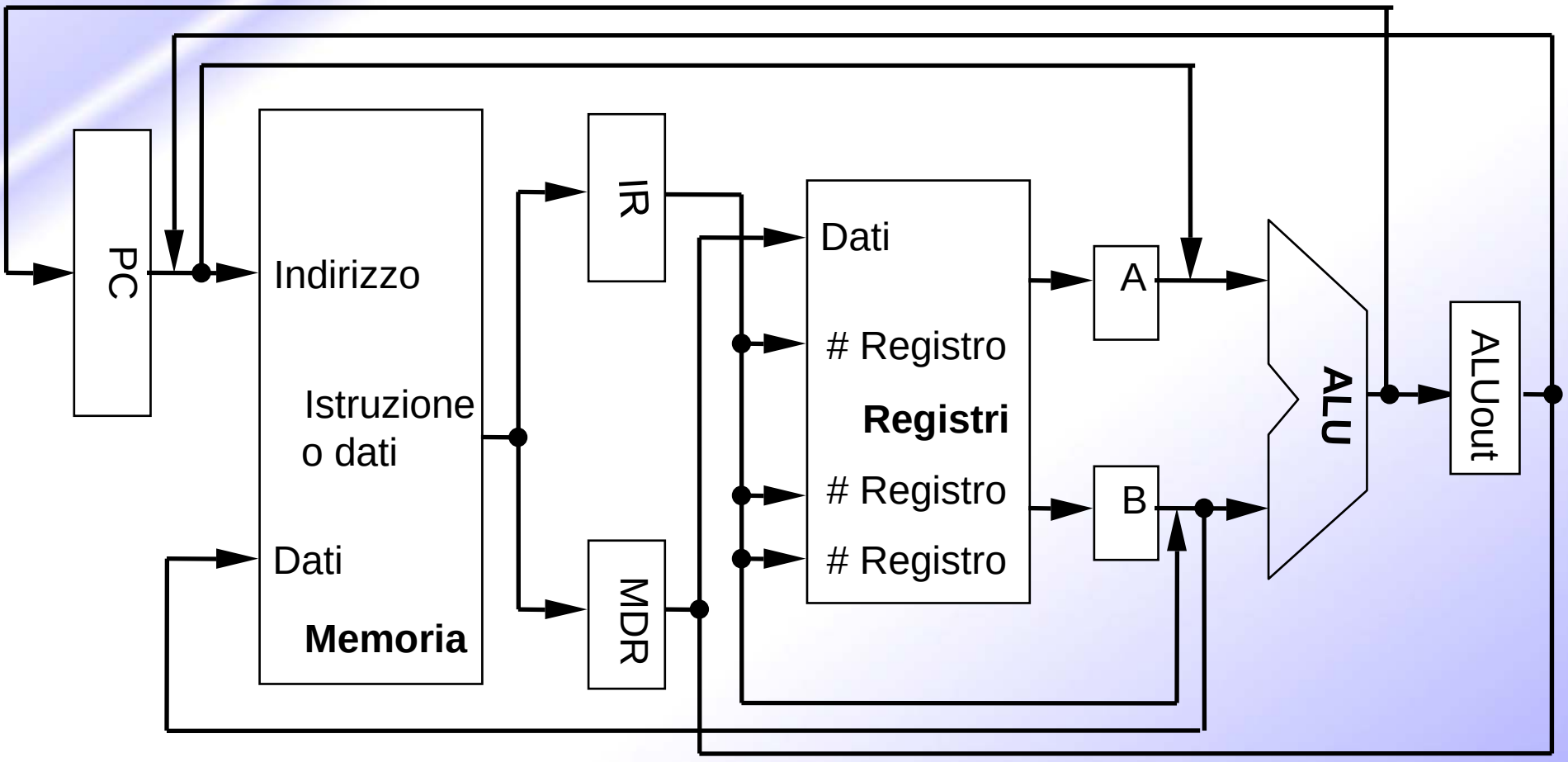
Il processore a ciclo multiplo: unità di elaborazione dati e unità di controllo

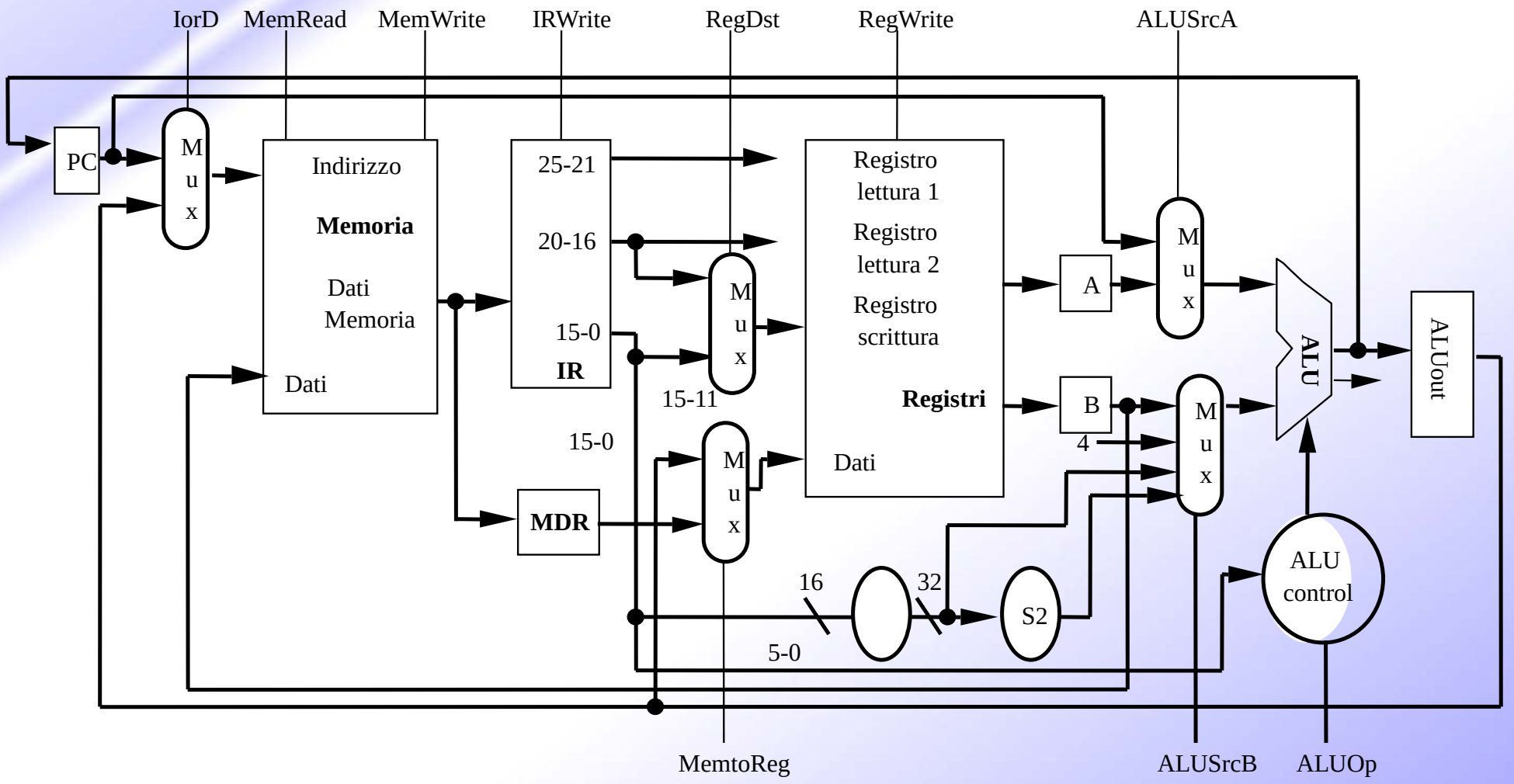
Approccio con più cicli

- Riuseremo le unità funzionali
  - ALU usata per calcolare indirizzi e per incrementare PC
  - memoria usata per istruzioni e dati
- I segnali di controllo non saranno determinati solo dall'istruzione
  - dipendono anche dal ciclo di clock
- Alla fine del ciclo
  - memorizza valori da usare in cicli successivi
    - introduce registri interni "addizionali"

Il cammino dei dati a più cicli



Il cammino dei dati a più cicli



Cinque passi di esecuzione

- Caricamento istruzione
- Decodifica istruzione e caricamento registri
- Esecuzione, calcolo indirizzo di memoria, o completamento del salto condizionato
- Accesso alla memoria o completamento dell'istruzione di tipo R
- Riscrittura

Caricamento istruzione

- Usa PC per prendere l'istruzione e metterla nel registro IR
- Incrementa il PC di 4 e mette il risultato in PC
- Può essere descritto brevemente usando il linguaggio RTL (Register Transfer Language)

```
IR = Memory[PC];
PC = PC + 4;
```

Decodifica e caricamento registri

- Legge registri rs e rt in caso ne avessimo bisogno
- Calcola indirizzo di salto in caso l'istruzione sia un salto condizionato
- RTL:

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

Terzo passo

- ALU esegue una di tre funzioni in base al tipo di istruzione
- Accesso alla memoria:

```
ALUOut = A + sign-extend(IR[15-0]);
```

- Tipo R:

```
ALUOut = A op B;
```

- Salto condizionato:

```
if (A==B) PC = ALUOut;
```

Quarto passo

Lettura dalla memoria

```
MDR = Memory[ALUOut];
```

Scrittura in memoria

```
Memory[ALUOut] = B;
```

Terminazione istruzioni di tipo R

```
Reg[IR[15-11]] = ALUOut;
```

Riscrittura

- Lettura dalla memoria

Reg[IR[20-16]] = MDR;

Riassunto

Passo	Azioni per istruzioni di tipo R	Azioni per istruzioni di riferimento alla memoria	Azioni per salti condizionati
1		IR = Memory[PC] PC = PC + 4	
2		A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)	
3	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A ==B) then PC = ALUOut
4	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B	
5		Load: Reg[IR[20-16]] = MDR	

Semplici domande - qui

- Quanti cicli richiede l'esecuzione del seguente codice?

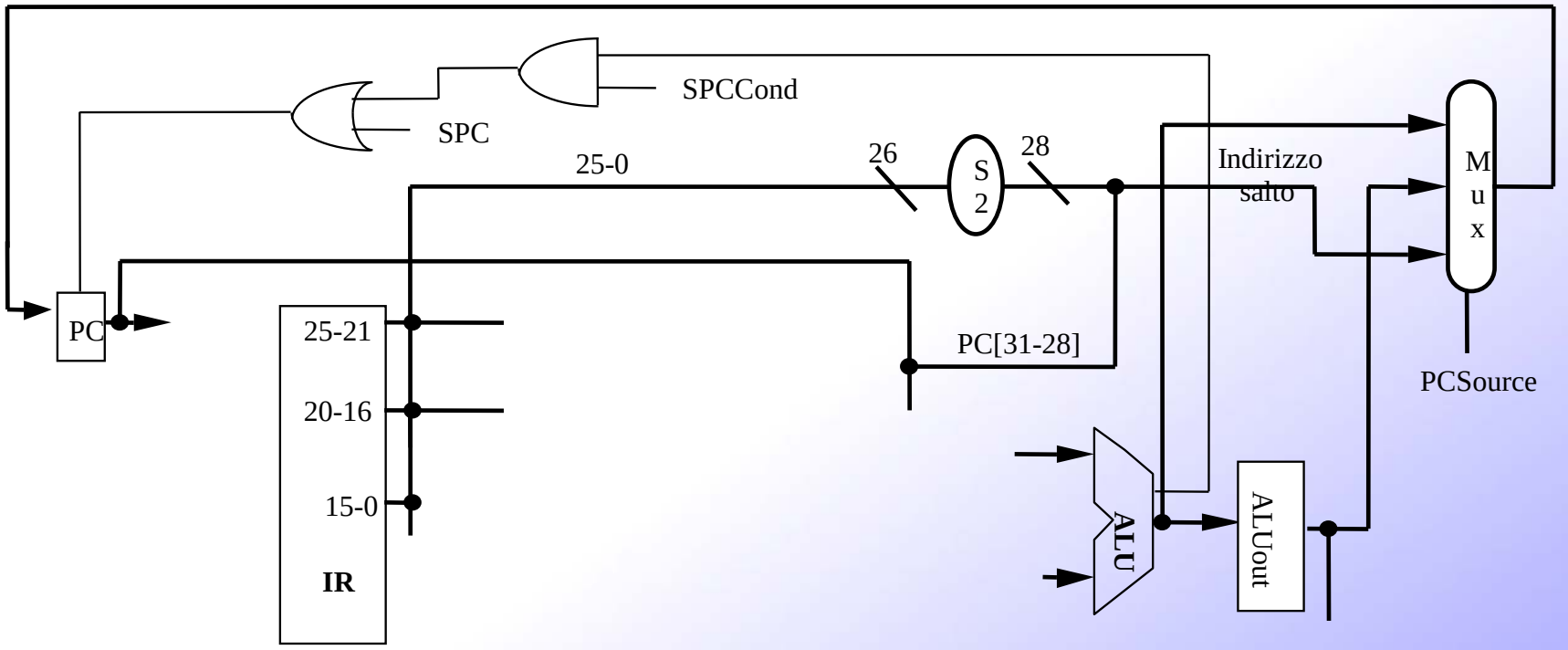
```
        lw $t2, 0($t3)
        lw $t3, 4($t3)
        beq $t2, $t3, Label  #no
        add $t5, $t2, $t3
        sw $t5, 8($t3)
Label:  ...
```

- Cosa succede durante l'ottavo ciclo di esecuzione?
- In quale ciclo l'addizione tra $t2 e $t3 ha effettivamente luogo?

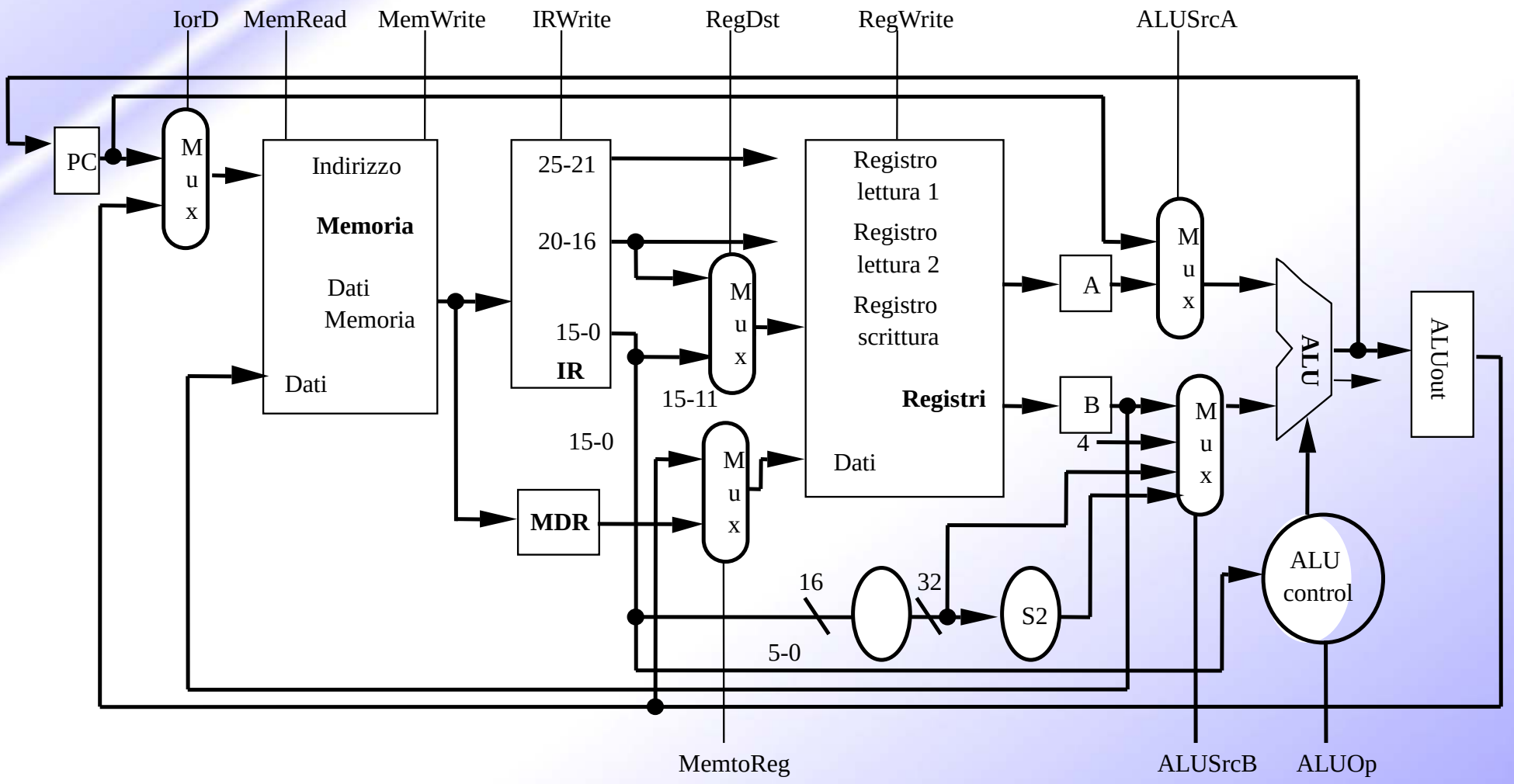
Istruzione j

- Nel terzo passo:

PC = PC[31-28] || (IR(25-0)<<2);



Il cammino dei dati a più cicli



Realizzazione del controllo

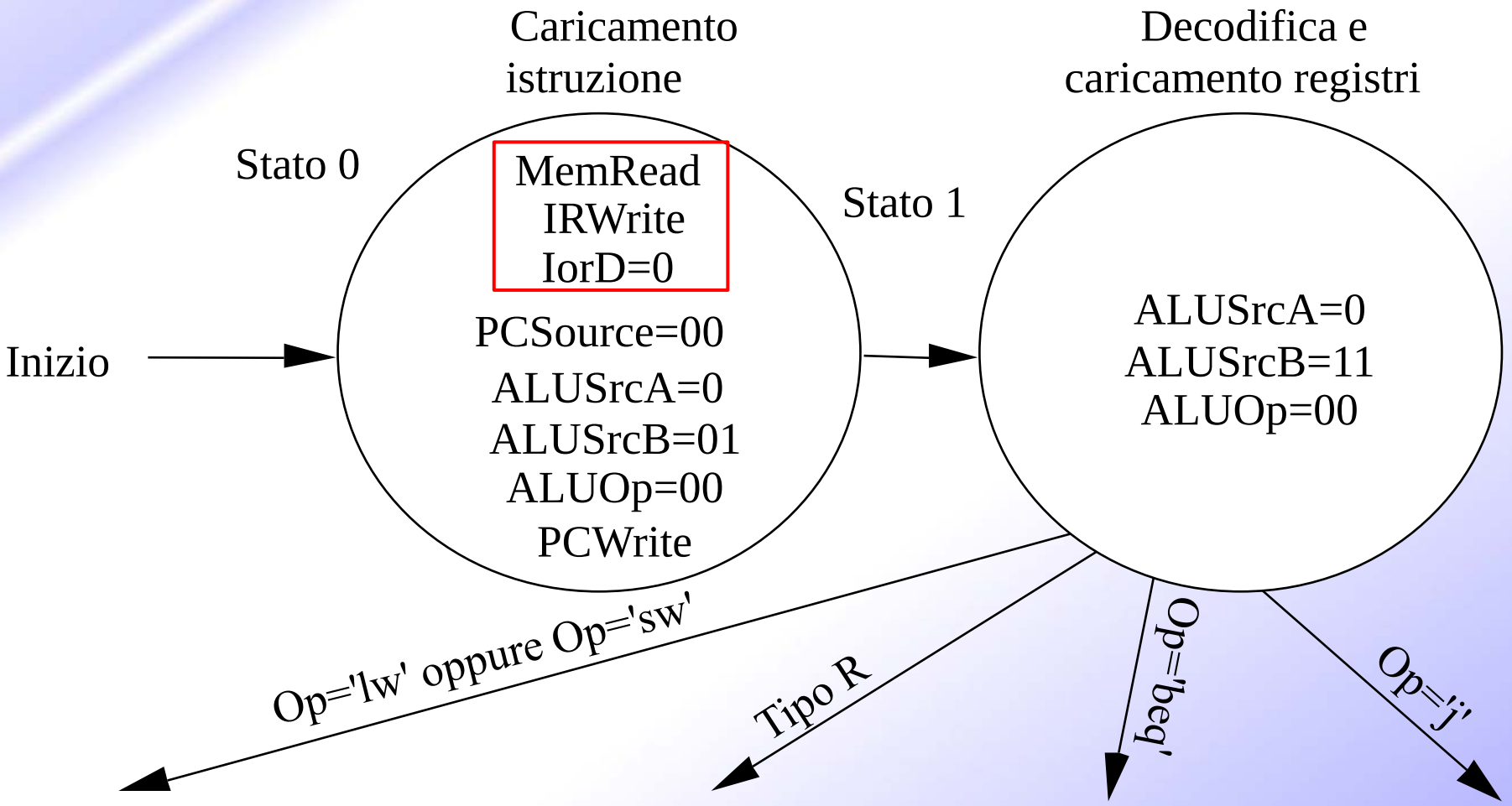
- I valori dei segnali di controllo dipendono da:
  - quale istruzione deve essere eseguita
  - quale passo stiamo realizzando
- Useremo l'informazione accumulata per specificare graficamente una macchina a stati finiti
  - segnali non specificati considerati non asseriti
  - segnali di controllo dei multiplexer sempre specificati
- L'implementazione può essere ricavata dalla specifica

Macchine a stati finiti

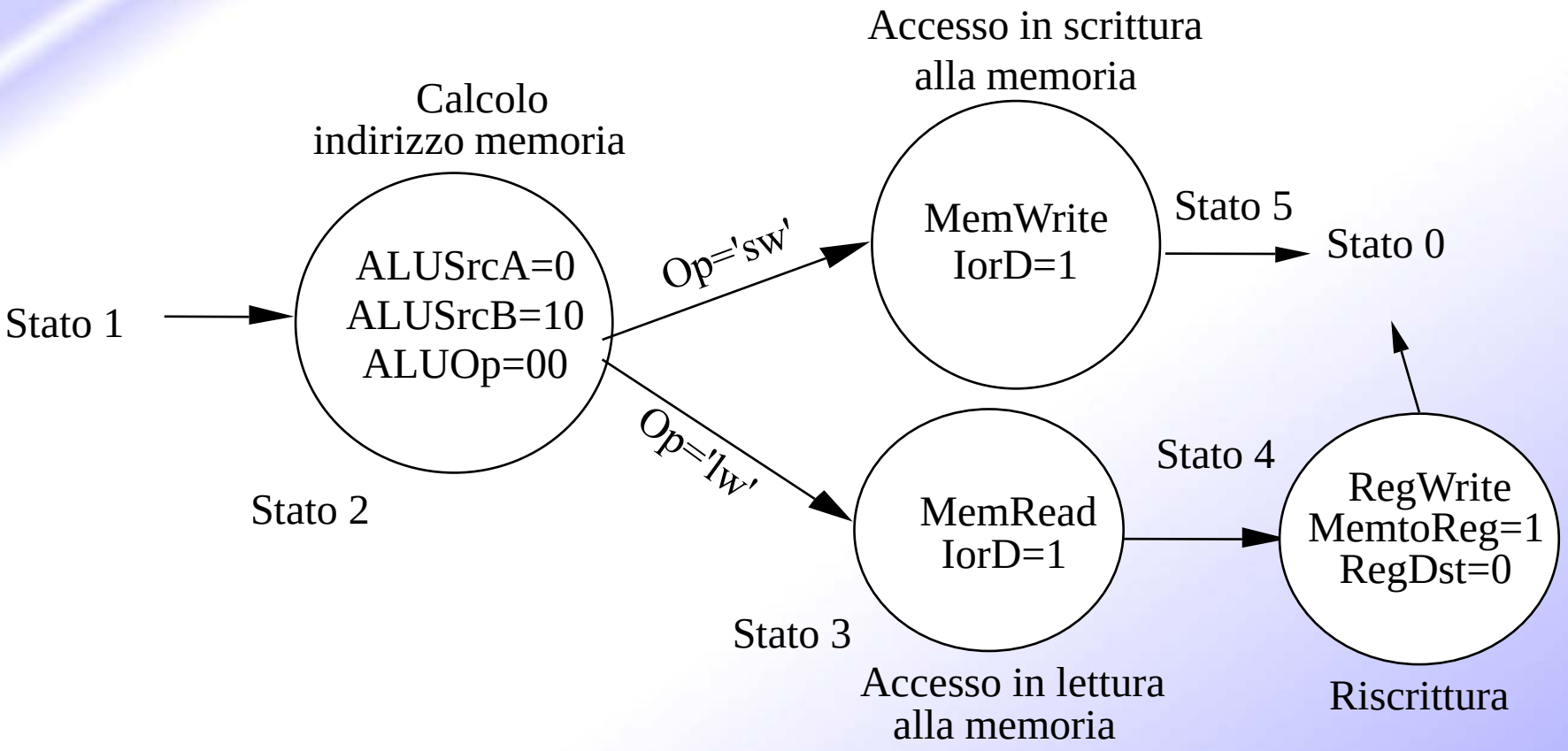
- Definizione:
  - un insieme di stati
  - una funzione che determina il prossimo stato in base allo stato corrente ed all'input
  - una funzione di output determinato dallo stato corrente ed eventualmente dall'input

- Useremo una macchina di Moore (output in base al solo stato corrente)

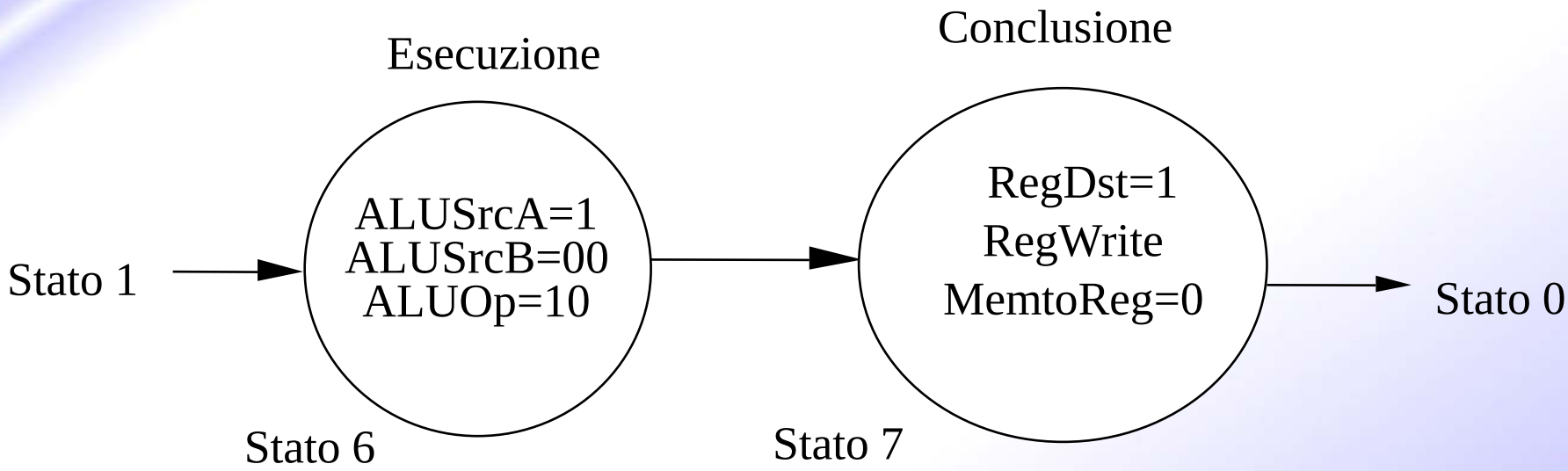
Primi due stati



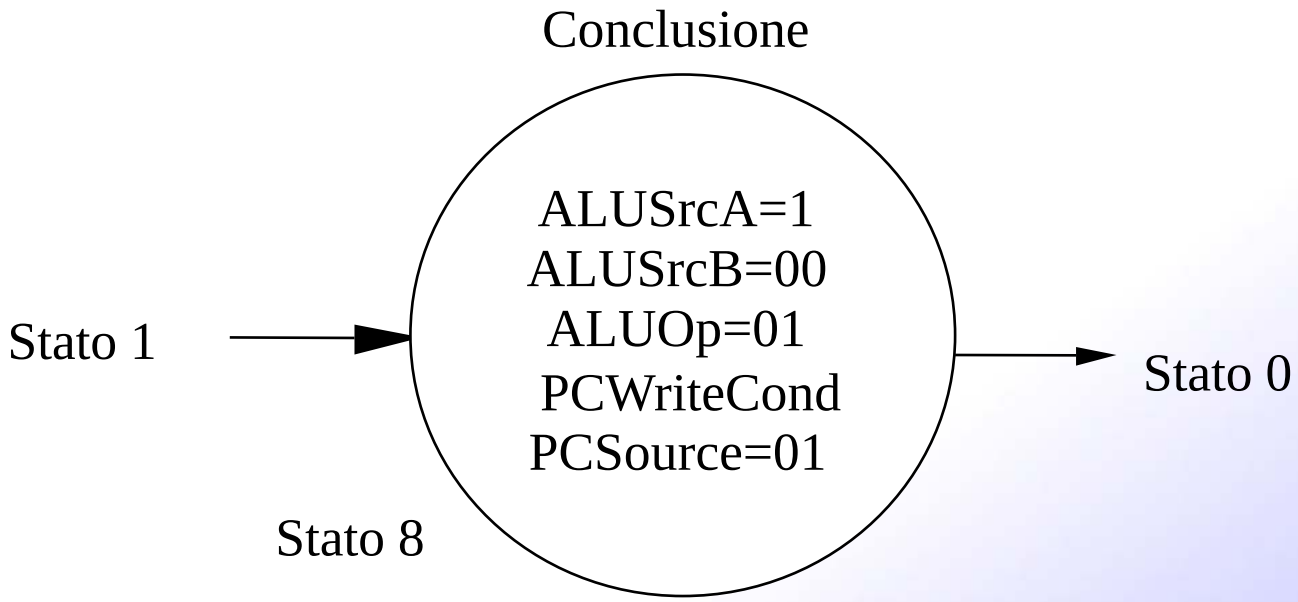
Accesso alla memoria



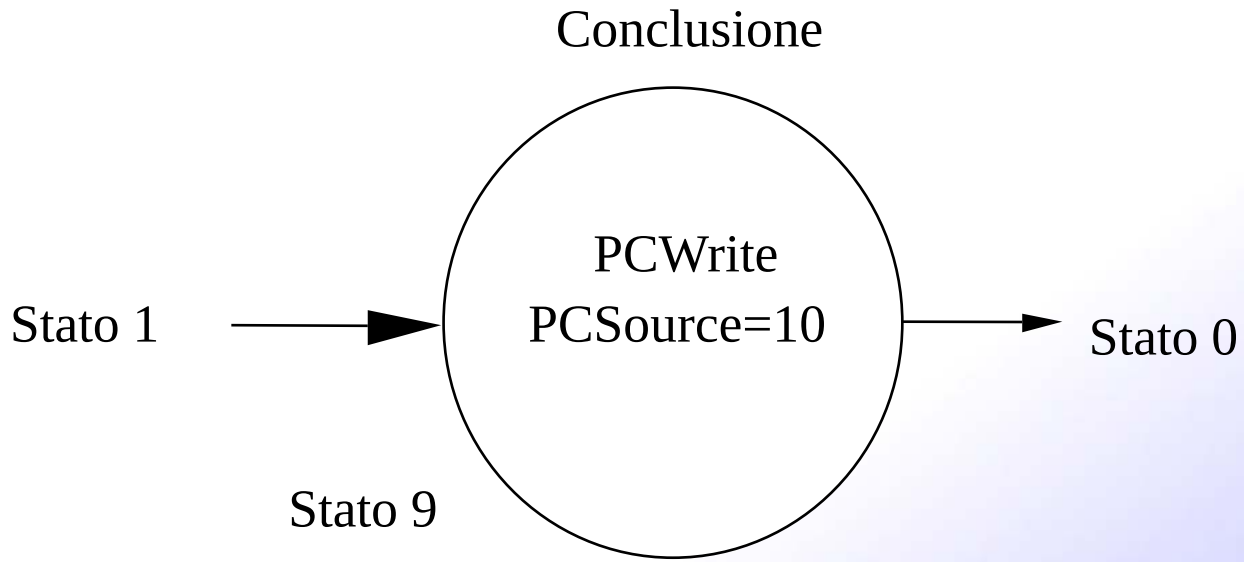
Tipo R


Esecuzione
Conclusione
ALUSrcA=1
ALUSrcB=00
ALUOp=10
RegDst=1
RegWrite
MemtoReg=0
Stato 1
Stato 6
Stato 7
Stato 0

Salto condizionato



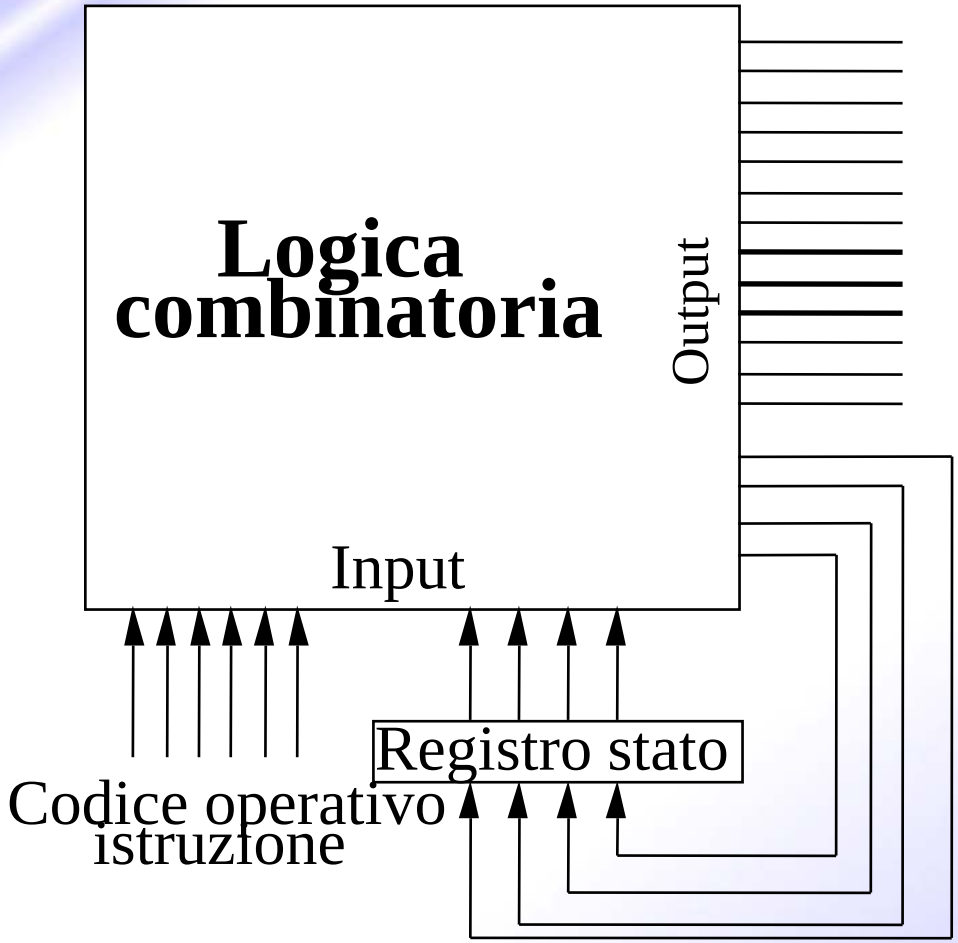
Salto incondizionato



CPI nell'approccio multiciclo

- Sappiamo che:
  - lw richiede 5 cicli
  - sw e tipo R richiede 4 cicli
  - salto (condizionato e non) richiede 3 cicli
- Assumiamo che:
  - 22% lw, 11% sw, 49% tipo R, 16% beq, 2% j
- CPI (cicli per istruzione):
  - $5 \times 0.22 + 4 \times 0.11 + 4 \times 0.49 + 3 \times 0.16 + 3 \times 0.2 = 4.04$
- Meglio che se tutte le istruzioni richiedessero lo stesso numero di cicli (5)

Implementazione



Segnali di controllo
cammino dei dati

Prossimo stato

Implementazione ROM

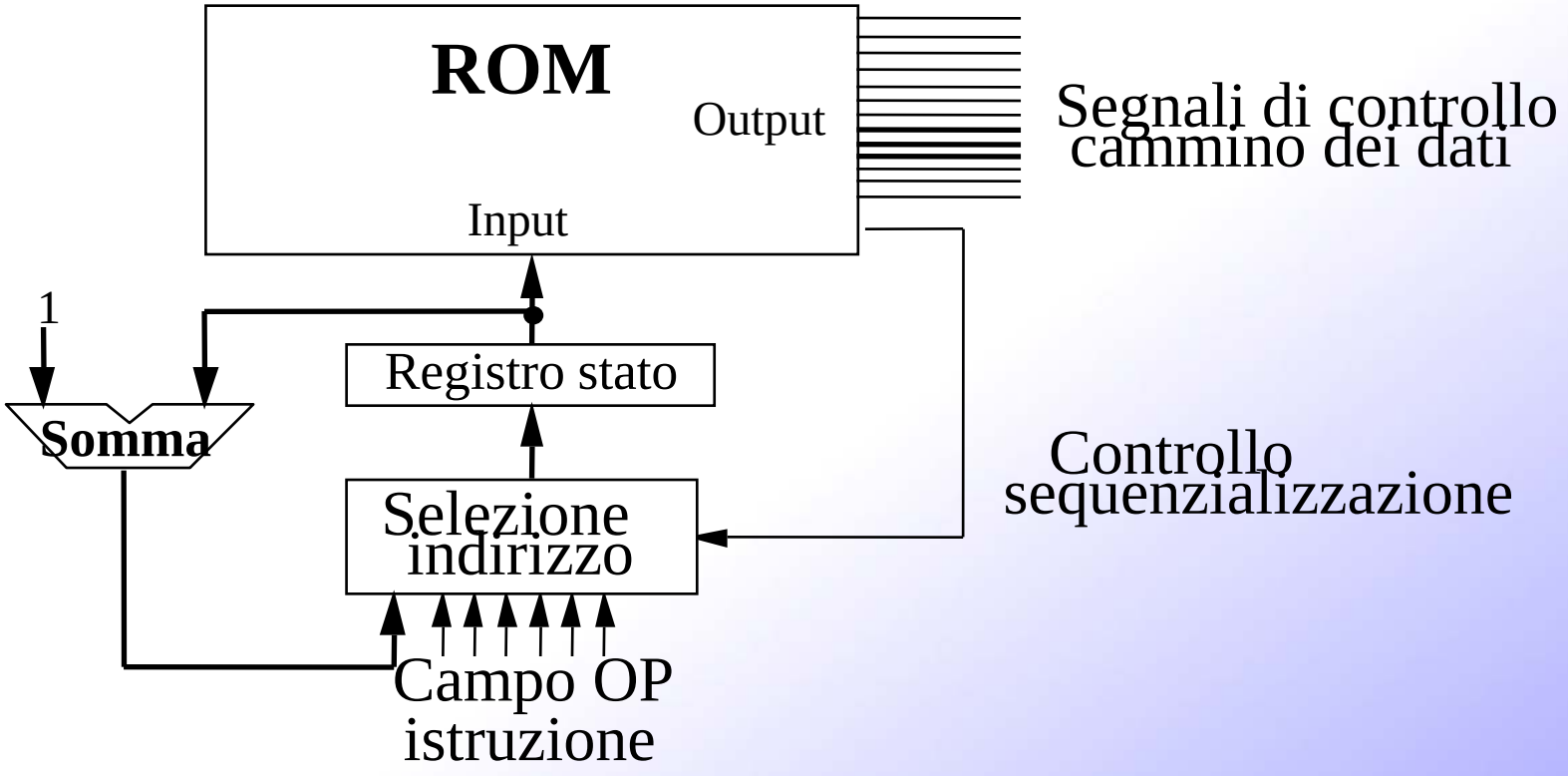
- ROM (Read Only Memory):
  - i valori delle locazioni di memoria sono fissati una volta per tutte
- ROM usata per realizzare tabella di verità
  - se l'indirizzo è di m bit, possiamo indirizzare 2m elementi nella ROM
  - gli output sono i bit di dati a cui l'indirizzo punta
- Quanti input abbiamo?
  - 6 bit di codice operativo + 4 bit di stato = 10 linee di indirizzo (quindi, 2^{10} = 1024 diversi indirizzi)
- Quanti output abbiamo?
  - 16 segnali di controllo + 4 bit di stato = 20 output
- Dimensione ROM: 2^{10} x 20 = 20K bit

Migliorare l'implementazione

- Per molti elementi della tabella gli output sono gli stessi
  - codice operativo è spesso ignorato
- Spezzare la tabella in due parti
  - 4 bit di stato specificano i 16 output
    - 2^4 x 16 bit di ROM
  - 10 bit specificano i 4 bit del prossimo stato
    - 2^{10} x 4 bit di ROM
  - totale: 256 + 4096 = 4.3K bit di ROM

Uso di un sequenzializzatore

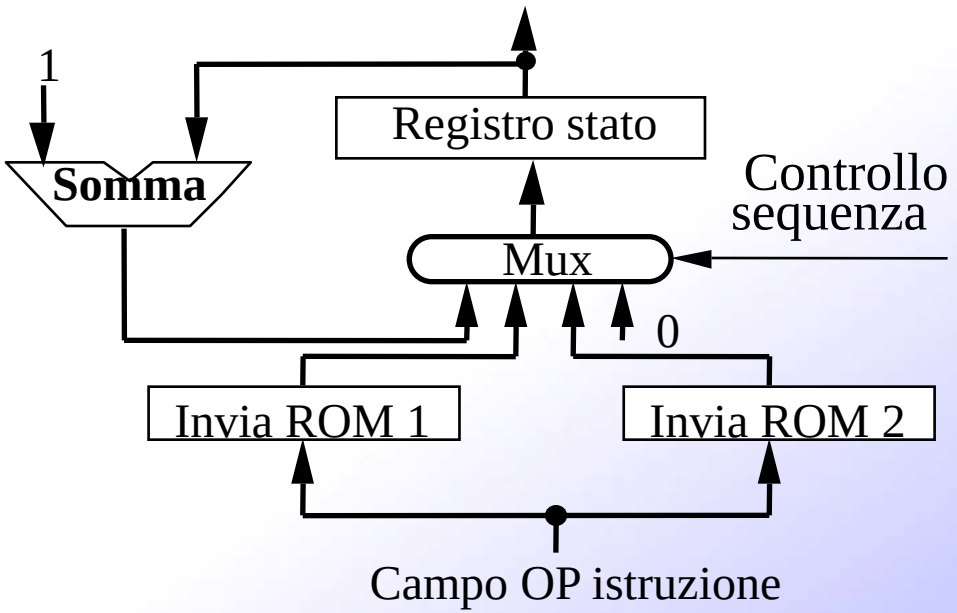
- Macchina di Moore
- Stato successivo spesso stato attuale +1



Dettagli

Invia ROM 1		
Op	**Codice operativo**	**Valore**
000000	Tipo R	0110
000010	j	1001
000100	beq	1000
100011	lw	0010
101011	sw	0010

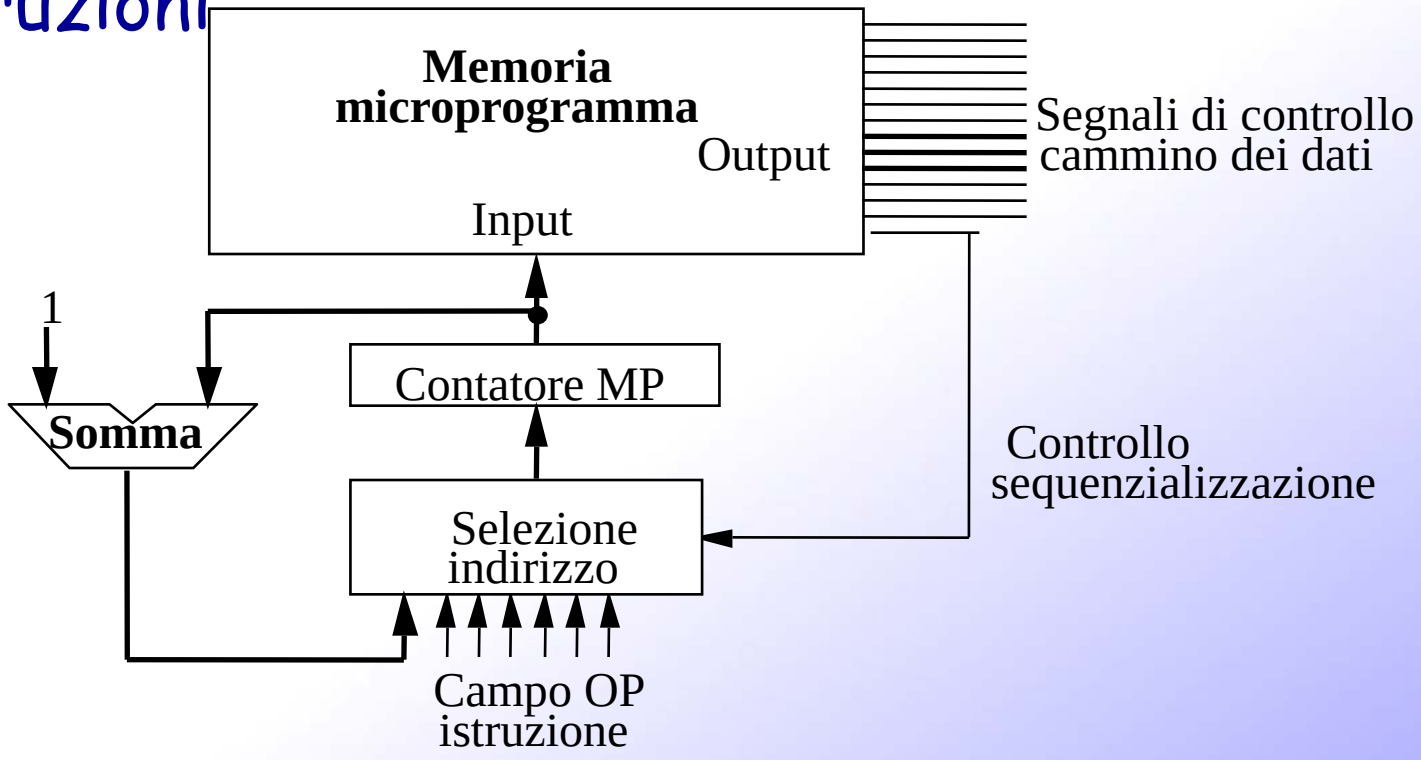
Invia ROM 2		
Op	**Codice operativo**	**Valore**
100011	lw	0011
101011	sw	0101



Numero stato	Azione selezione indirizzo	Controllo sequenza
0,3,6	Usa stato incrementato	3
1	Usa invia ROM 1	1
2	Use invia ROM 2	2
4,5,7,8,9	Sostituisci numero stato con 0	0

Microprogrammazione

- Metodologia di specifica
  - appropriata per centinaia di codici operativi e cicli
  - segnali specificati simbolicamente facendo uso di microistruzioni



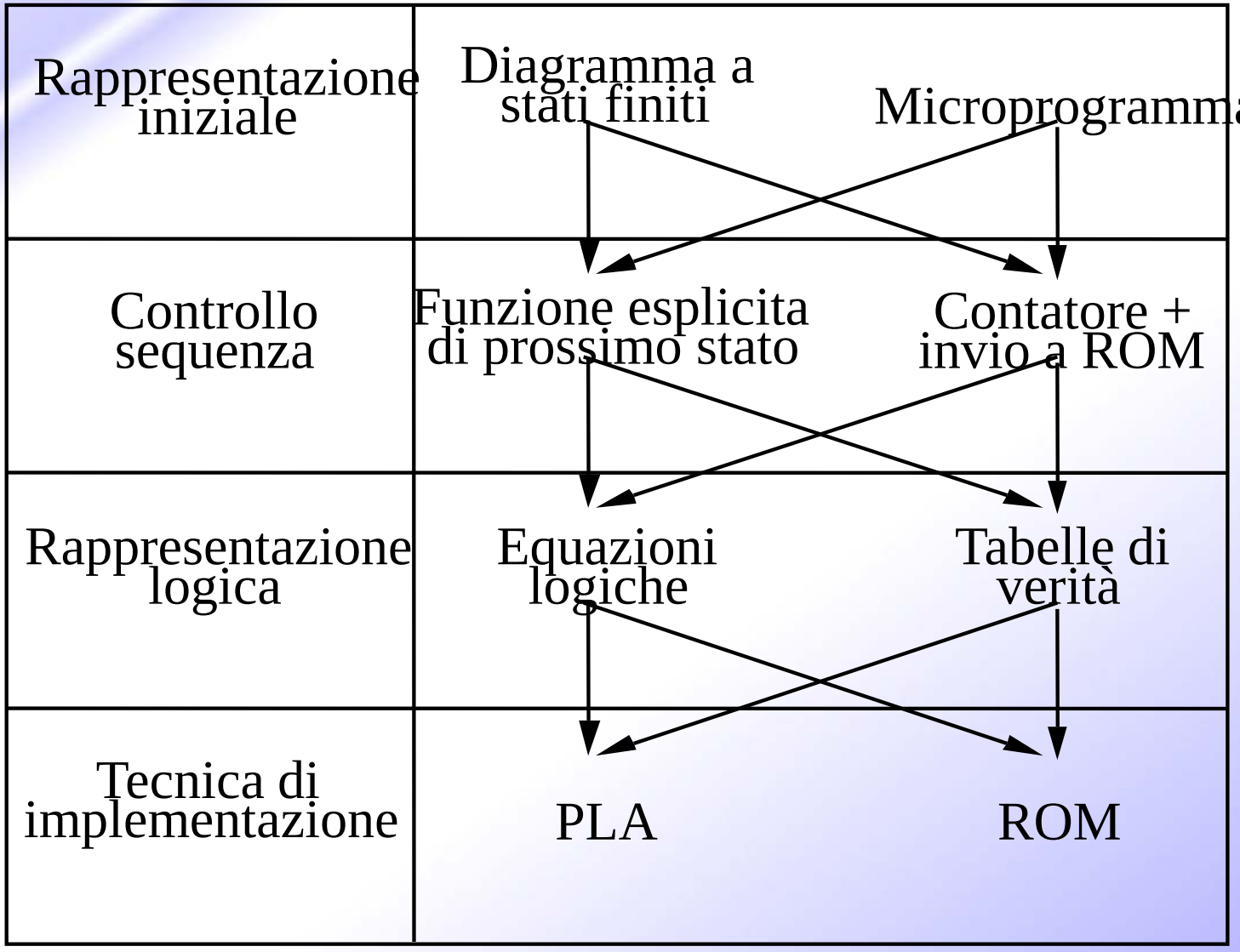
Formato microistruzione

Campo	Valore	Segnali attivi	Commento
ALU control	Add	ALUOp = 00	ALU effettua una somma
	Subt	ALUOp = 01	ALU effettua una sottrazione; permette di realizzare il confronto per salti condizionati.
	Funct code	ALUOp = 10	Il codice di funzione dell'istruzione determina il controllo dell'ALU
SRC1	PC	ALUSrcA = 0	Il PC è il primo input dell'ALU.
	A	ALUISrcA = 1	Il registro A è il primo input dell'ALU.
SRC2	B	ALUSrcB = 00	Il registro B è il secondo input dell'ALU.
	4	ALUSrcB = 01	4 è il secondo input dell'ALU.
	Extend	ALUSrcB = 10	L'uscita dell'unità di estensione in segno è il secondo input dell'ALU.
	Extshft	ALUSrcB = 11	L'uscita dell'unità di shift di 2 bit è il secondo input dell'ALU.
Register control	Read		Legge due registri usando rs ed rt dell'IR come numeri di registro e mette i dati nei registri A e B.
	Write ALU	RegWrite, RegDst = 1, MemtoReg = 0	Scrive un registro usando rd dell'IR come numero di registro ed il contenuto di ALUOut come dato.
	Write MDR	RegWrite, RegDst = 0, MemtoReg = 1	Scrive un registro usando rt dell'IR come numero di registro ed il contenuto di MDR come dato.
Memory	Read PC	MemRead, IorD = 0	Legge la memoria usando PC come indirizzo; scrive il risultato in IR (e MDR).
	Read ALU	MemRead, IorD = 1	Legge la memoria usando ALUOut come indirizzo; scrive il risultato in MDR.
	Write ALU	MemWrite, IorD = 1	Scrive la memoria usando ALUOut come indirizzo ed il contenuto di B come dato.
PC write control	ALU	PCSource = 00 PCWrite	Scrive l'output dell'ALU nel PC.
	ALUOut-cond	PCSource = 01, PCWriteCond	Se lo Zero dell'ALU è attivo, scrive nel PC il contenuto di ALUOut.
	jump address	PCSource = 10, PCWrite	Scrive nel PC l'indirizzo di salto dell'istruzione.
Sequencing	Seq	AddrCtl = 11	Sceglie la prossima microistruzione sequenzialmente.
	Fetch	AddrCtl = 00	Va alla prima microistruzione che incomincia una nuova istruzione.
	Dispatch 1	AddrCtl = 01	Invia usando la ROM 1.
	Dispatch 2	AddrCtl = 10	Invia usando la ROM 2.

Il microprogramma

Label	ALU control	SRC1	SRC2	Regsiter control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Funct code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

Il punto


Rappresentazione iniziale
Diagramma a stati finiti
Microprogramma
Controllo sequenza
Funzione esplicita di prossimo stato
Contatore + invio a ROM
Rappresentazione logica
Equazioni logiche
Tabelle di verità
Tecnica di implementazione
PLA
ROM

Eccezioni

- Eventi che modificano il normale flusso di esecuzione delle istruzioni
- Eccezione:
  - evento inaspettato dall'interno del processore
  - esempio: overflow aritmetico
- Interrupt:
  - evento che proviene dall'esterno del processore
  - esempio: dispositivo I/O che vuole comunicare
- Nella nostra implementazione:
  - istruzione non definita
  - overflow aritmetico

Implementazione

- Due registri addizionali:
  - EPC:
    - registro a 32 bit per memorizzare l'indirizzo dell'istruzione che ha causato l'eccezione
  - Cause:
    - registro a 32 bit:
      - bit meno significativo=0 => istruzione non definita
      - bit meno significativo=1 => overflow aritmetico
- Tre nuovi segnali di controllo:
  - EPCWrite: abilita scrittura EPC
  - CauseWrite: abilita scrittura Cause
  - IntCause: pone in modo appropriato il bit meno significativo di Cause

Cammino dei dati e controllo

- Multiplexer di input a PC aumentato a 4 input:
  - PC può diventare l'indirizzo della procedura che gestisce l'eccezione
- Multiplexer di input a Cause
  - 0 oppure 1
- EPC collegato ad ALUout:
  - PC è stato aumentato di 4 e quindi va diminnuito nuovamente
- Controllo: