UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Datapath e controllo: 3 istruzioni complete

Prof.ssa Giulia Cisotto

giulia.cisotto@units.it

Trieste, 15 aprile 2025

ORGANIZZAZIONE (1/2)

Istruzioni da studiare/Gruppi:

Gruppo 1 (5): `jr $rs`

Gruppo 2 (4): `addi $rt $rs, imm`

Gruppo 3 (5): `swap $rs $rt`

Ruoli nel gruppo (#persone):

Operatore (1): come funziona l'istruzione assegnata, osservare le differenze tra il datapath multi-ciclo e singolo-ciclo (fatto ieri a lezione), rivedere unità funzionali (ALU, PC, Register File, ..).

Controllore (1+1*): rivedere i segnali di controllo per le varie parti del datapath (sia Control Unit che ALU Control) e capire se servono ulteriori segnali di controllo (modifiche sul datapath).

+1* per i gruppi da 5

Macchinisti (2): prendere confidenza con la visualizzazione di una macchina a stati finiti (FSM) dove ogni stato equivale ad una fase dell'esecuzione dell'istruzione (e anche ad un ciclo di clock).

ORGANIZZAZIONE (2/2)

Tempistiche

10’ istruzioni

15’ studio individuale in base al gruppo e al ruolo

20’ condivisione nel gruppo con l’obbiettivo di sapere tutti come funziona la propria istruzione, la FSM associata ed eventuali modifiche al datapath

15’ x3 PEER-TEACHING tra gruppi. Ogni gruppo a rotazione si divide e va ad insegnare la propria istruzione agli altri (2 persone per ogni gruppo)

10’ chiusura

UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Materiale per studio individuale (comune ai gruppi)

DATAPATH MULTI-CICLO (1/2): METODOLOGIA DI CLOCKING

Singolo ciclo

- Ciclo singolo di lunghezza fissa uguale al tempo necessario per eseguire l'istruzione più lunga
- Ogni istruzione viene eseguita in un ciclo di clock
- *Svantaggi*:
  - Istruzioni potenzialmente più veloci sono rallentate (spreco di tempo)
  - Unità funzionali replicate (ad es. memoria, ALU)

Multi-ciclo

- Ciclo di lunghezza fissa più corto
- Ogni istruzione viene eseguita in più cicli di clock
- Istruzioni di tipo diverso vengono eseguite in un numero di cicli di clock diverso
- Le unità funzionali vengono usate più volte durante l'esecuzione della stessa istruzione in cicli di clock differenti -> meno replicazione
- Si usano registri aggiuntivi per memorizzare i risultati parziali nell'esecuzione delle istruzioni

DATAPATH MULTI-CICLO (2/2): DIFFERENZE ARCHITETTURALI

Registri aggiuntivi:

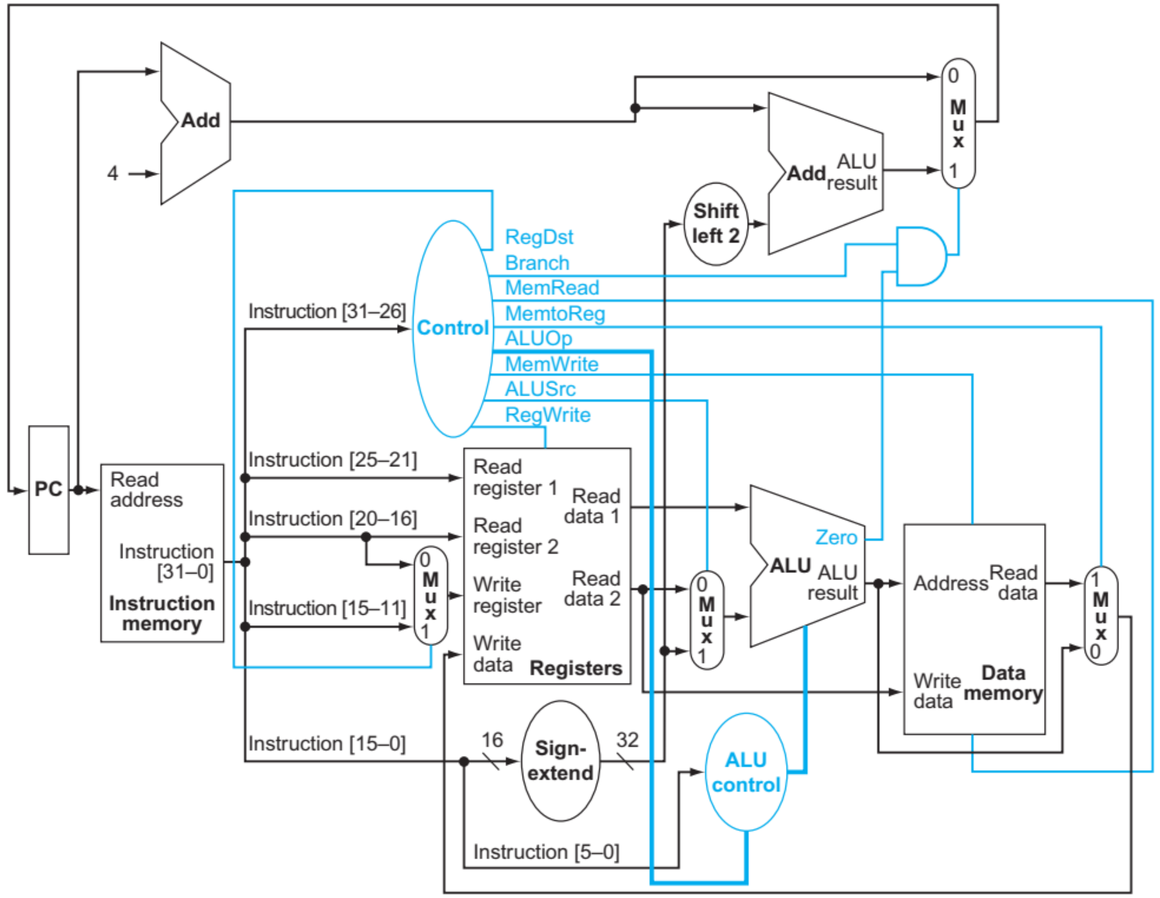
Memorizzano valori intermedi che vengono usati nel ciclo di clock successivo per continuare l'esecuzione della stessa istruzione:

- IR - Instruction Register
- MDR - Memory Data Register
- A, B - Registri tra Register File e l'ingresso ALU
- ALUout - L’output della ALU

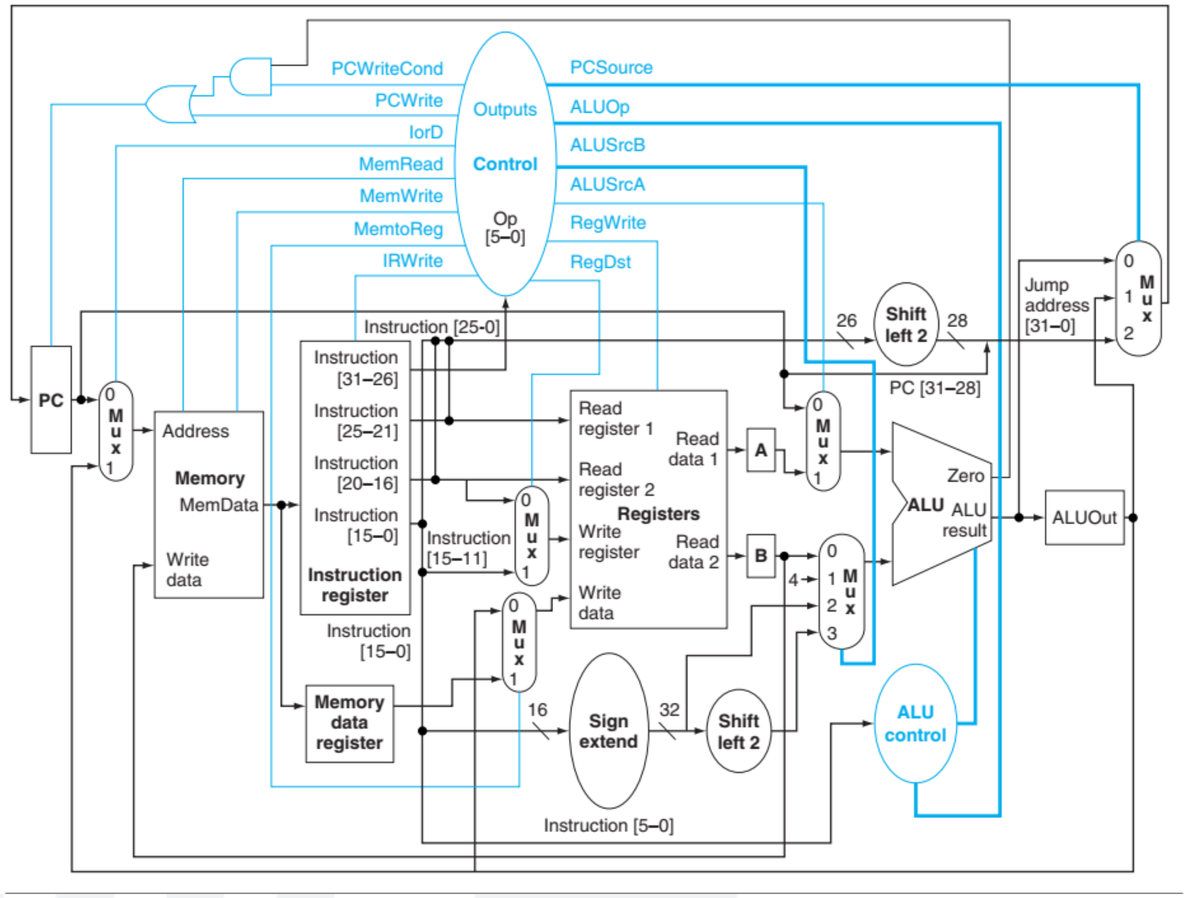
Riutilizzo di unità funzionali:

- ALU usata non solo per le operazioni aritmetico-logiche ma anche per calcolare l’indirizzo dei salti e per incrementare il PC
- Memoria usata sia per leggere le istruzioni che per leggere/scrivere i dati

DATAPATH SINGOLO-CICLO VS MULTI-CICLO (CON CONTROLLO)



Datapath singolo-ciclo (slide 19, lezione di ieri)



Datapath multi-ciclo (Cap5, fig. 5.28)

CONTROLLORE

- Video da guardare fino al minuto 10': https://www.youtube.com/watch?v=lqHKJyYCkXk

Actions of the 1-bit control signals

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register file destination number for the Write register comes from the rt field.	The register file destination number for the Write register comes from the rd field.
RegWrite	None.	The general-purpose register selected by the Write register number is written with the value of the Write data input.
ALUSrcA	The first ALU operand is the PC.	The first ALU operand comes from the A register.
MemRead	None.	Content of memory at the location specified by the Address input is put on Memory data output.
MemWrite	None.	Memory contents at the location specified by the Address input is replaced by value on Write data input.
MemtoReg	The value fed to the register file Write data input comes from ALUOut.	The value fed to the register file Write data input comes from the MDR.
IorD	The PC is used to supply the address to the memory unit.	ALUOut is used to supply the address to the memory unit.
IRWrite	None.	The output of the memory is written into the IR.
PCWrite	None.	The PC is written; the source is controlled by PCSource.
PCWriteCond	None.	The PC is written if the Zero output from the ALU is also active.

Actions of the 2-bit control signals

Signal name	Value (binary)	Effect
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the B register.
	01	The second input to the ALU is the constant 4.
	10	The second input to the ALU is the sign-extended, lower 16 bits of the IR.
	11	The second input to the ALU is the sign-extended, lower 16 bits of the IR shifted left 2 bits.
PCSource	00	Output of the ALU (PC + 4) is sent to the PC for writing.
	01	The contents of ALUOut (the branch target address) are sent to the PC for writing.
	10	The jump target address (IR[25:0] shifted left 2 bits and concatenated with PC + 4[31:28]) is sent to the PC for writing.

FIGURE 5.29 The action caused by the setting of each control signal in Figure 5.28 on page 323. The top table describes the 1-bit control signals, while the bottom table describes the 2-bit signals. Only those control lines that affect multiplexors have an action when they are deasserted. This information is similar to that in Figure 5.16 on page 306 for the single-cycle datapath, but adds several new control lines (IRWrite, PCWrite, PCWriteCond, ALUSrcB, and PCSource) and removes control lines that are no longer used or have been replaced (PCSrc, Branch, and Jump).

PASSO 1: FETCH (COMUNE A TUTTE LE ISTRUZIONI)

Operazioni

IR <- M [PC]

PC <- PC+4

Segnali di controllo

- Per leggere della memoria: MemRead
- Per scrivere IR: IRWrite
- Per indicare l indirizzo ’indirizzo da dove leggere dalla memoria: IorD
- Per incrementare il PC: ALUSrcA, ALUSrcB, ALUop
- Per salvare il nuovo valore del PC in PC: PCWrite

Cap5 (pp.325-329)

PASSO 2: DECODE (COMUNE A TUTTE LE ISTRUZIONI)

Operazioni

A <- Reg[IR[25:21]]

B <- Reg[IR[20:16]]

ALUOut <- PC+(sign-extend (IR[15:0]) << 2

Segnali di controllo

- Per il calcolo di un eventuale indirizzo di branch:

ALUSrcA, ALUSrcB, ALUop

Cap5 (pp.325-329)

PASSO 3: EXECUTE (PER ISTRUZIONI LW/SW)

Operazioni

ALUOut <- A+(sign-extend (IR[15:0])

Segnali di controllo

- Per il calcolo dell' indirizzo di memoria per lw o sw:

ALUSrcA, ALUSrcB, ALUop

Cap5 (pp.325-329)

PASSO 3: EXECUTE (PER ISTRUZIONI R-TYPE ARITMETICO-LOGICHE)

Operazioni

ALUOut <- A op B

Segnali di controllo

- Per il calcolo aritmetico o logico: ALUSrcA, ALUSrcB, ALUop

Cap5 (pp.325-329)

PASSO 3: EXECUTE (PER ISTRUZIONI BEQ)

Operazioni

if A == B then PC <- ALUOut

Segnali di controllo

- Per la comparazione tra A e B: ALUSrcA, ALUSrcB, ALUOp
- Per scrivere PC: PCWriteCond, PCSource

Cap5 (pp.325-329)

PASSO 3: EXECUTE (PER ISTRUZIONI JUMP)

Operazioni

PC <- (PC[31:28] IR[25:0]<<2) , IR[25:0]<<2)

Segnali di controllo

- Per scrivere PC: PCWriteCond, PCSource

Cap5 (pp.325-329)

PASSO 4: EXECUTE (PER ISTRUZIONI LW/SW)

Operazioni

MDR <- M[ALUOut]

OPPURE

M[ALUOut] <- B

Segnali di controllo

- Per indicare l'indirizzo di memoria: IorD
- Per leggere dalla memoria (lw): MemRead
- Per scrivere nella memoria (sw): MemWrite

Cap5 (pp.325-329)

PASSO 4: EXECUTE (PER ISTRUZIONI R-TYPE ARITMETICO-LOGICHE)

Operazioni

Reg[IR[15:11]] <- ALUOut

Segnali di controllo

- Per poter scrivere nel Register File: RegWrite
- Per indicare il registro da scrivere: RegDest
- Per scrivere il valore nel ALUOut: MemToReg

Cap5 (pp.325-329)

PASSO 5: EXECUTE (PER ISTRUZIONI LW)

Operazioni

Reg[IR[20:16]] <- MDR

Segnali di controllo

- Per poter scrivere nel Register File: RegWrite
- Per indicare il registro da scrivere: RegDest
- Per scrivere il valore dalla memoria: MemToReg

Cap5 (pp.325-329)

TUTTI I PASSI DELLE VARIE ISTRUZIONI NOTE

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) PC <= ALUOut	PC <= {PC [31:28], (IR[25:0]],2'b00)}
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		

FIGURE 5.30 Summary of the steps taken to execute any instruction class. Instructions take from three to five execution steps. The first two steps are independent of the instruction class. After these steps, an instruction takes from one to three more cycles to complete, depending on the instruction class. The empty entries for the Memory access step or the Memory read completion step indicate that the particular instruction class takes fewer cycles. In a multicycle implementation, a new instruction will be started as soon as the current instruction completes, so these cycles are not idle or wasted. As mentioned earlier, the register file actually reads every cycle, but as long as the IR does not change, the values read from the register file are identical. In particular, the value read into register B during the Instruction decode stage, for a branch or R-type instruction, is the same as the value stored into B during the Execution stage and then used in the Memory access stage for a store word instruction.

Cap5 (pp.325-329)

FASI DELL'ESECUZIONE DI UN'ISTRUZIONE

- **Fetch**: fetch istruzione + incremento PC
- **Decode**: decodifica istruzione + lettura registri + calcolo dell'indirizzo per un eventuale branch
- **Execute**: esecuzione operazioni relative a R-type *OPPURE* calcolo di memoria *OPPURE* completa branch *OPPURE* completa jump
  - Execute: completa R-type *OPPURE* accesso alla memoria
  - Execute: scrittura registro (solo per l'istruzione lw)

L'istruzione più lunga si esegue in 5 passi

L'istruzione più corta si esegue in 3 passi

DIAGRAMMA DEGLI STATI (ALIAS: MACCHINA A STATI FINITI)

Visione di alto livello

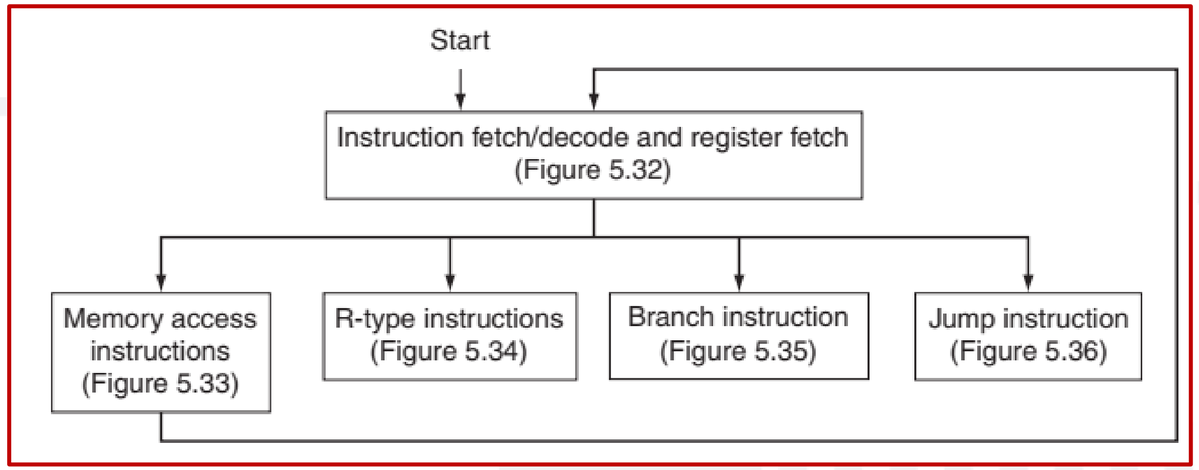
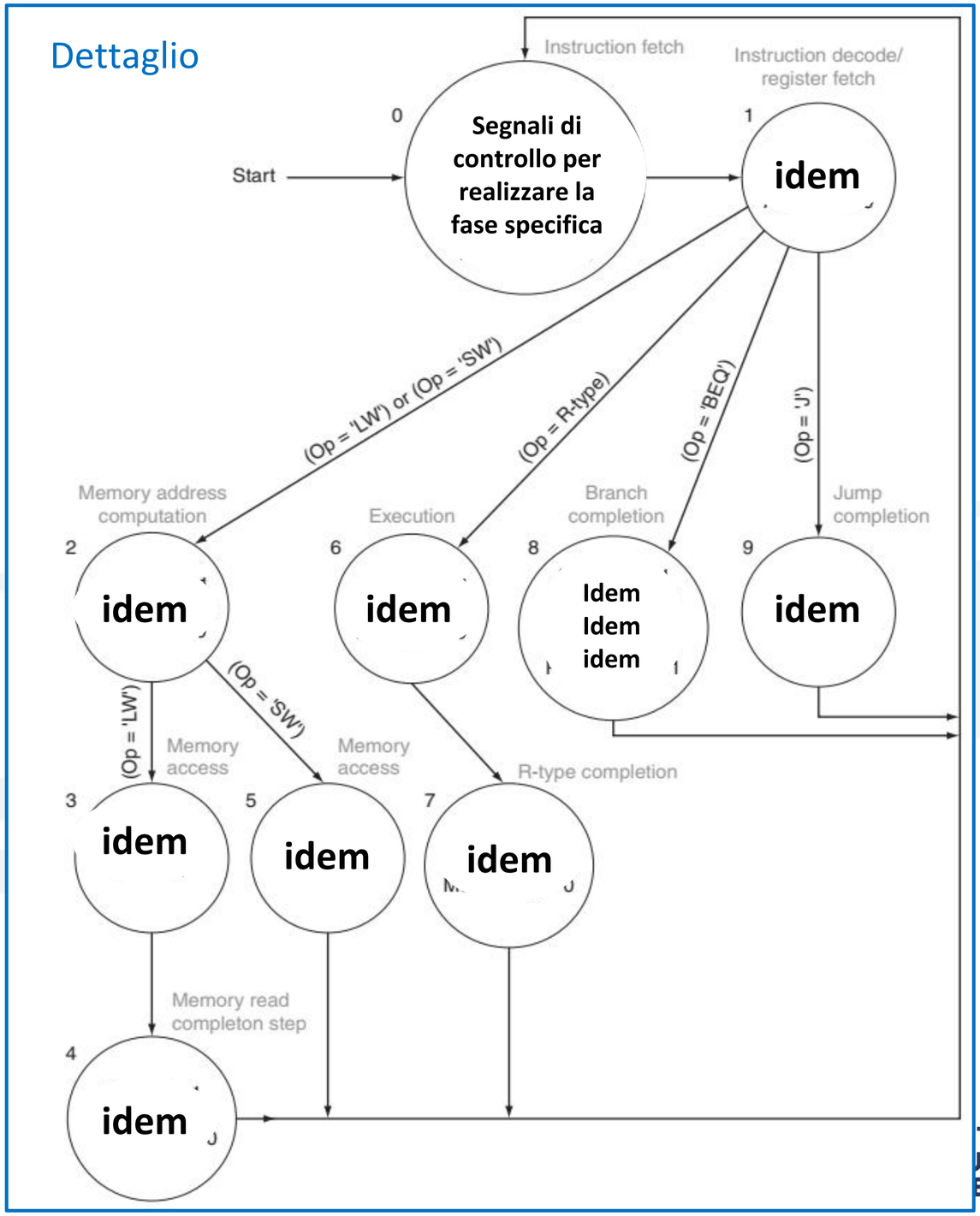


FIGURE 5.38 The complete finite state machine control for the datapath shown in Figure 5.28. The labels on the arcs are conditions that are tested to determine which state is the next state; when the next state is unconditional, no label is given. The labels inside the nodes indicate the output signals asserted during that state; we always specify the setting of a multiplexor control signal if the correct operation requires it. Hence, in some states a multiplexor control will be set to 0.

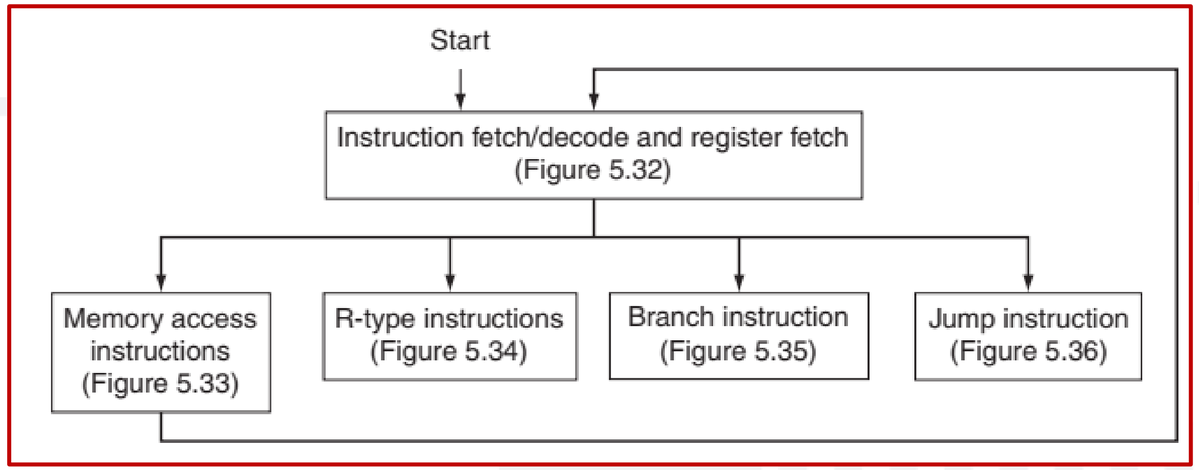
Dettaglio



Cap5 (p.331,p.339)

DIAGRAMMA DEGLI STATI (ALIAS: MACCHINA A STATI FINITI)

Visione di alto livello



Dettaglio

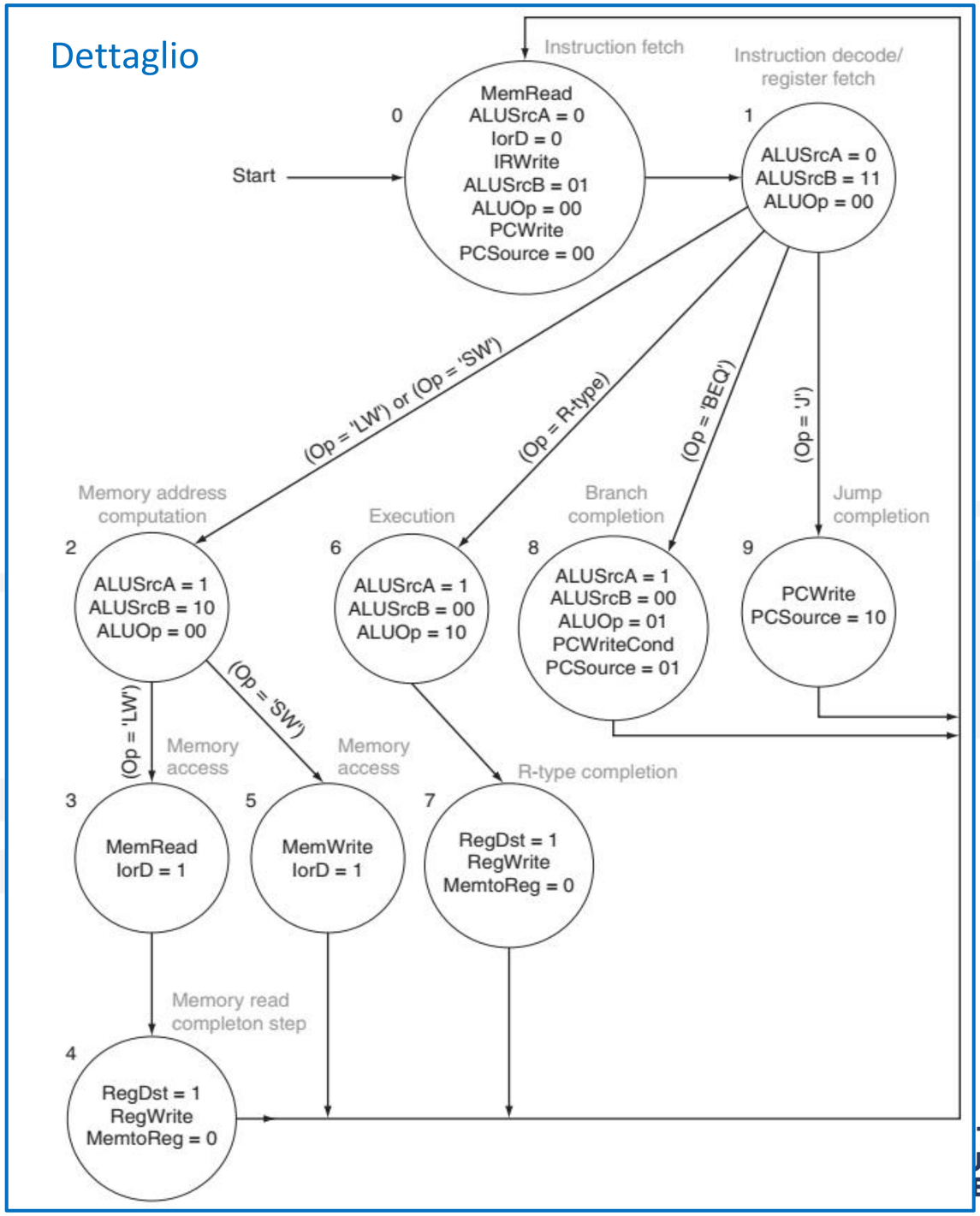


FIGURE 5.38 The complete finite state machine control for the datapath shown in Figure 5.28. The labels on the arcs are conditions that are tested to determine which state is the next state; when the next state is unconditional, no label is given. The labels inside the nodes indicate the output signals asserted during that state; we always specify the setting of a multiplexor control signal if the correct operation requires it. Hence, in some states a multiplexor control will be set to 0.

Cap5 (p.331,p.339)

UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

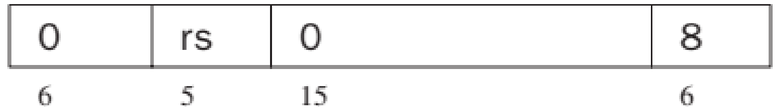
Gruppo 1: jump register

JUMP REGISTER: FORMATO ISTRUZIONE

OPERATORE

Jump register

`jr rs`



Unconditionally jump to the instruction whose address is in register `rs`.

Cosa fa?

Istruzione di tipo: R-type (usa campo funct)
Aggiorna il PC con i 32 bit contenuti nel registro il cui indirizzo è indicato dai 5 bit [25:21] dell'istruzione

1. **Legge il valore del registro $rs**
2. **Copia quel valore nel PC, quindi il flusso di esecuzione salta a quell'indirizzo**

Note:
- nessun offset
- nessun calcolo con ALU
- usa direttamente un indirizzo già presente in un registro

(p.A-63)

JUMP REGISTER: NOTE ULTERIORI

CONTROLLORE

Tenendo conto che per ogni istruzione, **nella fase di decode**, viene scritto

- in A il contenuto del registro il cui indirizzo è indicato dai 5 bit [25:21] dell'istruzione
- in B il contenuto del registro il cui indirizzo è indicato dai 5 bit [20:16] dell'istruzione

Impostare i **segnali di controllo** in modo tale da scrivere in PC il risultato della somma tra il contenuto del registro A e il registro B:

- abilitare la scrittura del PC (PCWrite=1)
- selezionare l'output della ALU come valore da scrivere nel PC (PCSource=0)
- selezionare il registro A come primo input per la ALU (ALUScrA=1)
- selezionare il registro B come secondo input per la ALU (ALUScrB=00)
- sommare (ALUOp=00)

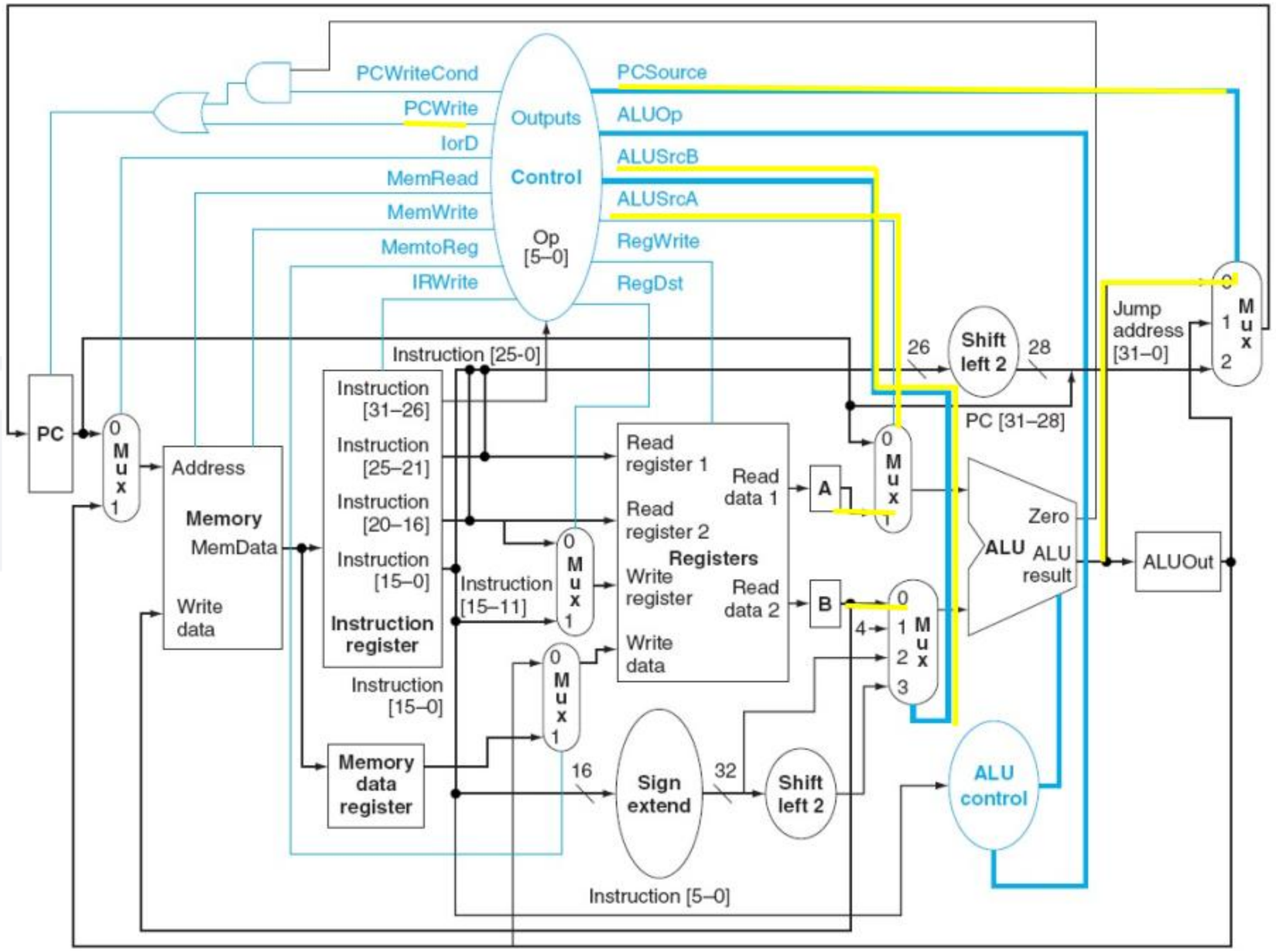
Una possibile soluzione per l'istruzione assegnata: https://www.youtube.com/watch?v=pWVw70W6ypA

JUMP REGISTER: DATAPATH MULTI-CICLO

OPERATORE

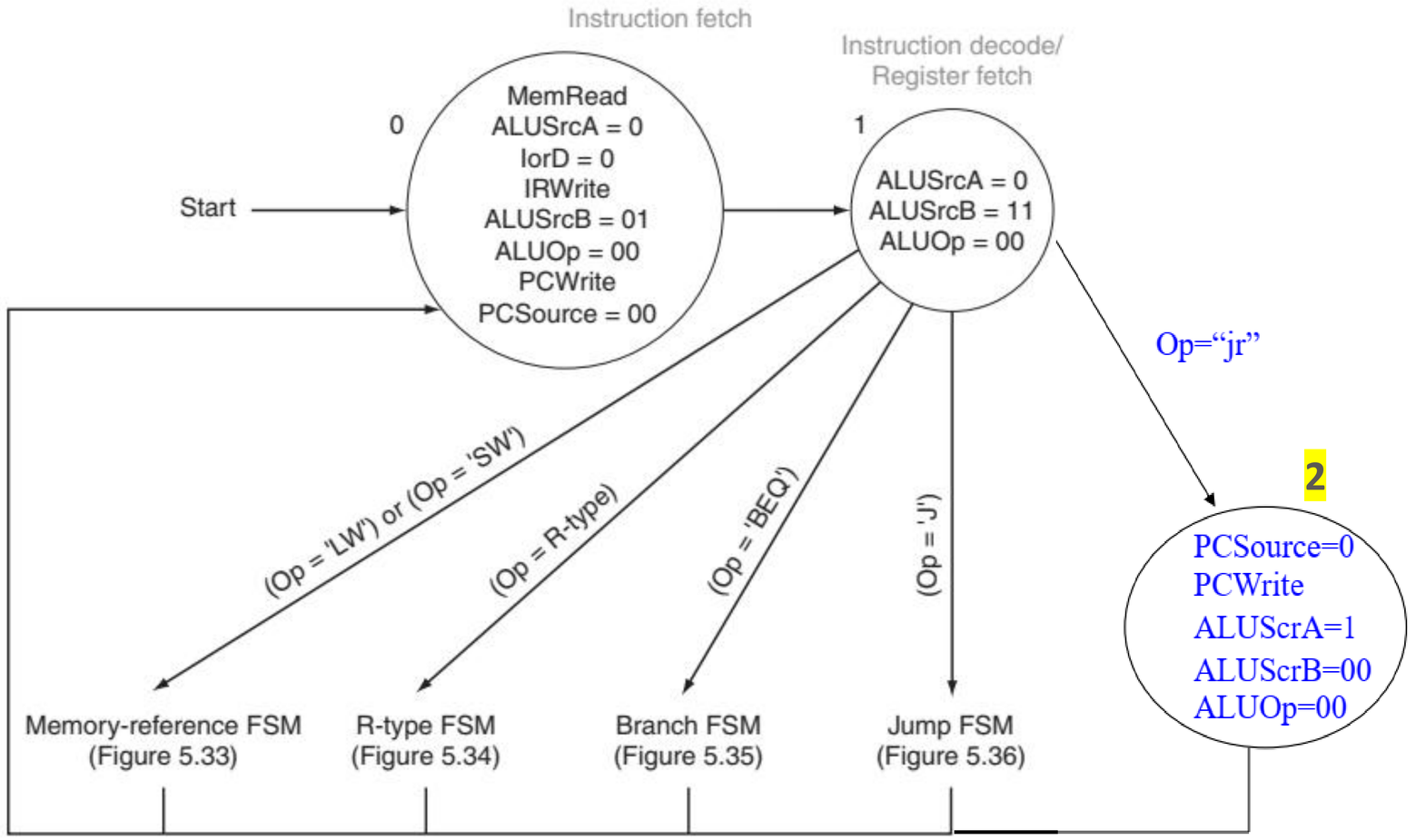
CONTROLLORE

Stato 2



JUMP REGISTER: MACCHINA A STATI FINITI

MACCHINISTA



Gruppo 2: add immediate

ADDI: FORMATO ISTRUZIONE

OPERATORE

Addition immediate (with overflow)

```
addi rt, rs, imm
```

Cosa fa?

Istruzione di tipo: I-type

Somma un immediato (con segno, in CA2) al registro $rs e salva il risultato in $rt.
Nel dettaglio:

1. **Legge il valore di $rs**
2. **Estende con segno l'immediato (16 → 32 bit)**
3. **Somma: $rs + imm**
4. **Scrive il risultato in $rt**

Note:
- immediato in CA2 (positivo/negativo)
- usa ALU (add)
- non accede alla memoria

Appendice A (p.A-51)

ADDI: DATAPATH MULTI-CICLO

OPERATORE

CONTROLLORE

Stato 2

Stato 3

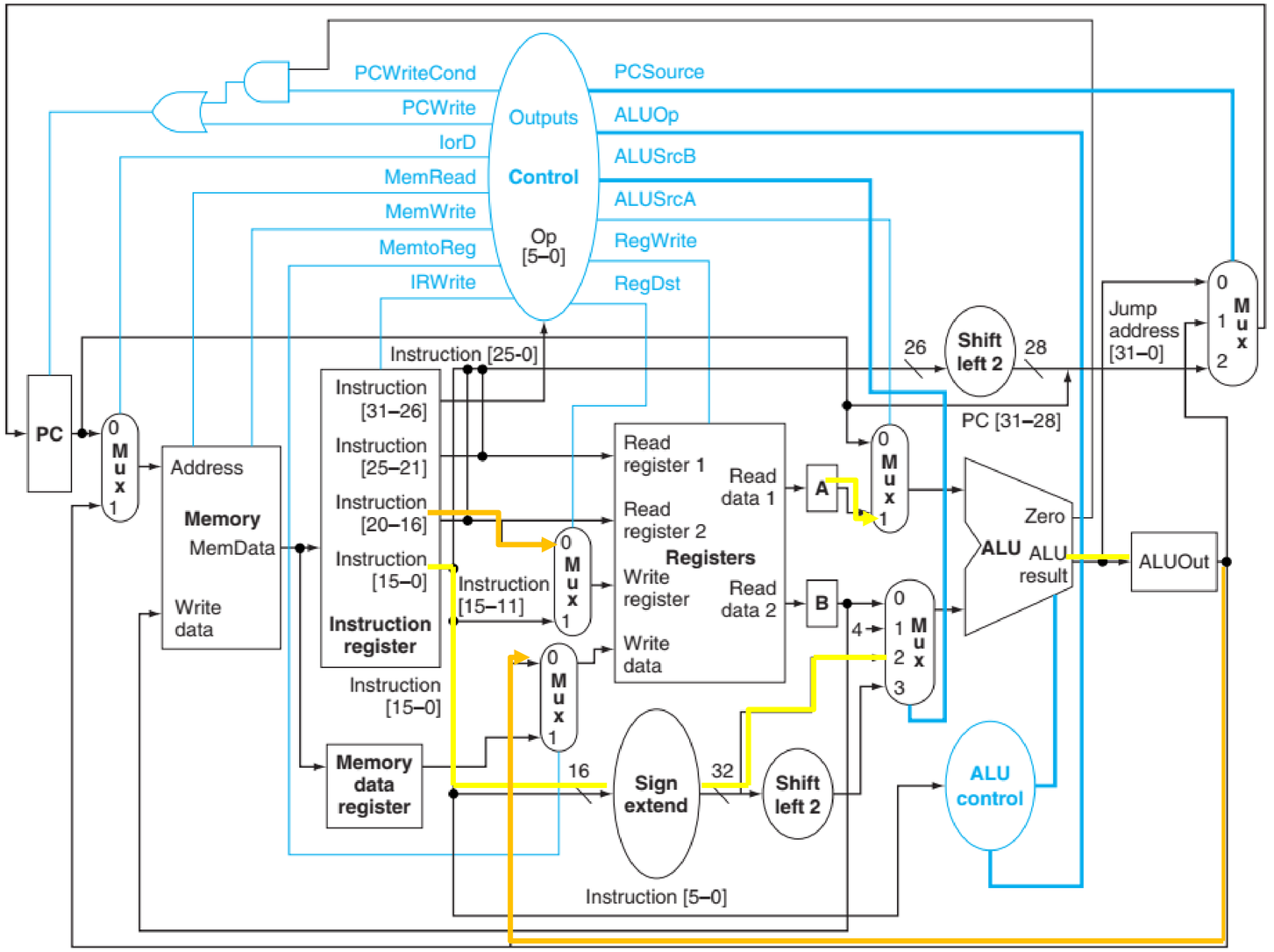
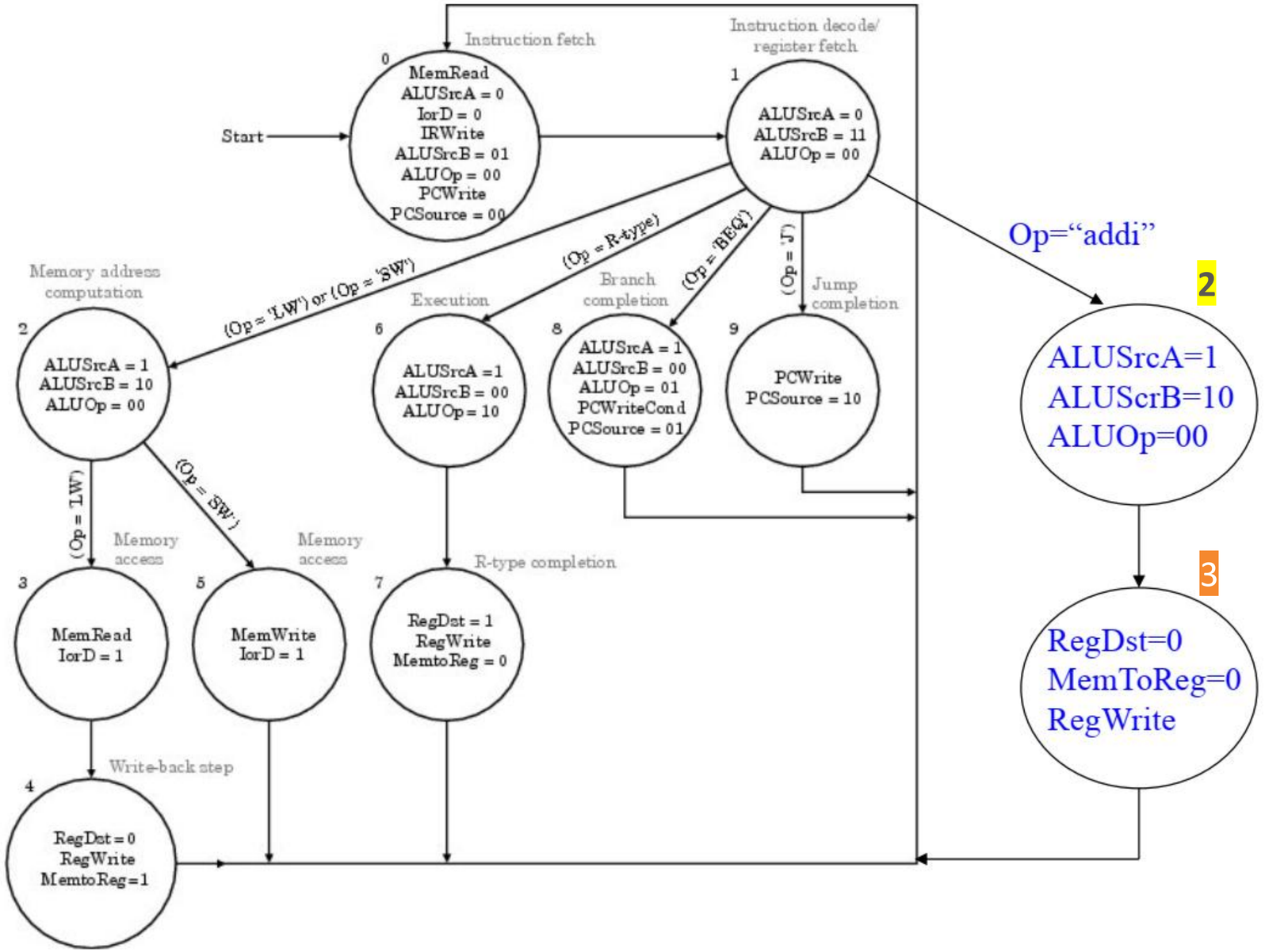


FIGURE 5.28 The complete datapath for the multicycle implementation together with the necessary control lines. The control lines of Figure 5.27 are attached to the control unit, and the control and datapath elements needed to effect changes to the PC are included. The major additions from Figure 5.27 include the multiplexor used to select the source of a new PC value; gates used to combine the PC write signals; and the control signals PCSource, PCWrite, and PCWriteCond. The PCWriteCond signal is used to decide whether a conditional branch should be taken. Support for jumps is included.

ADDI: MACCHINA A STATI FINITI

MACCHINISTA



UNIVERSITÀ DEGLI STUDI DI TRIESTE

UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Gruppo 3: swap

SWAP: FORMATO ISTRUZIONE

Non è codificata, ovvero non esiste nativamente in MIPS32.
Potrebbe essere una *pseudo istruzione* con la seguente sintassi:

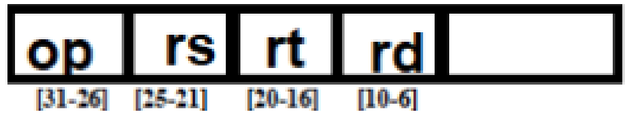
```
swap $rs $rt
```

Cosa fa?

Scambia il contenuto di due registri

$$R[rt] \leftarrow R[rs]$$
$$R[rs] \leftarrow R[rt]$$

L'istruzione potrebbe avere il seguente formato, con $rd contenente una copia di $rs (in modo da avere un backup):



Appendice A (p.A-51)

SWAP: NOTE ULTERIORI

OPERATORE

CONTROLLORE

L'istruzione viene implementata come:

A ← R[rs] # lettura da registro $rs e scrittura (immediata) nel registro temporaneo A

B ← R[rt] # lettura da registro $rt e scrittura (immediata) nel registro temporaneo B

WB1: R[rs] ← B # prima (sovra)scrittura, ovvero scrivo nel registro $rs

WB2: R[rt] ← A # seconda (sovra)scrittura, ovvero scrivo nel registro $rt

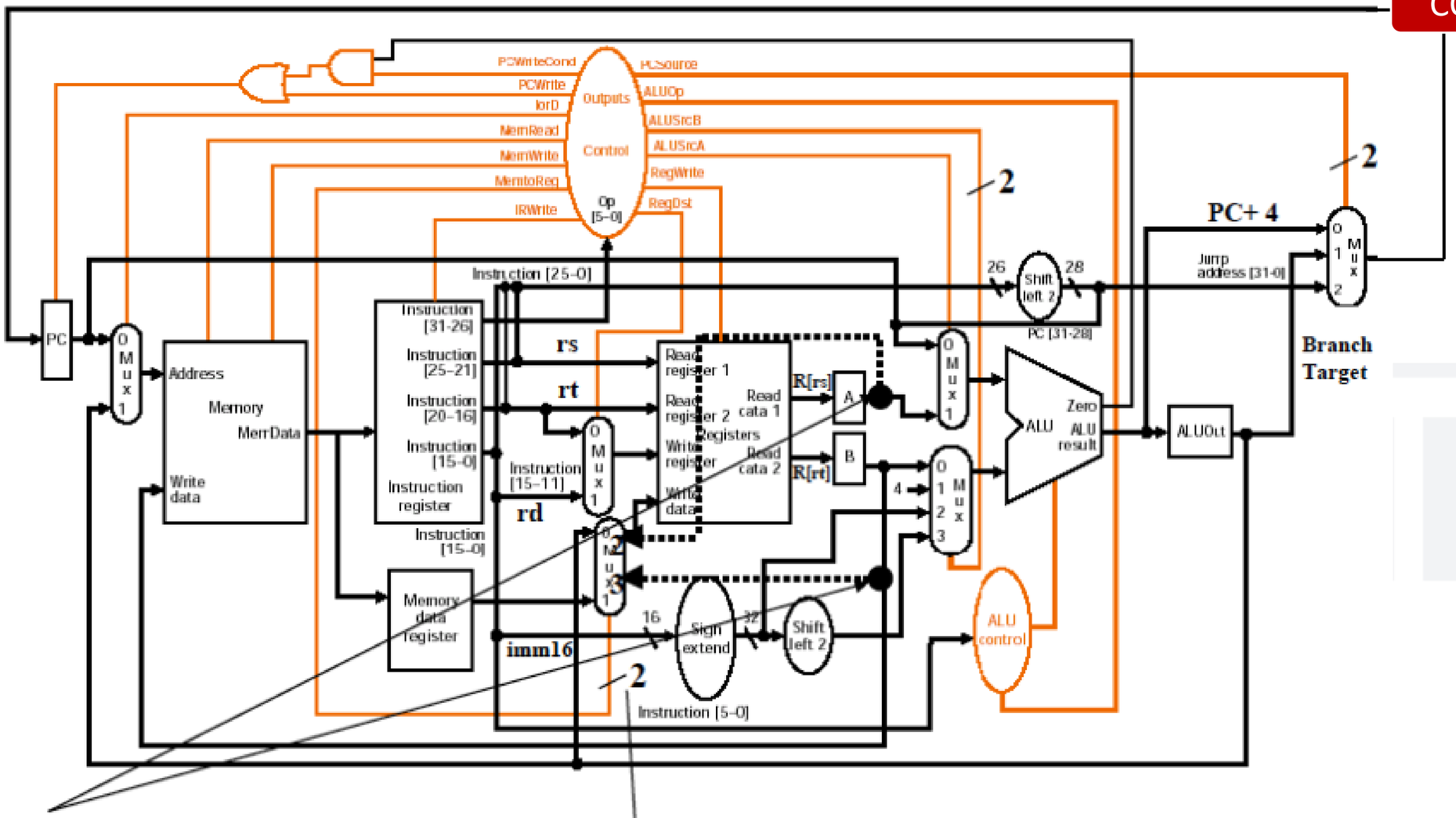
Note:

- usa solo operazioni ALU
- nessun accesso a memoria

SWAP: DATAPATH MULTI-CICLO

OPERATORE

CONTROLLORE

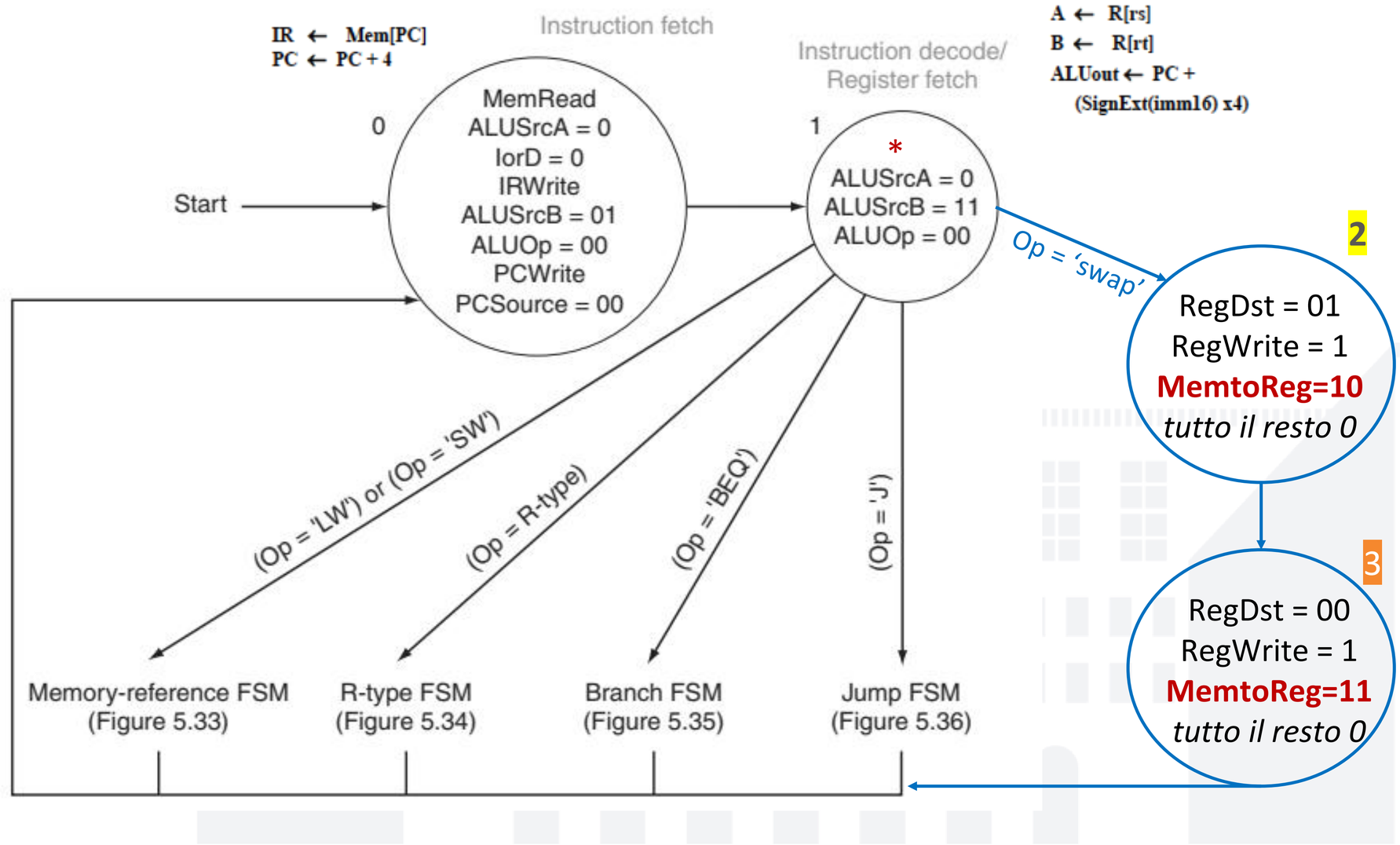


The outputs of A and B should be connected to the multiplexor controlled by MemtoReg if one of the two fi
(rs and rd) contains the name of one of the registers being swapped. The other register is specified by rt.
The MemtoReg control signal becomes two bits.

SWAP: MACCHINA A STATI FINITI

MACCHINISTA

*
RegWrite = 0
MemRead = 0
MemWrite = 0
IRWrite = 0
PCWrite = 0
PCWriteCond = 0



MemtoReg adesso diventa un segnale di controllo a 2 bit. Ha il compito di selezionare prima A (Stato 2=WriteBack1) e poi B (Stato 3=WriteBack2).

Materiale per la lezione

- Capitolo 5 (PDF a parte su Moodle): The processor: Datapath and Control

Prossima lezione: 16 ***aprile, h.14:00****, aula 4C*