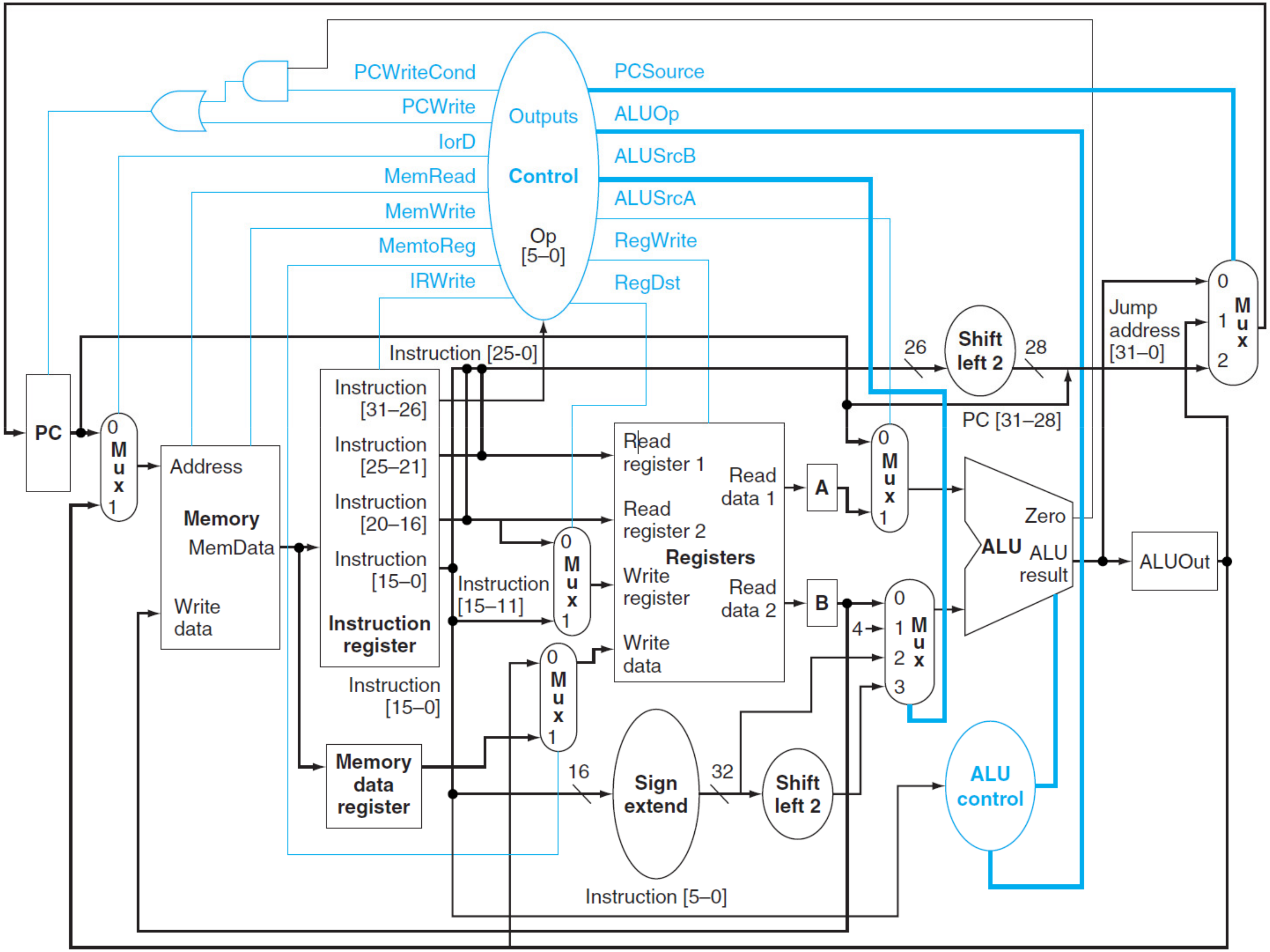
#10 : MIPS Multi-Cycle Implementation

Computer Architecture 2019/2020

João Soares & Ricardo Rocha

Computer Science Department, Faculty of Sciences, University of Porto

Multi-Cycle Data and Control Paths



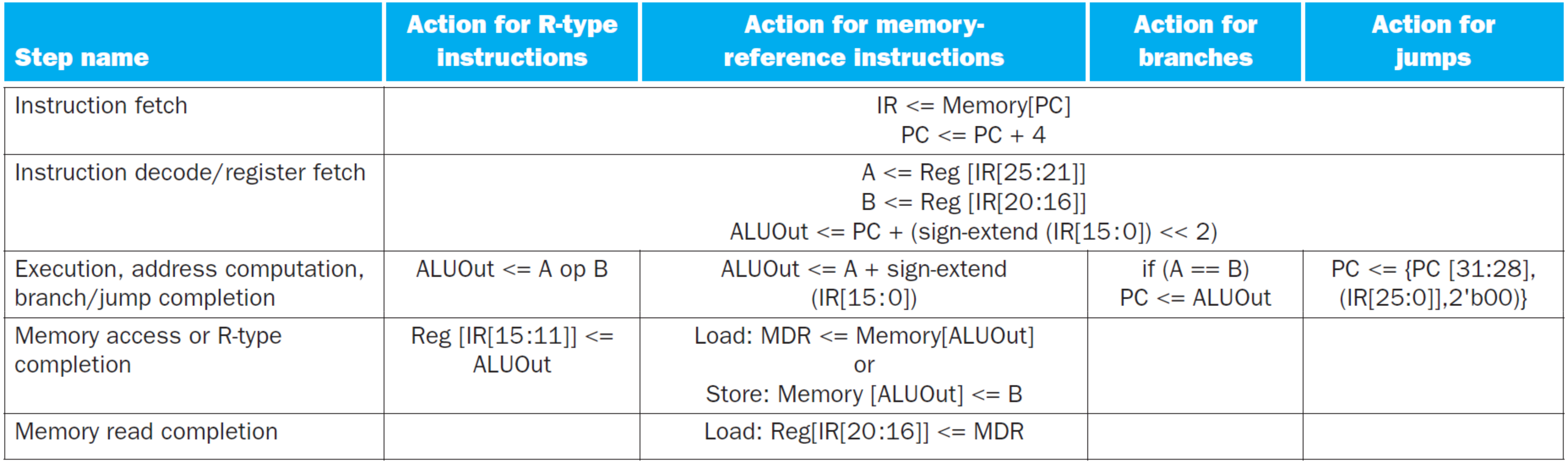
Multi-Cycle Steps

MIPS instructions classically include five steps:

- **IF** – instruction fetch
- **ID** – instruction decode and register fetch
- **EX** – execute operation , address calculation or branch/jump completion
- **MEM** – data memory access or R-type completion
- **WB** – write result back to register

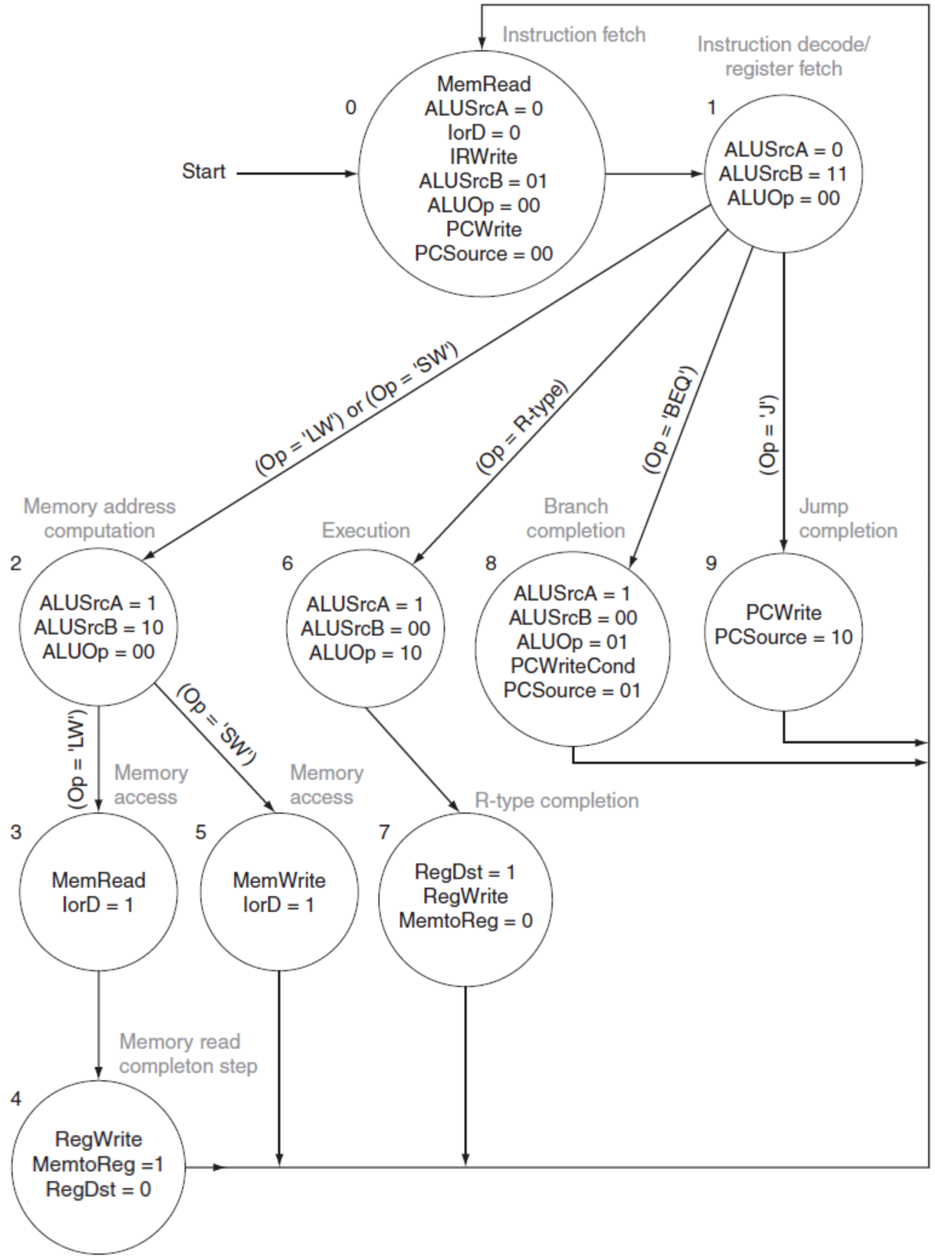
Instructions take from three to five execution steps. The first two steps are independent of the instruction type. After these steps, an instruction takes from one to three more cycles to complete, depending on the instruction type.

Multi-Cycle Step Actions



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		

Multi-Cycle Finite State Machine



Multi-Cycle Example

Example: **lw $t2, 100($t1)**   (I-format: op rs rt address --> **35 $t1 $t2 100**)

Step IF – machine state 0

- Signals ALUSrcA=0 / ALUSrcB=01 / ALUOp=00 / PCSource=00 / PCWrite
  PC <= PC + 4
- Signals IorD=0 / MemRead / IRWrite
  IR <= Memory[PC] = **35 $t1 $t2 100**

Step ID – machine state 1

- Signals ALUSrcA=0 / ALUSrcB=11 / ALUOp=00
  Ignored for load instructions
- A <= Reg[IR[25:21]] = **Reg[$t1]**
  B <= Reg[IR[20:16]] = **Reg[$t2]**

Multi-Cycle Example

Example: **lw $t2, 100($t1)** (I-format: op rs rt address --> **35 $t1 $t2 100**)

Step EX – machine state 2

- Signals ALUSrcA=1 / ALUSrcB=10 / ALUOp=00
  ALUOut <= A + sign-extend(IR[15:0]) = **Reg[$t1] + 100**

Step MEM – machine state 3

- Signals IorD=1 / MemRead
  MDR <= Memory[ALUOut] = **Memory[Reg[$t1] + 100]**

Step WB – machine state 4

- Signals RegDst=0 / RegWrite / MemtoReg=1
  Reg[IR[20:16]] = **Reg[$t2]** <= MDR = **Memory[Reg[$t1] + 100]**