Computer Organization
計算機組織

Designing a Multicycle Processor

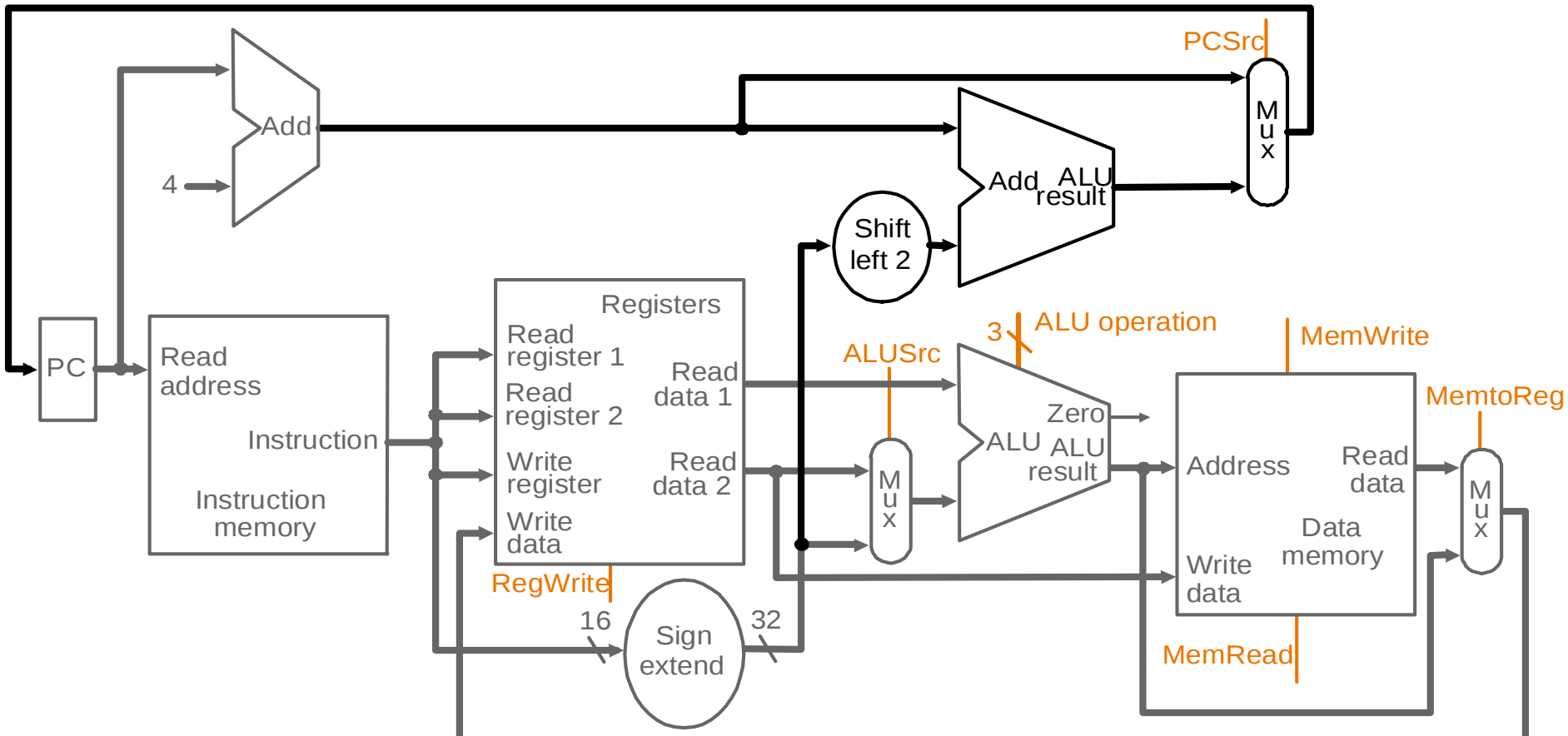
(Supplemental Material: Refer to Ch. 5.5 in 3rd Ed.)

國立成功大學資訊工程學系
100 年度第二學期

What to Learn

- **Comparison: single-cycle vs. multicycle**
- **Understand the design of a multicycle processor**
  - **Datapath**
  - **Control (e.g., FSM)**

Recap: A Single-Cycle Processor



CPI=1

Single-Cycle Design Problems

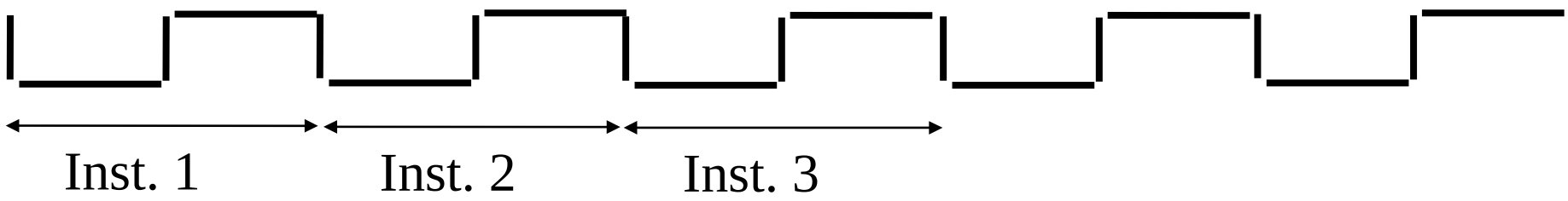
- **Fixed-period clock**
  - **CPI = 1**
  - **cycle time determined by length of longest instruction path (lw)**
    - **but several instructions could run in a shorter clock cycle**
      - *waste of time*
      - consider if we have more complicated instructions like floating point!
  - **resources used more than once in the same cycle need to be duplicated**
    - **waste of hardware and chip area**

Fixing the Problem with Single-Cycle Design

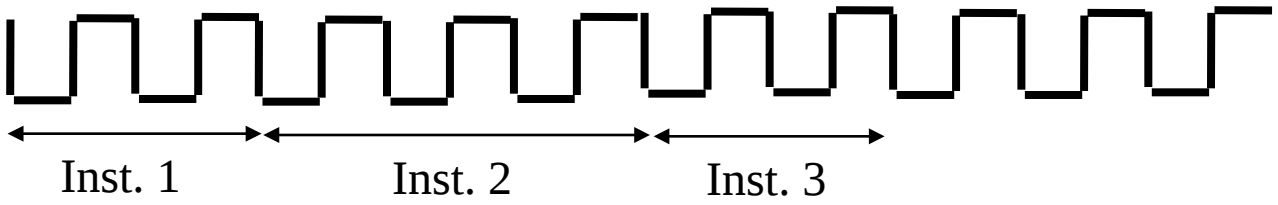
- **One of the solutions:**
  - **use a smaller cycle time…**
  - **…by have different instructions take different numbers of cycles**
  - **Can take advantage of the technology**
  - ***Multicyle approach*!**

Single-Cycle vs. Multi-Cycle

- **Fixed-period clock cycle (single cycle impl.)**

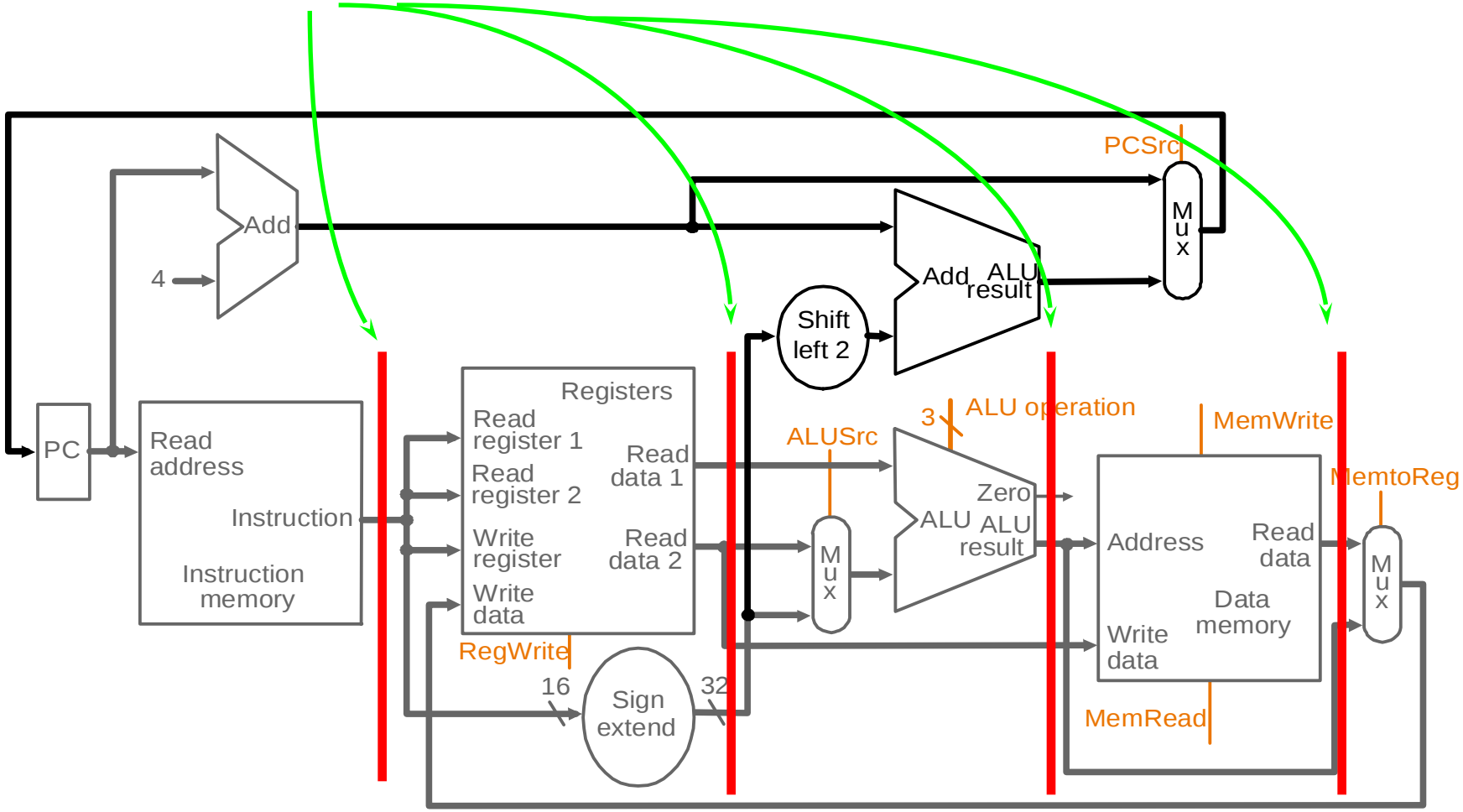


- **Variable-period clock cycle (multicycle impl.)**



Partition Single-Cycle Datapath (1/2)

- **Add registers between <u>smallest steps</u>**



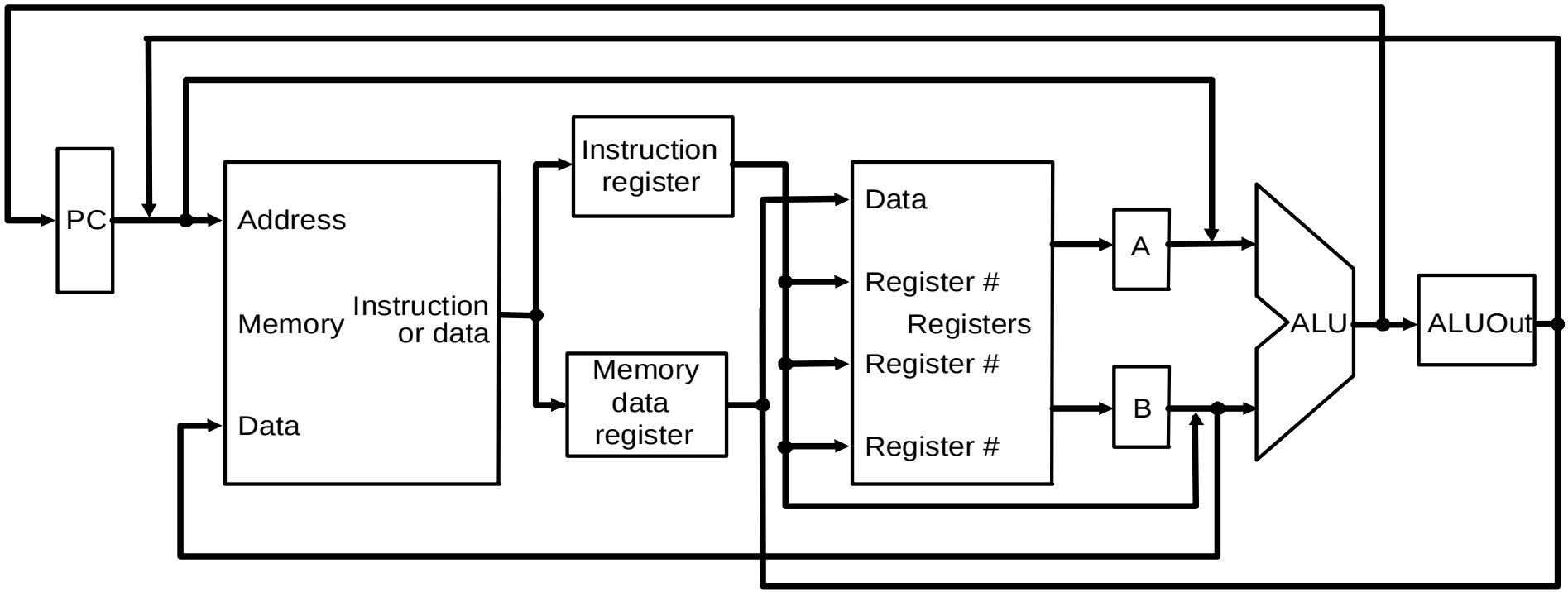
Partition Single-Cycle Datapath (2/2)

- **We break instructions into the following *potential* execution steps–not all instructions require all the steps–each step takes one clock cycle**
  1. **Instruction fetch and PC increment (IF)**
  2. **Instruction decode and register fetch (ID)**
  3. **Execution, memory address computation, or branch completion (EX)**
  4. **Memory access or R-type instruction completion (MEM)**
  5. **Memory read completion (WB)**

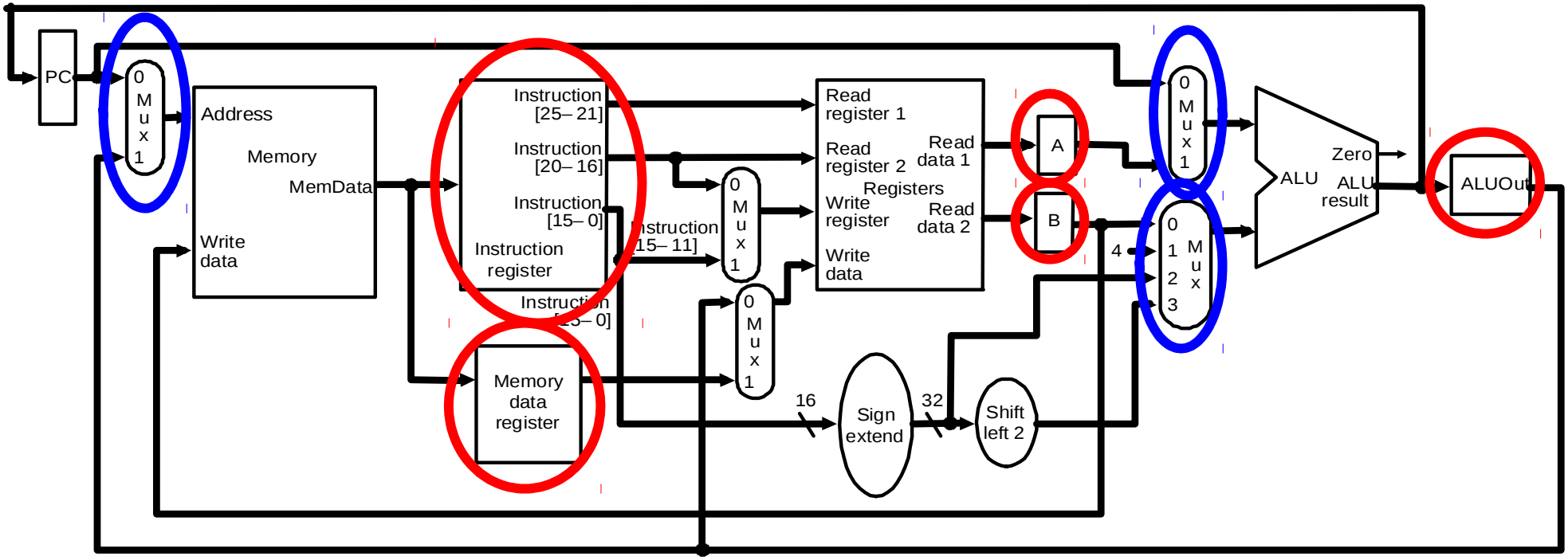
- **An MIPS instruction takes 3~5 cycles (steps)**
  - **Not every instruction has the same execution time!**

Multicycle Datapath (1/2)

- 1 memory (instr. & data), 1 ALU (addr, PC+4, add,…), registers (IR, MDR, A, B, ALUOut)



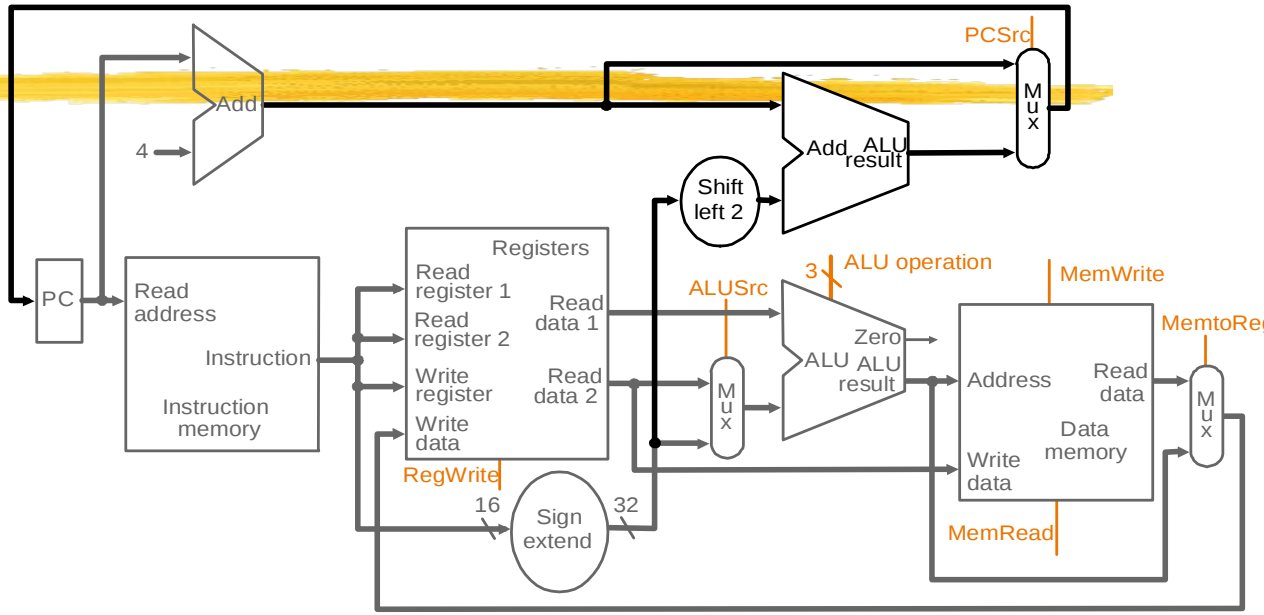
Multicycle Datapath (2/2)



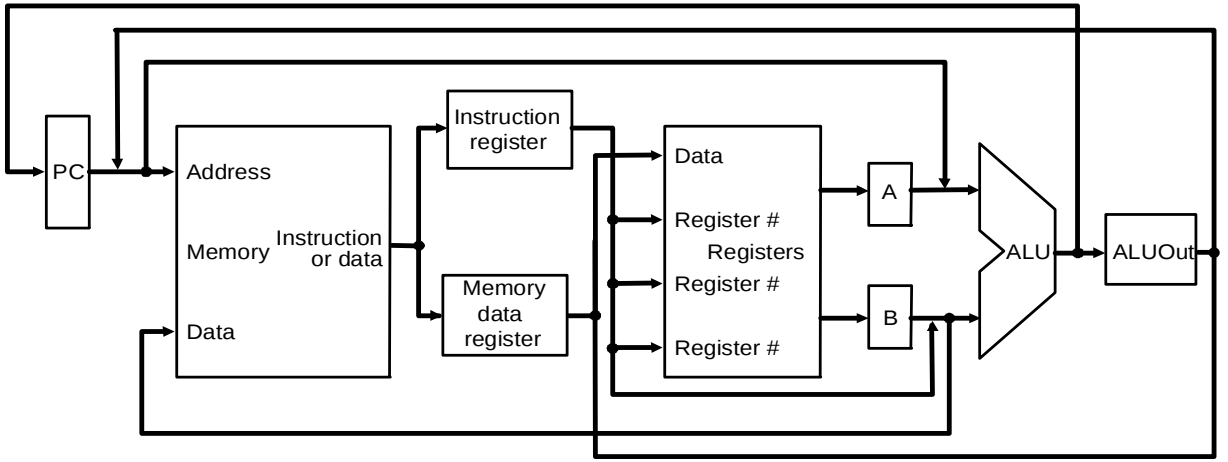
Basic multicycle MIPS datapath handles R-type instructions and load/stores (new internal register in red ovals, new multiplexors in blue ovals)

Single- vs. Multi-cycle Approach

- **Note particularities of multicyle vs. single-diagrams**
  - **single memory for data and instructions**
  - **single ALU, no extra adders**
  - **extra registers to hold data between clock cycles**



Single-cycle datapath



Multicycle datapath (high-level view)

Example: CPI in a Multicycle CPU

- **Assume: an instruction mix of 22% *loads*, 11% *stores*, 49% *R-type operations*, 16% *branches*, and 2% *jumps***

- **What is the CPI assuming each step requires 1 clock cycle?**
  - **Solution:**
    - **Number of clock cycles from previous slide for each instruction class:**
      - *loads* 5, *stores* 4, *R-type instructions* 4, *branches* 3, *jumps* 3
    - **CPI = CPU clock cycles / instruction count**
      = Σ (instruction count$_{\text{class i}}$ $\times$ CPI$_{\text{class i}}$) / instruction count
      = Σ ((instruction count$_{\text{class i}}$ / instruction count) $\times$ CPI$_{\text{class i}}$)
      = 0.22 $\times$ 5 + 0.11 $\times$ 4 + 0.49 $\times$ 4 + 0.16 $\times$ 3 + 0.02 $\times$ 3
      = 4.04

- **Compared with CPI=1 in a single-cycle design**

Outline

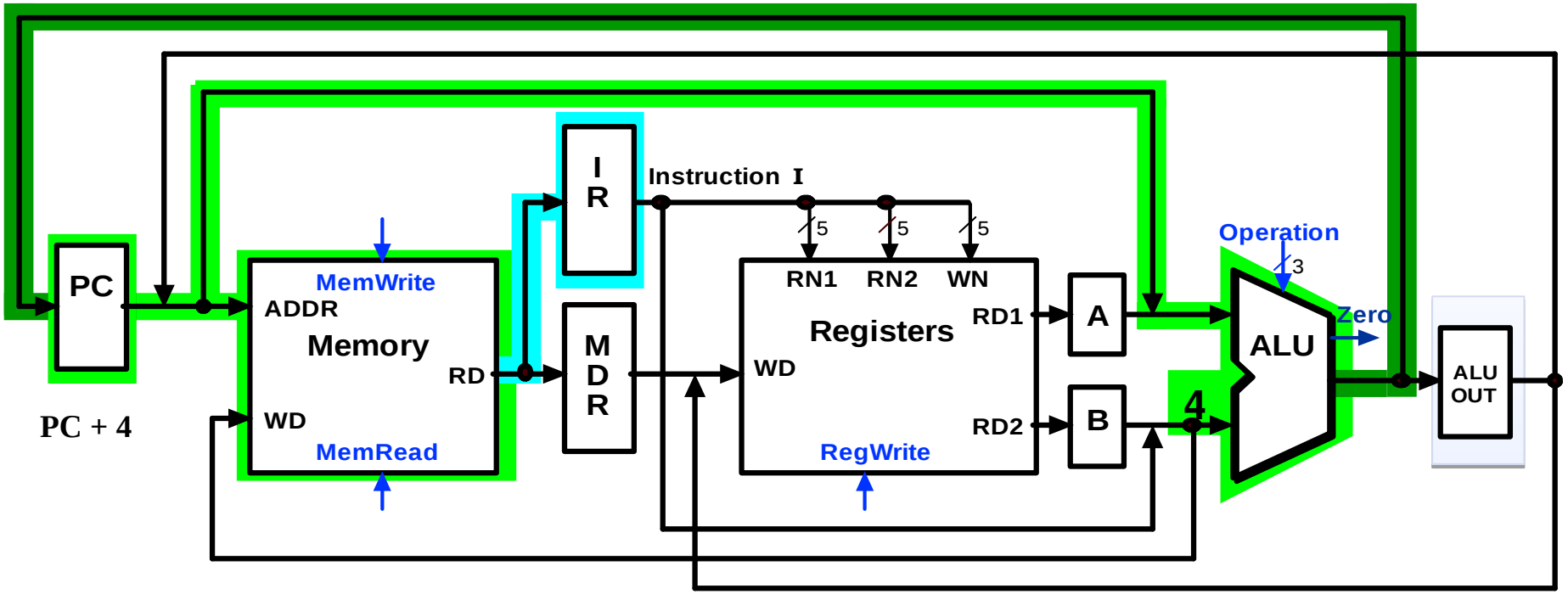
- A multicycle implementation
  - Multicycle datapath
  - **Multicycle execution steps**
  - **Multicycle control**

Instruction Execution in a Multicycle Processor

Step	Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
1: IF	Instruction fetch	IR = Memory[PC] PC = PC + 4			
2: ID	Instruction decode/register fetch	A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
3: EX	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)
4: MEM	Memory access or R-type completion	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
5: WB	Memory read completion		Load: Reg[IR[20-16]] = MDR		

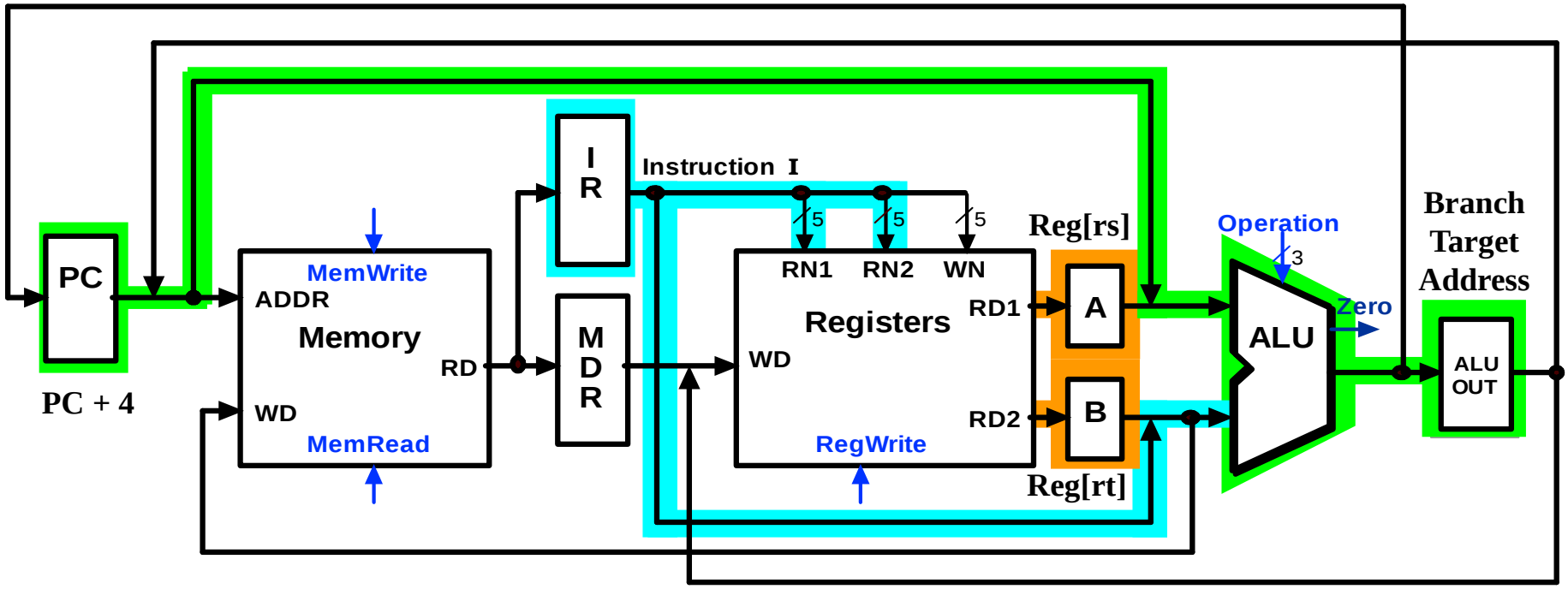
Multicycle Execution Step (1): Instruction Fetch

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



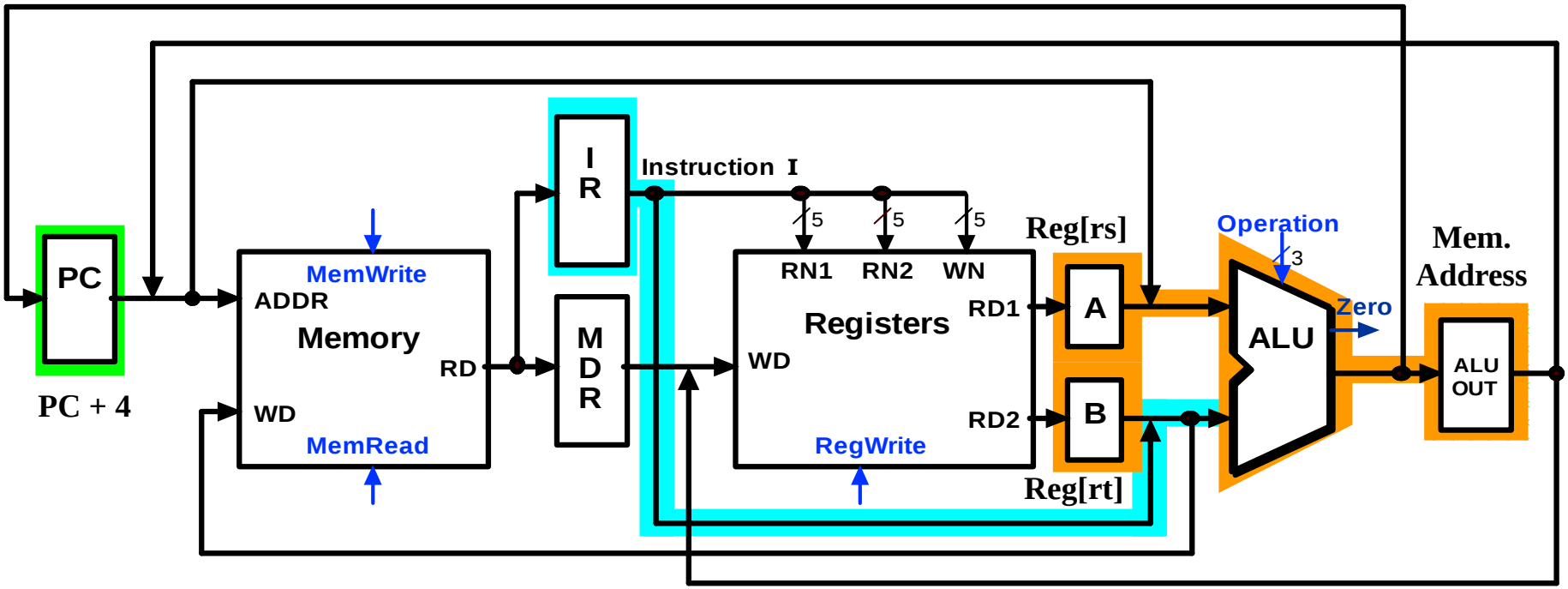
Multicycle Execution Step (2): Instruction Decode & Register Fetch

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



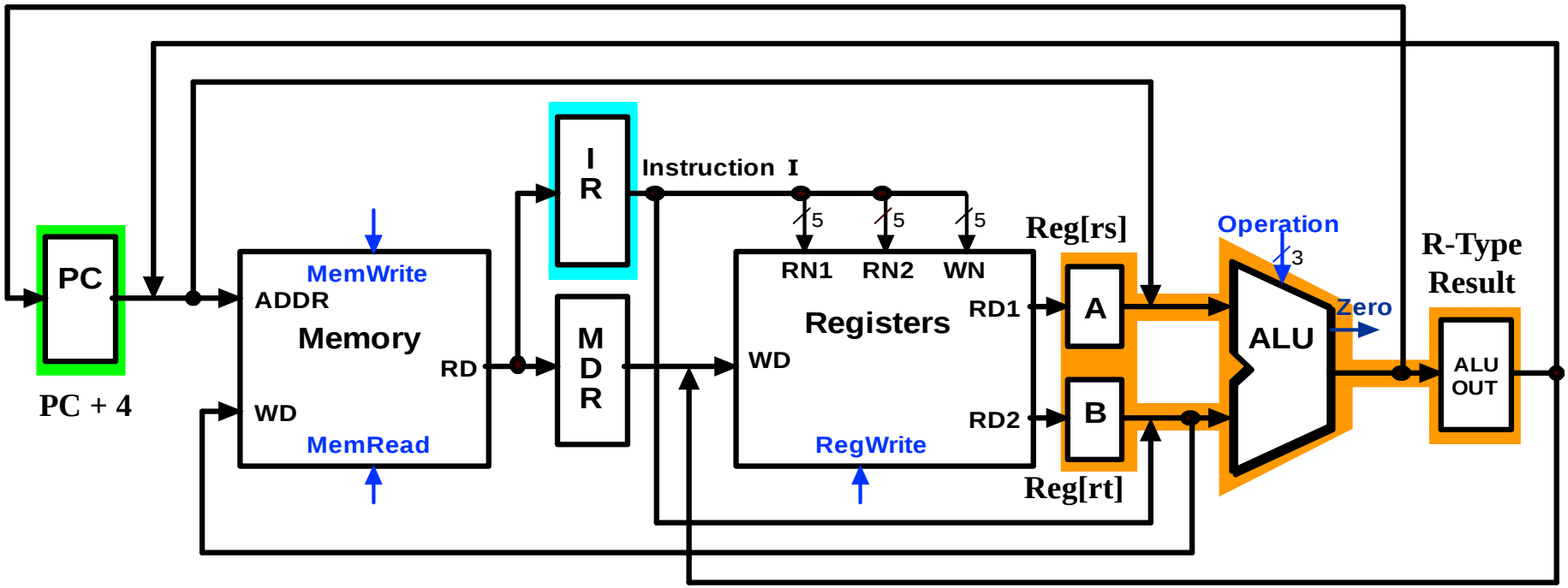
Multicycle Execution Step (3): Memory Reference Instructions

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



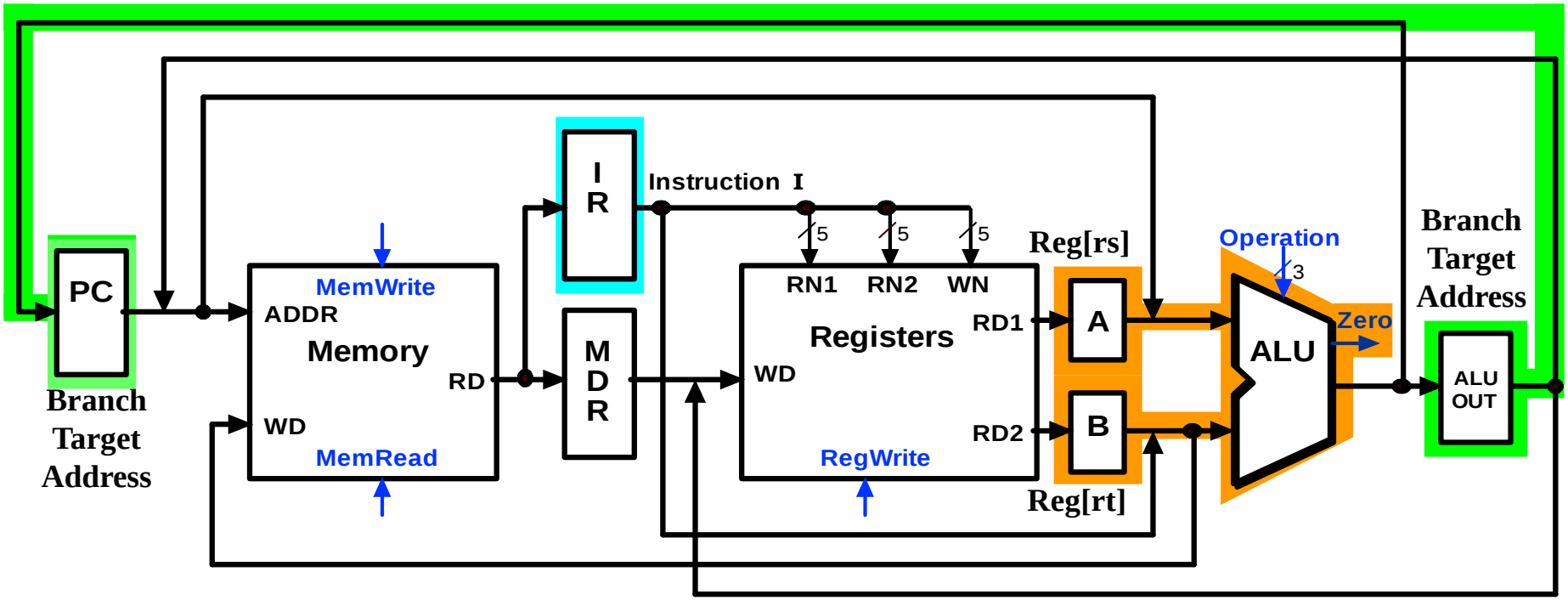
Multicycle Execution Step (3): ALU Instruction (R-Type)

`ALUOut = A op B`



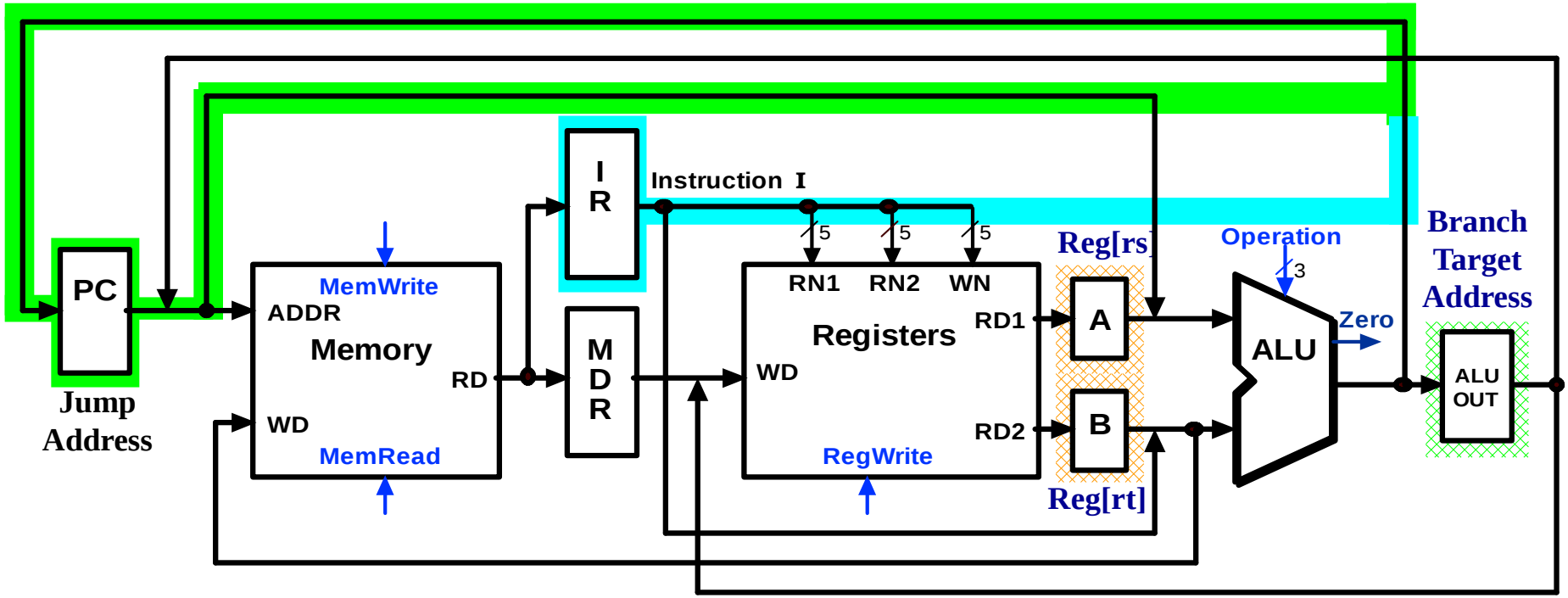
Multicycle Execution Step (3): Branch Instructions

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



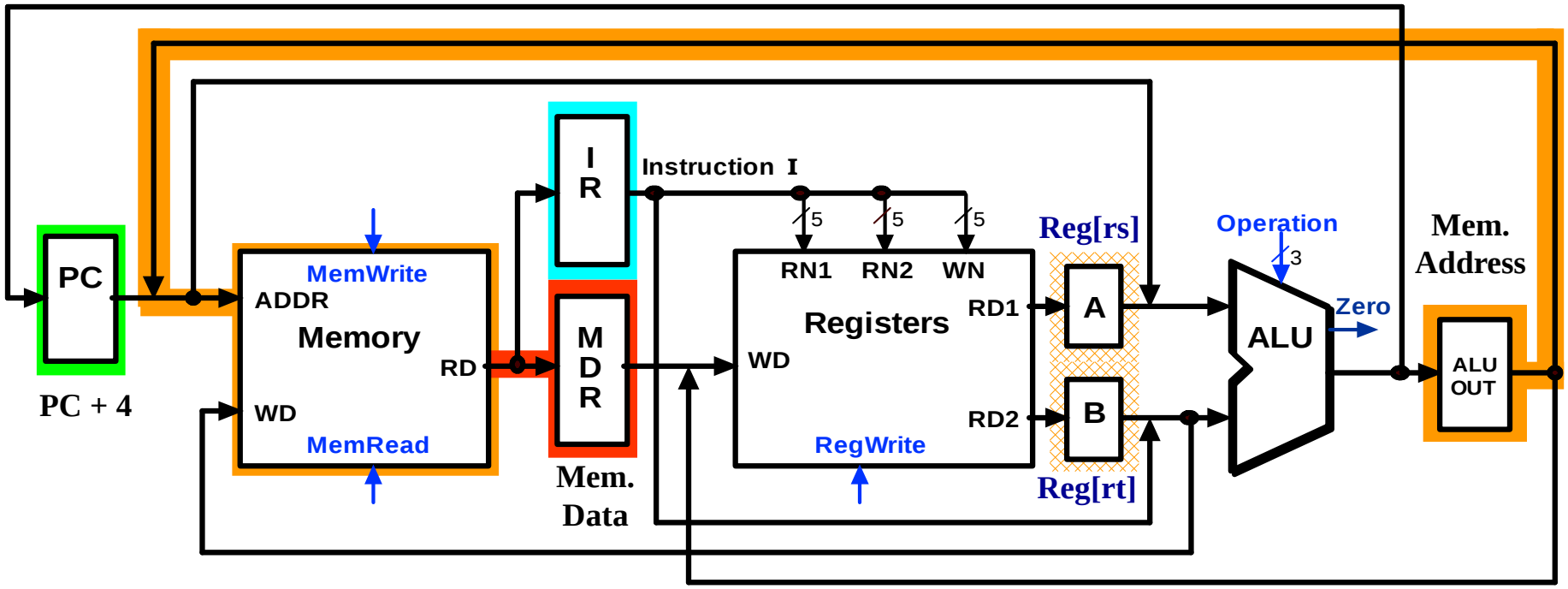
Multicycle Execution Step (3): Jump Instruction

```
PC = PC[31-28] concat (IR[25-0] << 2)
```



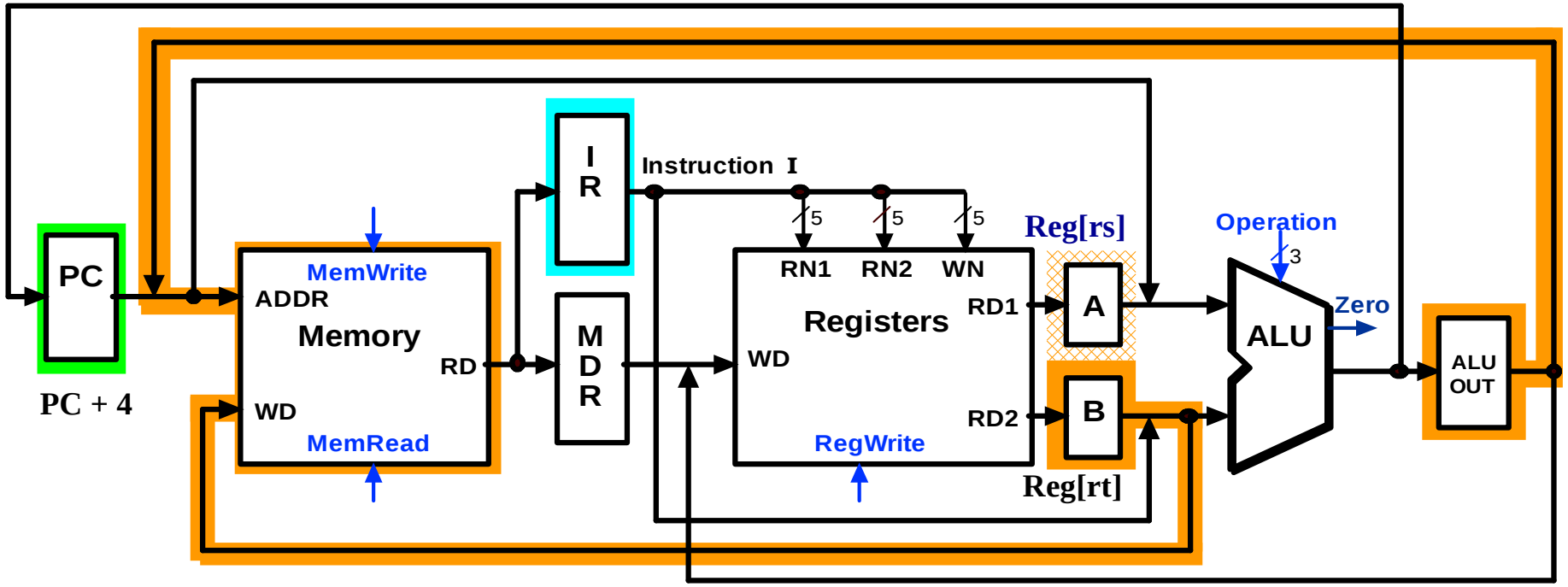
Multicycle Execution Step (4): Memory Access - Read (lw)

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



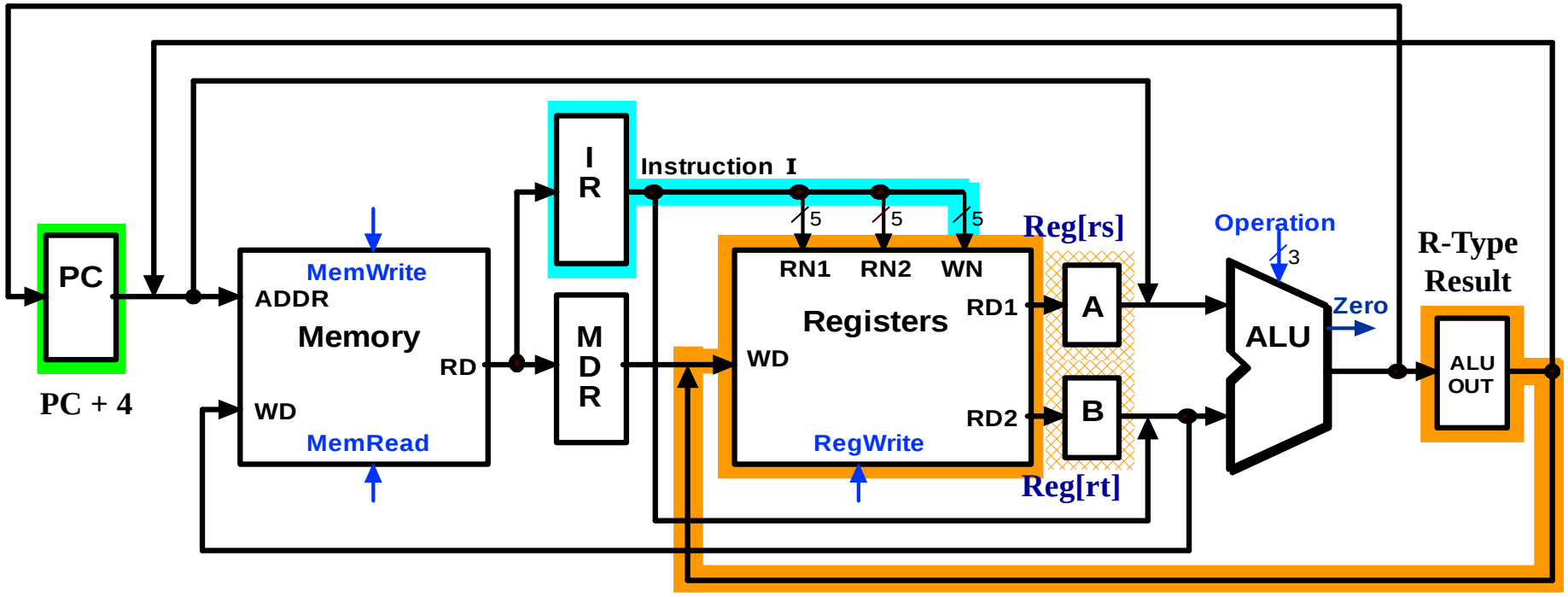
Multicycle Execution Step (4): Memory Access - Write (sw)

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



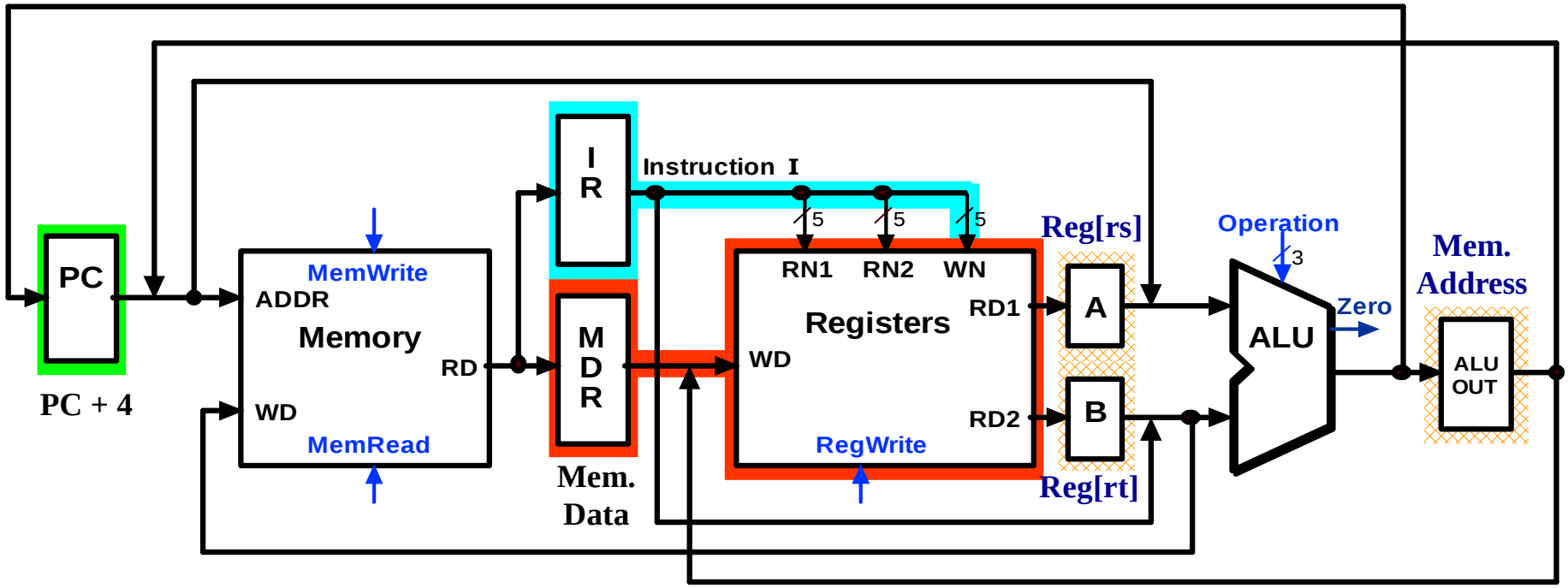
Multicycle Execution Step (4): ALU Instruction (R-Type)

`Reg[IR[15:11]] = ALUOUT`

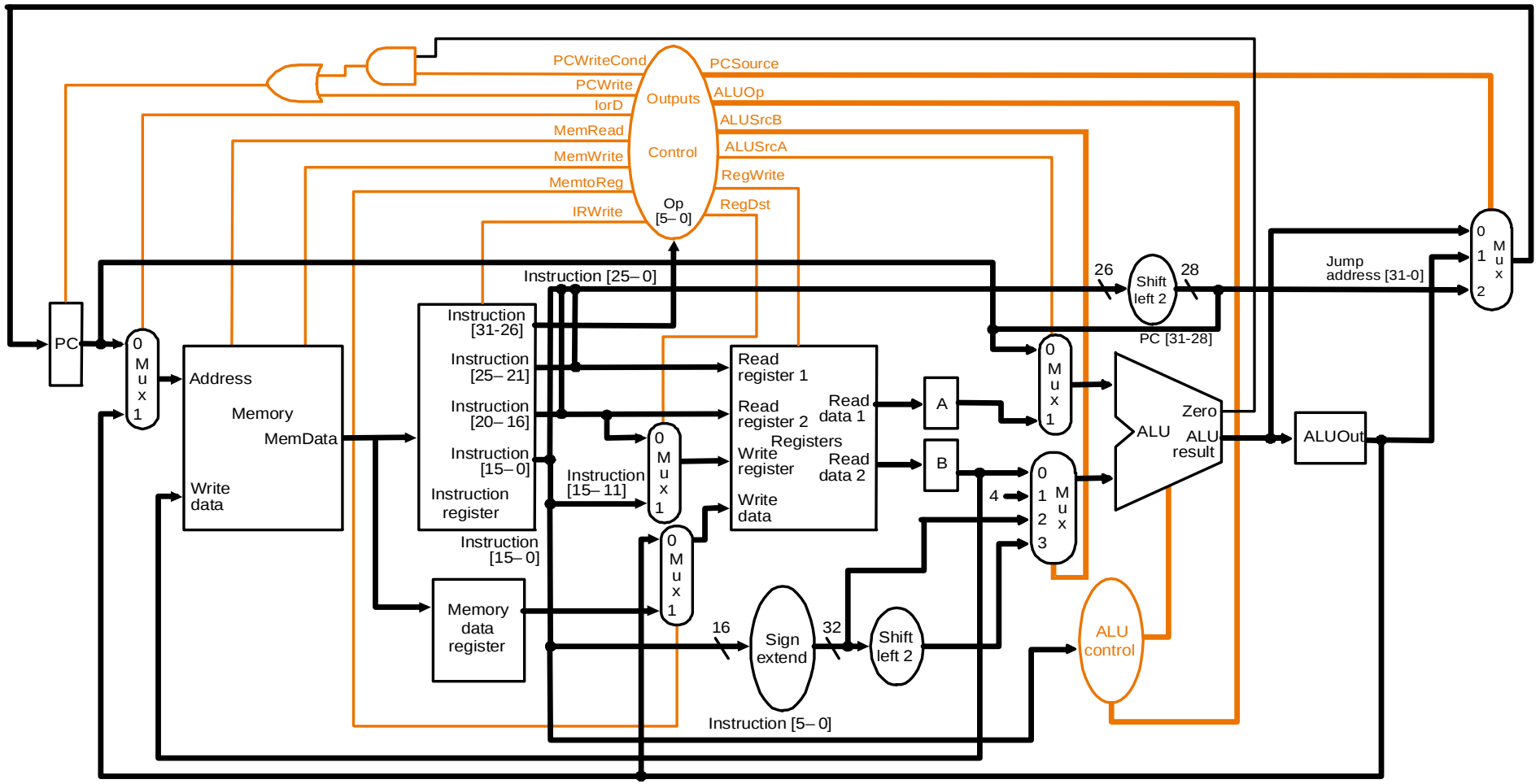


Multicycle Execution Step (5): Memory Read Completion (lw)

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



Multicycle Datapath with Control



Complete multicycle MIPS datapath (with branch and jump capability) and showing the main control block and all control lines

Control Signals

Single Bit Control

Signal name	*Effect when deasserted*	*Effect when asserted*
ALUSrcA	1st ALU operand = PC	1st ALU operand = Reg[rs]
RegWrite	None	Reg file is written
MemtoReg	Reg. data input = ALU	Reg. write data input = MDR RegDst
	Reg. write dest. no. = rt	Reg. write dest. no. = rd
MemRead	None	Memory at address is read
MemWrite	None	Memory at address is written
IorD	Memory address = PC	Memory address = ALUout
IRWrite	None	IR = Memory
PCWrite	None	PC = PCSource
PCWriteCond	None	If zero then PC = PCSource

Multiple Bit Control

Signal name	*Value*	*Effect*
ALUOp	00	ALU adds
	01	ALU subtracts
	10	ALU operates according to func code
ALUSrcB	00	2nd ALU input = B
	01	2nd ALU input = 4
	10	2nd ALU input = sign extended IR[15-0]
	11	2nd ALU input = sign ext., shift left 2 IR[15-0]
PCSource	00	PC = ALU (PC + 4)
	01	PC = ALUout (branch target address)
	10	PC = PC+4[31-28] : IR[25-0] << 2

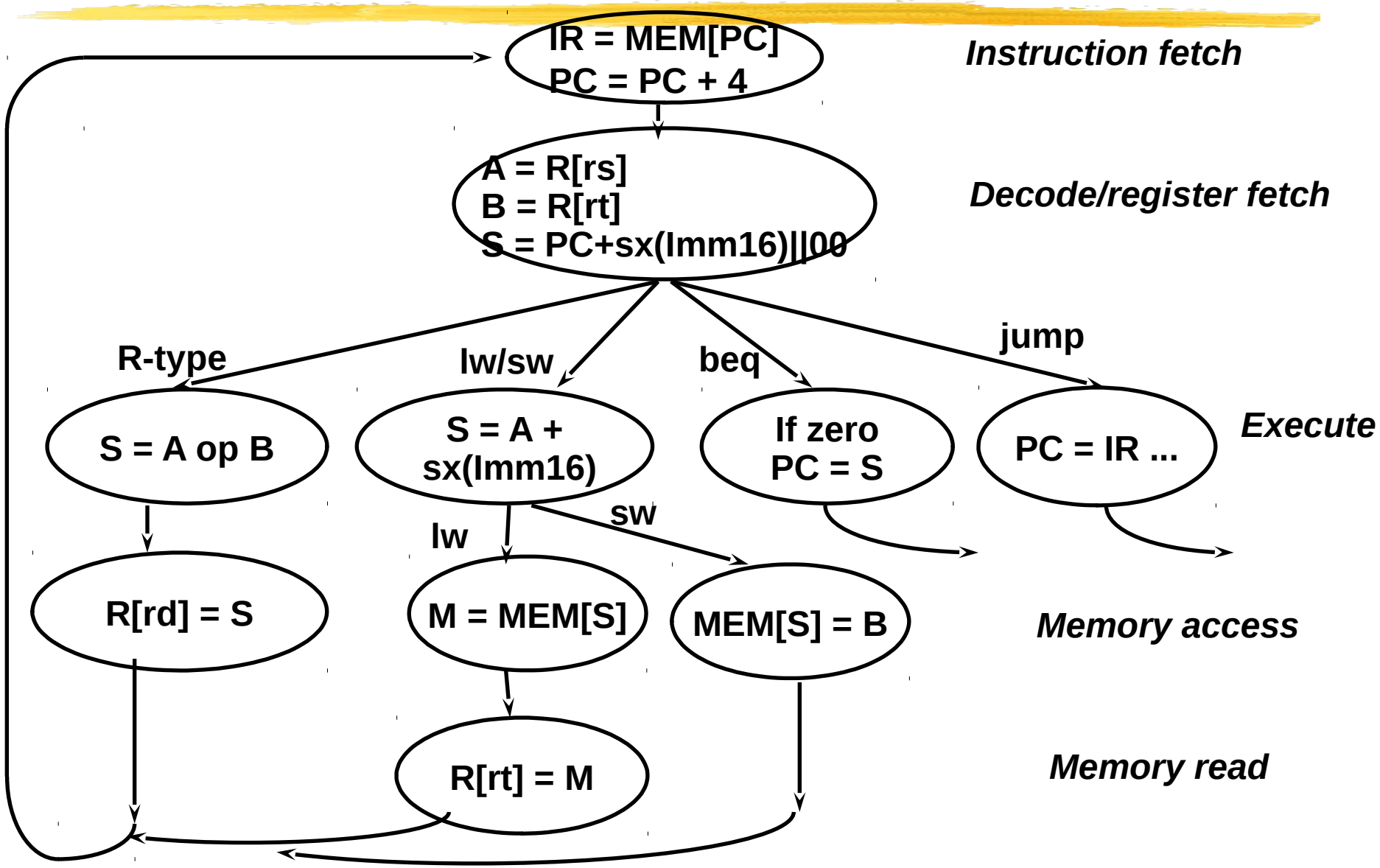
Outline

- A multicycle implementation
  - Multicycle datapath
  - Multicycle execution steps
  - **Multicycle control**

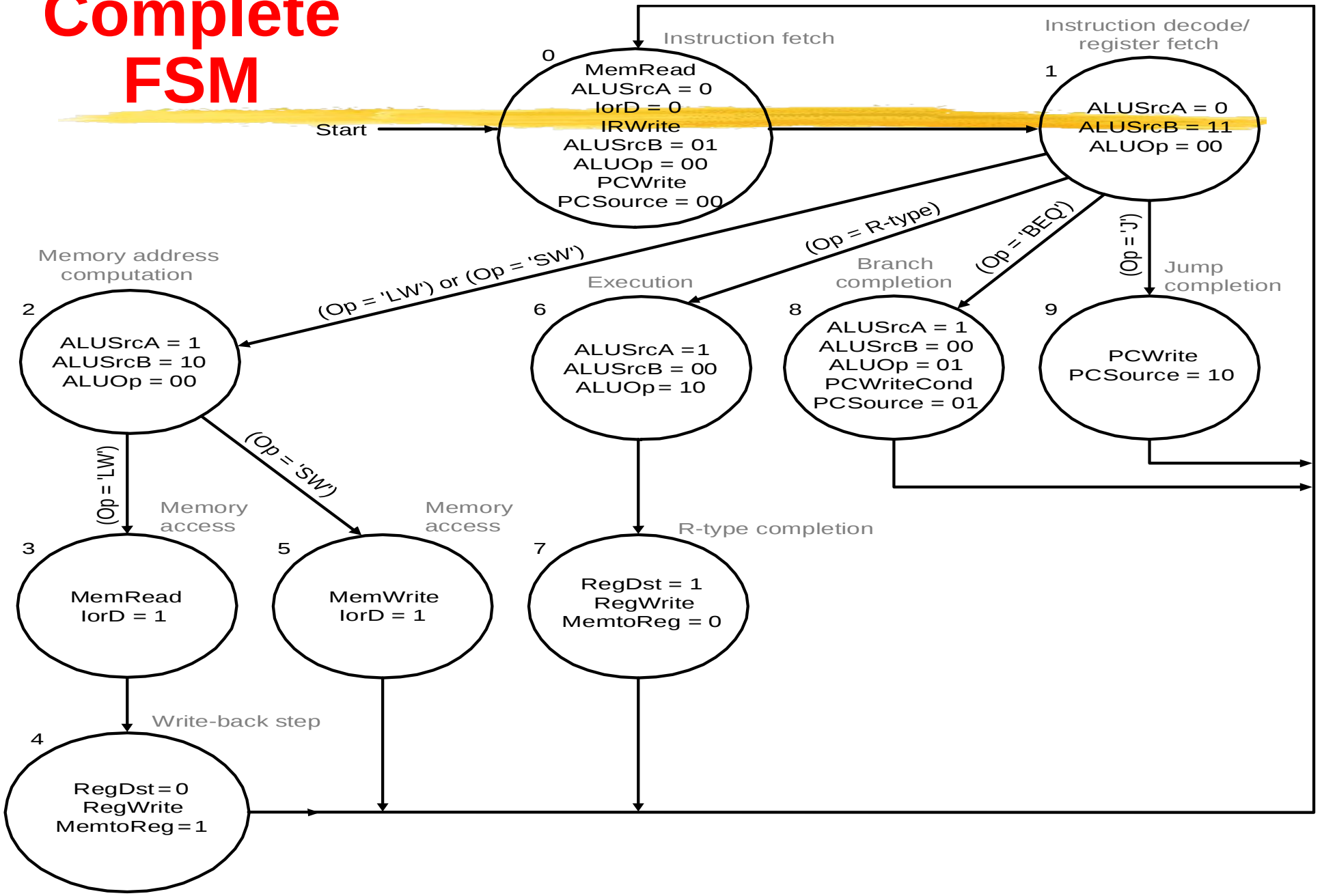
Implementing Control

- **Value of control signals is dependent upon:**
  - **what instruction is being executed**
  - **which step is being performed**
- **Based on the execution steps we can specify a finite state machine**
  - **specify the finite state machine graphically, or**
  - **use microprogramming**
- **Implementation is then derived from the specification**

RT Control Specification for Multicycle



Complete FSM

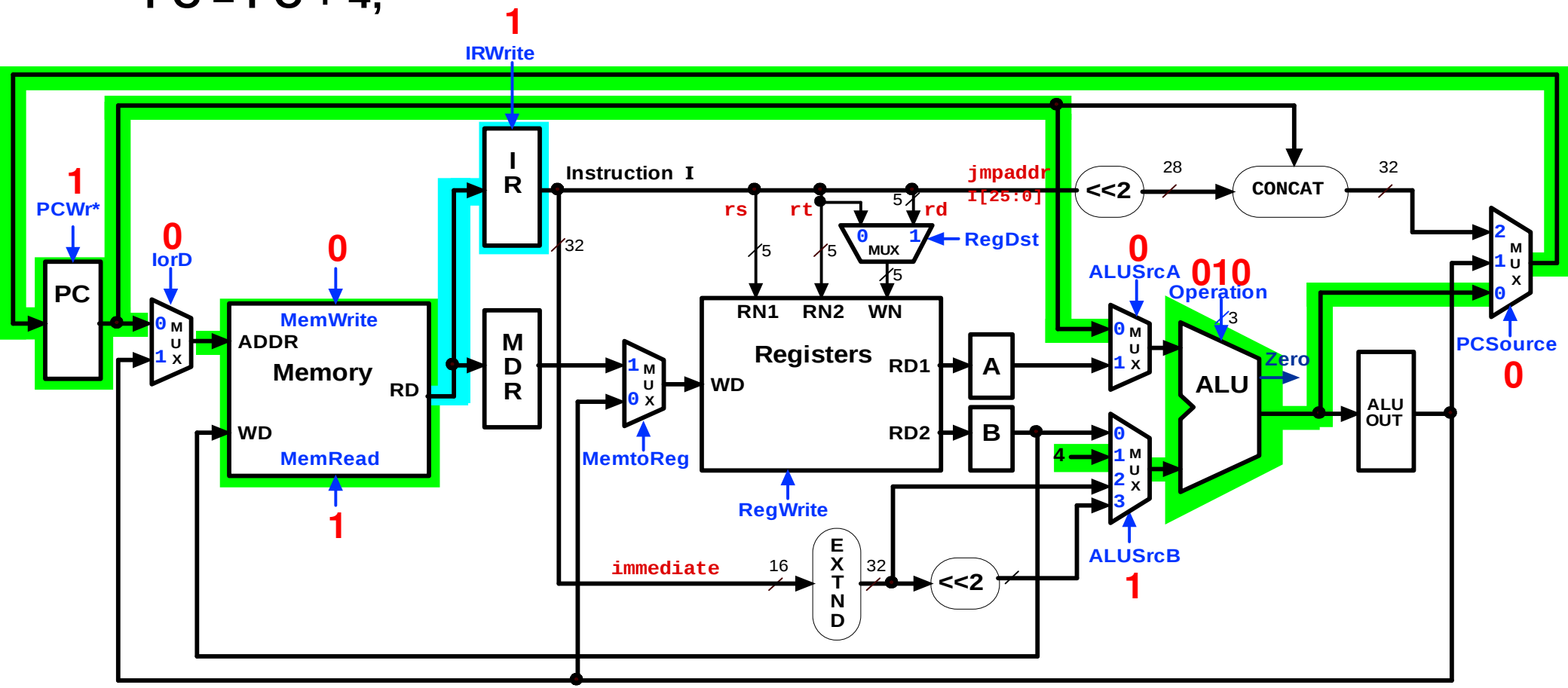


Assumptions

- **The implementation of a FSM usually assumes that**
  - **All outputs of control signals, except those controlling the MUXs, are deserted if they not explicitly specified**
  - **For those control signals attached to MUXs, their values are don't care if they are not explicitly specified**

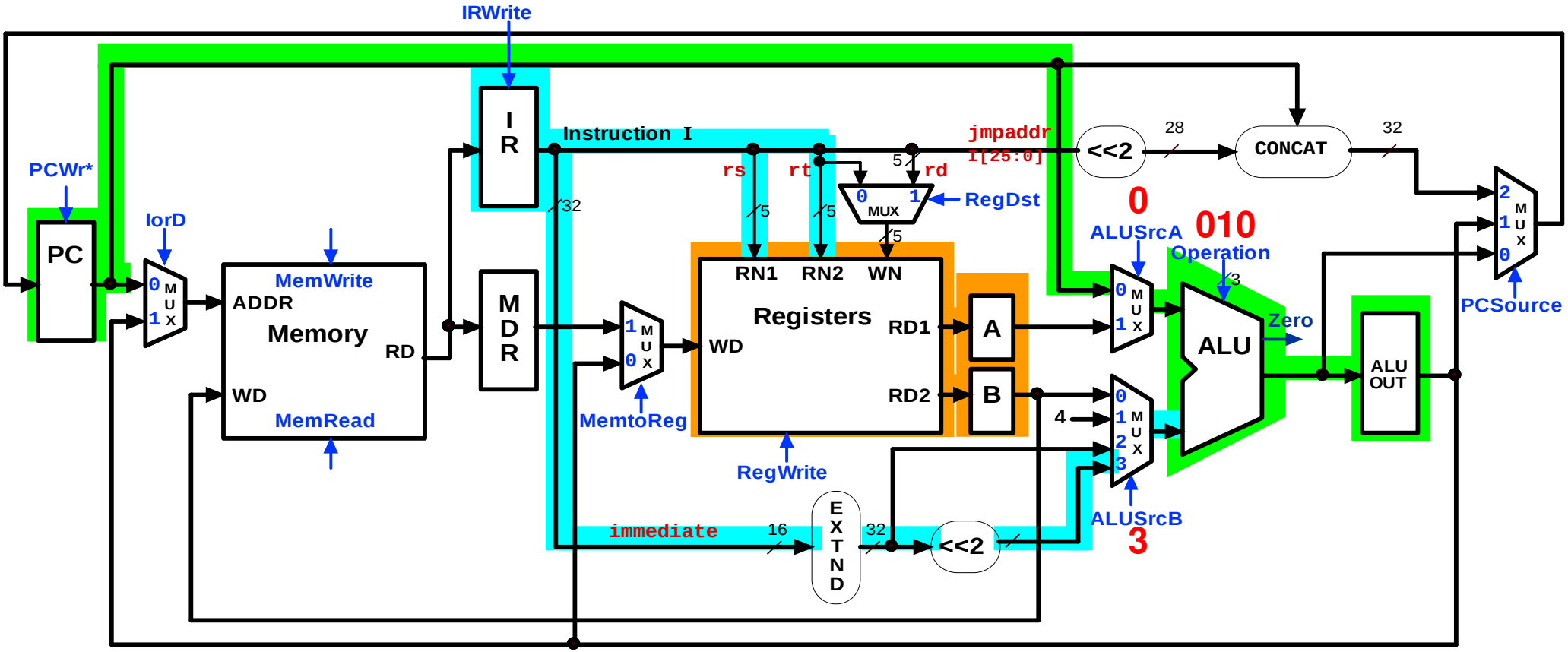
Multicycle Control Step (1): Fetch

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



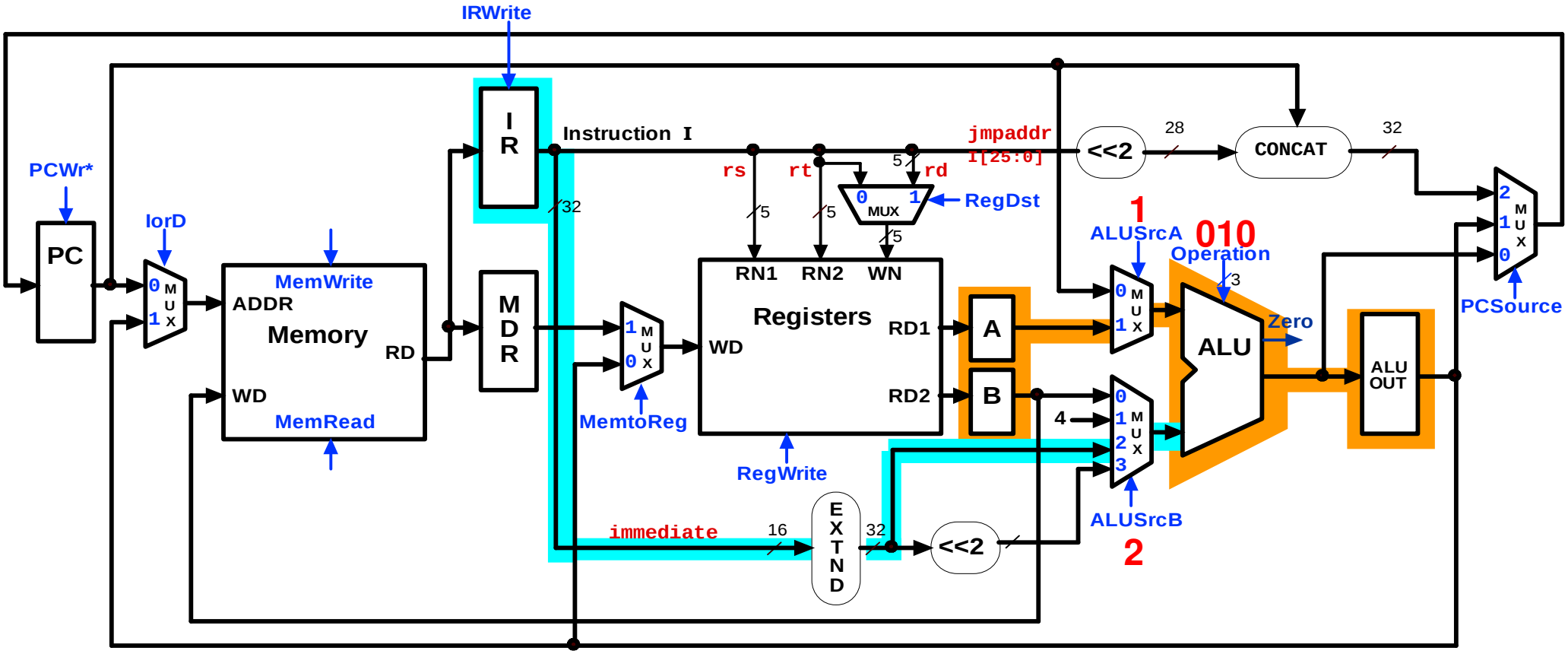
Multicycle Control Step (2): Instruction Decode & Register Fetch

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



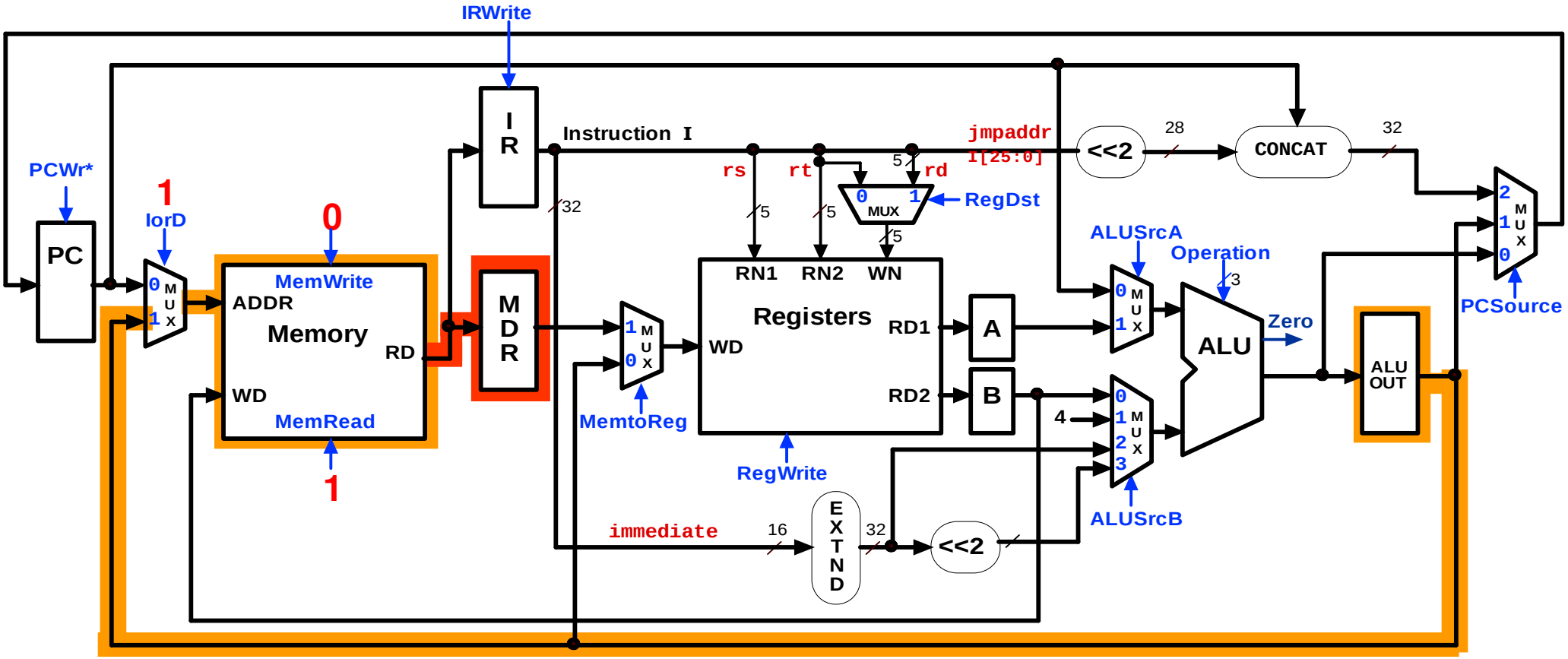
Multicycle Control Step (3): Memory Reference Instructions

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



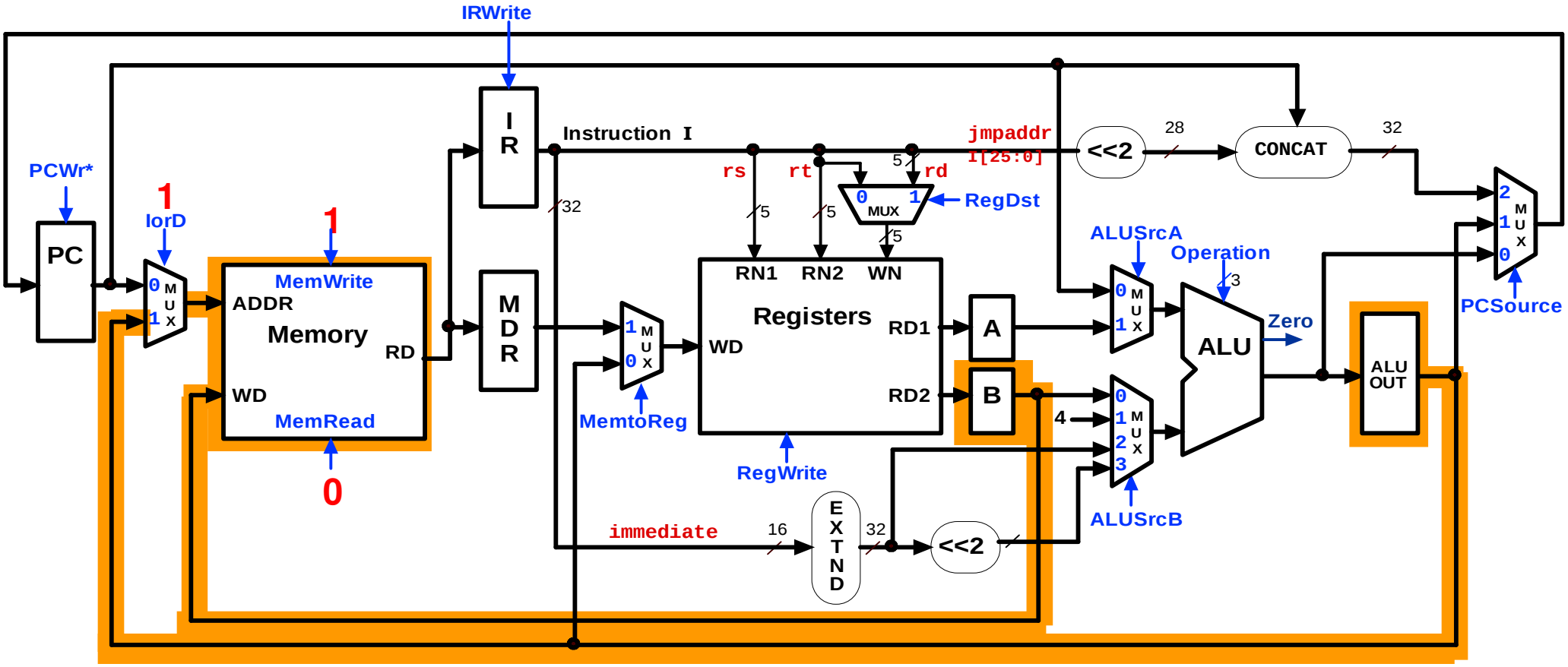
Multicycle Control Step (4): Memory Access - Read (lw)

MDR = Memory[ALUOut];



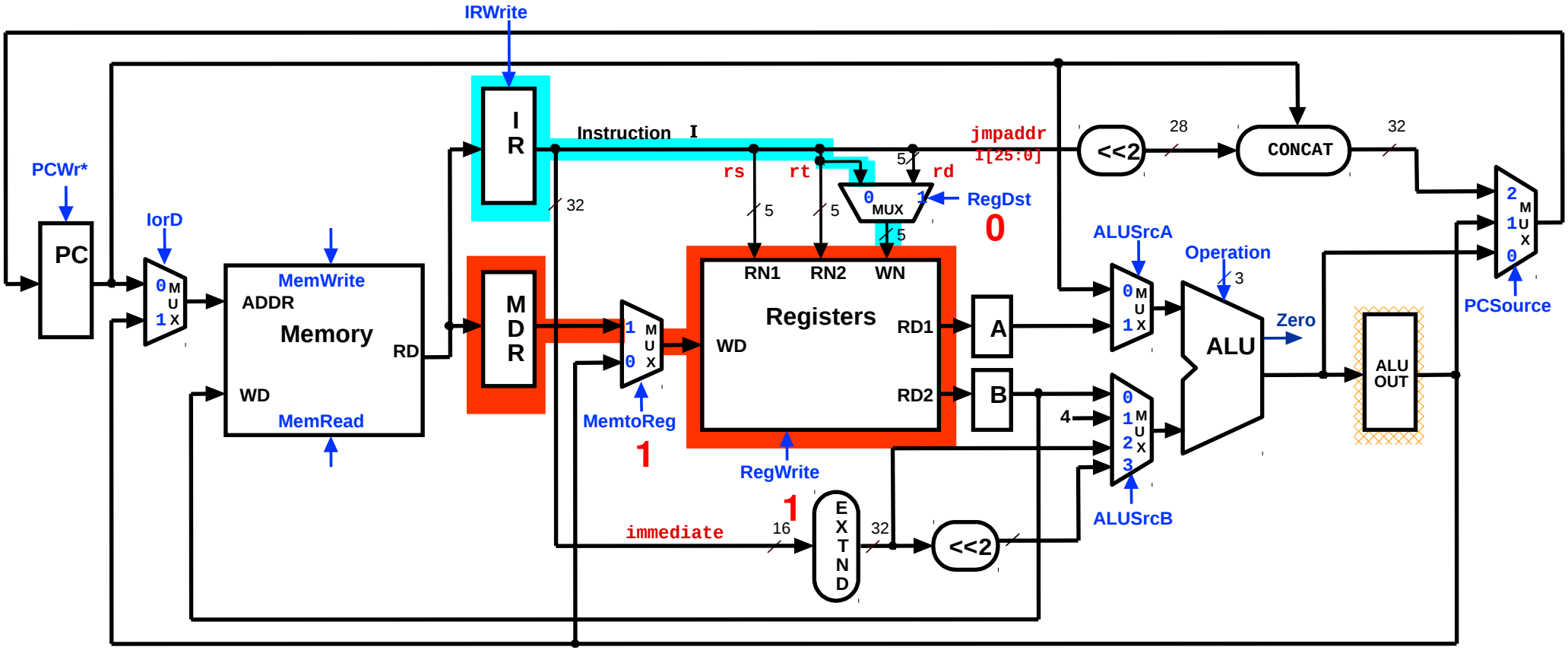
Multicycle Execution Steps (4)
Memory Access - Write (sw)

Memory[ALUOut] = B;



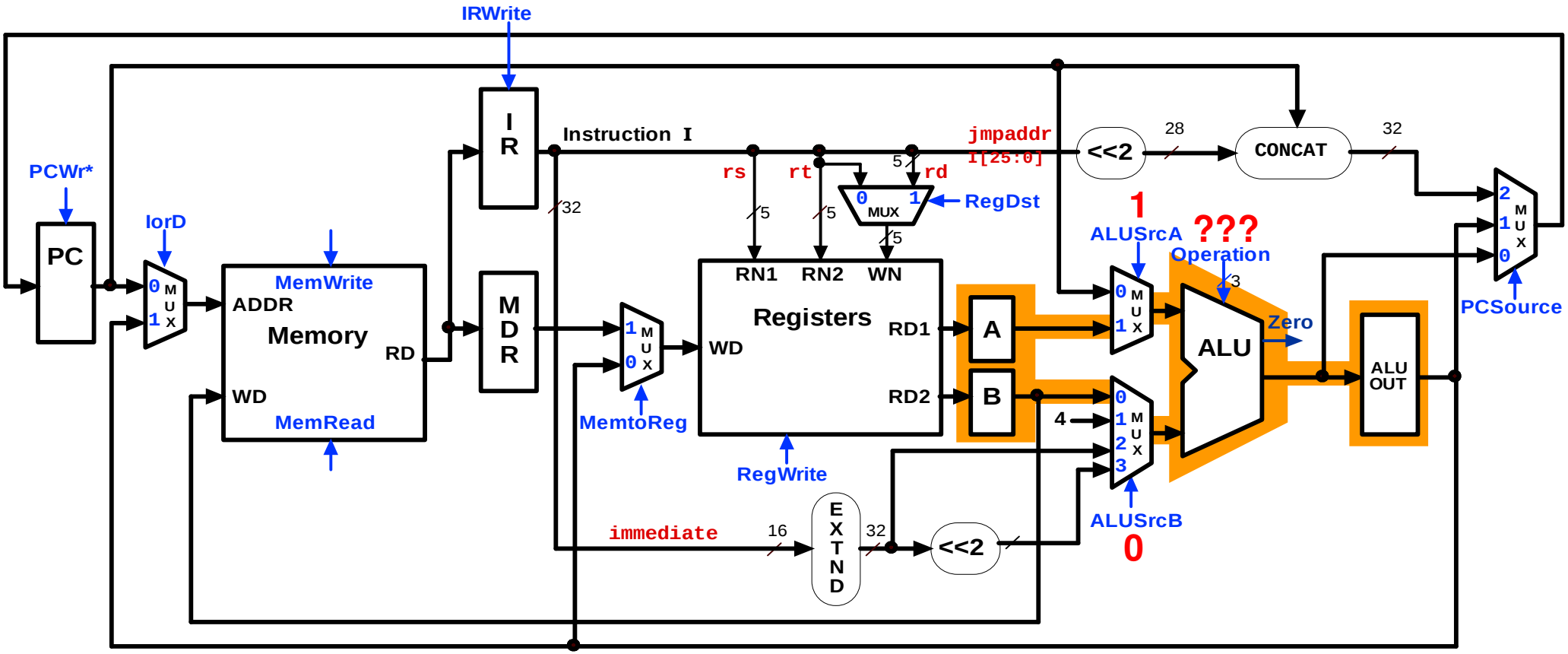
Multicycle Execution Steps (5)
Memory Read Completion (lw)

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



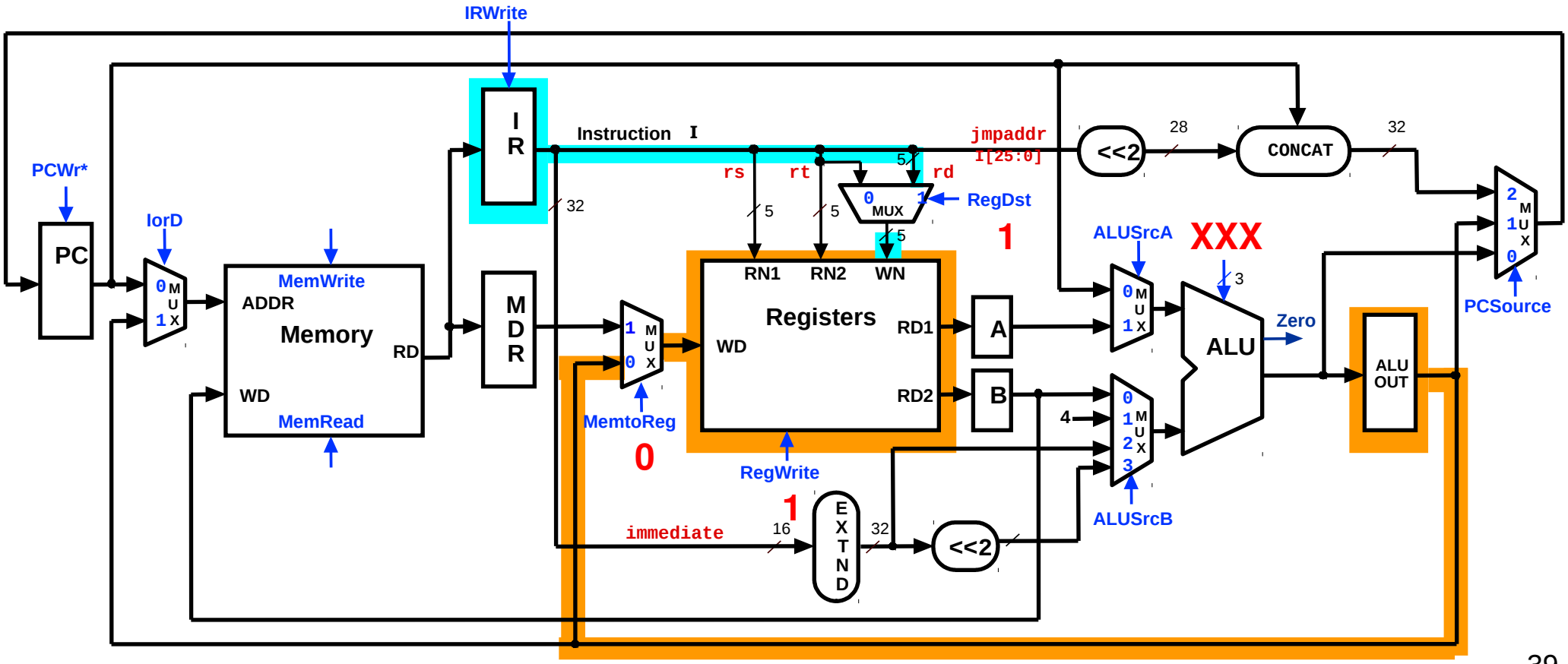
Multicycle Control Step (3): ALU Instruction (R-Type)

ALUOut = A op B;



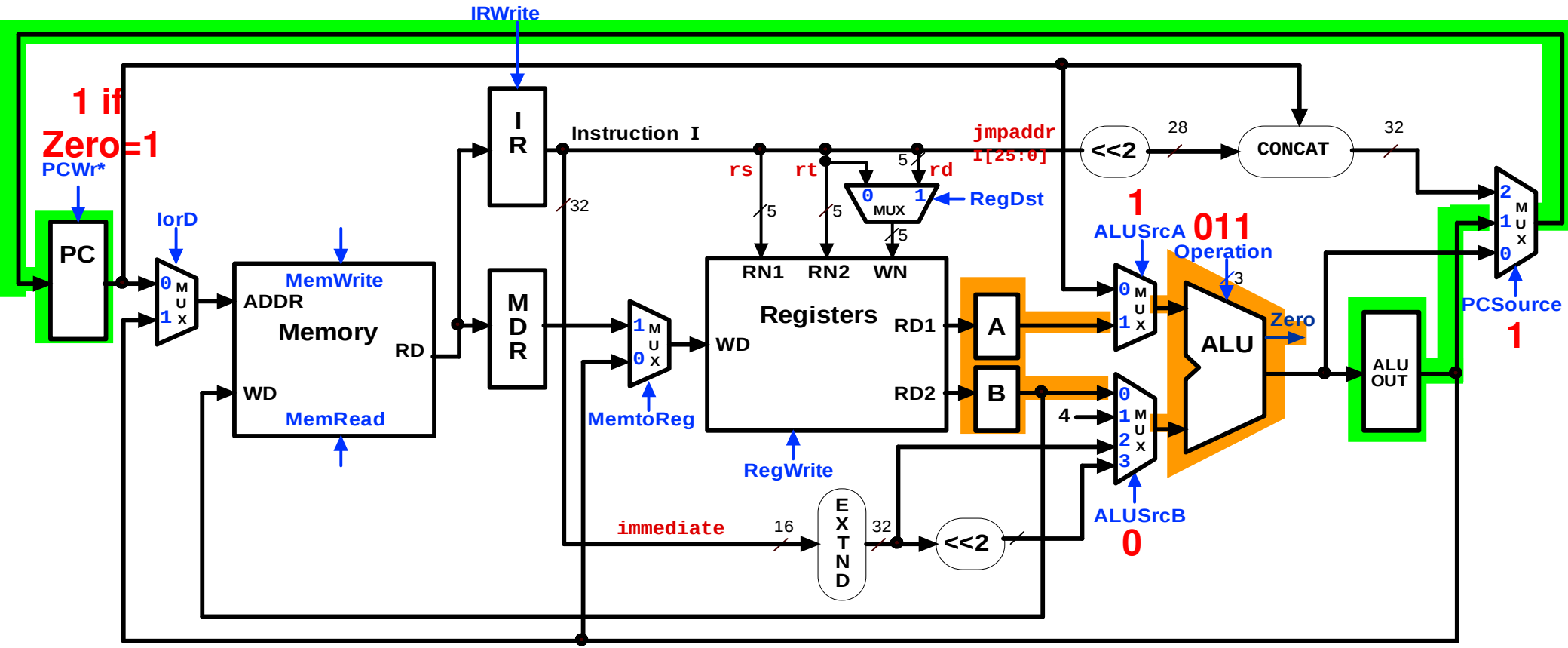
Multicycle Control Step (4): ALU Instruction (R-Type)

Reg[IR[15:11]] = ALUOut; (Reg[Rd] = ALUOut)



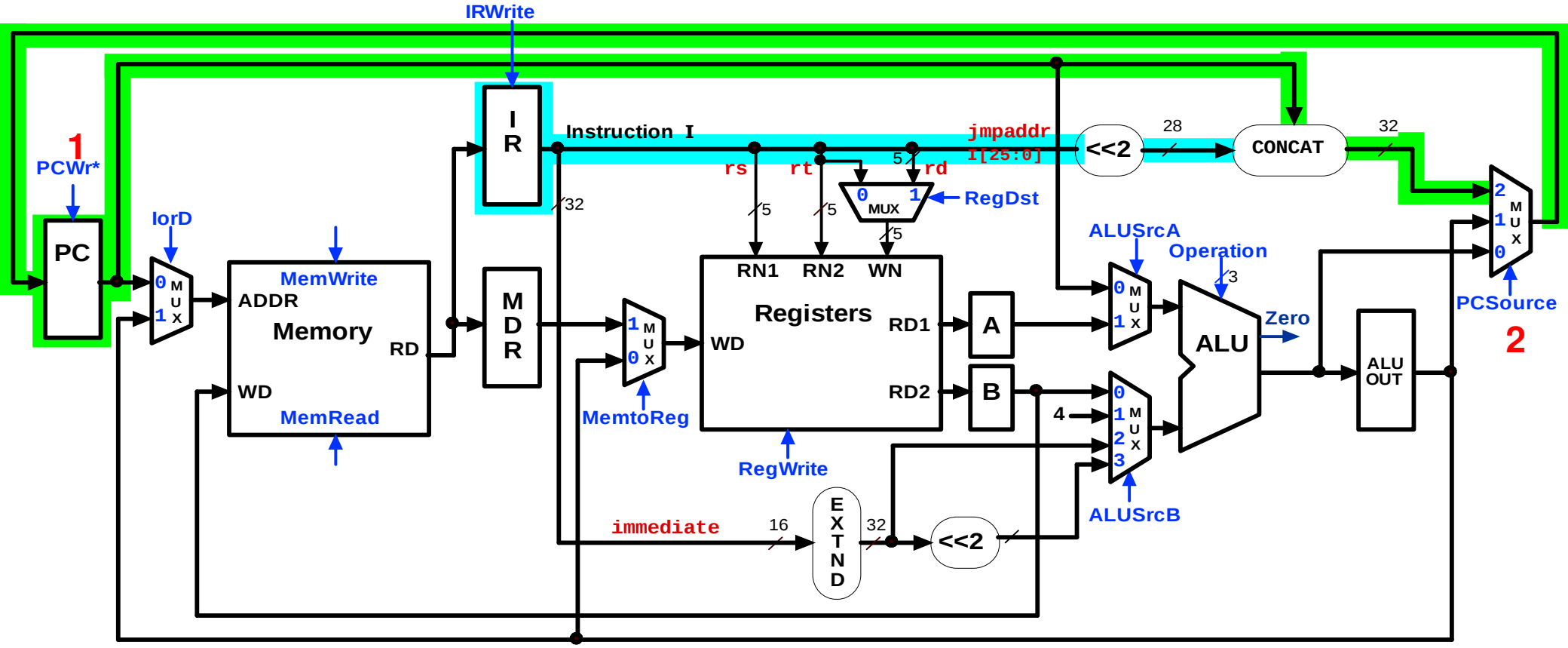
Multicycle Control Step (3): Branch Instructions

if (A == B) PC = ALUOut;



Multicycle Execution Step (3): Jump Instruction

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



Summary

- Single-cycle vs. multicycle
  - Motivation & comparison with an numerical example
- Understand the design of a multicycle processor
  - Datapath
  - Control (e.g., FSM)