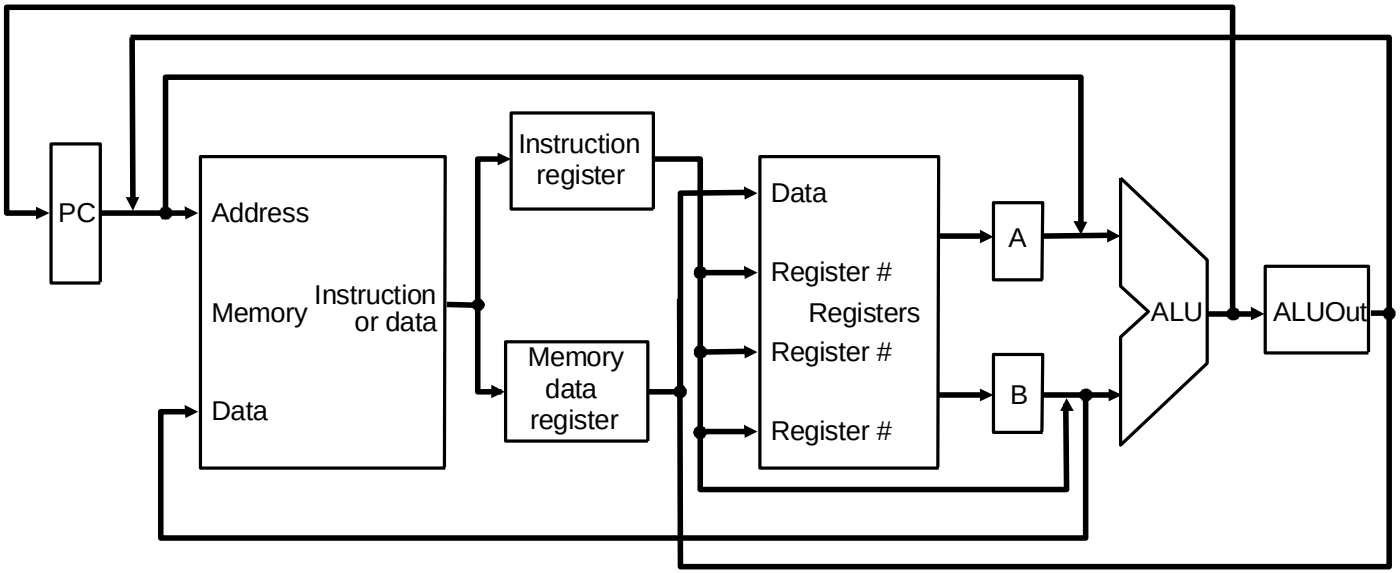
Lecture – 16

A Multicycle Implementation

- Each step in the execution will take one clock cycle.
- It allows a functional unit to be used more than once per instruction, as long as it is used on different clock cycles.
- Functional unit is shared between different clock cycles of an instruction.
- This sharing can help reduce the amount of hardware required.

The High-level view of the Multicycle Datapath

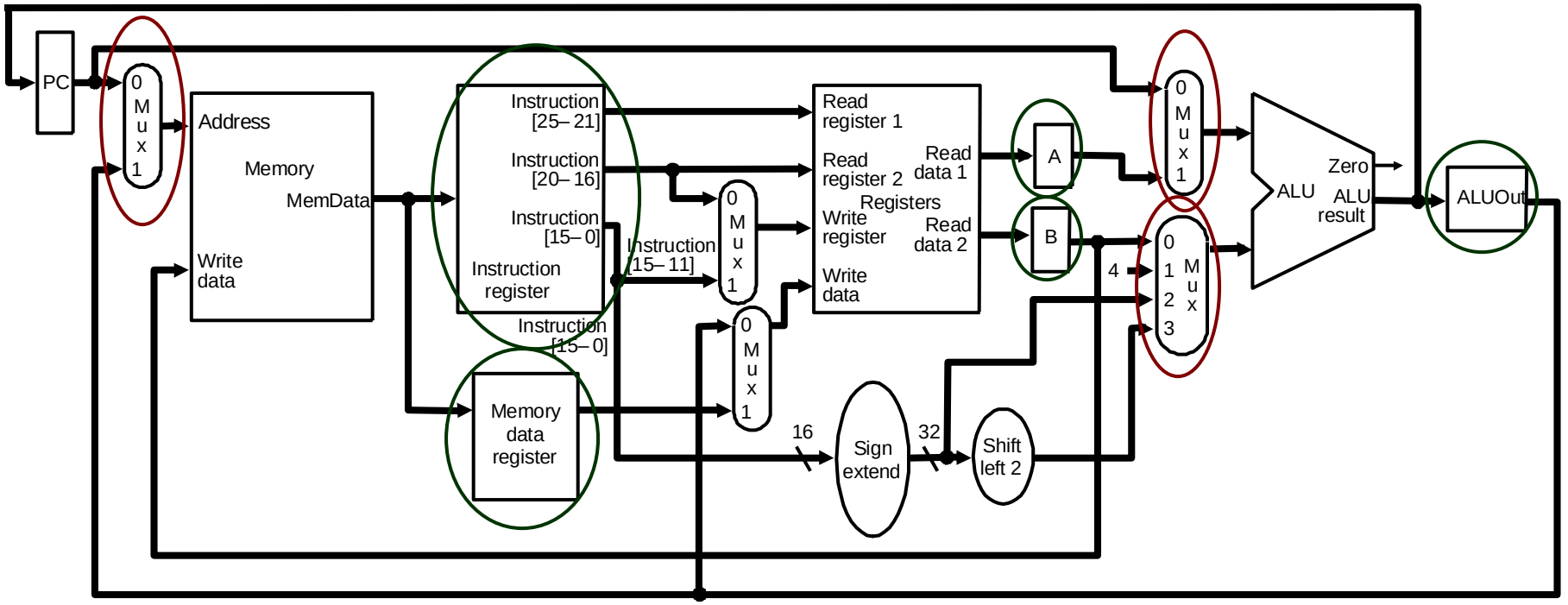


- A shared Memory Unit.
- A single ALU.
- Temporary registers to hold data between clock cycles of the same instruction. They are: Instruction register, Memory data register, A, B, and ALU Out.
- Data to be used in *later instructions* are stored in programmer-visible state elements: the register file, PC, memory.
- Data to be used in later clock cycles are stored in the additional registers.

Function of the Temporary Registers

- IR and MDR are added to save the output of the memory for an instruction read and a data read, respectively. Two separate registers are used as both values are needed during the same clock cycle.
- The A and B registers are used to hold the register operand values read from the register file.
- The ALUOut register holds the output of the ALU.
- All the registers except the IR hold data only between a pair of adjacent clock cycles.
- The IR needs to hold the instruction until the end of execution of that instruction.

Multicycle Datapath



- As several functional units are shared for different purposes, new multiplexors are added and existing multiplexors are expanded.

Additional Multiplexors

- Replacing the three ALUs of the single-cycle datapath by a single ALU means that the single ALU must accommodate all the inputs that used to go to the three different ALUs. Handling the additional inputs requires two changes to the datapath:

1. An additional multiplexor is added for the first ALU input. The multiplexor chooses between the A register and the PC.

2. The multiplexor on the second ALU input is changed from a 2-way to 4-way multiplexor. The two additional inputs to the multiplexor are the constant 4 and the sign-extended and shifted offset field.

Breaking Instruction into Steps

Instruction	Description	Inside processor's task
Add Sub	R[rd] <– R[rs] + R[rt]; PC <– PC + 4 CLK CYCLE :– 4	1. IR <– MEM[pc]; PC <– PC + 4 2. A<– R[rs]; B <– R[rt] 3. S <– A + B 4. R[rd] <– S;
Load	R[rt]<– MEM[R[rs]+SExt(Im16)]; PC <– PC + 4 CLK CYCLE :- 5	1. IR <– MEM[pc]; PC <– PC + 4 2. A<– R[rs]; 3. S <– A + SExt(Im16) 4. M <– MEM[S]; 5. R[rt] <– M;
Store	MEM[R[rs]+SExt(Im16)] <– R[rt]; PC <– PC + 4 CLK CYCLE :– 4	1. IR <– MEM[pc]; PC <– PC + 4 2. A<– R[rs]; B <– R[rt]; 3. S <– A + SExt(Im16); 4. MEM[S] <– B;
Branch	if R[rs] == R[rt] then PC <-PC + 4+SExt(Im16) || 00 else PC <= PC + 4 CLK CYCLE :- 3	1. IR <– MEM[pc]; PC <– PC + 4 3. E<– (R[rs] = R[rt]) if !E then *do nothing* 2. Else PC<-PC+SExt(Im16)||00

Step 1: Instruction Fetch & PC Increment (IF)

- Use PC to get instruction and put it in the instruction register.
- Increment the PC by 4 and put the result back in the PC.
- IR = Memory[PC];
  PC = PC + 4;

Step 2: Instruction Decode and Register Fetch (ID)

- Read registers rs and rt in case we need them.

- Compute the branch address.

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

Step 3: Execution, Address Computation or Branch Completion (EX)

- ALU performs one of four functions *depending* on instruction type
  - memory reference:
    ALUOut = A + sign-extend(IR[15-0]);
  - R-type:
    ALUOut = A op B;
  - branch (instruction *completes*):
    if (A==B) PC = ALUOut;
  - jump (instruction *completes*):
    PC = PC[31-28] || (IR(25-0) << 2)

Step 4: Memory access or R-type Instruction Completion (MEM)

- Again *depending* on instruction type:
- Loads and stores access memory
  - load

    MDR = Memory[ALUOut];
  - store (instruction *completes*)

    Memory[ALUOut] = B;

- R-type (instructions *completes*)
  Reg[IR[15-11]] = ALUOut;

Step 5: Memory Read Completion (WB)

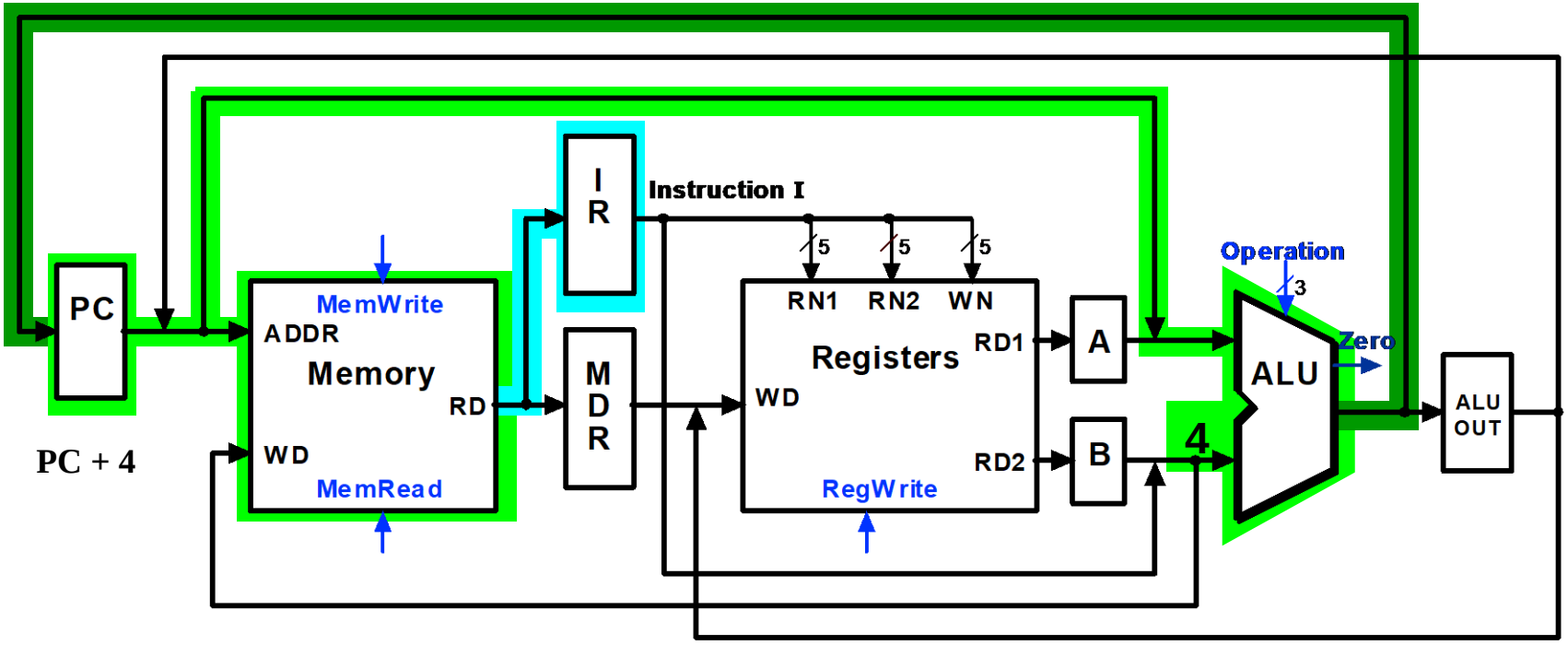
- Again *depending* on instruction type:
- Load writes back (instruction *completes*)
  Reg[IR[20-16]]= MDR;

Summary of Instruction Execution

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	A = Reg [IR[25-21]]			
	decode/register fetch	B = Reg [IR[20-16]]			
		ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
3: EX	Execution, address	ALUOut = A op B	ALUOut = A + sign-extend	if (A ==B) then	PC = PC [31-28] II
	computation, branch/		(IR[15-0])	PC = ALUOut	(IR[25-0]<<2)
	jump completion				
4: MEM	Memory access or R-type	Reg [IR[15-11]] =	Load: MDR = Memory[ALUOut]		
	completion	ALUOut	or		
			Store: Memory [ALUOut] = B		
5: WB	Memory read completion		Load: Reg[IR[20-16]] = MDR		

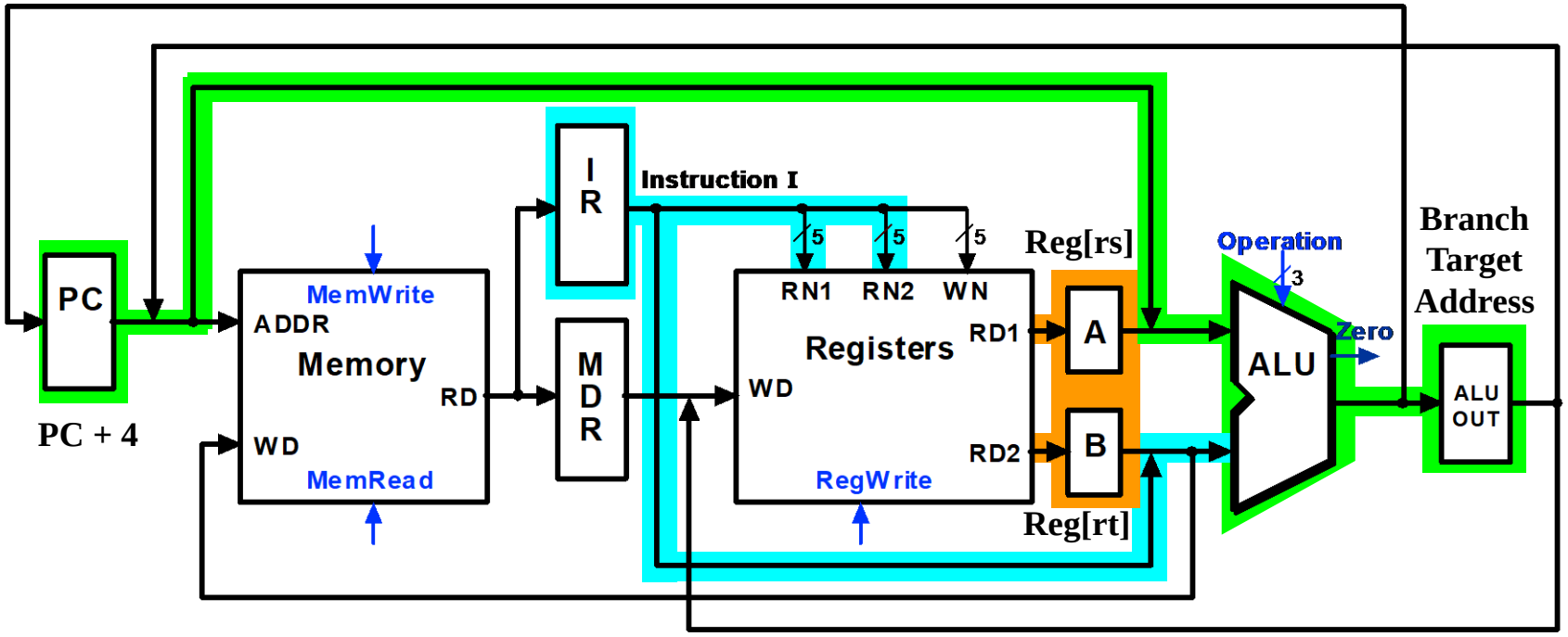
Multicycle Execution Step (1): Instruction Fetch

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



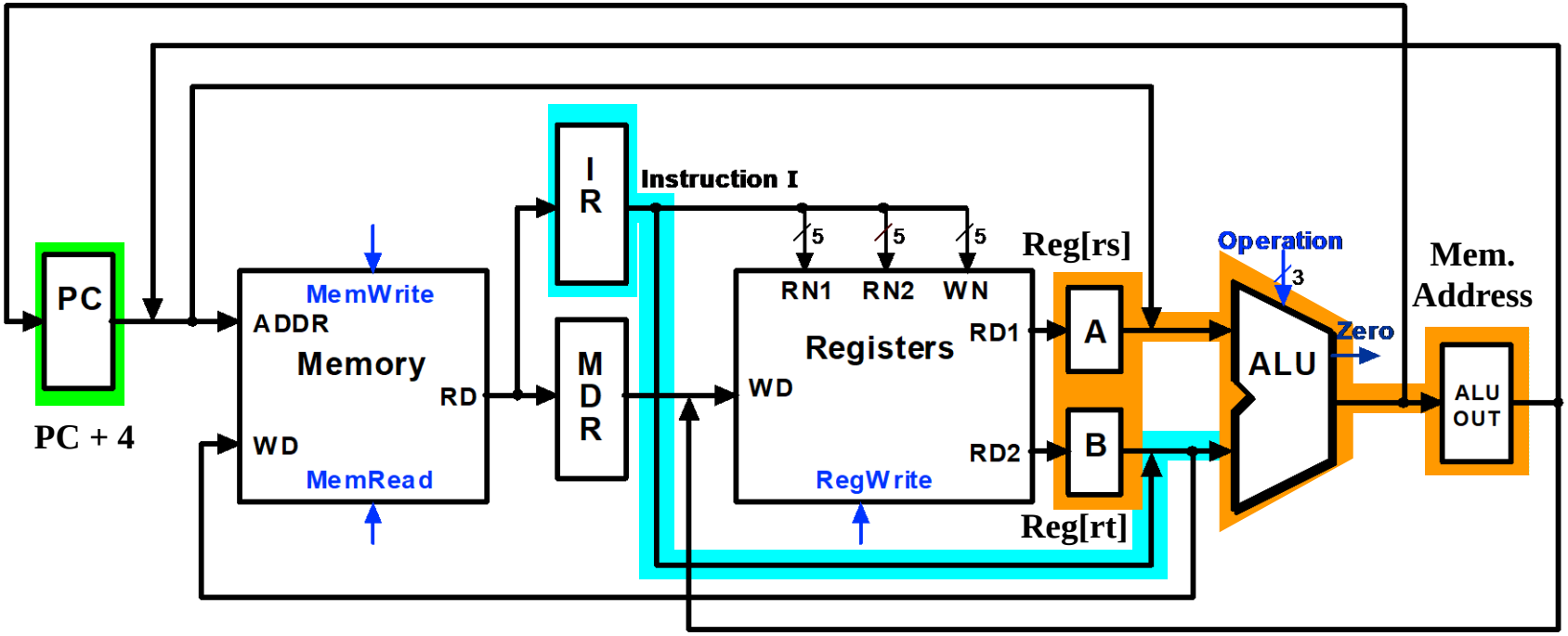
Multicycle Execution Step (2): Instruction Decode & Register Fetch

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



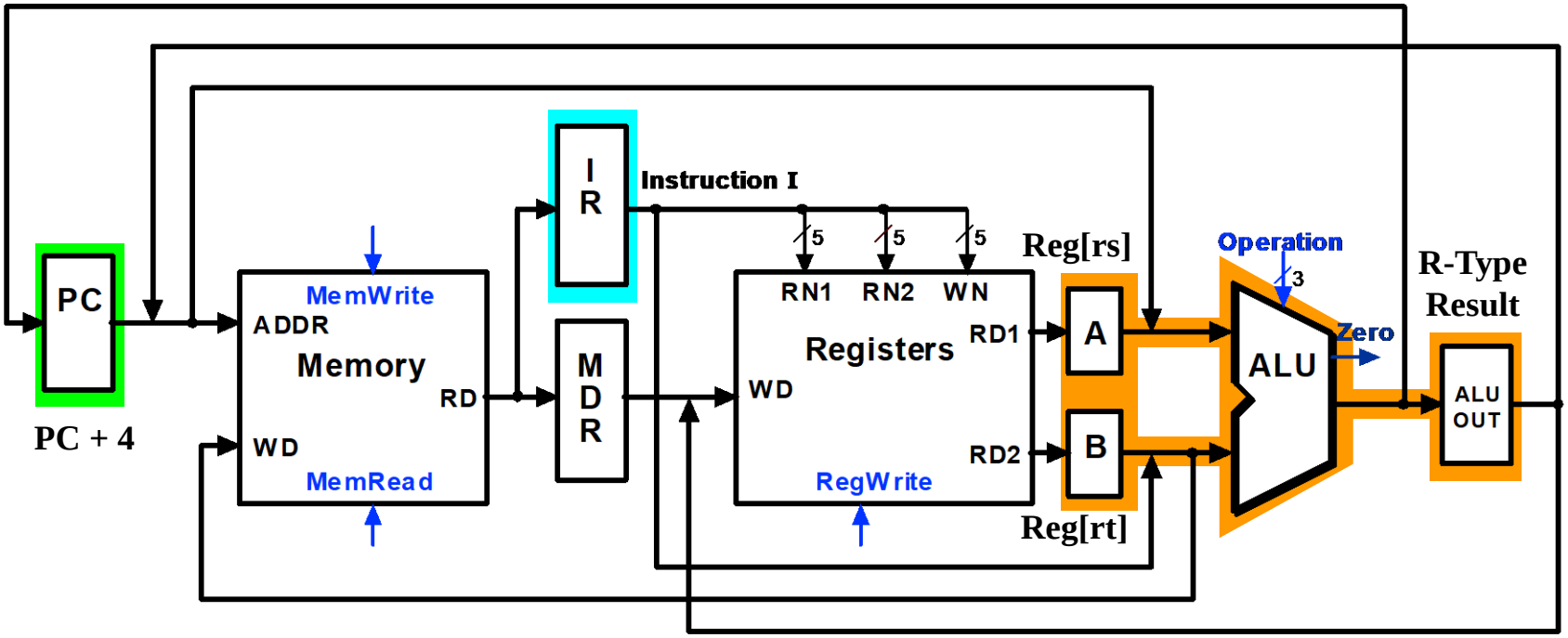
Multicycle Execution Step (3): Memory Reference Instructions

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



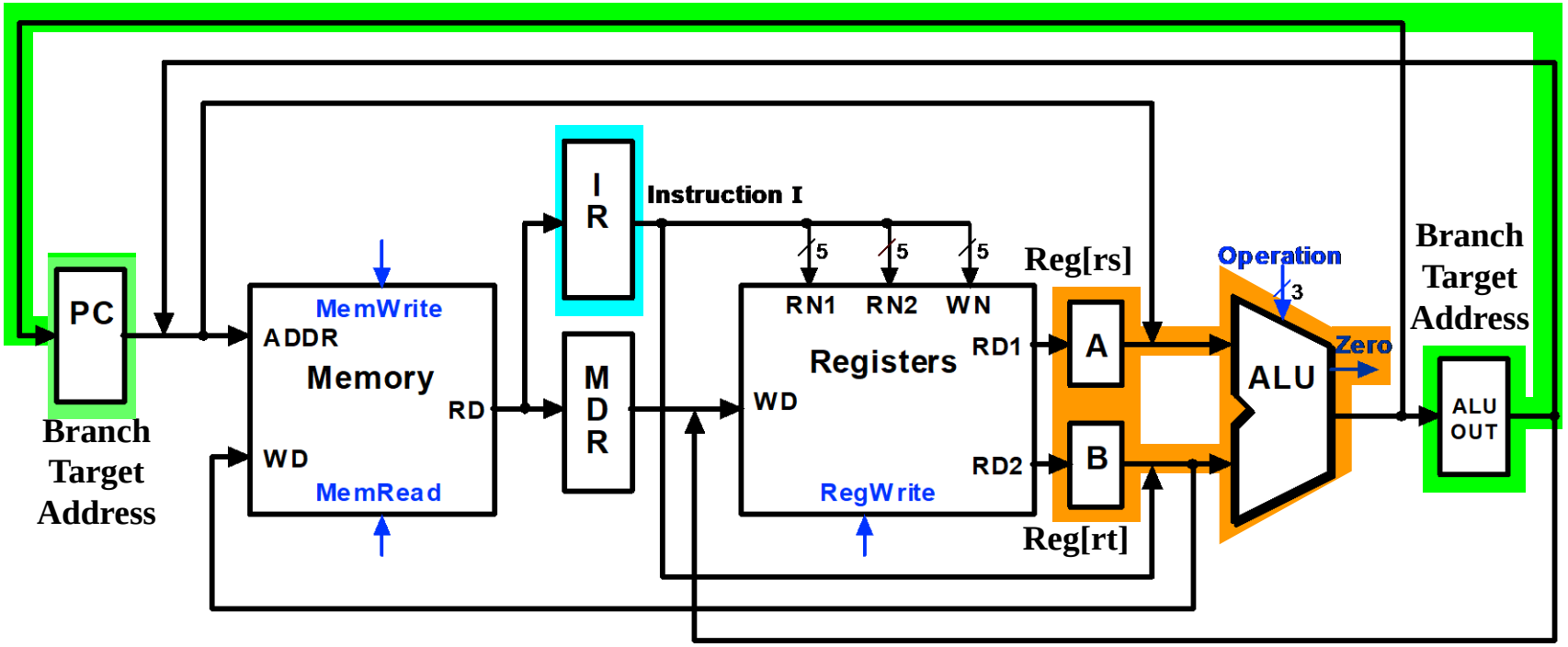
Multicycle Execution Step (3): ALU Instruction (R-Type)

ALUOut = A op B



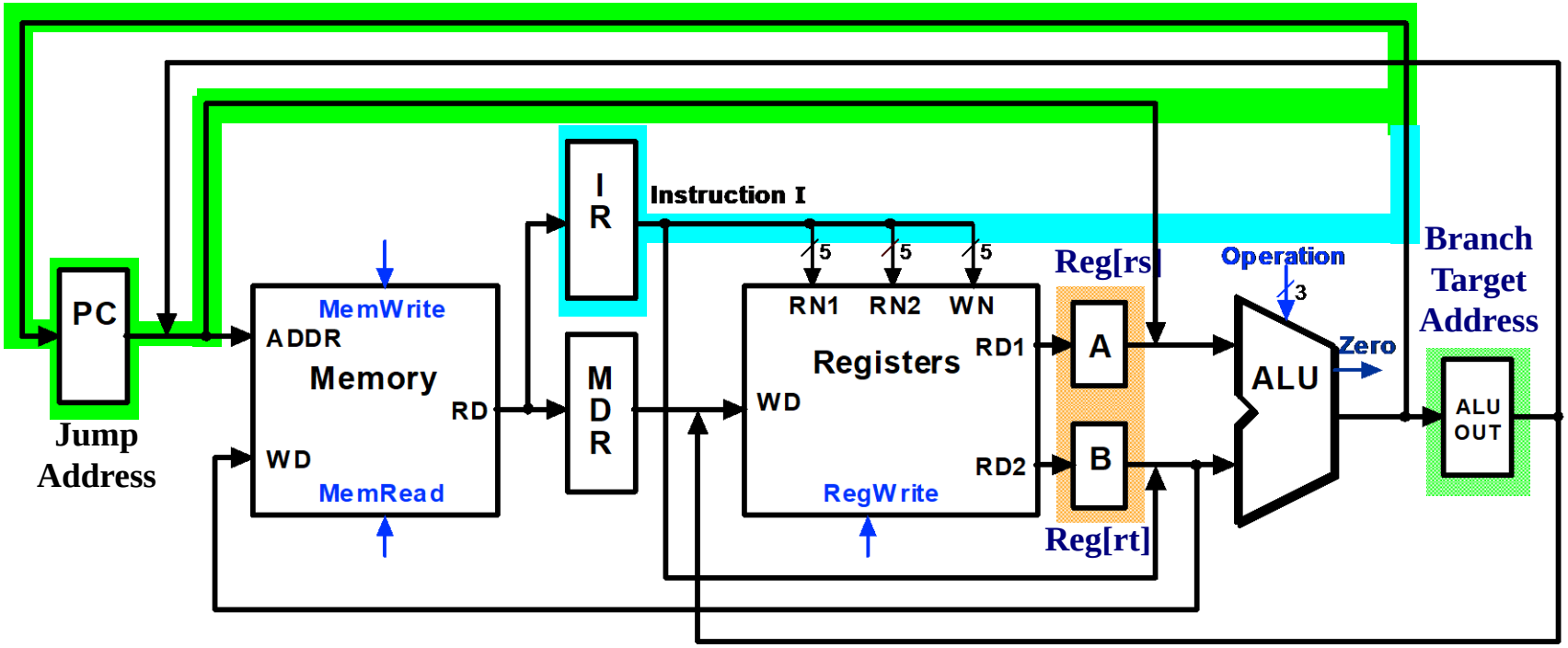
Multicycle Execution Step (3): Branch Instructions

if (A == B) PC = ALUOut;



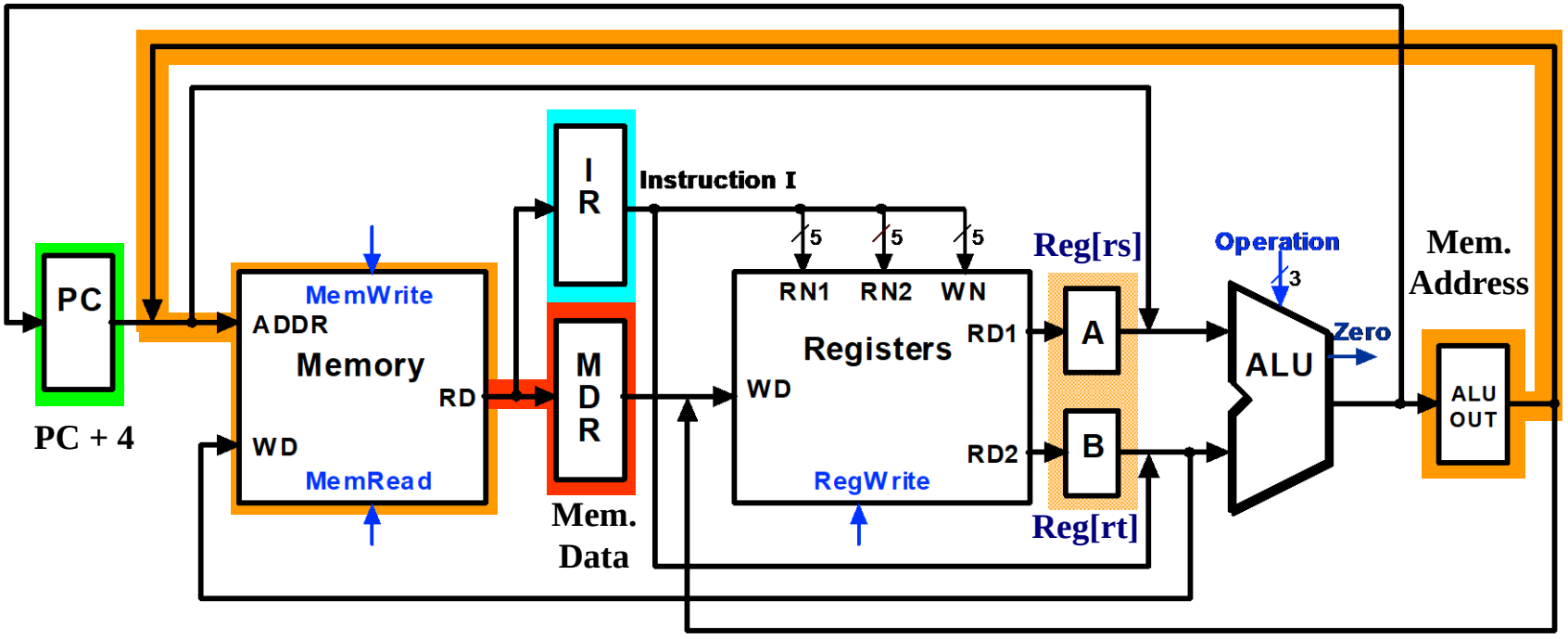
Multicycle Execution Step (3): Jump Instruction

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



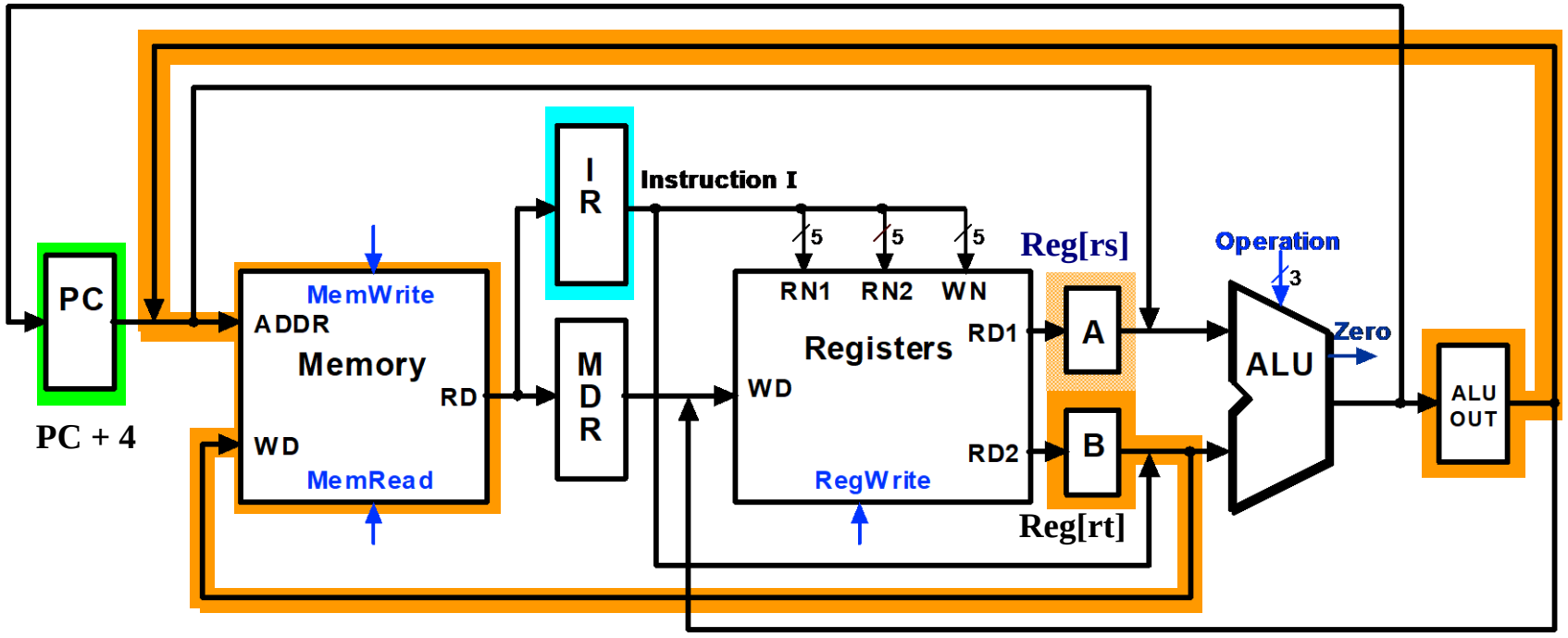
Multicycle Execution Step (4): Memory Access - Read (lw)

MDR = Memory[ALUOut];



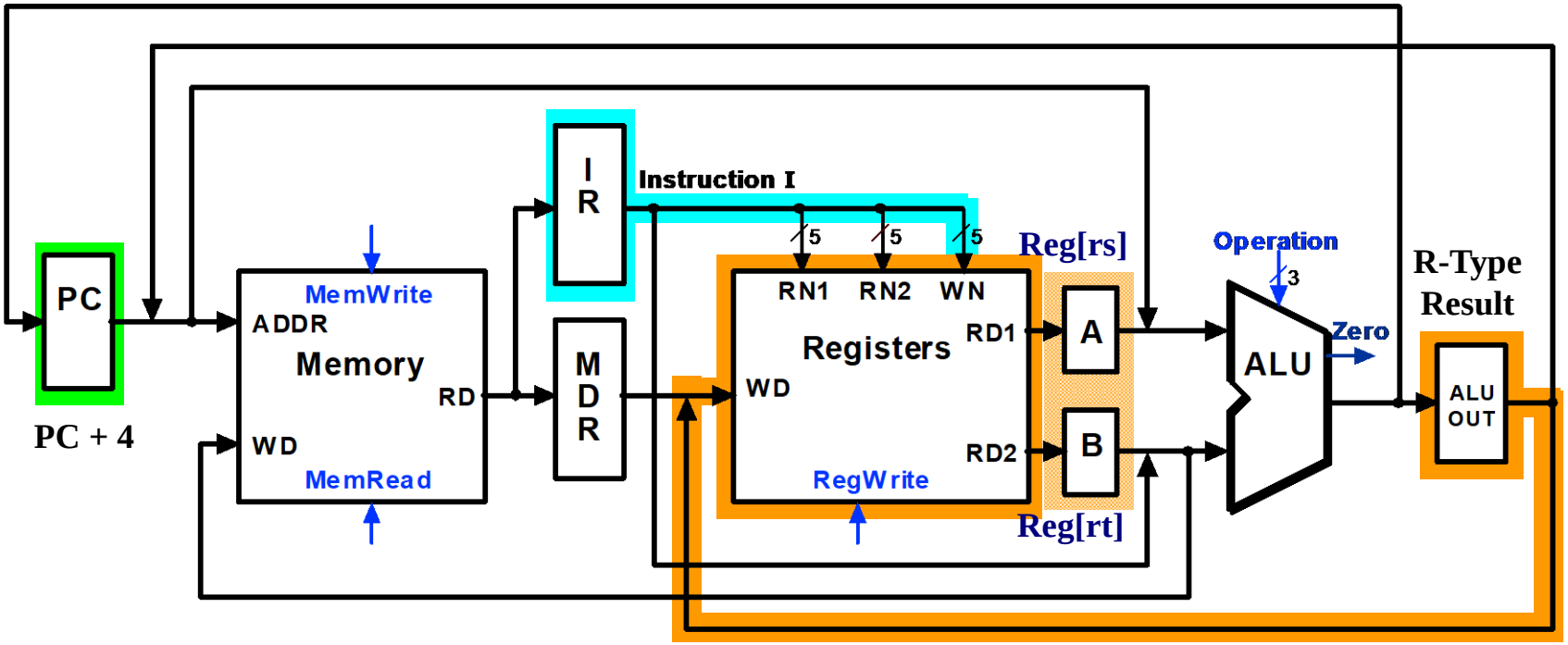
Multicycle Execution Step (4): Memory Access - Write (sw)

Memory[ALUOut] = B;



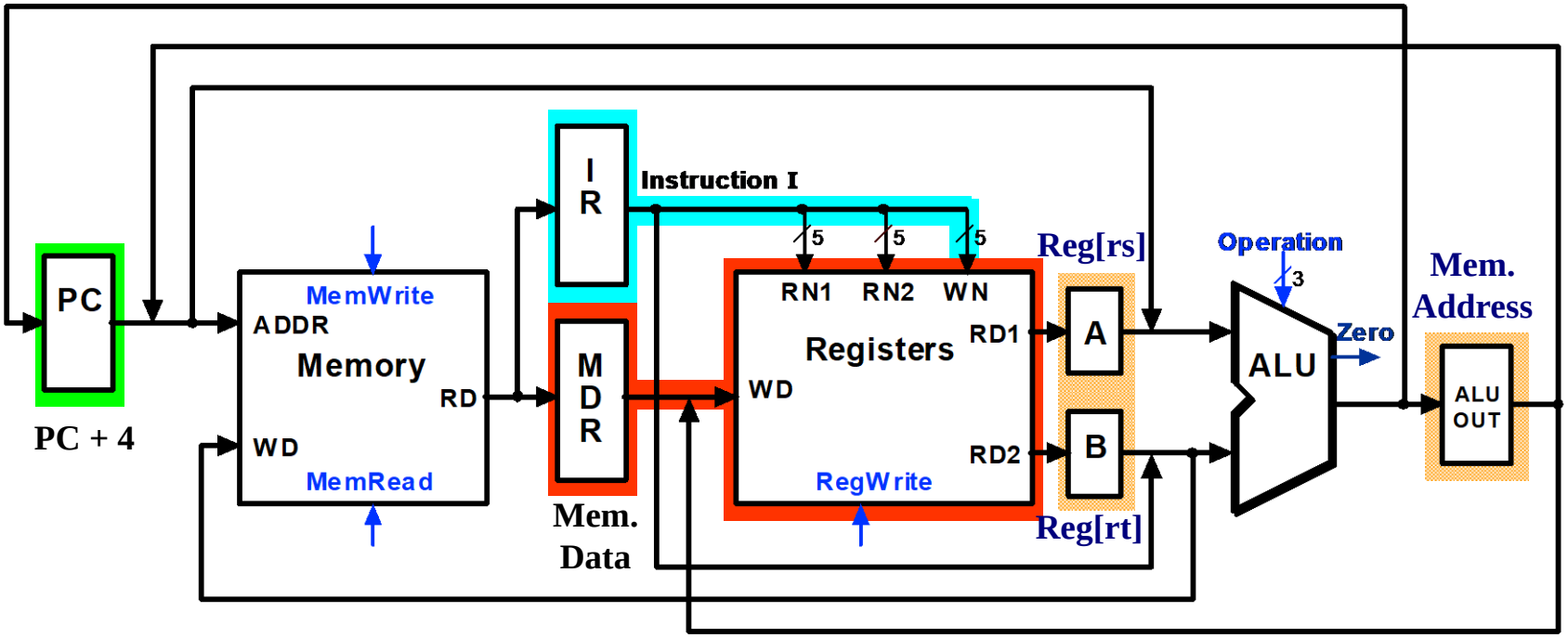
Multicycle Execution Step (4): ALU Instruction (R-Type)

Reg[IR[15:11]] = ALUOUT

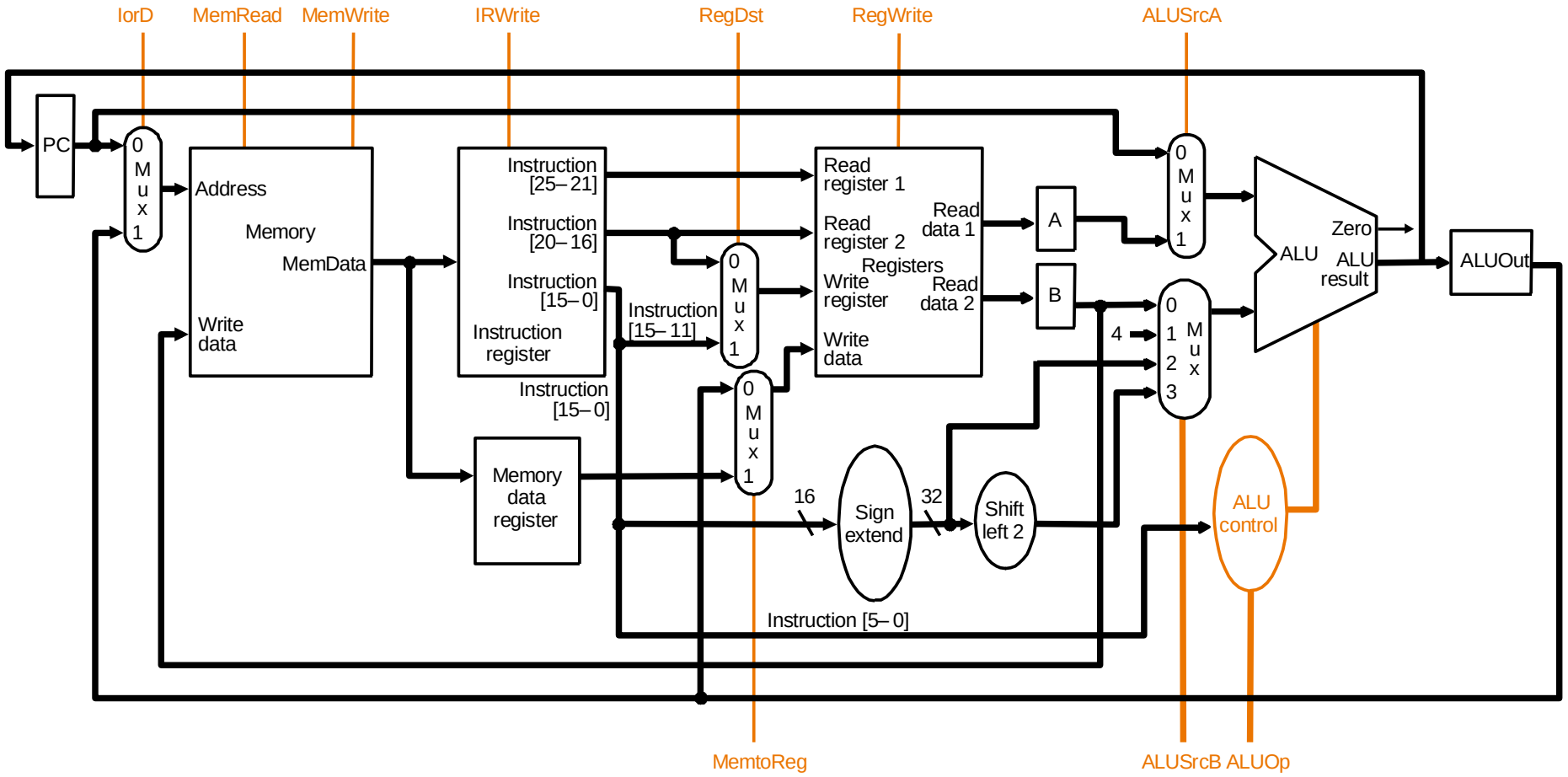


Multicycle Execution Step (5): Memory Read Completion (lw)

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

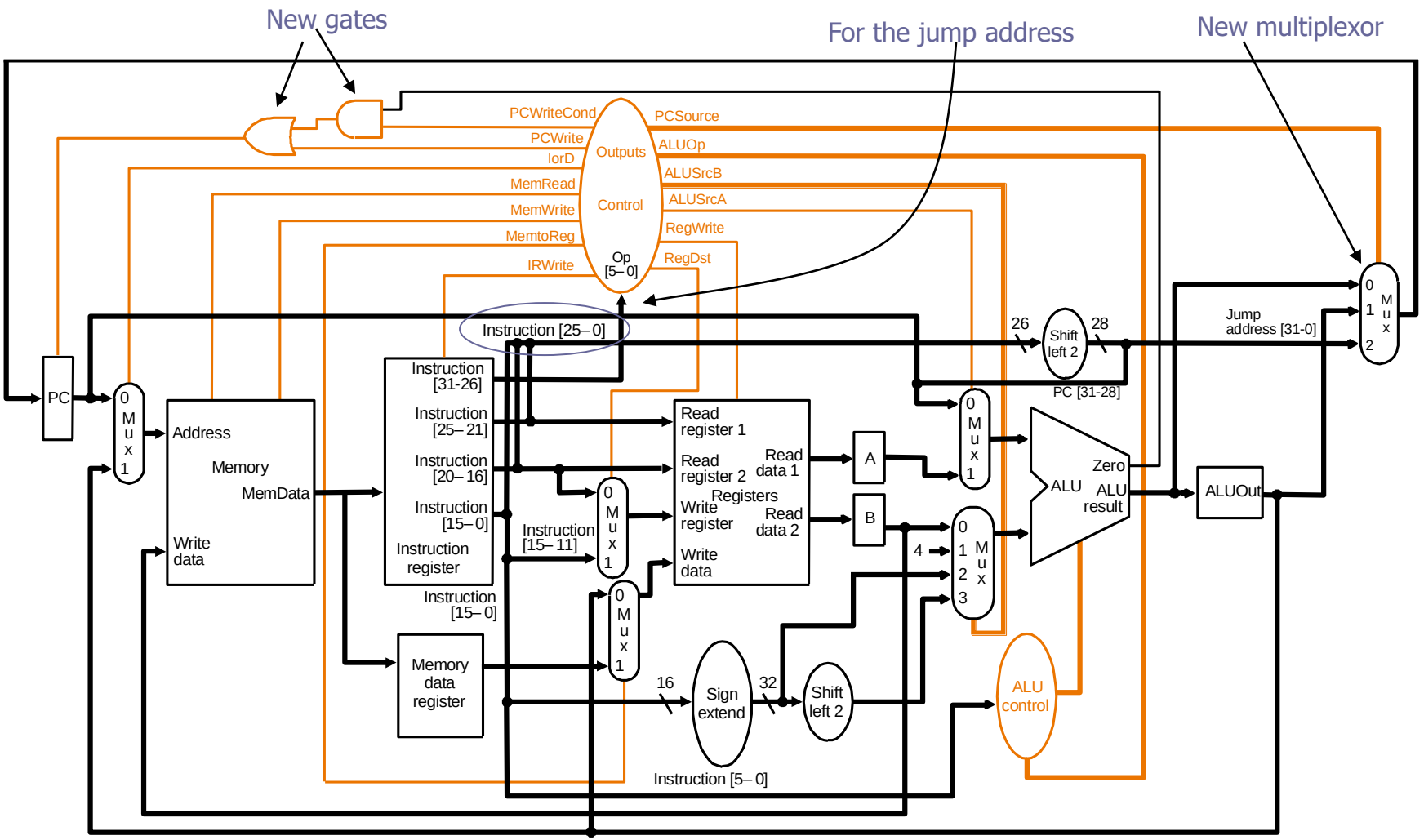


Multicycle Datapath with Control I



… with control lines and the ALU control block added – *not all* control lines are shown

Multicycle Datapath with Control II



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

The actions caused by the setting of each control line

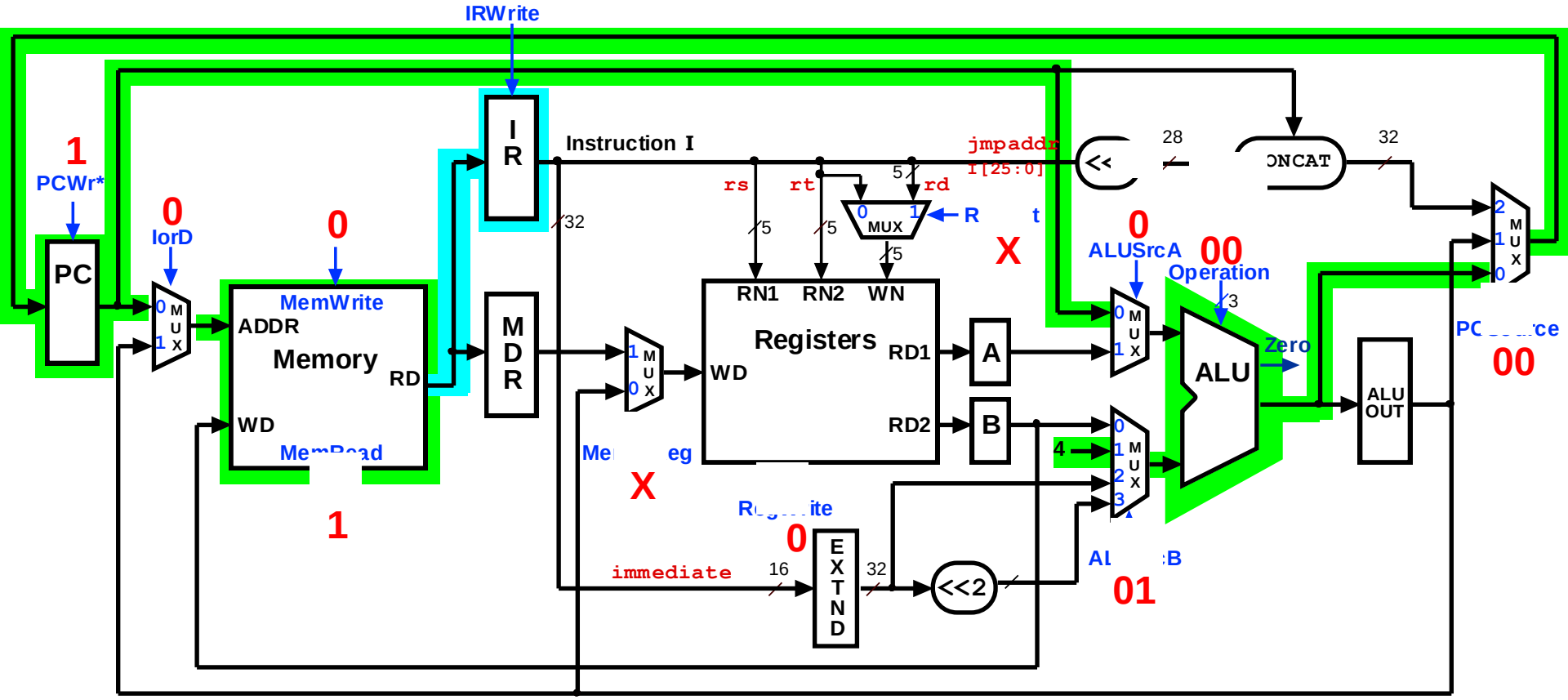
Signal name	Effect when deasserted	Effect when asserted
RegDst	The register file destination number for the Write register comes from the rt field.	The register file destination number for the Write register comes from the rd field.
RegWrite	None.	The general-purpose register selected by the Write register 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.

The actions caused by the setting of each control line

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.

Multicycle Control Step (1): Fetch

IR = Memory[I |;
PC = PC + 4;

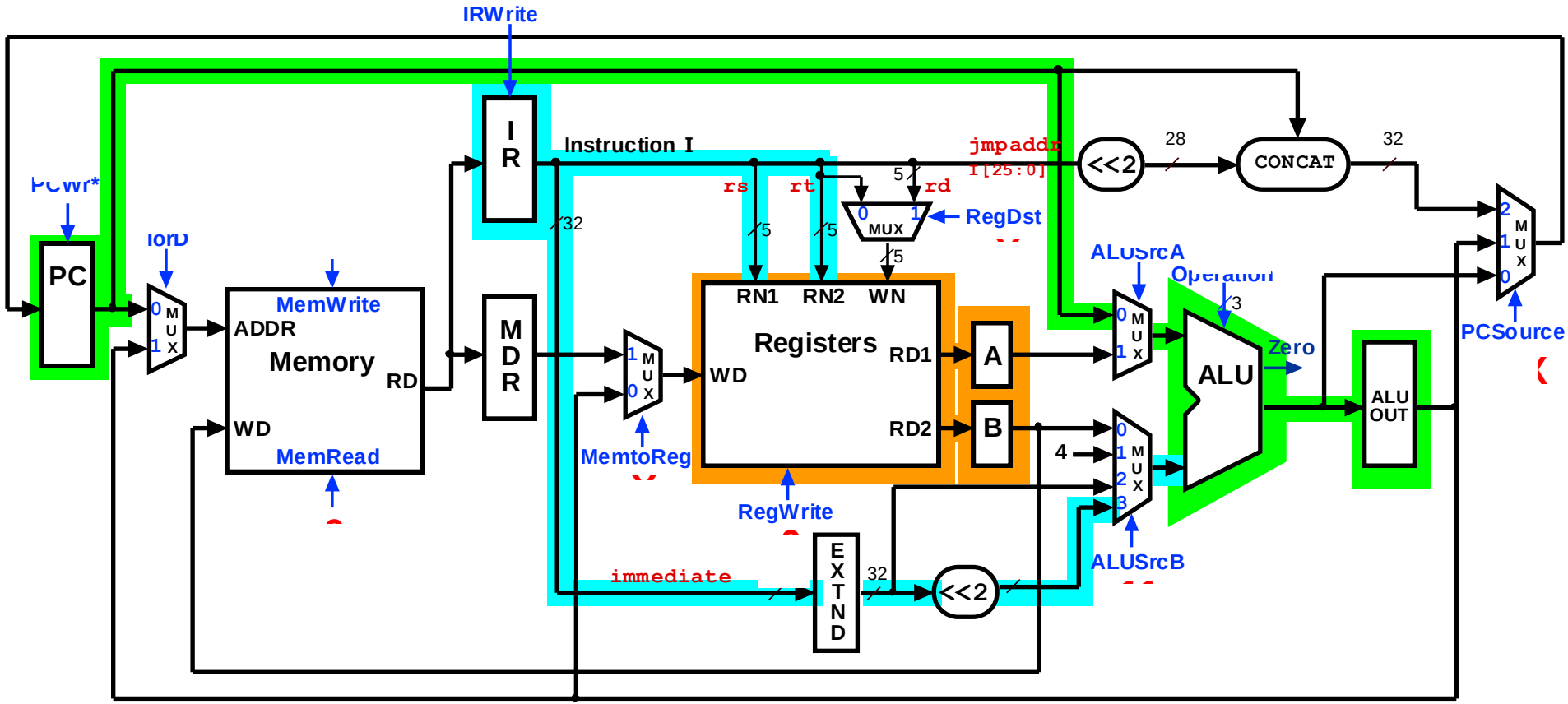


Multicycle Control Step (2): Instruction Decode & Register Fetch

A = Reg[IR[25-21]]; (A = Reg[rs])

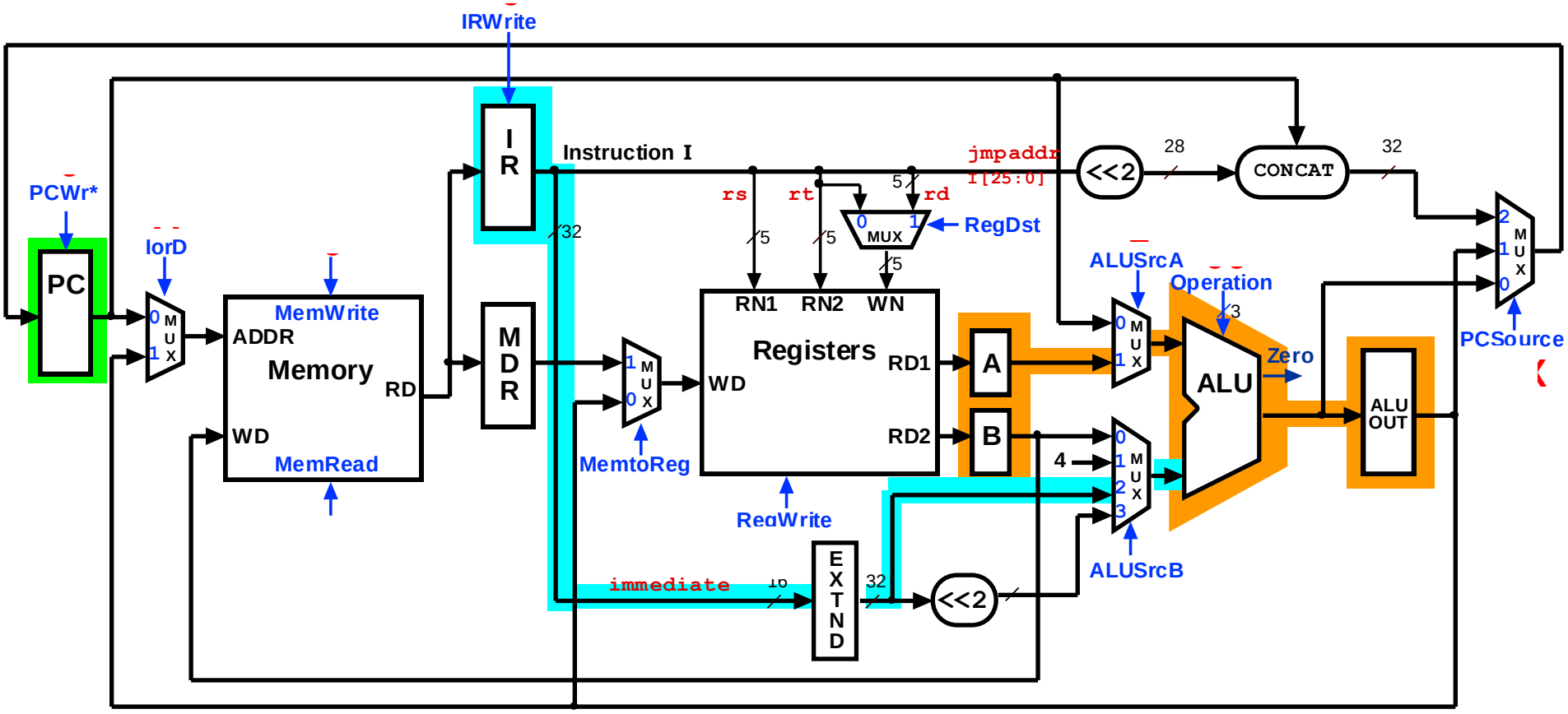
B = Reg[IR[20-15]]; (B = Reg[rt])

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



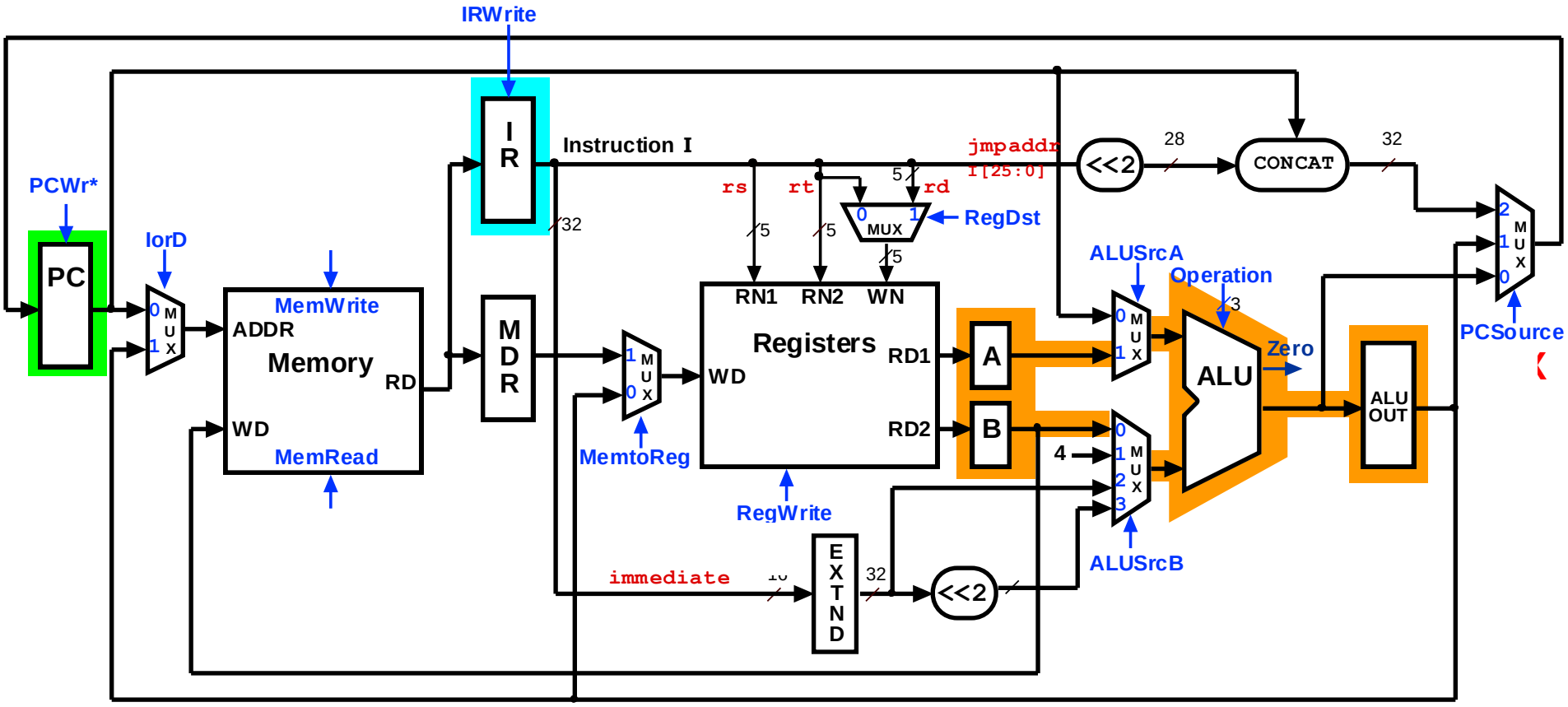
Multicycle Control Step (3): Memory Reference Instructions

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



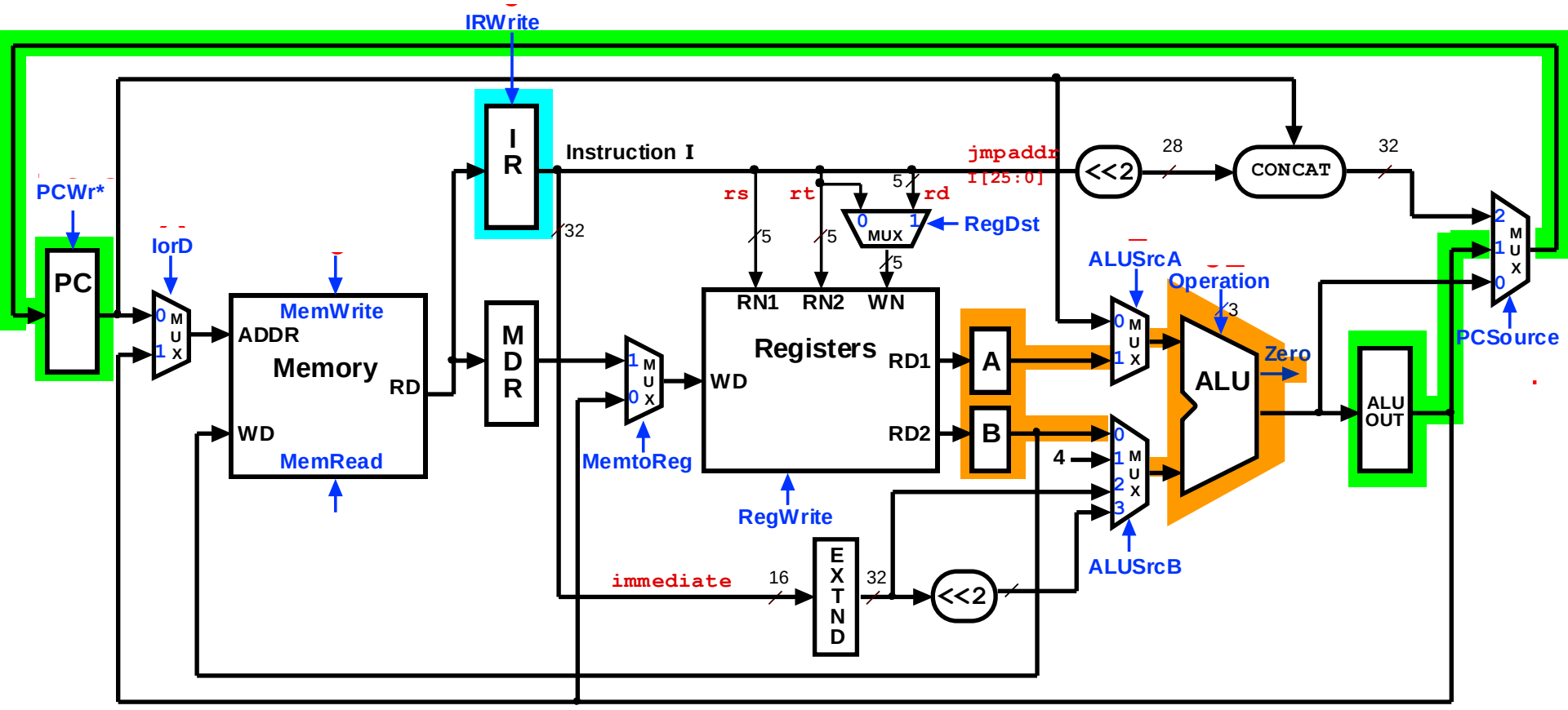
Multicycle Control Step (3): ALU Instruction (R-Type)

ALUOut = A o B;



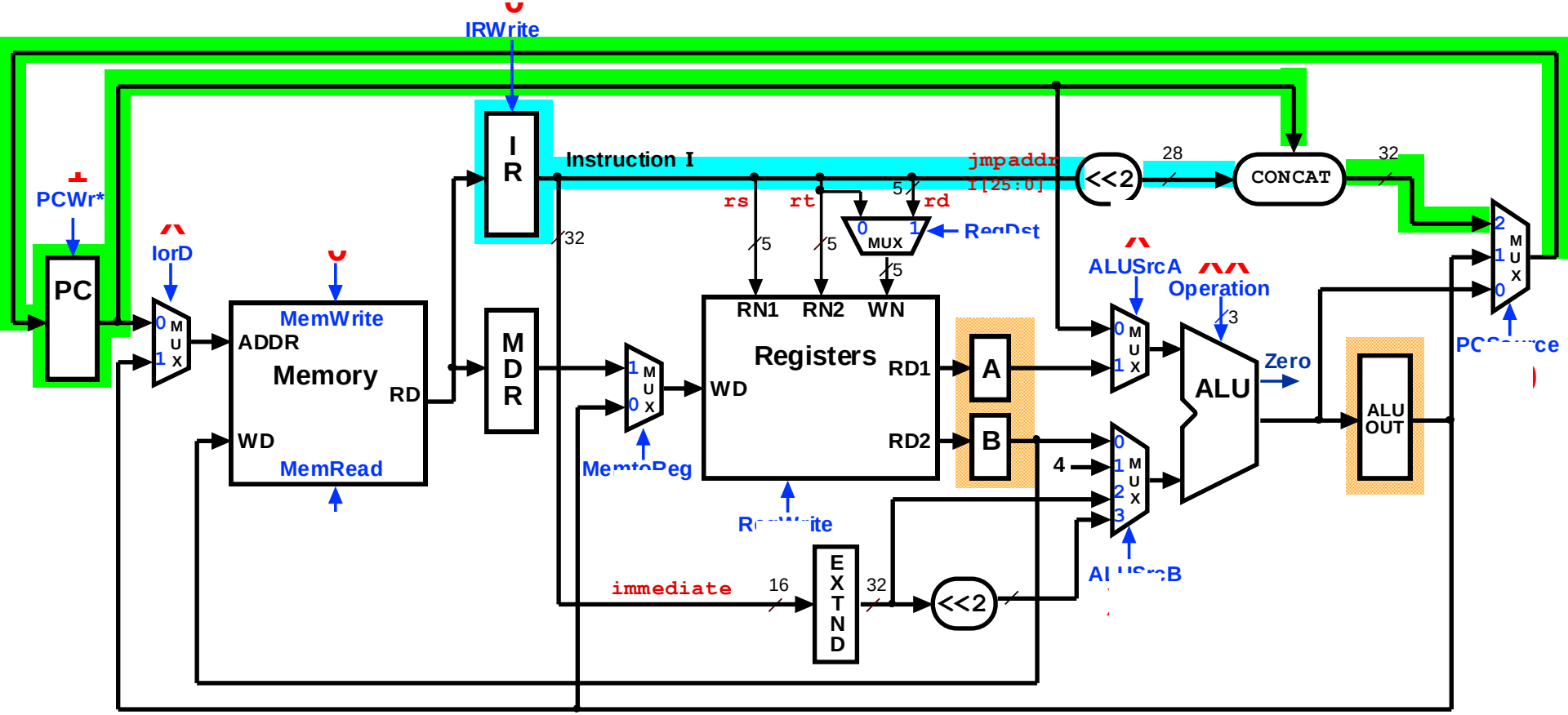
Multicycle Control Step (3): Branch Instructions

if (A == B) PC ALUOut;



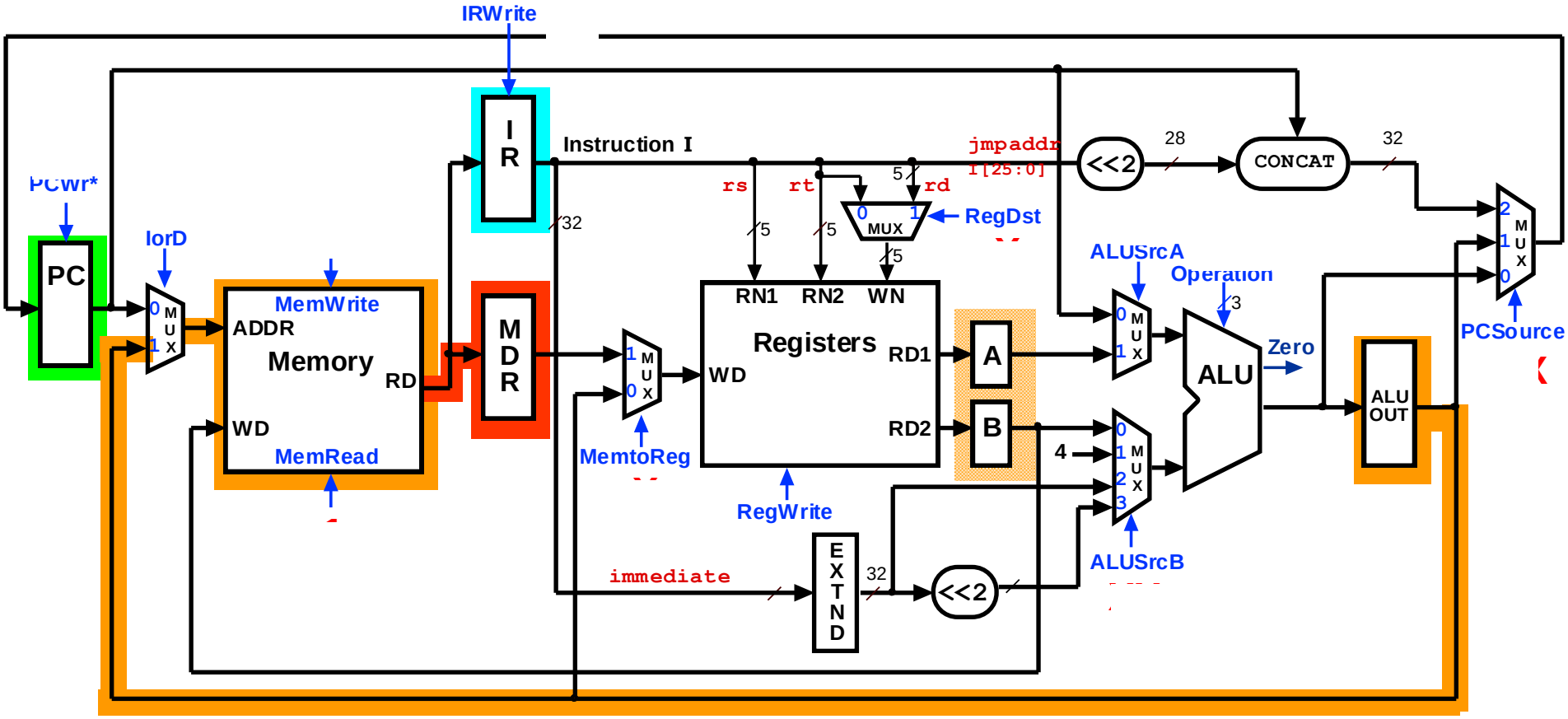
Multicycle Execution Step (3): Jump Instruction

PC = PC[21-28] concat (IR[25-0] << 2);



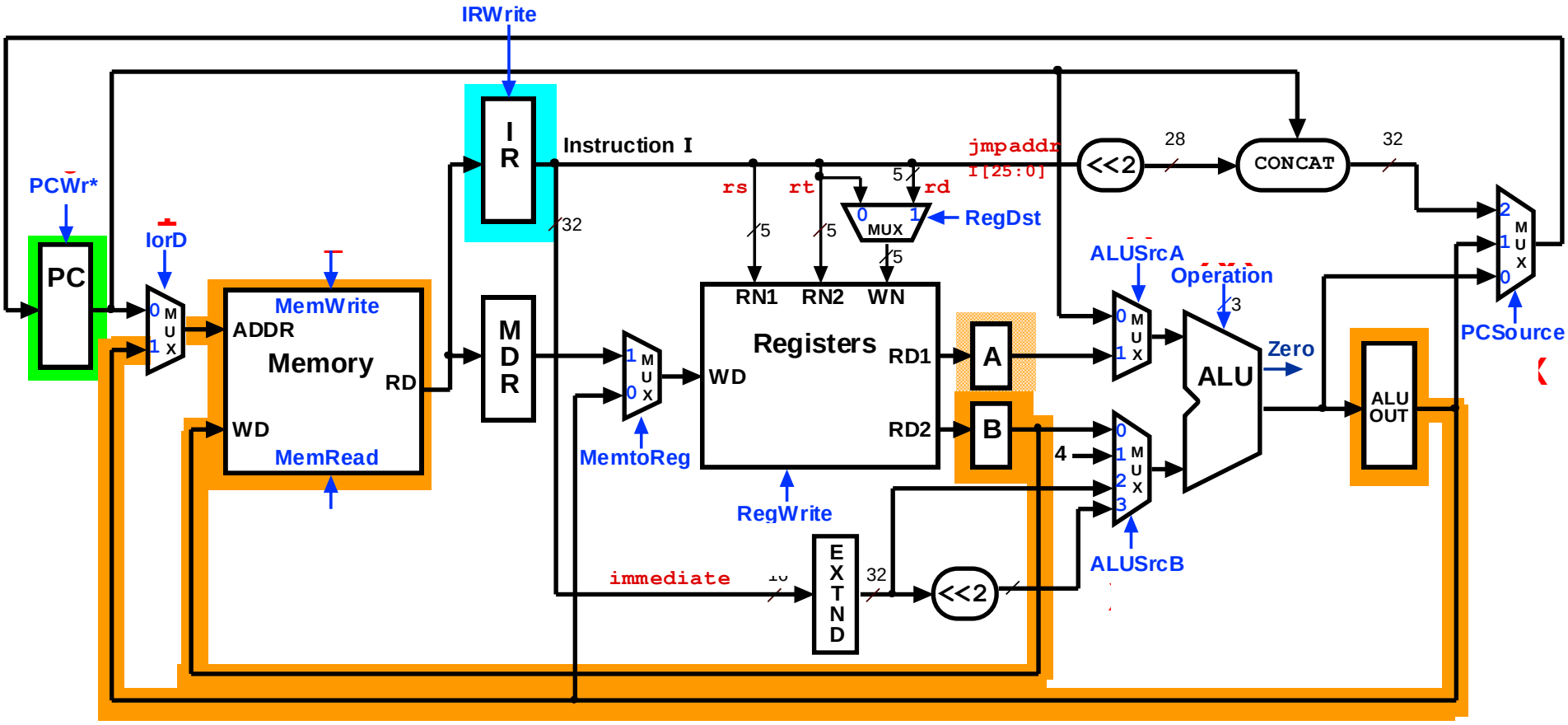
Multicycle Control Step (4): Memory Access - Read (lw)

MDR = Memory[ALUOut];



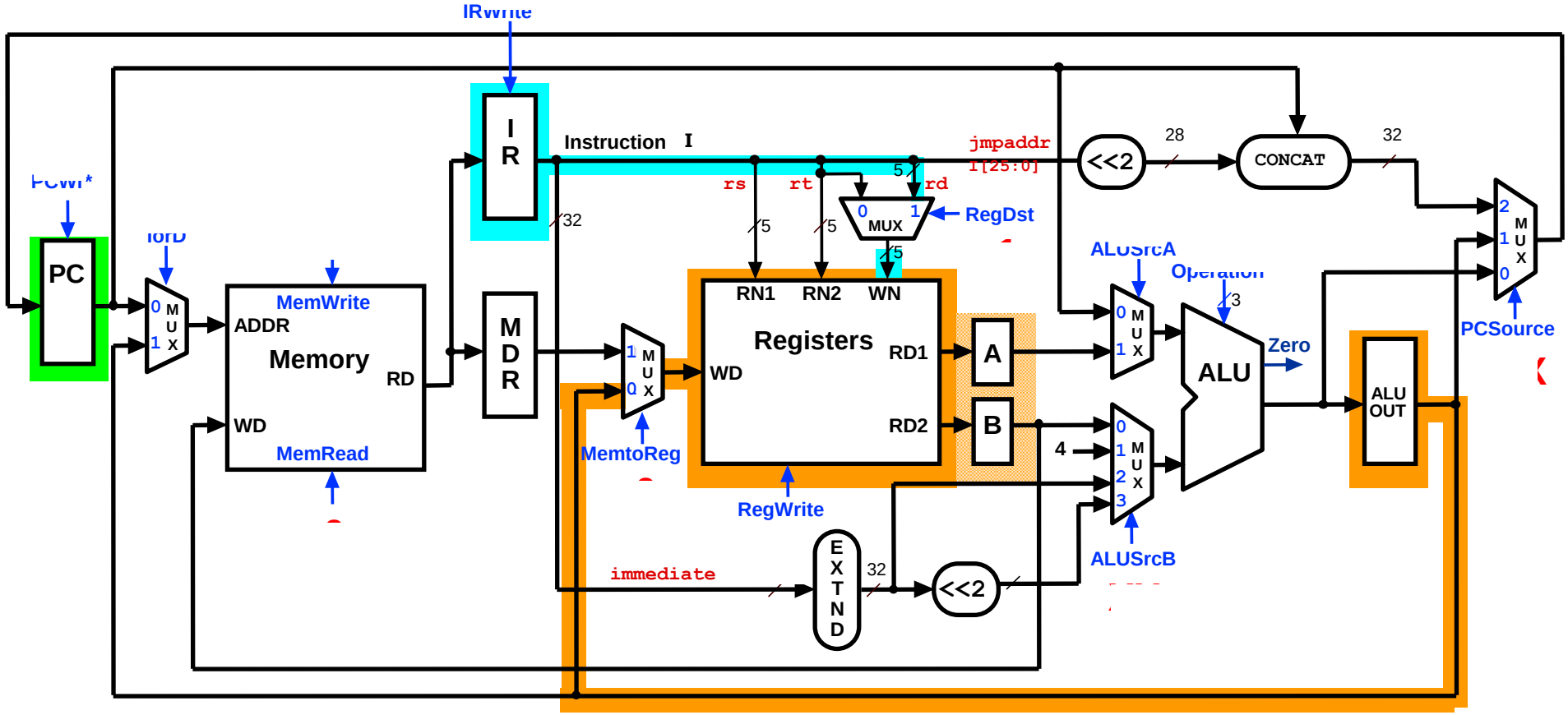
Multicycle Execution Steps (4)
Memory Access - Write (sw)

Memory[ALUOut] = B;



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

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



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

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