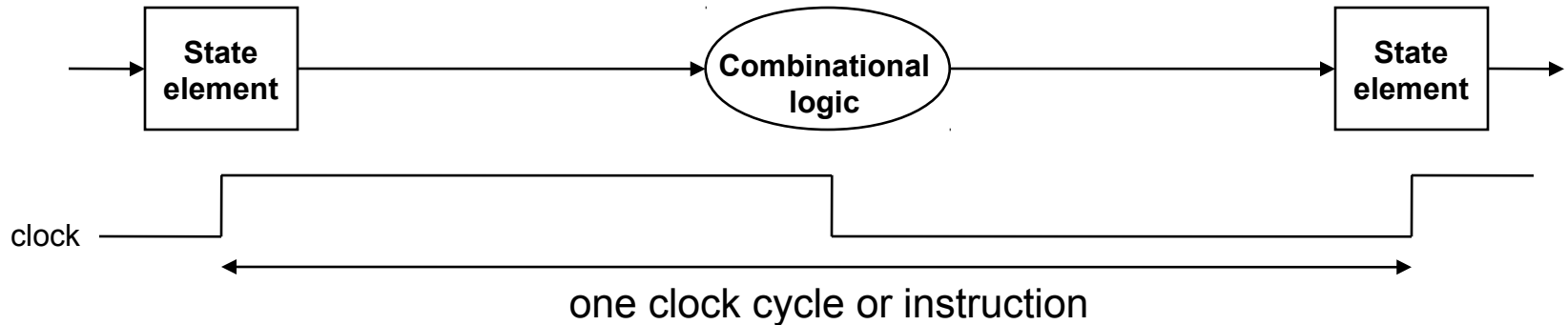


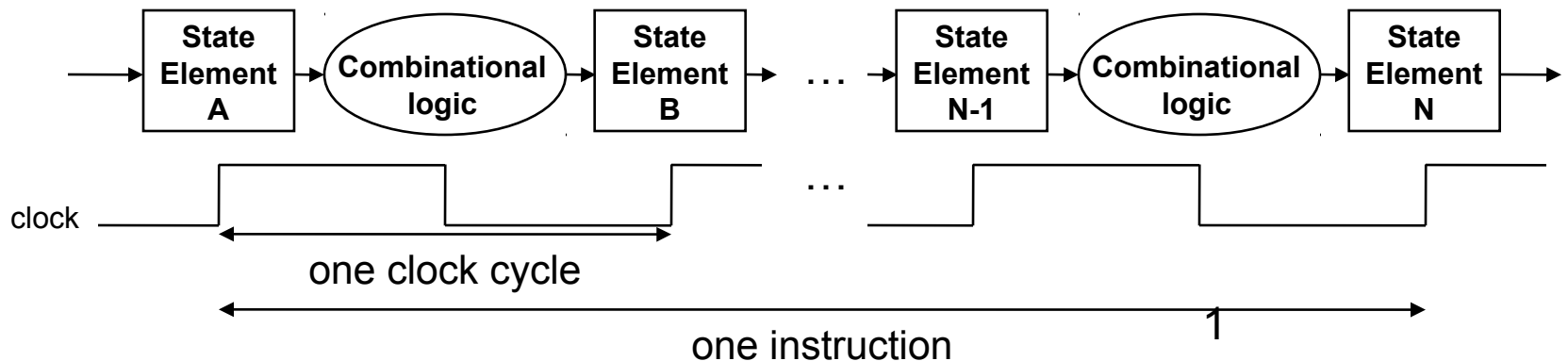
Multi-cycle Approach

- Single cycle CPU



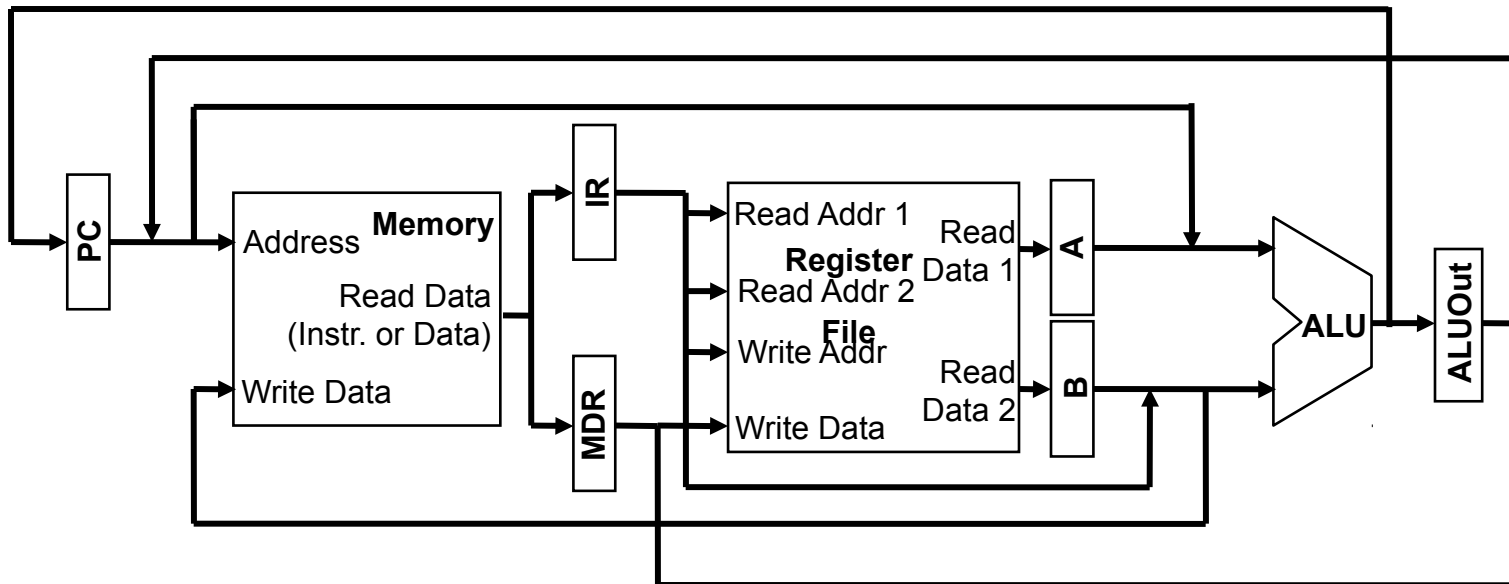
- Multi-cycle CPU

– Requires state elements to hold intermediate values



Multi-cycle Approach

- Each cycle must
 - Store values needed in a later cycle of the current instruction in an internal register. All except IR hold data for one clock cycle.



IR – Instruction Register

MDR – Memory Data Register

A, B – Register File data

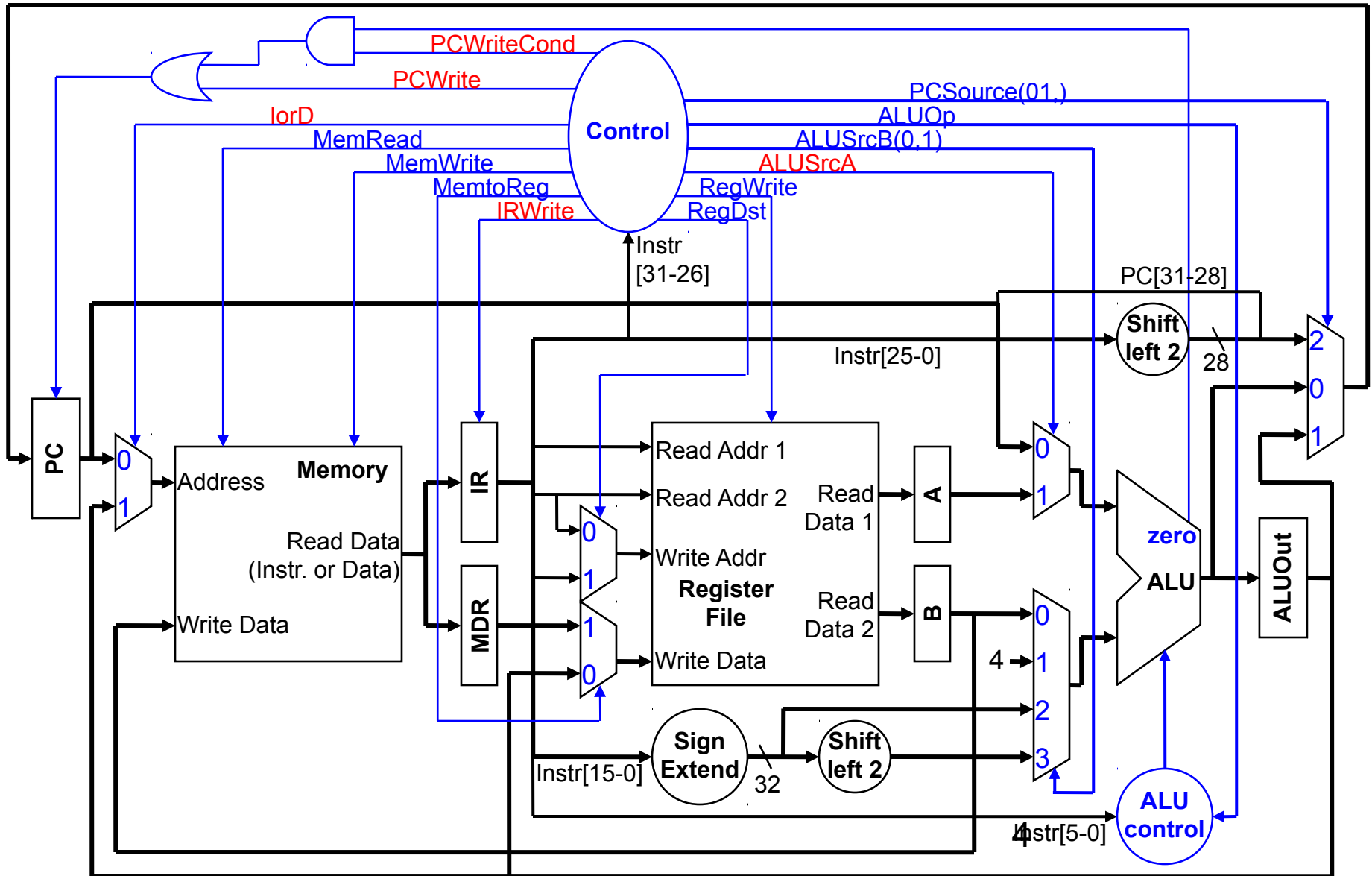
ALUOut – ALU Result Register

- Store values needed by subsequent instructions in the register file or memory

Multi-cycle Control

- New control signals needed
 - **PCWriteCond** is set during a beq instruction
 - Formerly called Branch signal
 - **PCWrite** is set to write PC
 - Unconditional write signal needed during Fetch cycle
 - **lorD** controls what address is used for the memory
 - PC holds address for fetch cycle
 - ALUOut holds address for memory access instructions
 - **IRWrite** controls when the IR is written
 - **ALUSrcA** control one input to ALU
 - rs register for most operations
 - PC for branch instructions
 - Old ALUSrc renamed ALUSrcB and expanded

Multi-cycle Control and Datapath



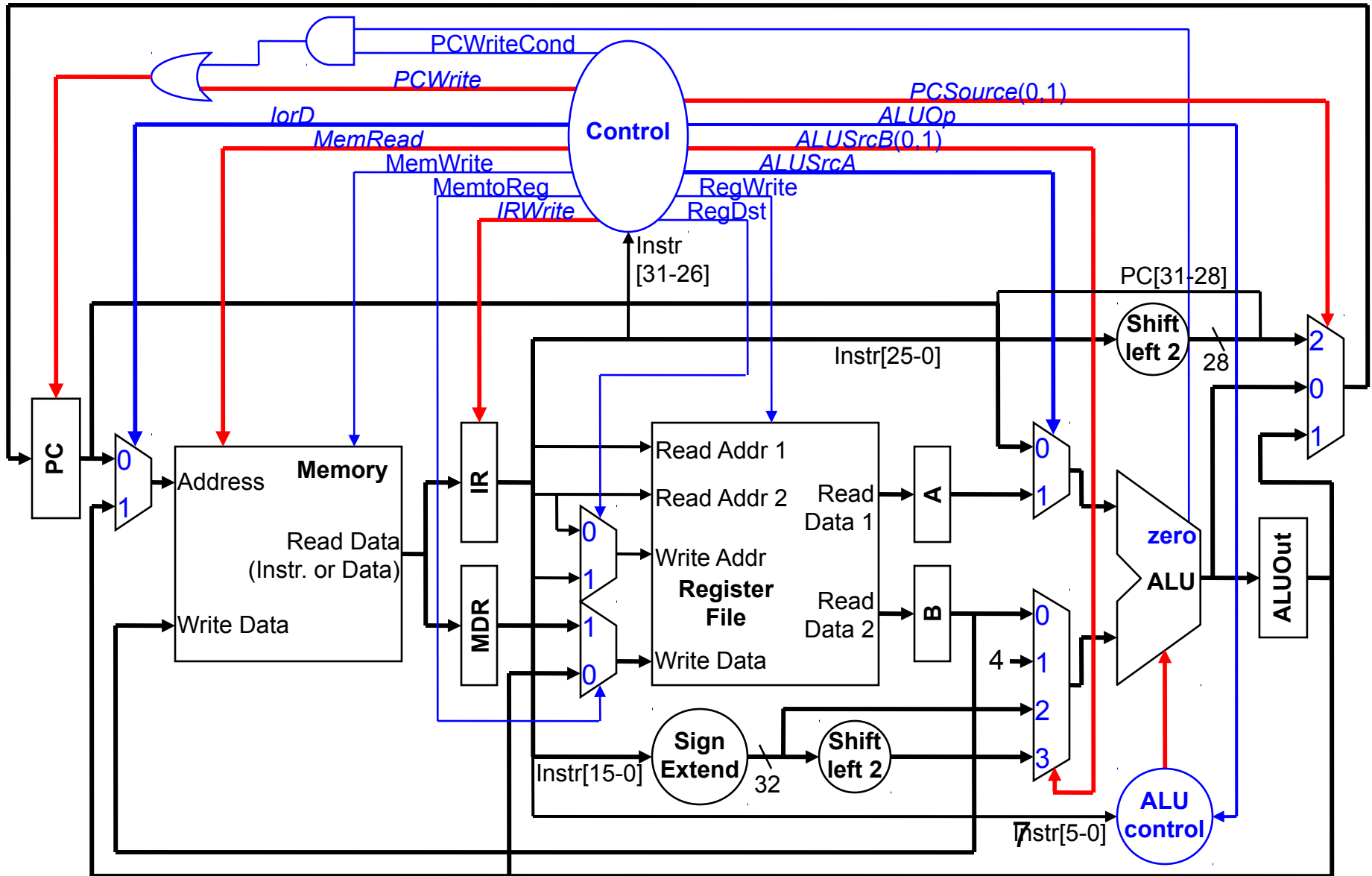
Multi-cycle Steps

- Instruction Fetch
- Decode and Register Fetch
- Execution
- Memory Access
- Write Register File

(1) Instruction Fetch Cycle

- Increment PC using ALU
 - $PC = PC + 4$
- Read instruction from memory
 - $IR = M[PC]$
- Control signals must
 - Select memory address source
 - Enable memory reading
 - Enable PC and IR write
 - Select PC source
 - Select ALU input as PC and constant 4
 - Select ALU operation (addition)

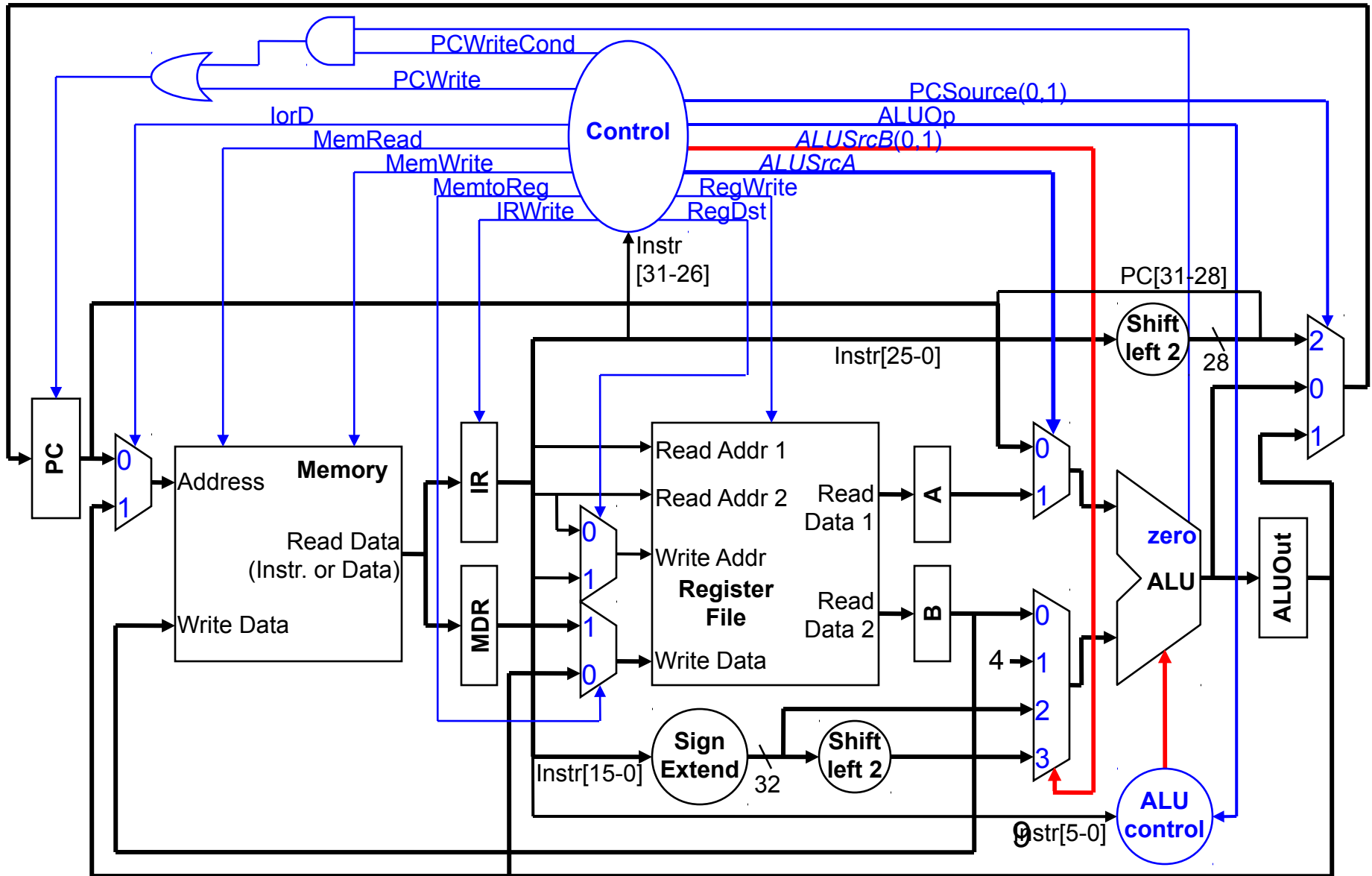
Instruction Fetch



(2) Decode and Register Fetch Cycle

- Read register values
 - $A = R[rs]$, $B = R[rt]$
- Compute branch destination
 - $ALUOut = PC + \text{sign extended immediate value}$
- Prepare for next step based on instruction
- Control signals must
 - Select ALU inputs as PC and immediate value
 - Select ALU operation (addition)

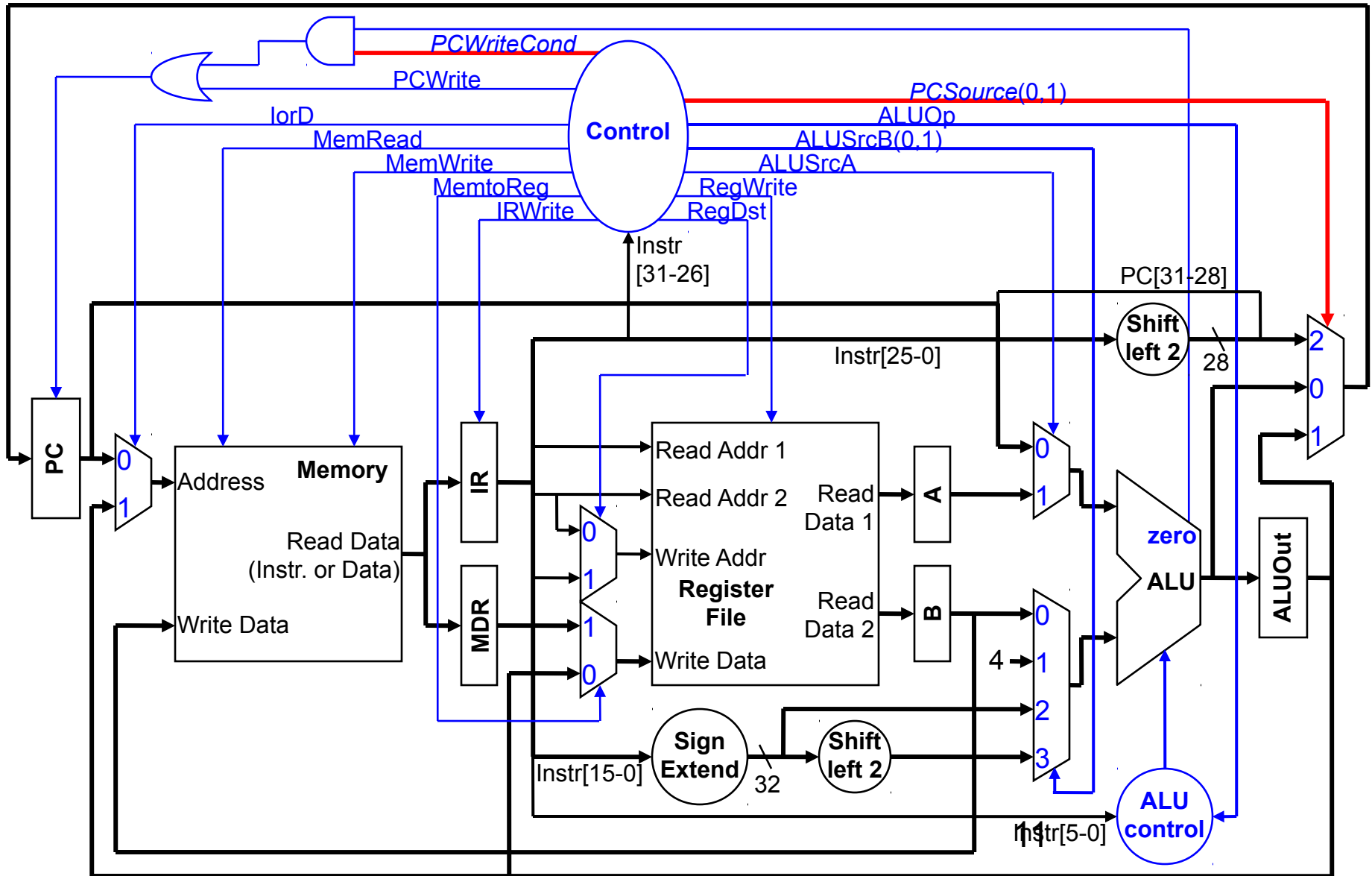
Decode and Register Fetch



(3) Execution Cycle

- Functionality varies with instructions
 - Memory reference
 - Compute address
 - $ALUOut = A + \text{sign extended immediate}$
 - R-type
 - Compute operation
 - $ALUOut = A \text{ op } B$
 - Branch
 - Store new PC if needed
 - $PC = ALUOut$
 - ALUOut contains branch destination from previous cycle
- Control signals will depend on instruction type
 - Mem/R-type: Select ALU input and operation
 - Branch: Select PC source and set PC write control signal if needed

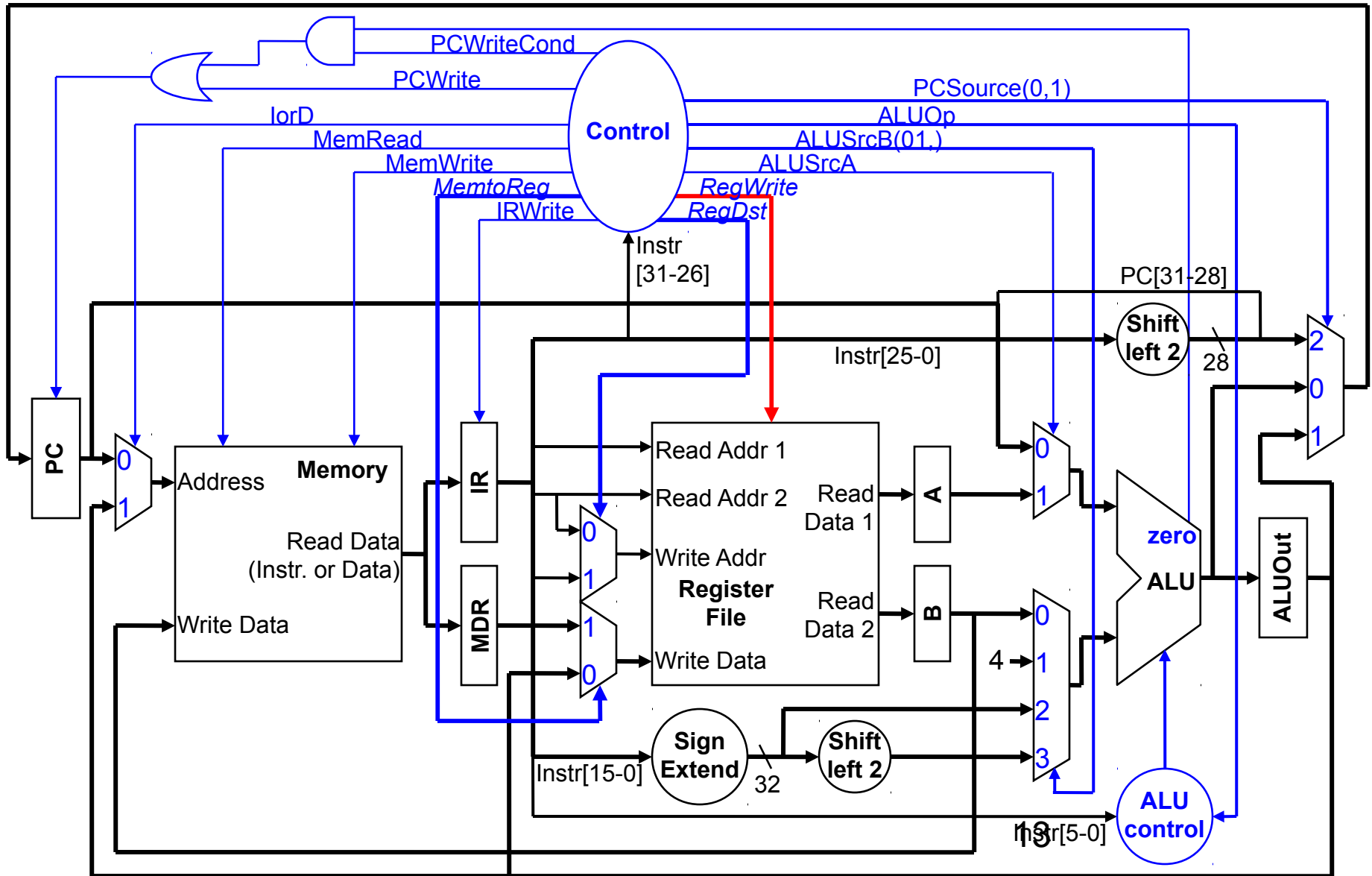
Execute Branch



(4) Memory Access Cycle

- Functionality varies with instructions
 - Memory reference
 - Read memory (lw) or write memory (sw)
 - $MDR = M[ALUOut]$ or $M[ALUOut] = B$
 - R-type
 - Write result to register file
 - $R[rd] = ALUOut$
- Control signals will depend on instruction type
 - Memory reference
 - Enable memory read or write
 - Select memory address
 - R-type
 - Select register file write address and data
 - Enable register file write

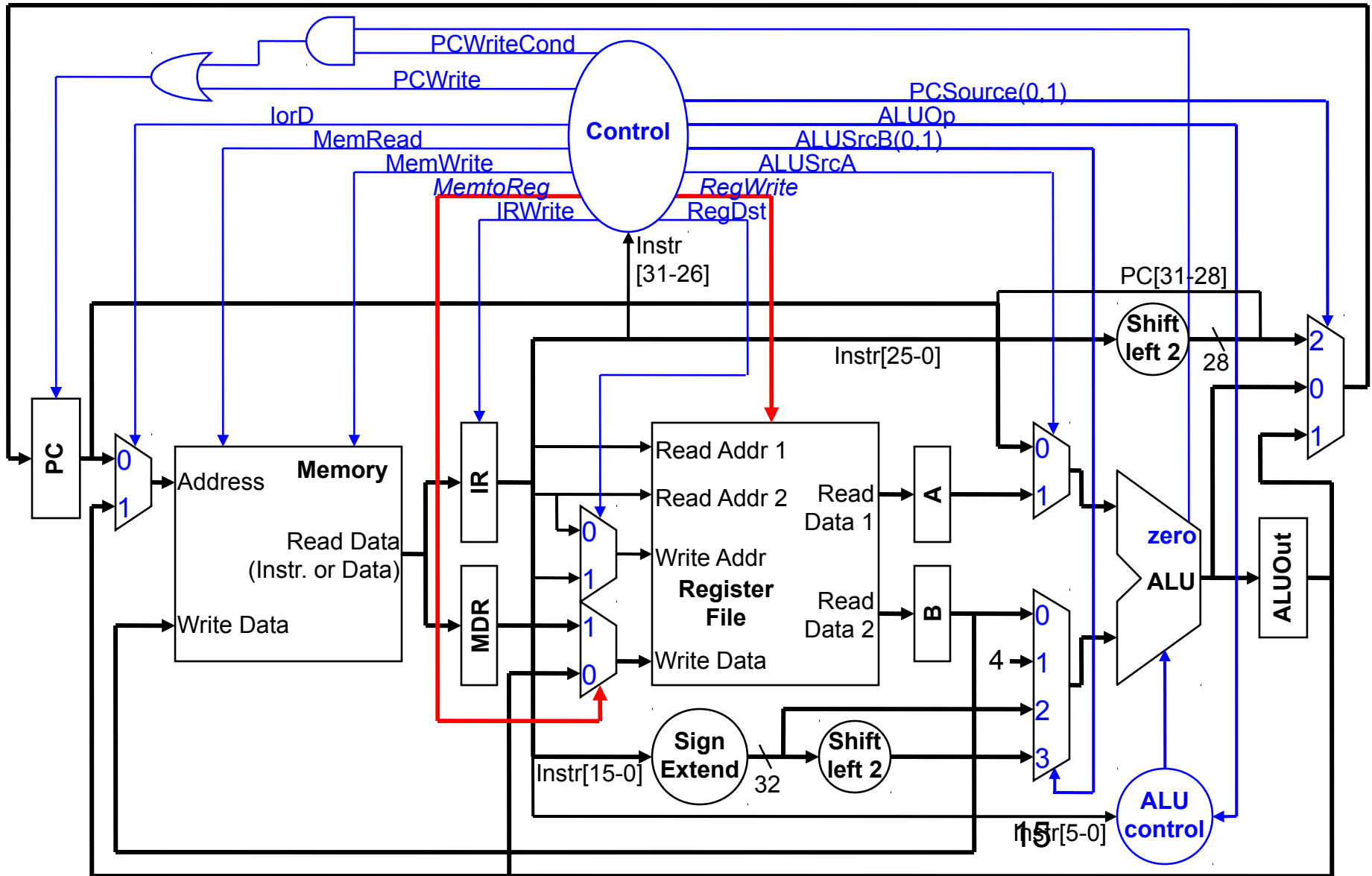
R-Type "Memory Access"



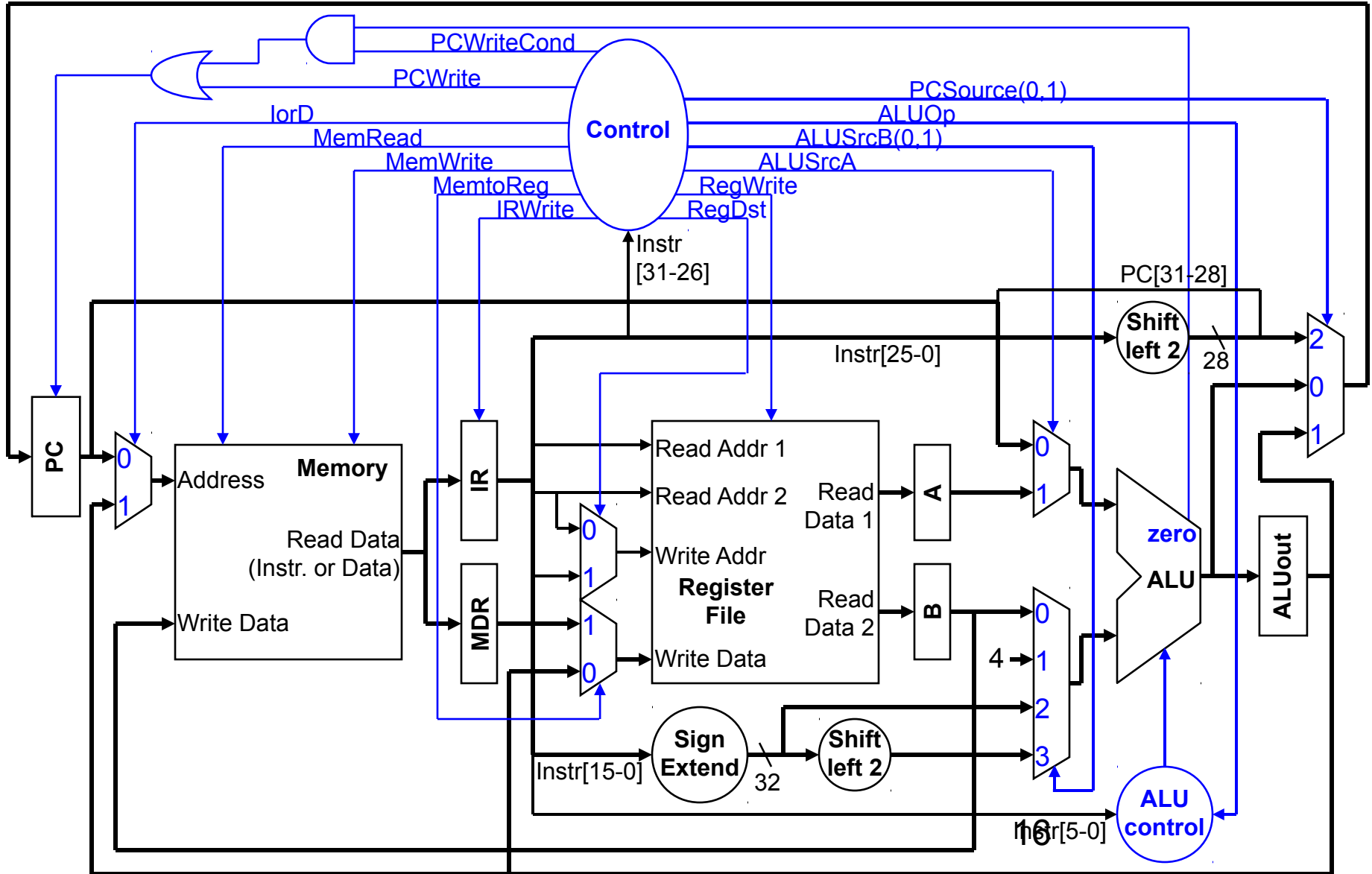
(5) Write Register File Cycle

- Only used by load instructions
- Write memory value to register
 - $\text{Reg}[\text{rt}] = \text{MDR}$
- Control signals must
 - Enable register file write
 - Select register file write address and data

Iw Write Registers



Review

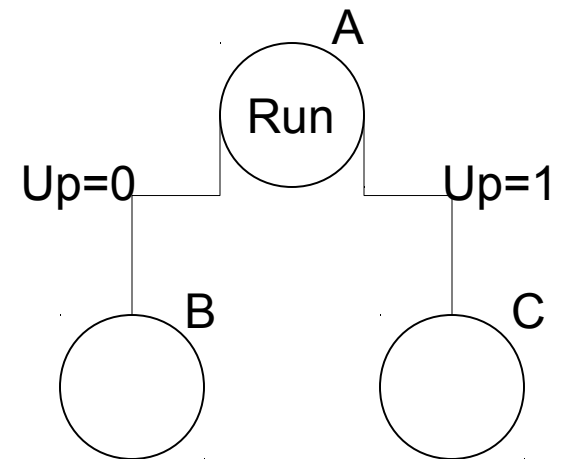


Defining the CPU Control

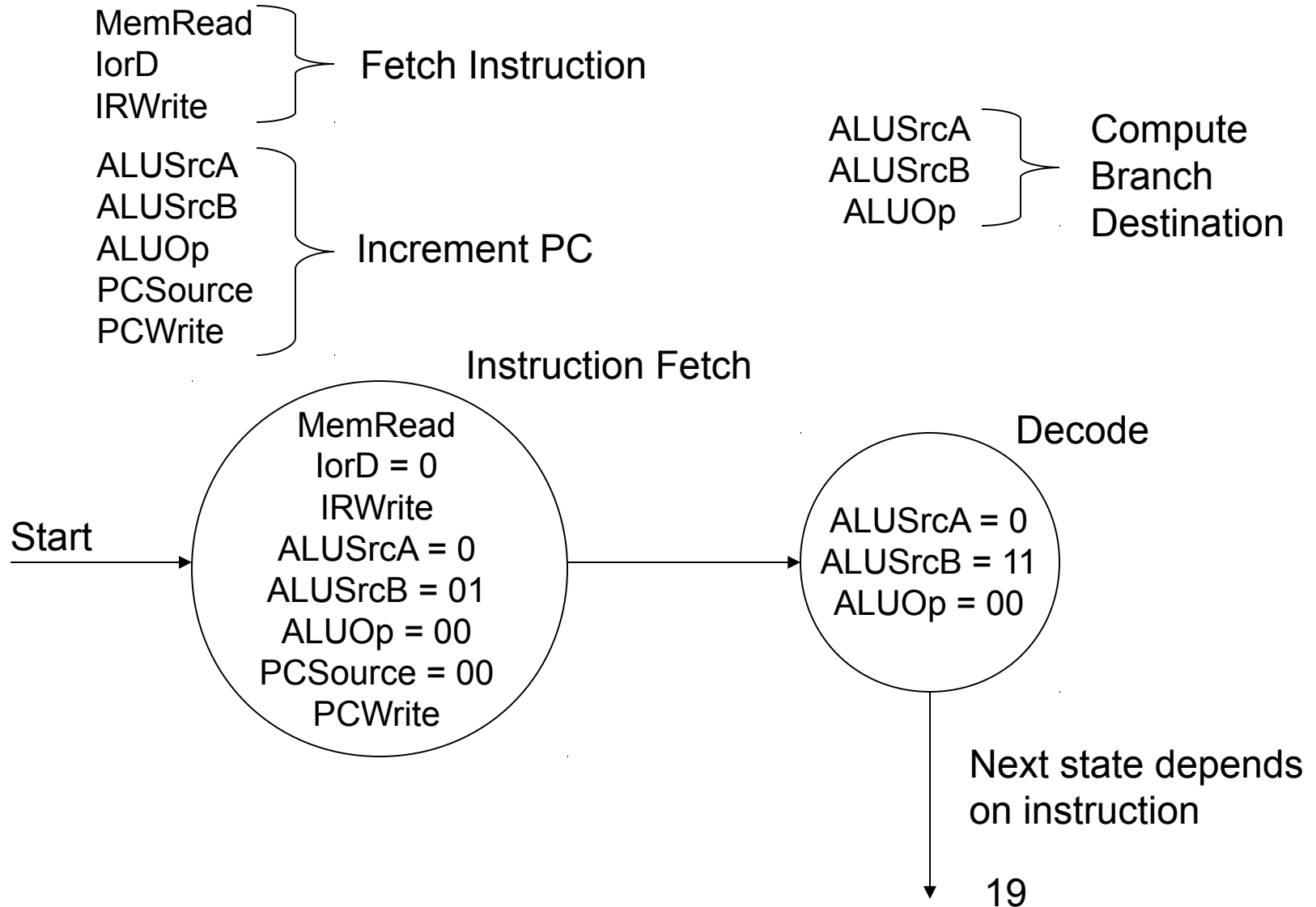
- Use a finite state machine model to design the control
 - Control now has state or memory
 - Action depends on input and current cycle
- Signal names in each state are asserted
 - Asserted signals are set or enabled
 - Unlisted signals are deasserted
- Arcs between states list conditions for transition

• Example State A

- Signal Run is asserted
- Goes to State B when input $Up = 0$
- Goes to State C when input $Up = 1$

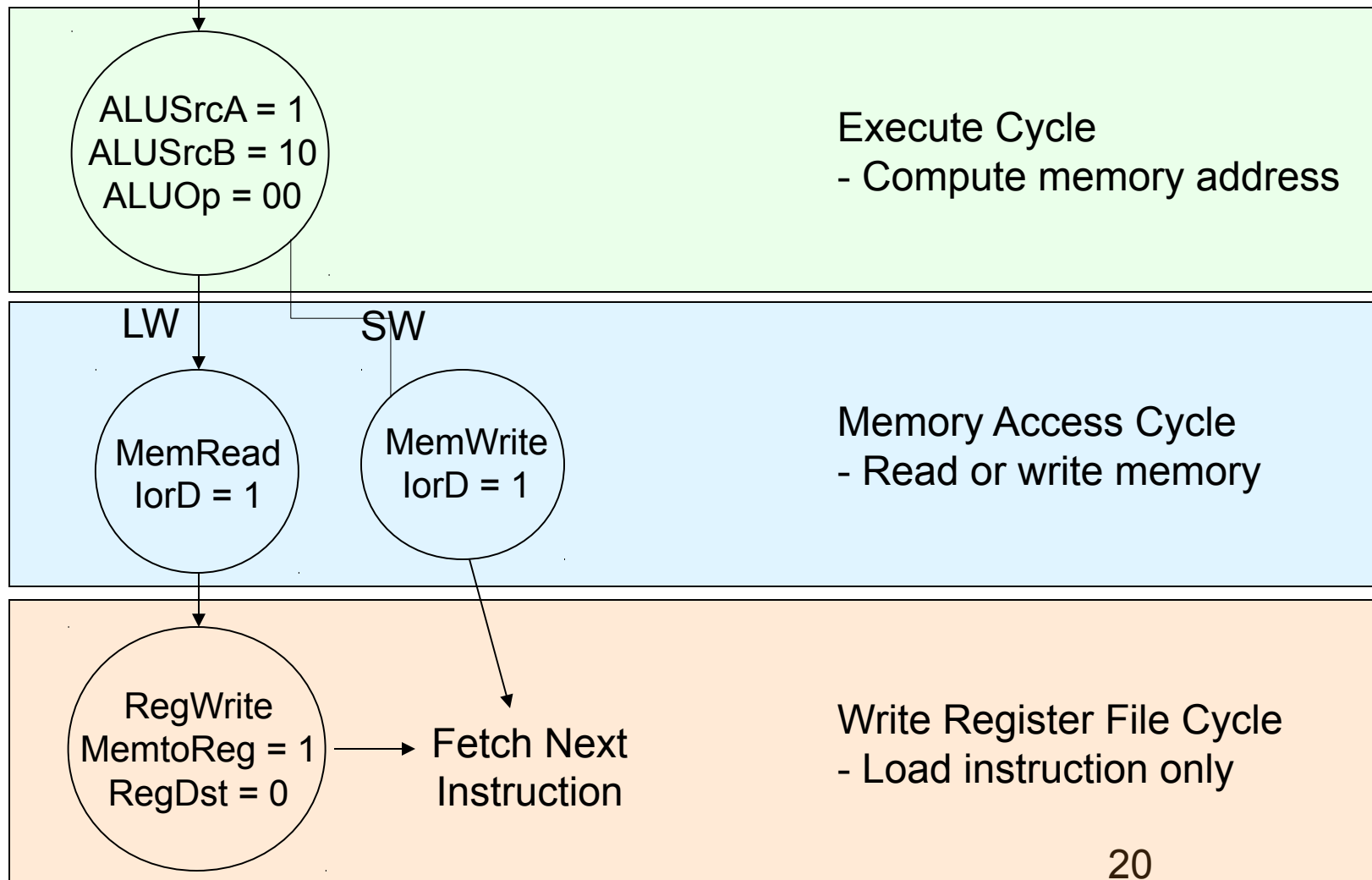


Instruction Fetch and Decode Cycles



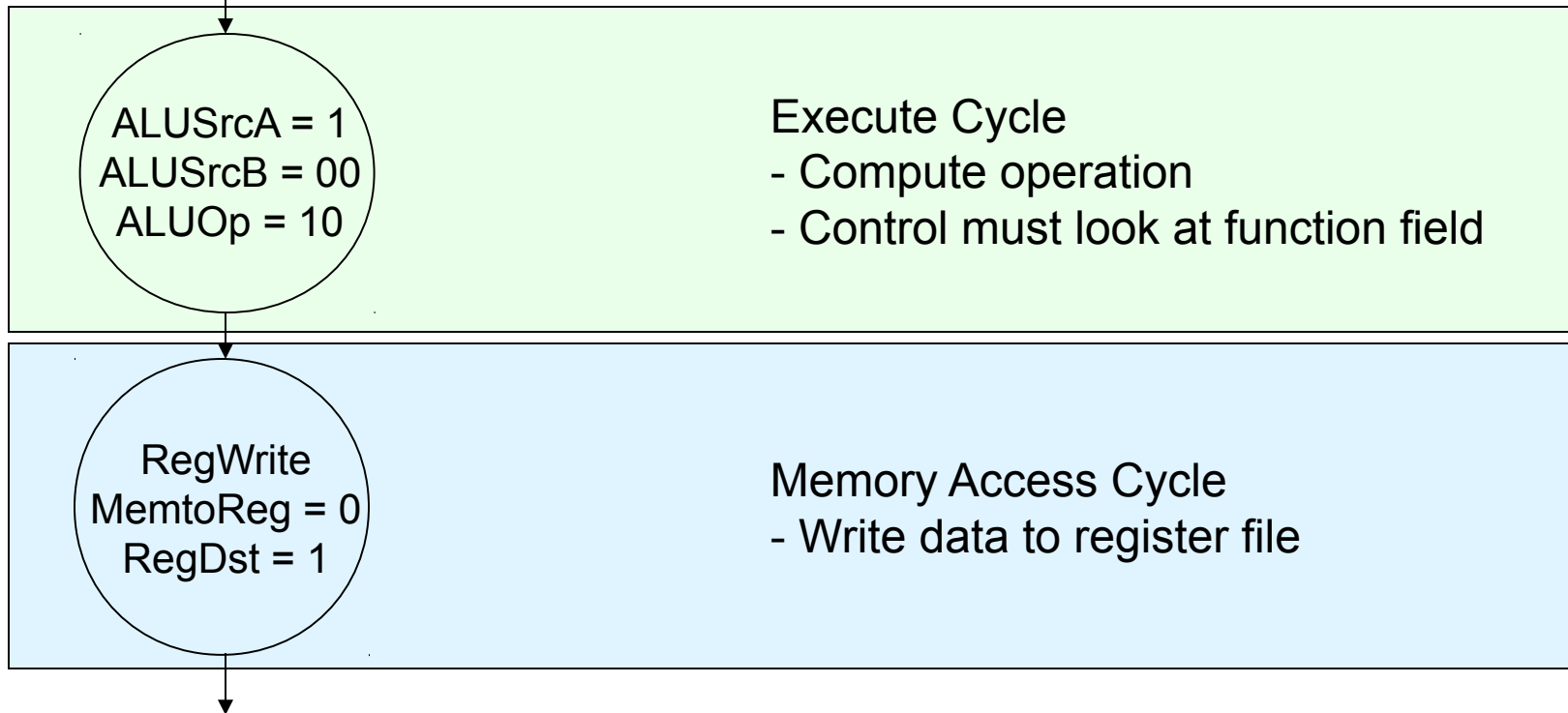
Memory Access Instructions

From Decode state if SW or LW



R-type Instructions

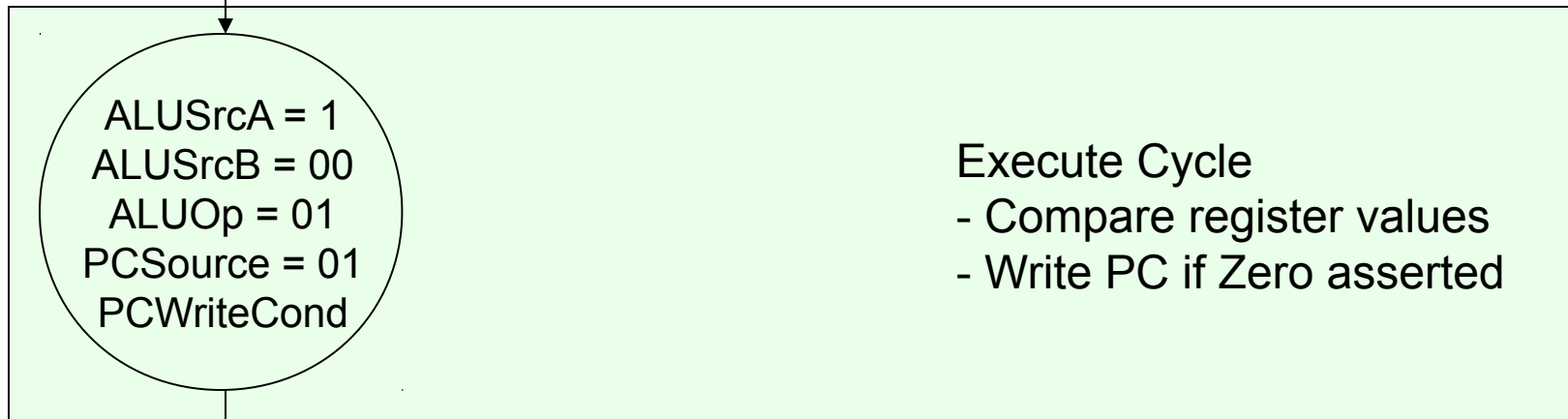
From Decode state if R-type



Fetch Next
Instruction

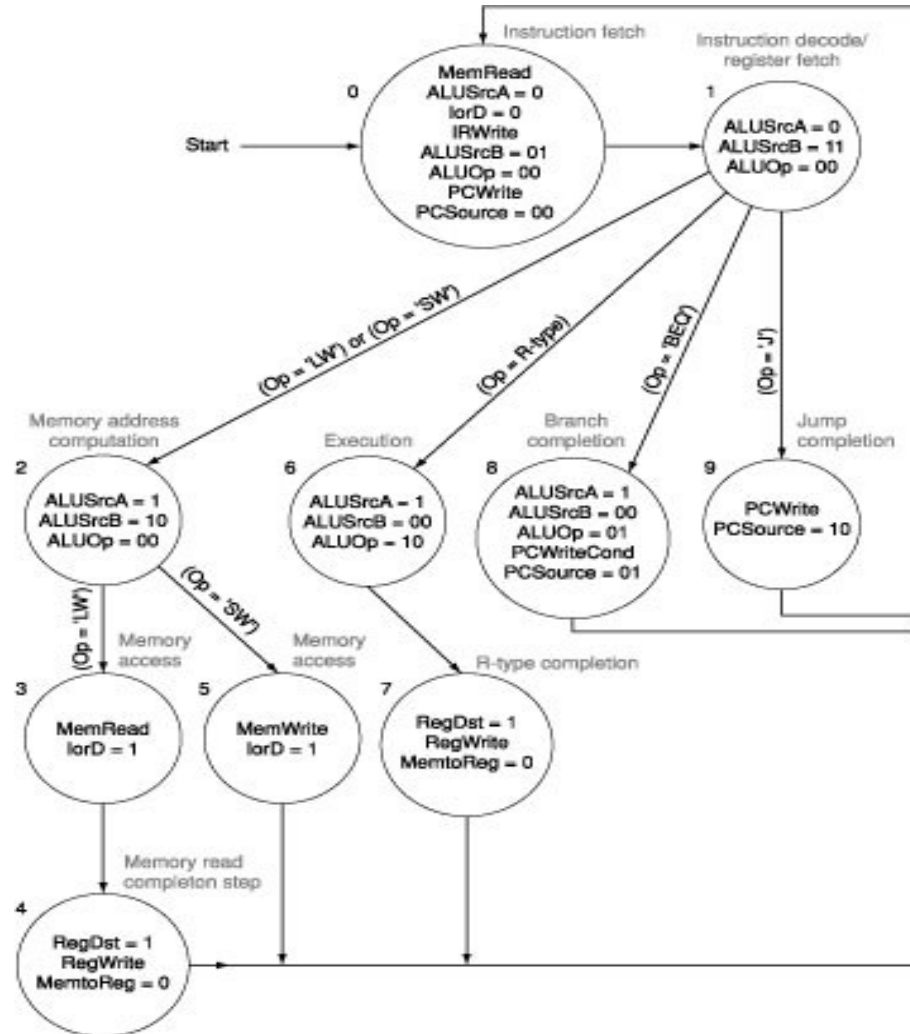
Branch on Equal Instruction

From Decode state if BEQ



Fetch Next
Instruction

Final State Diagram



Implementing jal

- Jump and link (jal)
 - J-Type
 - PC+4 → \$31
 - Jump destination → PC

- What modifications are required?
 - If doing it in 2 Cycles
 - Const 31 write destination
 - Expand PCSource mux with ALUOut as PC+4
 - If doing it in 3 Cycles
 - Const 31 write destination
 - PC connected to write data

Implementing jal

