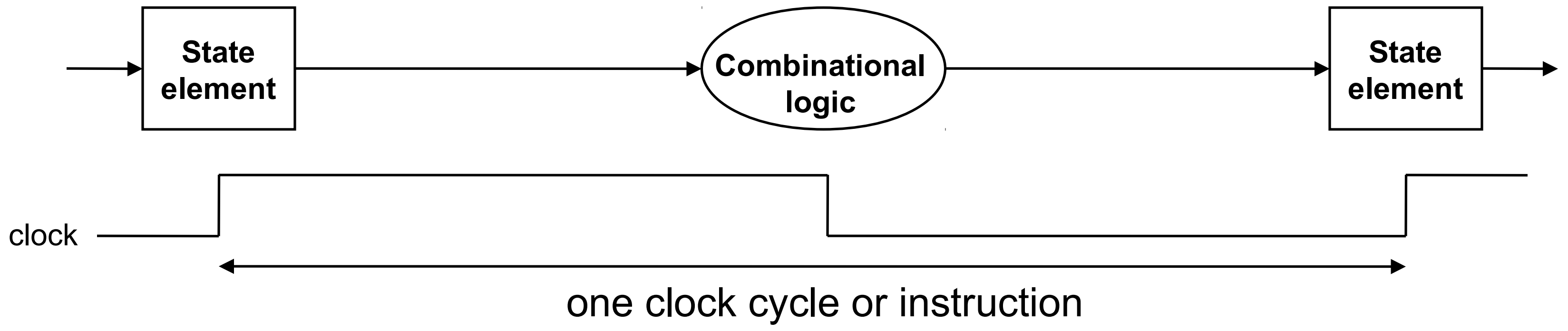
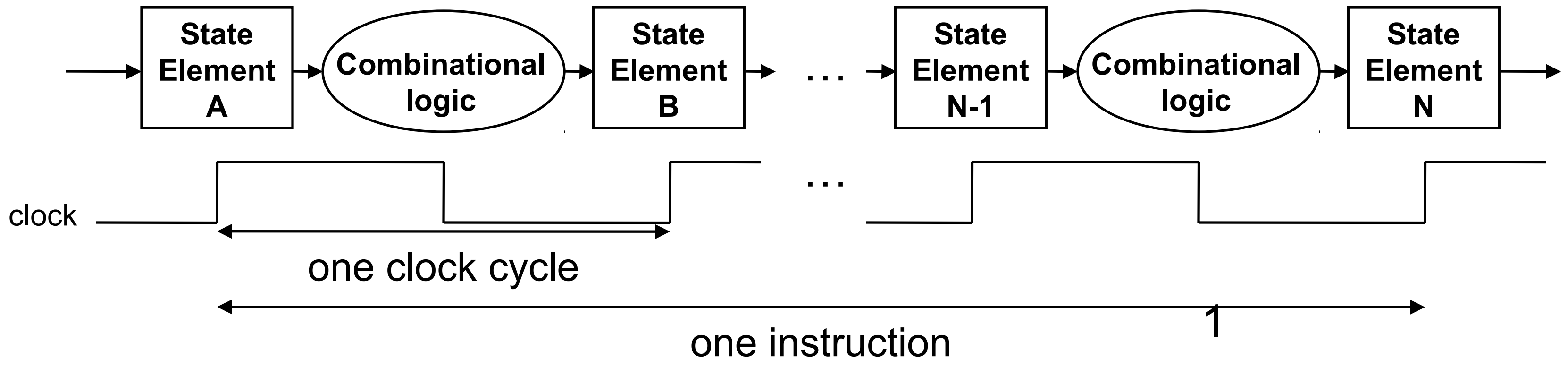
Multi-cycle Approach

- Single cycle CPU

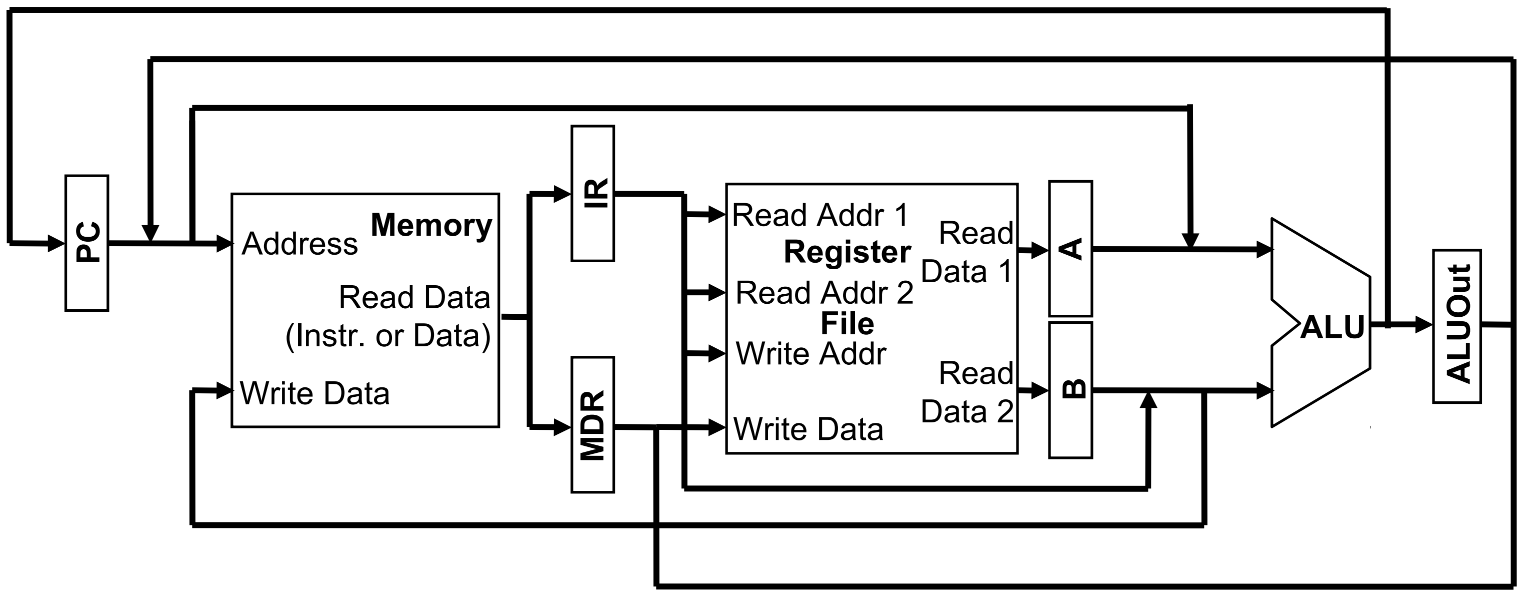


- Multi-cycle CPU
  - Requires state elements to hold intermediate values



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

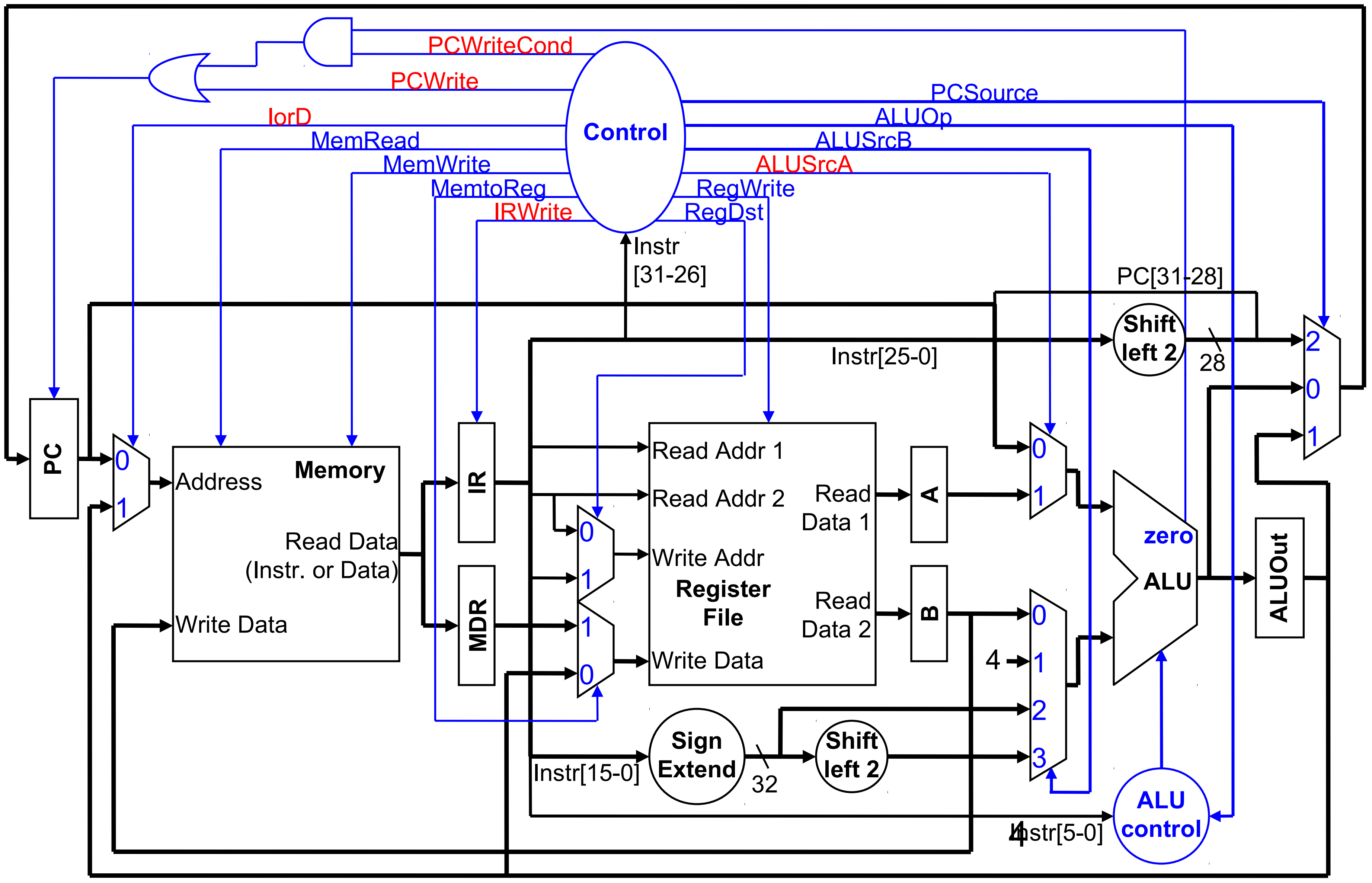
A, **B** – Register File data **ALUOut** – ALU Result Register

  - 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
  - IorD 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



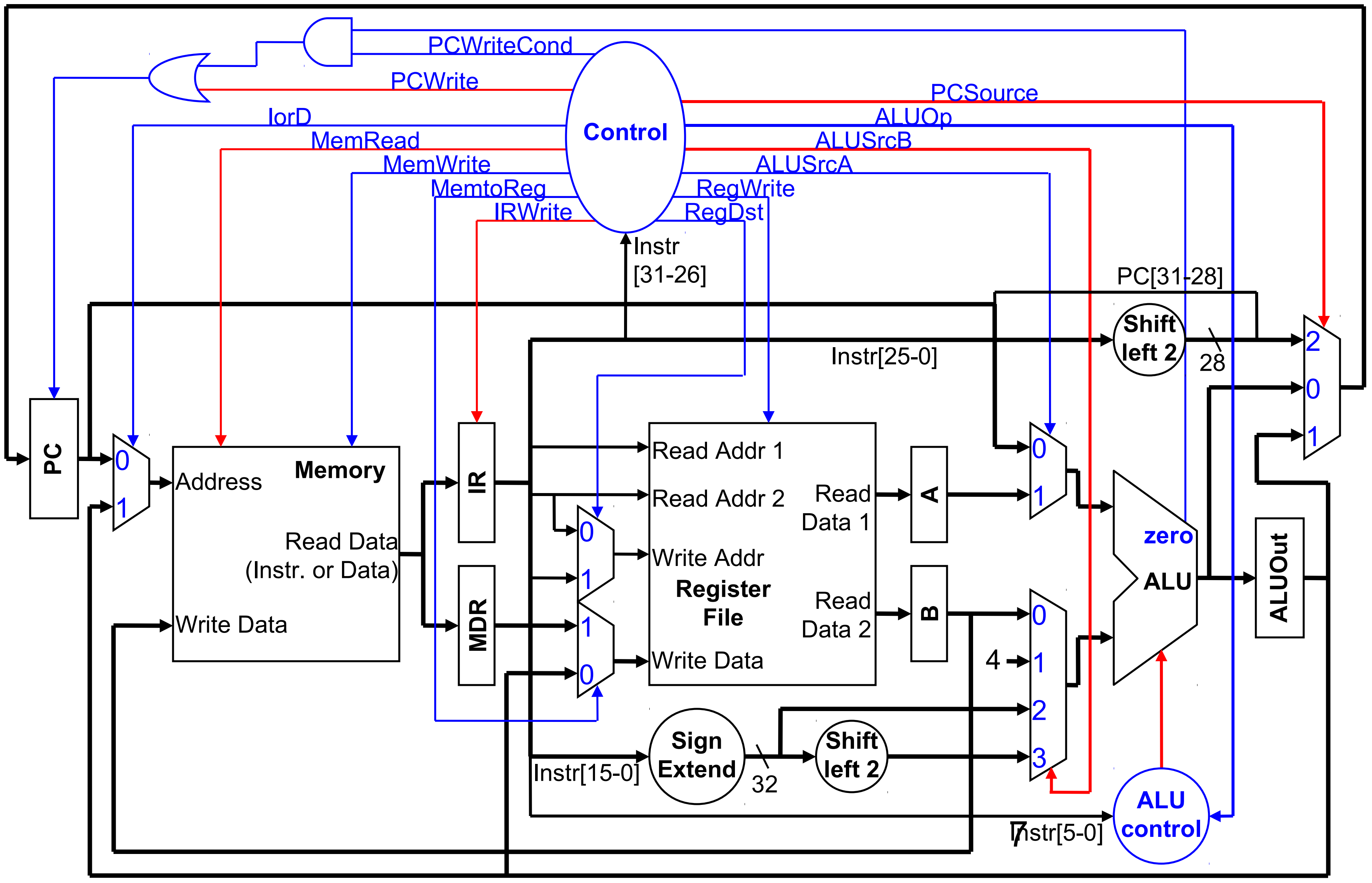
Multi-cycle Steps

- 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)

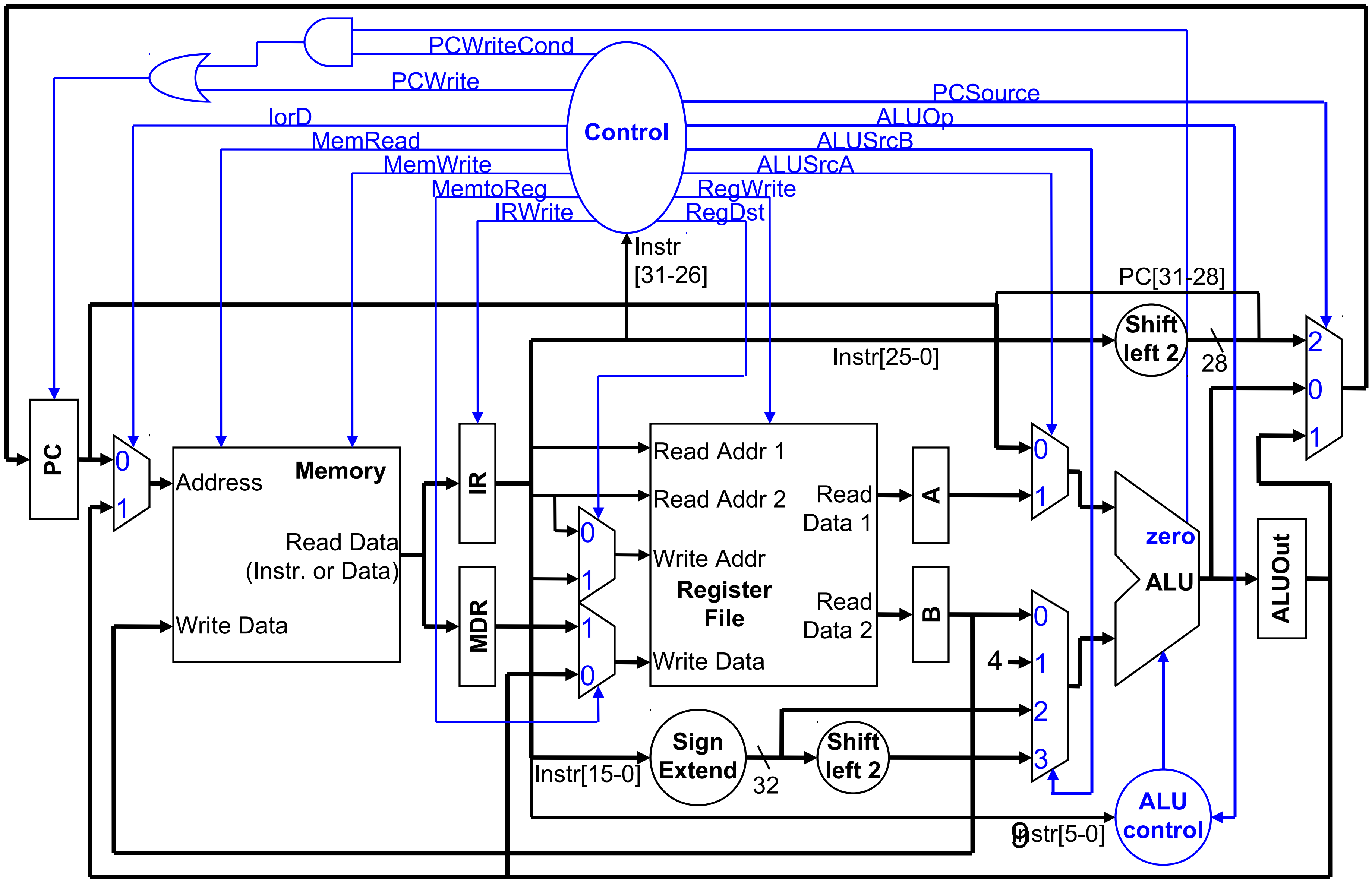
Instruction Fetch



(2) Decode and Register Fetch Cycle

- Read register values
  - A = R[rs], B = R[rt]
- Compute branch destination
  - ALUOut = PC + 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)

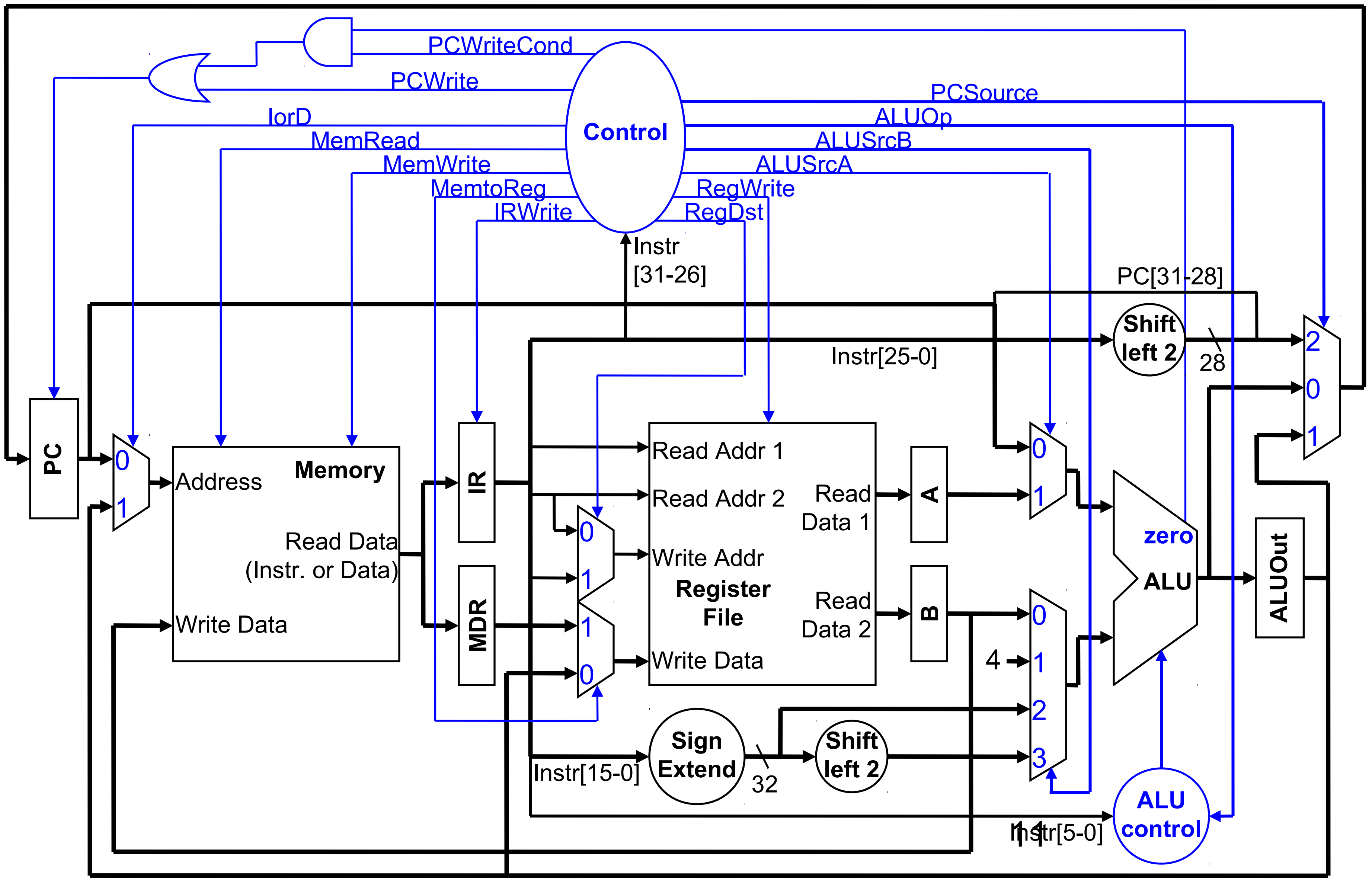
Decode and Register Fetch



(3) Execution Cycle

- Functionality varies with instructions
  - Memory reference
    - Compute address
    - ALUOut = A + sign extended immediate
  - R-type
    - Compute operation
    - ALUOut = A 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

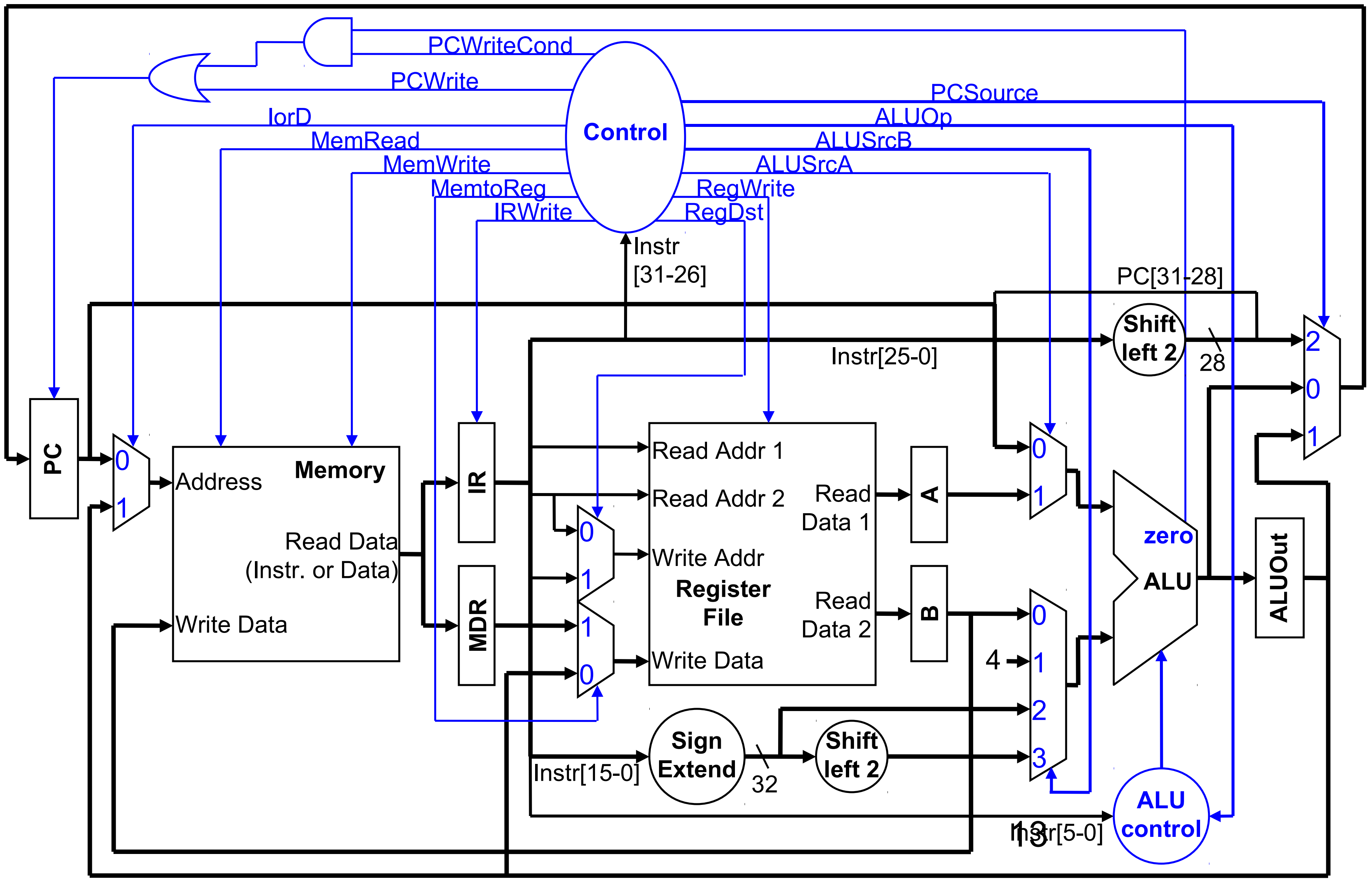
Execute Branch



(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

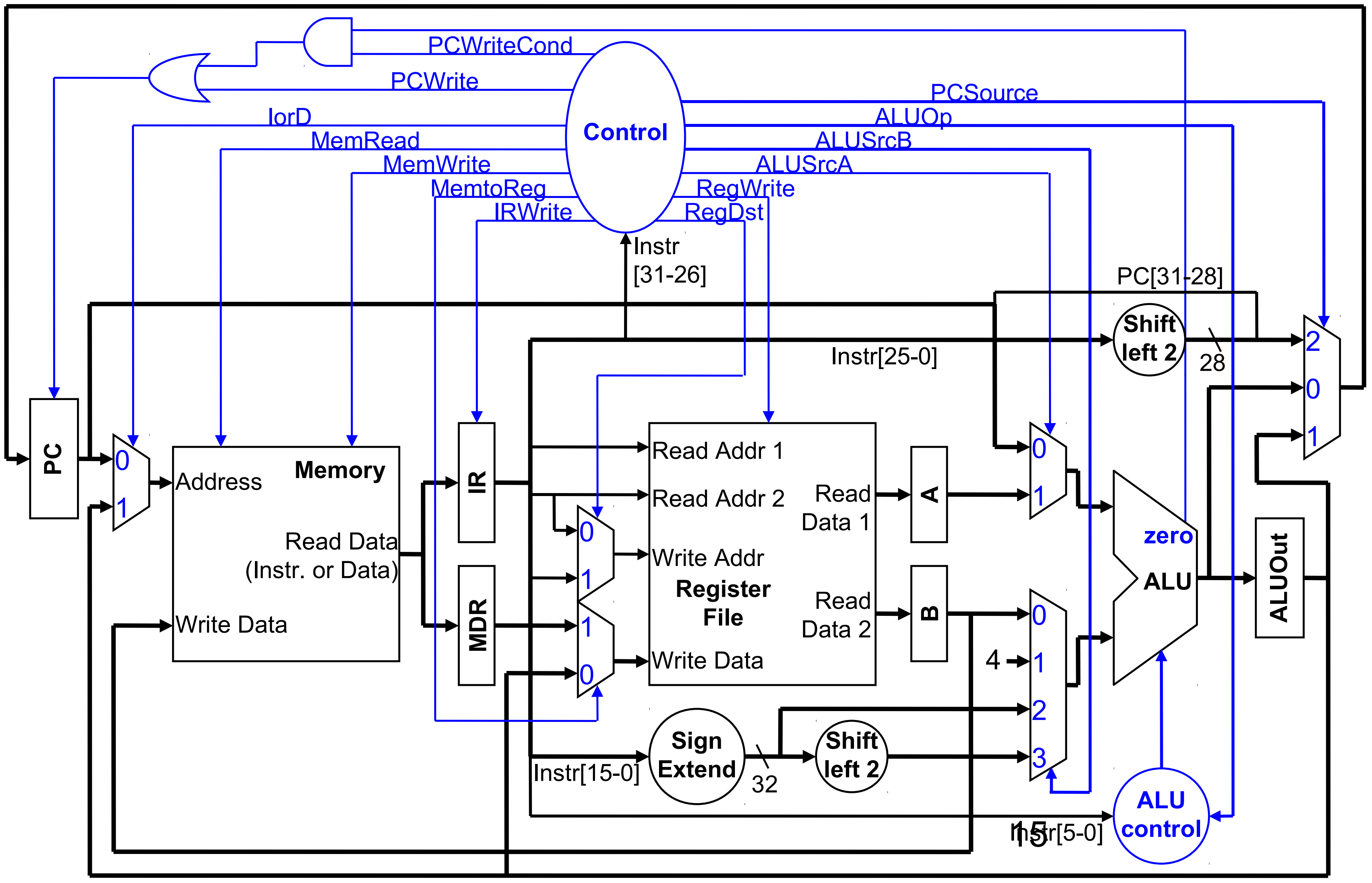
R-Type “Memory Access”



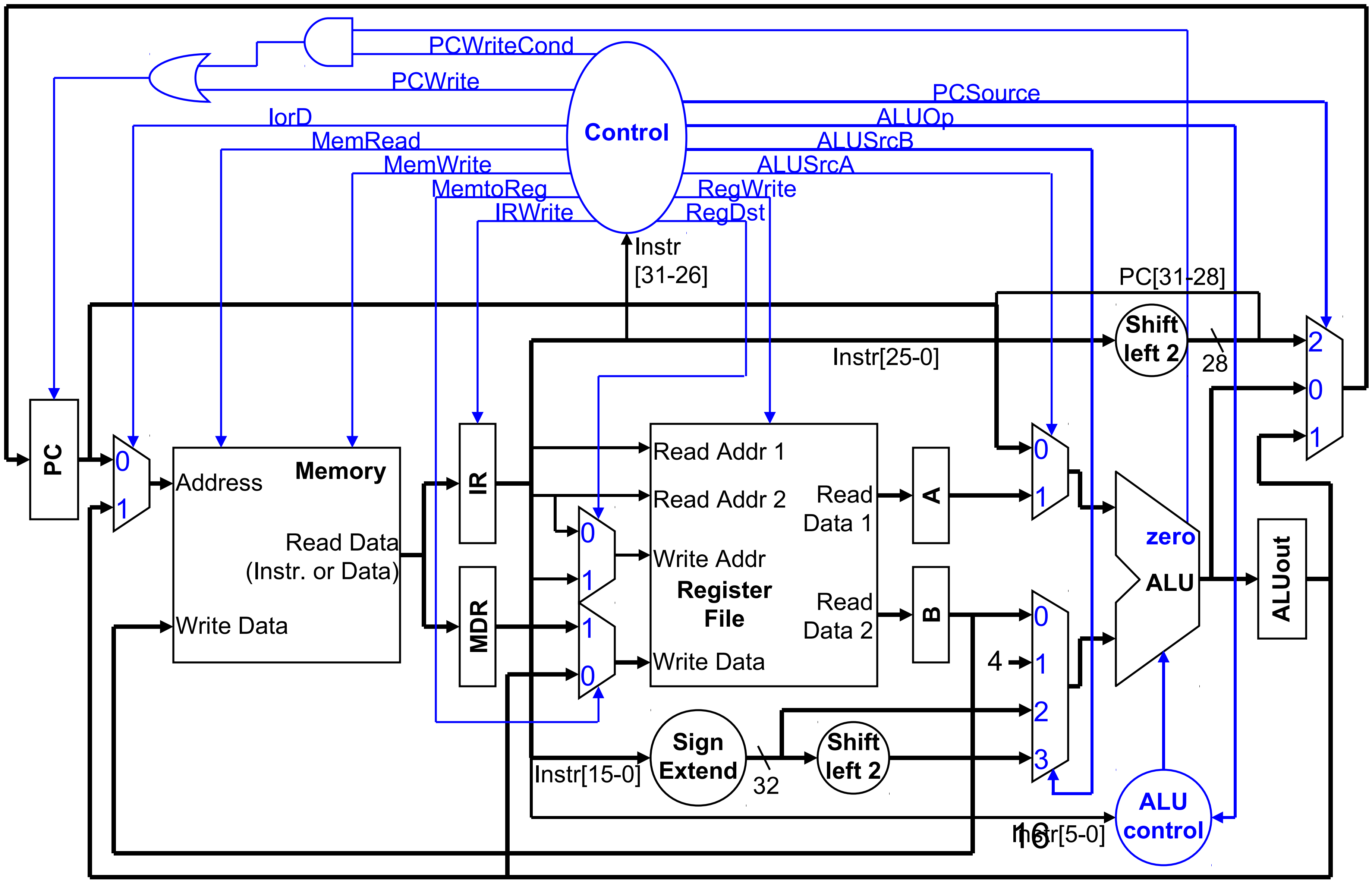
(5) Write Register File Cycle

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

lw Write Registers



Review

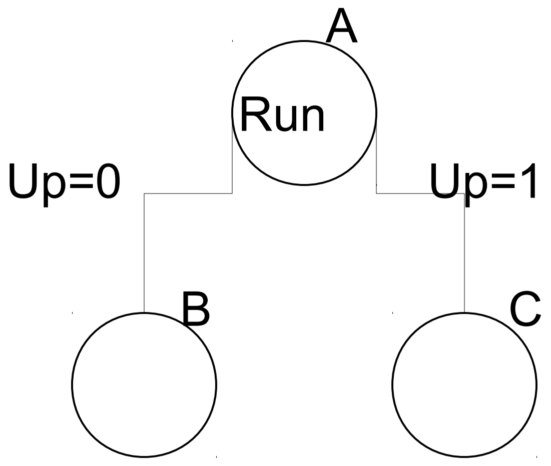


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Control
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Instr
[31-26]
PC[31-28]
Shift left 2
28
Instr[25-0]
PC
0
1
Address
Memory
Read Data
(Instr. or Data)
Write Data
IR
MDR
Read Addr 1
Read Addr 2
Write Addr
Register File
Write Data
Read Data 1
Read Data 2
A
B
4
zero
ALU
ALUout
2
3
Sign Extend
32
Shift left 2
Instr[15-0]
Instr[5-0]
ALU control

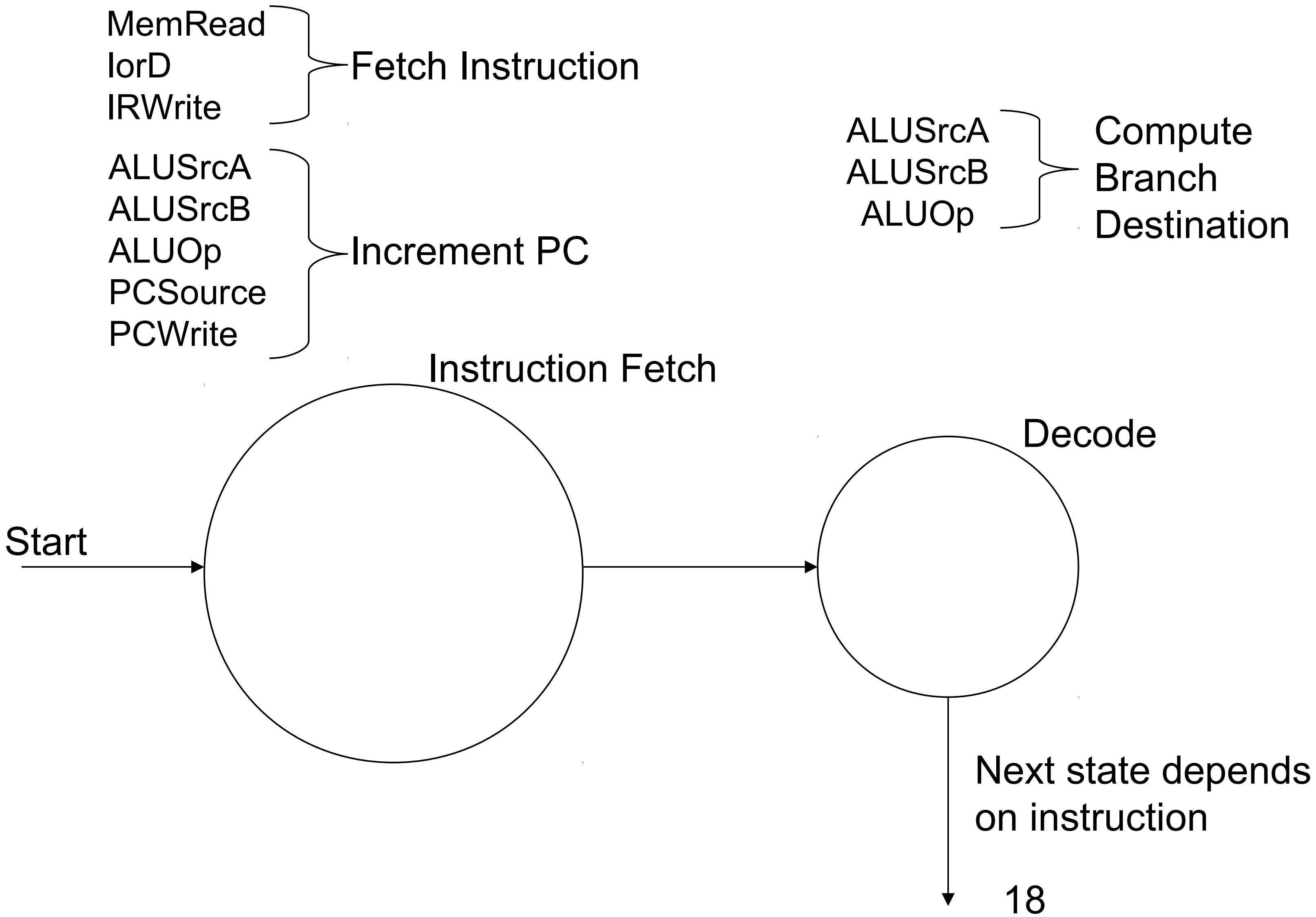
Defining the CPU Control

- 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

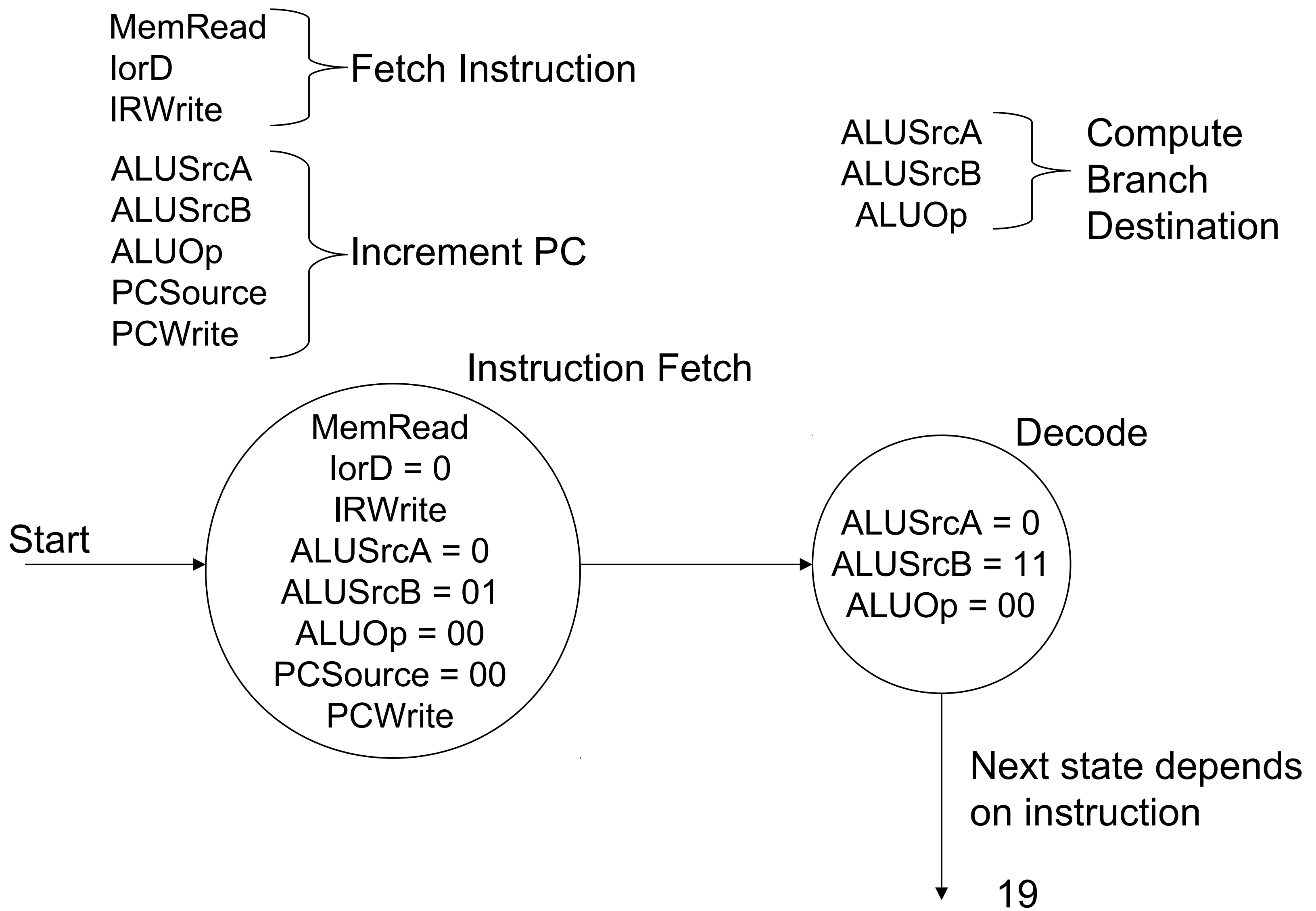
- Example State A
  - Signal Run is asserted
  - Goes to State B when input Up = 0
  - Goes to State C when input Up = 1



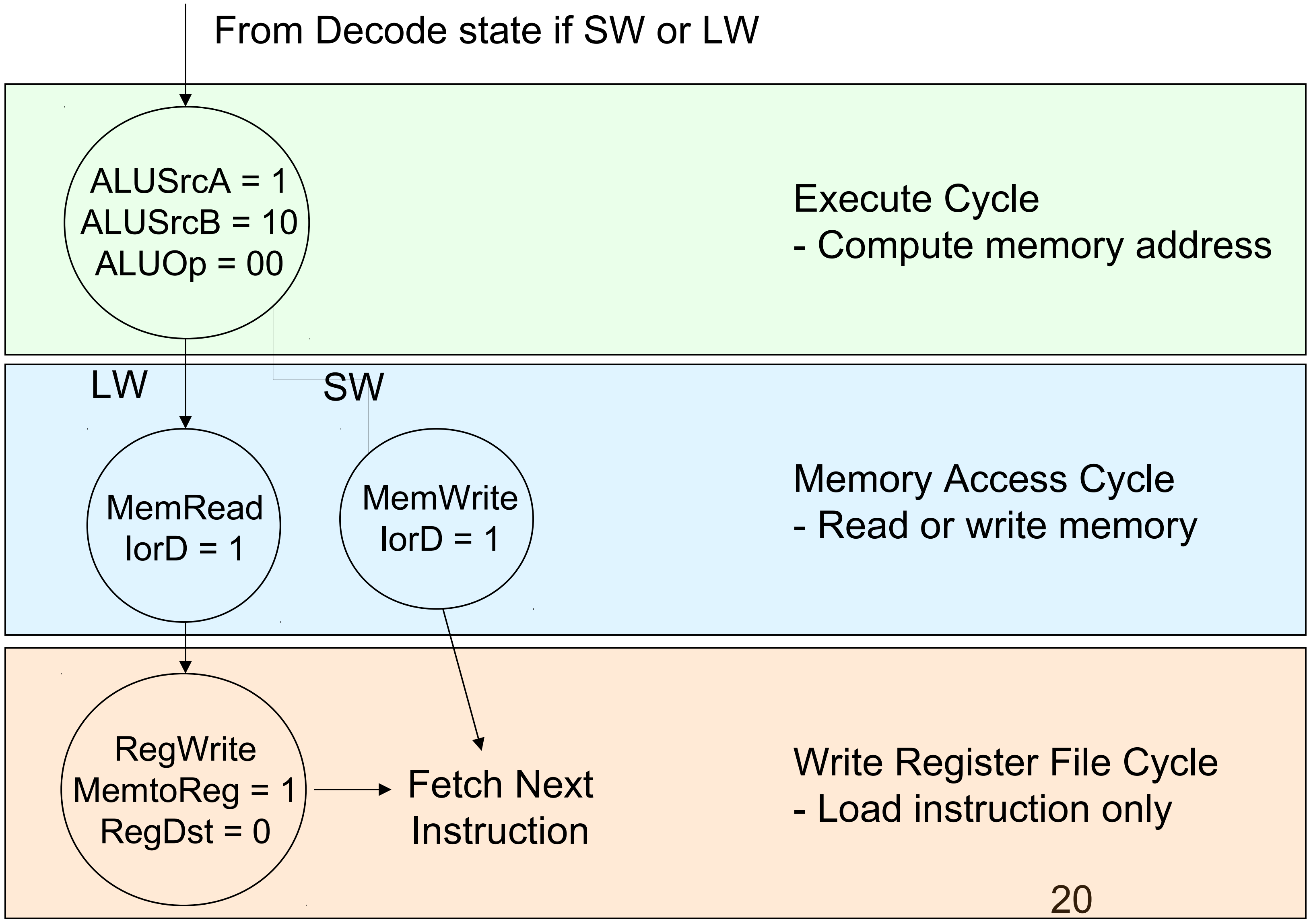
Instruction Fetch and Decode Cycles



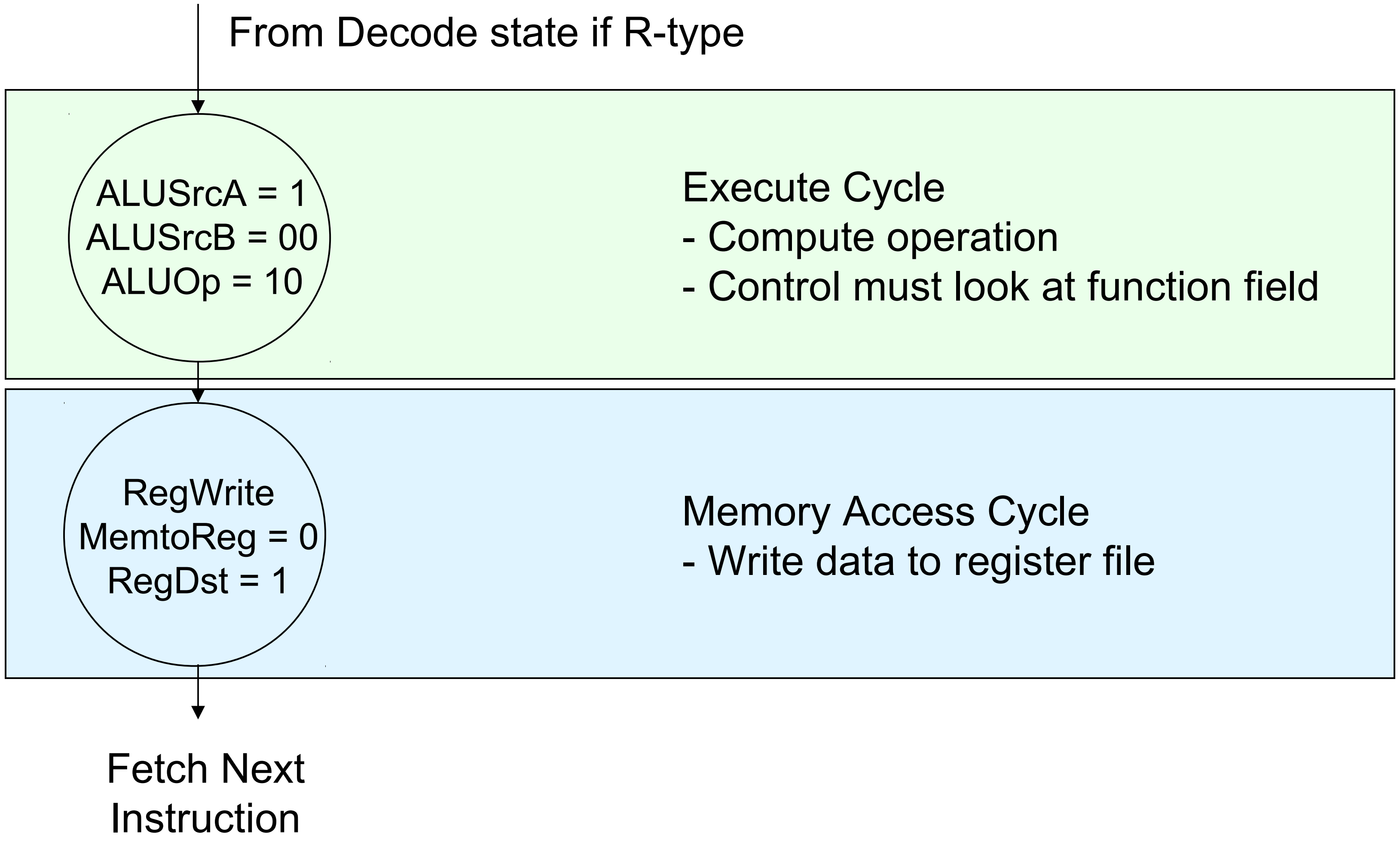
Instruction Fetch and Decode Cycles



Memory Access Instructions



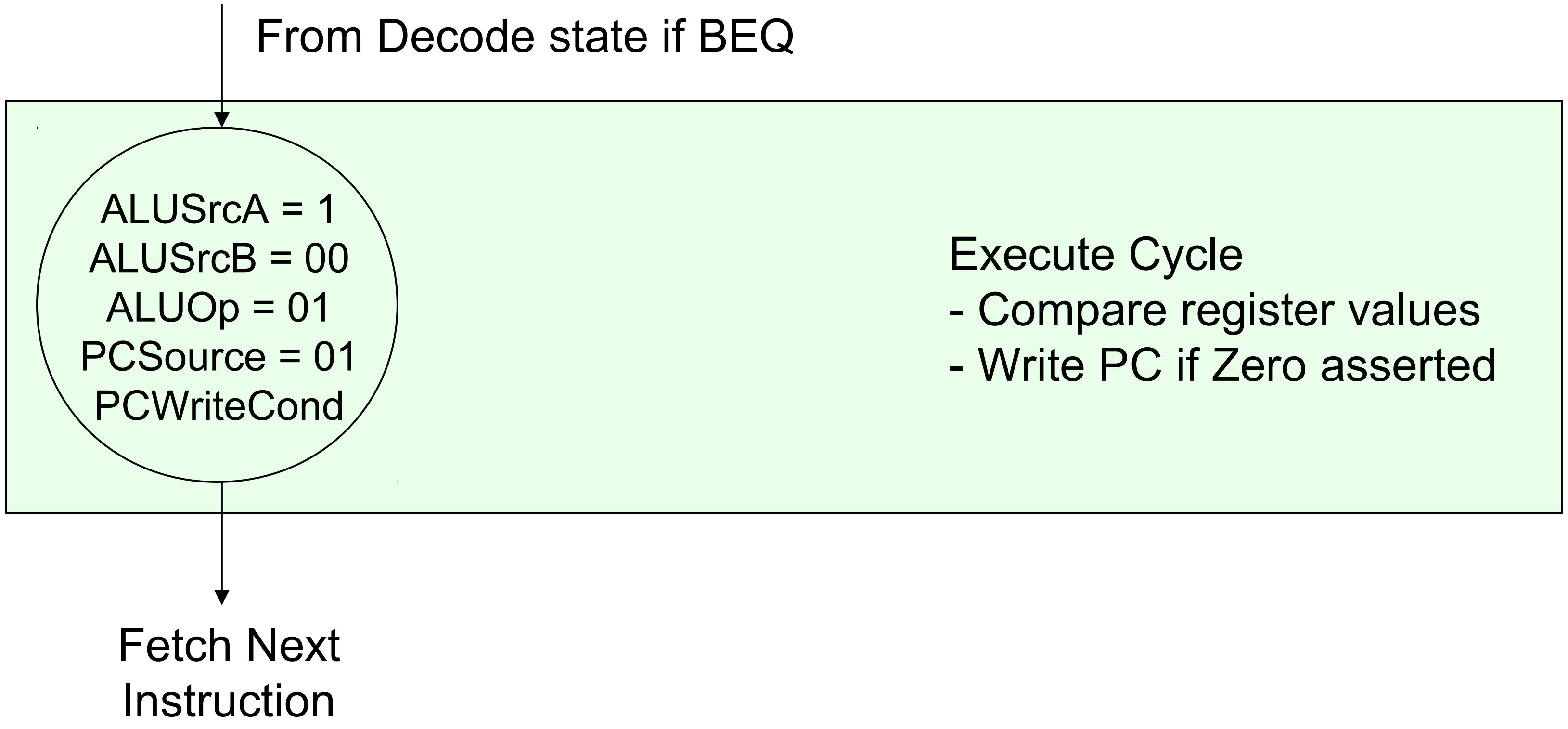
R-type Instructions



Execute Cycle
- Compute operation
- Control must look at function field

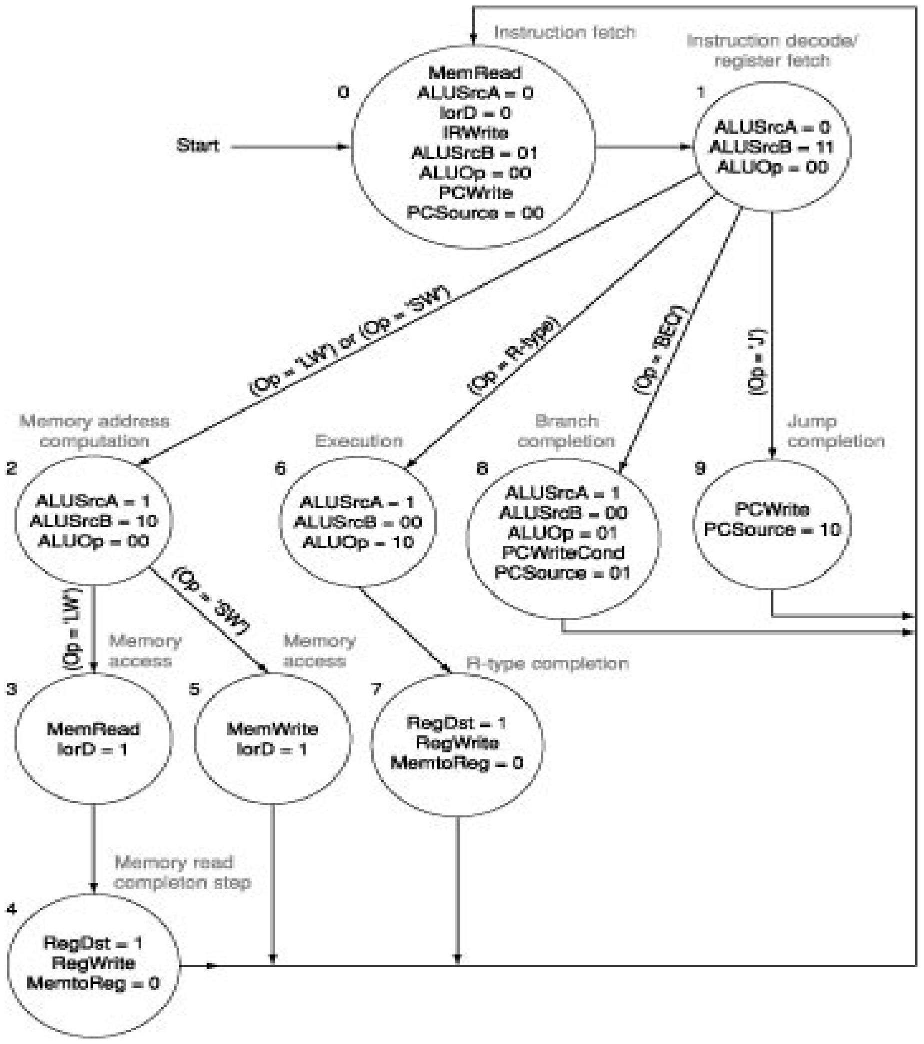
Memory Access Cycle
- Write data to register file

Branch on Equal Instruction



Execute Cycle
- Compare register values
- Write PC if Zero asserted

Final State Diagram

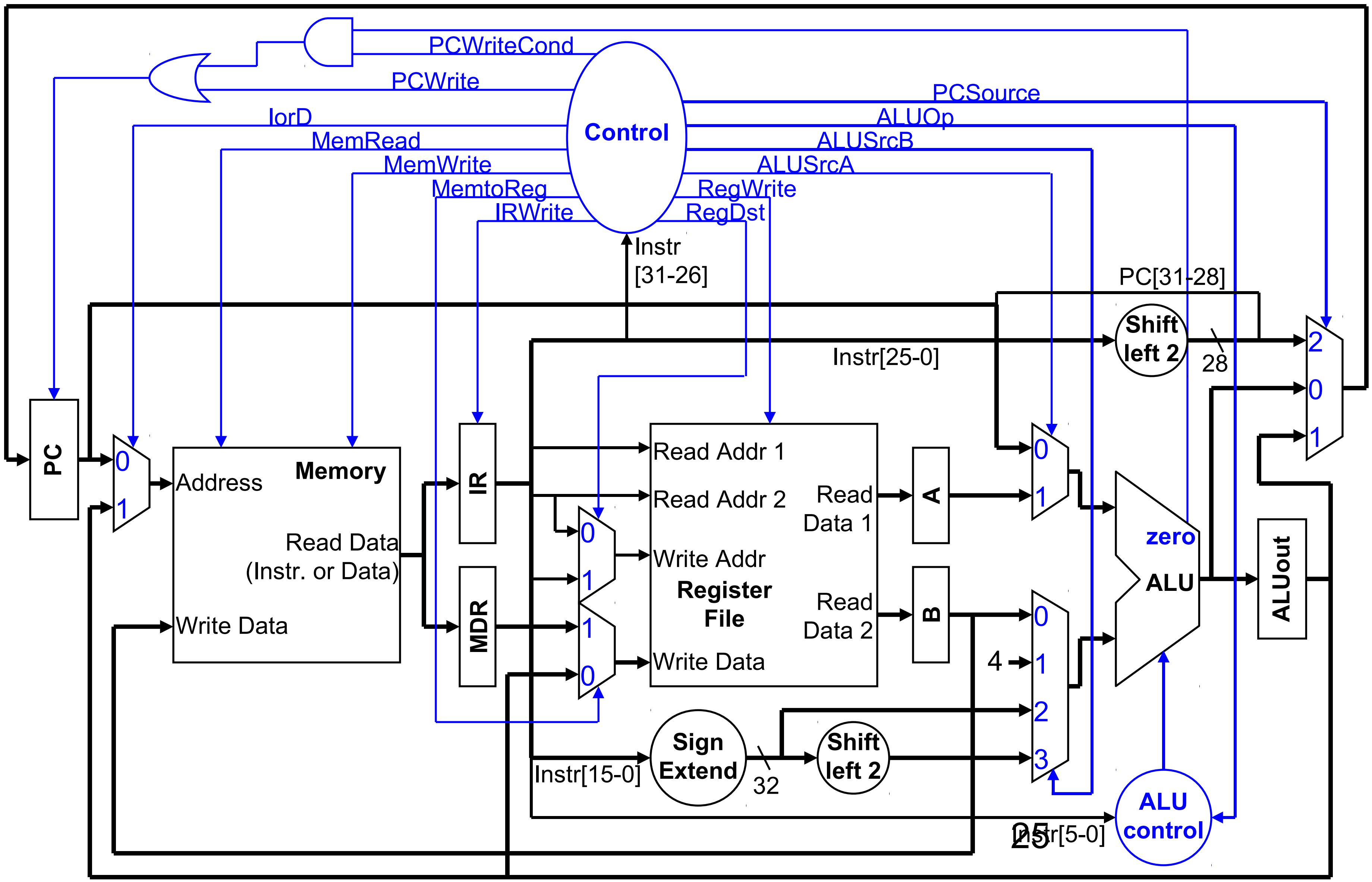


Implementing jal

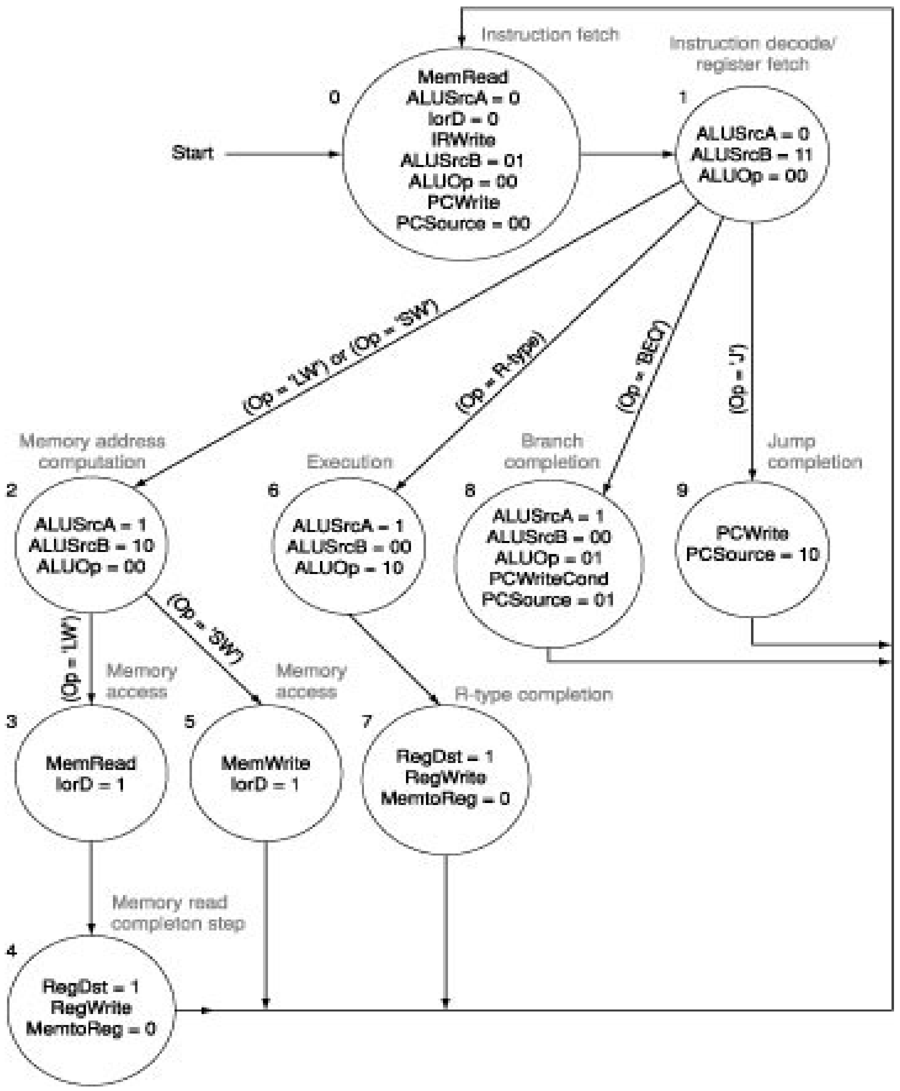
- Jump and link (jal)
  - J-Type
  - PC+4 → $31
  - Jump destination → PC

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

Implementing jal



Implementing jal (needs fixing MF)


Instruction fetch
Instruction decode/ register fetch
Start
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address computation
Execution
Branch completion
Jump completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory access
Memory access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Memory read completion step
4
RegDst = 1
RegWrite
MemtoReg = 0