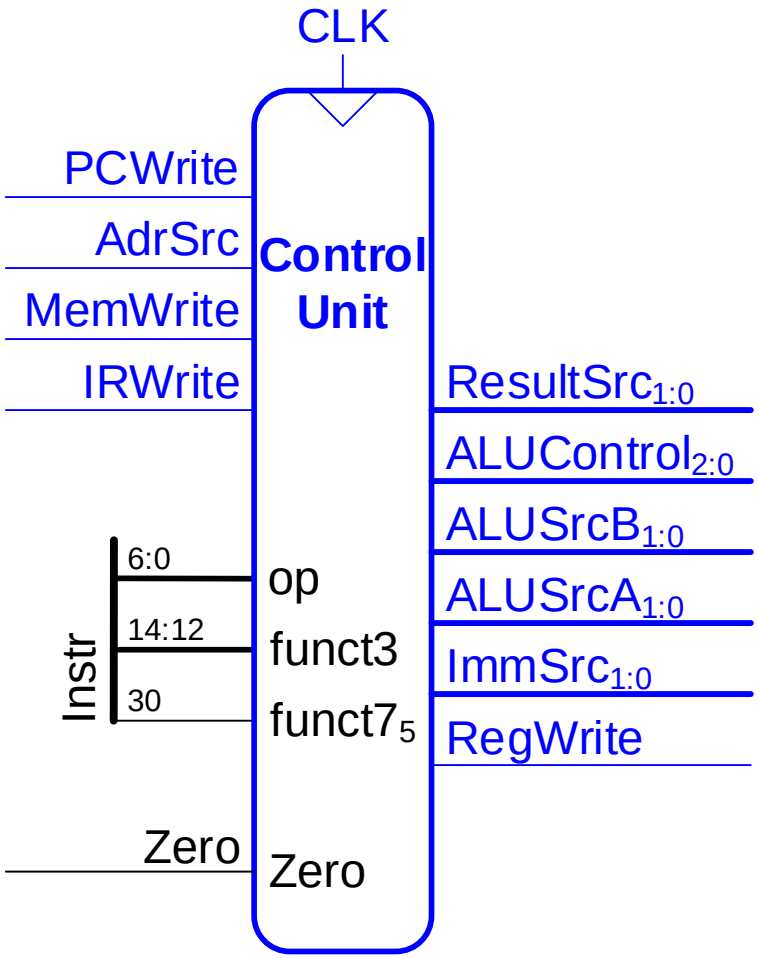
Chapter 7: Microarchitecture

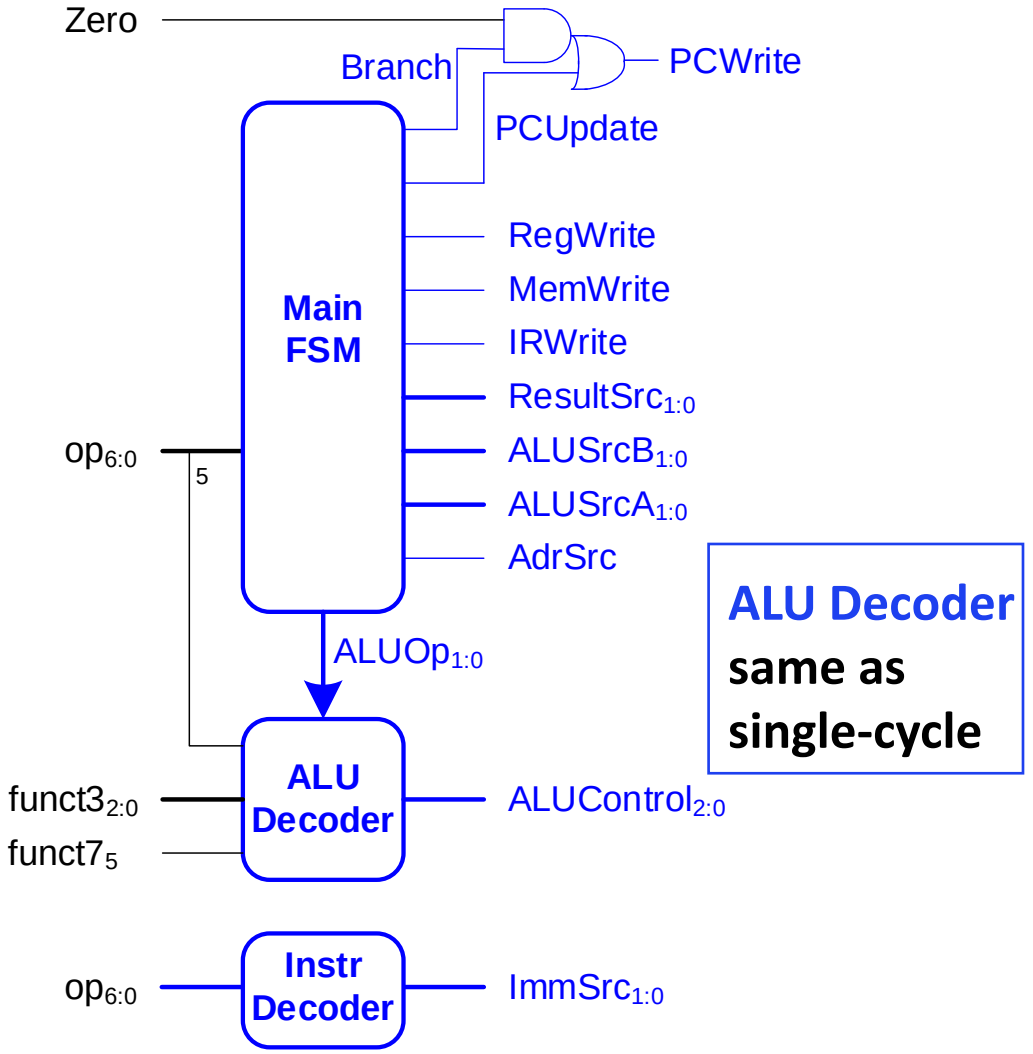
Multicycle Control

Multicycle Control

High-Level View

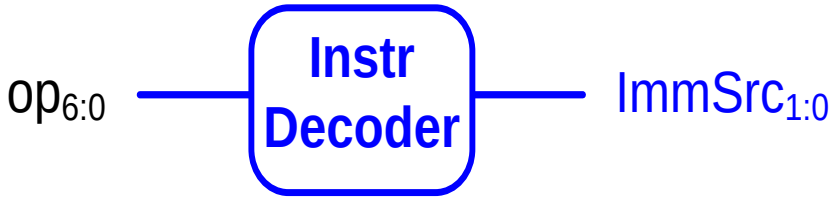


Low-Level View



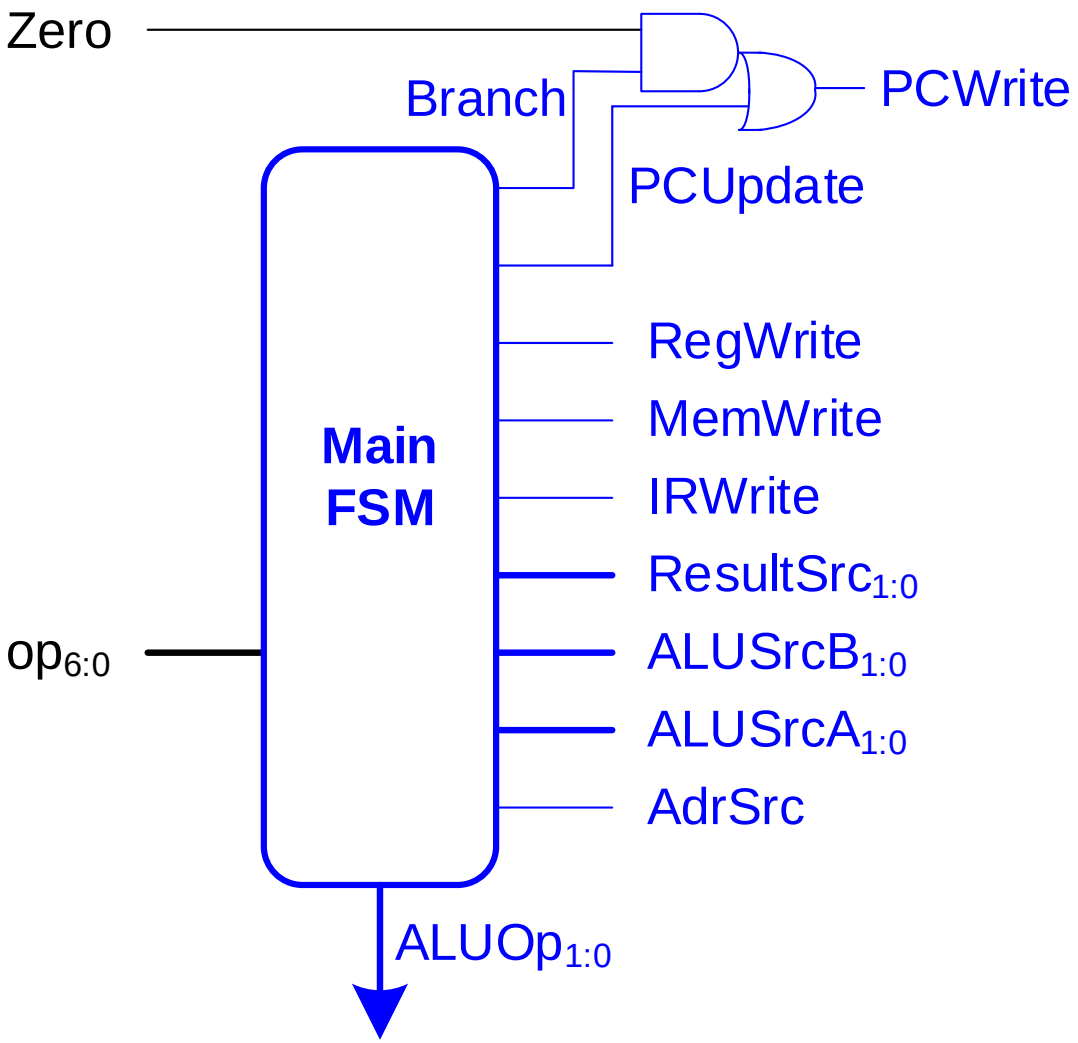
ALU Decoder same as single-cycle

Multicycle Control: Instruction Decoder



op	Instruction	ImmSrc
3	**`lw`**	00
35	**`sw`**	01
51	R-type	XX
99	**`beq`**	10

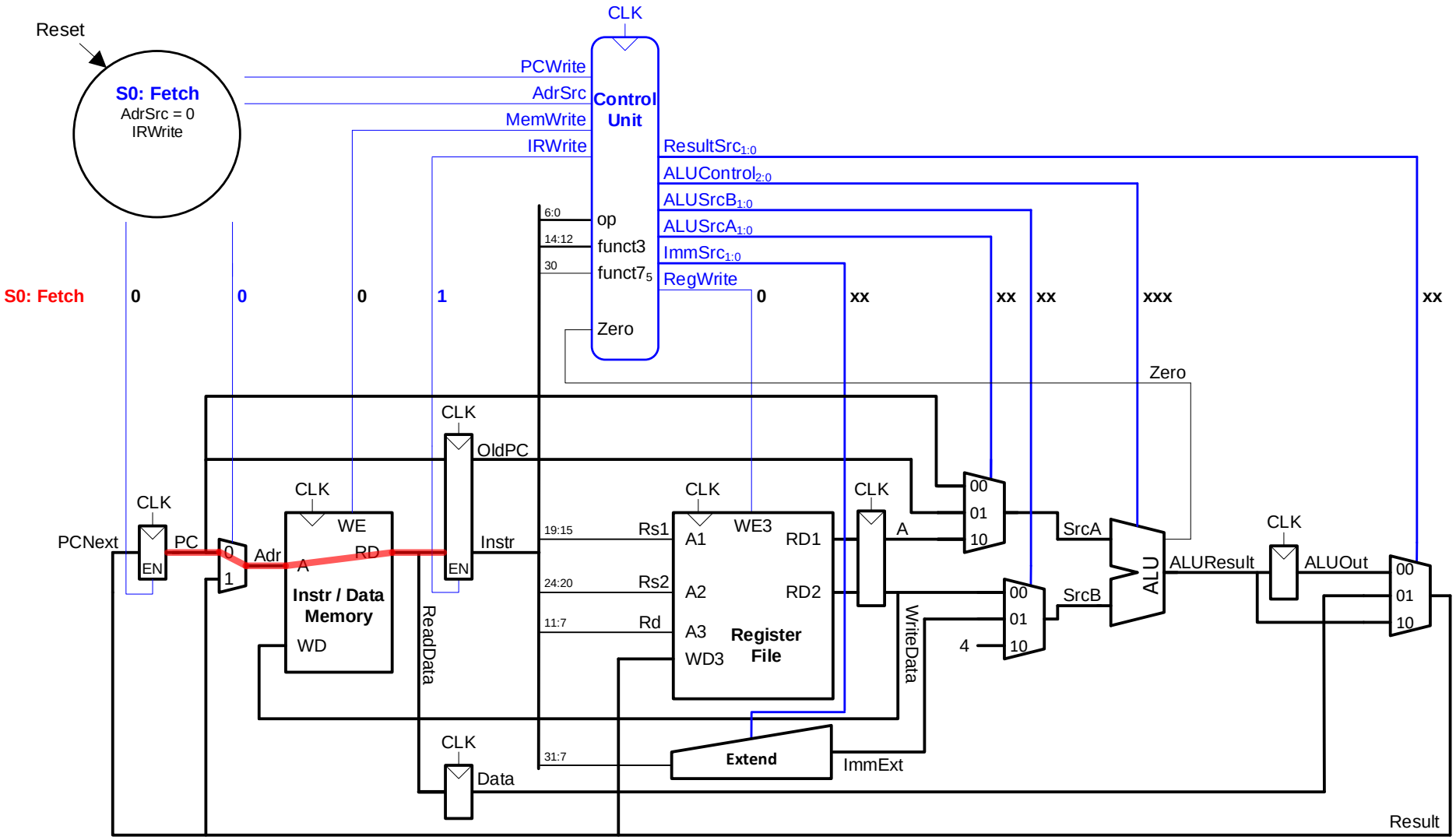
Multicycle Control: Main FSM



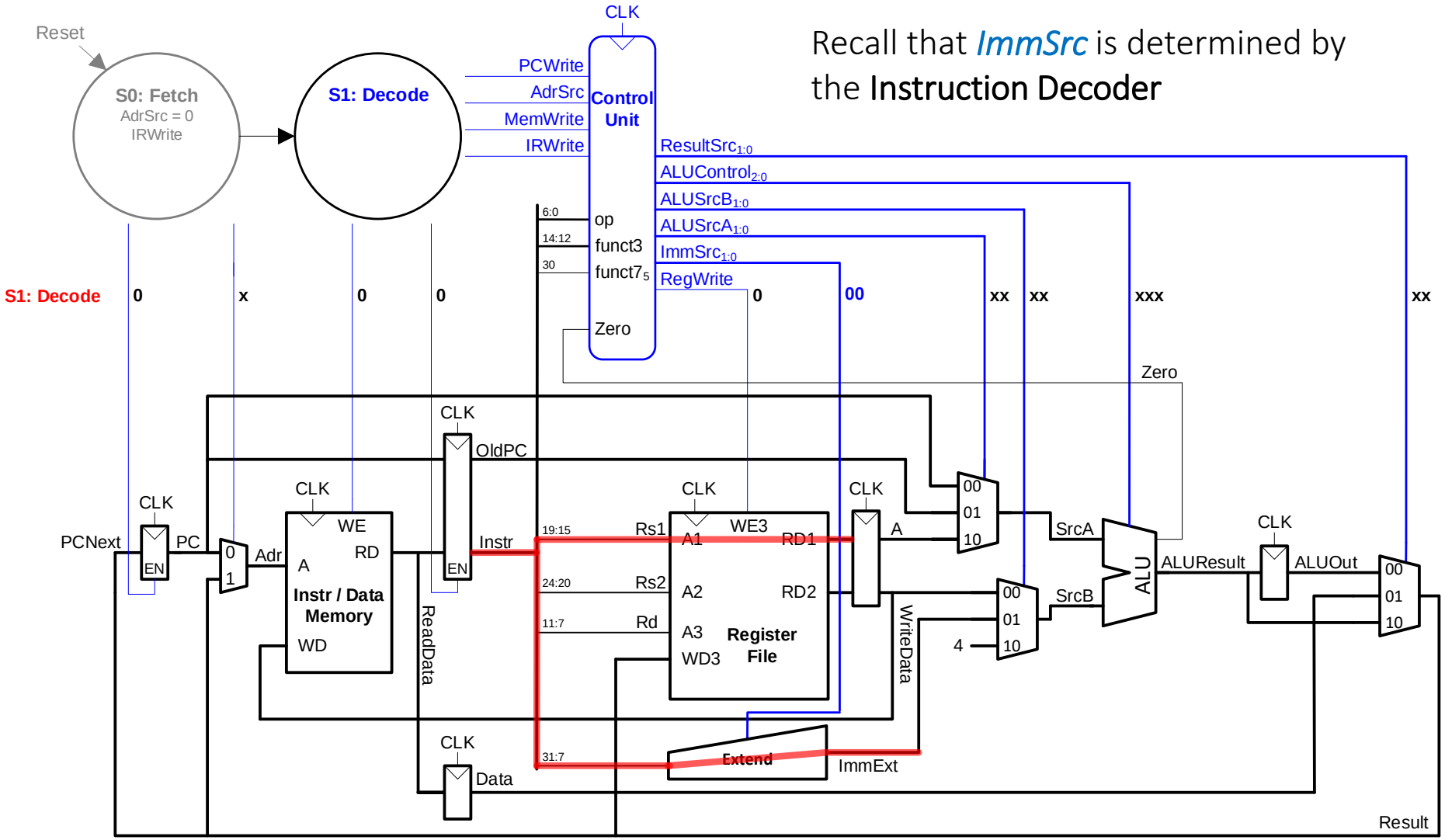
To declutter FSM:

- Write enable signals (RegWrite, MemWrite, IRWrite, PCUpdate, and Branch) are 0 if not listed in a state.
- Other signals are don't care if not listed in a state

Main FSM: Fetch

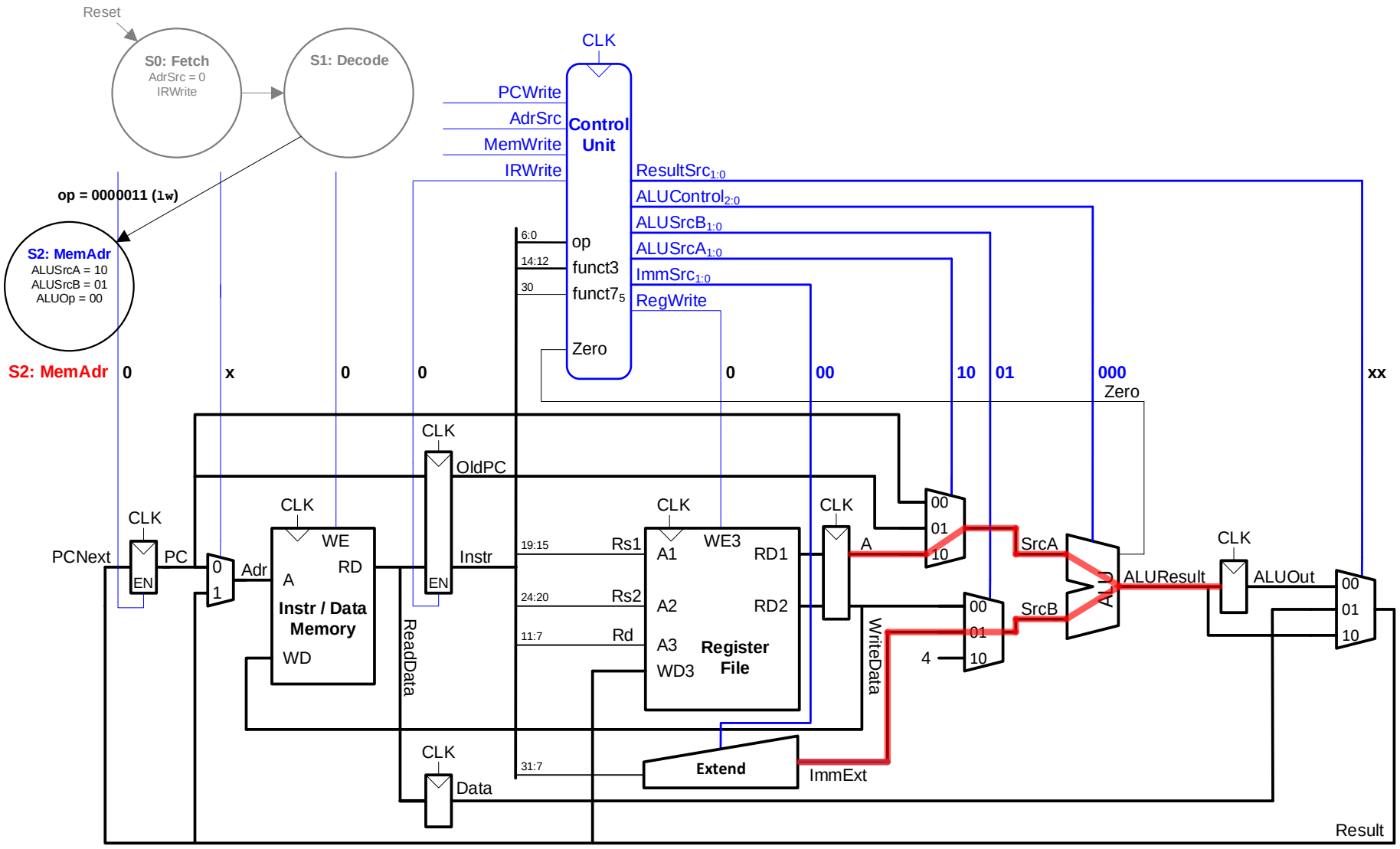


Main FSM: Decode

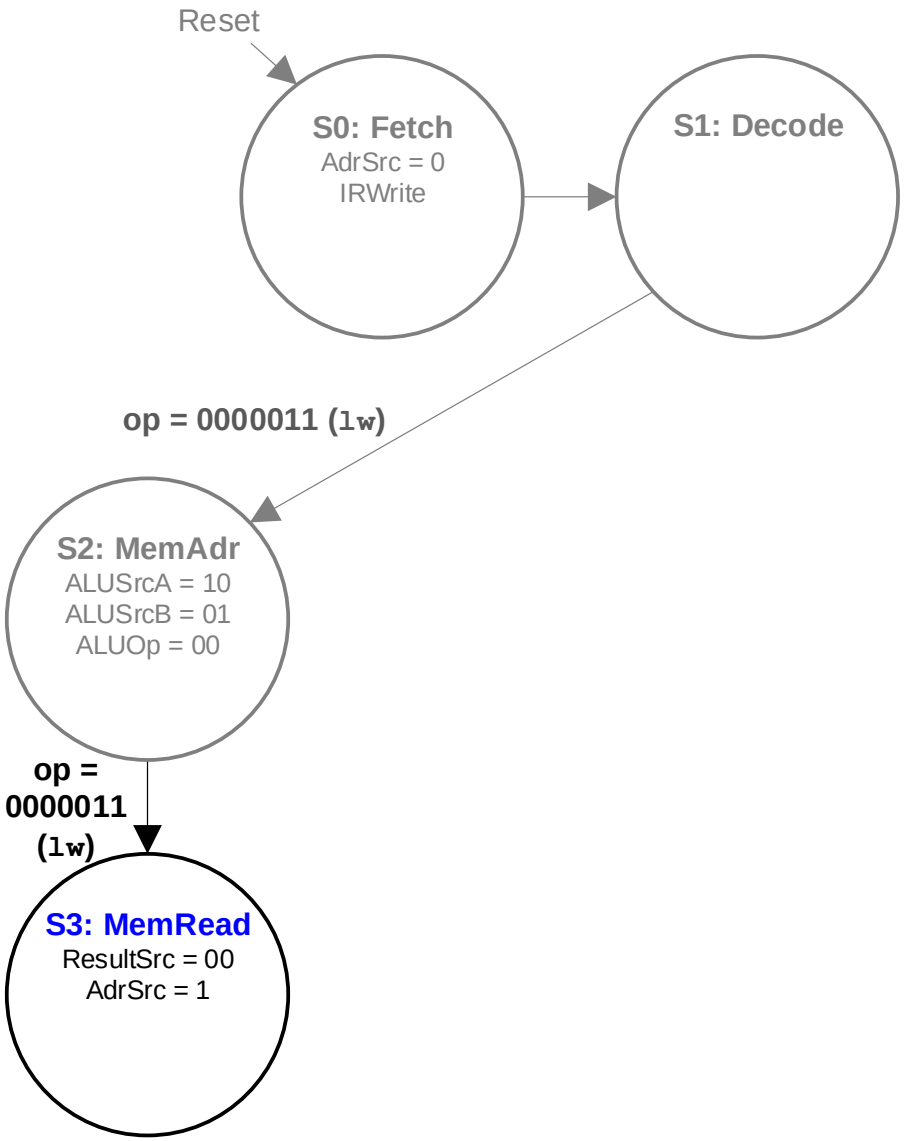


Recall that *ImmSrc* is determined by the **Instruction Decoder**

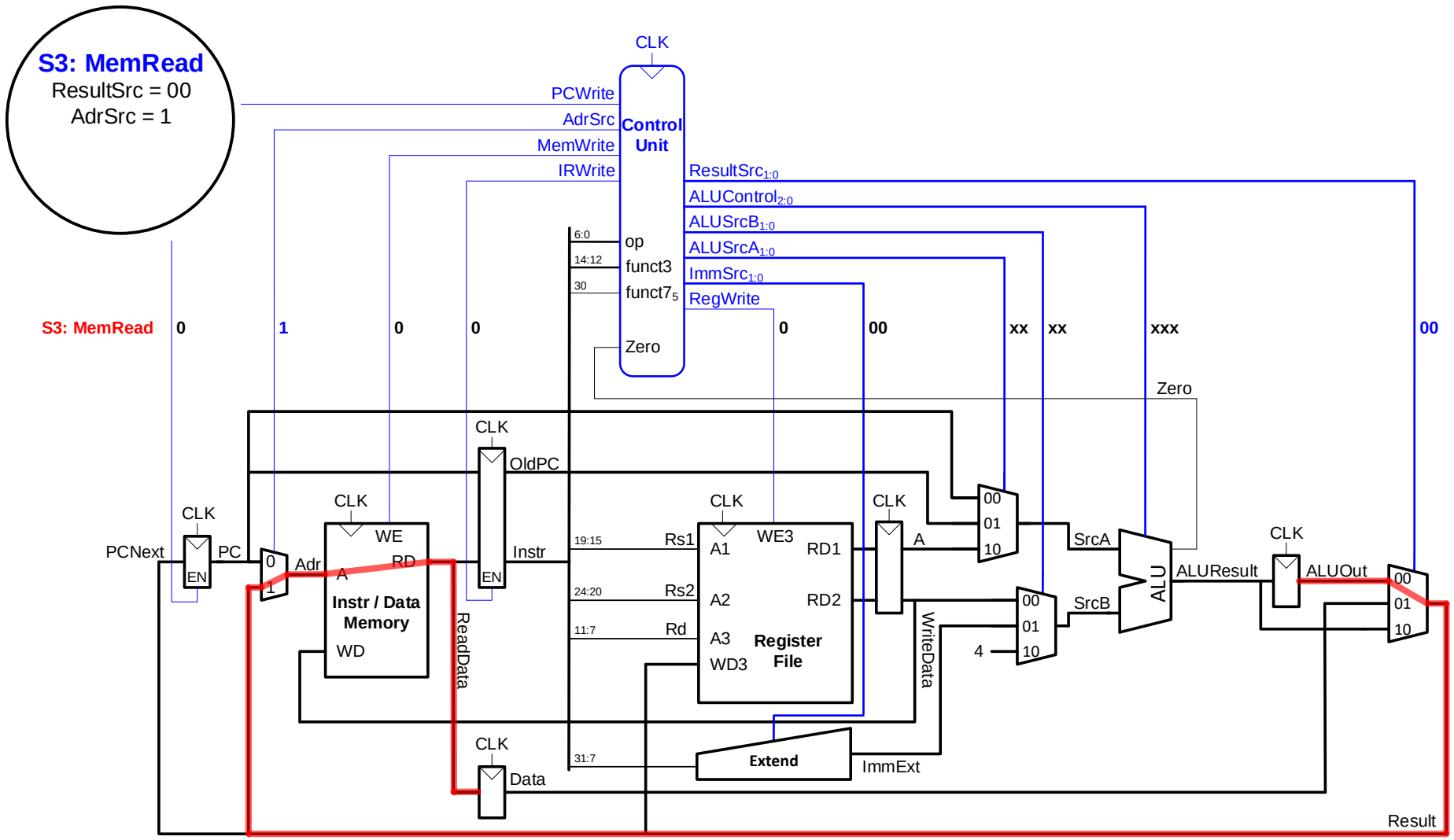
Main FSM: Address



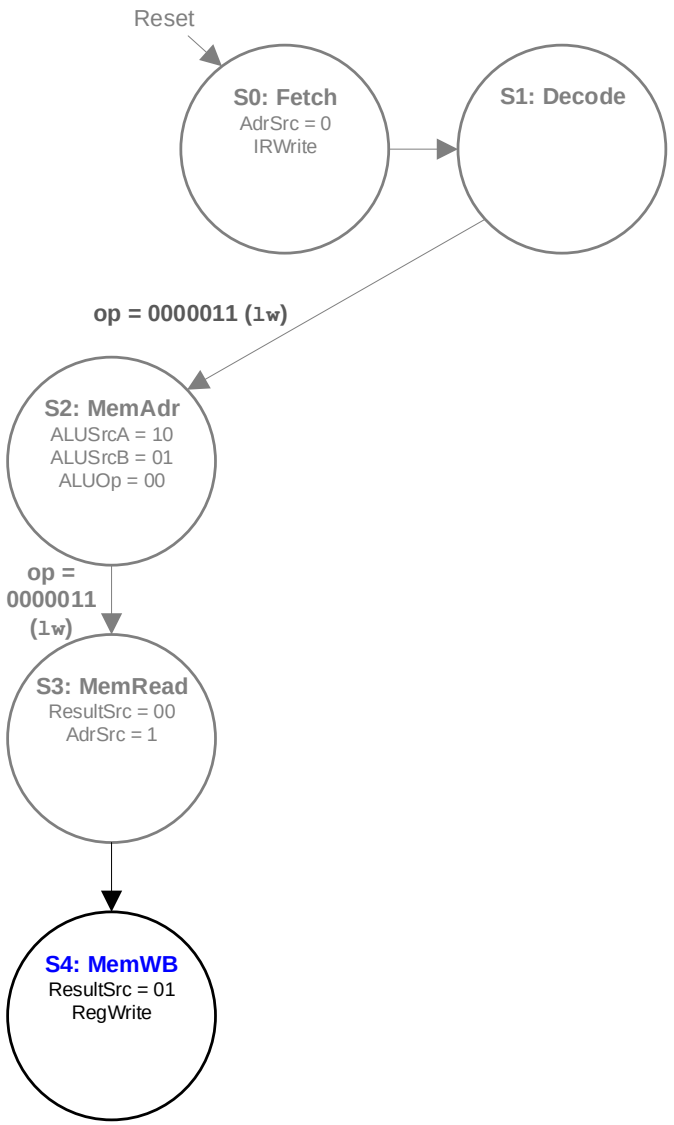
Main FSM: Read Memory



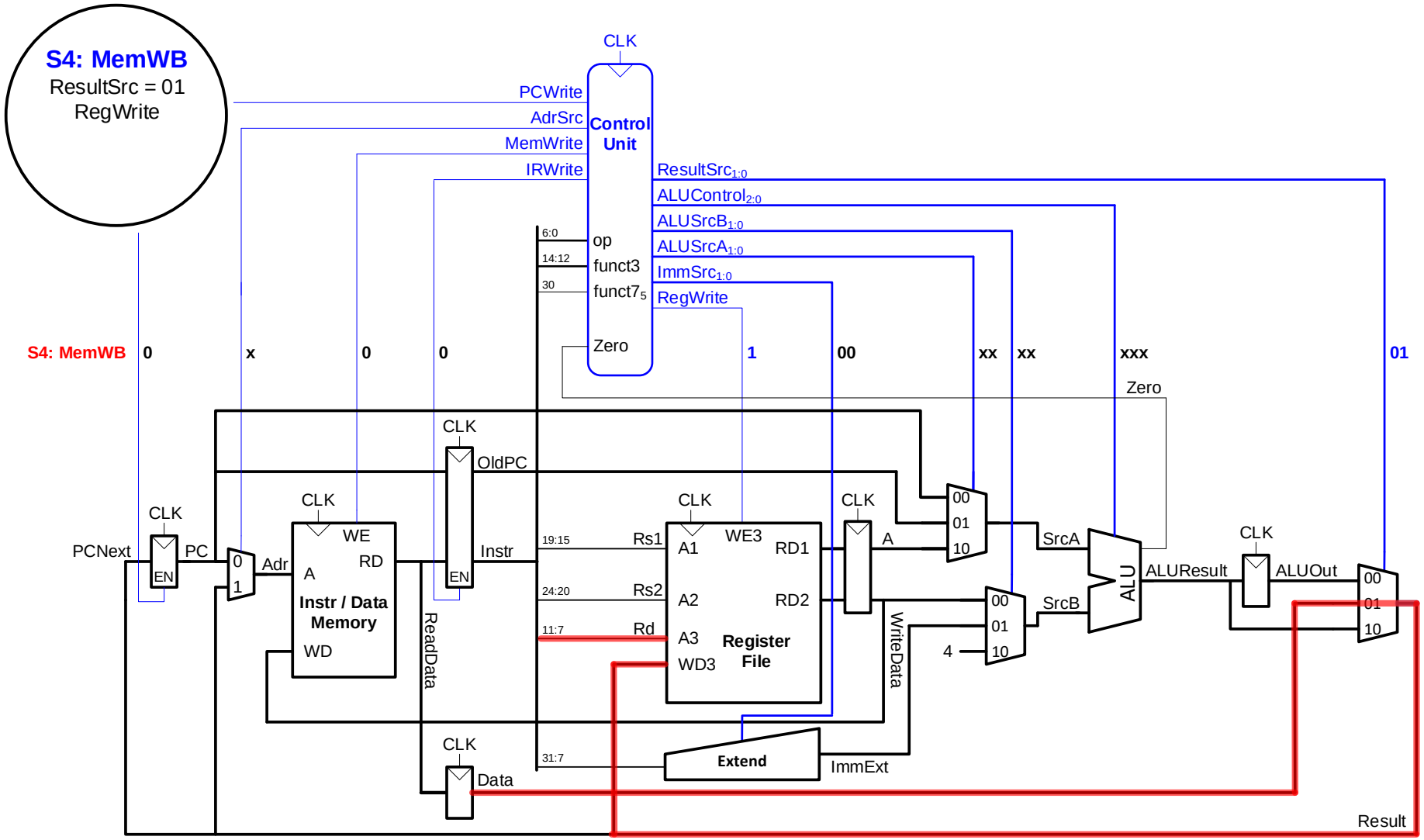
Main FSM: Read Memory Datapath



Main FSM: Write RF

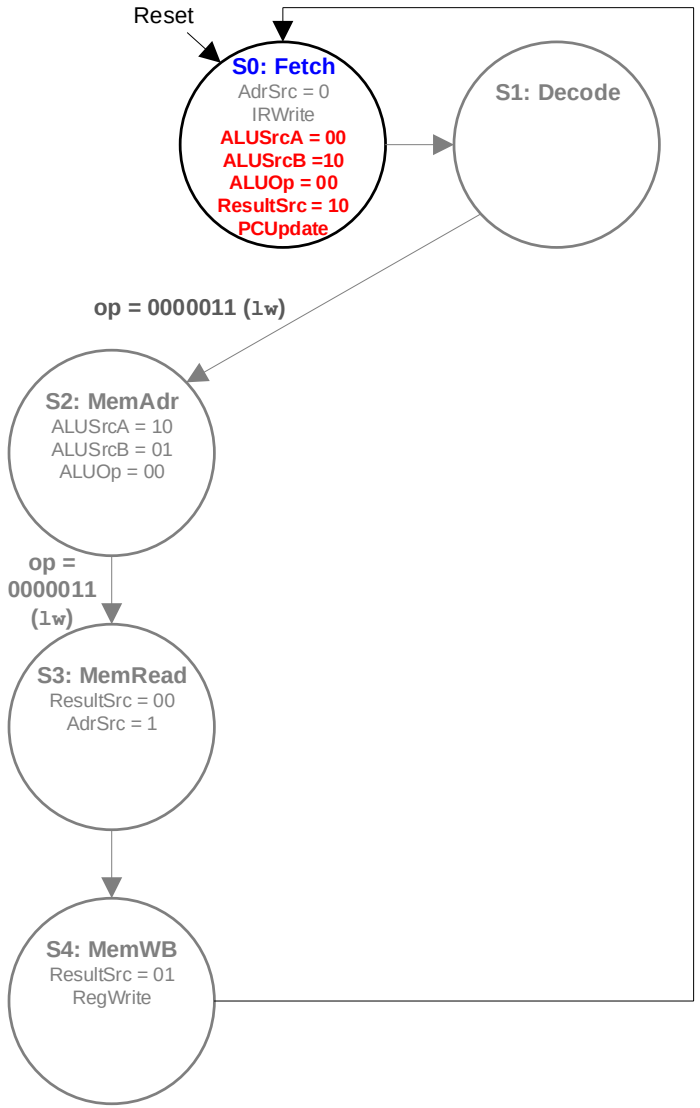


Main FSM: Write RF Datapath

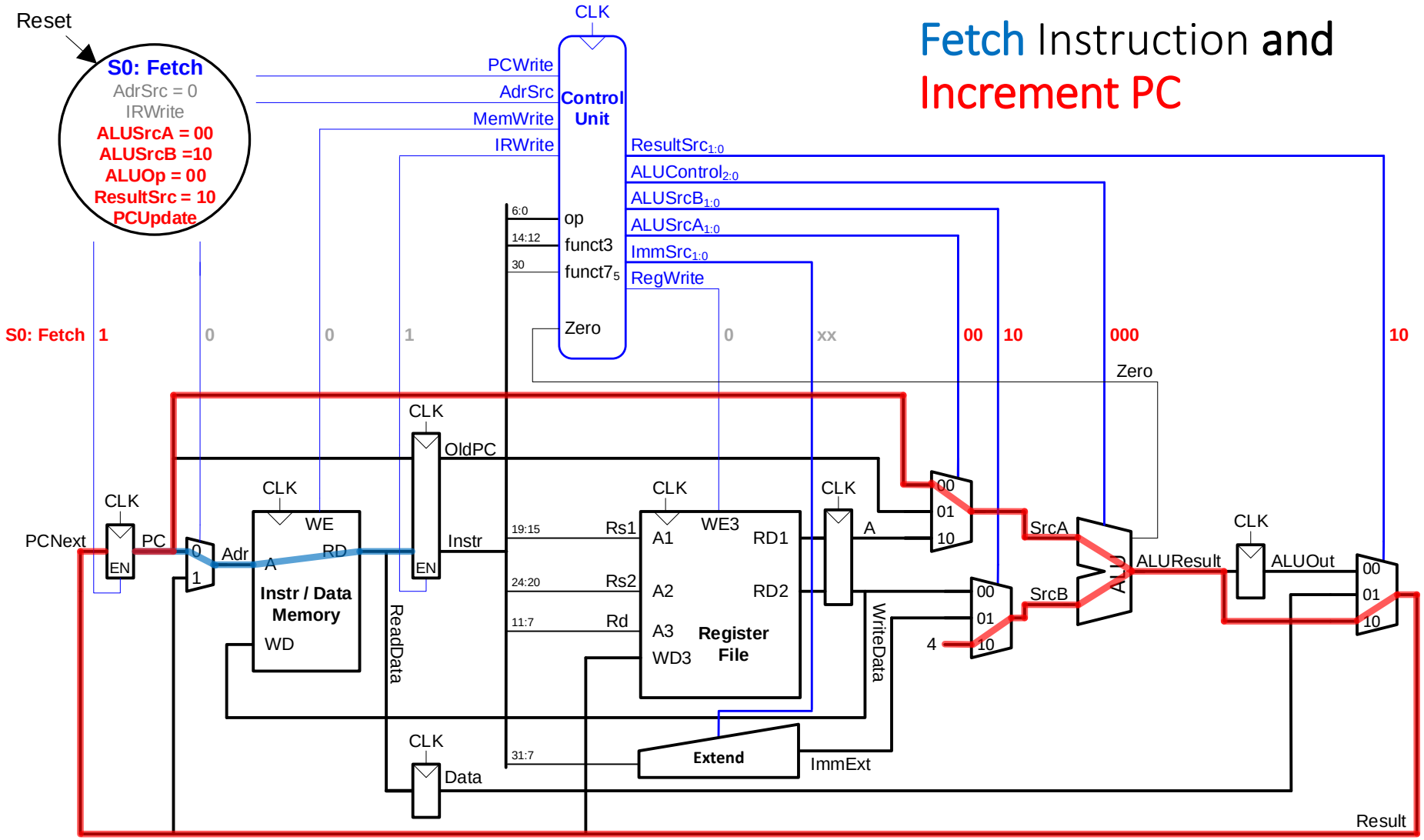


Main FSM: Fetch Revisited

Calculate PC+4 during Fetch stage (ALU isn't being used)

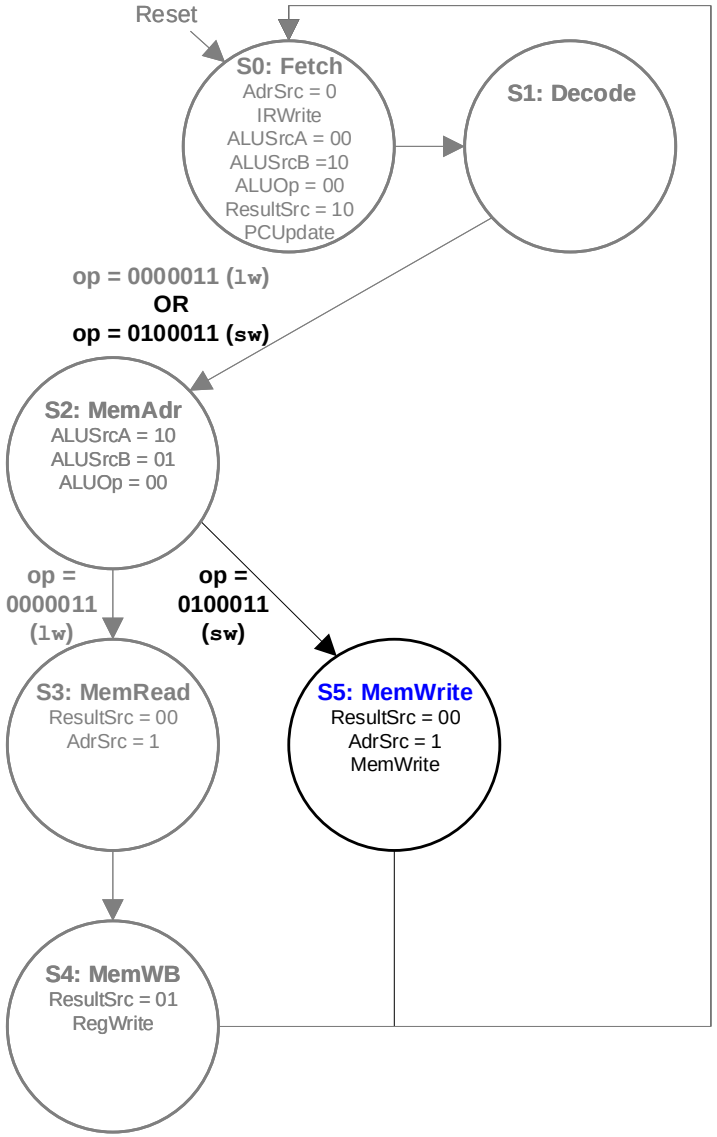


Main FSM: Fetch (PC+4) Datapath

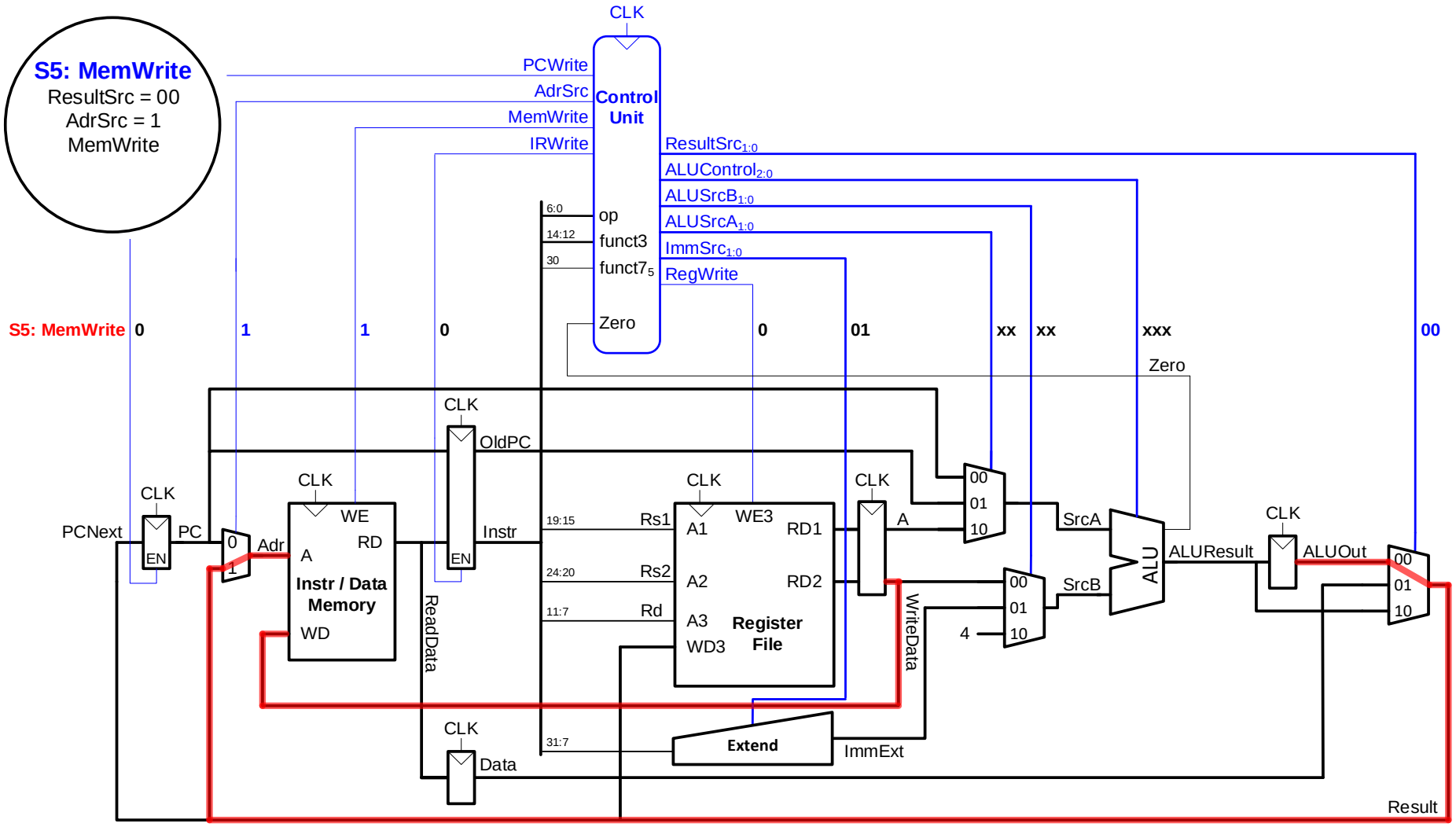


Multicycle Control: Other Instructions

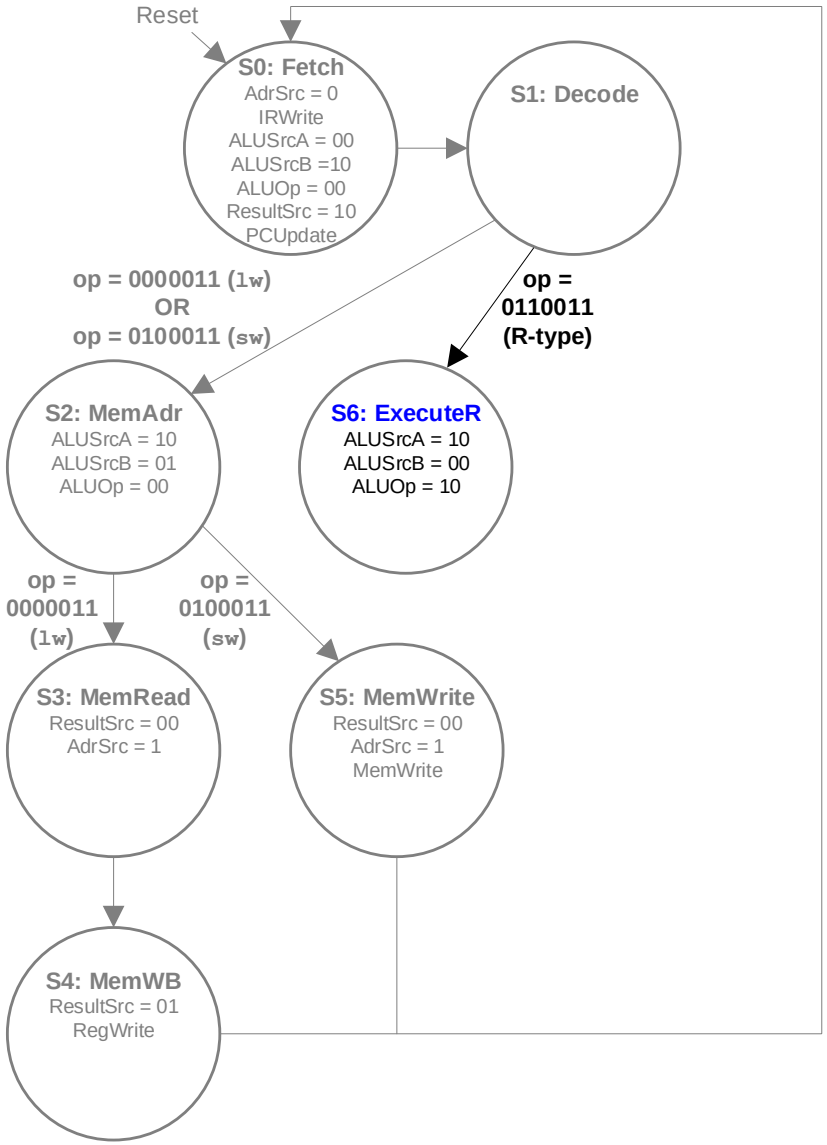
Main FSM: sw


Reset
S0: Fetch
AdrSrc = 0
IRWrite
ALUSrcA = 00
ALUSrcB =10
ALUOp = 00
ResultSrc = 10
PCUpdate
S1: Decode
op = 0000011 (lw)
OR
op = 0100011 (sw)
S2: MemAdr
ALUSrcA = 10
ALUSrcB = 01
ALUOp = 00
op = 0000011 (lw)
op = 0100011 (sw)
S3: MemRead
ResultSrc = 00
AdrSrc = 1
S5: MemWrite
ResultSrc = 00
AdrSrc = 1
MemWrite
S4: MemWB
ResultSrc = 01
RegWrite

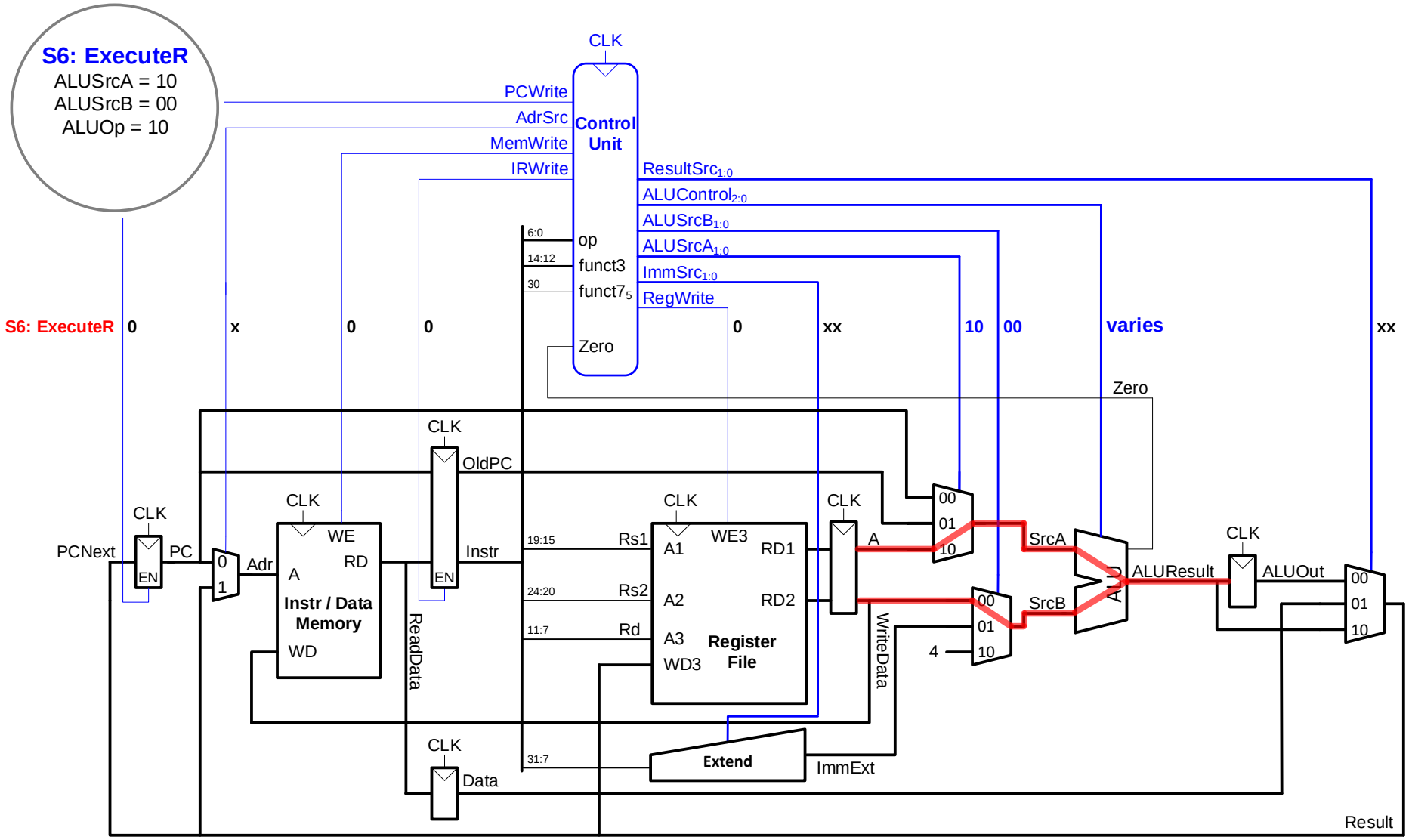
Main FSM: sw Datapath



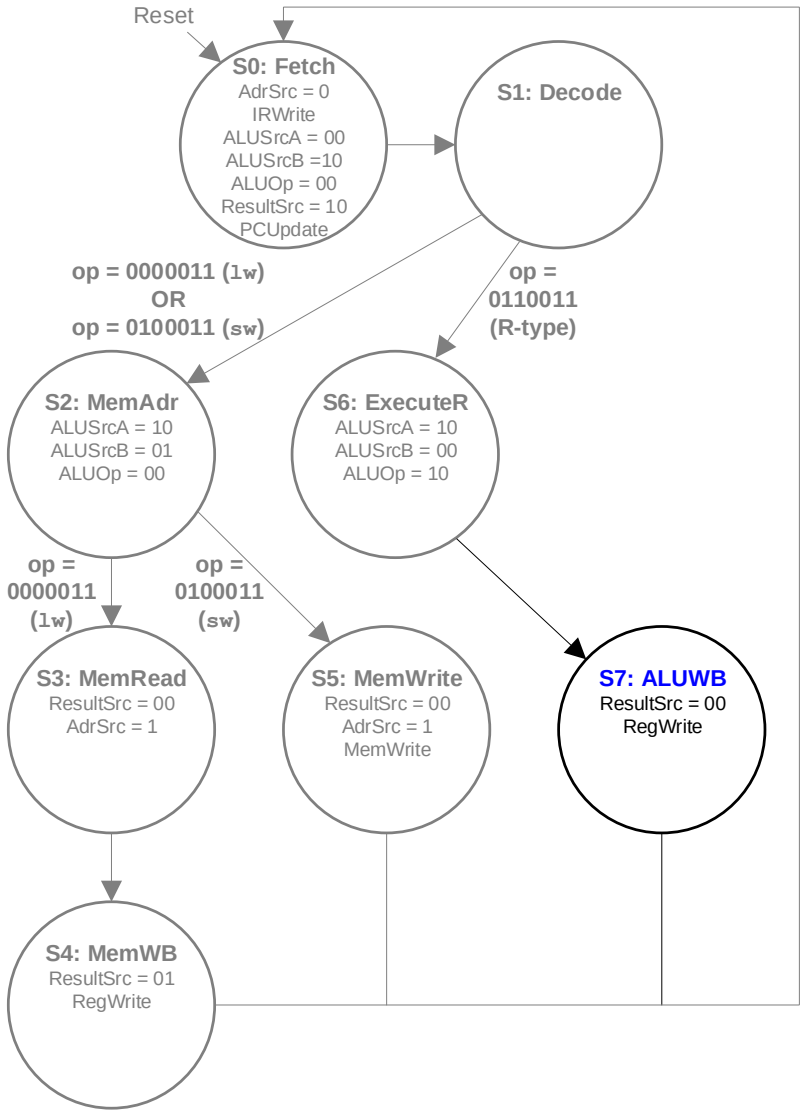
Main FSM: R-Type Execute



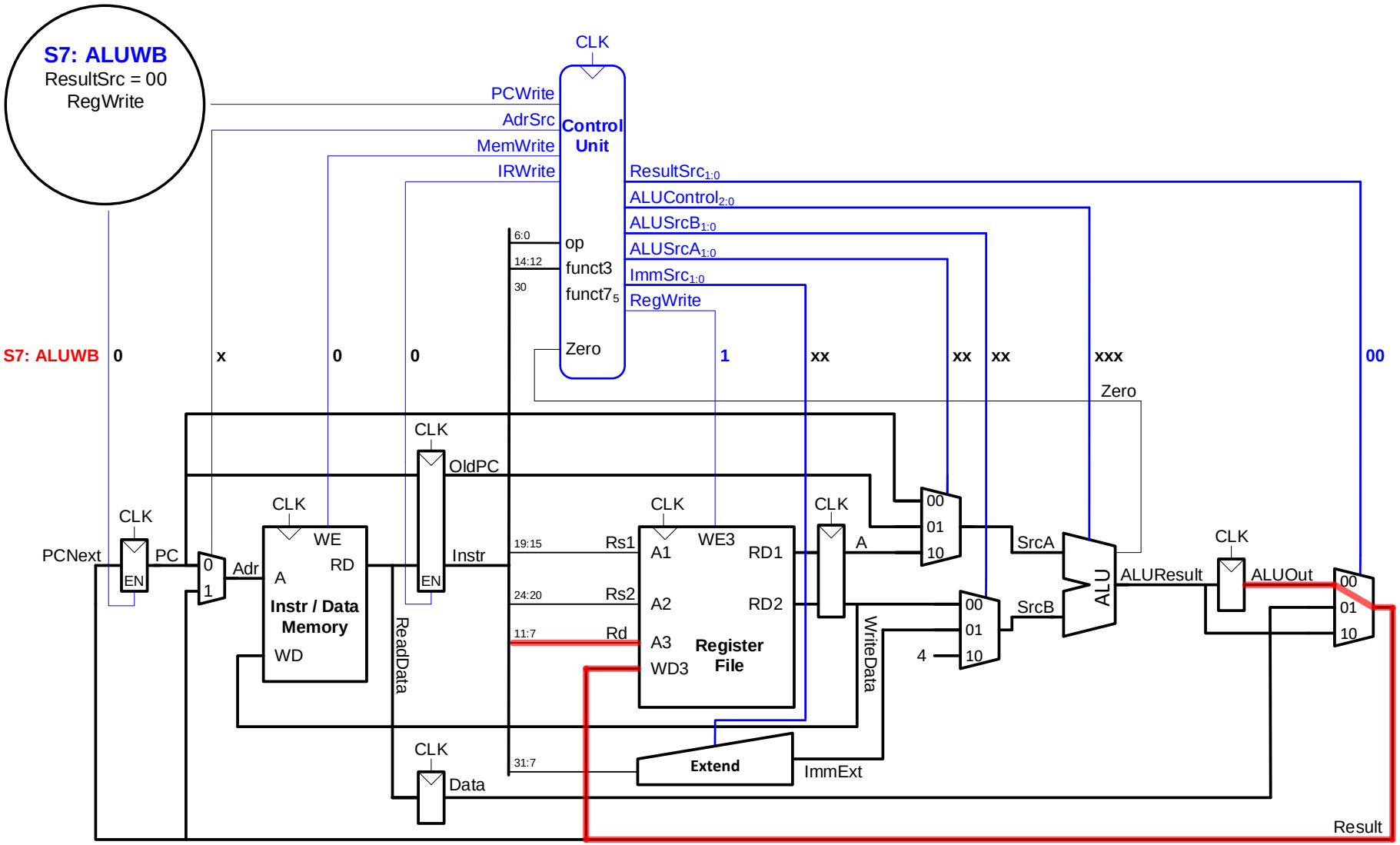
Main FSM: R-Type Execute Datapath



Main FSM: R-Type ALU Write Back



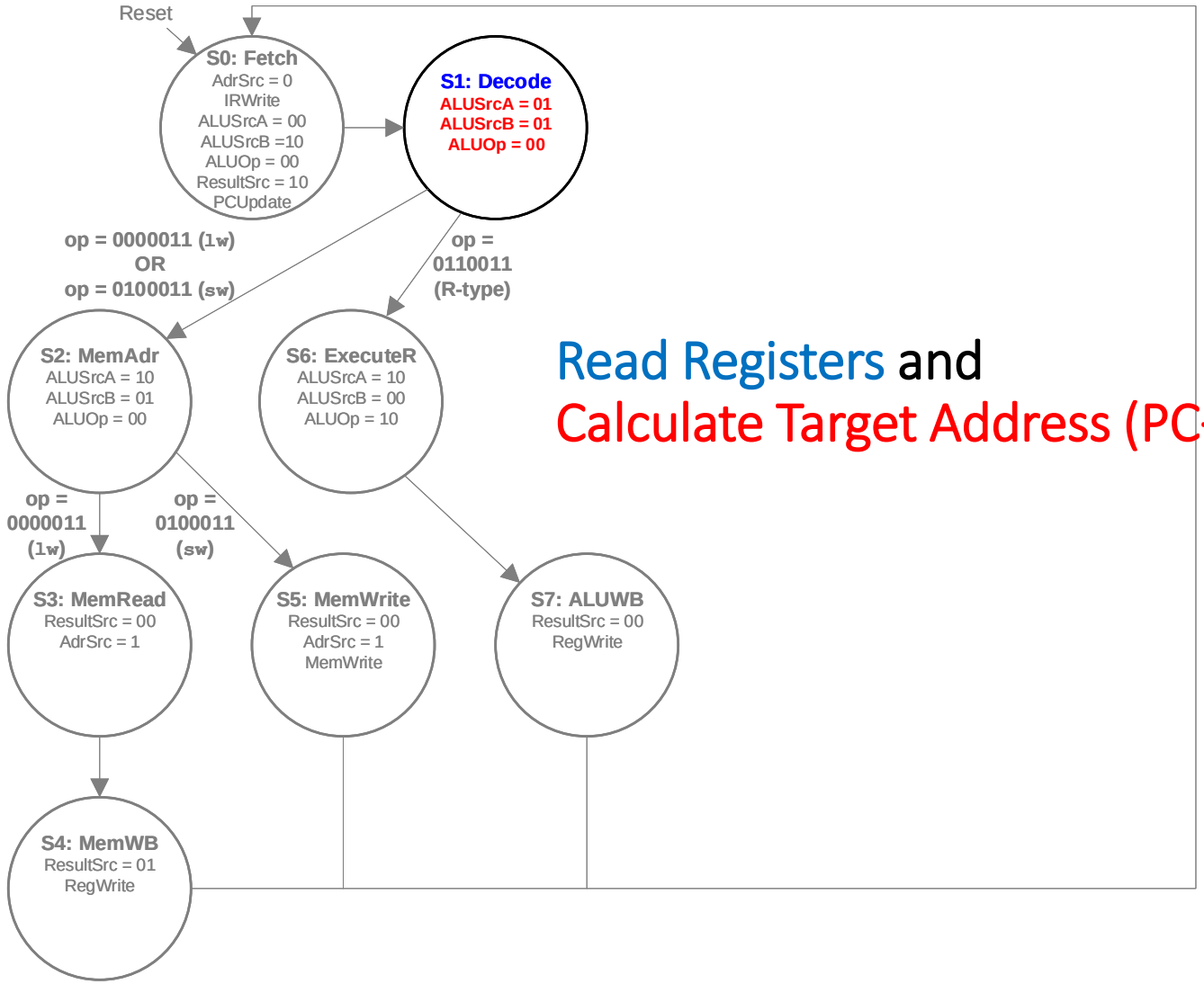
Main FSM: R-Type ALU Write Back



Main FSM: beq

- **Need to calculate:**
  - Branch Target Address
  - **rs1** - **rs2** (to see if equal)
- **ALU** isn't being used in Decode stage
  - Use it to calculate Target Address (PC + imm)

Main FSM: Decode Revisited

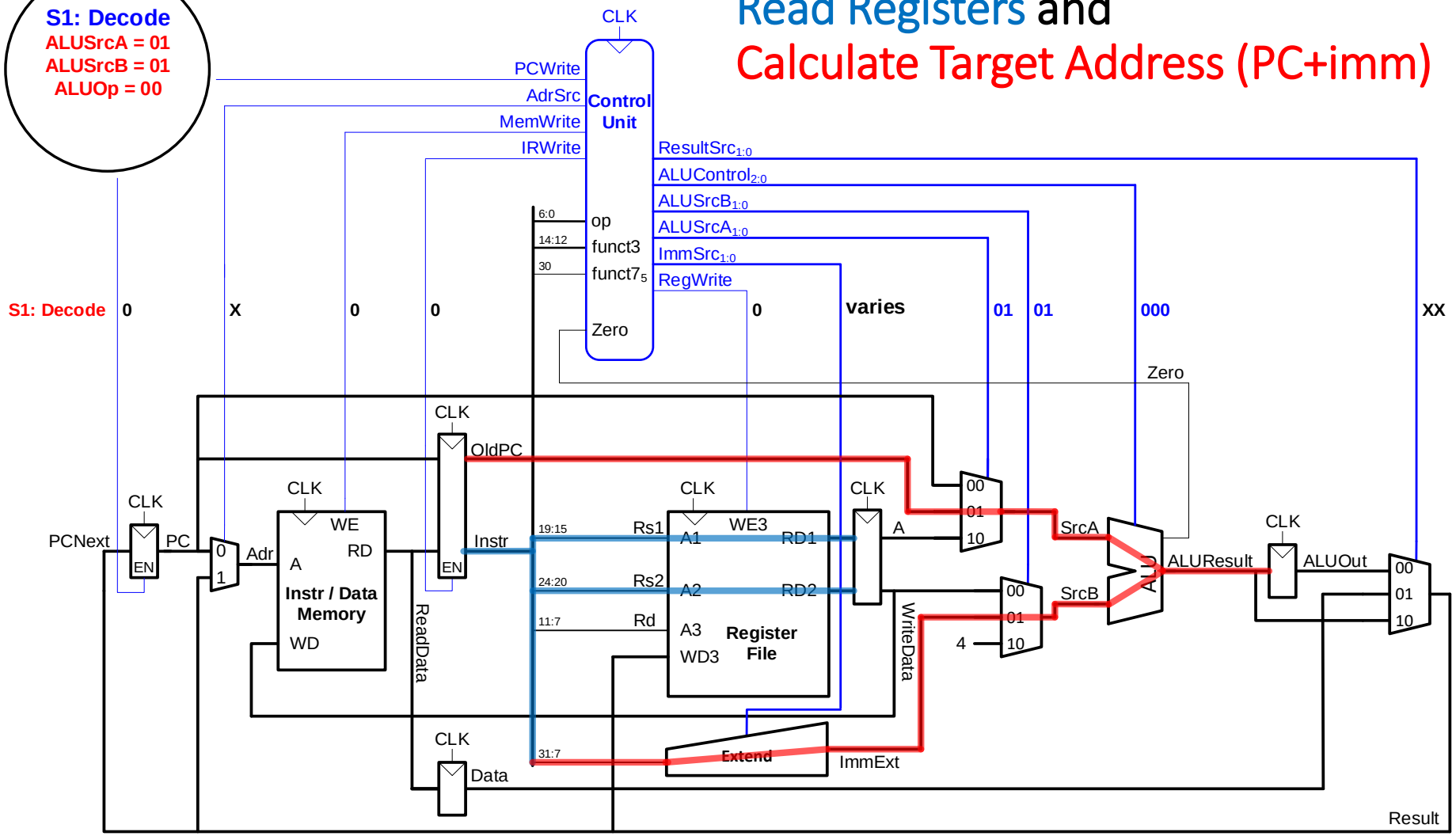


Read Registers and
Calculate Target Address (PC+imm)

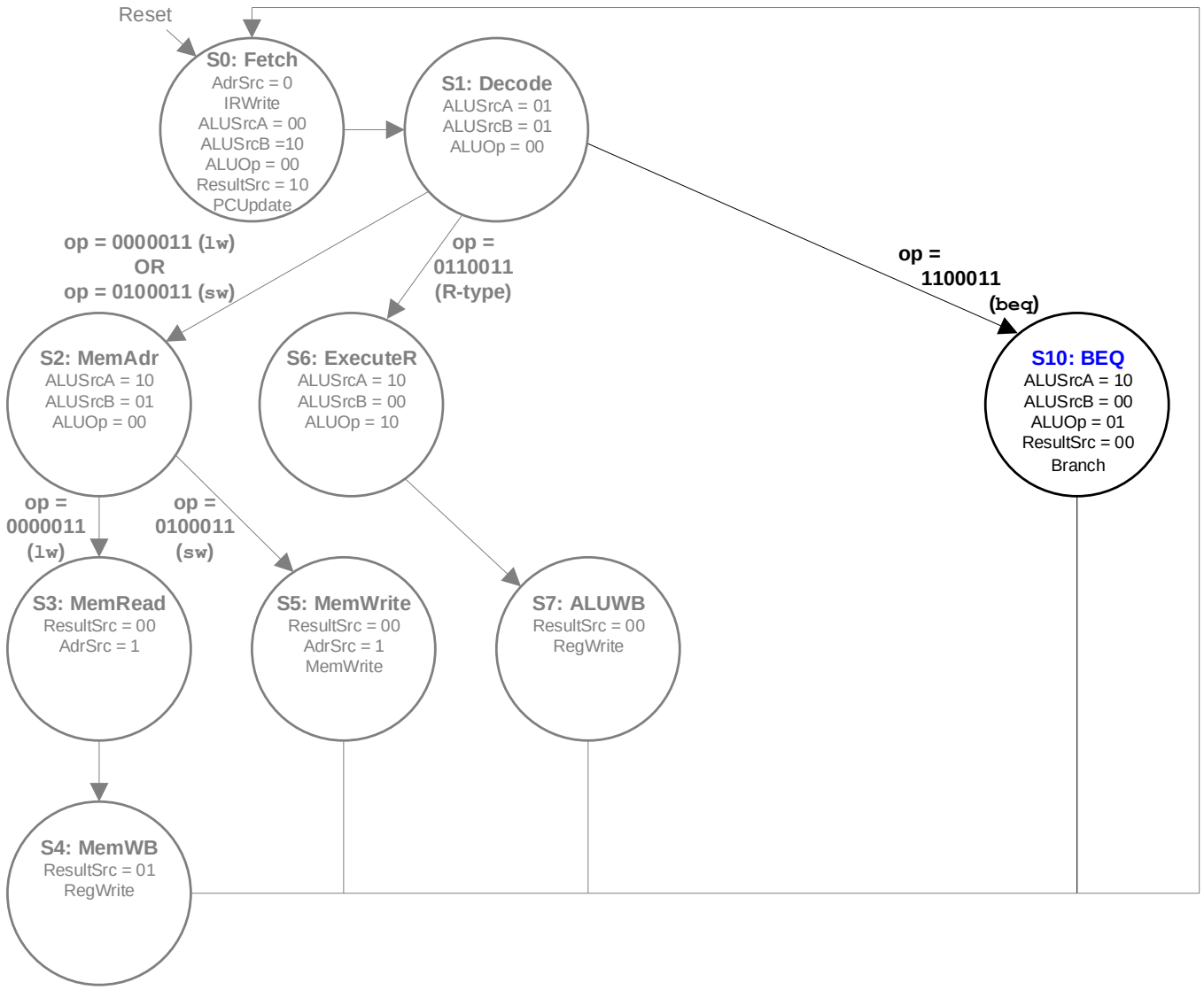
Main FSM: Decode (Target Address)

S1: Decode
ALUSrcA = 01
ALUSrcB = 01
ALUOp = 00

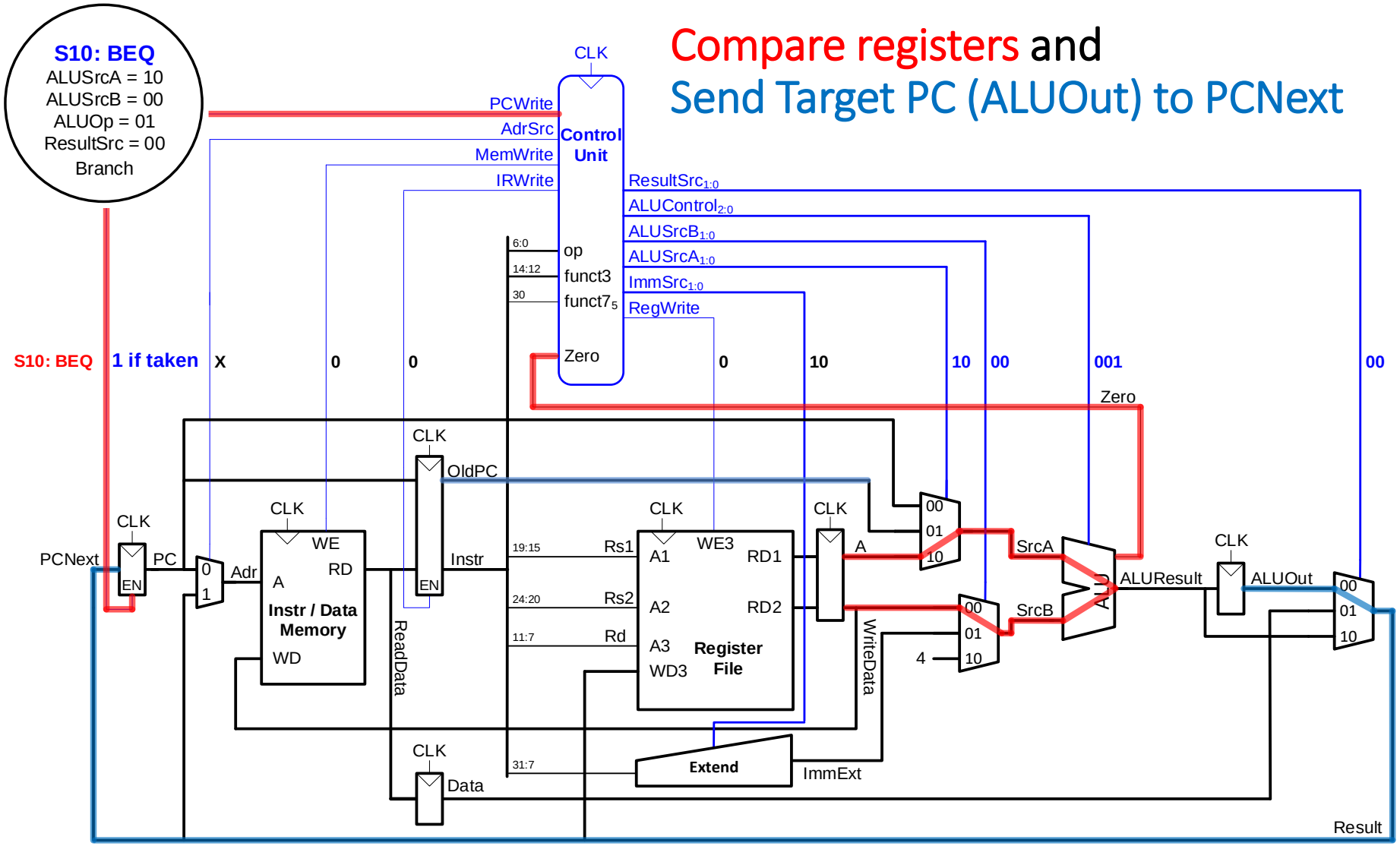
Read Registers and Calculate Target Address (PC+imm)



Main FSM: beq

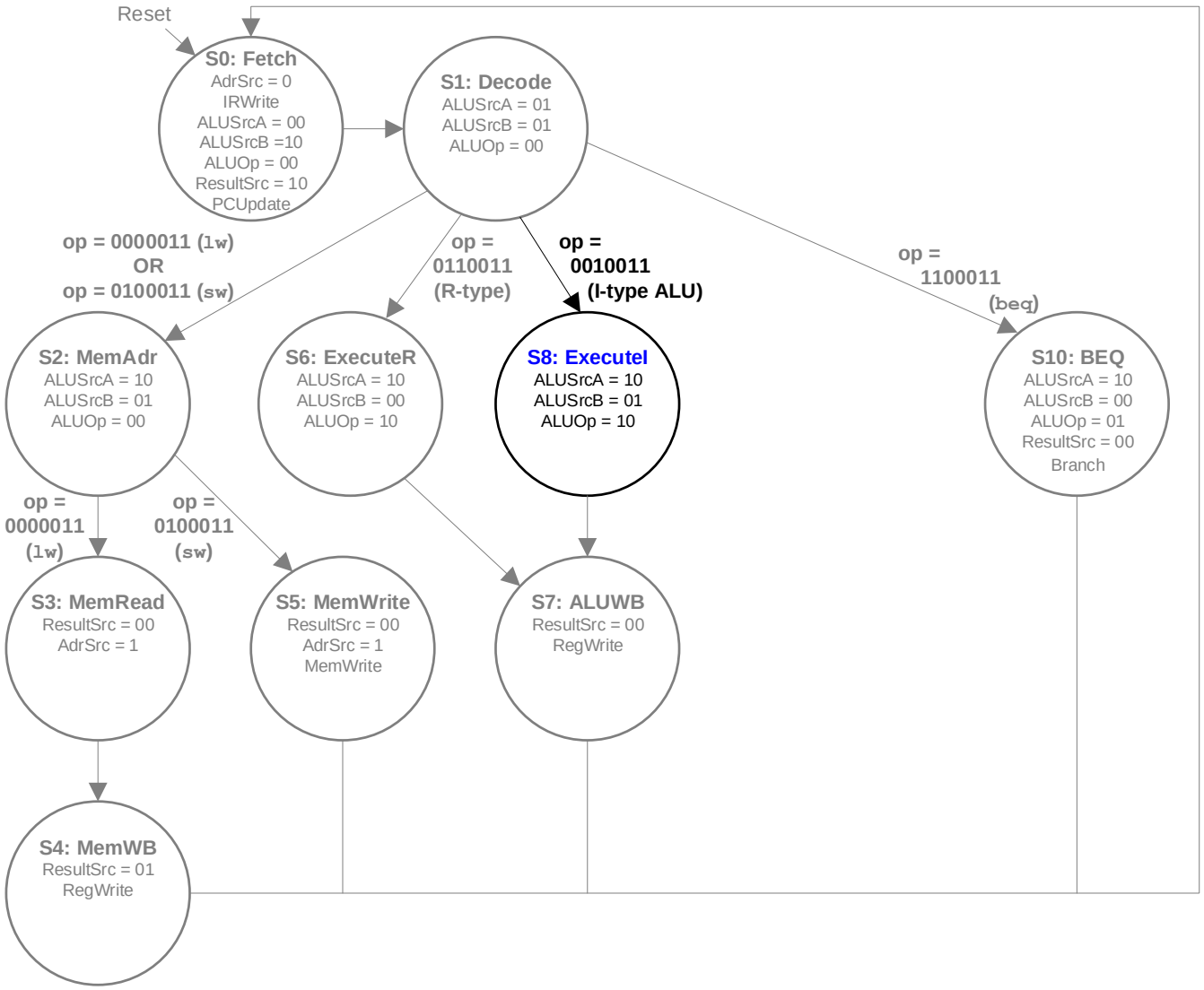

Reset
S0: Fetch
AdrSrc = 0
IRWrite
ALUSrcA = 00
ALUSrcB =10
ALUOp = 00
ResultSrc = 10
PCUpdate
S1: Decode
ALUSrcA = 01
ALUSrcB = 01
ALUOp = 00
op = 0000011 (lw)
OR
op = 0100011 (sw)
op = 0110011 (R-type)
op = 1100011 (beq)
S2: MemAdr
ALUSrcA = 10
ALUSrcB = 01
ALUOp = 00
S6: ExecuteR
ALUSrcA = 10
ALUSrcB = 00
ALUOp = 10
S10: BEQ
ALUSrcA = 10
ALUSrcB = 00
ALUOp = 01
ResultSrc = 00
Branch
op = 0000011 (lw)
op = 0100011 (sw)
S3: MemRead
ResultSrc = 00
AdrSrc = 1
S5: MemWrite
ResultSrc = 00
AdrSrc = 1
MemWrite
S7: ALUWB
ResultSrc = 00
RegWrite
S4: MemWB
ResultSrc = 01
RegWrite

Main FSM: beq Datapath

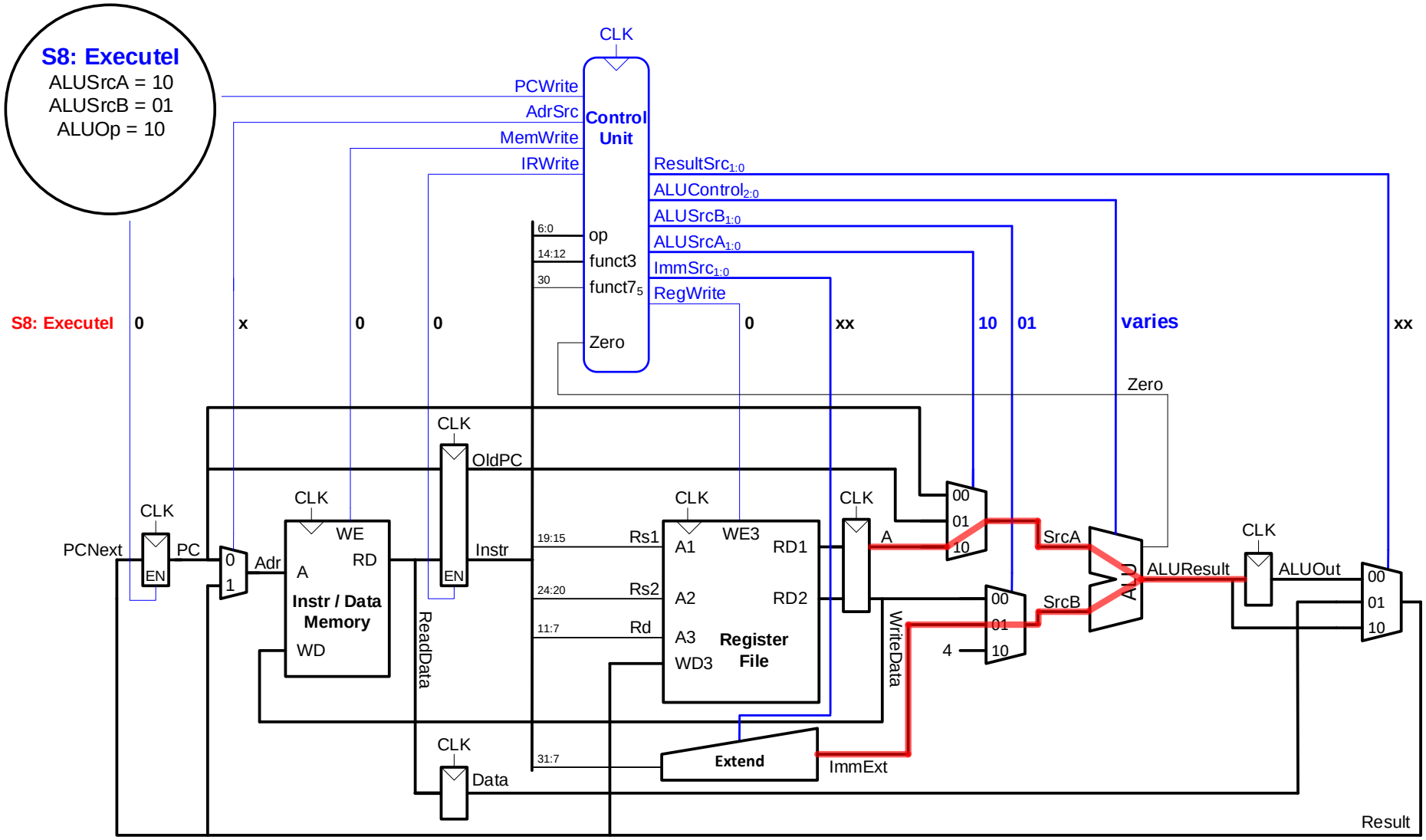


Extending the RISC-V Multicycle Processor

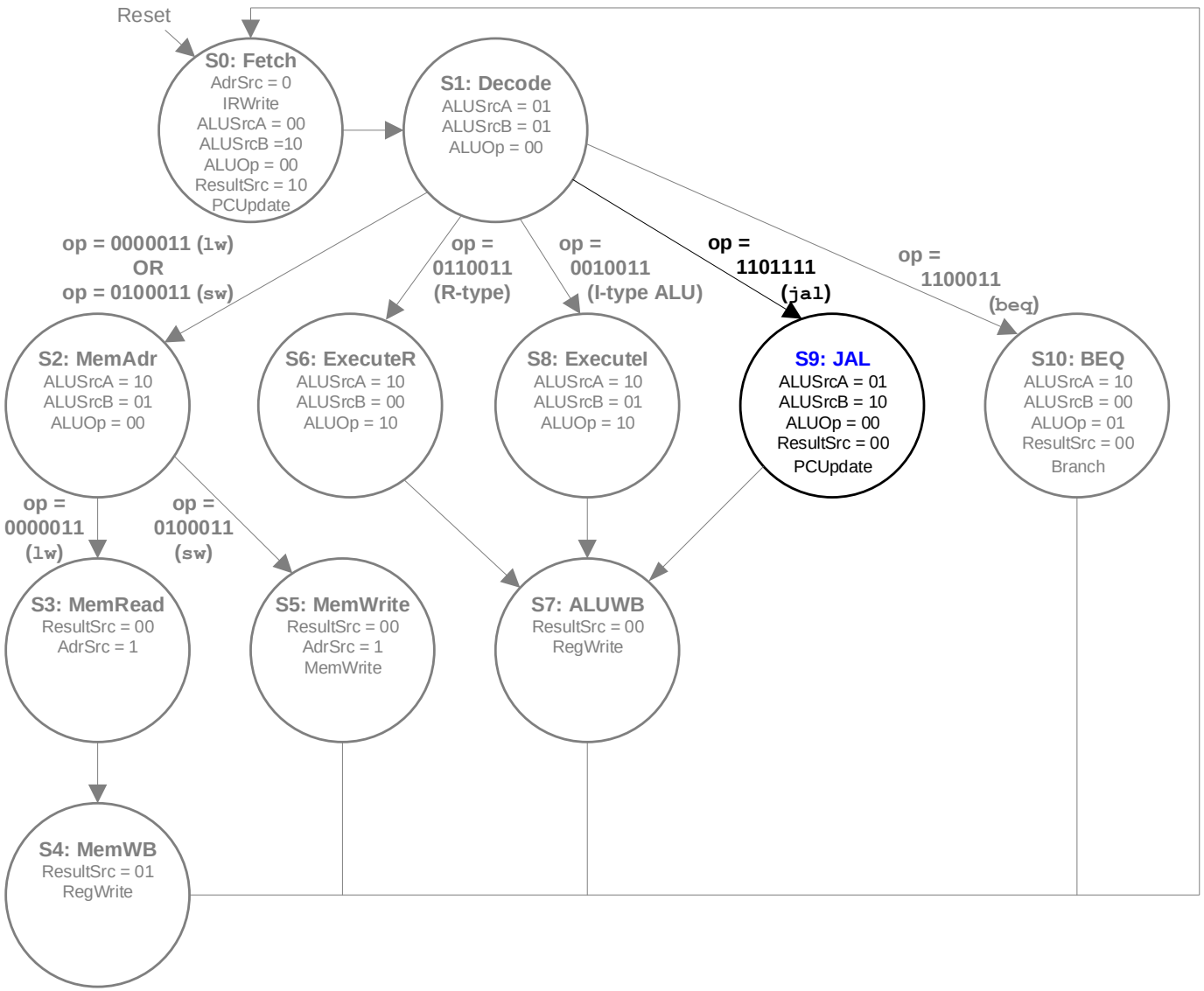
Main FSM: I-Type ALU Execute



Main FSM: I-Type ALU Exec. Datapath

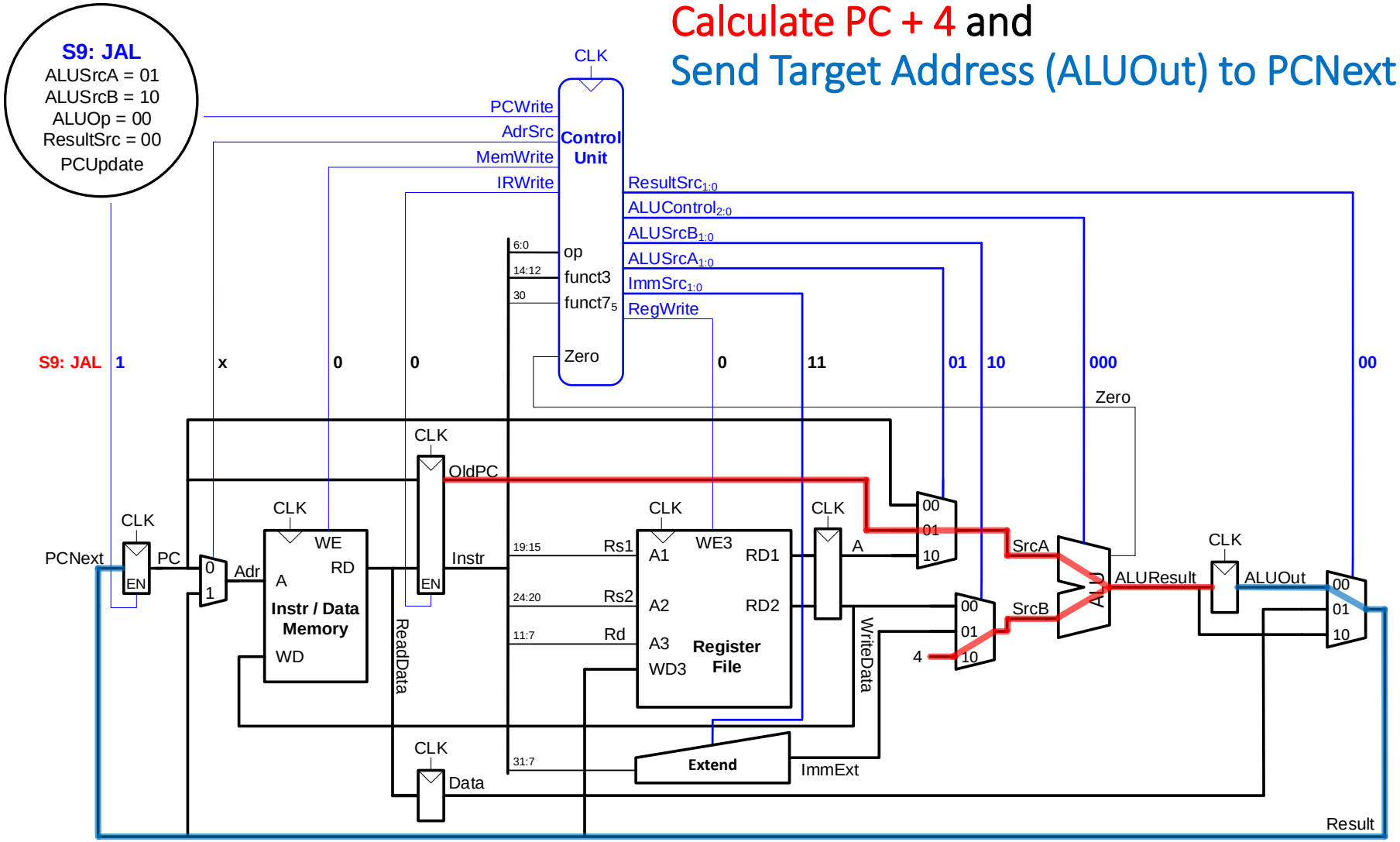


Main FSM: jal


Reset
S0: Fetch
AdrSrc = 0
IRWrite
ALUSrcA = 00
ALUSrcB =10
ALUOp = 00
ResultSrc = 10
PCUpdate
S1: Decode
ALUSrcA = 01
ALUSrcB = 01
ALUOp = 00
op = 0000011 (lw)
OR
op = 0100011 (sw)
op = 0110011 (R-type)
op = 0010011 (I-type ALU)
op = 1101111 (jal)
op = 1100011 (beq)
S2: MemAdr
ALUSrcA = 10
ALUSrcB = 01
ALUOp = 00
S6: ExecuteR
ALUSrcA = 10
ALUSrcB = 00
ALUOp = 10
S8: ExecuteI
ALUSrcA = 10
ALUSrcB = 01
ALUOp = 10
S9: JAL
ALUSrcA = 01
ALUSrcB = 10
ALUOp = 00
ResultSrc = 00
PCUpdate
S10: BEQ
ALUSrcA = 10
ALUSrcB = 00
ALUOp = 01
ResultSrc = 00
Branch
op = 0000011 (lw)
op = 0100011 (sw)
S3: MemRead
ResultSrc = 00
AdrSrc = 1
S5: MemWrite
ResultSrc = 00
AdrSrc = 1
MemWrite
S7: ALUWB
ResultSrc = 00
RegWrite
S4: MemWB
ResultSrc = 01
RegWrite

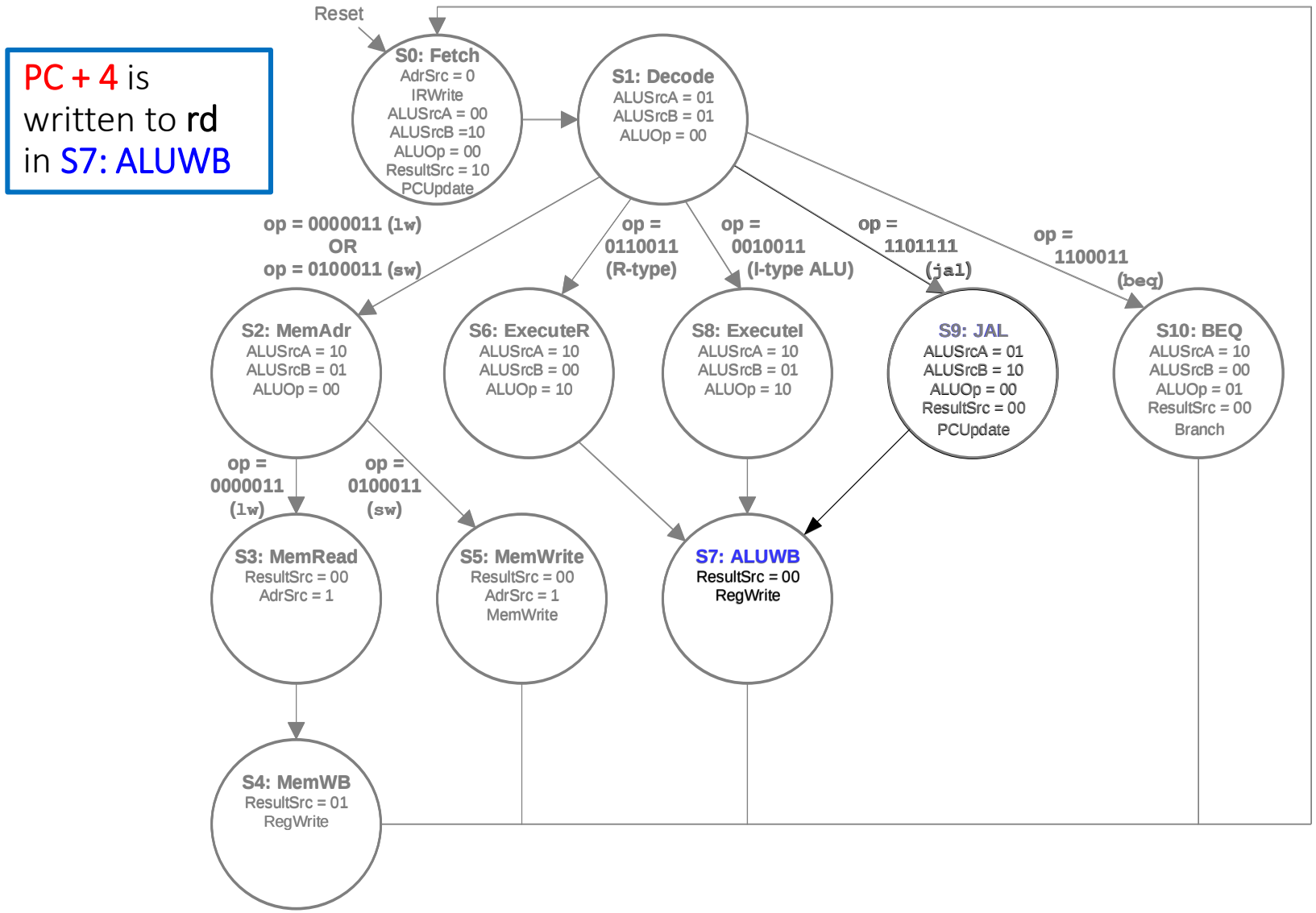
Main FSM: `jal` Datapath

Calculate PC + 4 and
Send Target Address (ALUOut) to PCNext



Main FSM: `jal`

PC + 4 is written to **rd** in S7: ALUWB



Multicycle Processor Main FSM

State	Datapath μOp
Fetch	Instr ←Mem[PC]; PC ← PC+4
Decode	ALUOut ← PCTarget
MemAdr	ALUOut ← rs1 + imm
MemRead	Data ← Mem[ALUOut]
MemWB	rd ← Data
MemWrite	Mem[ALUOut] ← rd
ExecuteR	ALUOut ← rs1 op rs2
ExecuteI	ALUOut ← rs1 op imm
ALUWB	rd ← ALUOut
BEQ	ALUResult = rs1-rs2; if Zero, PC ← ALUOut
JAL	PC ← ALUOut; ALUOut ← PC+4