RISC-V microarchitecture

These transparencies are based on those provided with the following book:
David Harris and Sarah Harris, "**Digital Design and Computer Architecture, RISC-V Edition**",
1st Edition, 2021, Elsevier

Introduction

- **Microarchitecture:** how to implement an architecture in hardware
- Processor:
  - **Datapath:** functional blocks
  - **Control:** control signals

- RISC-V standard defines the ISA, not the architecture
- Different RISC-V architectures can be / have been developed
- We will consider a (CISC-like) multicycle architecture (because it is the simplest to understand)

Processor Performance

- **Program execution time**

Execution Time = (#instructions)(cycles/instruction)(seconds/cycle)

- **Definitions:**
  - CPI: cycles/instruction
  - clock period: seconds/cycle
- **Challenge is to satisfy constraints of:**
  - Cost
  - Power
  - Performance

RISC-V Processor

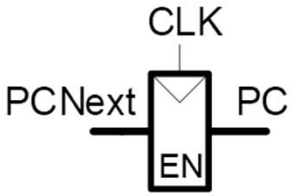
- Consider **subset** of RISC-V instructions:
  - **R-type ALU instructions:**
    - **`add`, `sub`, `and`, `or`, `slt`**
  - **Memory instructions:**
    - **`lw`, `sw`**
  - **Branch instructions:**
    - **`beq`**
  - **I-type ALU instructions:**
    - **`addi`, `andi`, `ori`, `xori`**
  - **J-type instructions:**
    - **`jal`**

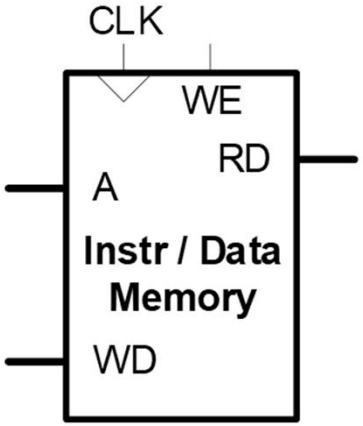
Architectural State Elements

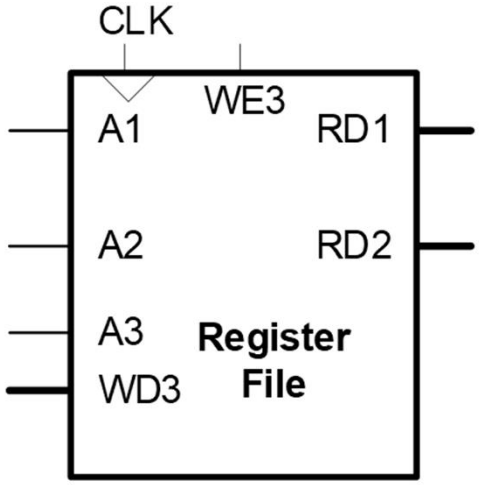
Determine everything about a processor:

- **Architectural state:**
  - 32 registers
  - PC
  - Memory

RISC-V Architectural State Elements

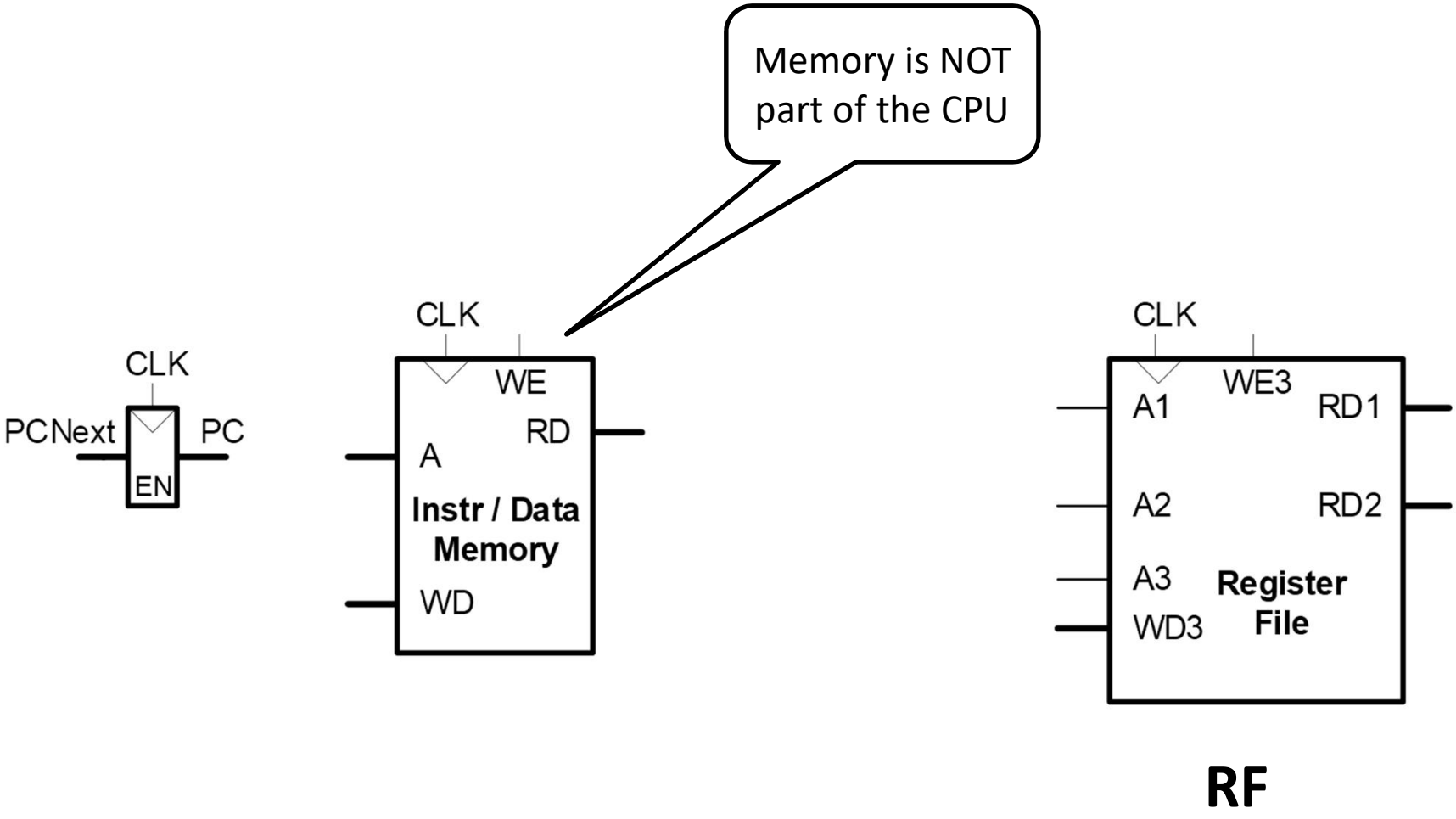




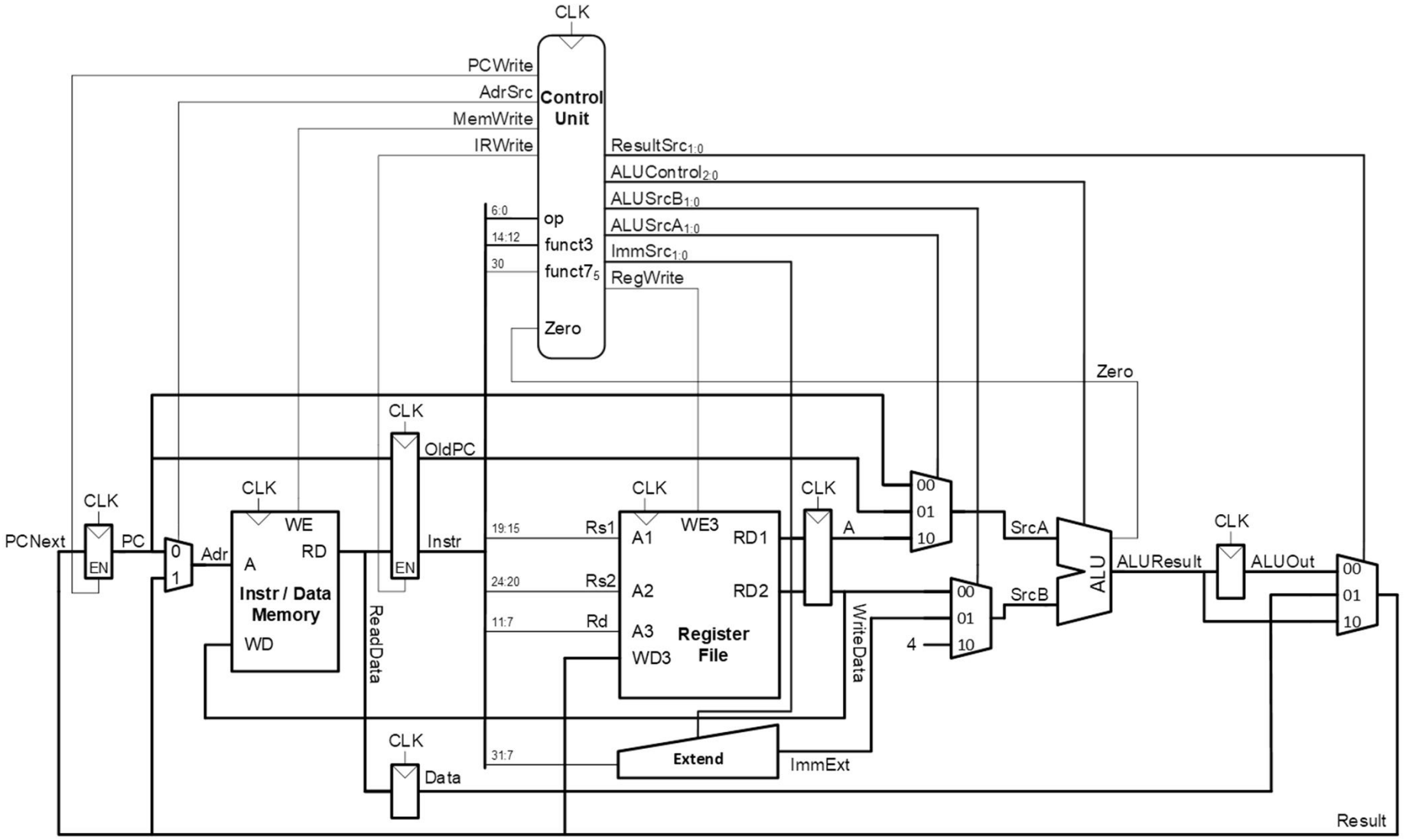


RF

RISC-V Architectural State Elements

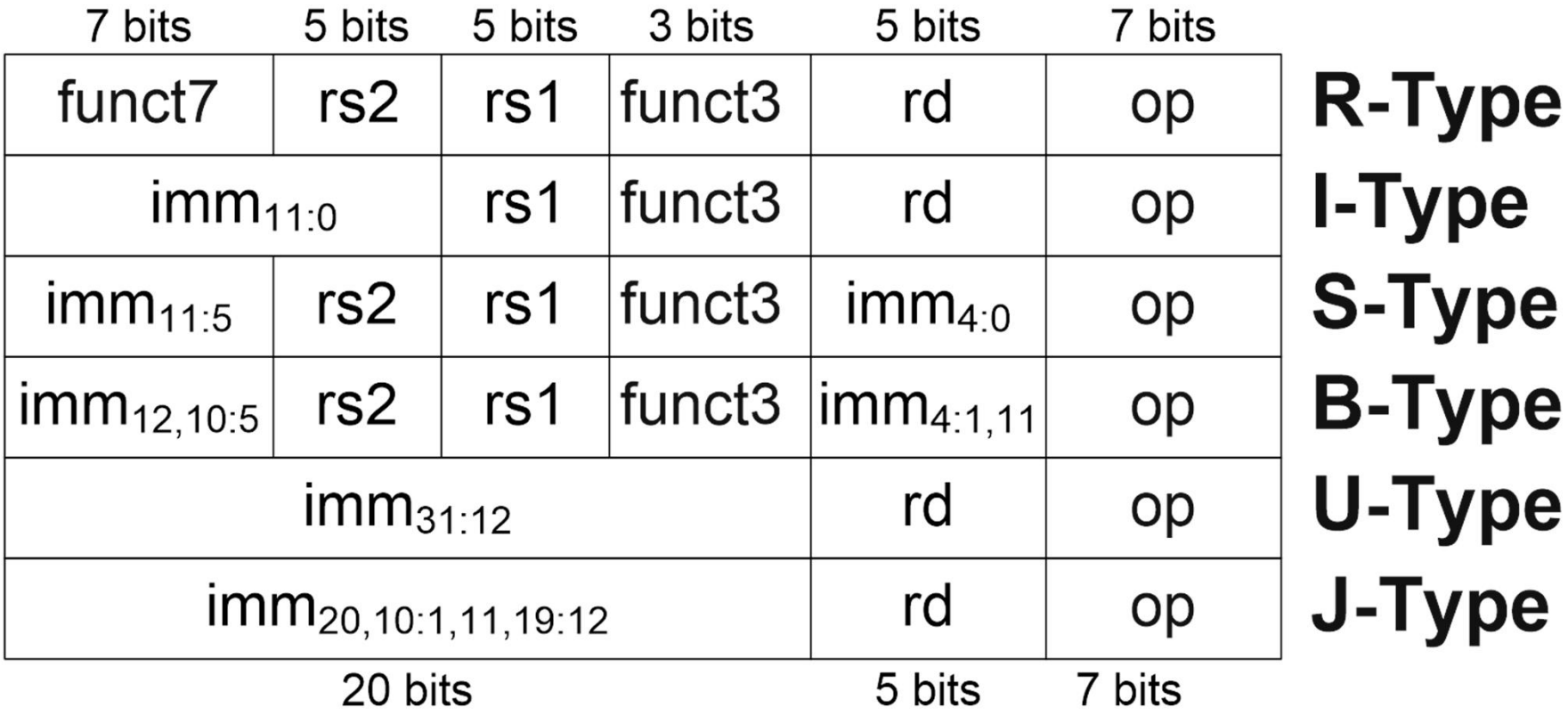


Multicycle RISC-V Processor



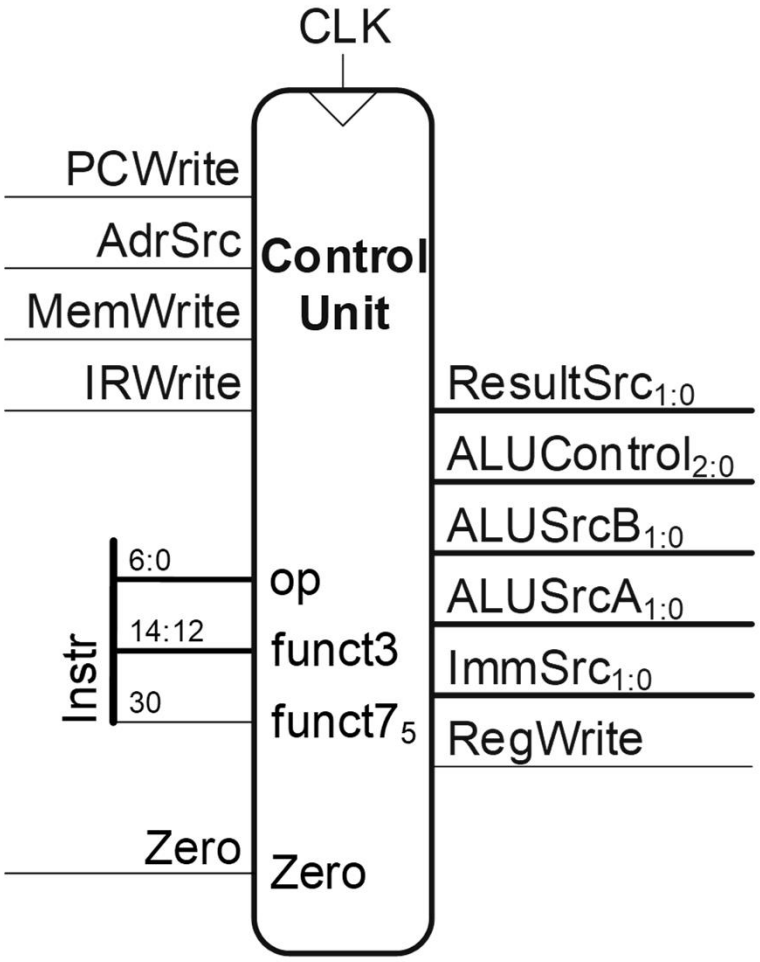
Multicycle Control

RISC-V instruction formats

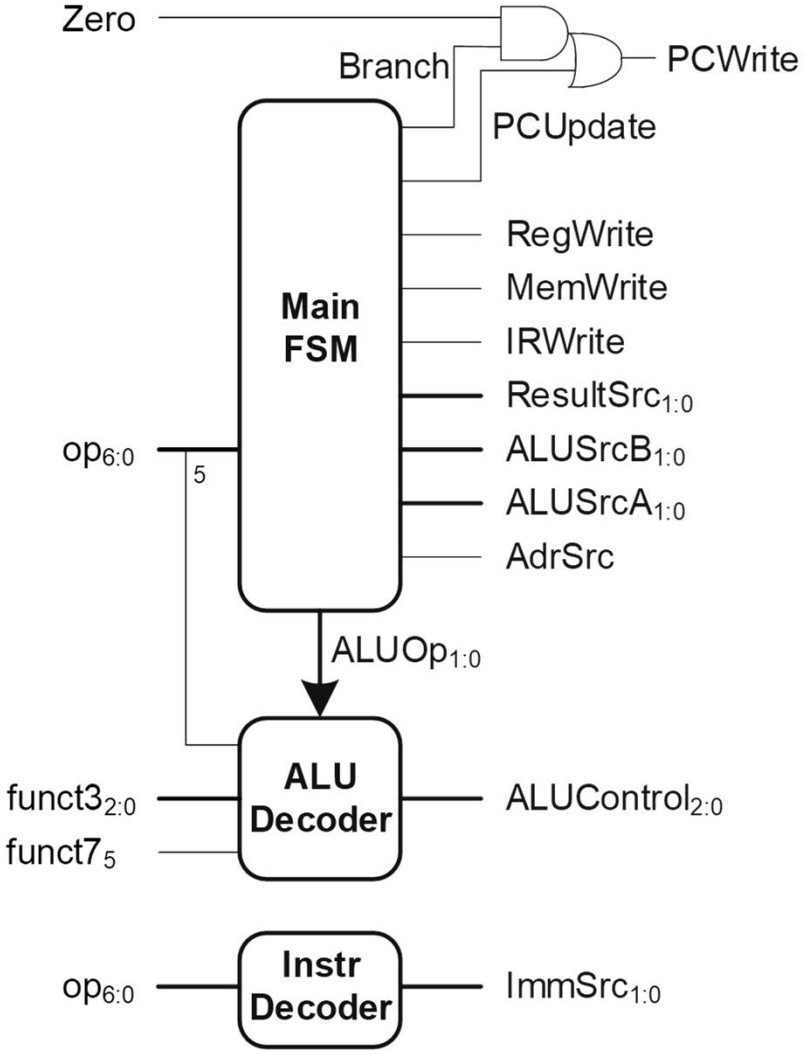


Multicycle Control

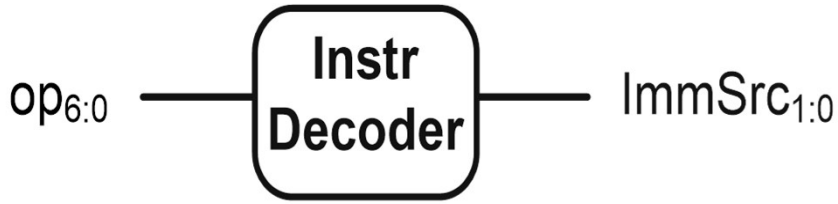
High-Level View



Low-Level View

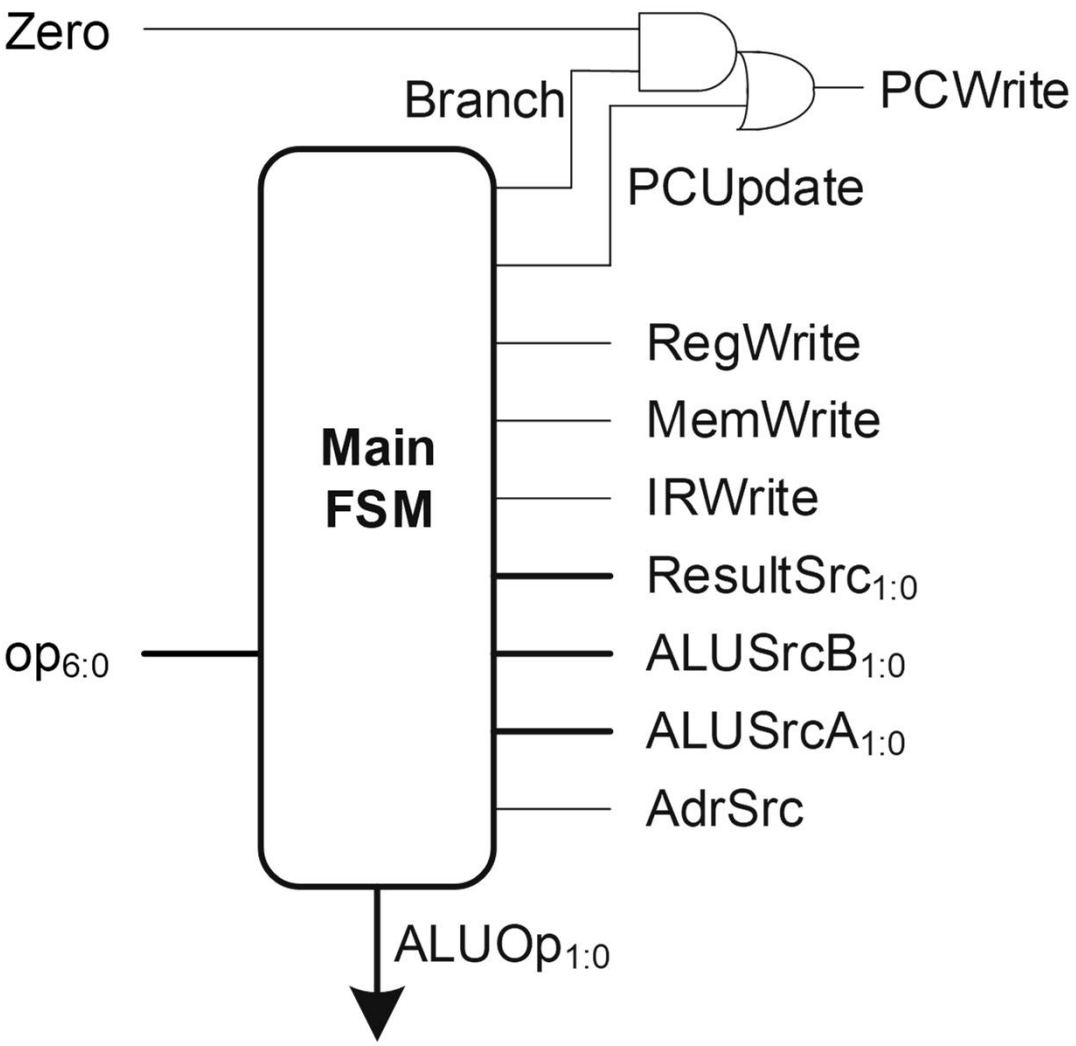


Multicycle Control: Instruction Decoder



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

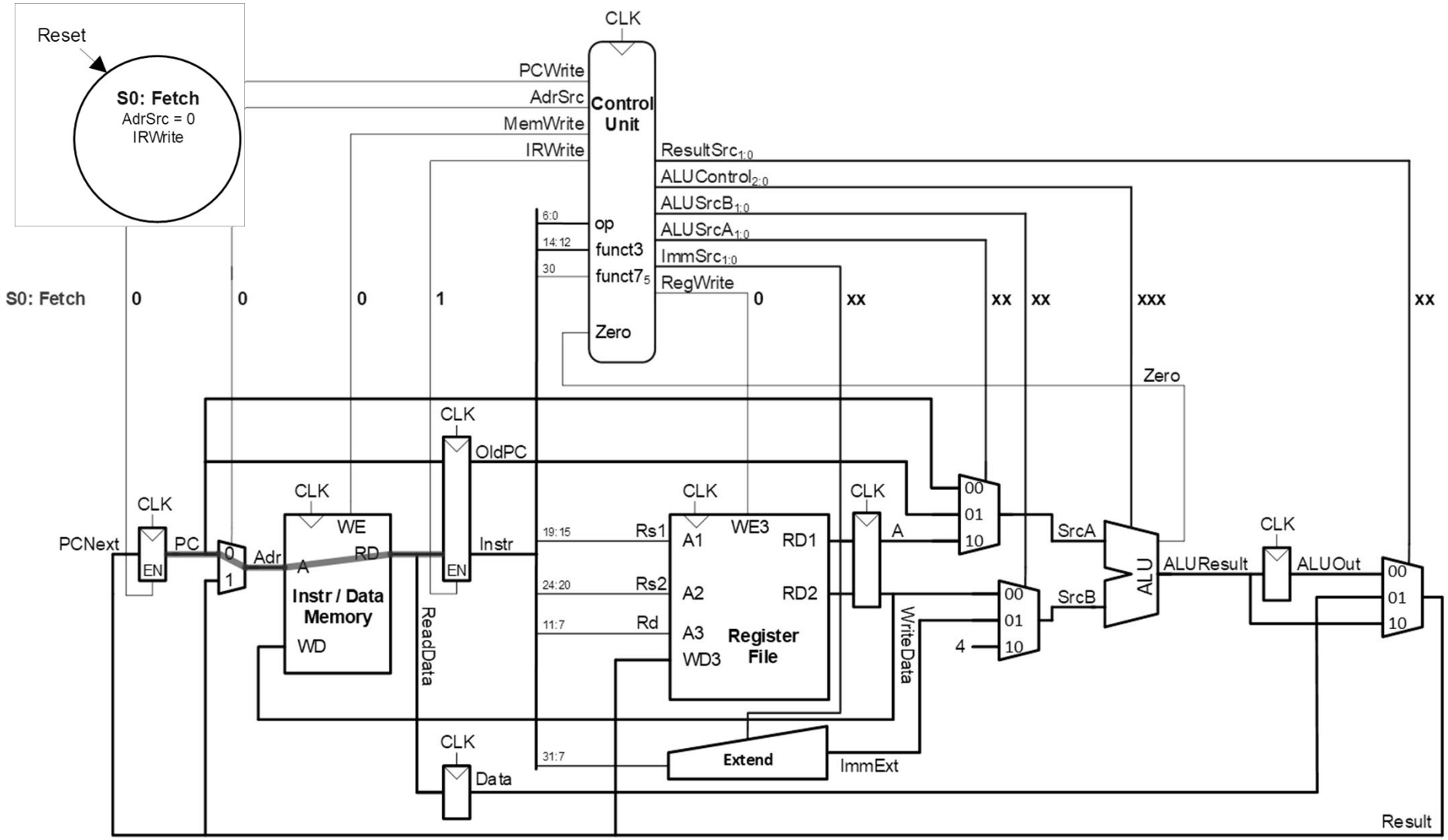
Multicycle Control: Main FSM



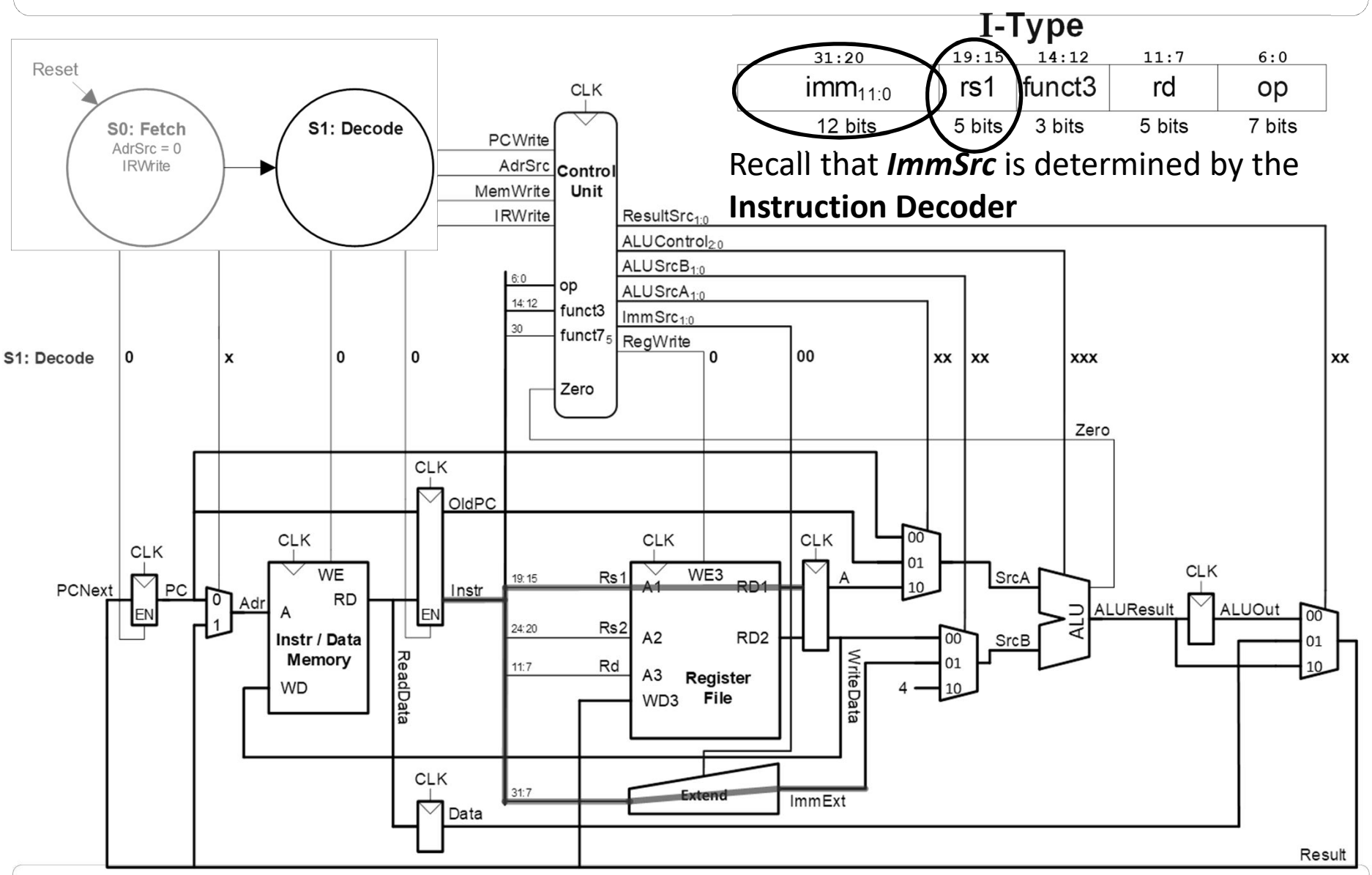
To declutter FSM:

- **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

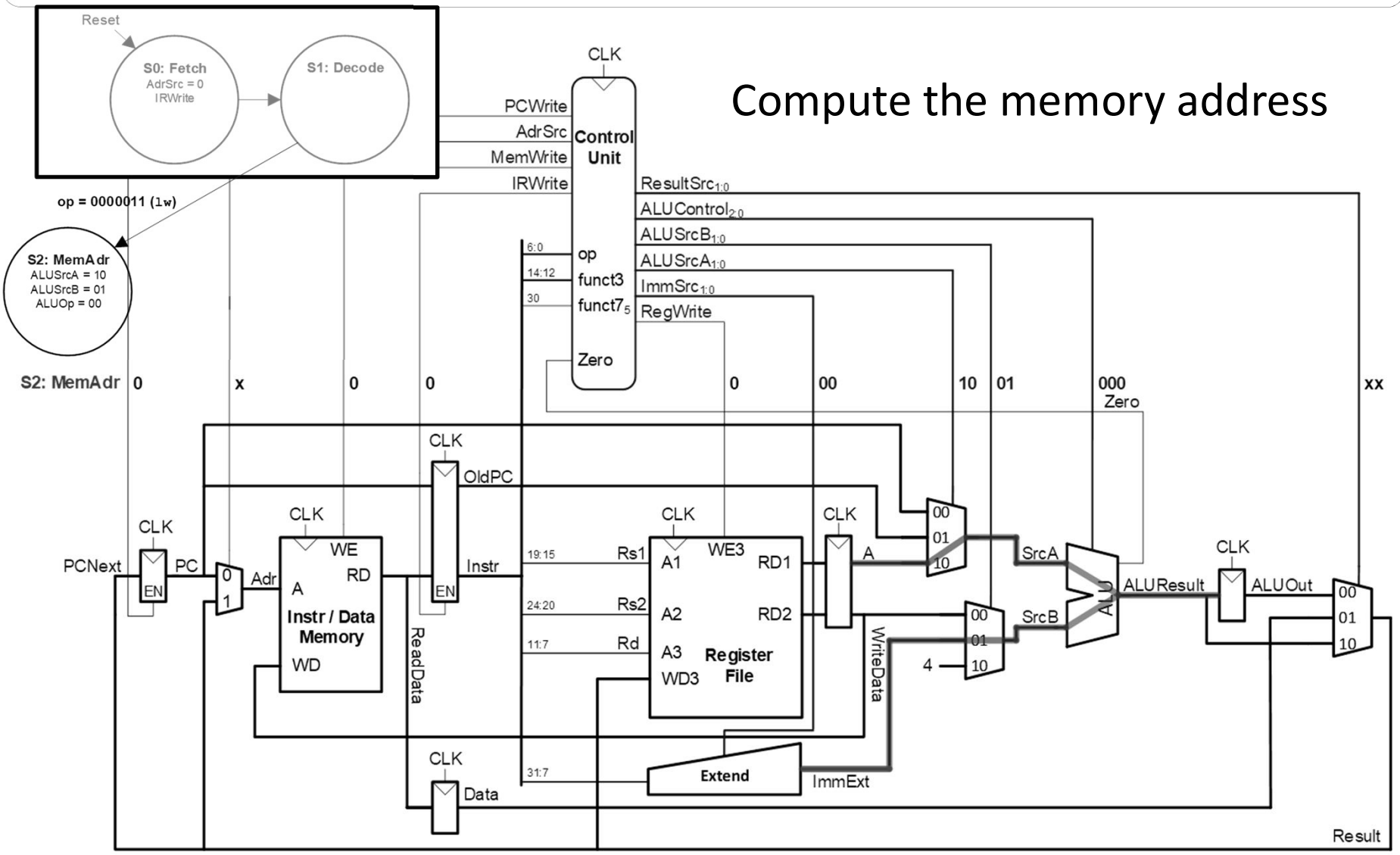
Main FSM: Fetch



Main FSM: Decode `lw` instruction

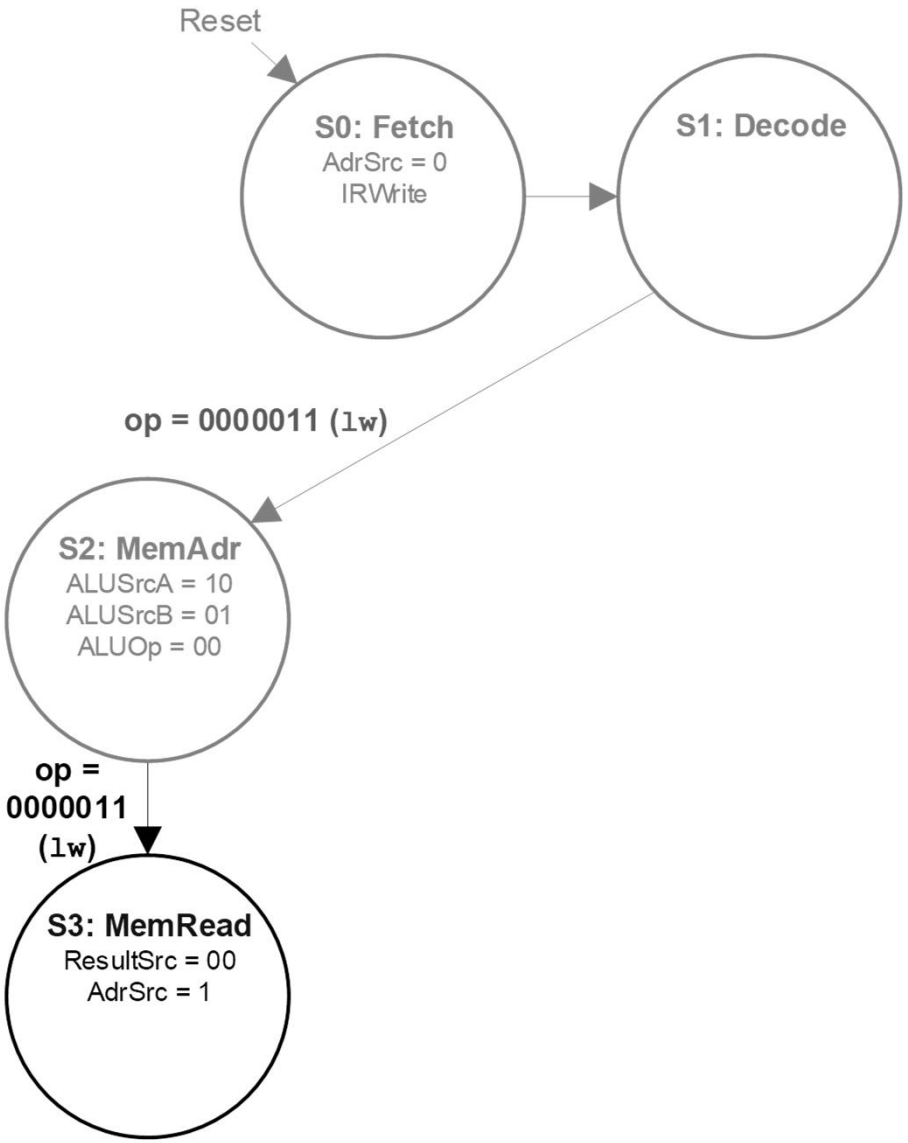


Main FSM: Address

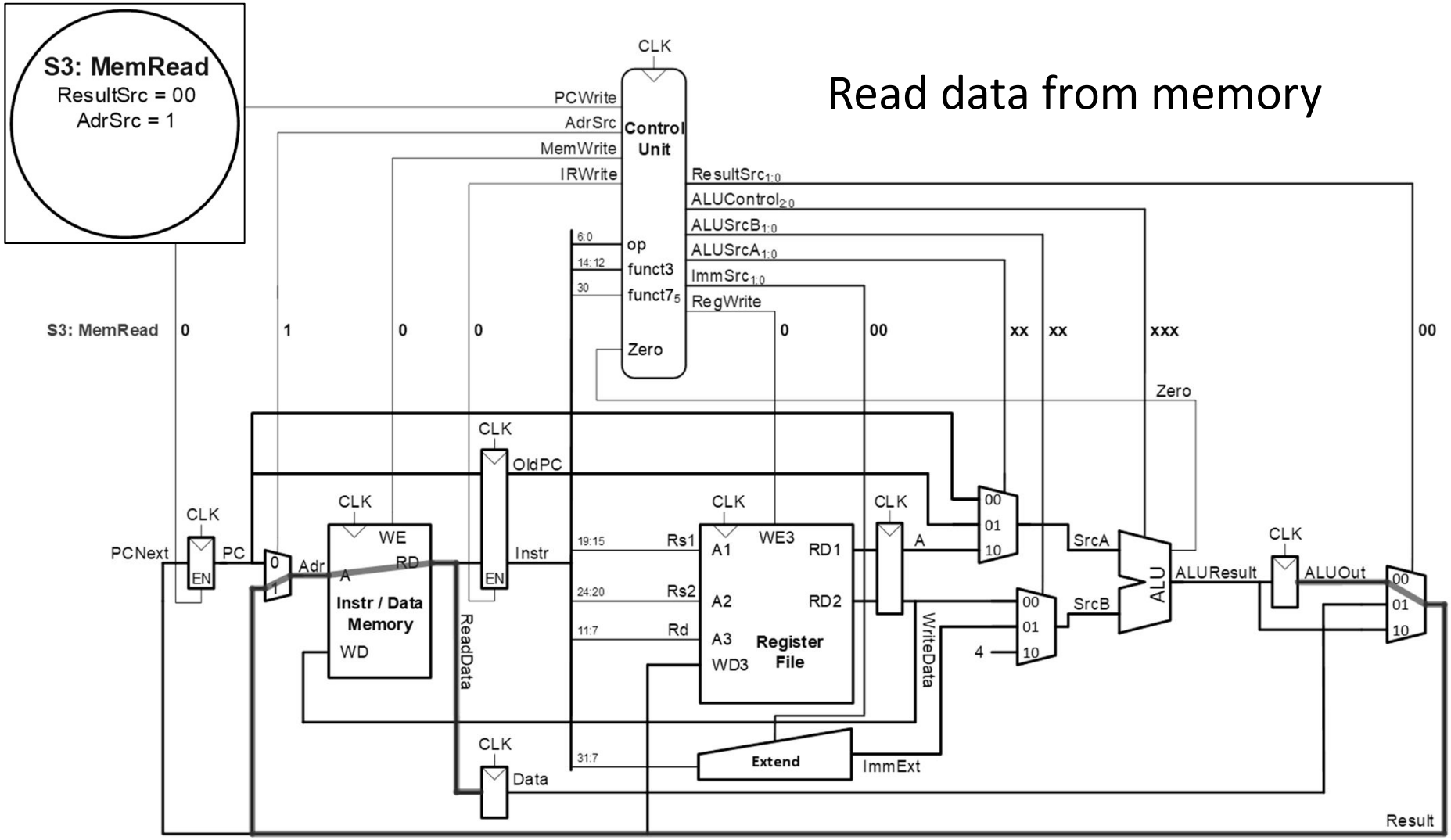


Compute the memory address

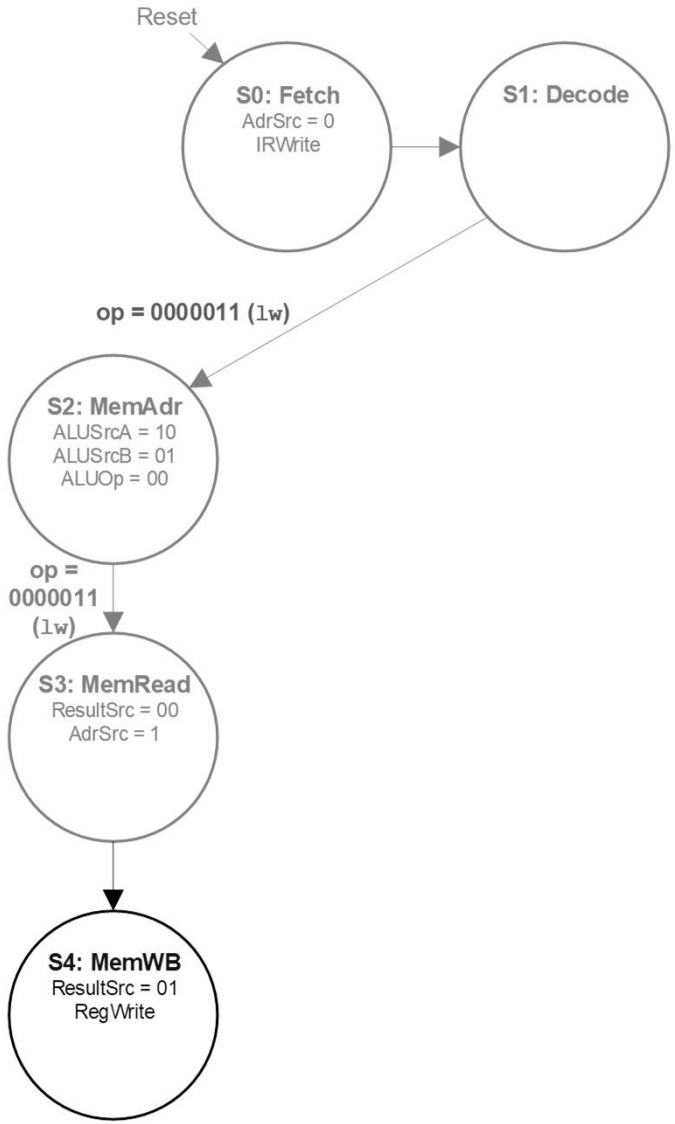
Main FSM: Read Memory



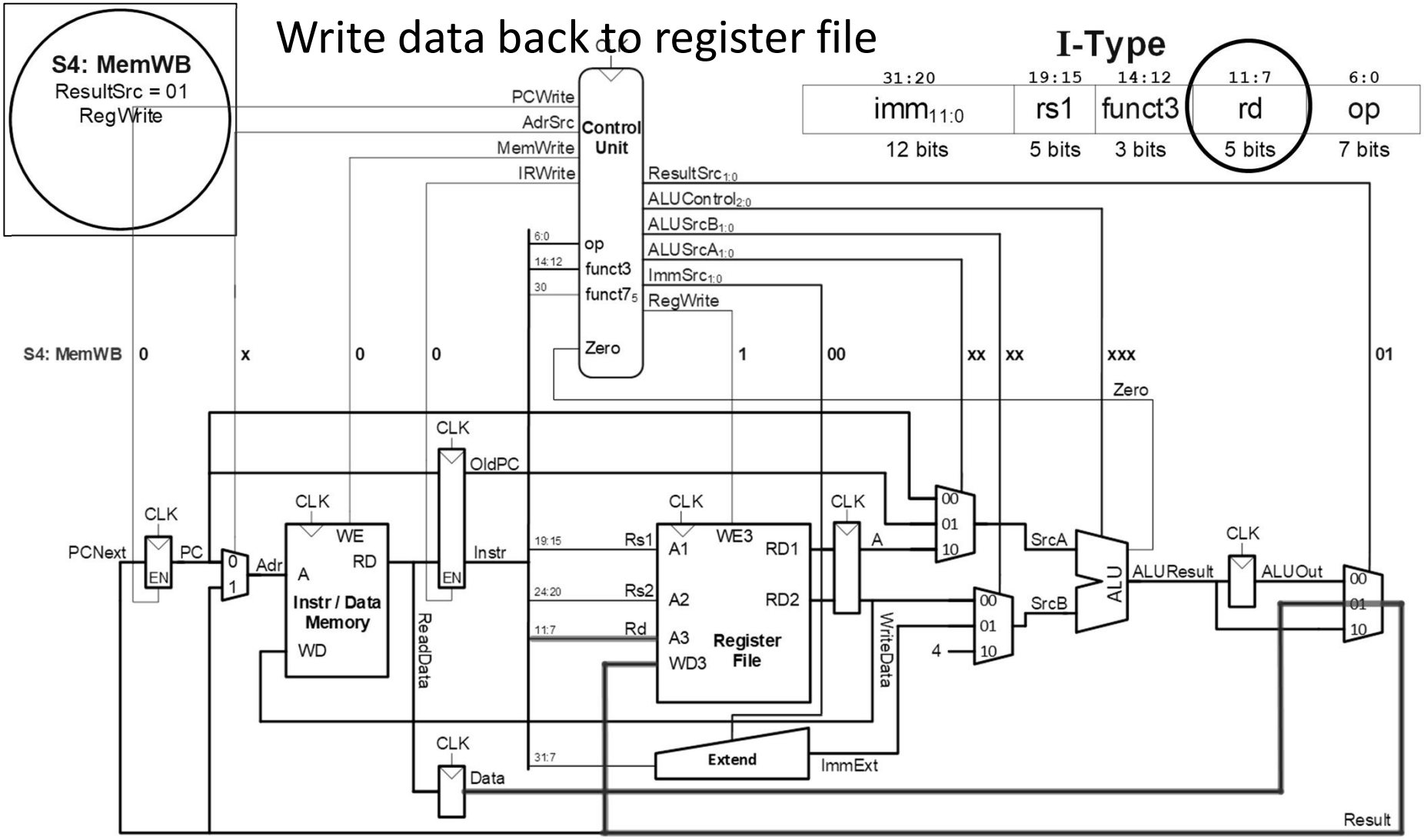
Main FSM: Read Memory Datapath



Main FSM: Write RF

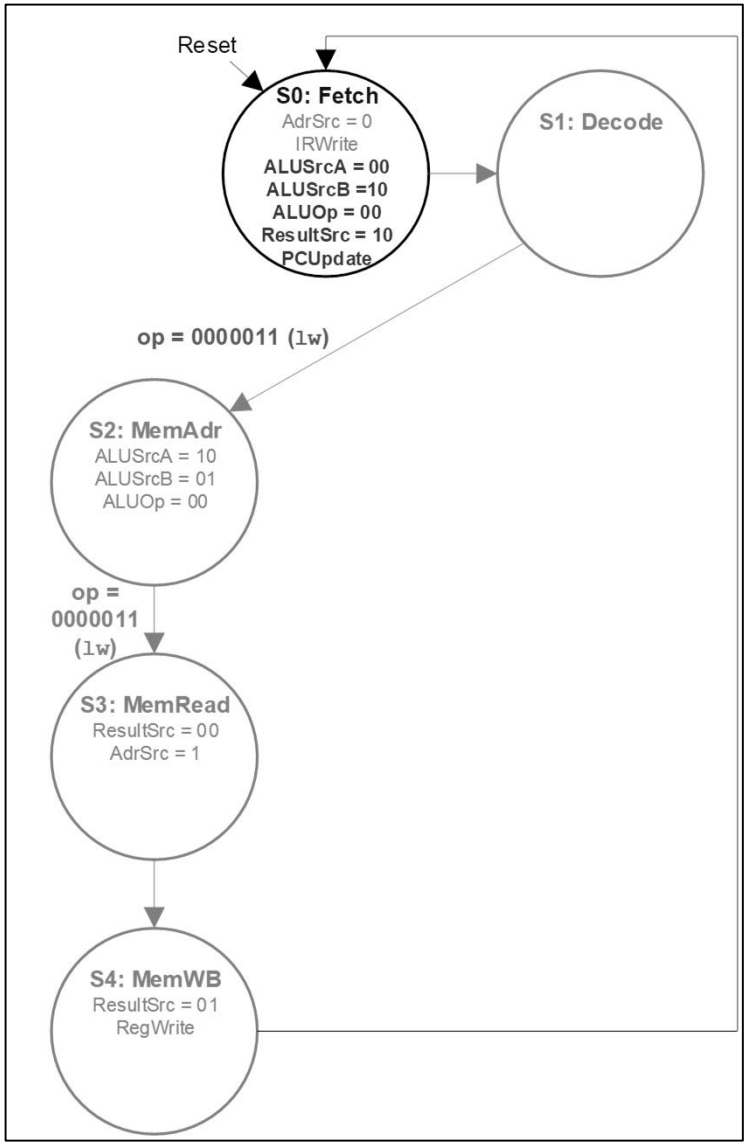


Main FSM: Write RF Datapath

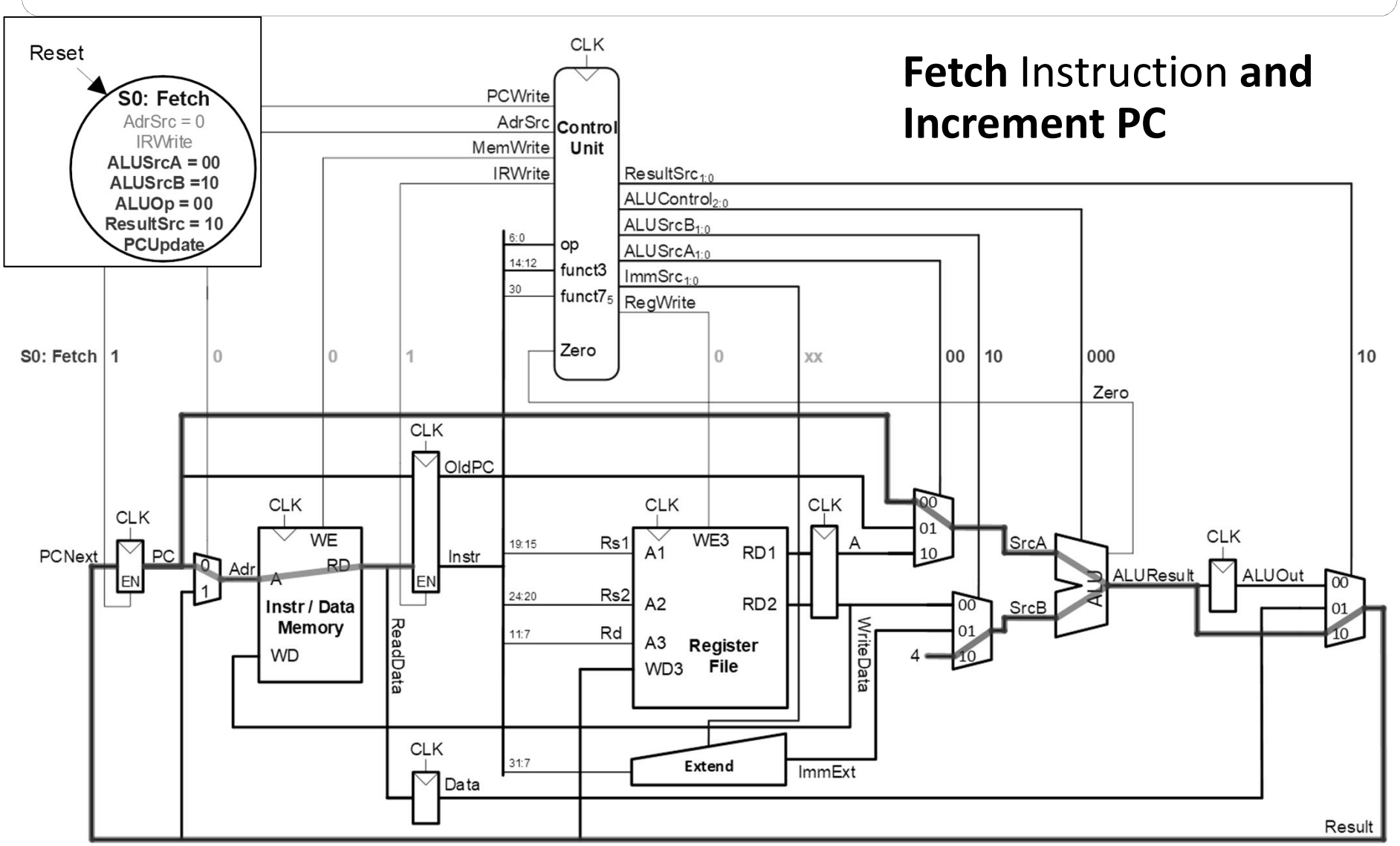


Main FSM: Fetch Revisited

Calculate **PC+4** during Fetch stage (ALU isn't being used for other purposes)

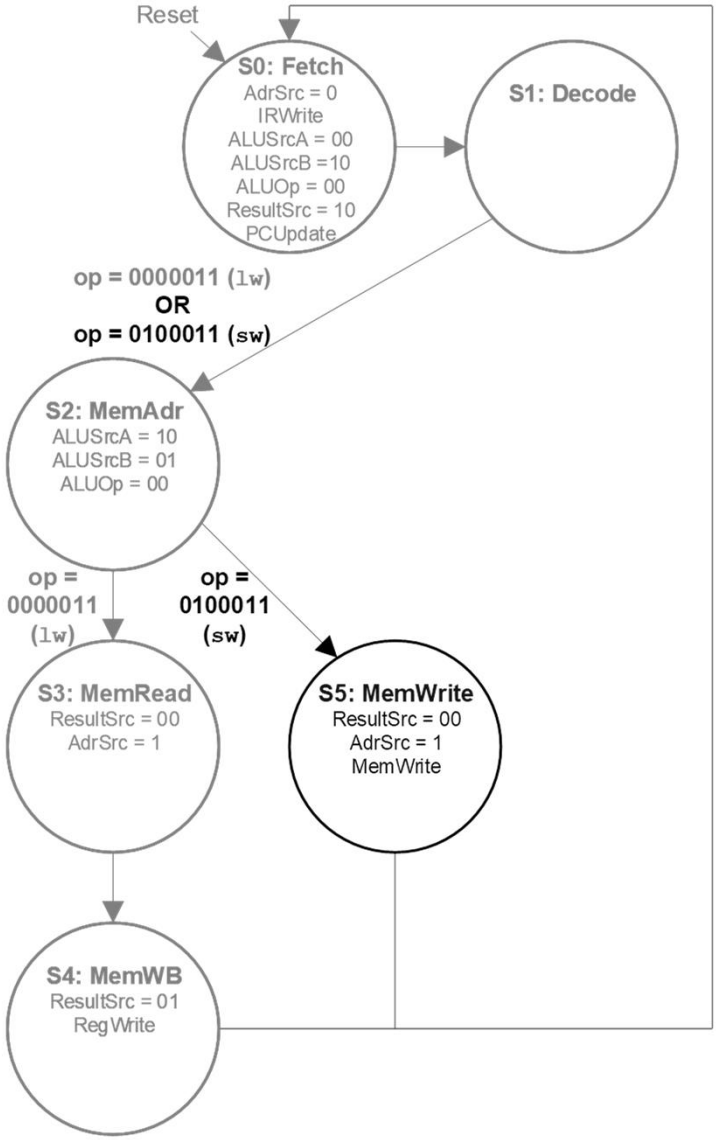


Main FSM: Fetch (PC+4) Datapath

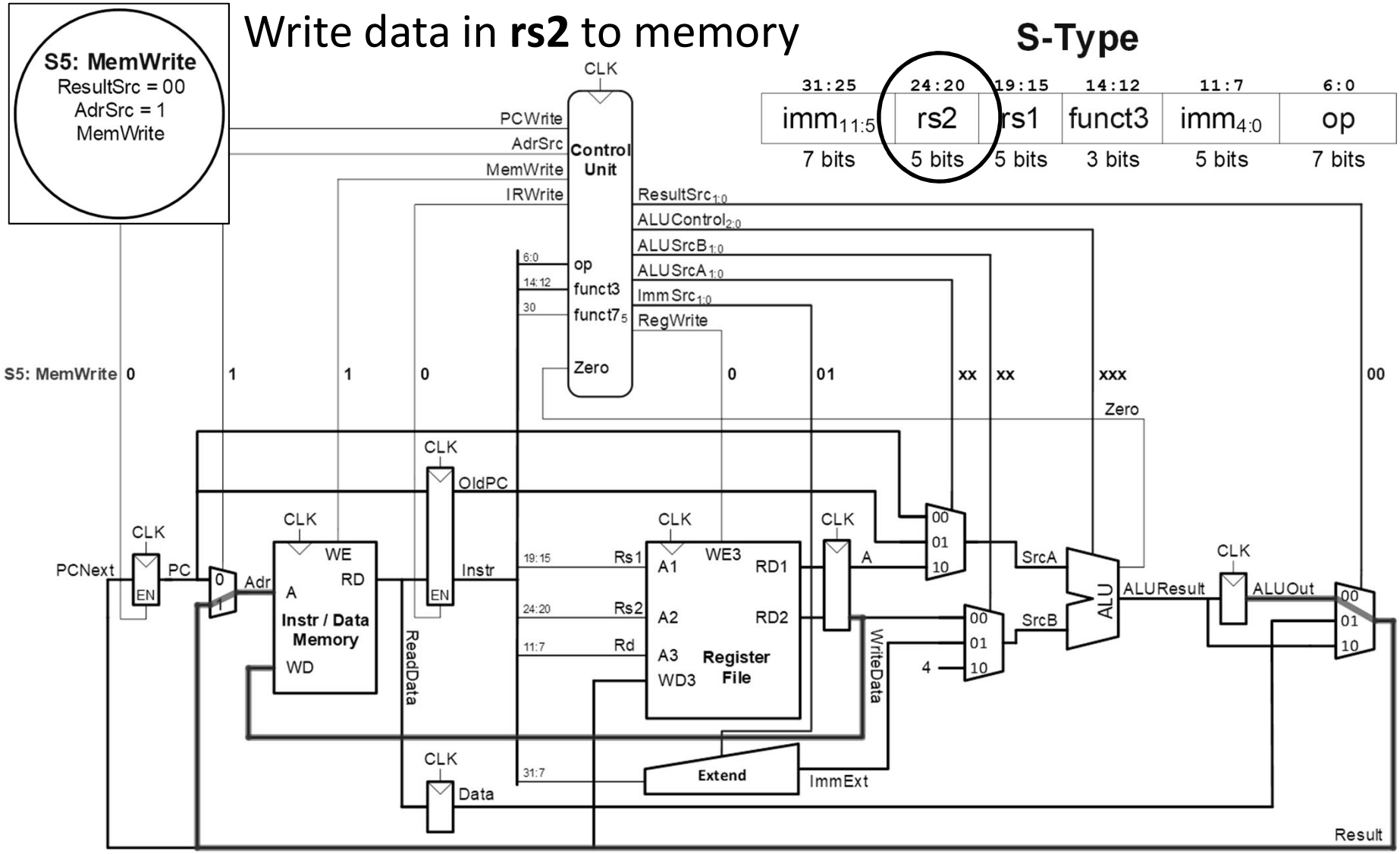


Multicycle Control: Other Instructions

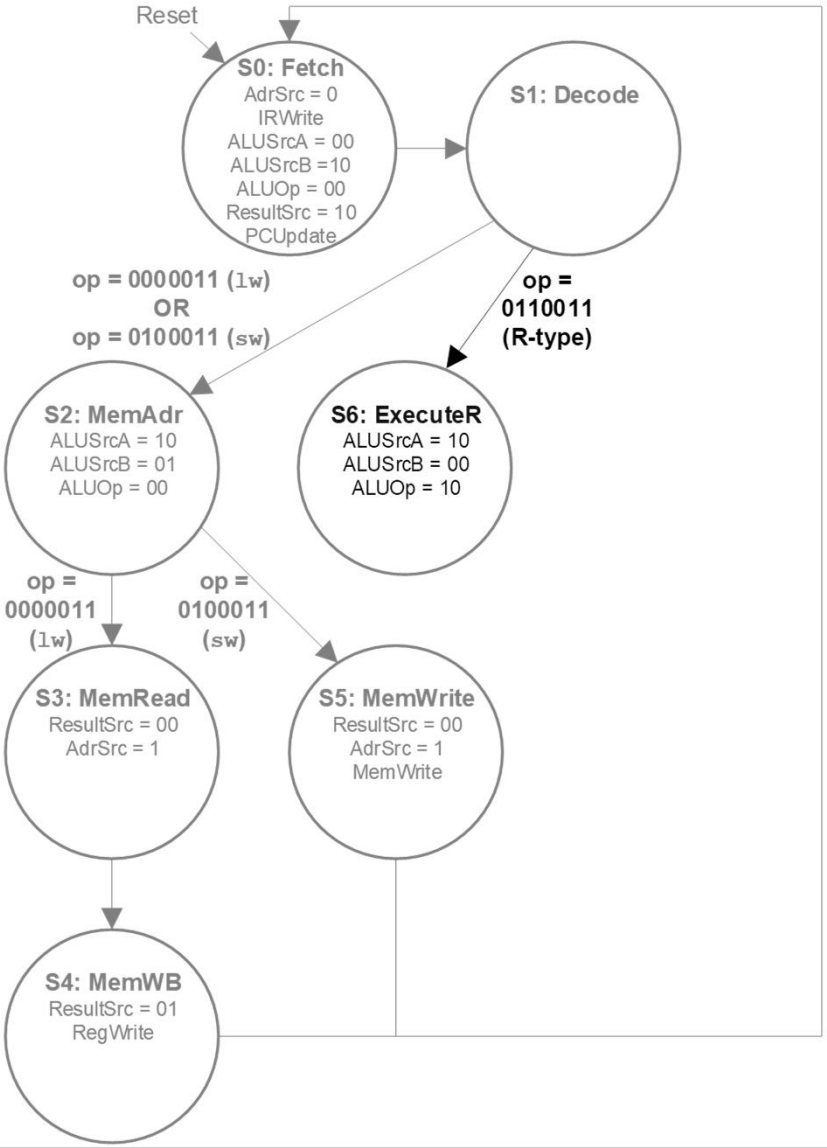
Main FSM: `sw`


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

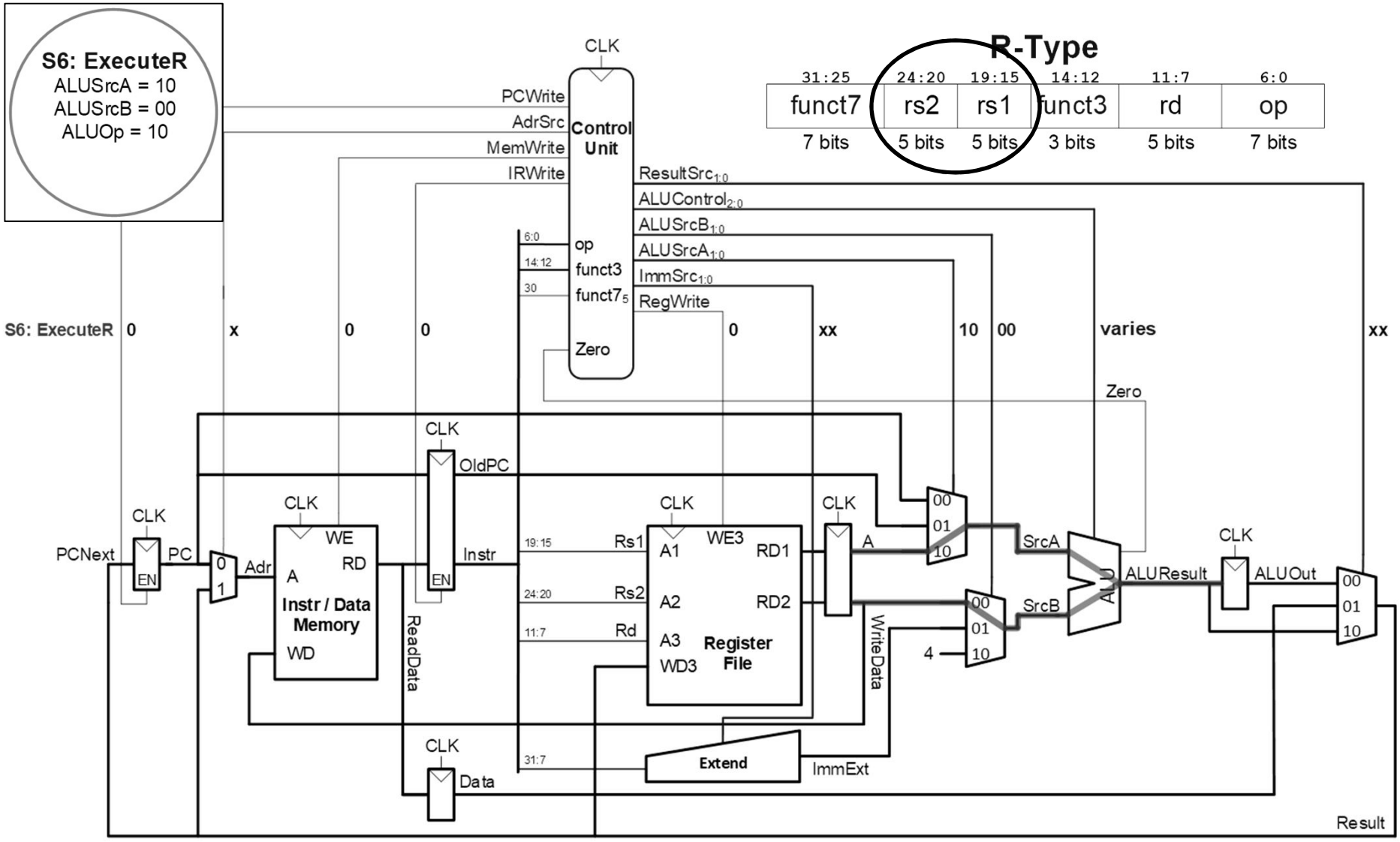


Main FSM: R-Type Execute

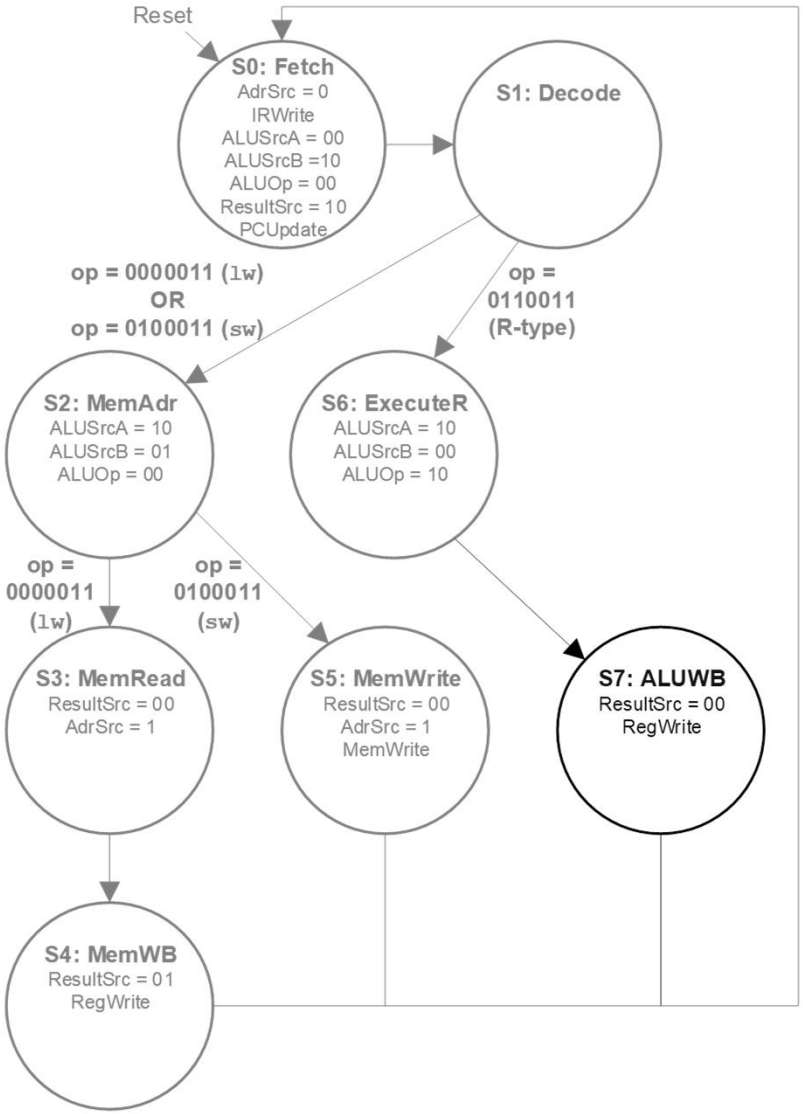


- **R-type ALU instructions:**
  `add`, `sub`, `and`, `or`, `slt`

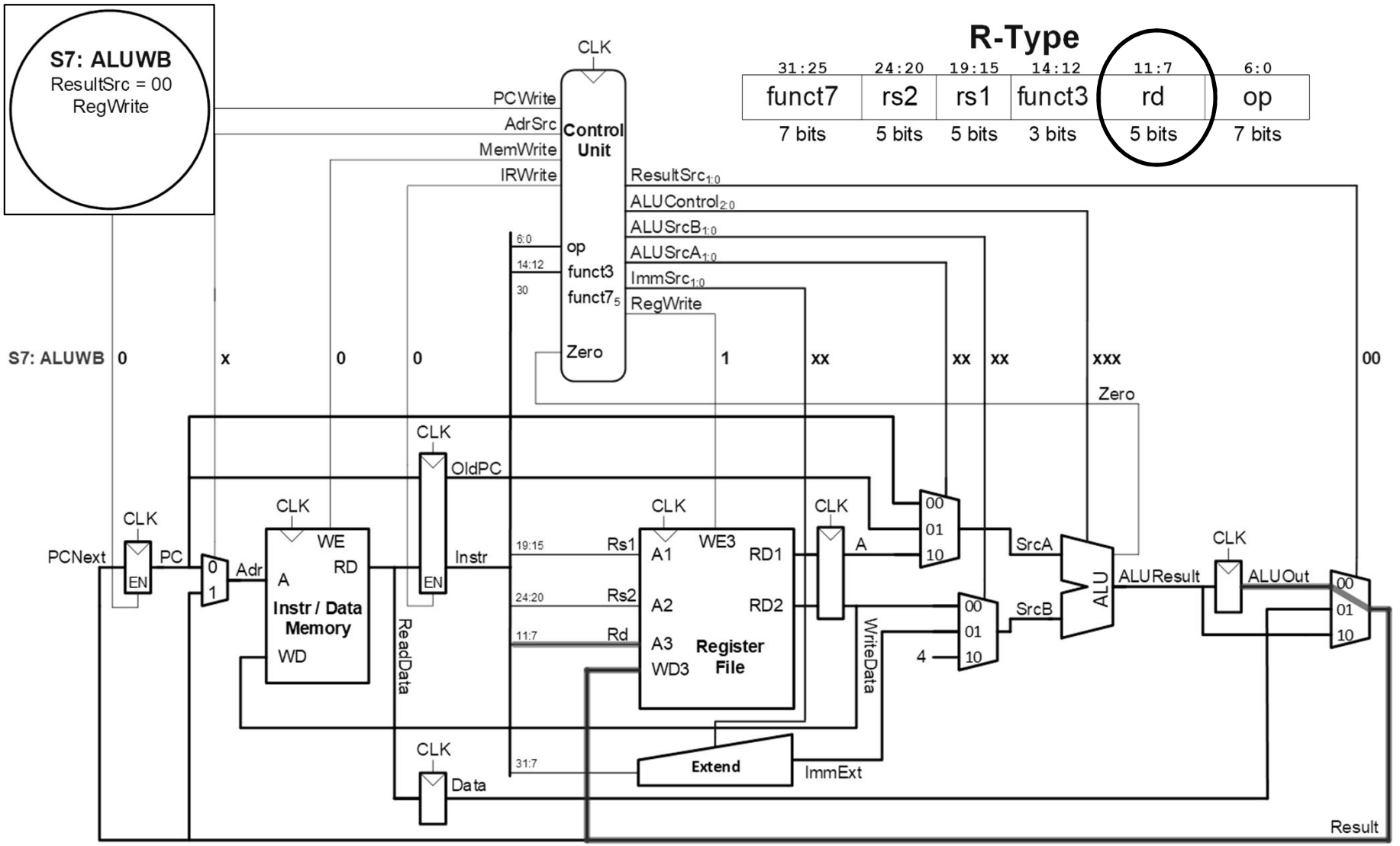
Main FSM: R-Type Execute Datapath



Main FSM: R-Type ALU Write Back



Main FSM: R-Type ALU Write Back



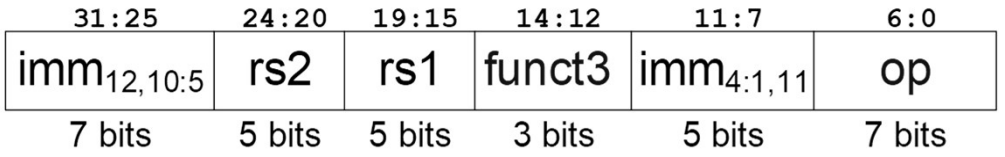
Main FSM: `beq`

- **Need to calculate:** Branch Target Address:
  Calculate branch target address = PC + imm
  - Branch Target Address (BTA = PC + imm)
  - **rs1** - **rs2** (to see if equal)

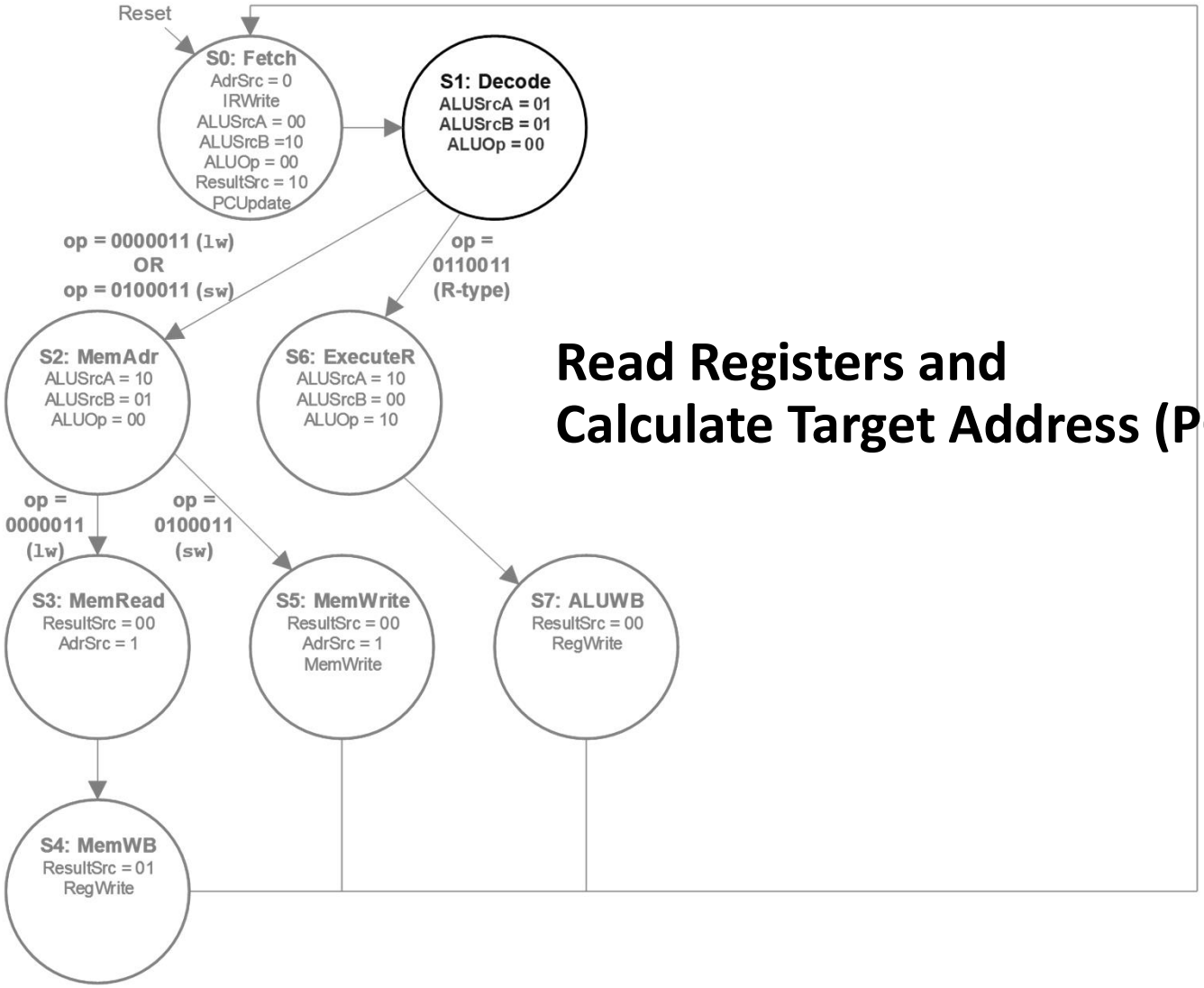
- **ALU** isn't being used in Decode stage
  - Use it to calculate Target Address (PC + imm)

- PC is updated in Fetch stage, so need to save **old (current) PC**

B-Type

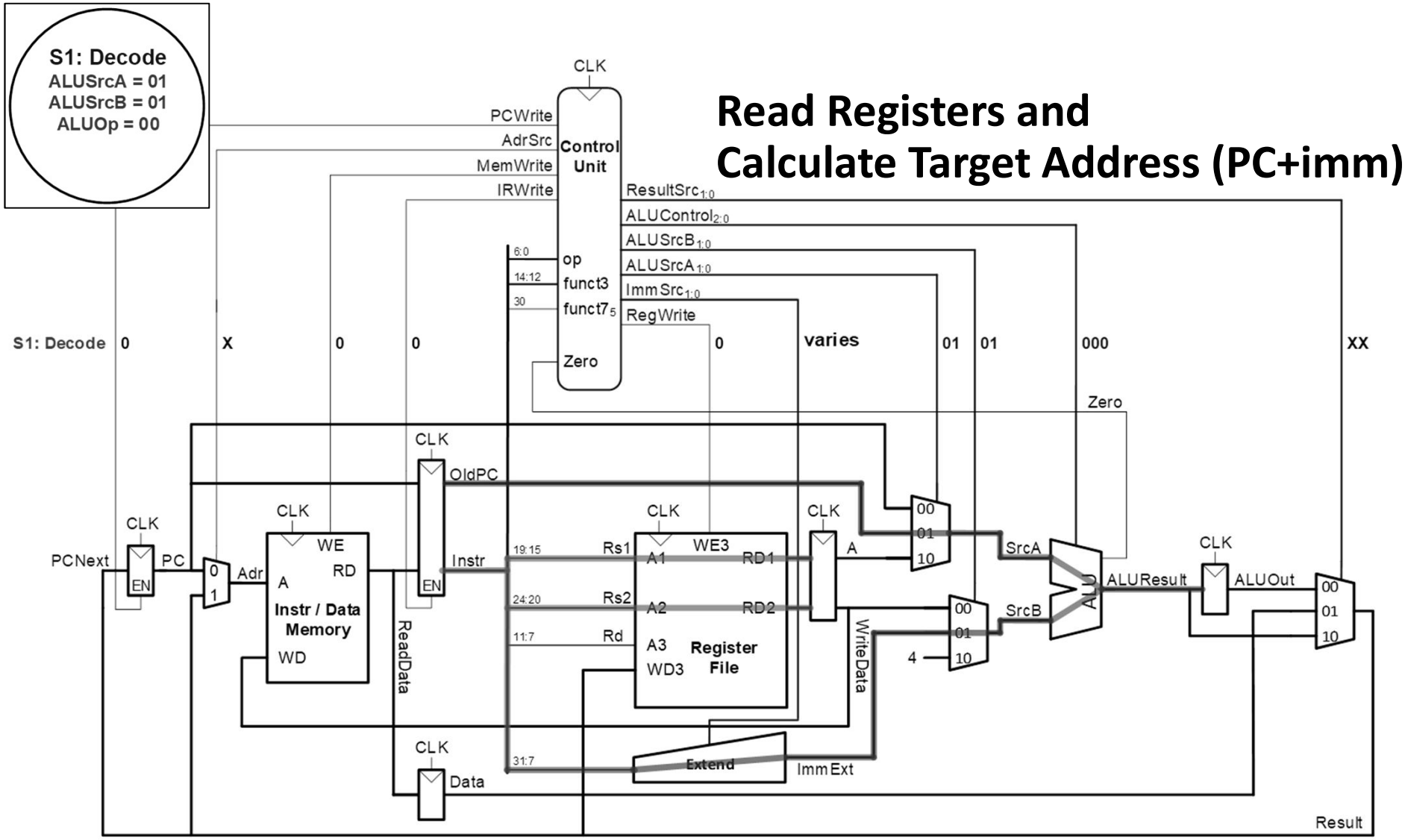


Main FSM: Decode Revisited

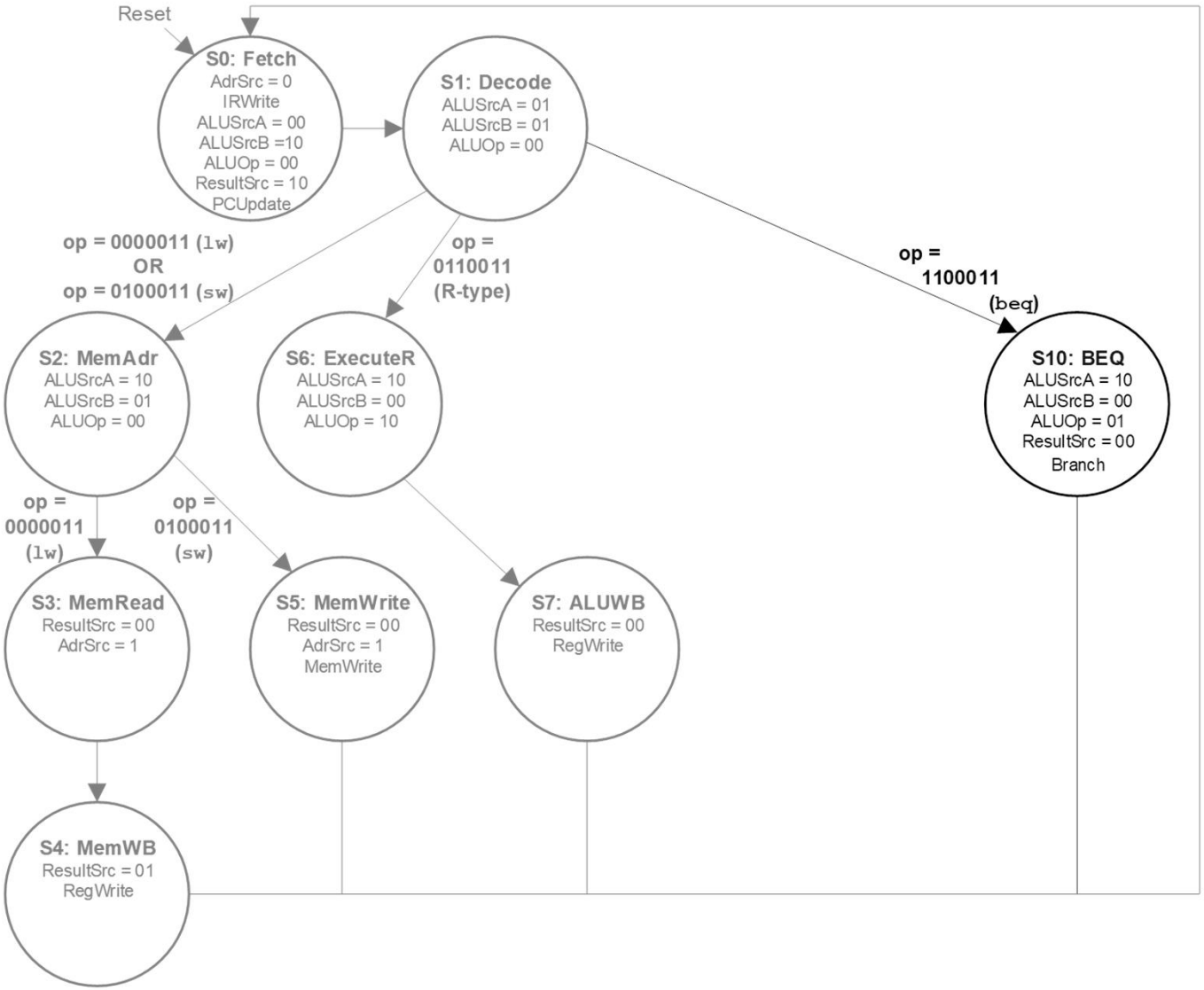


Read Registers and
Calculate Target Address (PC+imm)

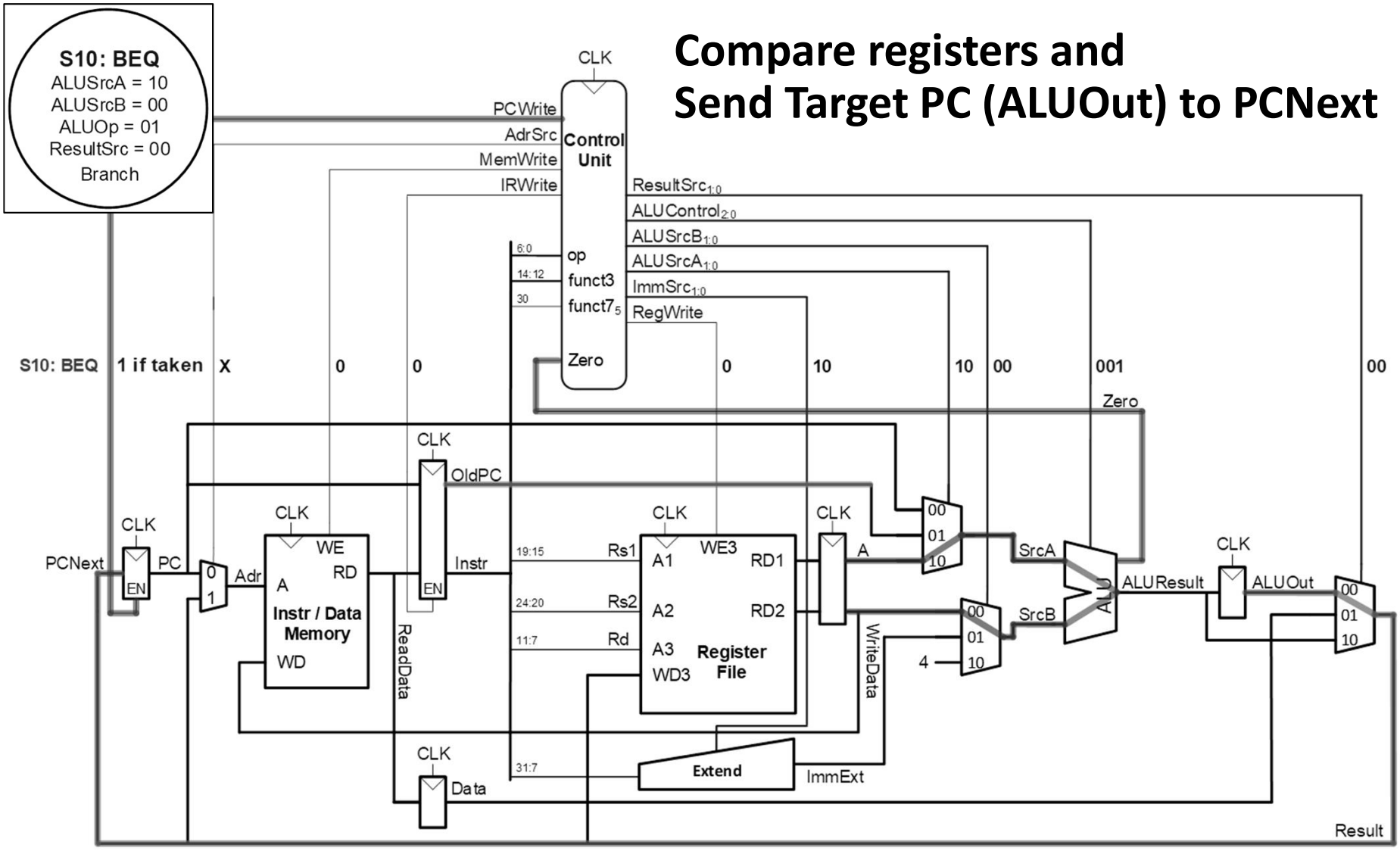
Main FSM: Decode (Target Address)



Main FSM: beq

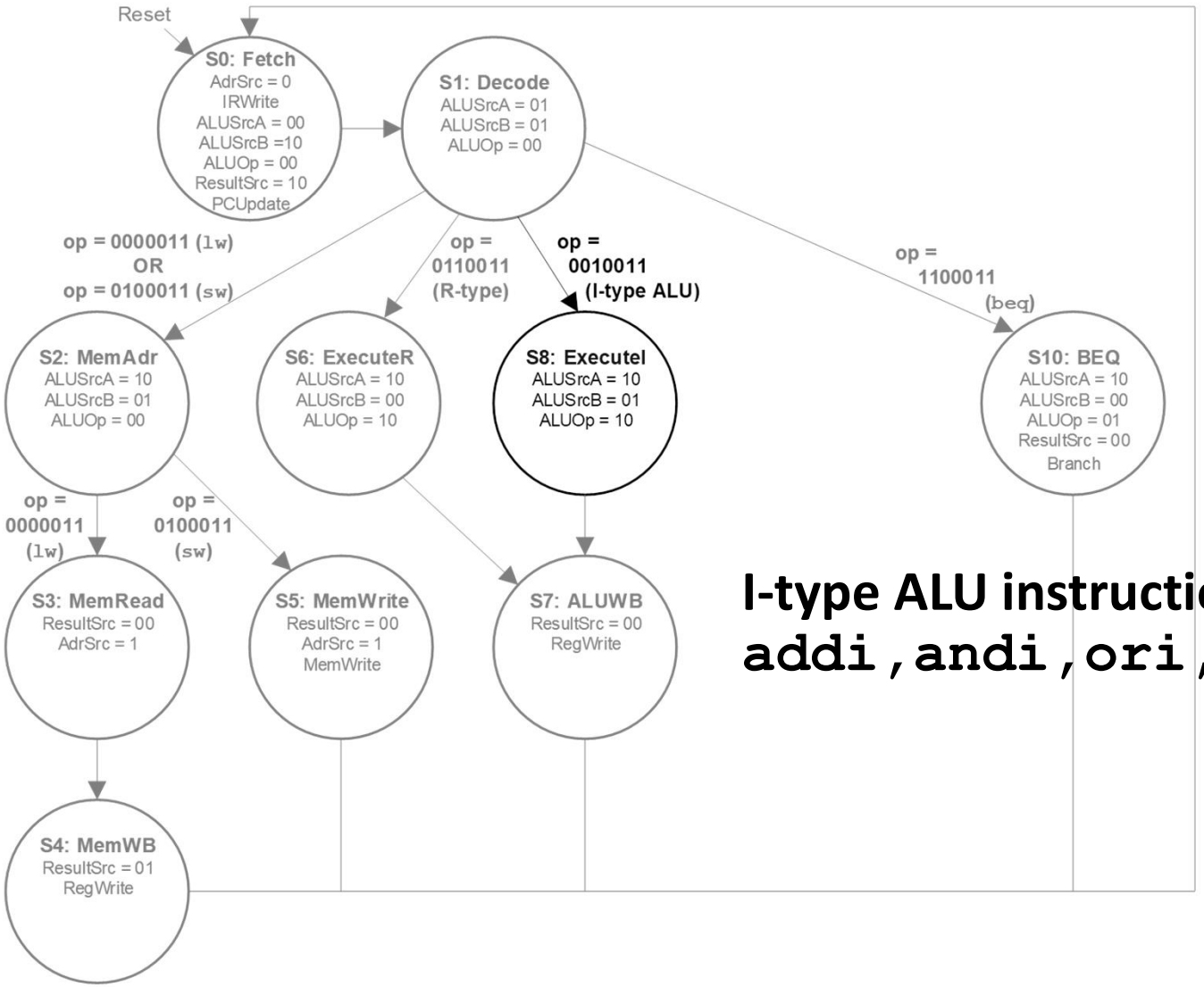

Reset
S0: Fetch
AdrSrc = 0
IRWrite
ALUSrcA = 00
ALUSrcB =10
ALUOp = 00
ResultSrc = 10
PCUpdate
S1: Decode
ALUSrcA = 01
ALUSrcB = 01
ALUOp = 00
op = 0000011 (lw)
OR
op = 0100011 (sw)
op = 0110011 (R-type)
op = 1100011 (beq)
S2: MemAdr
ALUSrcA = 10
ALUSrcB = 01
ALUOp = 00
S6: ExecuteR
ALUSrcA = 10
ALUSrcB = 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



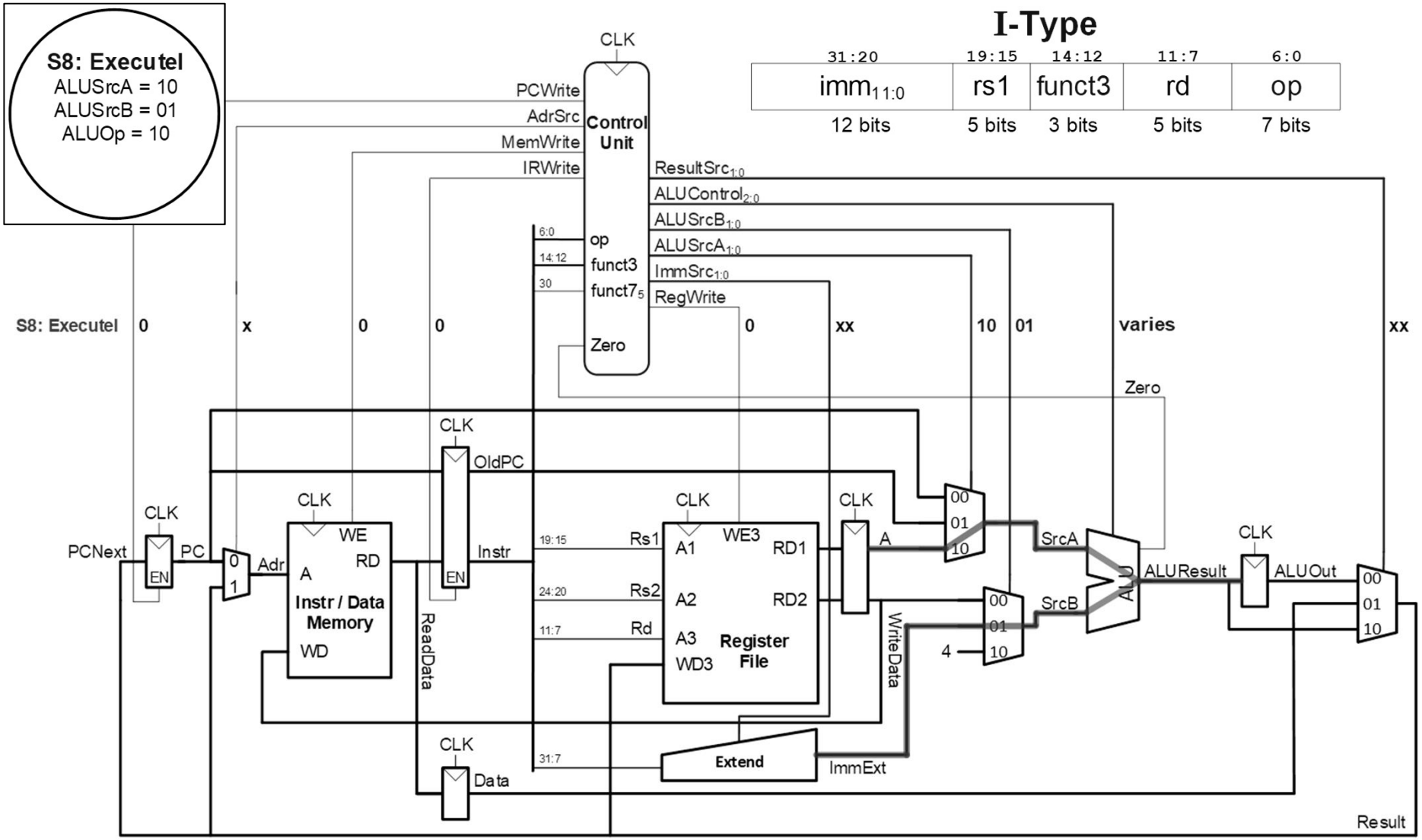
Extending the RISC-V Multicycle Processor

Main FSM: I-Type ALU Execute

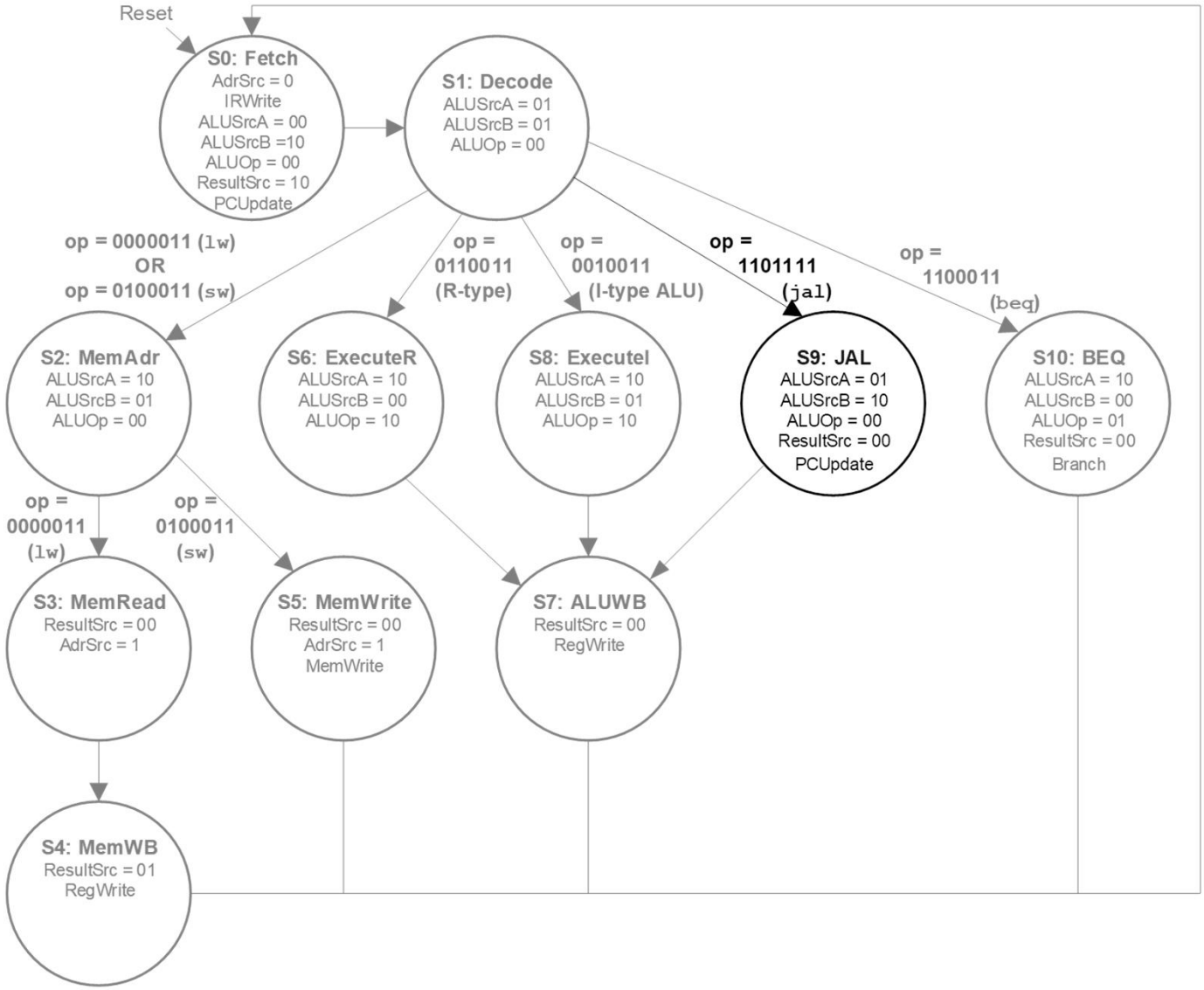


I-type ALU instructions:
`addi, andi, ori, xori`

Main FSM: I-Type ALU Exec. Datapath

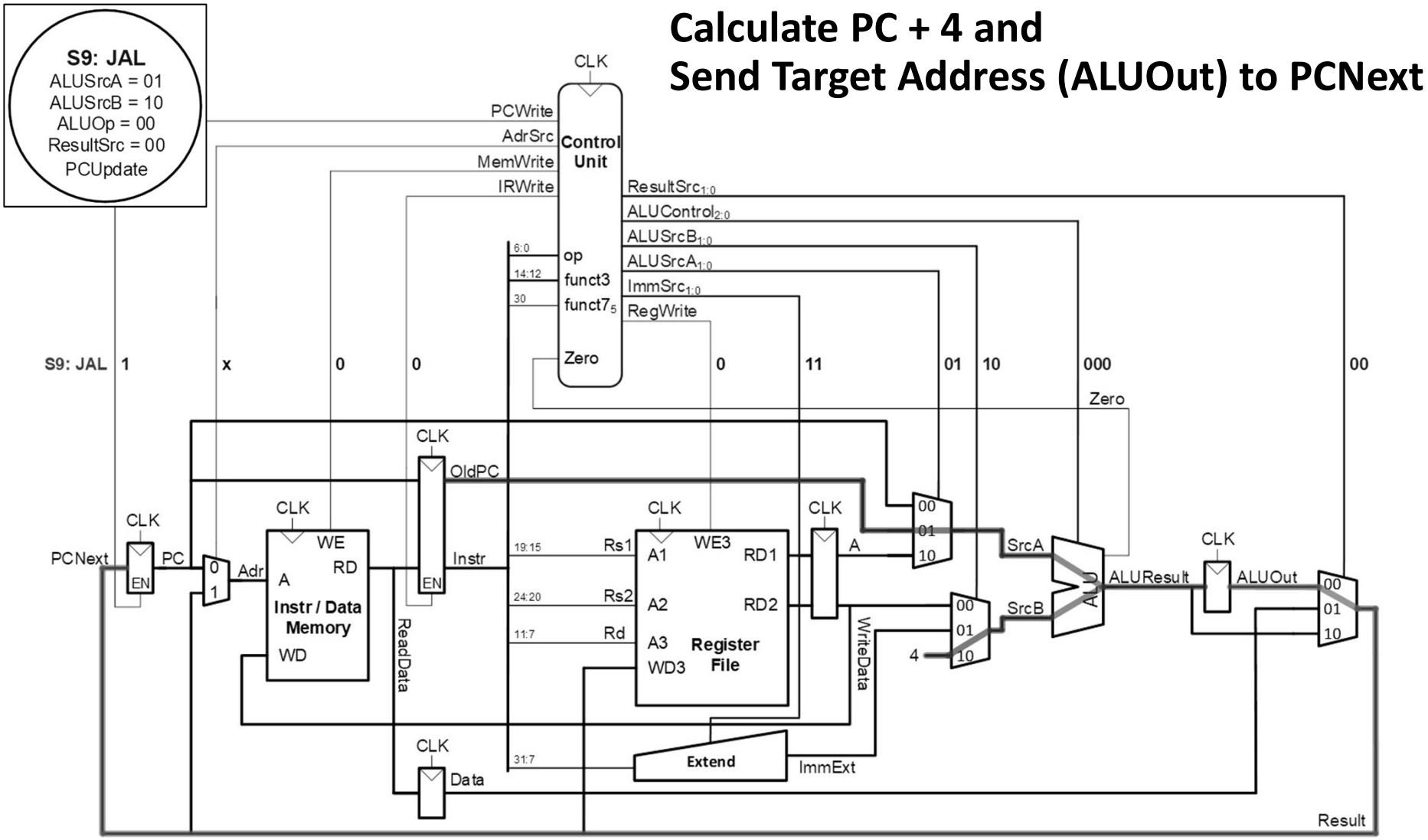


Main FSM: `jal`

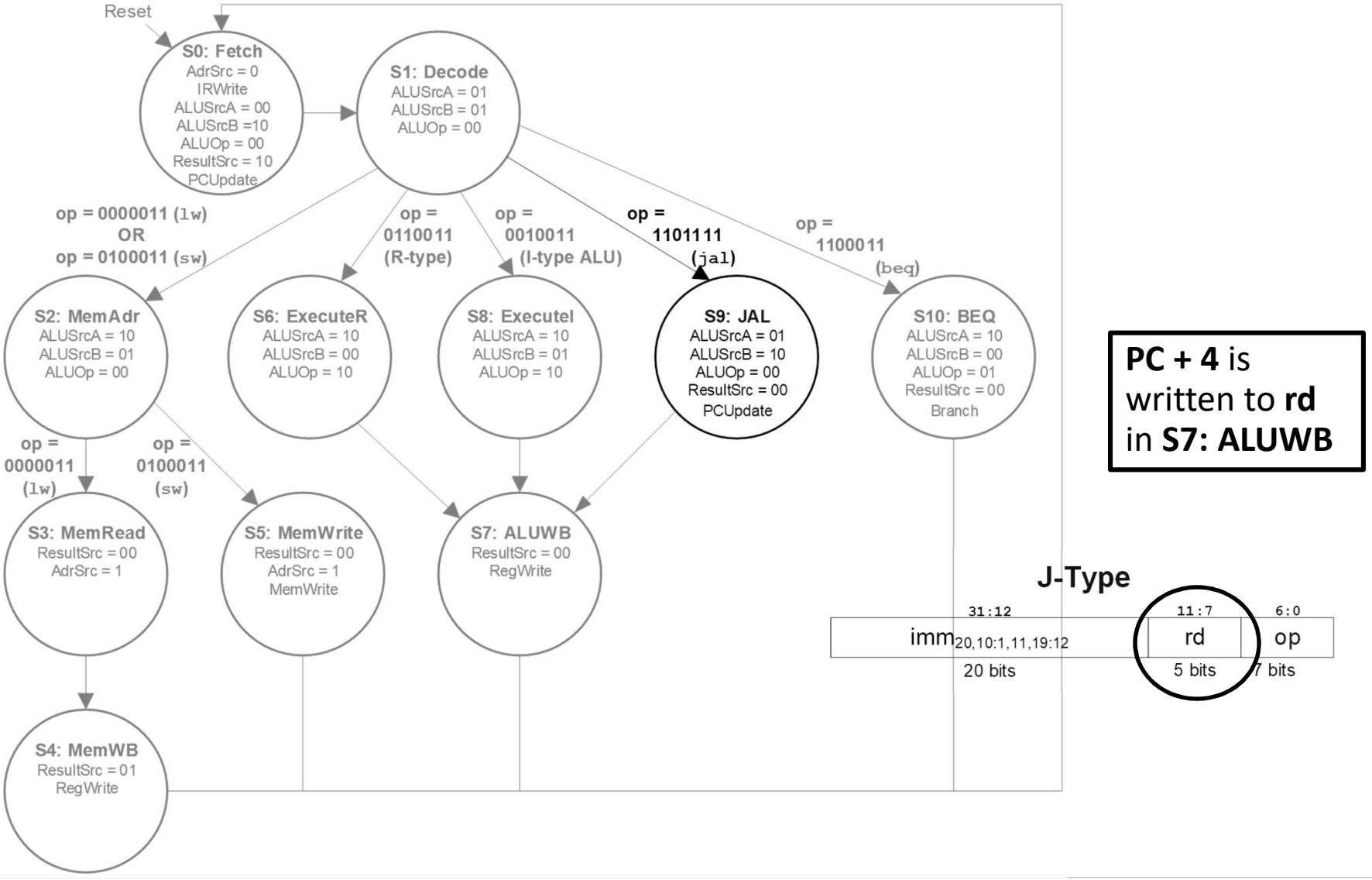

Reset
S0: Fetch
AdrSrc = 0
IRWrite
ALUSrcA = 00
ALUSrcB =10
ALUOp = 00
ResultSrc = 10
PCUpdate
S1: Decode
ALUSrcA = 01
ALUSrcB = 01
ALUOp = 00
op = 0000011 (lw)
OR
op = 0100011 (sw)
op = 0110011 (R-type)
op = 0010011 (I-type ALU)
op = 1101111 (jal)
op = 1100011 (beq)
S2: MemAdr
ALUSrcA = 10
ALUSrcB = 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



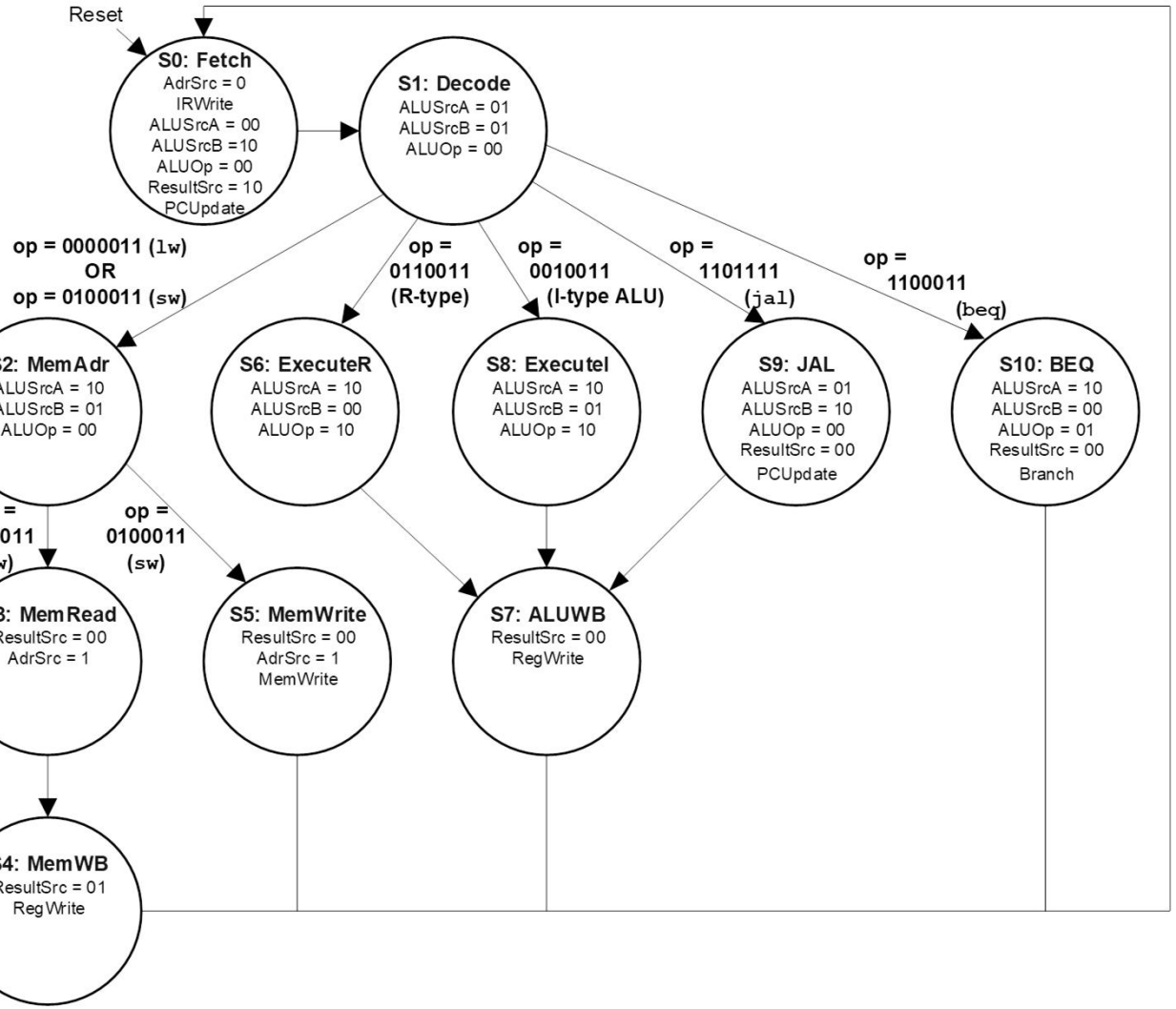
Main FSM: `jal`



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



Performance

Processor Performance

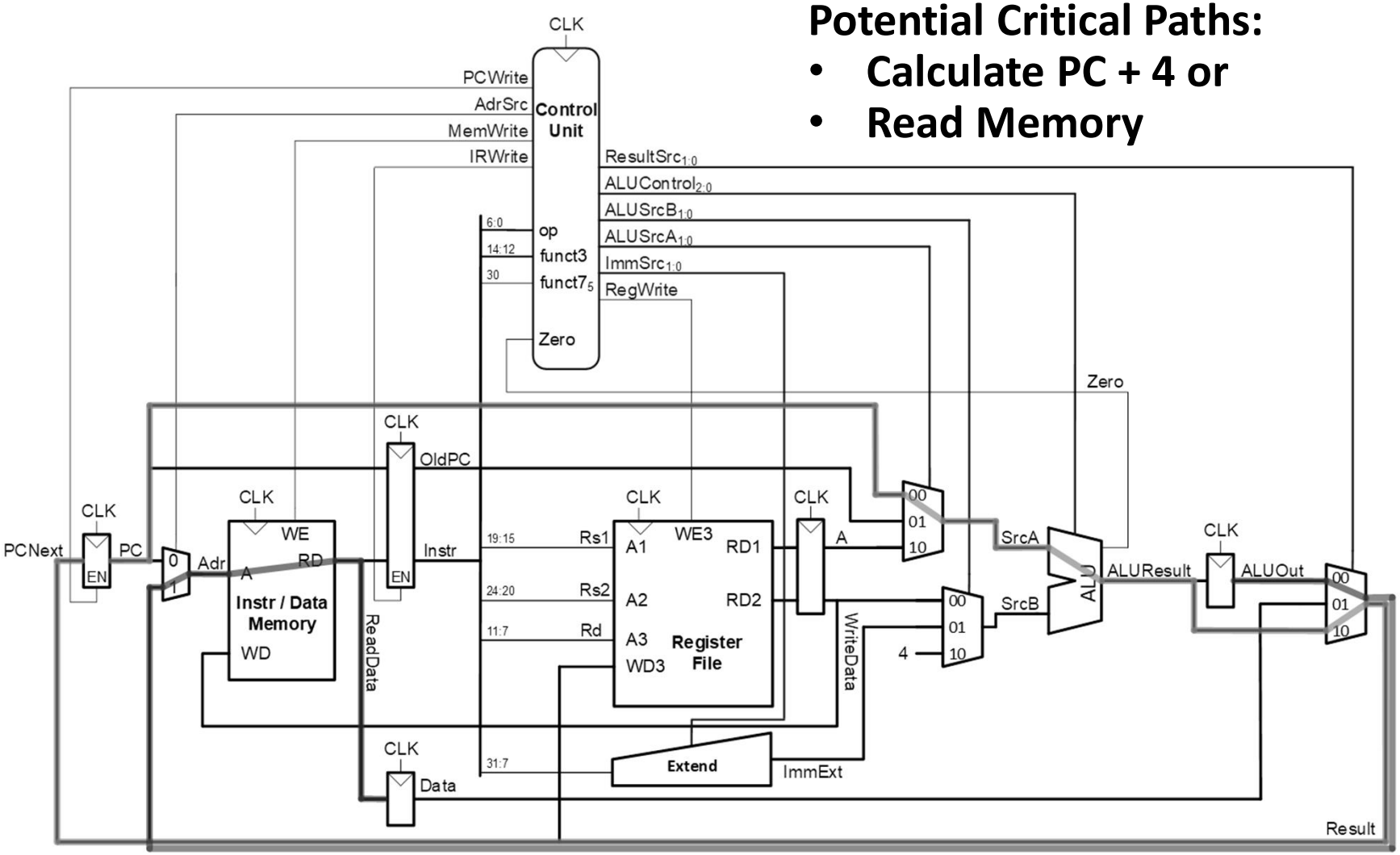
- Instructions take different number of cycles:
  - 3 cycles: `beq`
  - 4 cycles: R-type, `addi`, `sw`, `jal`
  - 5 cycles: `lw`
- CPI is weighted average
- SPECINT2000 benchmark:
  - **25%** loads
  - **10%** stores
  - **13%** branches
  - **52%** R-type

Average CPI = (0.13)(3) + (0.52 + 0.10)(4) + (0.25)(5) = 4.12

Critical Path

Potential Critical Paths:

- **Calculate PC + 4 or**
- **Read Memory**



Processor Performance

Multicycle critical path:

- **Assumptions:**
  - RF is faster than memory
  - Writing memory is faster than reading memory

$$T_{c\_multi} = t_{pcq} + t_{\text{dec}} + 2t_{\text{mux}} + \max(t_{\text{ALU}}, t_{\text{mem}}) + t_{\text{setup}}$$

Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	30
AND-OR gate	$t_{AND\text{-}OR}$	20
ALU	t_{ALU}	120
Decoder (Control Unit)	t_{dec}	25
Extend unit	t_{dec}	35
Memory read	t_{mem}	200
Register file read	t_{RFread}	100
Register file setup	$t_{RFsetup}$	60

$$T_{c\_multi} = t_{pcq} + t_{dec} + 2t_{mux} + \max(t_{ALU}, t_{mem}) + t_{setup}$$

$$= \boxed{}$$

Performance Example

For a program with **100 billion** instructions executing on a **multicycle** RISC-V processor

- **CPI** = 4.12 cycles/instruction
- **Clock cycle time:** T_{c_multi} = 375 ps

Execution Time = (# instructions) × CPI × T_c

.