Multi-cycle MIPS Processor

Design of Digital Circuits 2014
Srdjan Capkun
Frank K. Gürkaynak

http://www.syssec.ethz.ch/education/Digitaltechnik_14

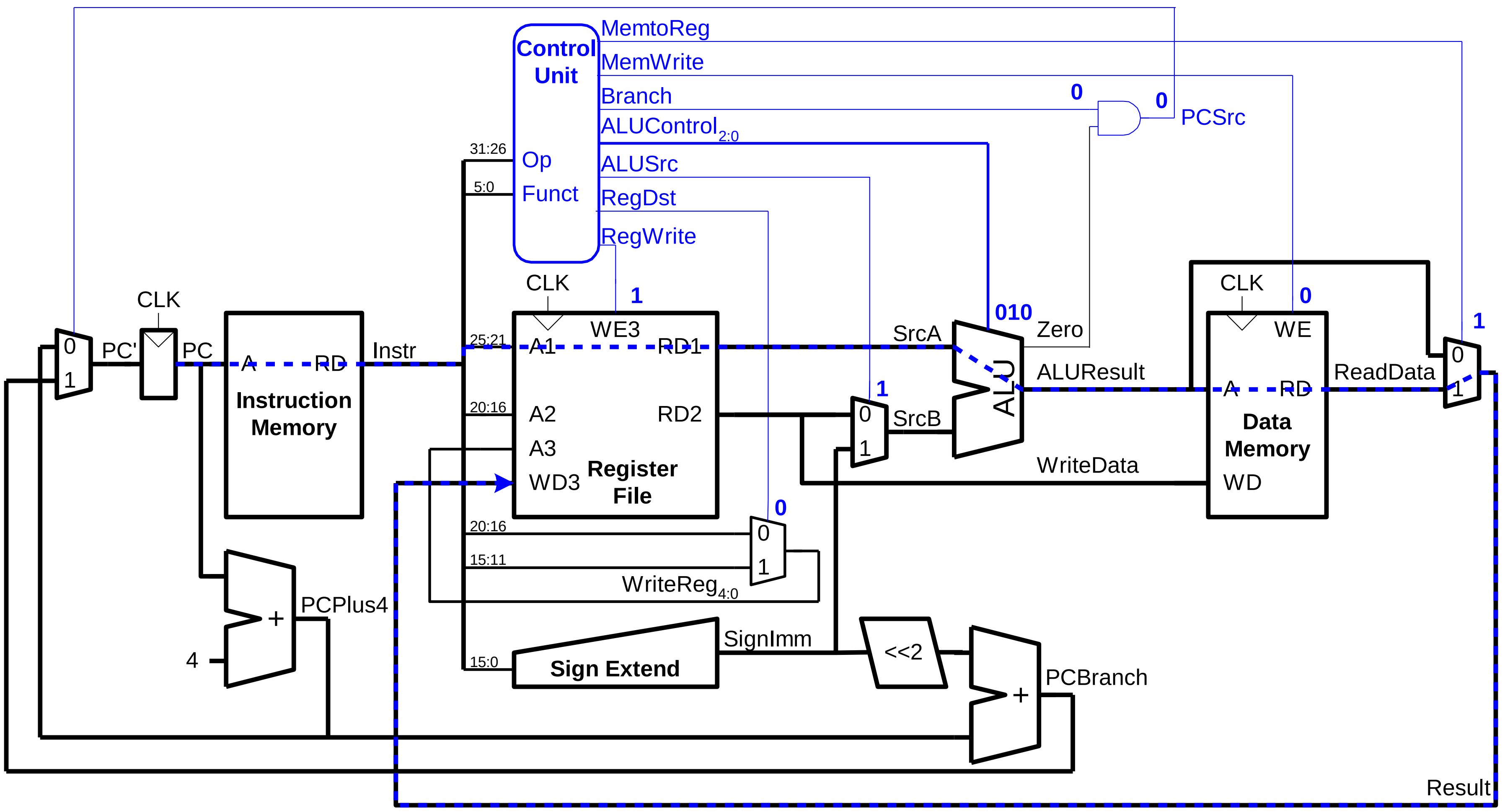
What Will We Learn?

- What are the problems of Single-cycle Processor
- Multi-cycle Architecture for the MIPS
- Determine the performance of Multi-cycle Processor

Single-cycle MIPS Processor

- **Single-cycle microarchitecture:**
  - + simple
  - - cycle time limited by longest instruction (`lw`)
  - - two adders/ALUs and two memories

Single-cycle Performance



Delay of Individual Components (Example)

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	***200***
Memory read	t_{mem}	***250***
Register file read	t_{RFread}	***150***
Register file setup	$t_{RFsetup}$	20

Processor Performance

- **How fast is my program?**
  - Every program consists of a series of instructions
  - Each instruction needs to be executed.
- **So how fast are my instructions ?**
  - Instructions are realized on the hardware
  - They can take one or more clock cycles to complete
  - ***Cycles per Instruction = CPI***
- **How much time is one clock cycle?**
  - The critical path determines how much time one cycle requires = ***clock period***.
  - 1/clock period = ***clock frequency*** = how many cycles can be done each second.

Processor Performance

- **Now as a general formula**
  - Our program consists of executing **N** instructions.
  - Our processor needs **CPI** cycles for each instruction.
  - The maximum clock speed of the processor is **f**,
    and the clock period is therefore **T**=1/f

- **Our program will execute in**

$$N \times CPI \times (1/f) = N \times CPI \times T \text{ seconds}$$

How can I Make the Program Run Faster?

N x CPI x (1/f)

- **Reduce the number of instructions**
  - Make instructions that ‘do’ more (CISC)
  - Use better compilers
- **Use less cycles to perform the instruction**
  - Simpler instructions (RISC)
  - Use multiple units/ALUs/cores in parallel
- **Increase the clock frequency**
  - Find a ‘newer’ technology to manufacture
  - Redesign time critical components
  - Adopt pipelining

Multi-cycle MIPS Processor

- **Single-cycle microarchitecture:**
  - + simple
  - - cycle time limited by longest instruction (`lw`)
  - - two adders/ALUs and two memories

- **Multi-cycle microarchitecture:**
  - + higher clock speed
  - + simpler instructions run faster
  - + reuse expensive hardware on multiple cycles
  - - sequencing overhead paid many times

- **Same design steps: datapath & control**

What Do We Want To Optimize

- **Single Cycle Architecture uses two memories**
  - One memory stores instructions, the other data
  - We want to use a single memory (Smaller size)

What Do We Want To Optimize

- **Single Cycle Architecture uses two memories**
  - One memory stores instructions, the other data
  - We want to use a single memory (Smaller size)
- **Single Cycle Architecture needs three adders**
  - ALU, PC, Branch address calculation
  - We want to use the ALU for all operations (smaller size)

What Do We Want To Optimize

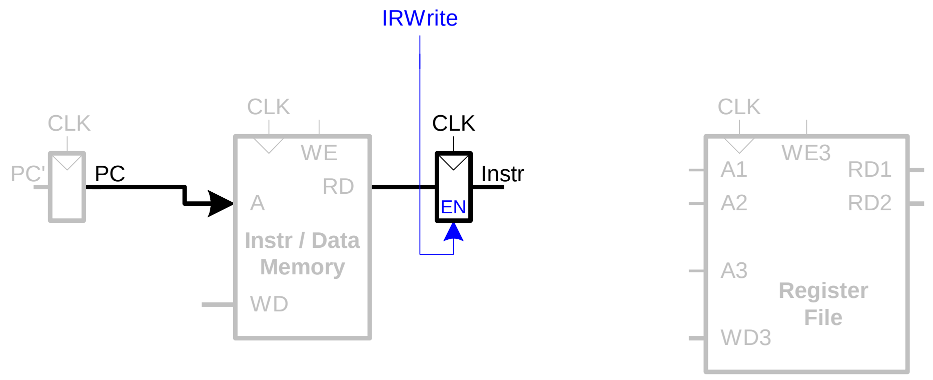
- **Single Cycle Architecture uses two memories**
  - One memory stores instructions, the other data
  - We want to use a single memory (Smaller size)
- **Single Cycle Architecture needs three adders**
  - ALU, PC, Branch address calculation
  - We want to use the ALU for all operations (smaller size)
- **In Single Cycle Architecture all instructions take one cycle**
  - The most complex operation slows down everything!
  - Divide all instructions into multiple steps
  - Simpler instructions can take fewer cycles (average case may be faster)

Consider lw instruction

- **For an instruction such as: `lw $t0, 0x20($t1)`**
- **We need to:**
  - Read the instruction from memory
  - Then read **`$t1`** from register array
  - Add the immediate value (**`0x20`**) to calculate the memory address
  - Read the content of this address
  - Write to the register **`$t0`** this content

Multi-cycle Datapath: instruction fetch

- **First consider executing lw**
  - STEP 1: Fetch instruction

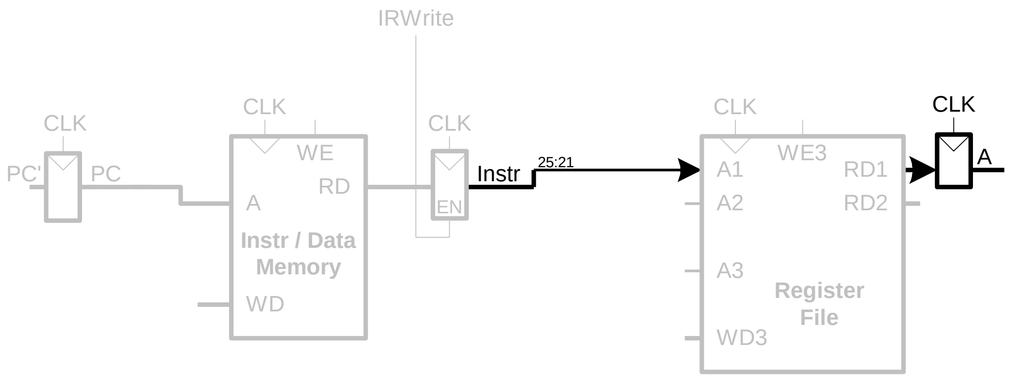


`read from the memory location [rs]+imm to location [rt]`

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

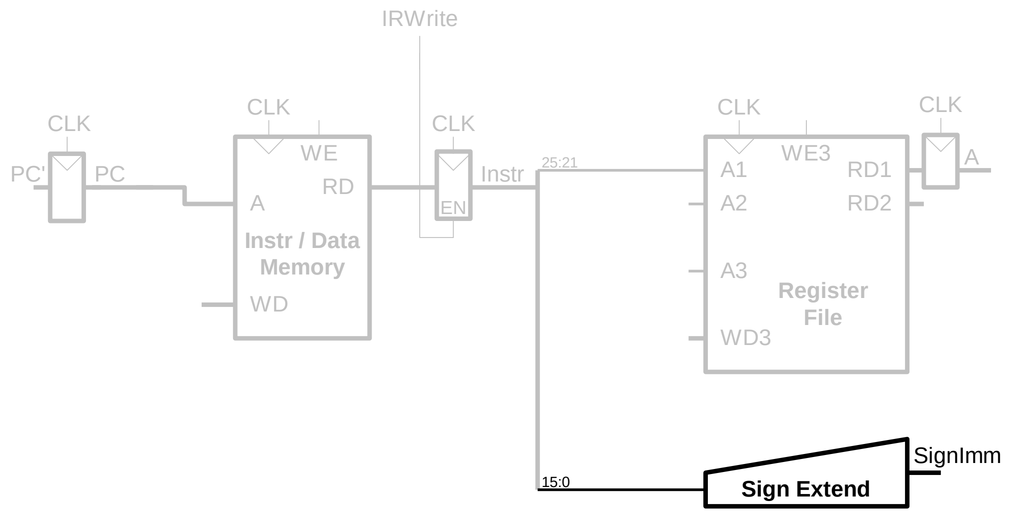
Multi-cycle Datapath: `lw` register read



I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

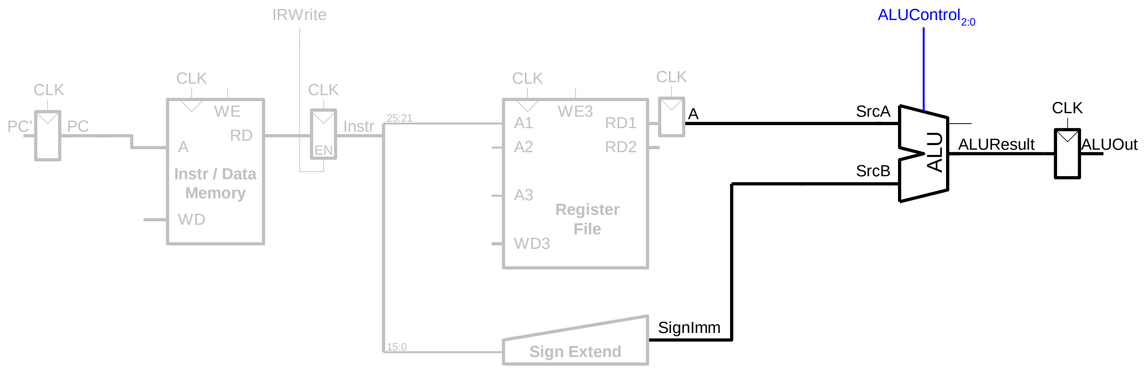
Multi-cycle Datapath: `lw` immediate



I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

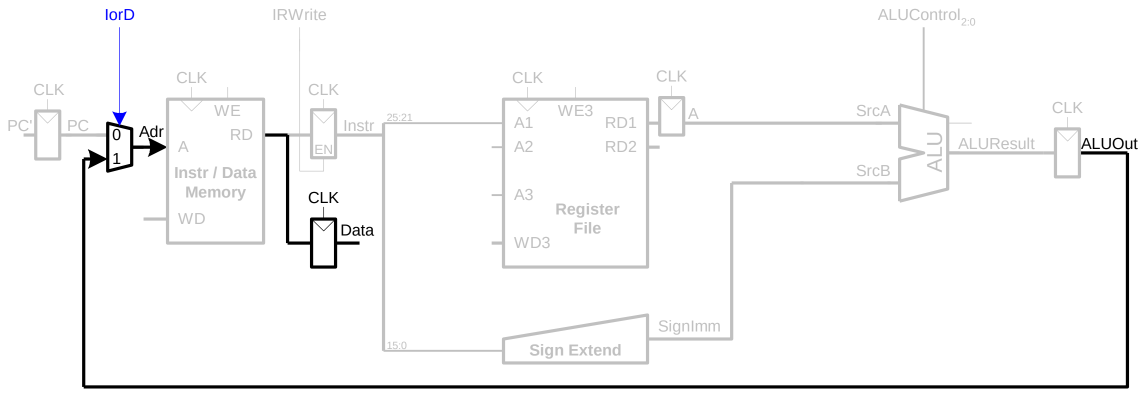
Multi-cycle Datapath: `lw` address



I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

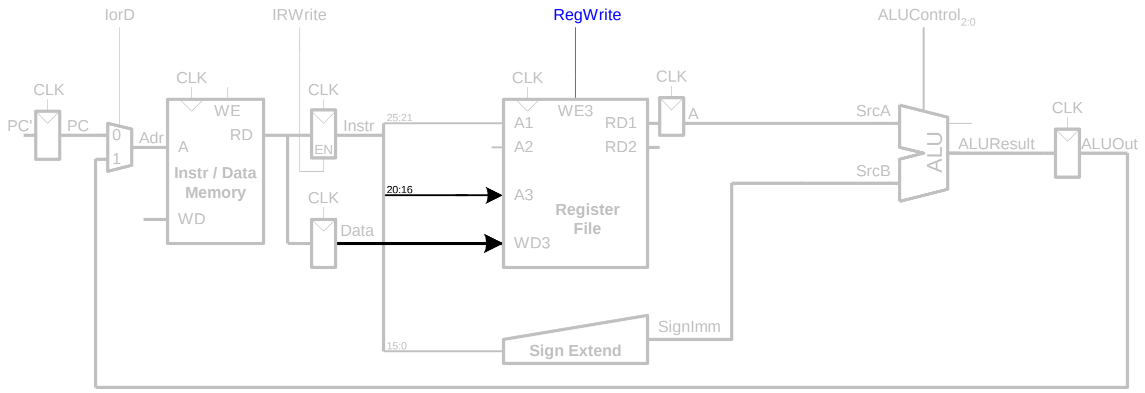
Multi-cycle Datapath: `lw` memory read



I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

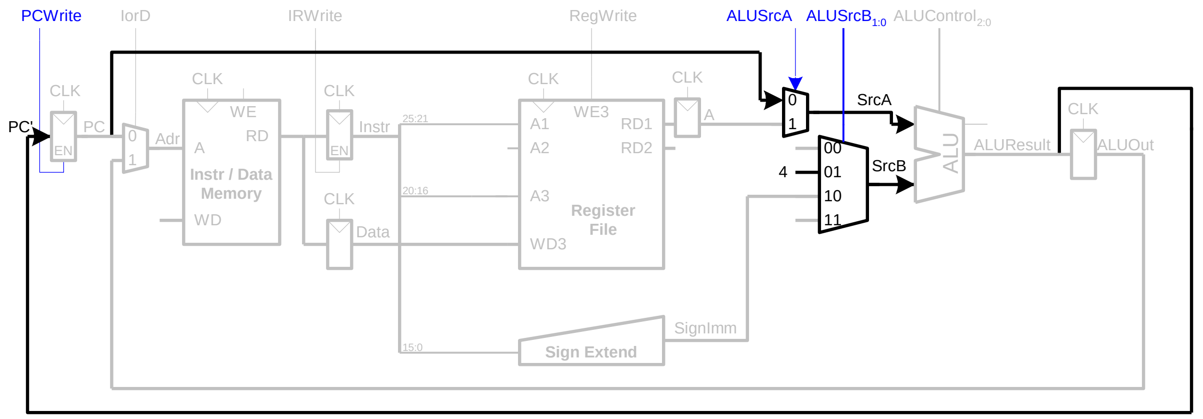
Multi-cycle Datapath: `lw` write register



I-Type

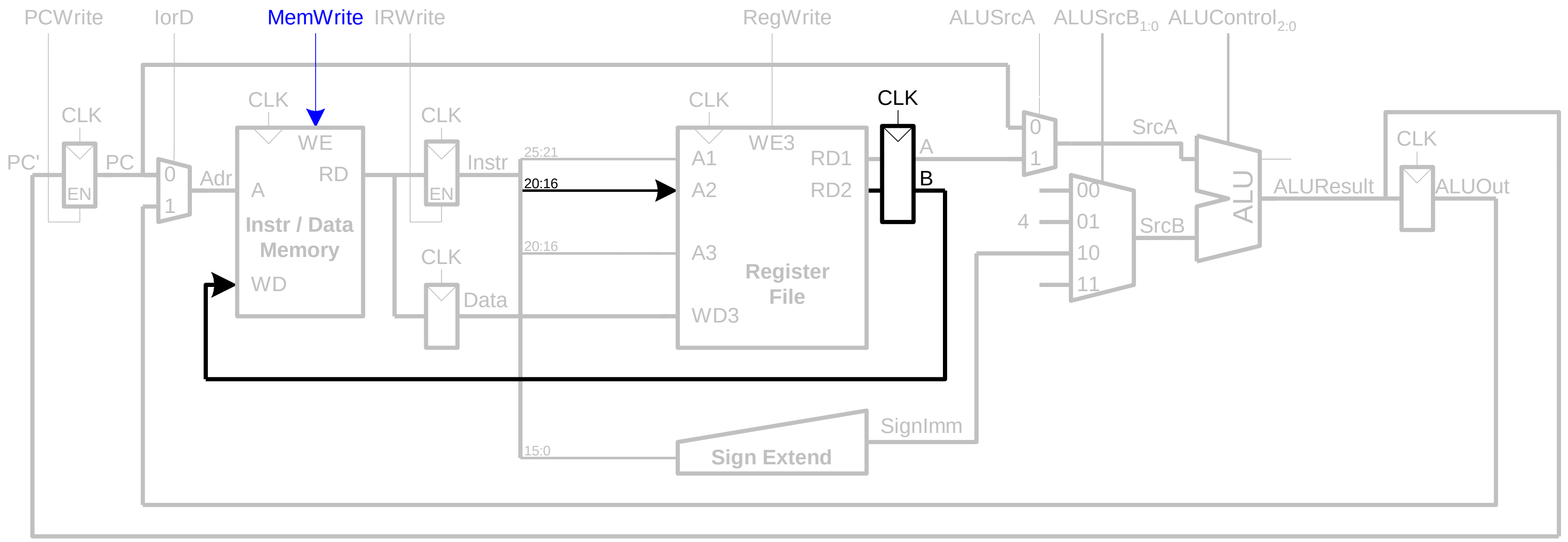
op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

Multi-cycle Datapath: increment PC



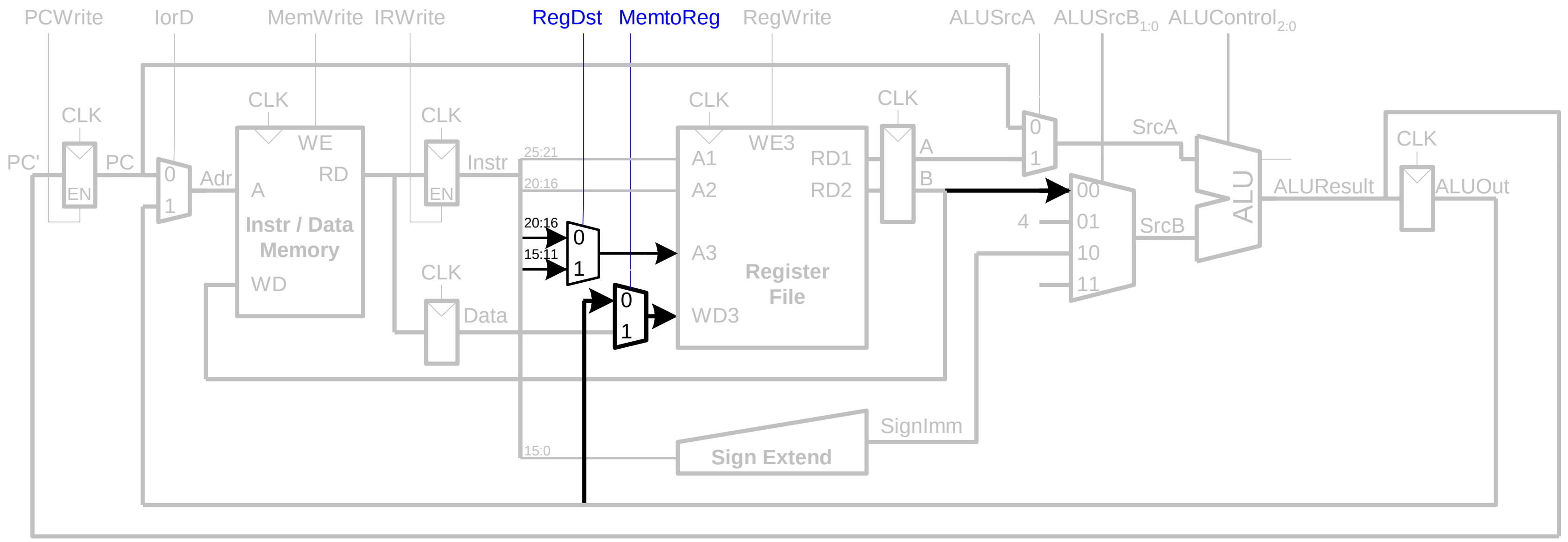
Multi-cycle Datapath: sw

- Write data in rt to memory



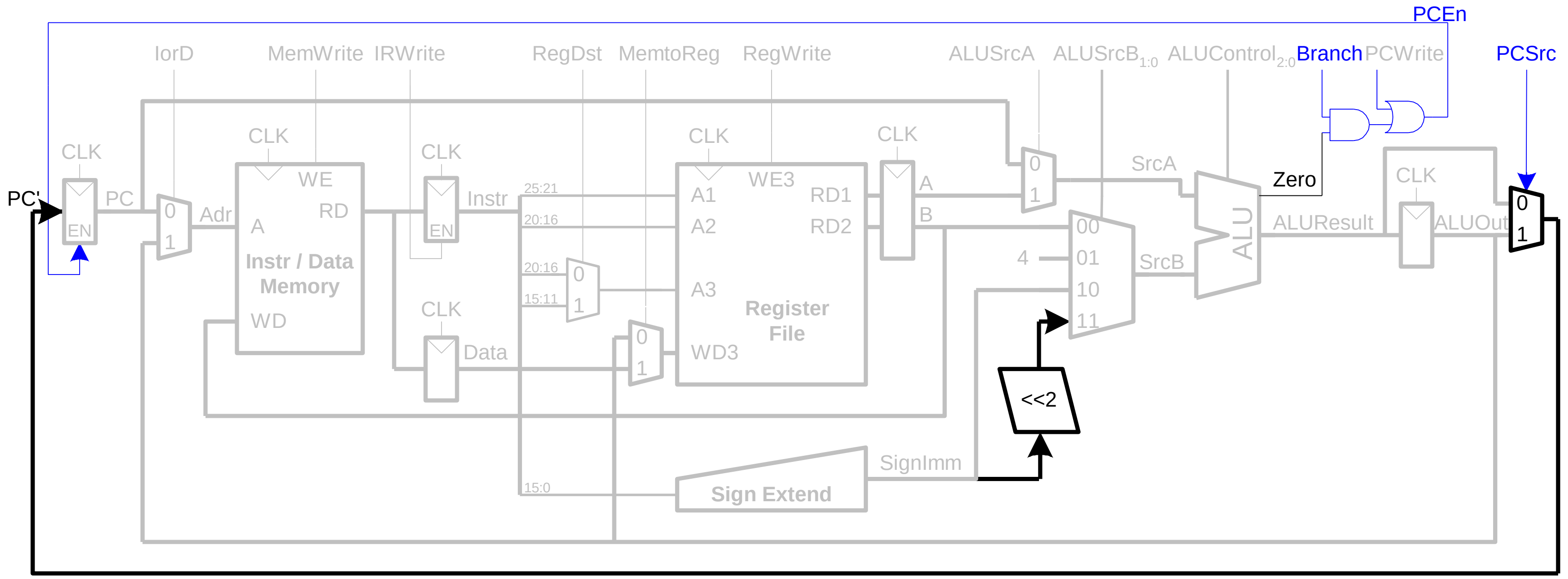
Multi-cycle Datapath: R-type Instructions

- **Read from rs and rt**
  - Write ALUResult to register file
  - Write to rd (instead of rt)

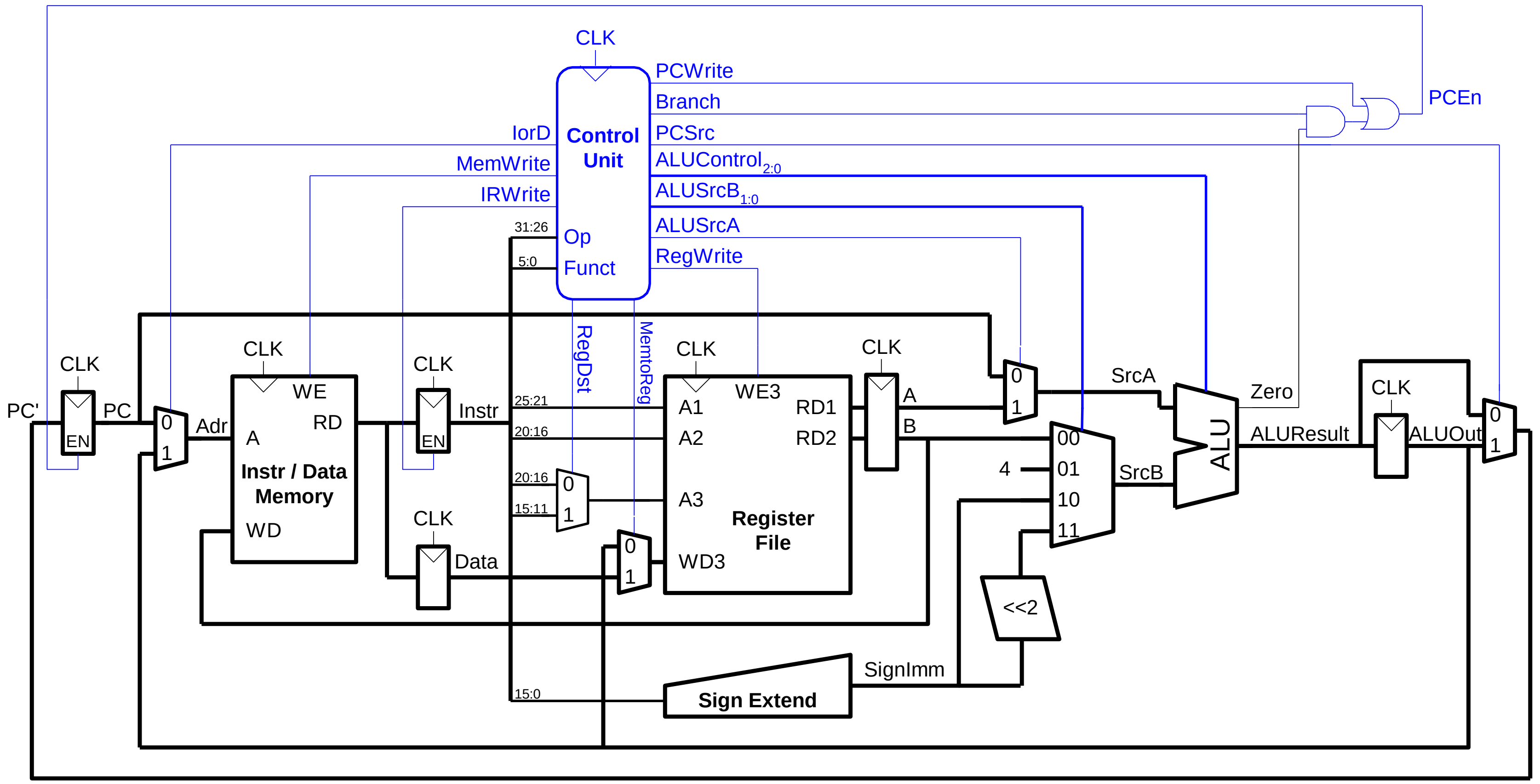


Multi-cycle Datapath: beq

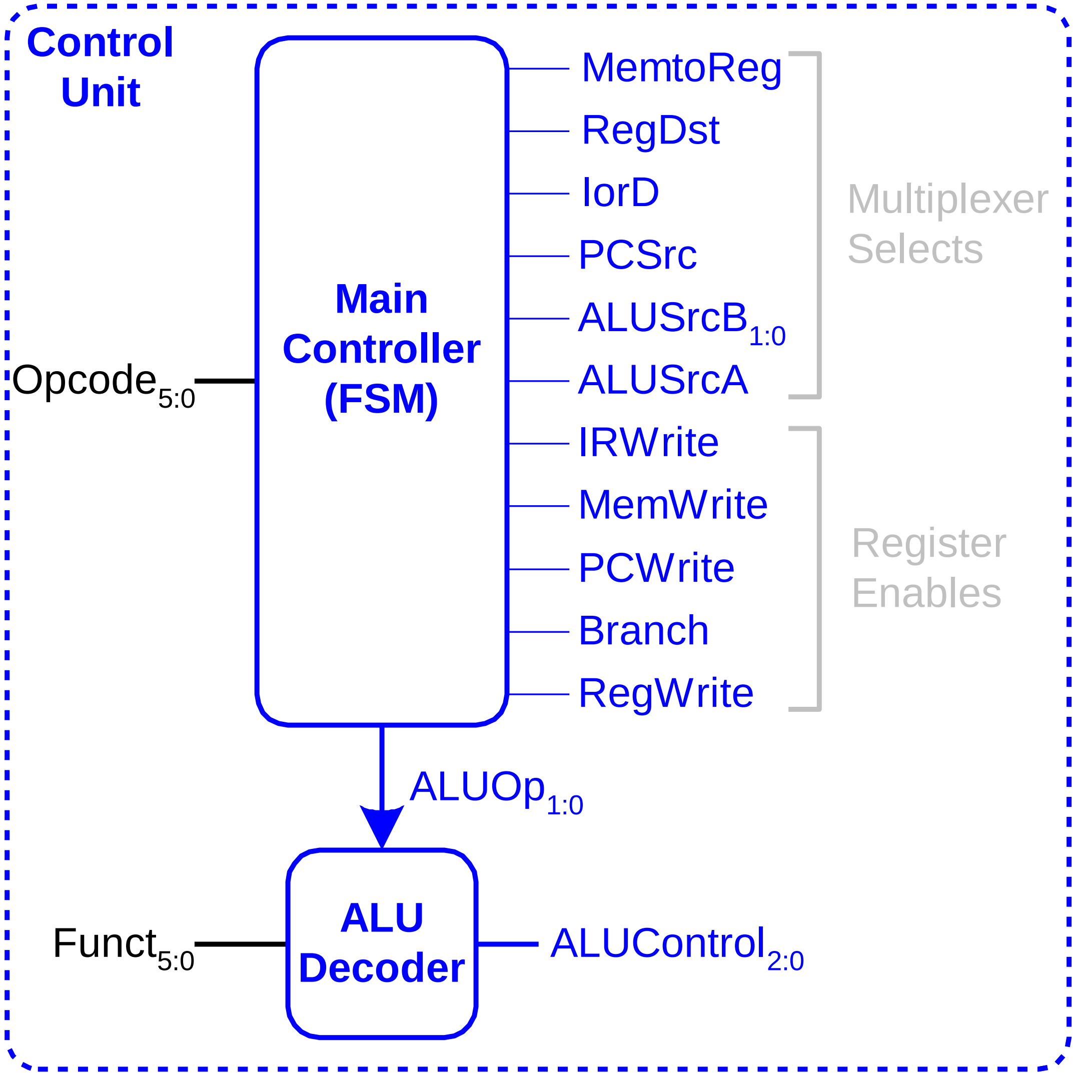
- **Determine whether values in rs and rt are equal**
  - Calculate branch target address:
    BTA = (sign-extended immediate << 2) + (PC+4)



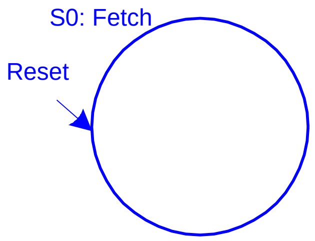
Complete Multi-cycle Processor

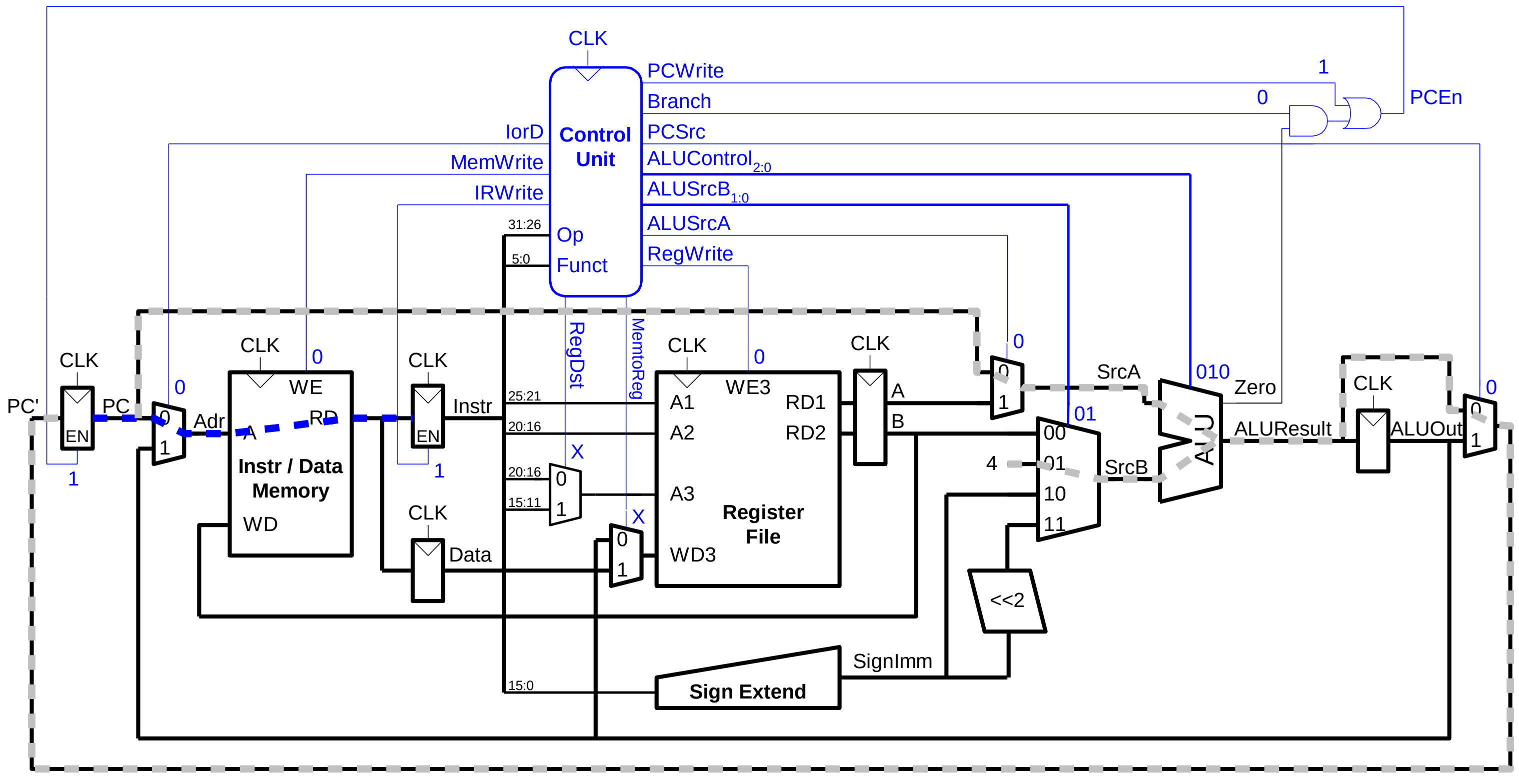


Control Unit

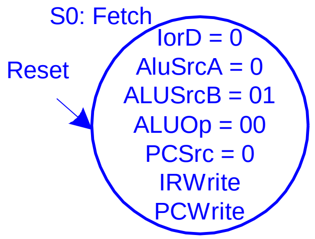


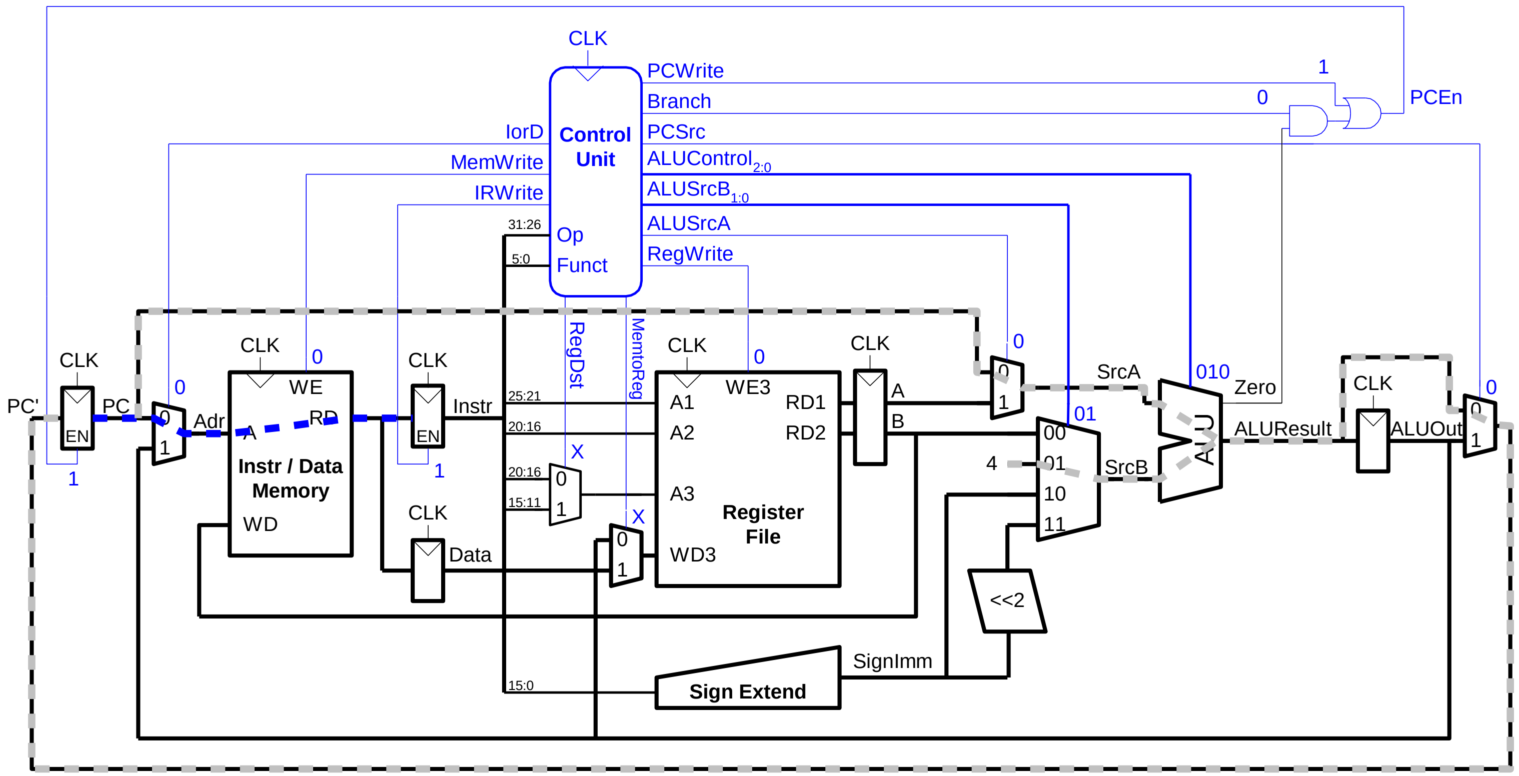
Main Controller FSM: Fetch



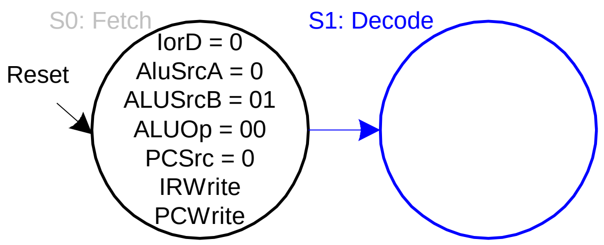


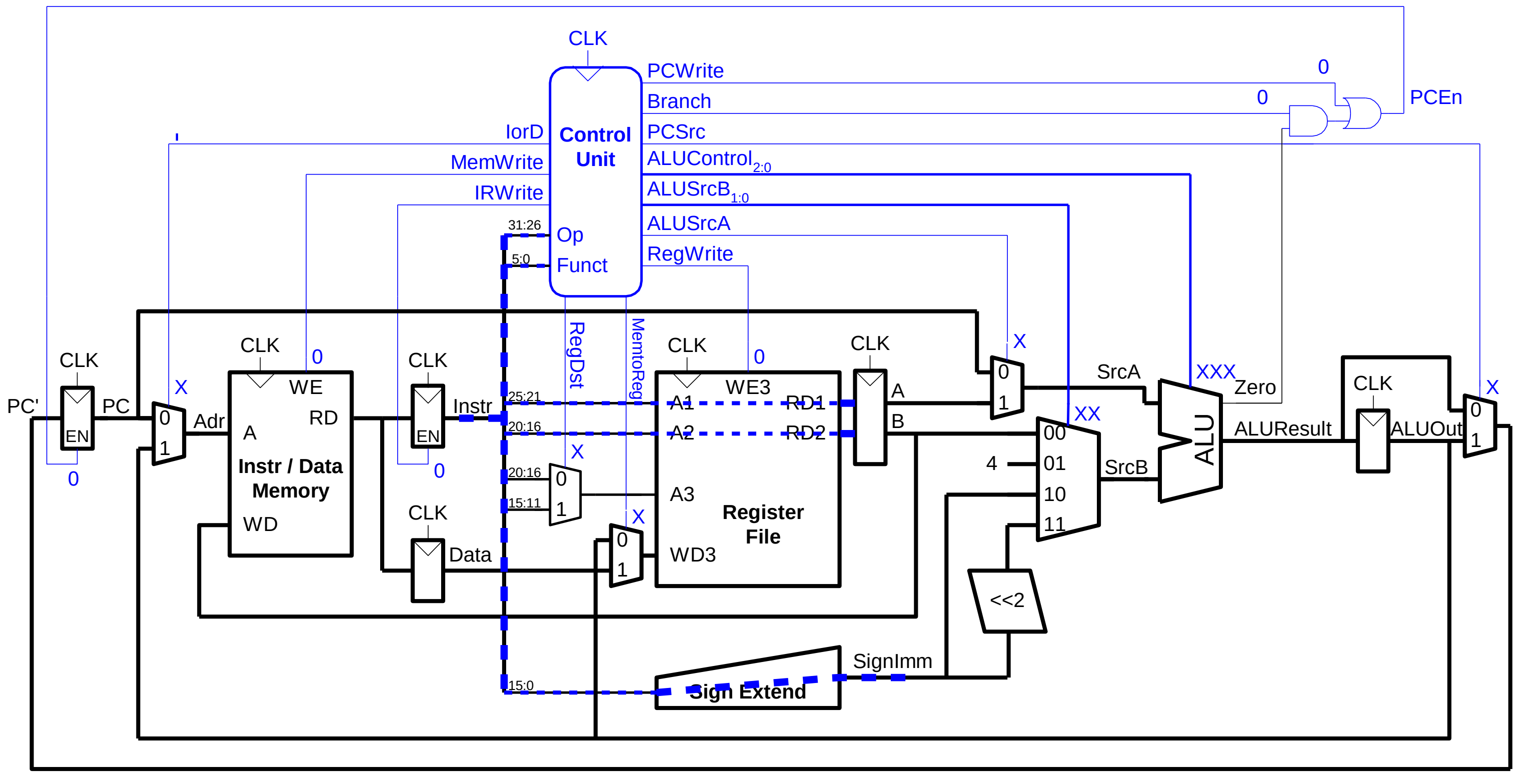
Main Controller FSM: Fetch



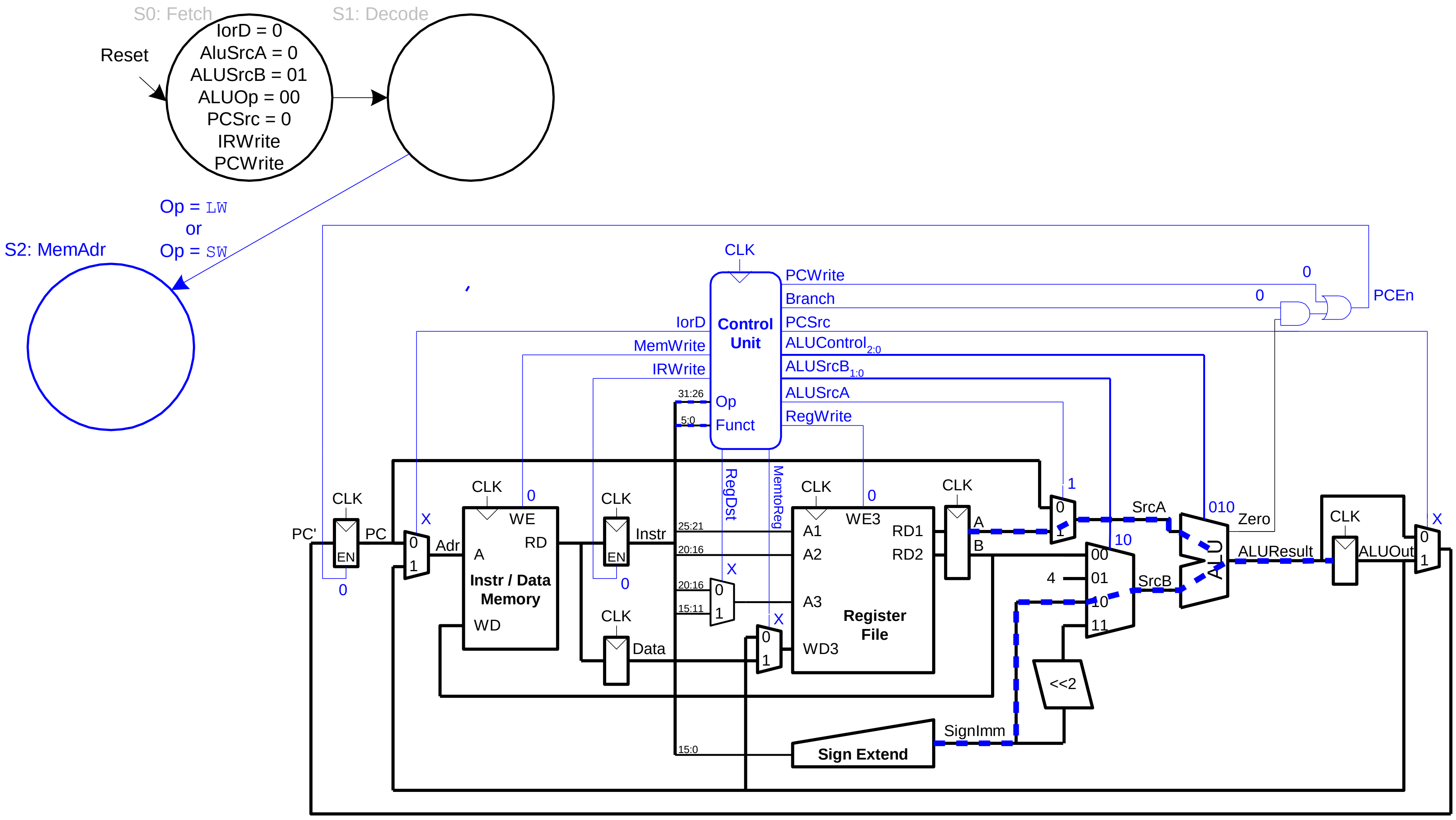


Main Controller FSM: Decode

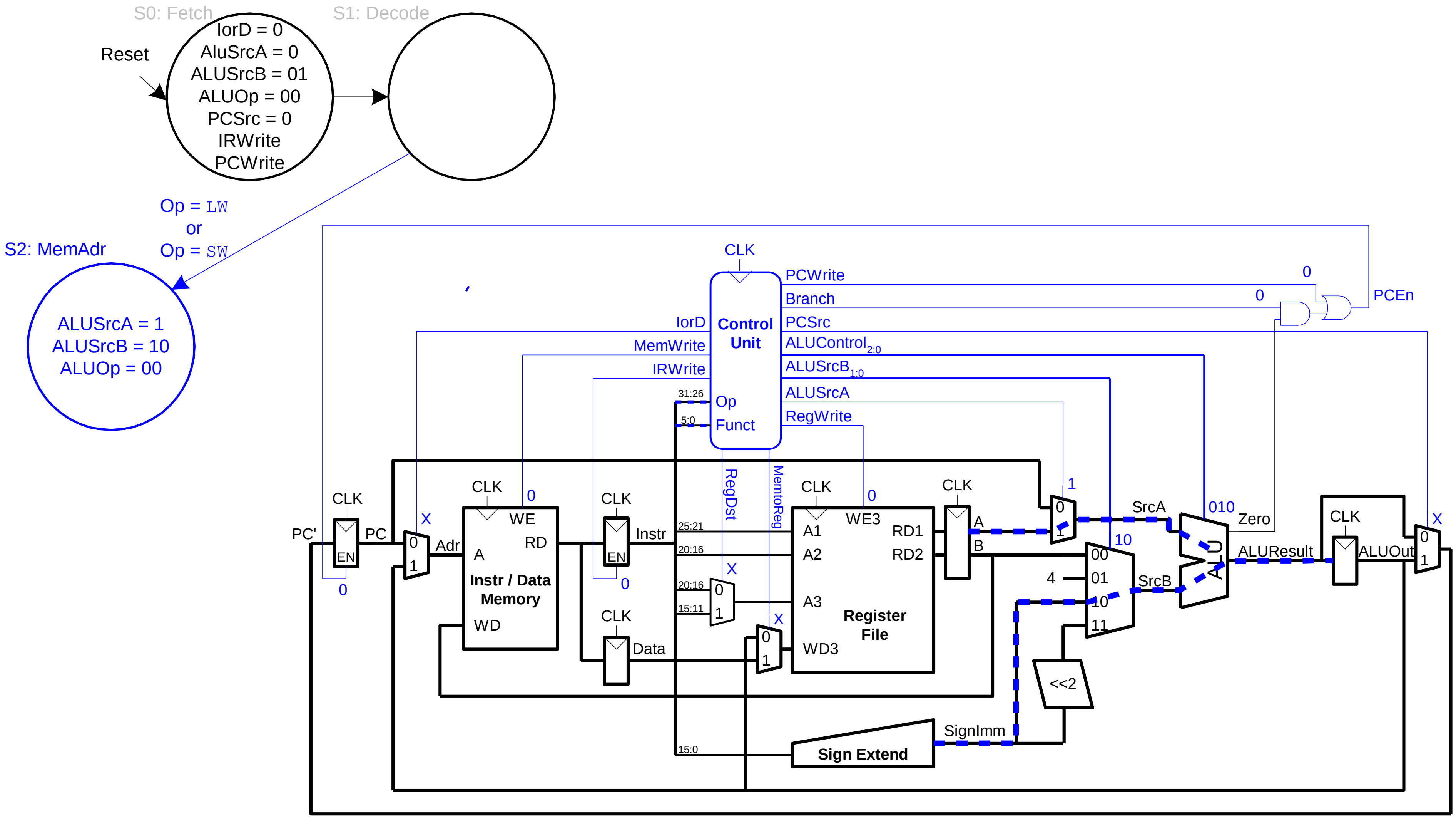




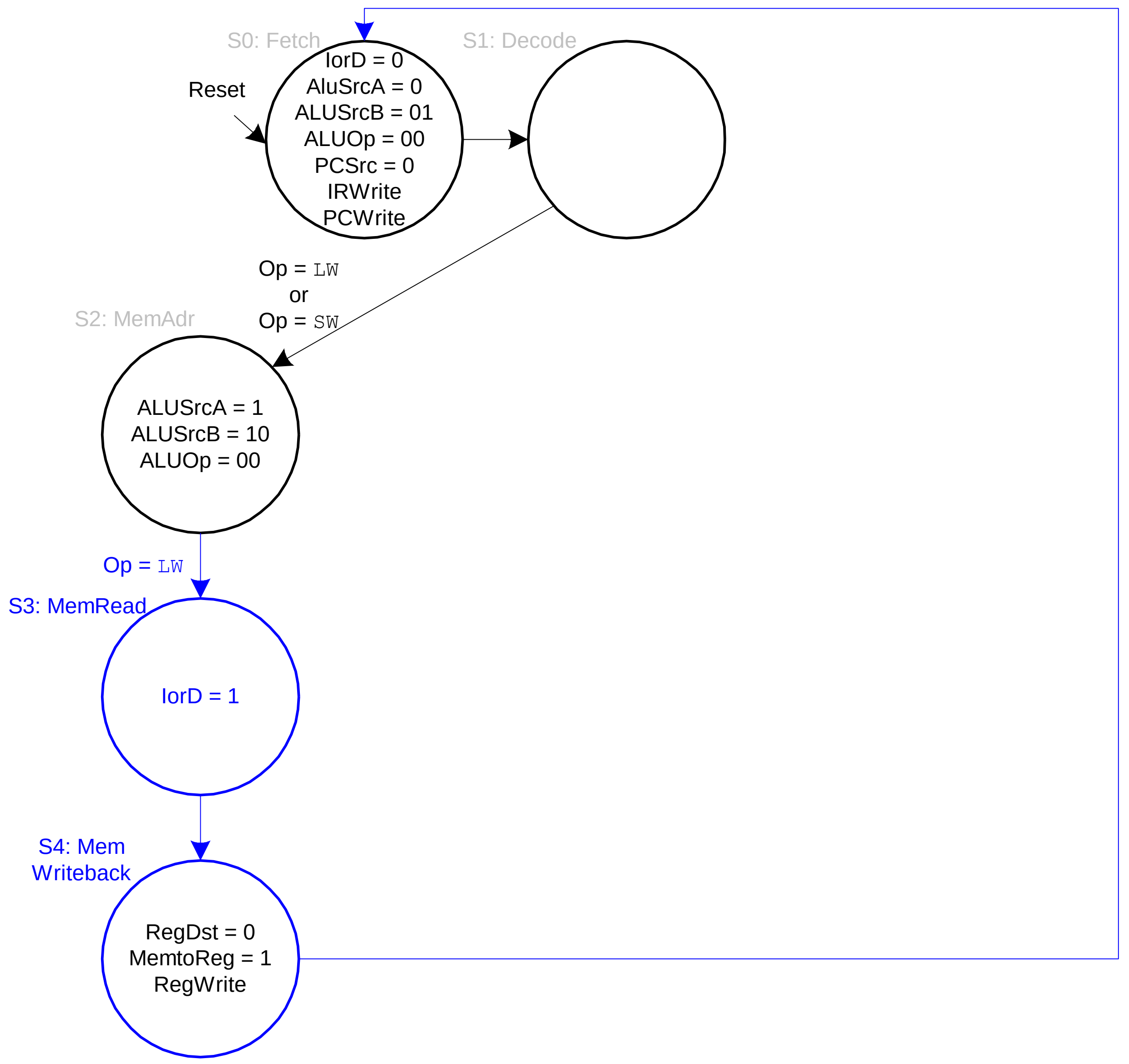
Main Controller FSM: Address Calculation



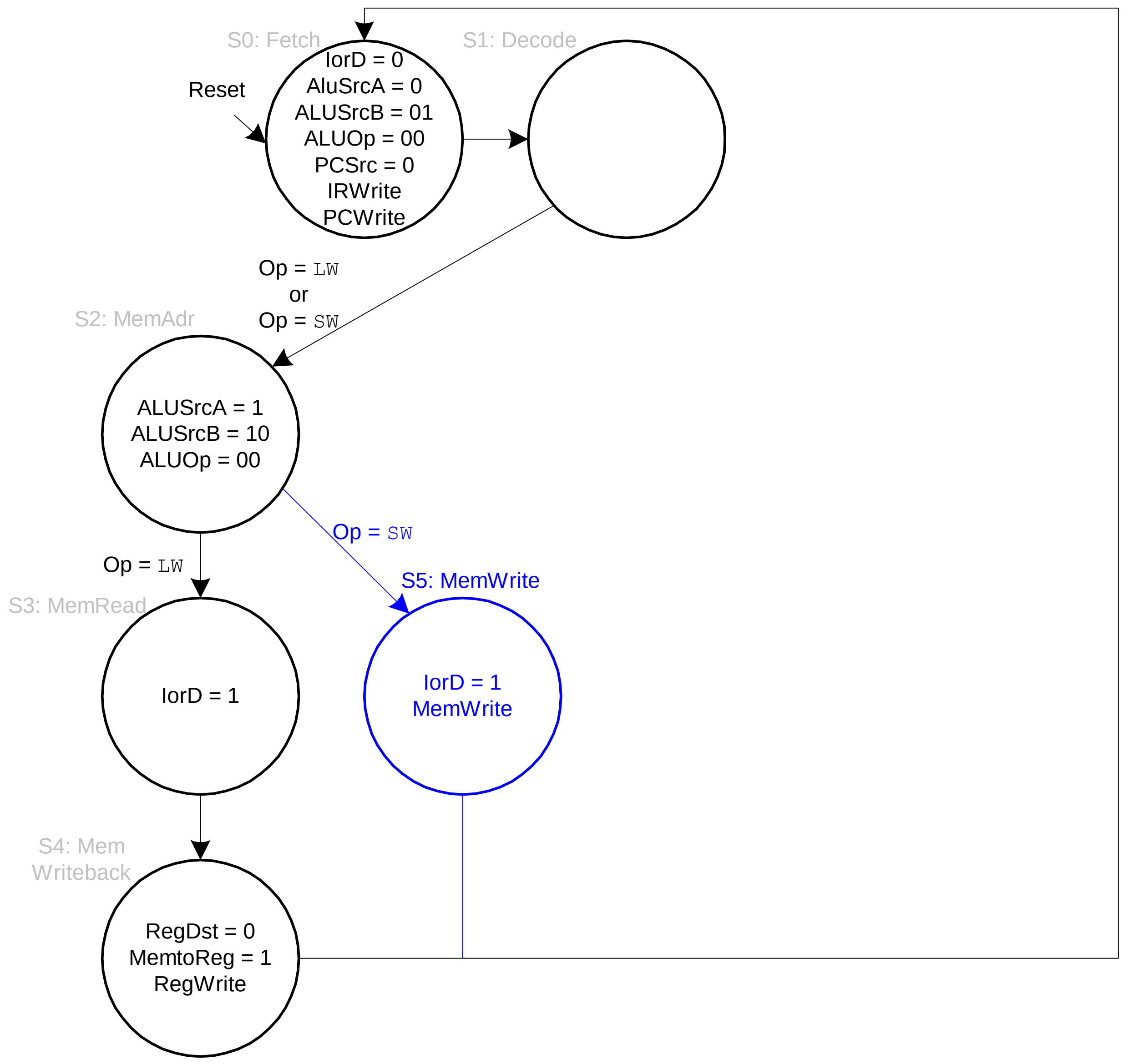
Main Controller FSM: Address Calculation



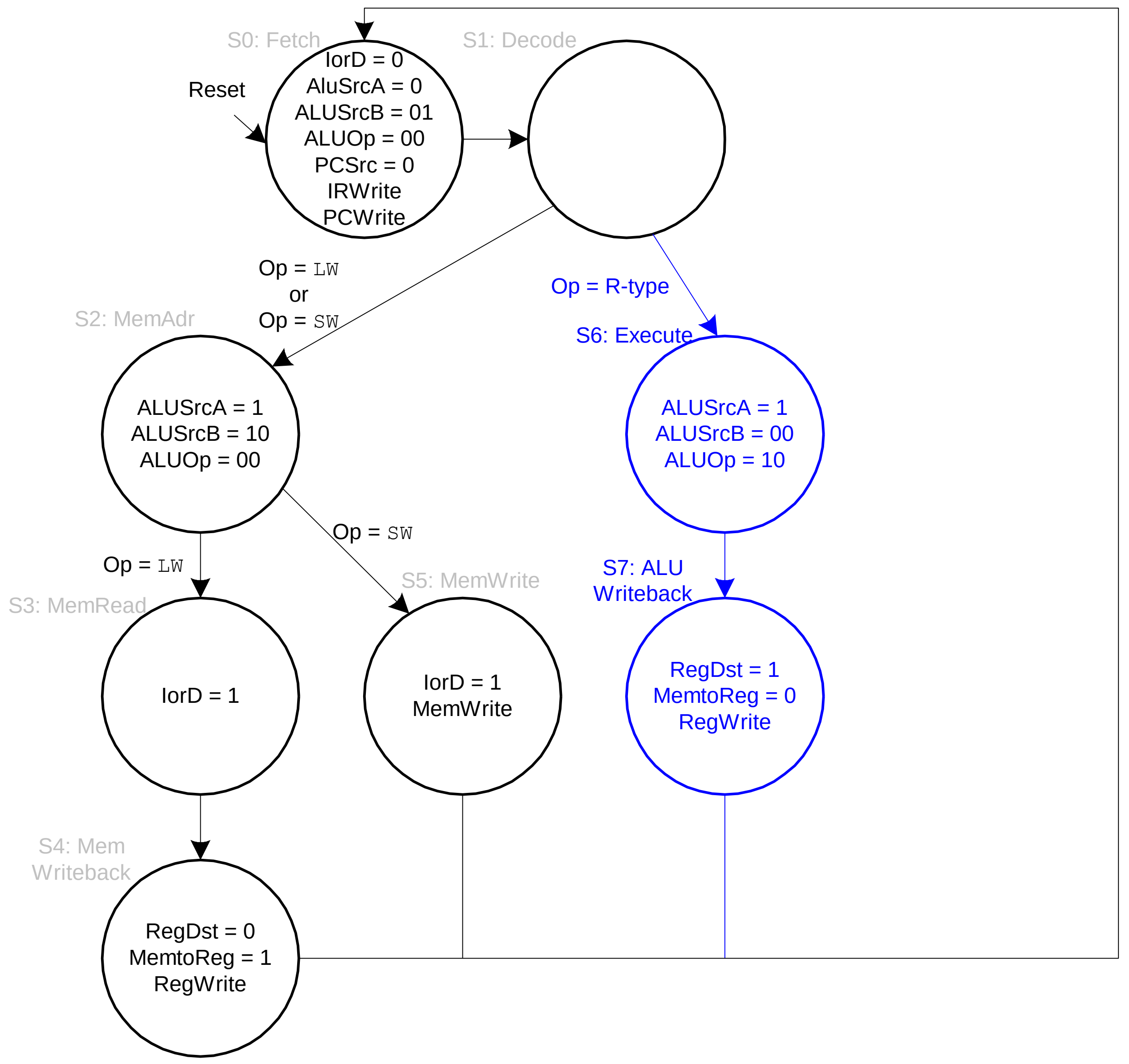
Main Controller FSM: `lw`



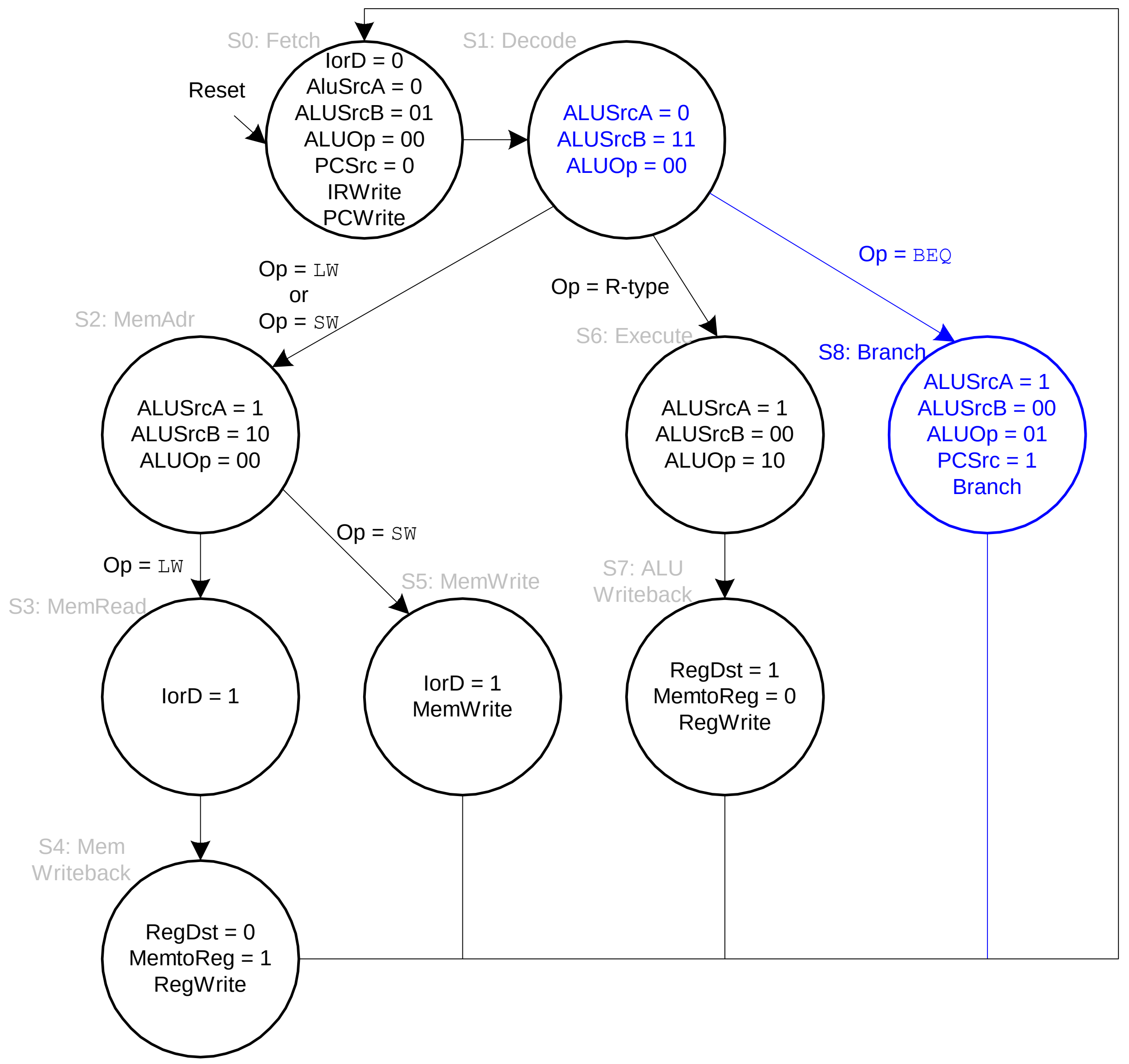
Main Controller FSM: sw



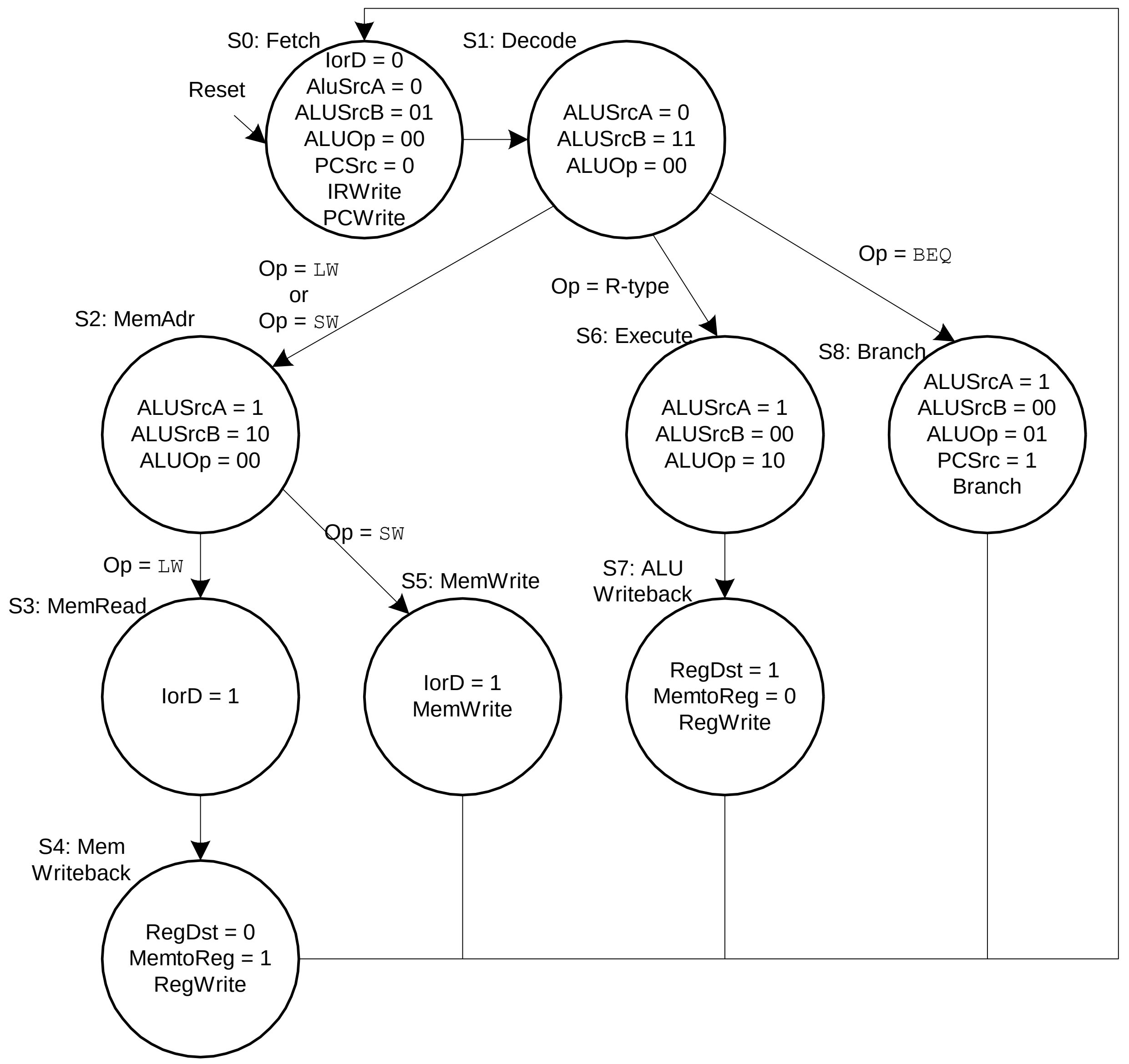
Main Controller FSM: R-Type



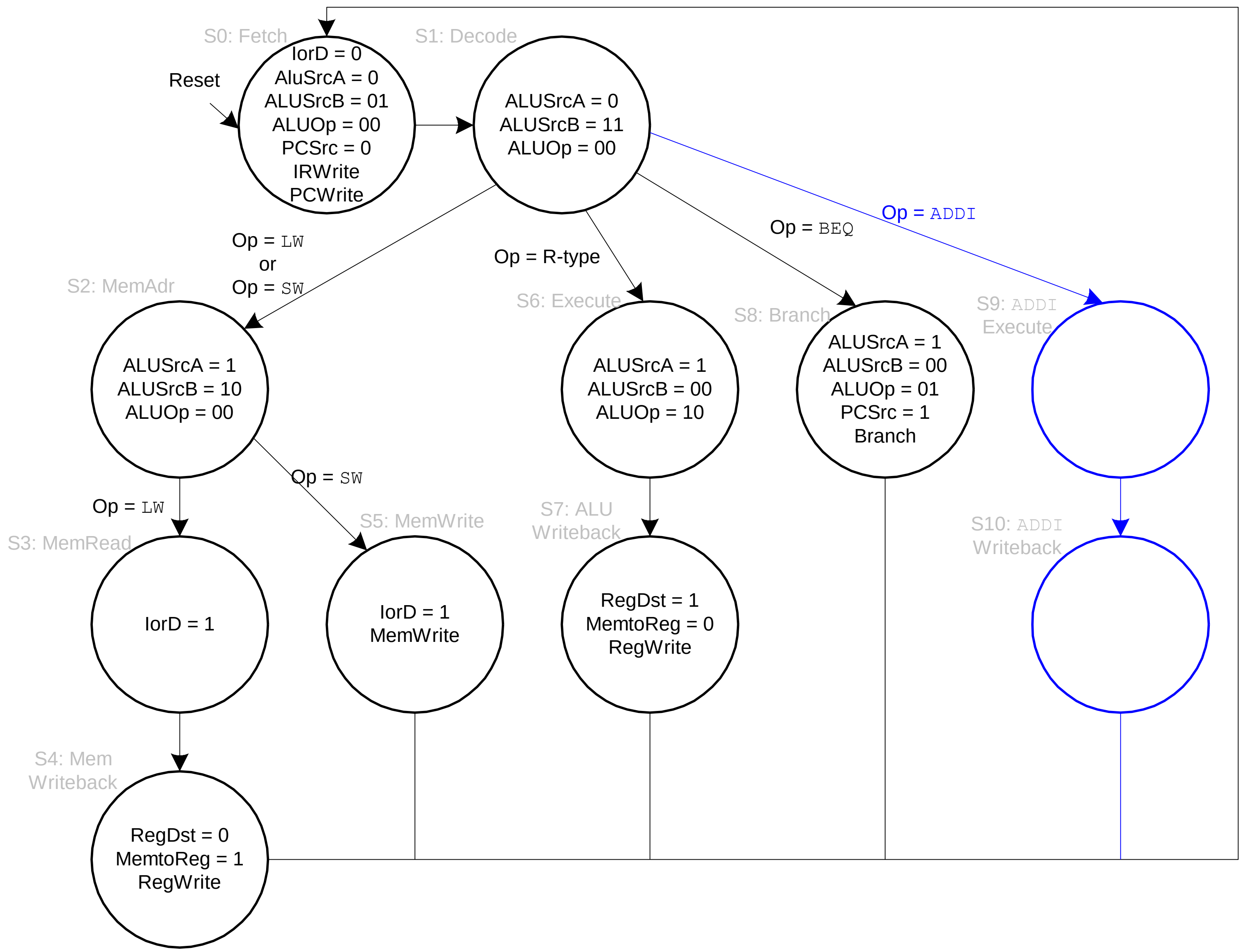
Main Controller FSM: beq



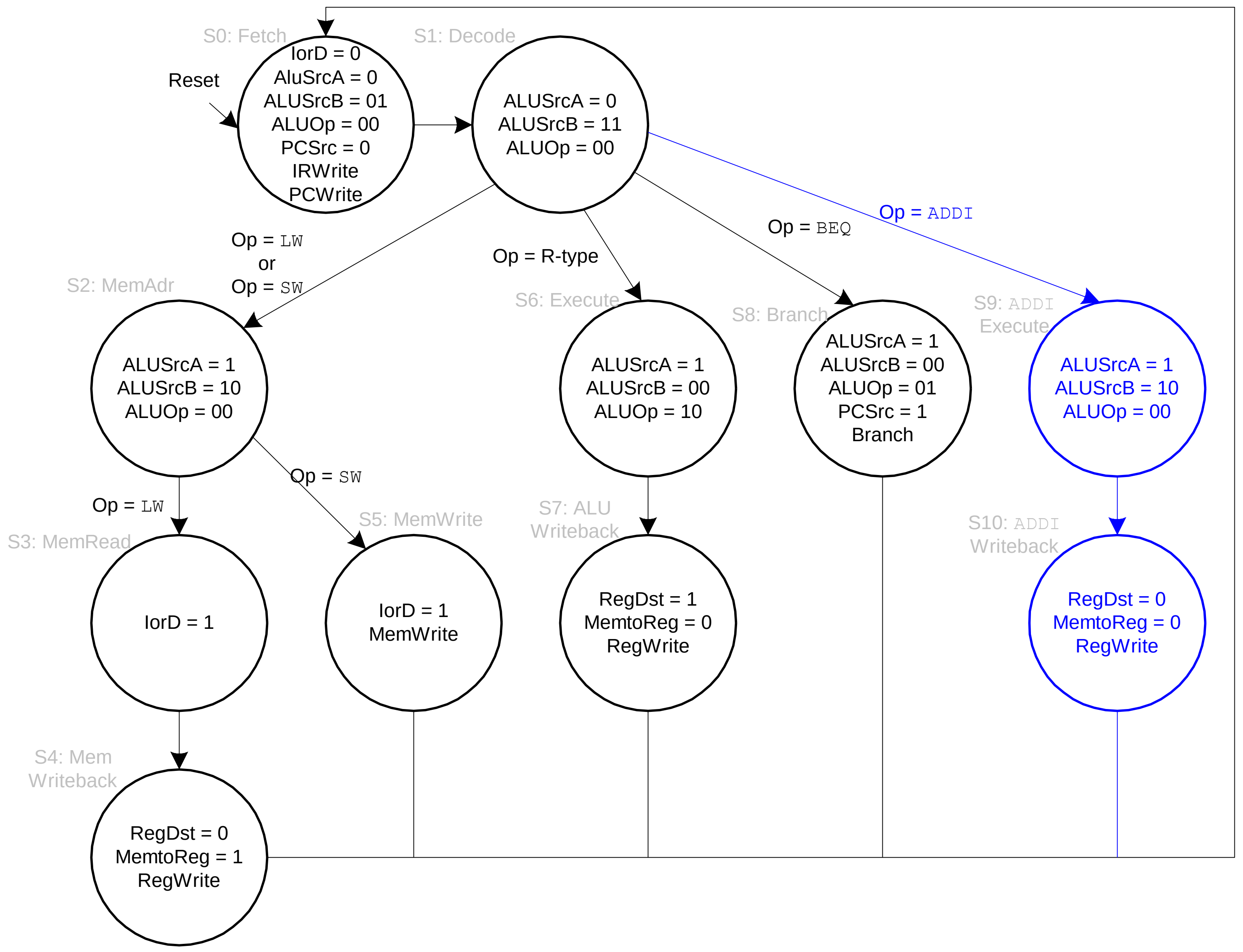
Complete Multi-cycle Controller FSM



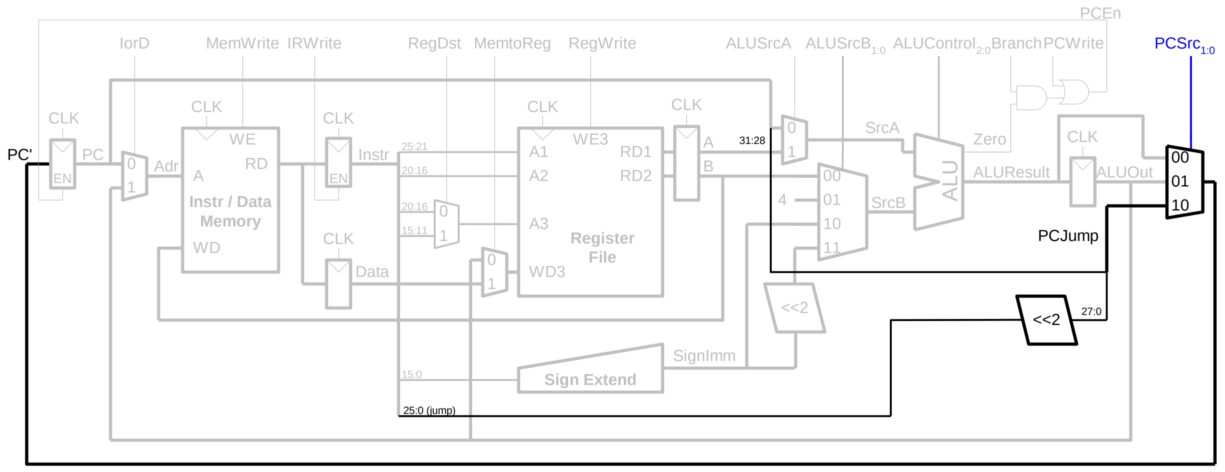
Main Controller FSM: addi



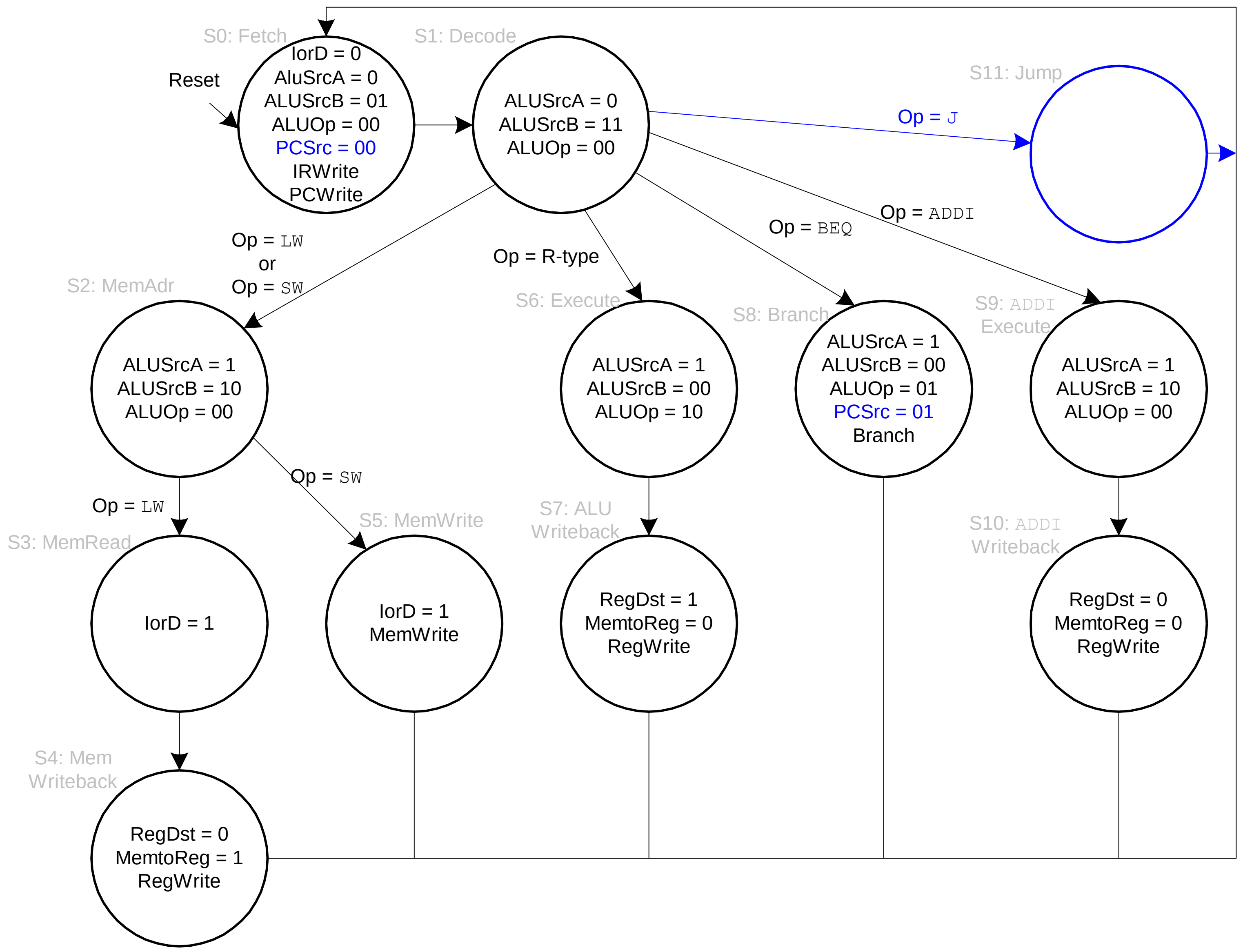
Main Controller FSM: addi



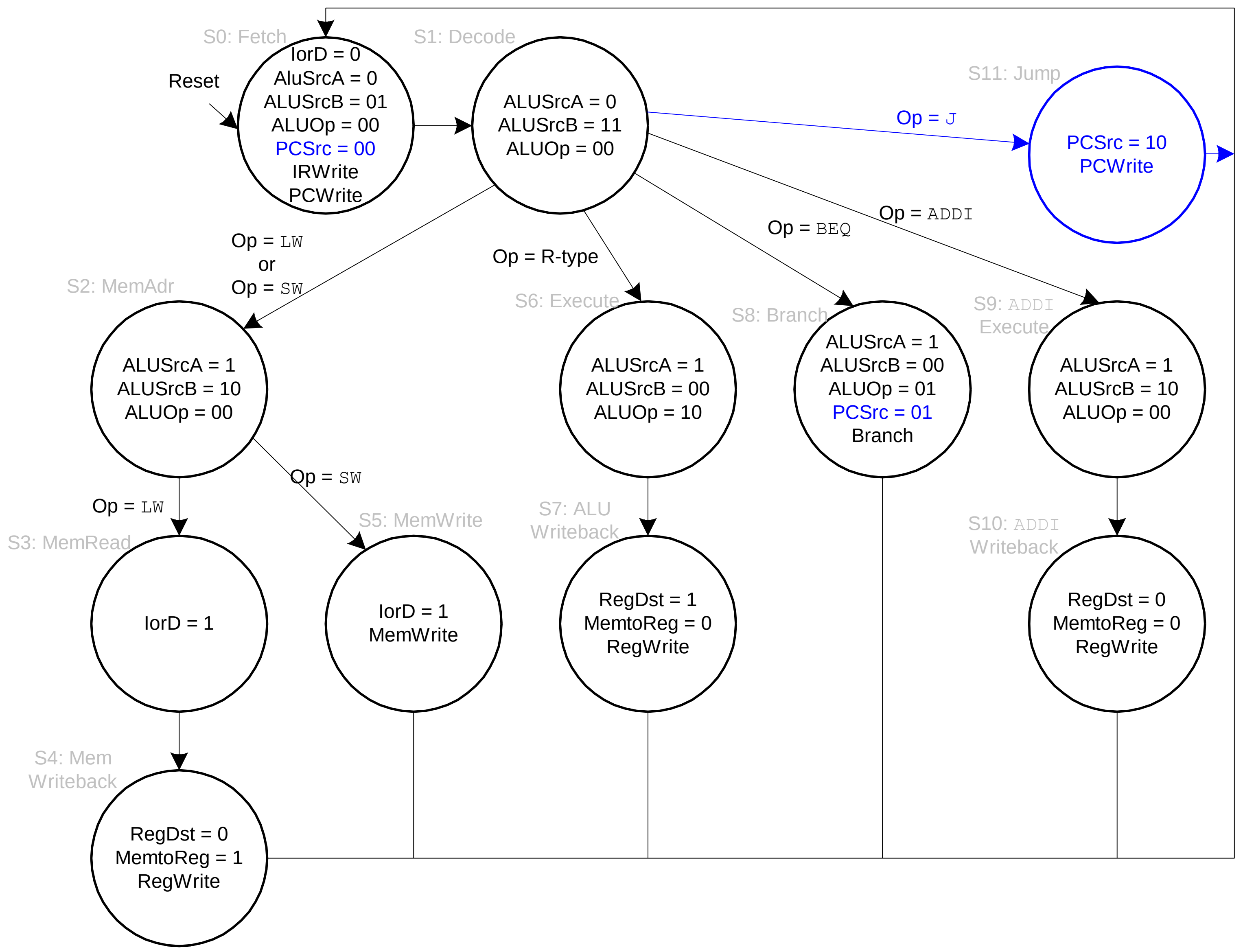
Extended Functionality: j



Control FSM: j



Control FSM: j



Multi-cycle Performance

- **Instructions take different number of cycles:**
  - 3 cycles: `beq, j`
  - 4 cycles: `R-Type, sw, addi`
  - 5 cycles: `lw`

- **CPI is weighted average, i.e. SPECINT2000 benchmark:**

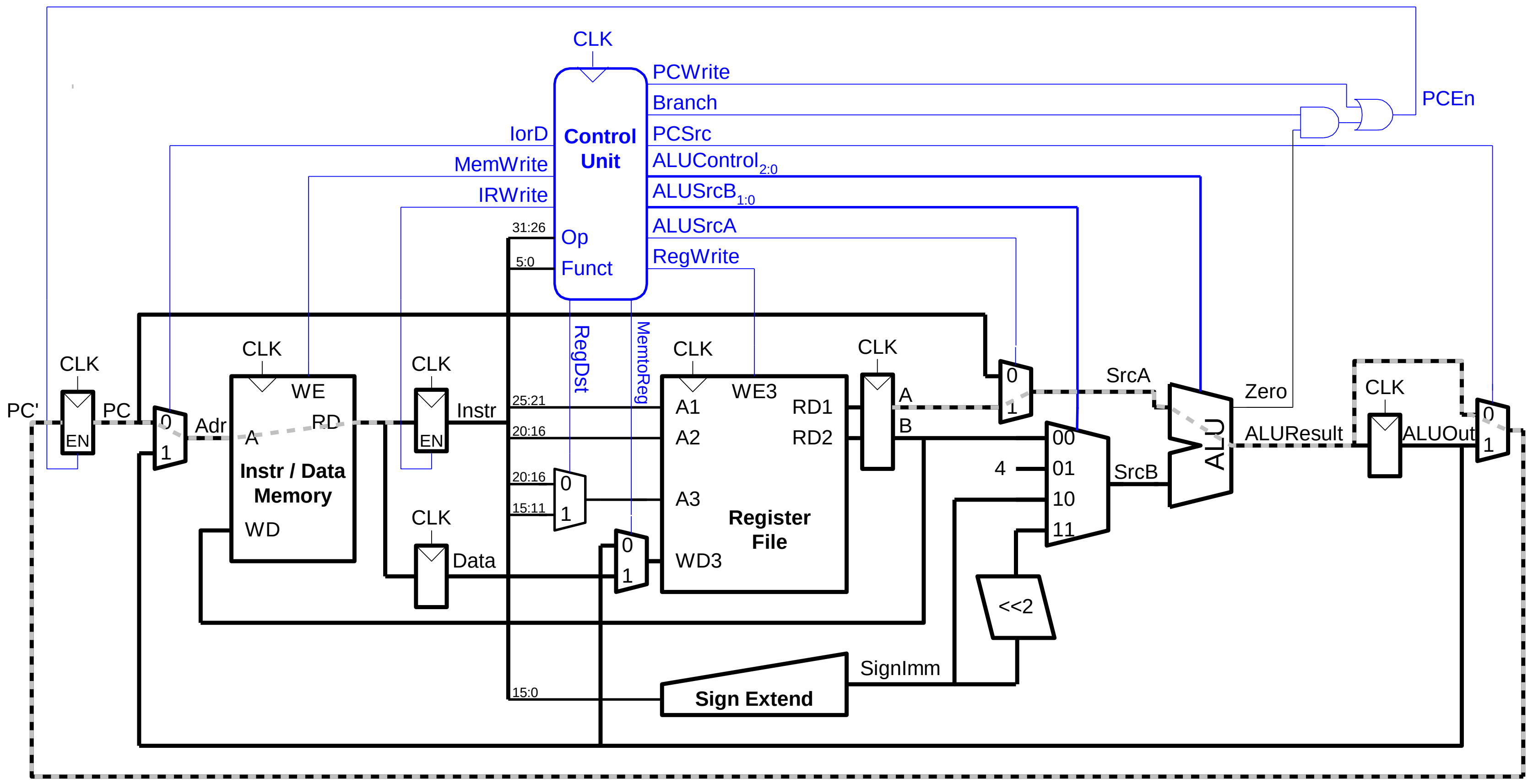
25%	loads
10%	stores
11%	branches
2%	jumps
52%	R-type

- ***Average CPI*** **=** (0.11 + 0.02) 3 +(0.52 + 0.10) 4 +(0.25) 5
  = 4.12

Multi-cycle Performance

- **Multi-cycle critical path:**

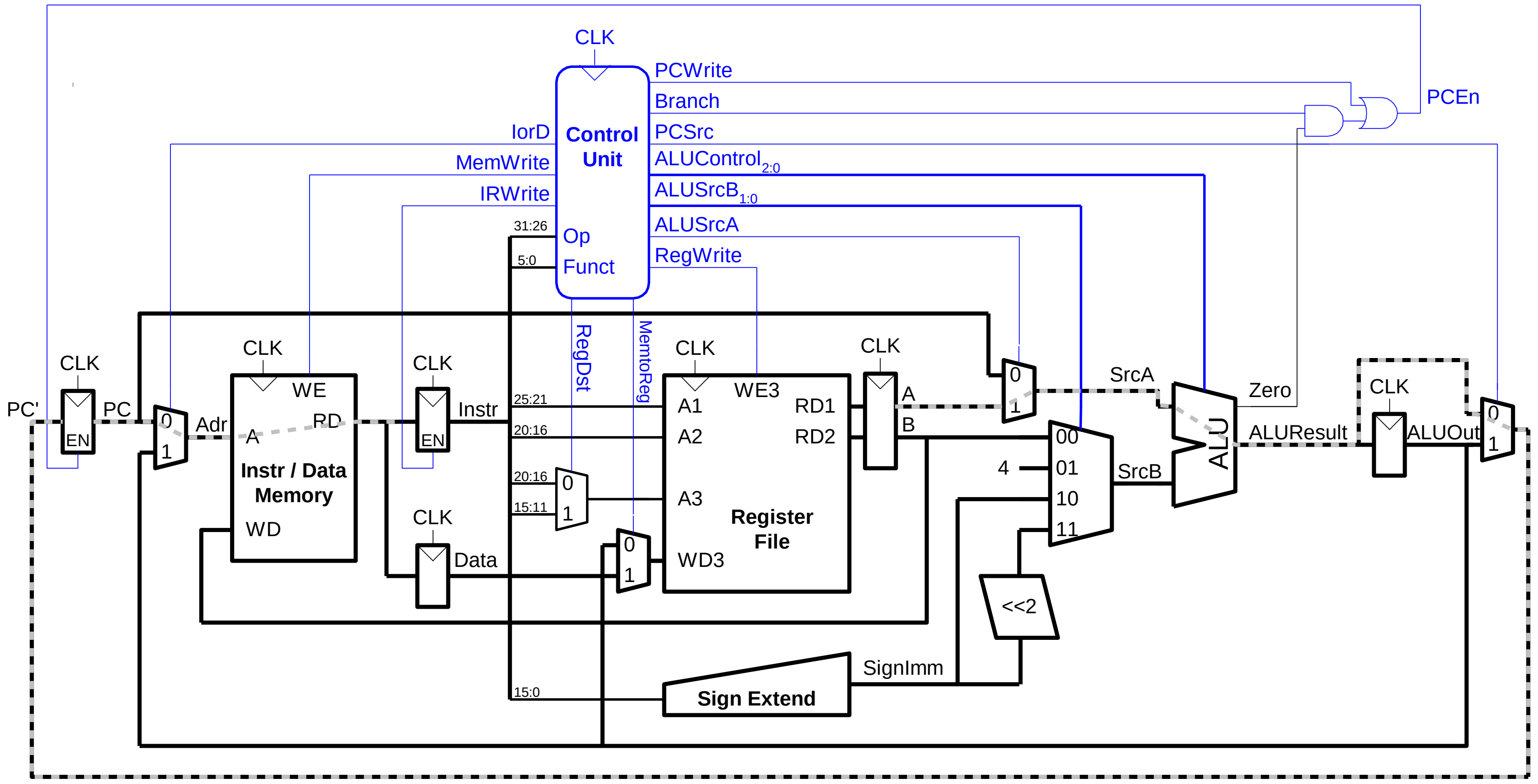
$T_c =$



Multi-cycle Performance

- **Multicycle critical path:**

$$T_c = t_{pcq} + t_{mux} + \max(t_{ALU} + t_{mux}, t_{mem}) + t_{setup}$$



Multicycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	$t_{RFsetup}$	20

T_c =

Multicycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	$t_{RFsetup}$	20

$$
\begin{aligned}
T_c &= t_{pcq\_PC} + t_{mux} + \max(t_{ALU} + t_{mux}, t_{mem}) + t_{setup} \\
&= [30 + 25 + 250 + 20] \text{ ps} \\
&= 325 \text{ ps}
\end{aligned}
$$

Multi-cycle Performance Example

- **For a program with 100 billion instructions executing on a multi-cycle MIPS processor**
  - CPI = 4.12
  - T_c = 325 ps

- ***Execution Time*** **= (# instructions) × CPI × T_c**
  = (100 × 109)(4.12)(325 × 10-12)
  = 133.9 seconds

- **This is slower than the single-cycle processor (92.5 seconds). Why?**

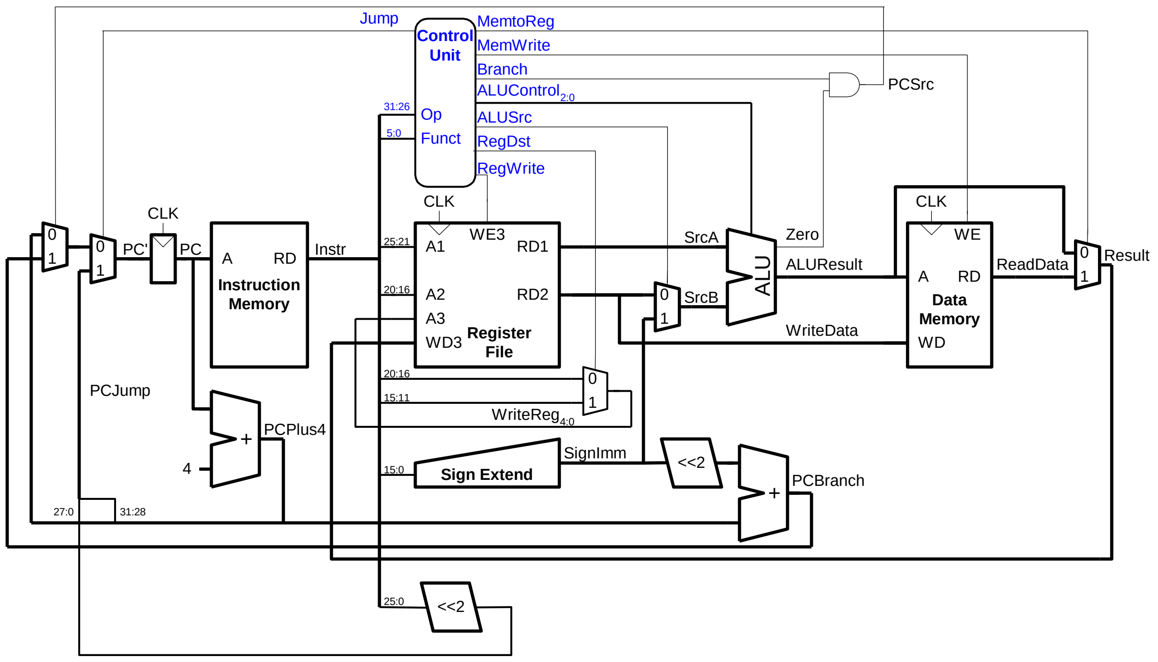
Multi-cycle Performance Example

- **For a program with 100 billion instructions executing on a multi-cycle MIPS processor**
  - CPI = 4.12
  - T_c = 325 ps

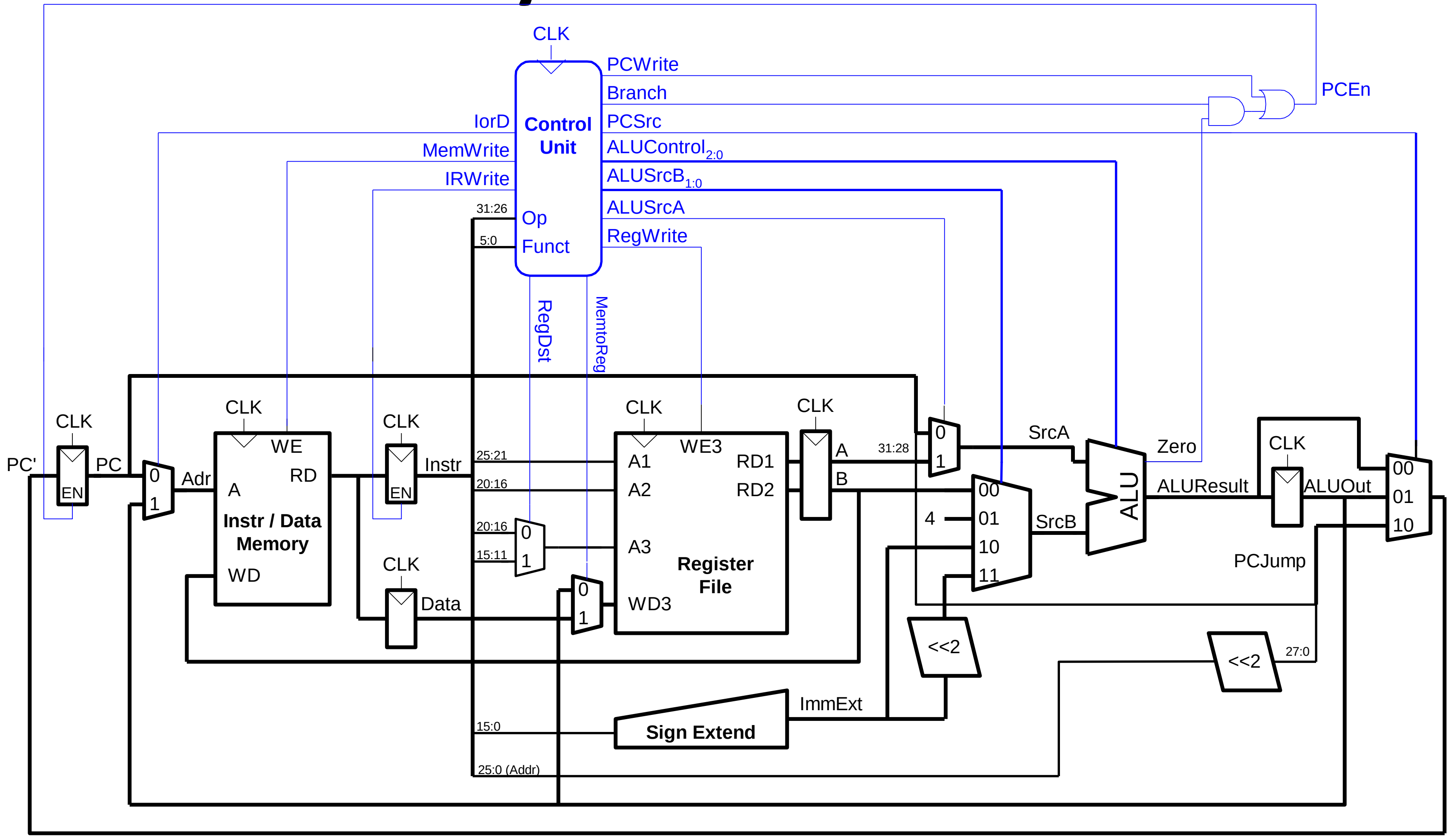
- ***Execution Time*** **= (# instructions) × CPI × T_c**
  = (100 × 109)(4.12)(325 × 10-12)
  = 133.9 seconds

- **This is slower than the single-cycle processor (92.5 seconds). Why?**
  - Not all steps the same length
  - Sequencing overhead for each step (t_{pcq} + t_{setup}= 50 ps)

Review: Single-Cycle MIPS Processor



Review: Multicycle MIPS Processor



What Have We Learned?

- **A more 'realistic' architecture**
  - Shared data and program memory
  - A single ALU for all operations
- **Multi-cycle: Operations take different number of cycles**
  - Simpler operations take less steps
  - More complex operations take more steps
  - Average CPI
- **Bottom line**
  - Smaller
  - More complex control
  - Not necessarily faster (overhead)