CSEE 3827: Fundamentals of Computer Systems, Spring 2011

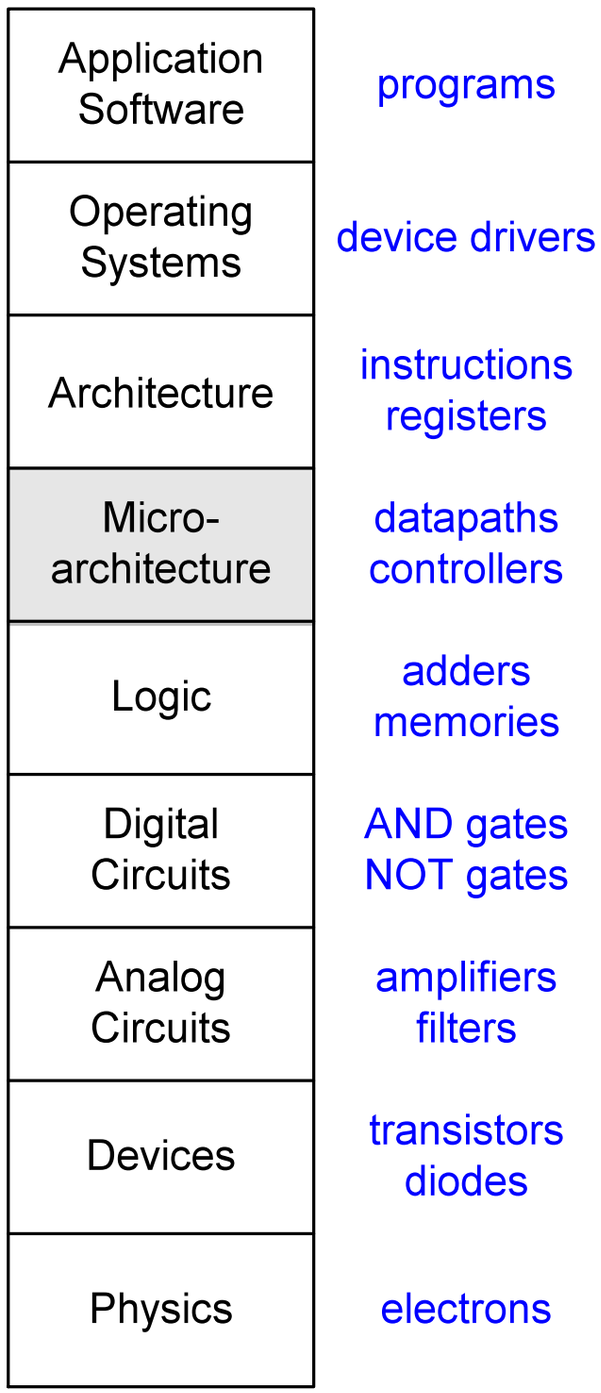
9. Single Cycle MIPS Processor

Prof. Martha Kim (martha@cs.columbia.edu)
Web: http://www.cs.columbia.edu/~martha/courses/3827/sp11/

Outline (H&H 7.2-7.3)

- Single Cycle MIPS Processor
  - Datapath (functional blocks)
  - Control (control signals)
- Single Cycle Performance

Microarchitecture



- **Microarchitecture:** an implementation of a particular architecture
- Multiple implementations for a single architecture
  - **Single-cycle:** Each instruction executes in a single cycle
  - **Multi-cycle:** Each instruction is broken up into a series of shorter steps
  - **Pipelined:** Each instruction is broken up into a series of steps; Multiple instructions execute at once

Our MIPS Processor

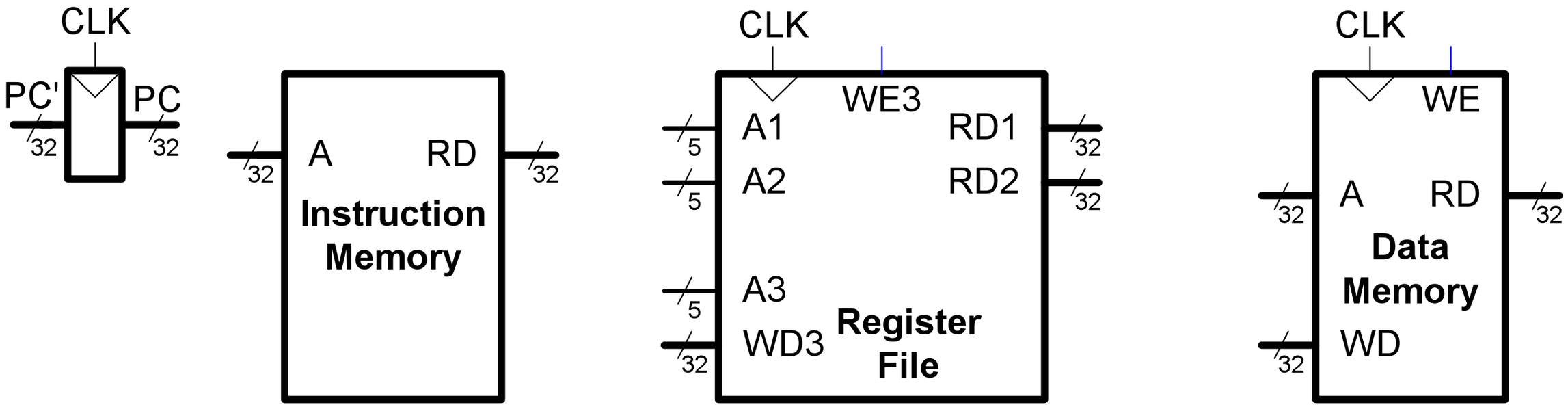
- We consider a subset of MIPS instructions:
  - R-type instructions: `and, or, add, sub, slt`
  - Memory instructions: `lw, sw`
  - Branch instructions: `beq`
- Later consider adding `addi` and `j`

Instruction Execution

- PC → instruction memory, fetch instruction
- Register numbers → register file, read registers
- Depending on instruction class:
    - Use ALU to calculate:
        - Arithmetic or logical result
        - Memory address for load/store
        - Branch target address
    - Access data for load/store
    - PC ← target address or PC + 4

MIPS State Elements

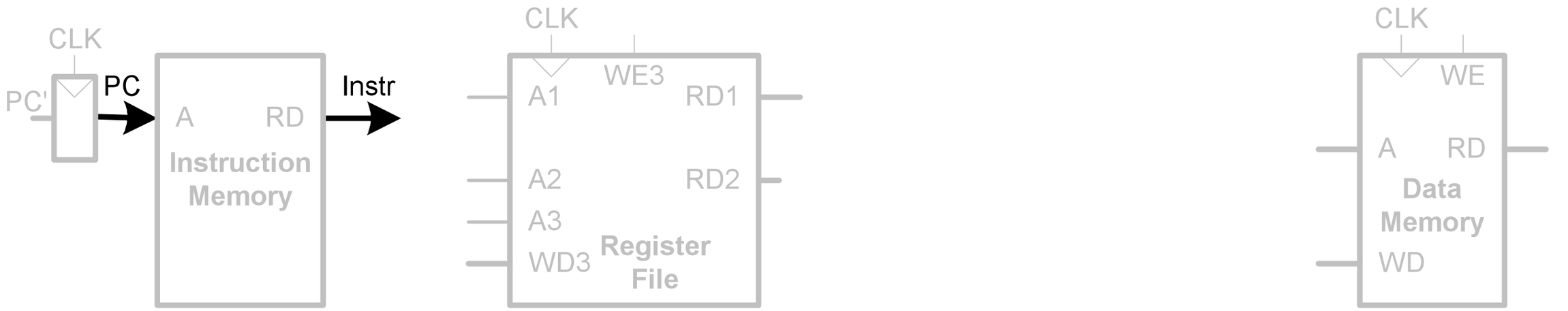
- Architectural state determines everything about a processor: PC, 32 registers, memory



Single-Cycle Datapath: `lw` fetch

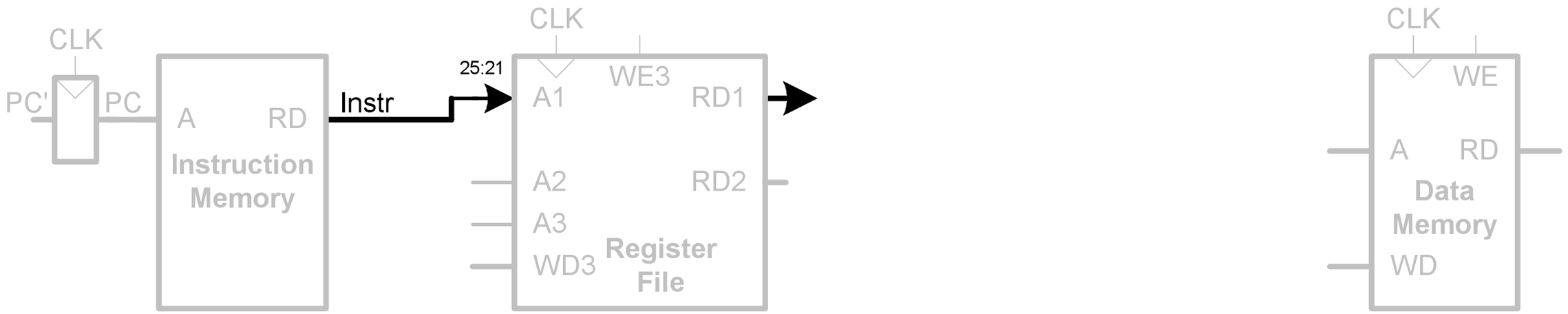
- First consider executing `lw`

- STEP 1: Fetch instruction



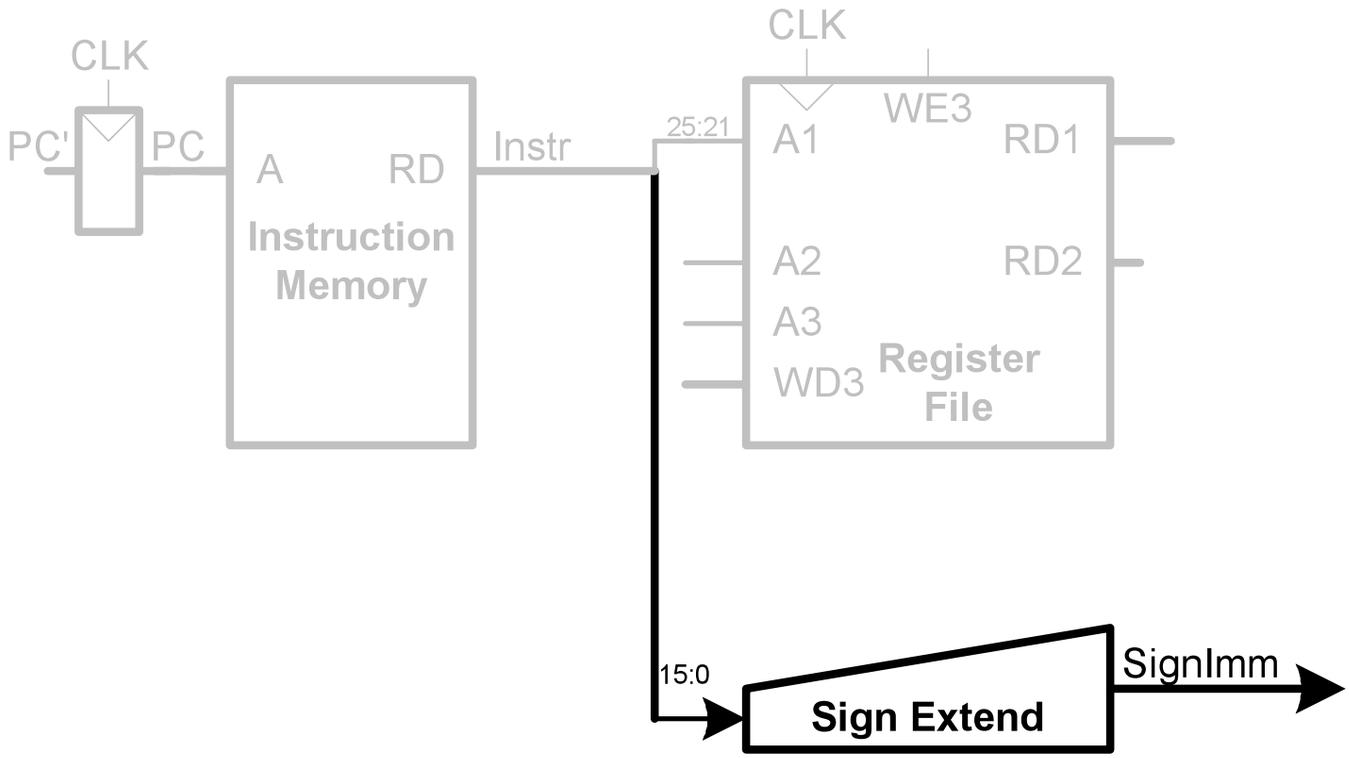
Single-Cycle Datapath: `lw` register read

- STEP 2: Read source operands from register file



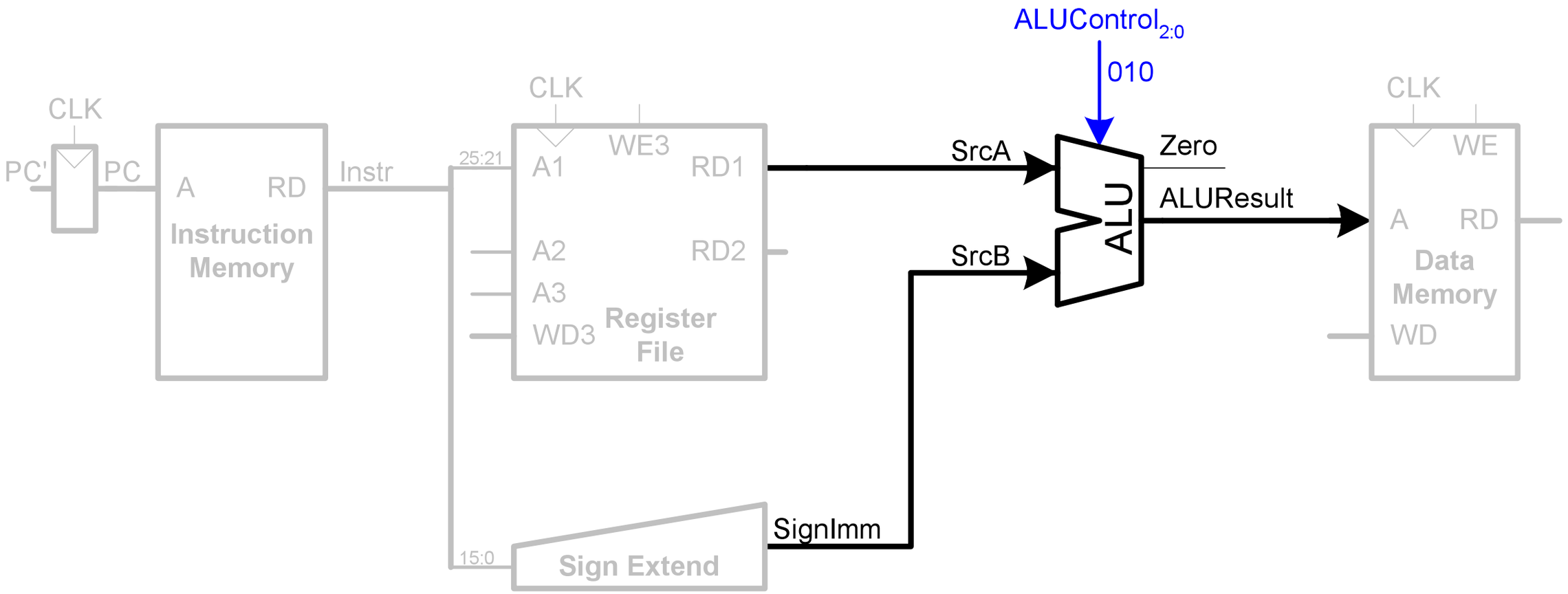
Single-Cycle Datapath: `lw` immediate

- STEP 3: Sign-extend the immediate



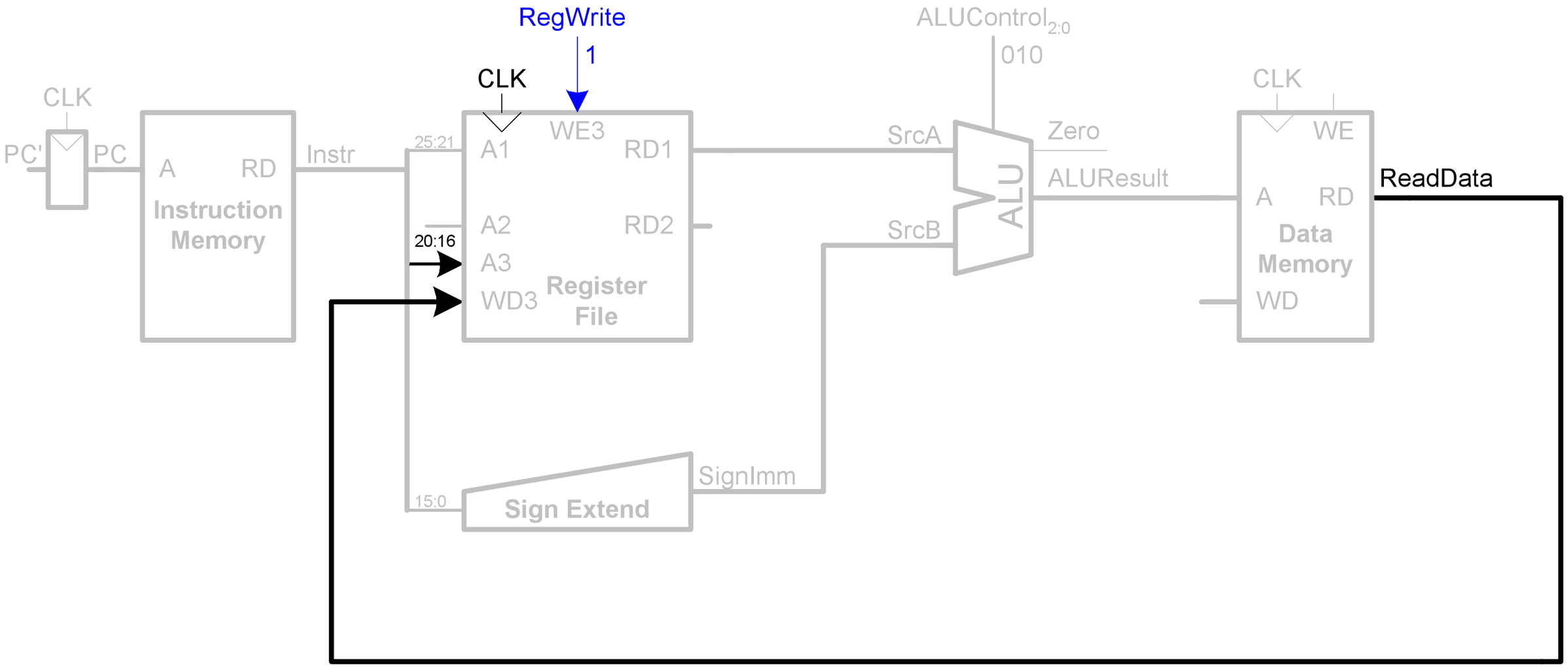
Single-Cycle Datapath: `lw` address

- STEP 4: Compute the memory address



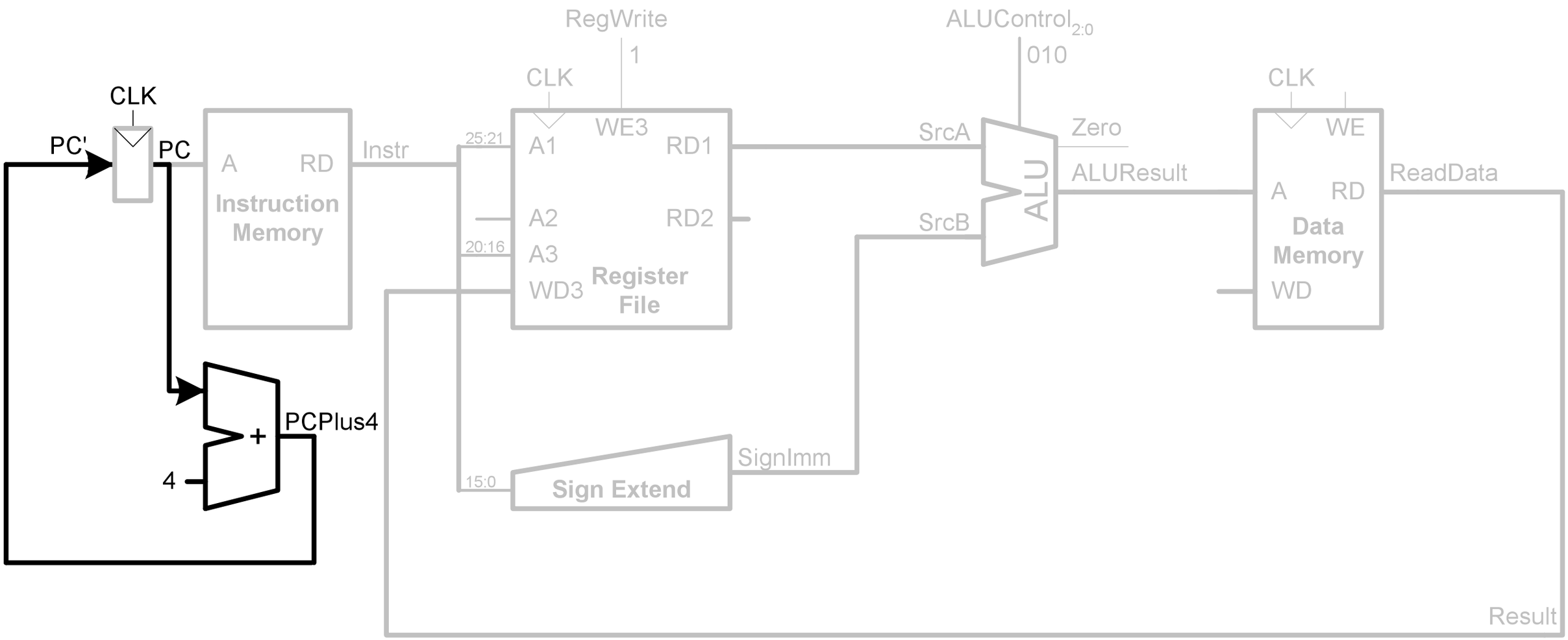
Single-Cycle Datapath: `lw` memory read

- STEP 5: Read data from memory and write it back to register file



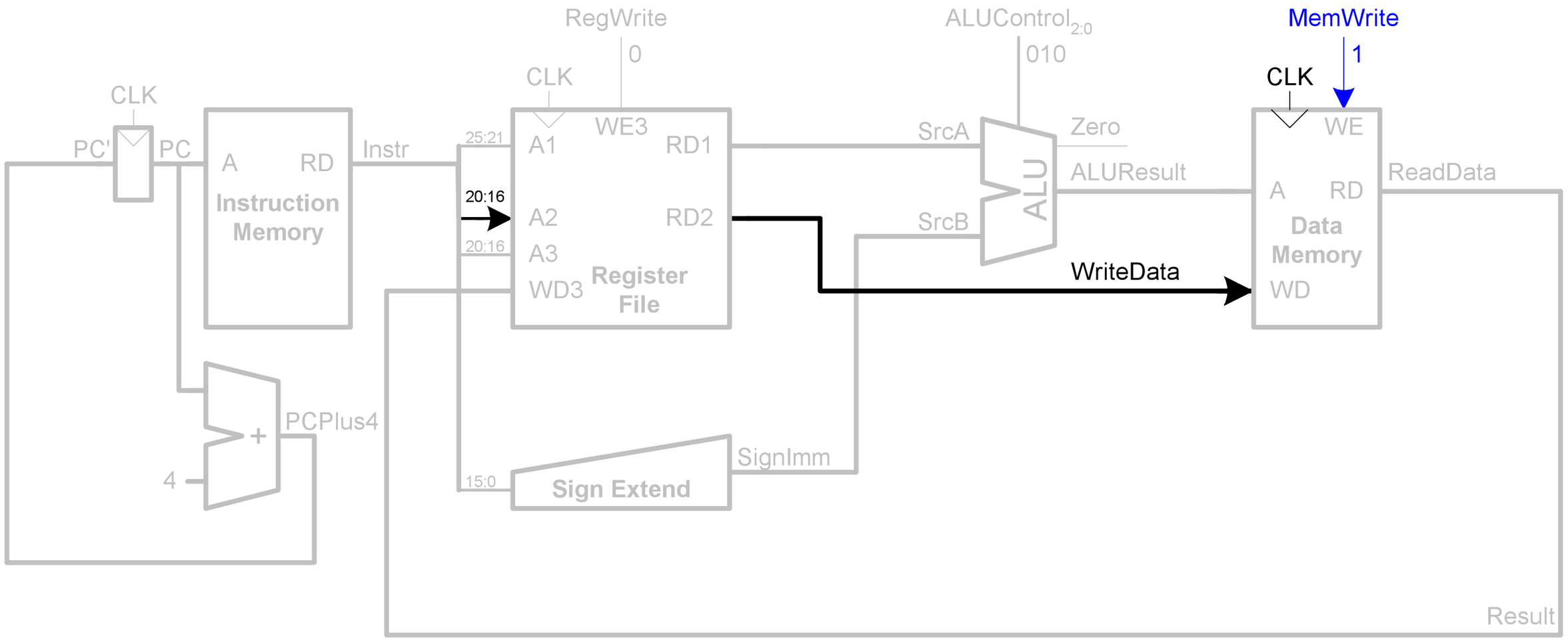
Single-Cycle Datapath: `lw` PC increment

- STEP 6: Determine the address of the next instruction



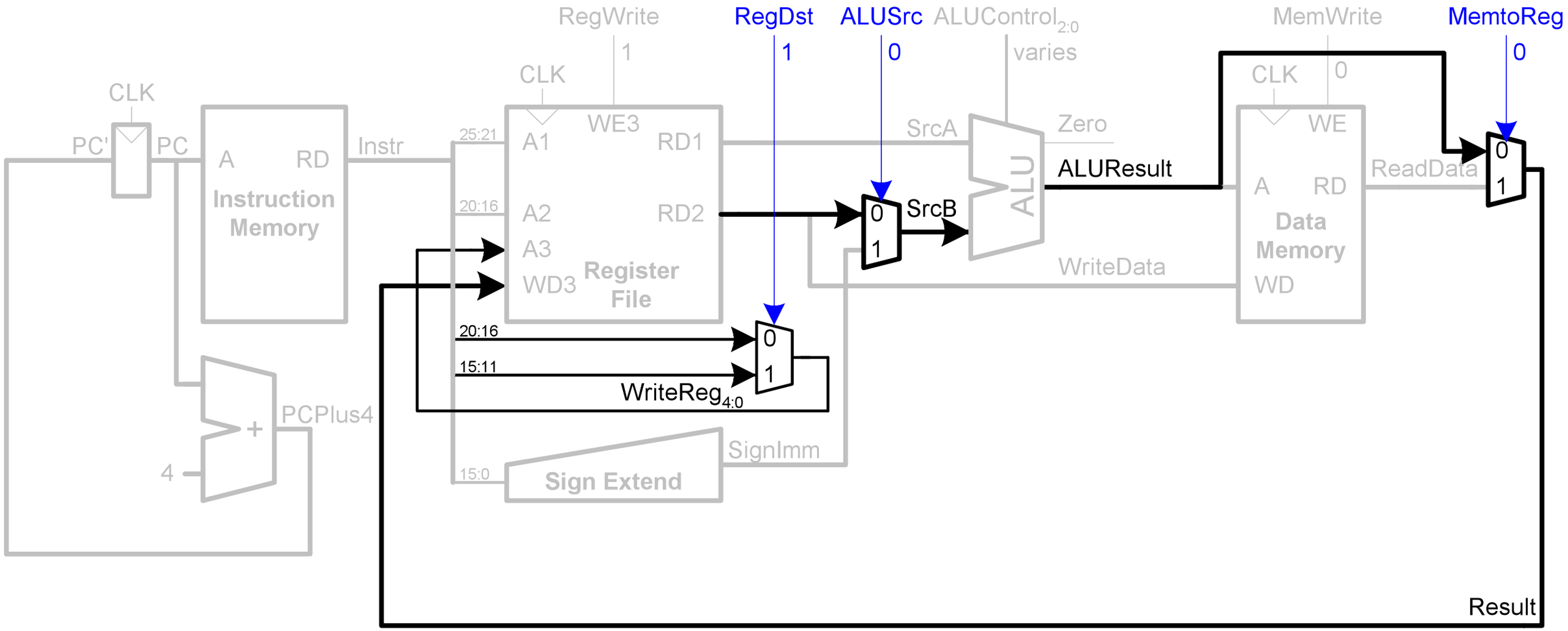
Single-Cycle Datapath: `sw`

- Write data in `rt` to memory



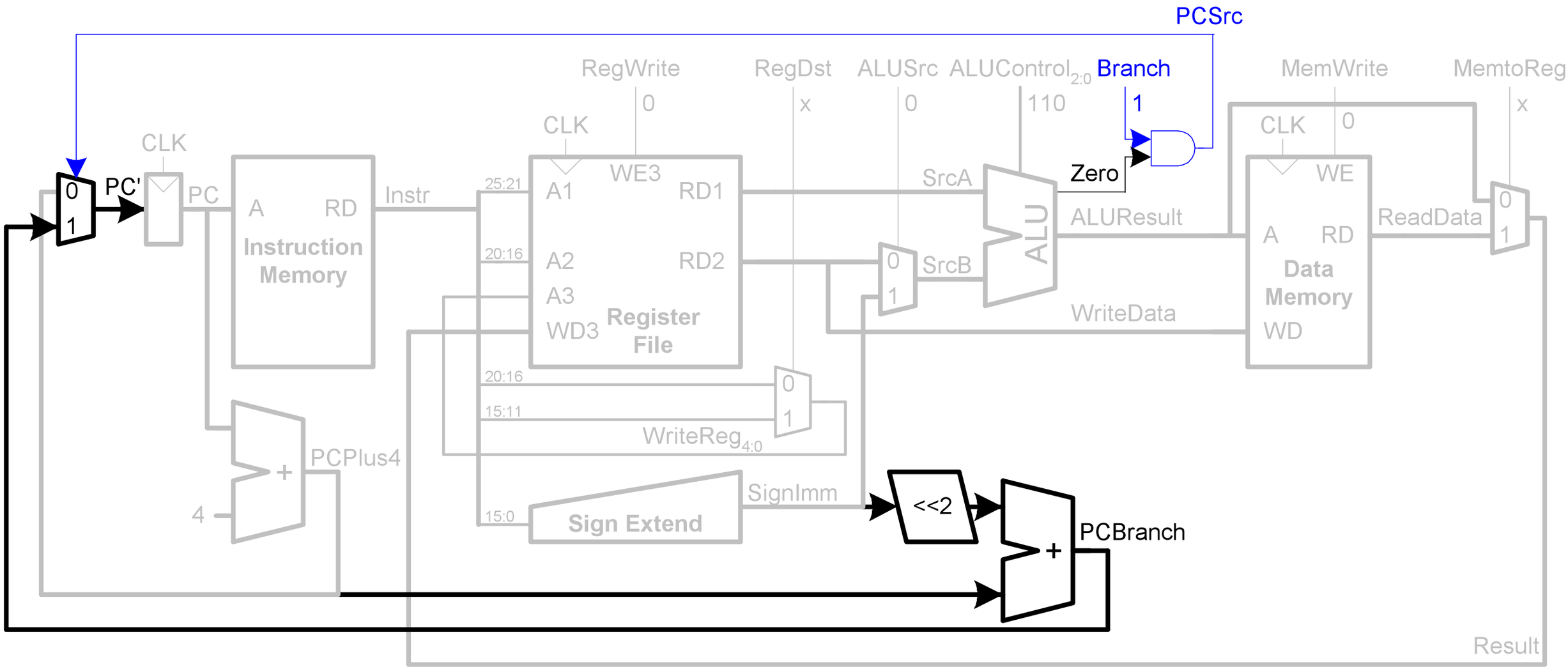
Single-Cycle Datapath: R-type instructions

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

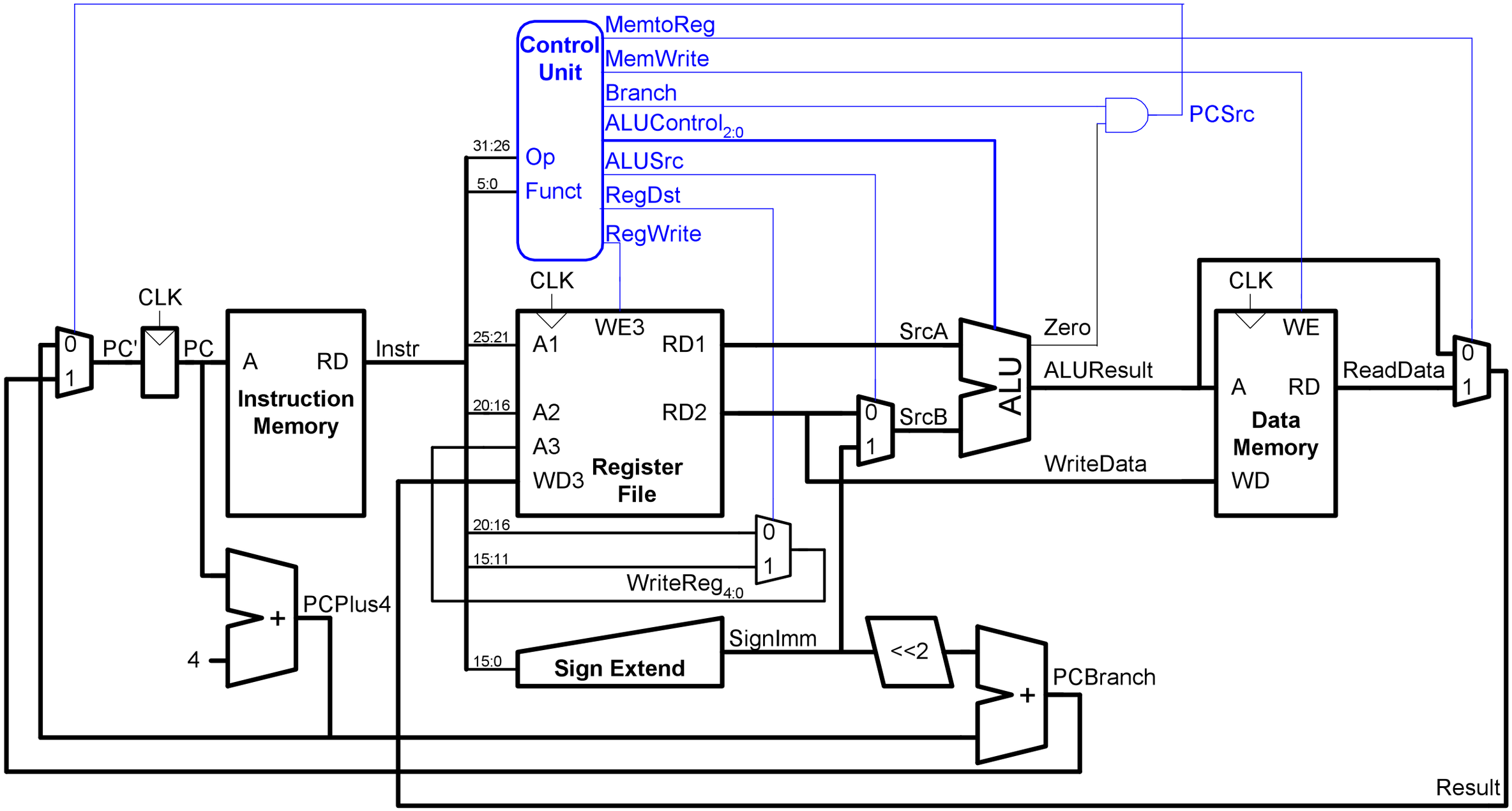


Single-Cycle Datapath: `beq`

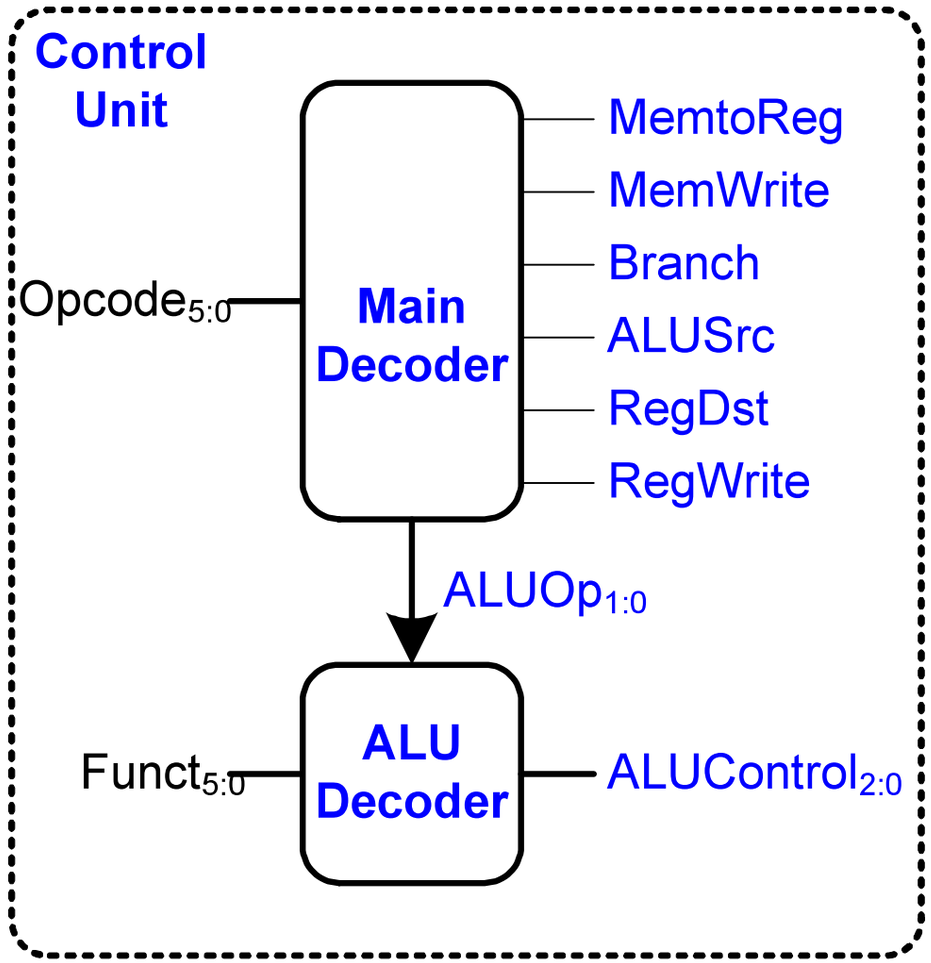
- Determine whether values in `rs` and `rt` are equal
- Calculate branch target address: **BTA = (sign-extended immediate)x4 + (PC+4)**



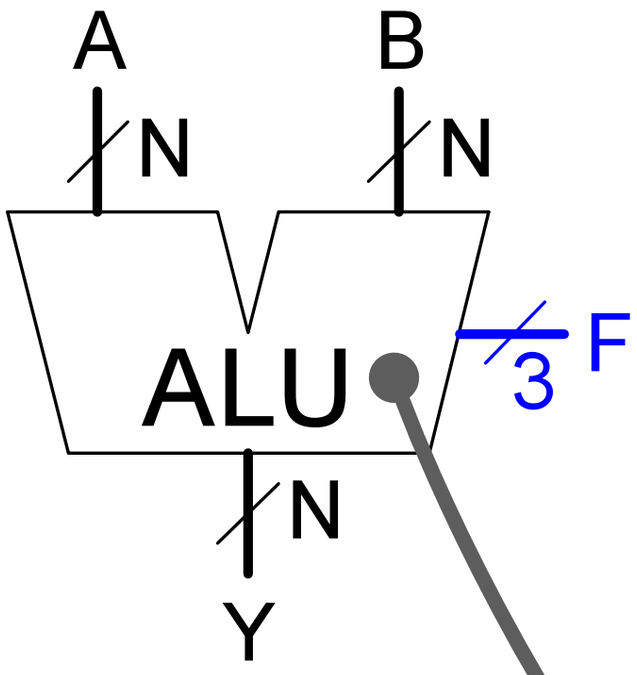
Complete Single-Cycle Processor



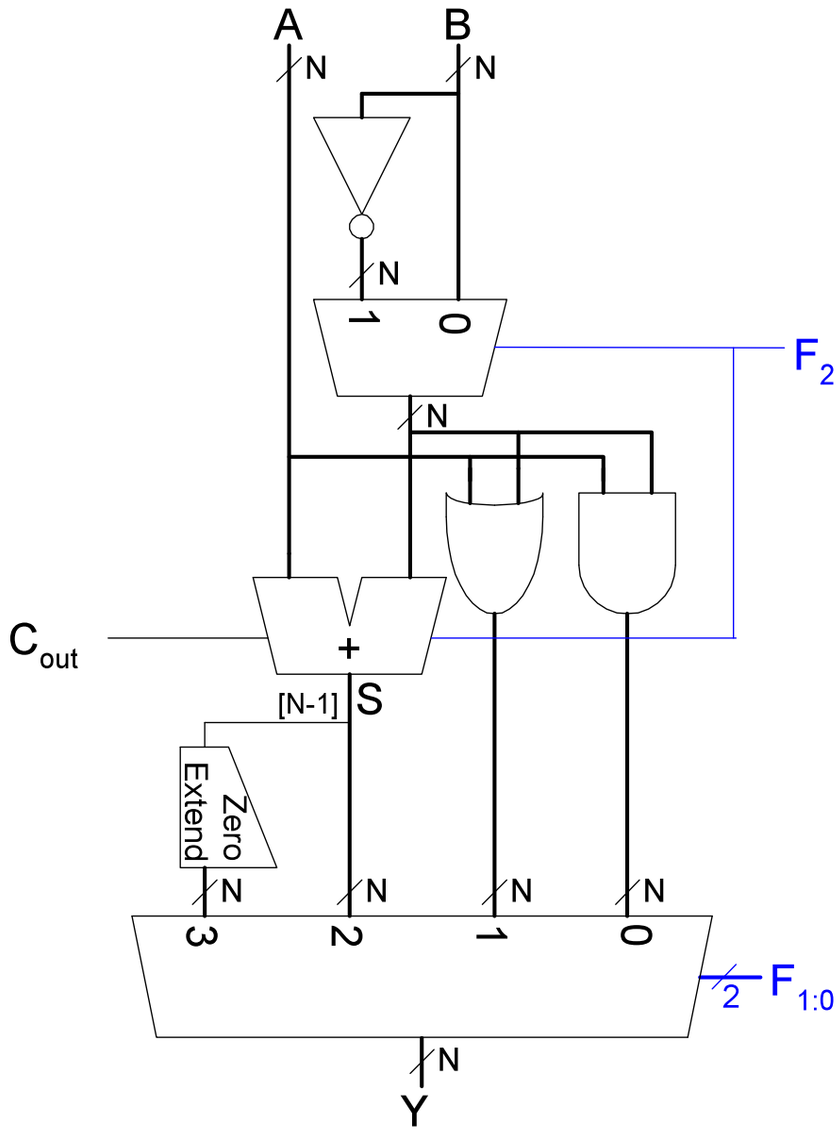
Control Unit



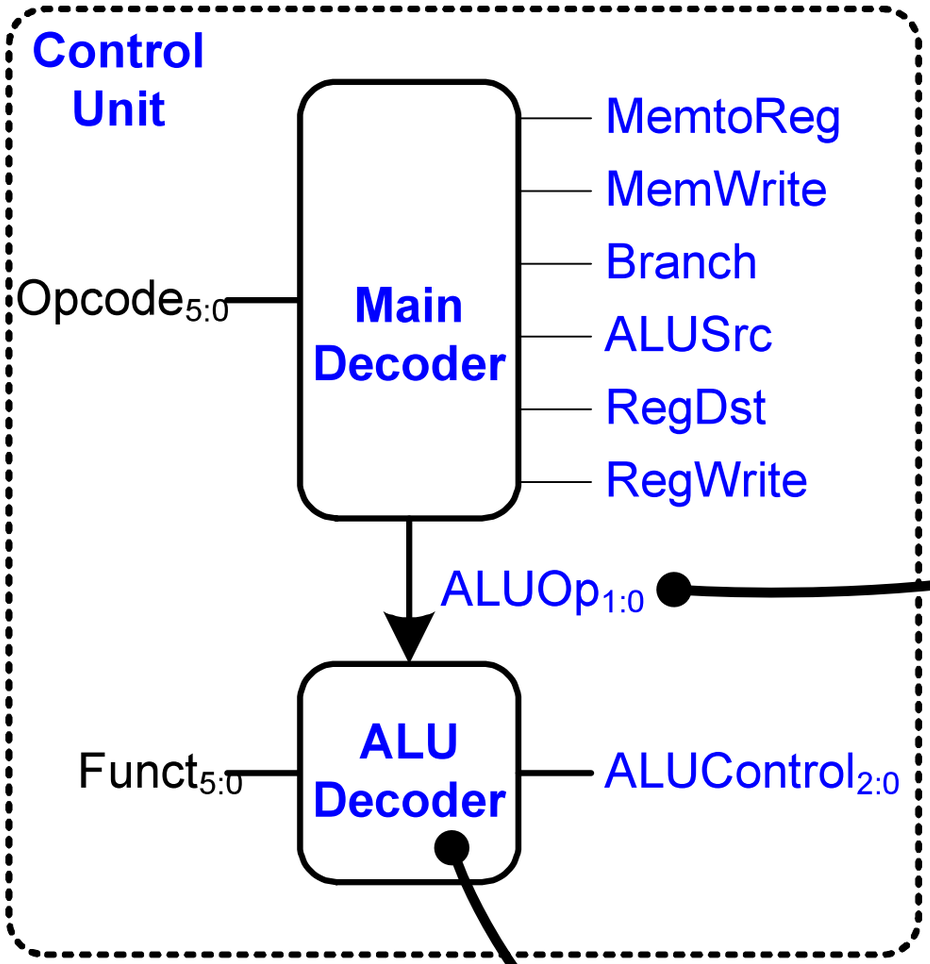
ALU Interface and Implementation



$F_{2:0}$	Function
0 0 0	A & B
0 0 1	A | B
0 1 0	A + B
0 1 1	*not used*
1 0 0	A & ~B
1 0 1	A | ~B
1 1 0	A - B
1 1 1	SLT



Control Unit: ALU Decoder

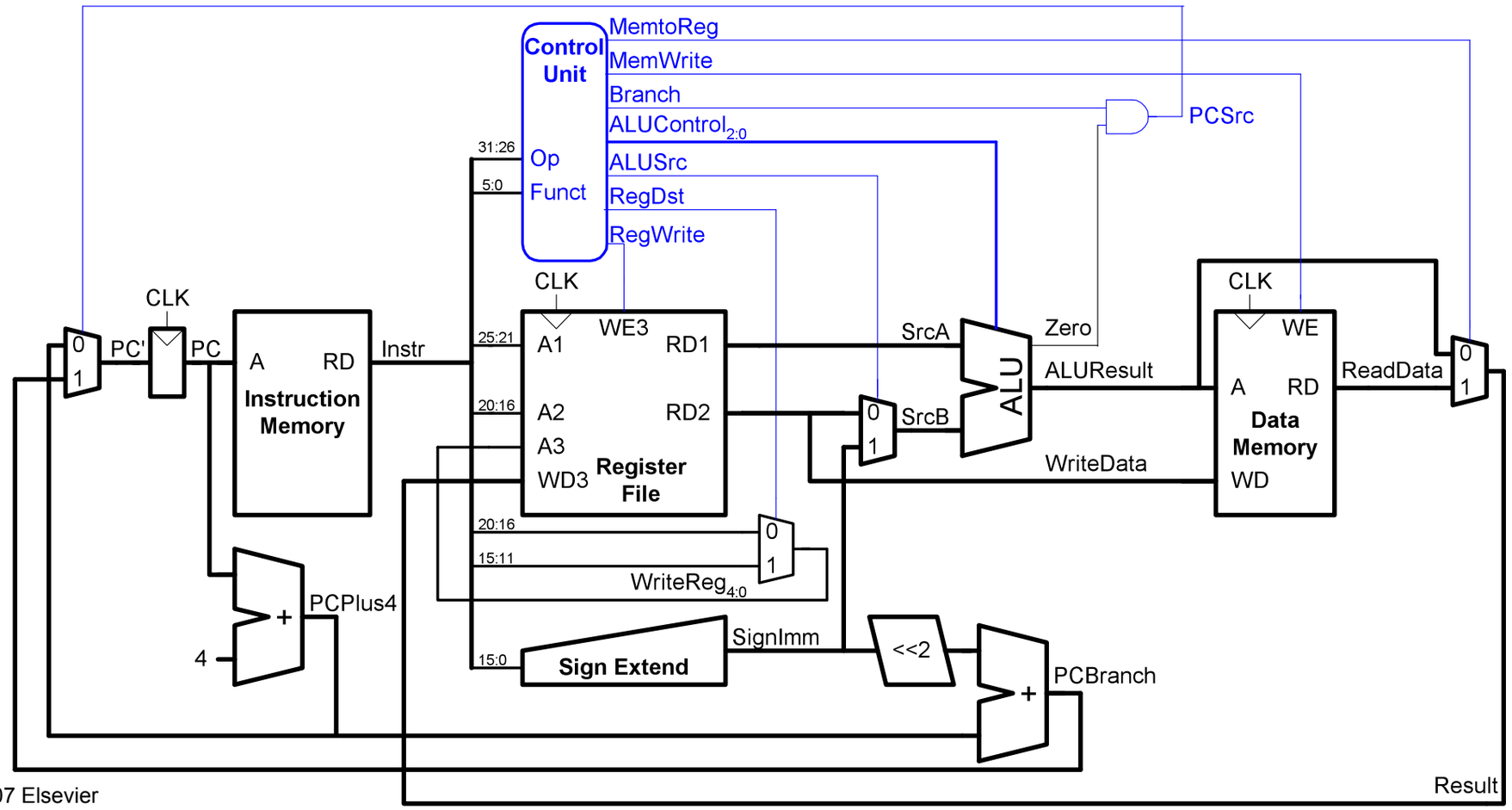


$ALUOp_{1:0}$	Meaning
0 0	Add
0 1	Subtract
1 0	Look at Funct
1 1	*not used*

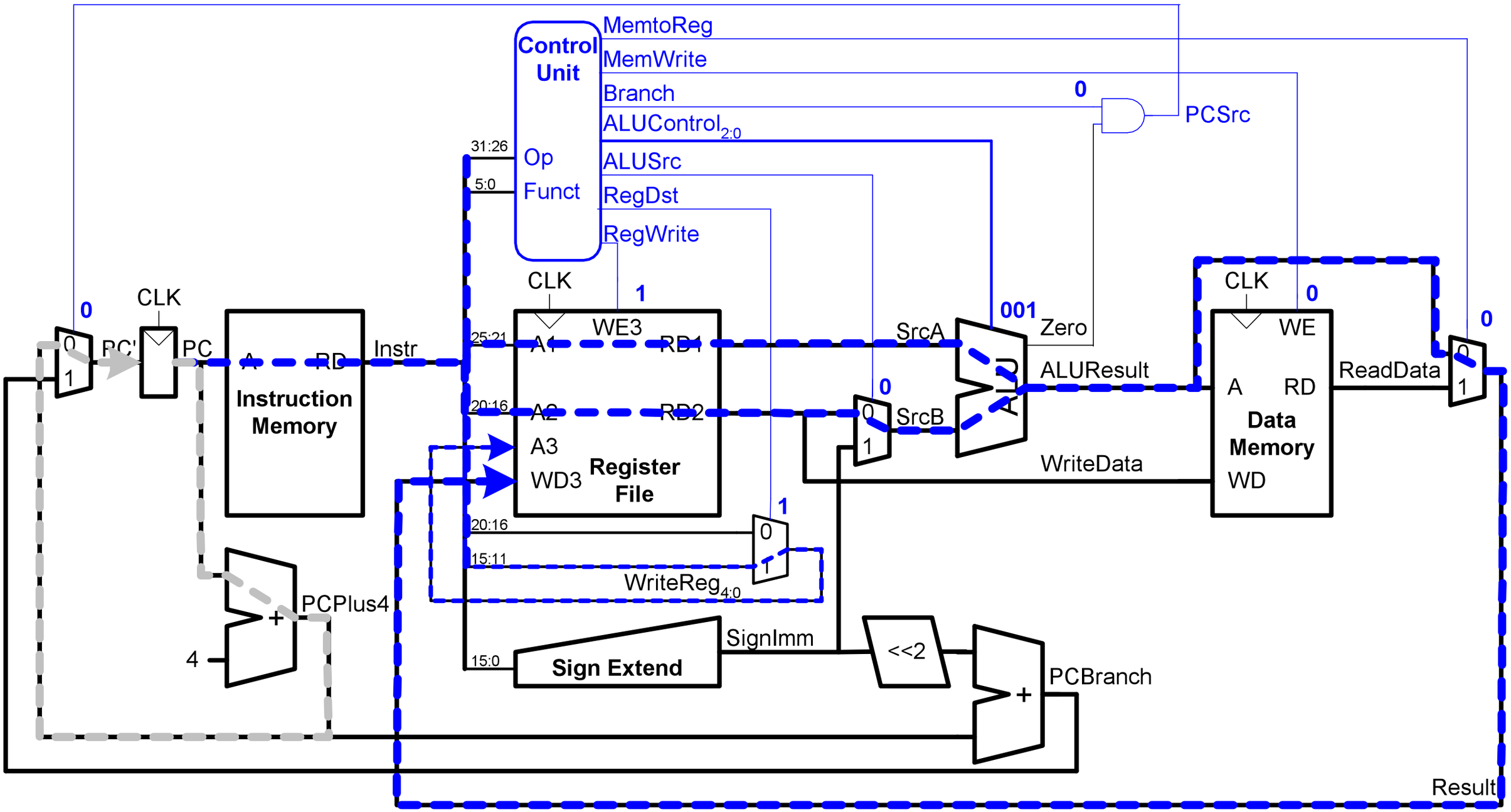
$ALUOp_{1:0}$	Funct	$ALUControl_{2:0}$
0 0	x	010 (Add)
x 1	x	110 (Subtract)
1 x	100000 (add)	010 (Add)
1 x	100010 (sub)	110 (Subtract)
1 x	100100 (and)	000 (And)
1 x	100101 (or)	001 (Or)
1 x	101010 (slt)	111 (SLT)

Control Unit: Main Decoder

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	$ALUOp_{1:0}$
R-type	0 0 0 0 0 0							
`lw`	1 0 0 0 1 1							
`sw`	1 0 1 0 1 1							
`beq`	0 0 0 1 0 0							

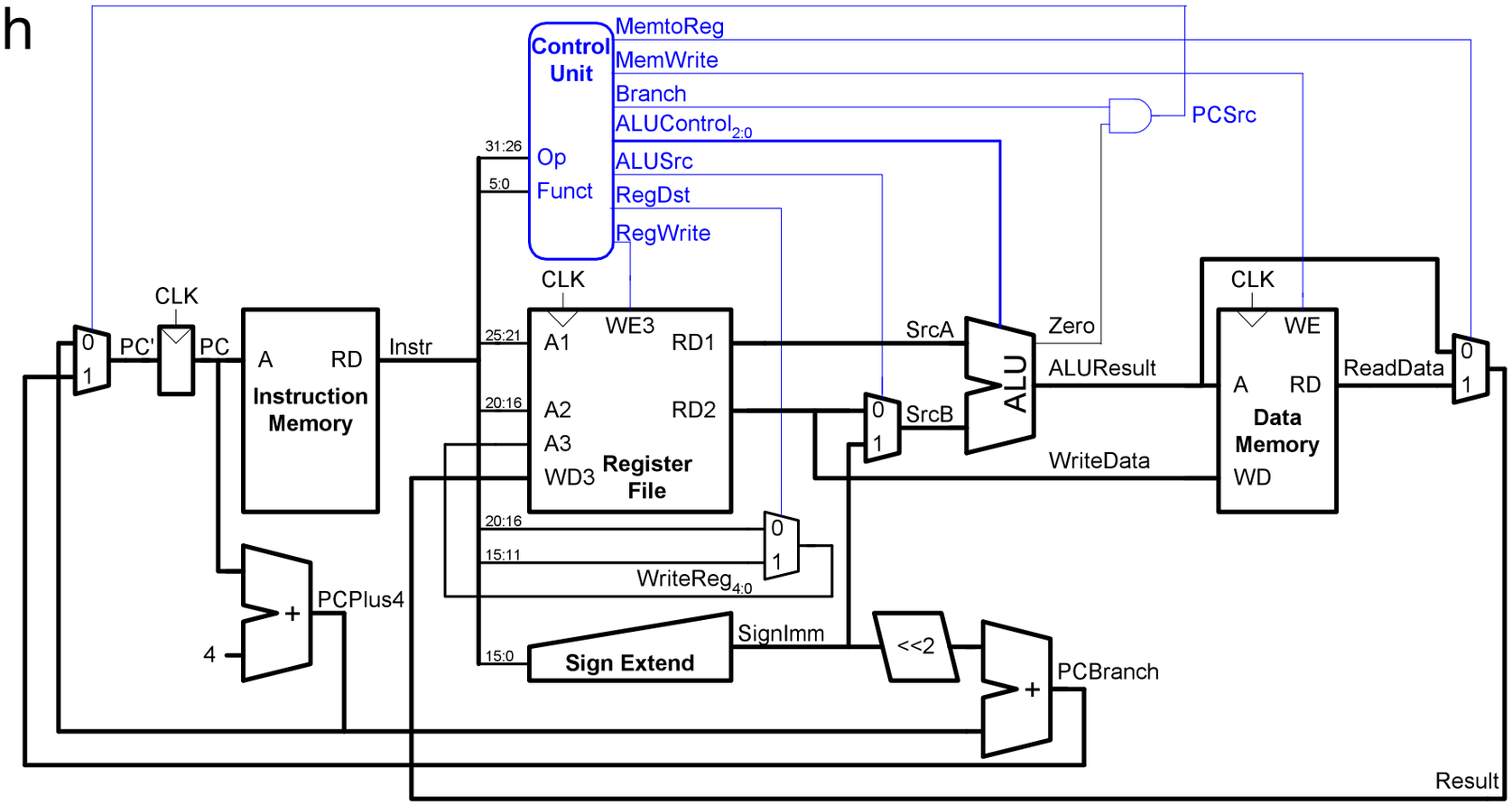


Single-Cycle Datapath Example: `or`



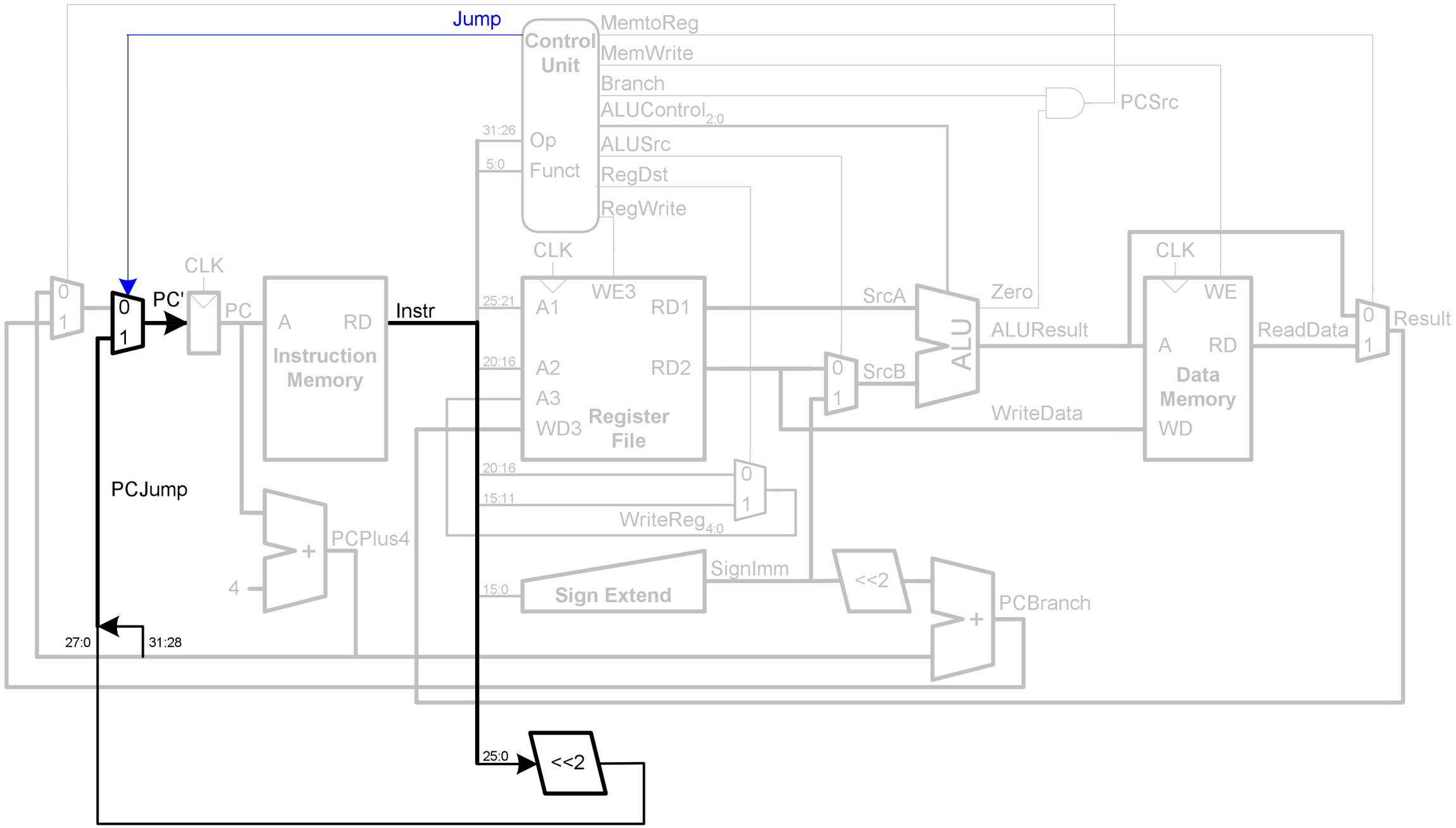
Extended Functionality: `addi`

- No change to datapath



Instruction	Op5:0	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	ALUOp1:0
R-type	0 0 0 0 0 0	1	1	0	0	0	0	1 0
`lw`	1 0 0 0 1 1	1	0	1	0	0	1	0 0
`sw`	1 0 1 0 1 1	0	x	1	0	1	x	0 0
`beq`	0 0 0 1 0 0	0	x	0	1	0	x	0 1
`addi`	0 0 1 0 0 0	1	0	1	0	0	0	0 0

Extended Functionality: `j`



Control Unit: Main Decoder with `j`

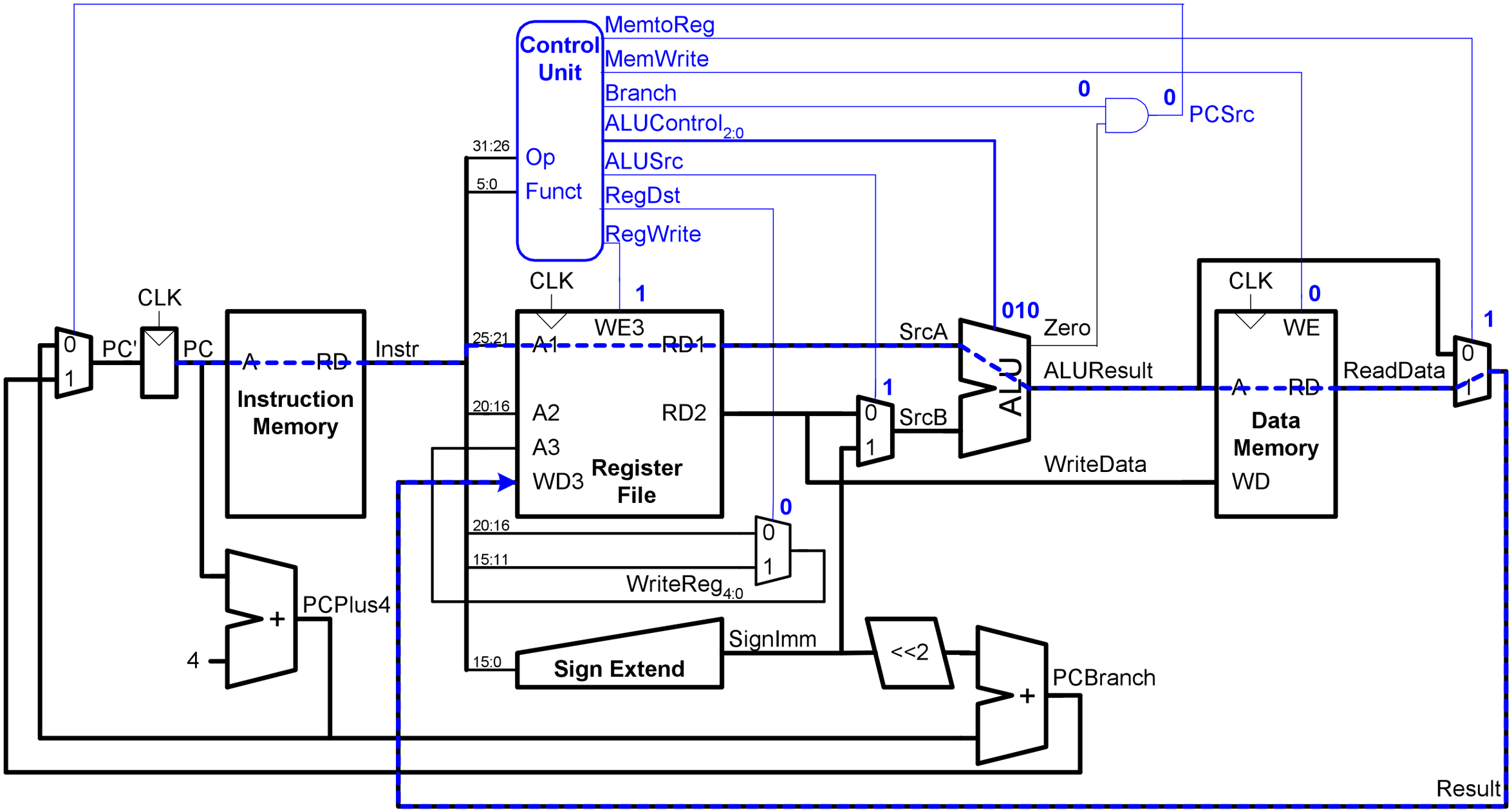
Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	$ALUOp_{1:0}$	Jump
R-type	0 0 0 0 0 0	1	1	0	0	0	0	1 0	0
`lw`	1 0 0 0 1 1	1	0	1	0	0	1	0 0	0
`sw`	1 0 1 0 1 1	0	x	1	0	1	x	0 0	0
`beq`	0 0 0 1 0 0	0	x	0	1	0	x	0 1	0
`addi`	0 0 1 0 0 0	1	0	1	0	0	0	0 0	0
`j`	0 0 0 0 1 0	0	x	x	x	0	x	x x	1

Review: Processor Performance

$$\frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Single-Cycle Performance

- Seconds/cycle (or T_C) is limited by the critical path (`lw`)



Single-Cycle Performance

- Single-cycle critical path:

$$T_c = t_{pcq\_PC} + t_{\text{mem}} + \max(t_{RF\text{read}}, t_{sext} + t_{\text{mux}}) + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$$

- In most implementations, limiting paths are:
  - memory, ALU, register file
  - $T_c = t_{pcq\_PC} + 2t_{\text{mem}} + t_{RF\text{read}} + t_{\text{mux}} + t_{\text{ALU}} + t_{RF\text{setup}}$

Single-Cycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	$t_{pcq\ \mathrm{PC}}$	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	$t_{RF\mathrm{read}}$	150
Register file setup	$t_{RF\mathrm{setup}}$	20

$$
\begin{aligned}
T_c &= t_{pcq\_PC} + 2t_{\mathrm{mem}} + t_{RF\mathrm{read}} + t_{\mathrm{mux}} + t_{\mathrm{ALU}} + t_{RF\mathrm{setup}} \\
&= [30 + 2(250) + 150 + 25 + 200 + 20]\ \mathrm{ps} \\
&= 925\ \mathrm{ps}
\end{aligned}
$$

Single-Cycle Performance Example

- For a program with 100 billion instructions executing on a single-cycle MIPS processor,

Execution Time = # instructions x CPI x TC

= (100 × 109)(1)(925 × 10-12 s)

= 92.5 seconds