MICROARCHITECTURE

Chapter 7

***Digital Design and Computer Architecture*, 2nd Edition**

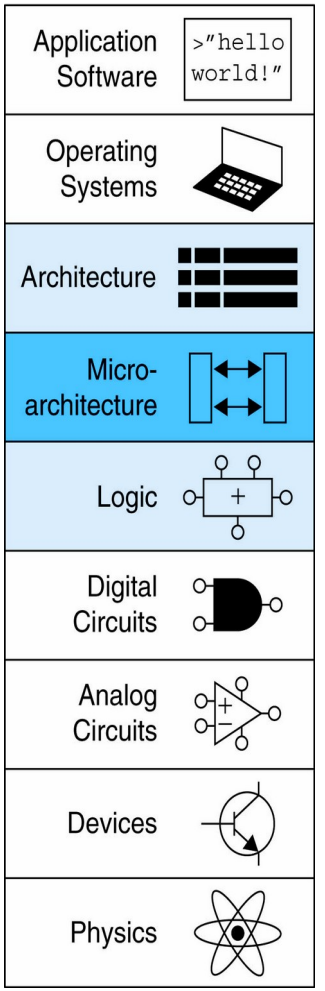
David Money Harris and Sarah L. Harris

ELSEVIER

MICROARCHITECTURE

Chapter 7 :: Topics

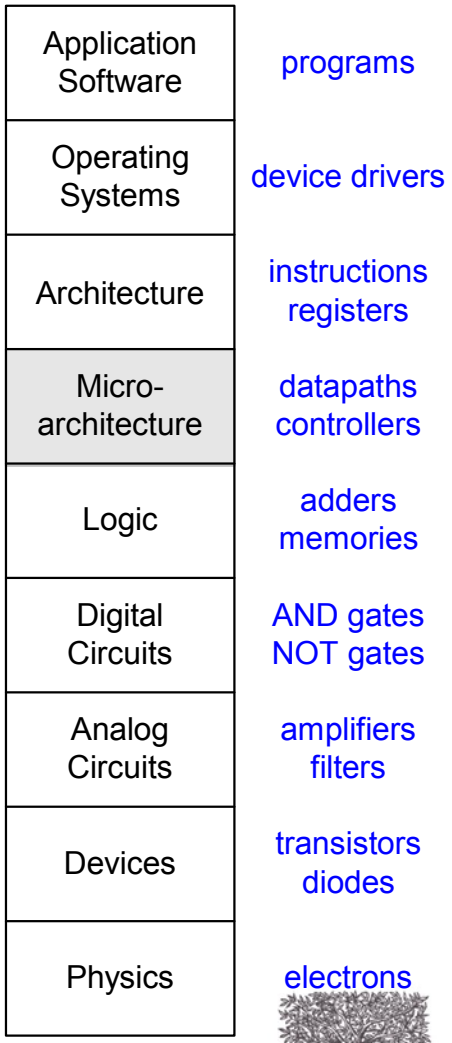
- Introduction
- State elements of MIPS Processor
- Design Process & Performance Analysis of Single Cycle Processor
- Single-Cycle Data path
- Single cycle control for R-type A/L instrns
- Design Process & Performance Analysis of Multi cycle Processor
- Multicycle Data path
- Multi cycle control for R-type A/L instrns



ELSEVIER

Introduction

- **Microarchitecture:** how to implement an architecture in hardware.
- Processor:
  - **Datapath(32 bit):** functional blocks
    - ✓ memories,registers,ALUs,mux
  - **Control:** control signals
  - Receives current instruction from datapath & tells datapath how to execute that instruction.
    - ✓ Mux select,register enable,memory write signals
- Computer Architecture:
  - Instruction set
  - Architectural state
    - ✓ PC and 32 registers



Microarchitecture

- Multiple implementations for a single architecture:
  - **Single-cycle:** Each instruction executes in a single cycle.no nonarchitectural state.cycle time is limited by the slowest instruction.
  - **Multicycle:** Each instruction is broken into series of shorter cycles.reduces hardware cost by reusing expensive hardware blocks adders/memories.cycle time limited by slowest functional unit.

Processor Performance

- Performance measured by program execution time

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

- Definitions:
  - CPI: Cycles/instruction
  - clock period: seconds/cycle
  - IPC: instructions/cycle = IPC
- Challenge is to satisfy constraints of:
  - Cost
  - Power
  - Performance

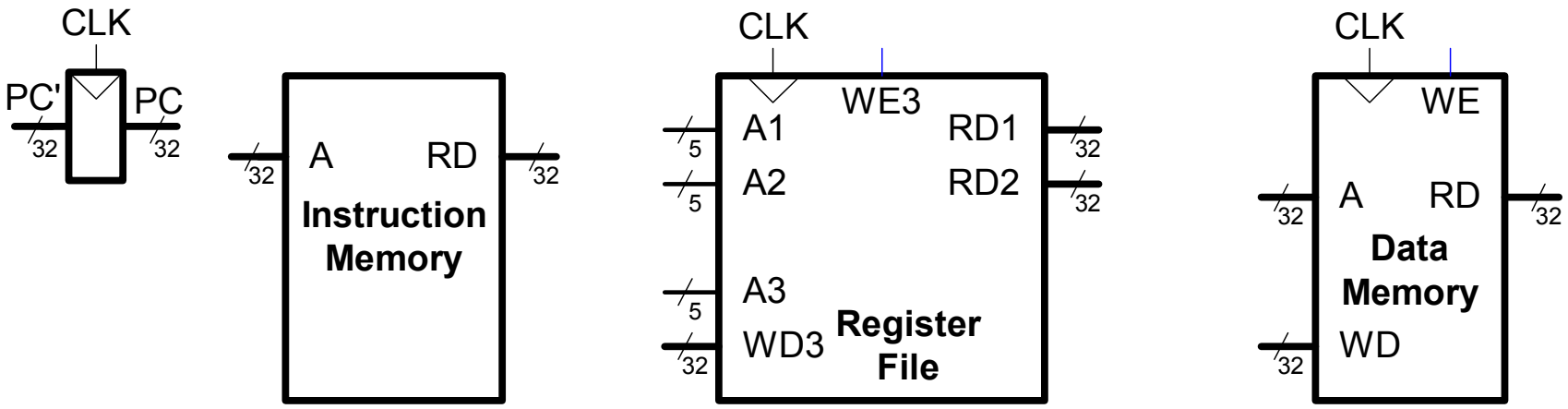
MIPS Processor

- Consider subset of MIPS instructions:
  - R-type instructions: `and`, `or`, `add`, `sub`, `slt`
  - Memory instructions: `lw`, `sw`
  - Branch instructions: `beq`

Architectural State

- Determines everything about a processor:
  - PC
  - 32 registers
  - Memory

MIPS State Elements

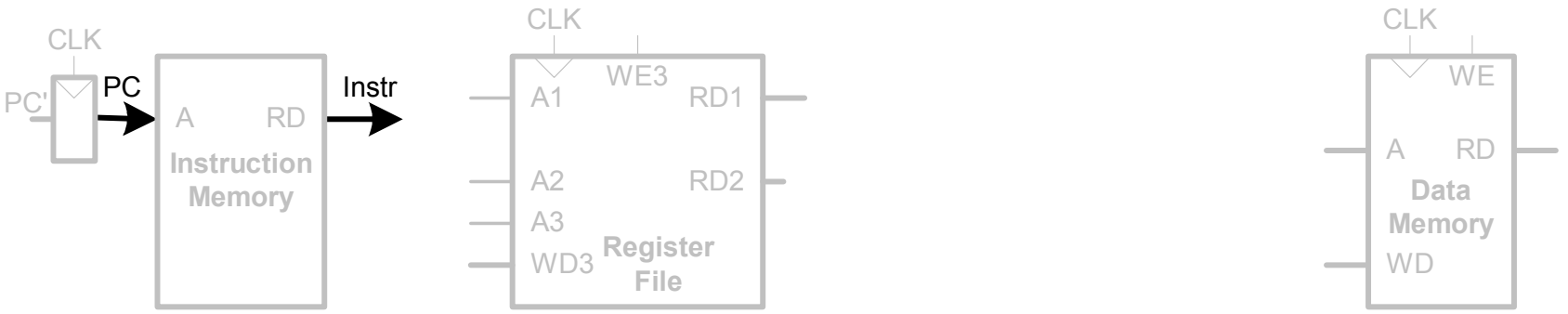


Single-Cycle MIPS Processor

- Datapath
- Control

Single-Cycle Datapath: `lw` fetch

STEP 1: Fetch instruction



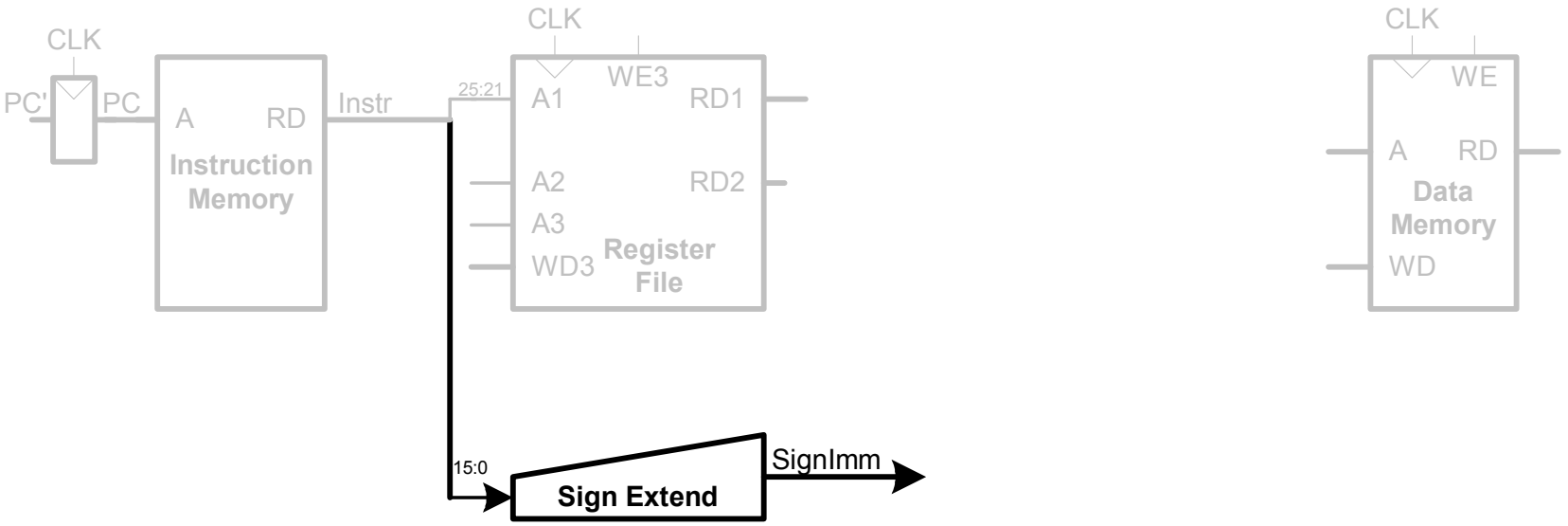
A1-rs

A2-rt(sw)

A3-rt(lw)

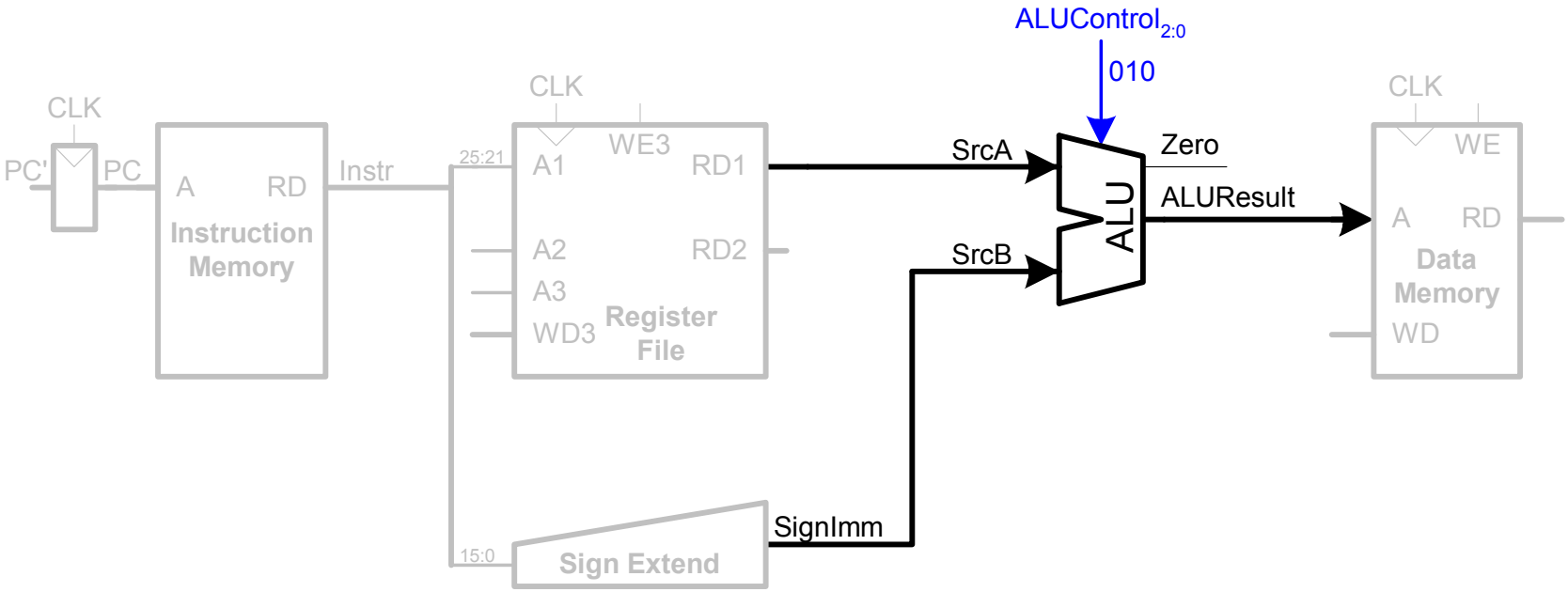
Single-Cycle Datapath: `lw` Immediate

STEP 3: Sign-extend the immediate



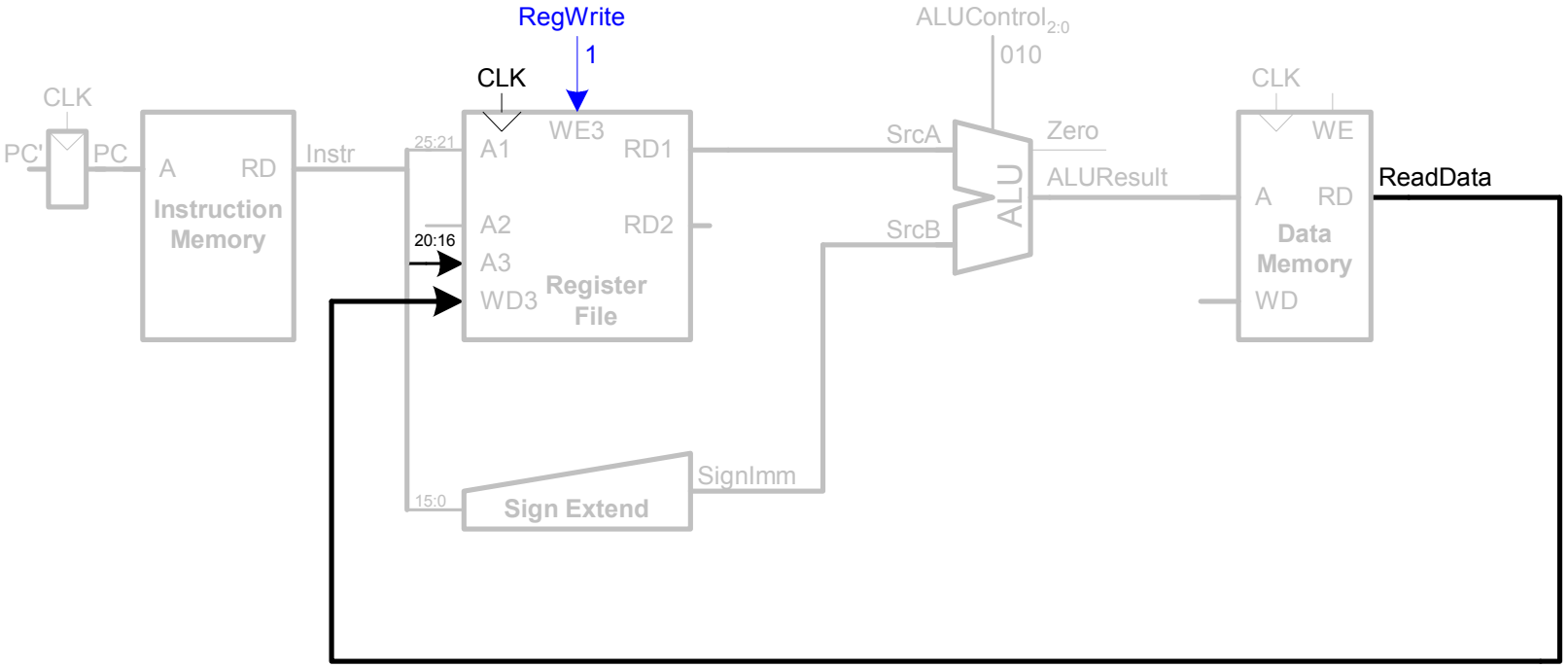
Single-Cycle Datapath: `lw` address

STEP 4: Compute the memory address



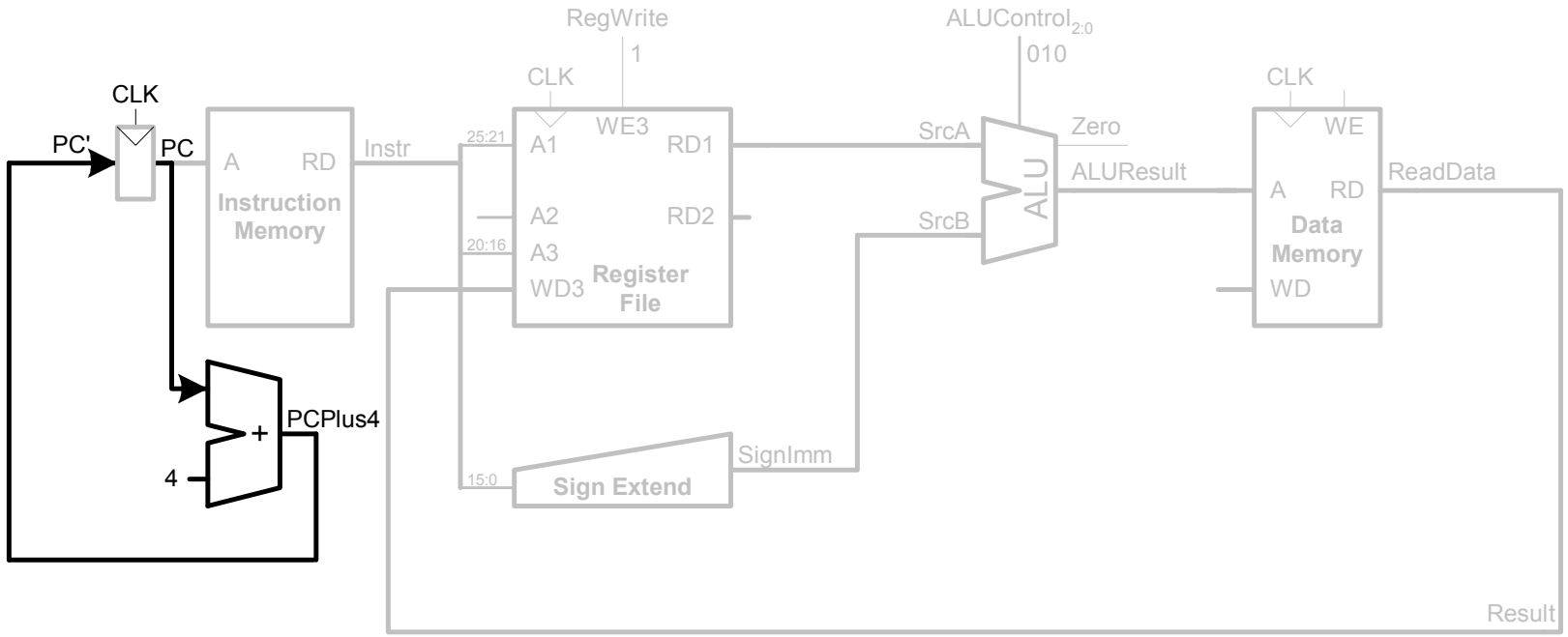
Single-Cycle Datapath: `lw` Memory Read

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



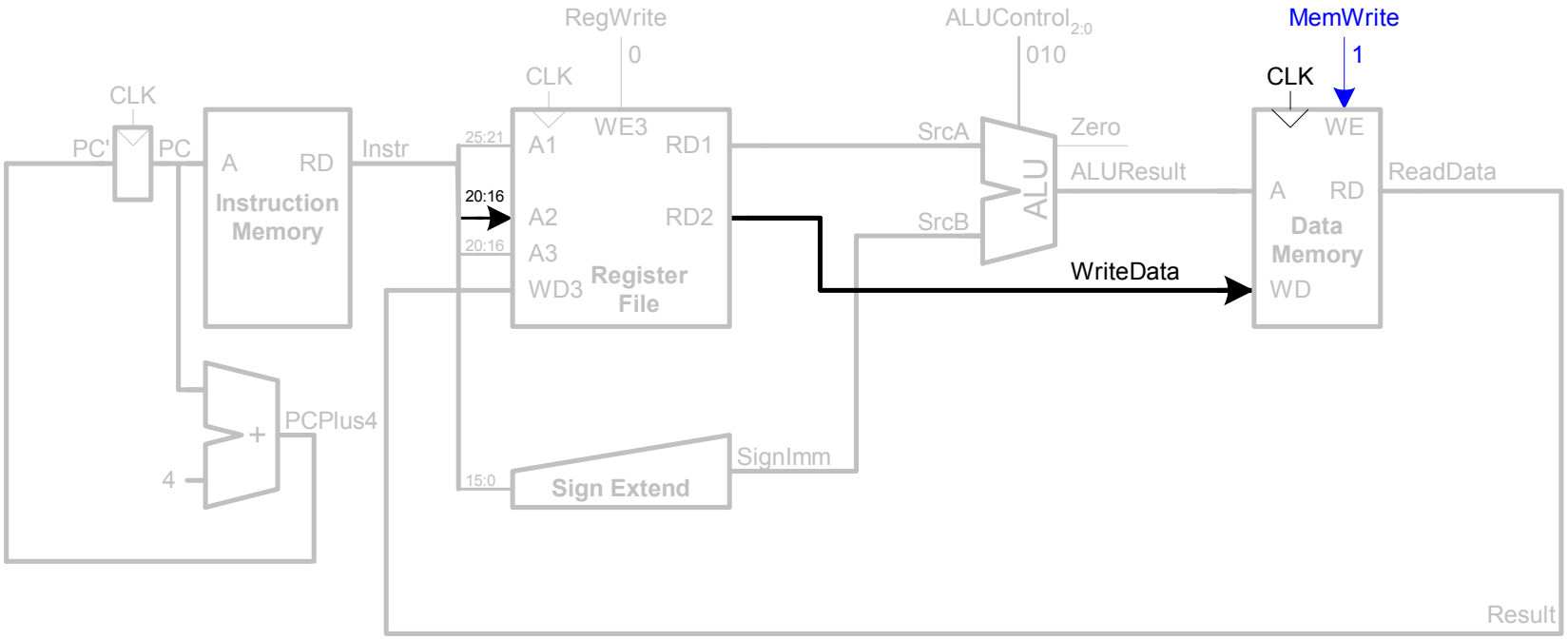
Single-Cycle Datapath: `lw` PC Increment

STEP 6: Determine address of next instruction



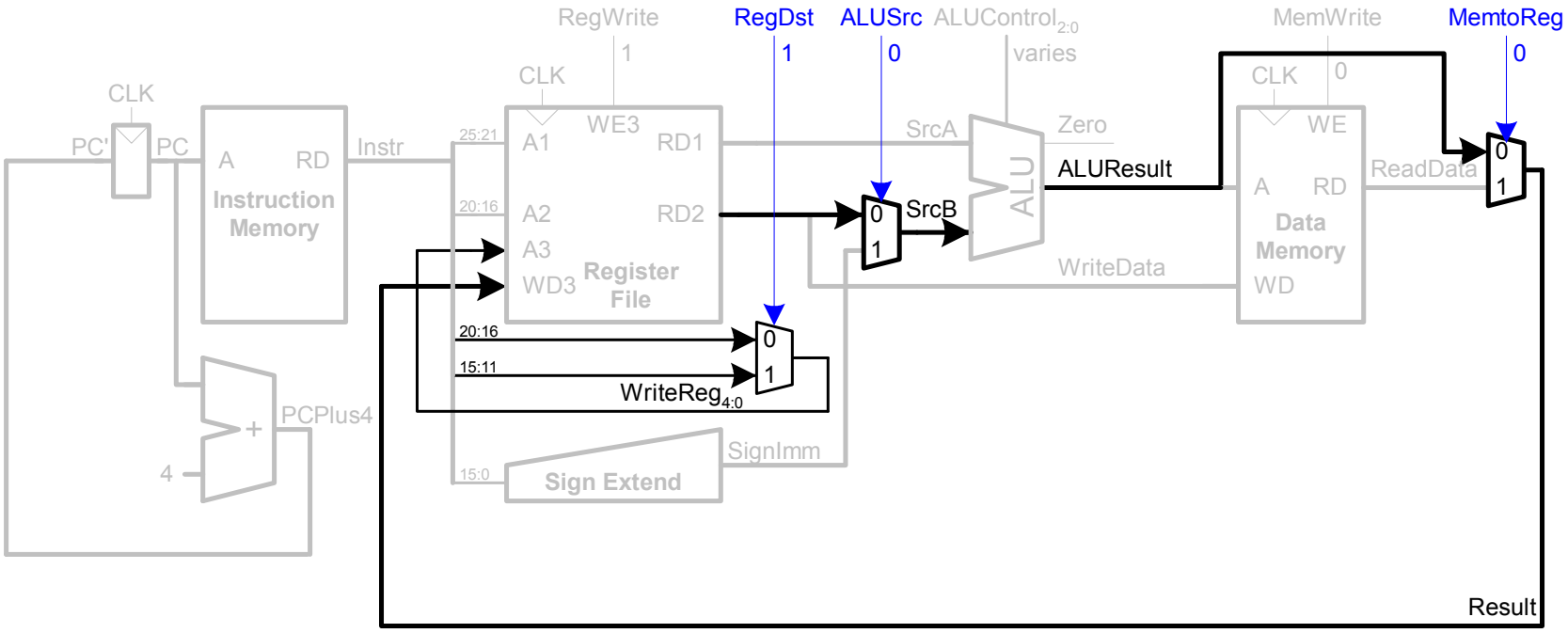
Single-Cycle Datapath: `sw`

Write data in `rt` to memory



Single-Cycle Datapath: R-Type

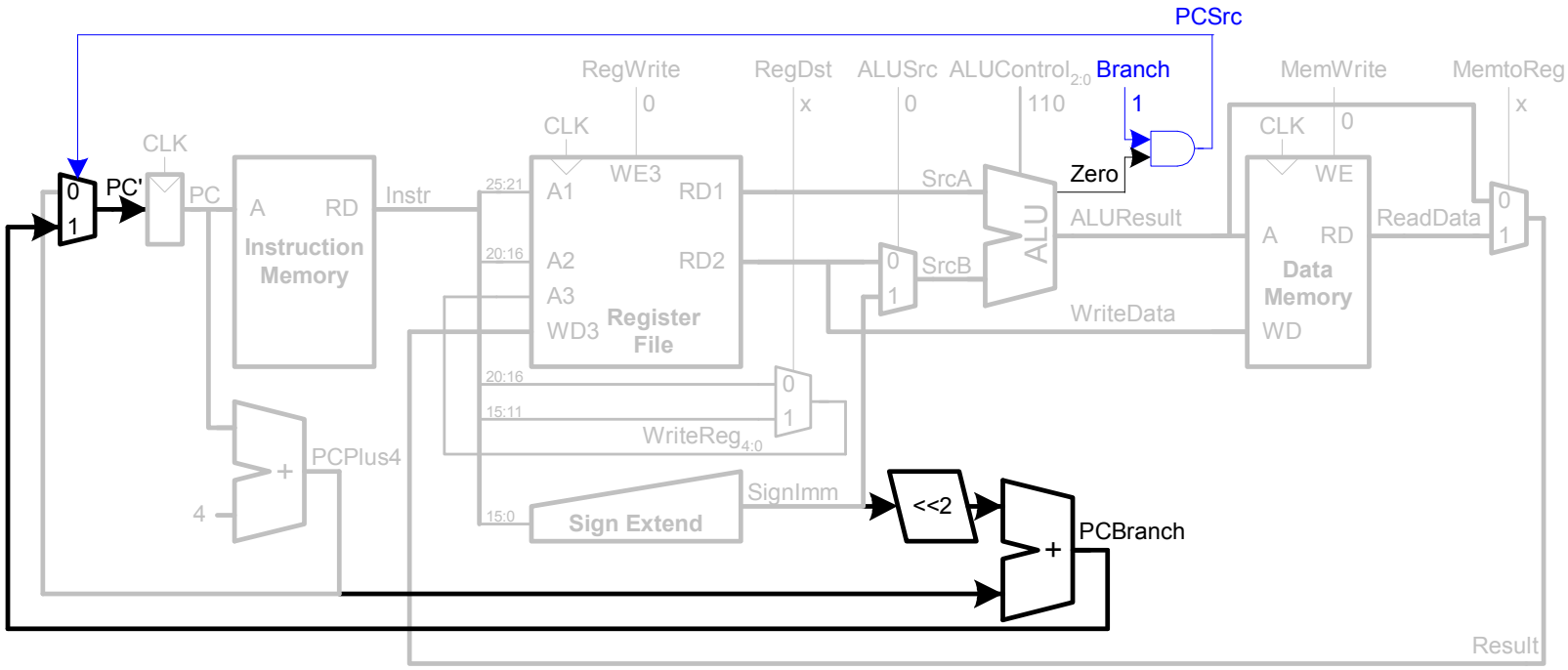
- Read from `rs` and `rt`
- Write *ALUResult* to register file
- Write to `rd` (instead of `rt`)



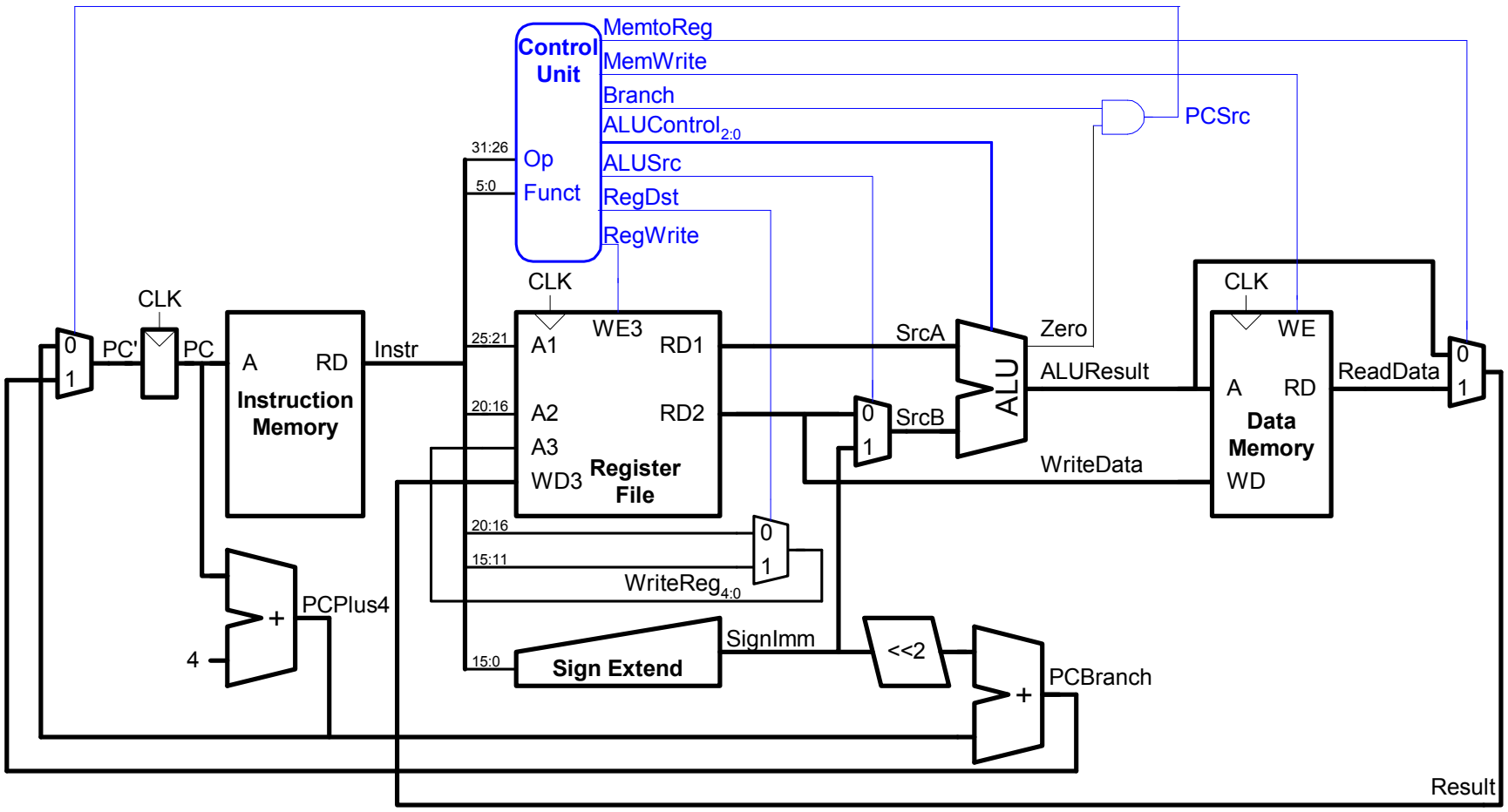
Single-Cycle Datapath: `beq`

- Determine whether values in `rs` and `rt` are equal
- Calculate branch target address:

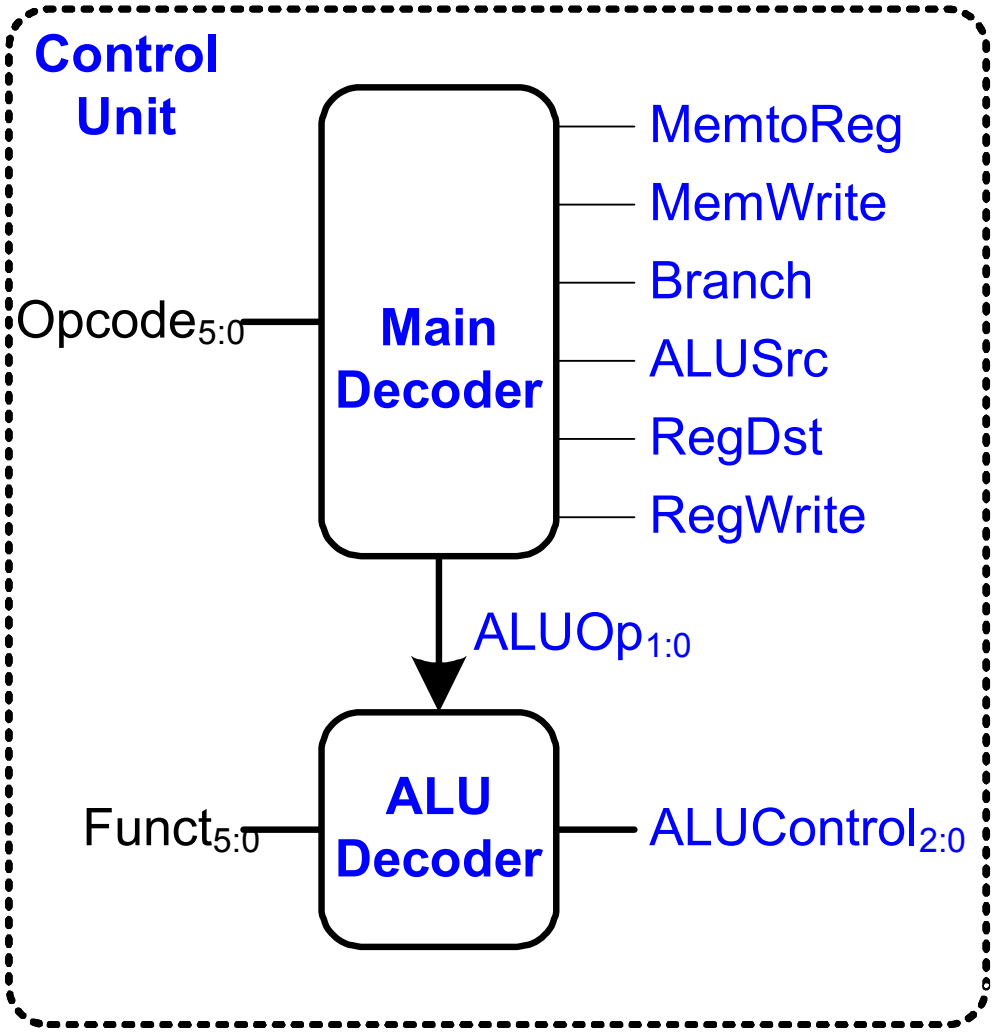
  BTA = (sign-extended immediate << 2) + (PC+4)



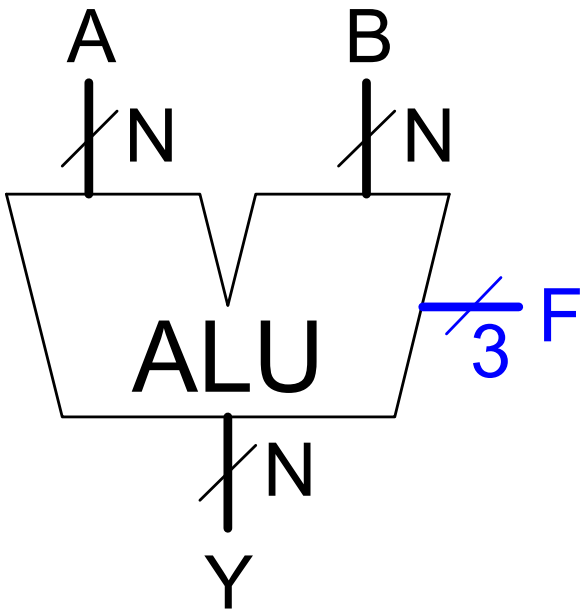
Single-Cycle Processor



Single-Cycle Control

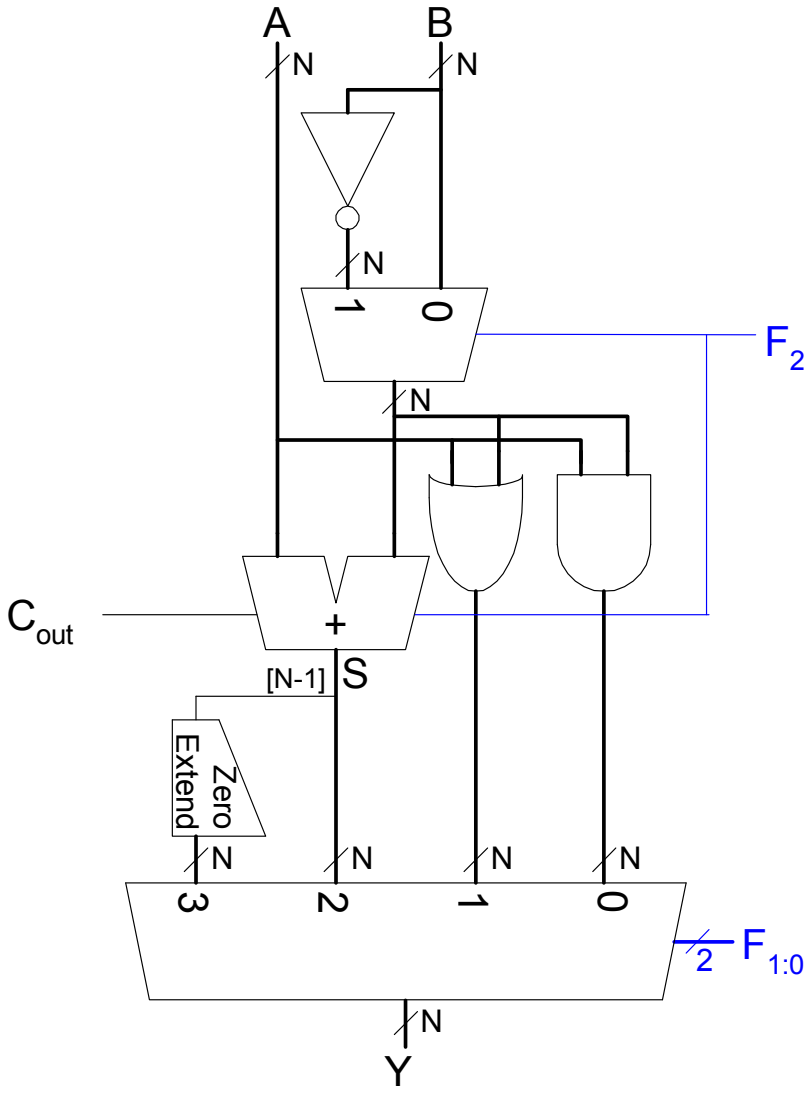


Review: ALU



$F_{2:0}$	Function
000	A & B
001	A | B
010	A + B
011	not used
100	A & ~B
101	A | ~B
110	A - B
111	SLT

Review: ALU



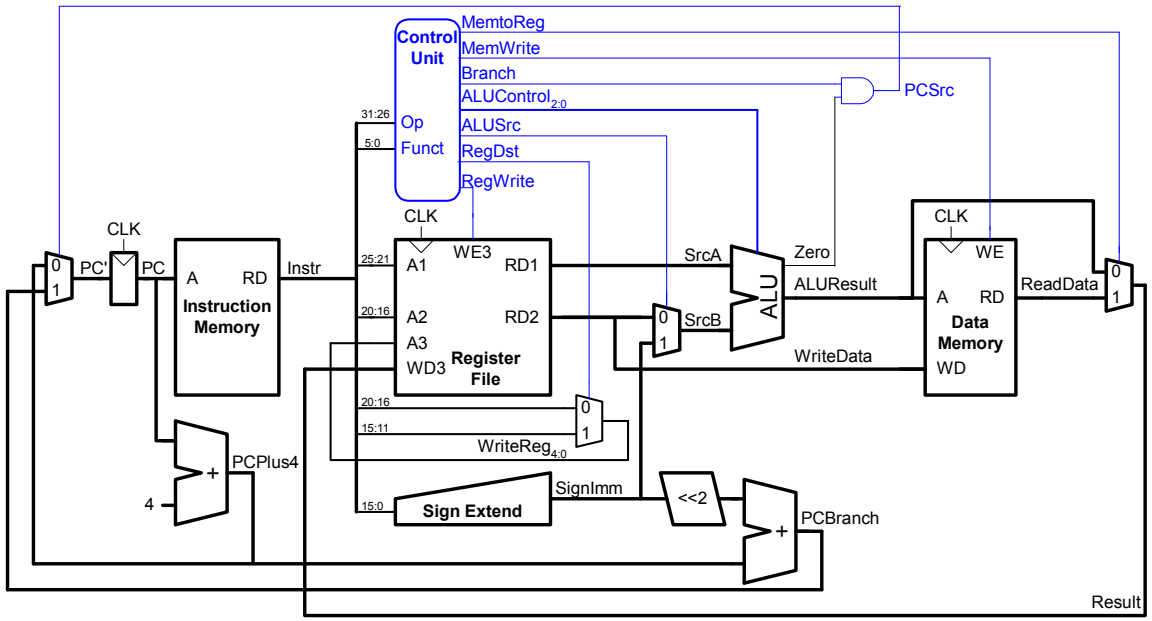
Control Unit: ALU Decoder

$ALUOp_{1:0}$	Meaning
00	Add
01	Subtract
10	Look at Funct
11	Not Used

$ALUOp_{1:0}$	Funct	$ALUControl_{2:0}$
00	X	010 (Add)
X1	X	110 (Subtract)
1X	100000 (`add`)	010 (Add)
1X	100010 (`sub`)	110 (Subtract)
1X	100100 (`and`)	000 (And)
1X	100101 (`or`)	001 (Or)
1X	101010 (`slt`)	111 (SLT)

Control Unit Main Decoder

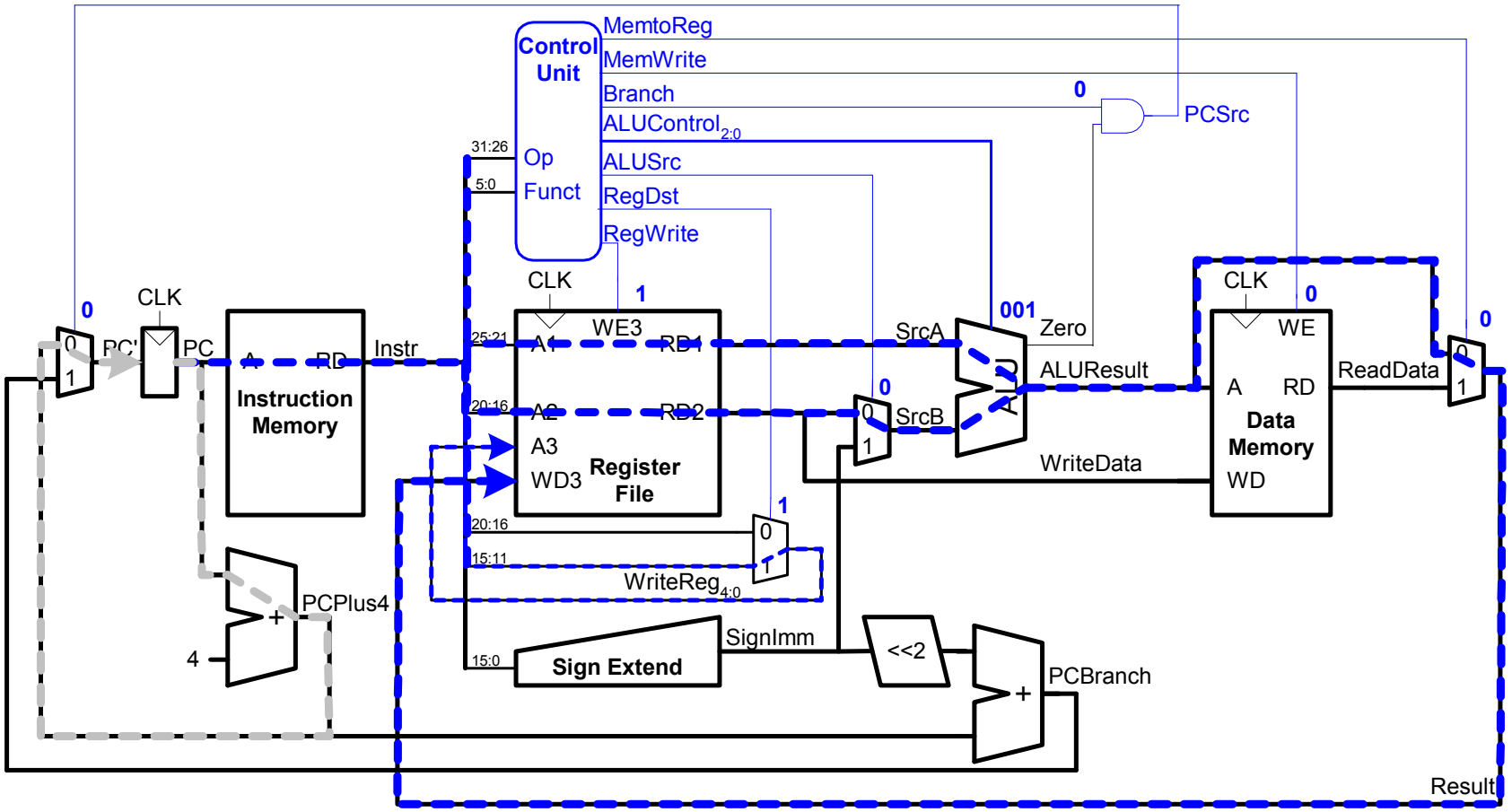
Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000							
`lw`	100011							
`sw`	101011							
`beq`	000100							



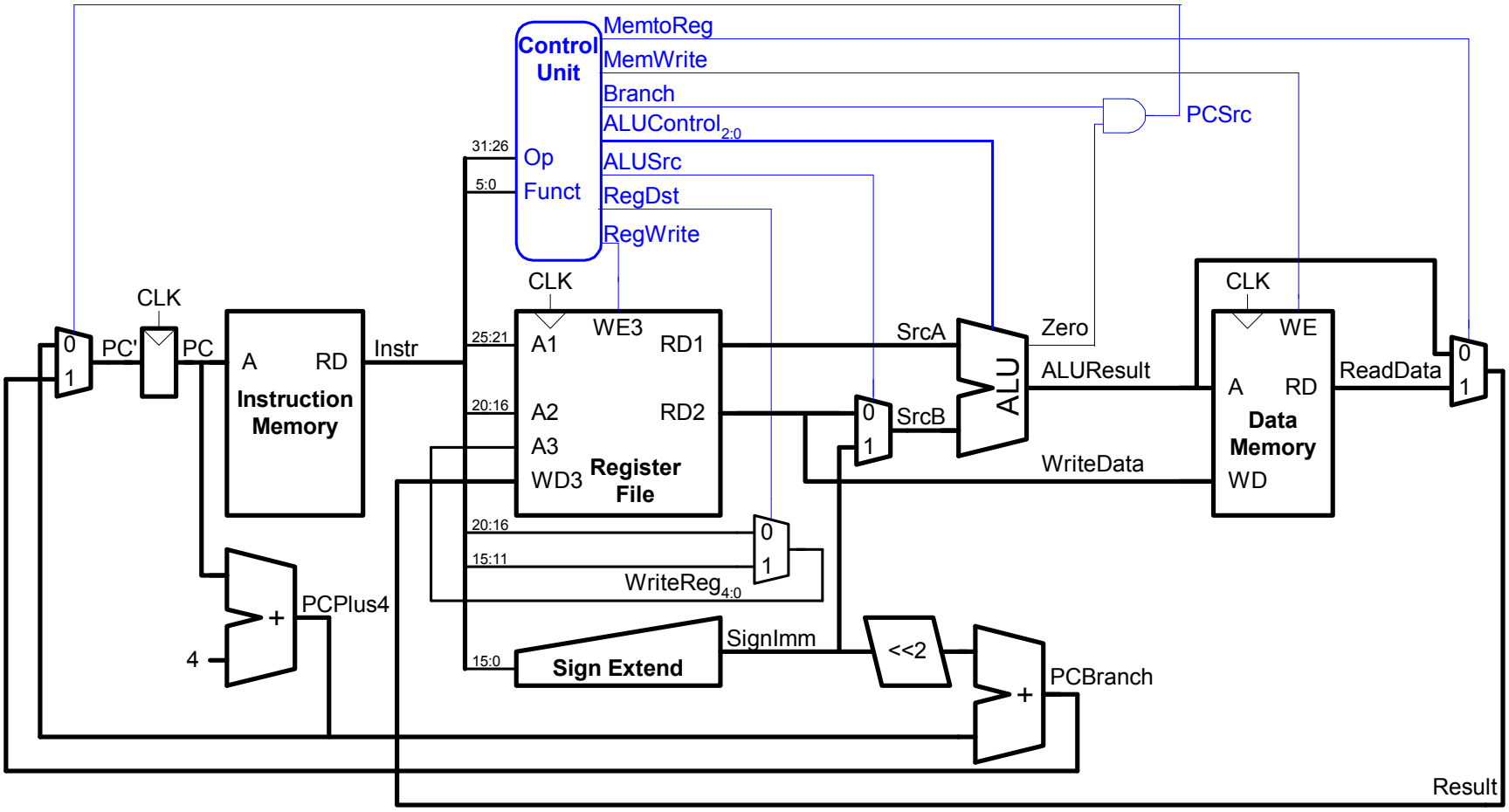
Control Unit: Main Decoder

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	0	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01

Single-Cycle Datapath: `or`



Extended Functionality: addi



No change to datapath

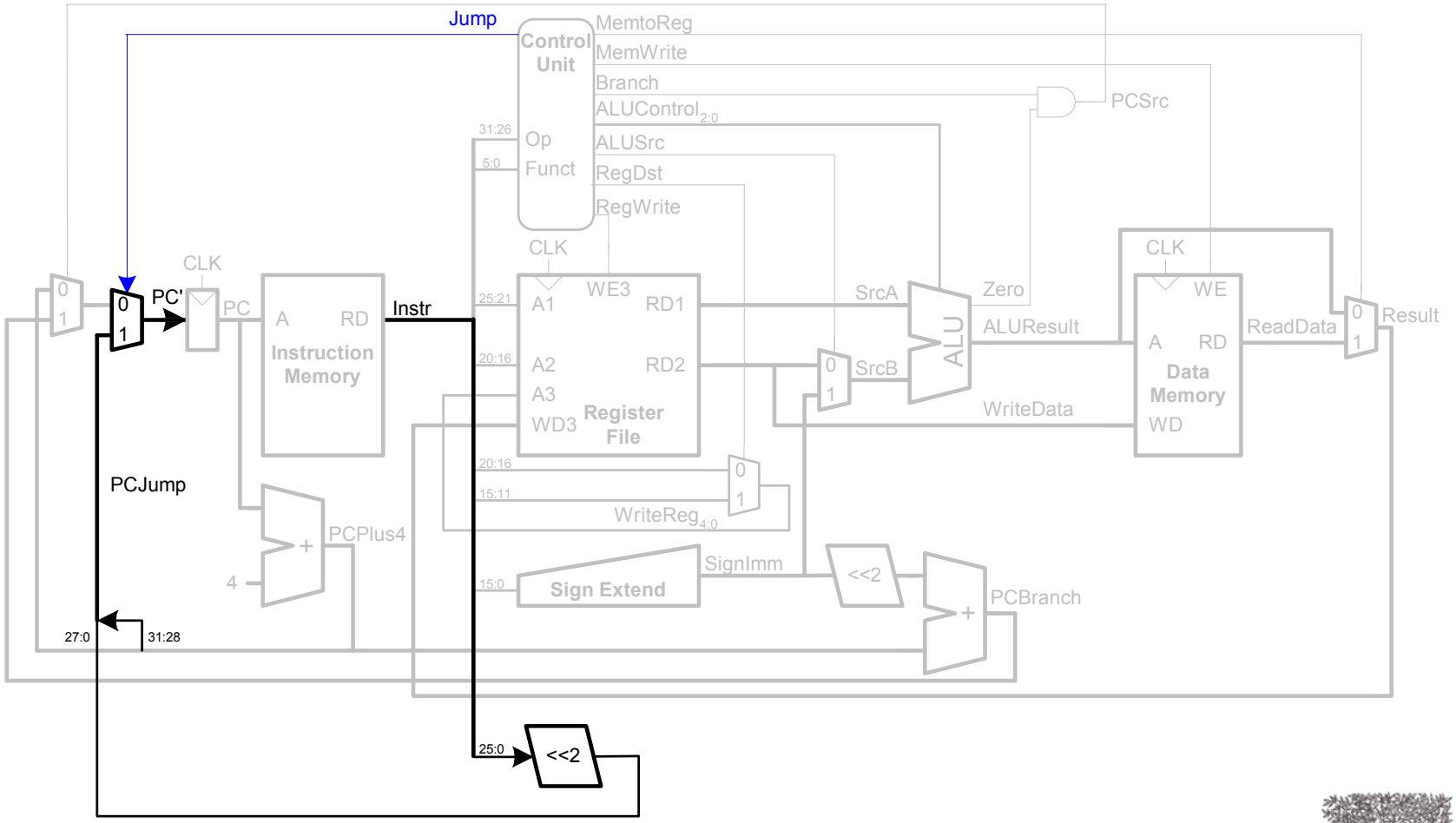
Control Unit: addi

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01
addi	**001000**							

Control Unit: addi

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01
addi	**001000**	**1**	**0**	**1**	**0**	**0**	**0**	**00**

Extended Functionality: j



Control Unit: Main Decoder

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump
R-type	000000	1	1	0	0	0	0	10	0
`lw`	100011	1	0	1	0	0	1	00	0
`sw`	101011	0	X	1	0	1	X	00	0
`beq`	000100	0	X	0	1	0	X	01	0
`j`	**000100**								

Control Unit: Main Decoder

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump
R-type	000000	1	1	0	0	0	0	10	0
`lw`	100011	1	0	1	0	0	1	00	0
`sw`	101011	0	X	1	0	1	X	00	0
`beq`	000100	0	X	0	1	0	X	01	0
`j`	000100	0	X	X	X	0	X	XX	1

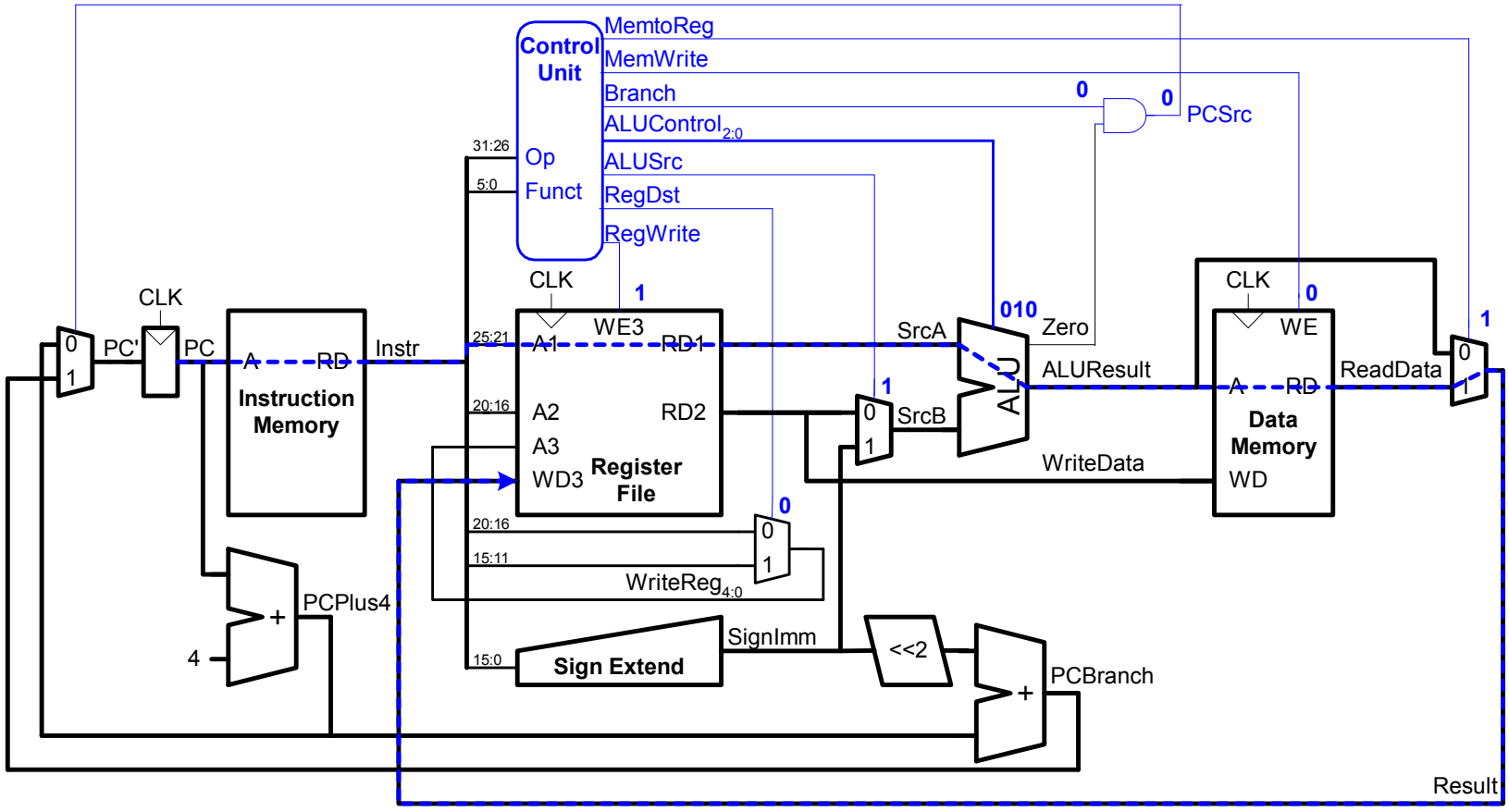
Review: Processor Performance

Program Execution Time

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

= # instructions x CPI x T_C

Single-Cycle Performance



T_c limited by critical path (`lw`)

Single-Cycle Performance

- Single-cycle critical path:

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

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

Single-Cycle 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

$$T_c = ?$$

Single-Cycle 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

$$T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + 2\,t_{mux} + t_{ALU} + t_{RFsetup}$$
$$= [30 + 2(250) + 150 + 2(25) + 200 + 20]\ ps$$
$$= 950\ ps$$

Single-Cycle Performance Example

Program with 100 billion instructions:

Execution Time = # instructions x CPI x T_C

$= (100 \times 10^9)(1)(950 \times 10^{-12}\ \text{s})$

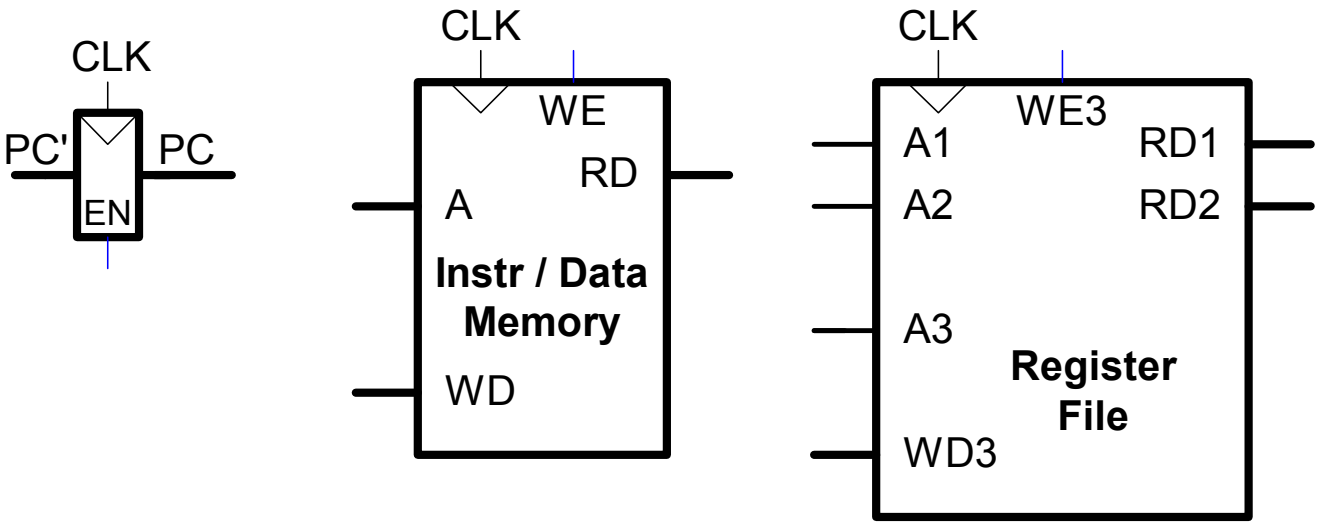
$=$ **95 seconds**

Multicycle MIPS Processor

- **Single-cycle:**
  - + simple
  - - cycle time limited by longest instruction (`lw`)
  - - 2 adders/ALUs & 2 memories
- **Multicycle:**
  - + higher clock speed
  - + simpler instructions run faster
  - + reuse expensive hardware on multiple cycles
  - - sequencing overhead paid many times
- **Same design steps: datapath & control**

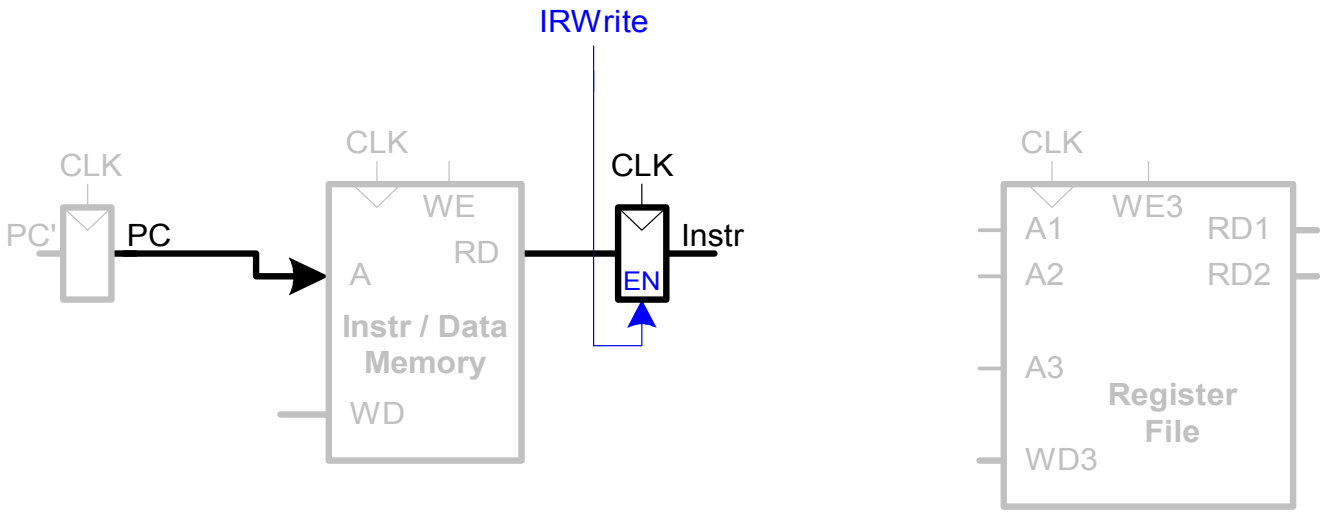
Multicycle State Elements

- Replace Instruction and Data memories with a single unified memory – more realistic



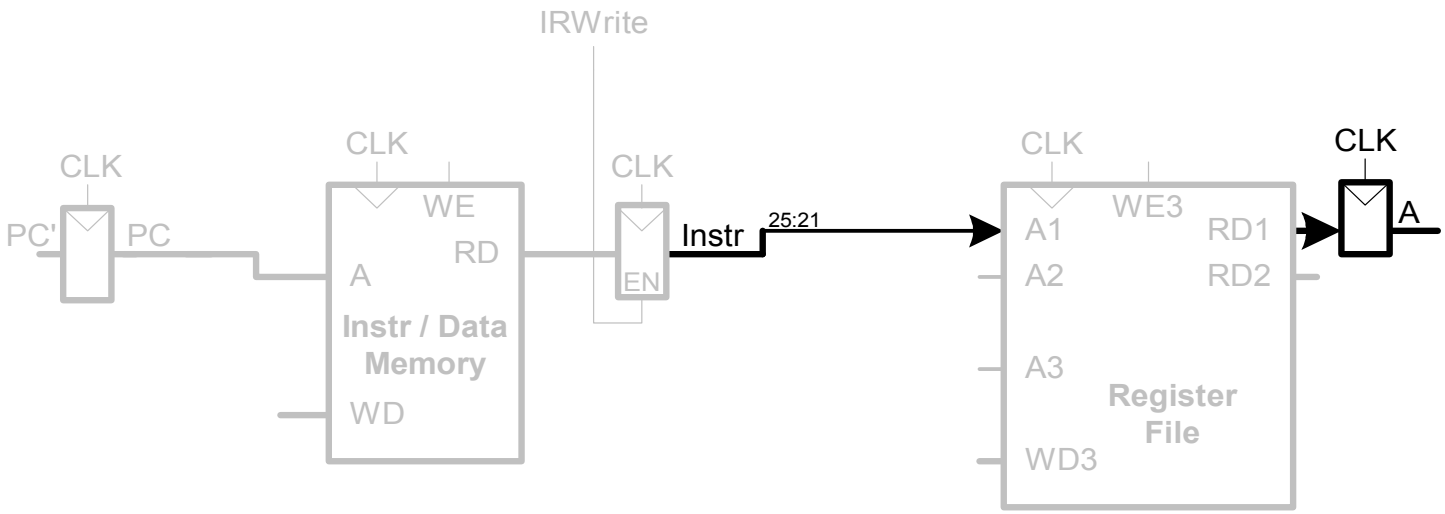
Multicycle Datapath: Instruction Fetch

STEP 1: Fetch instruction



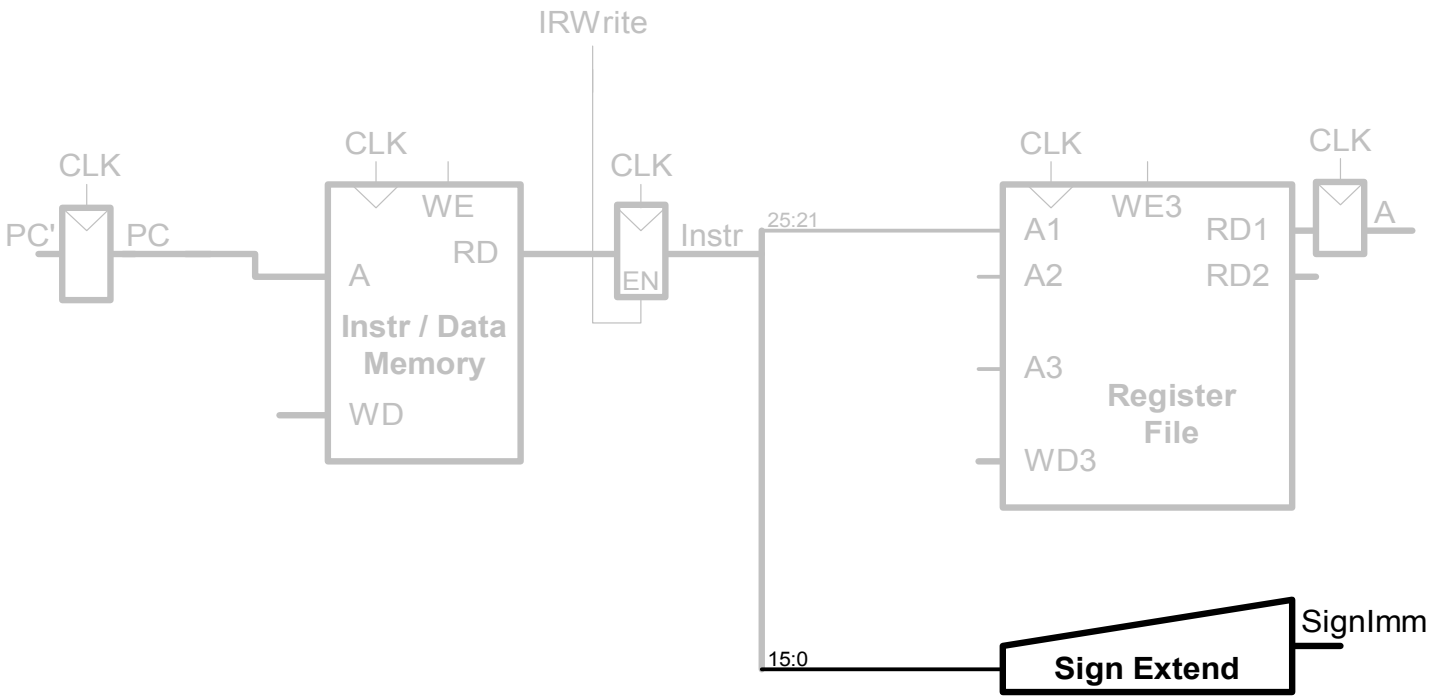
Multicycle Datapath: `lw` Register Read

STEP 2a: Read source operands from RF



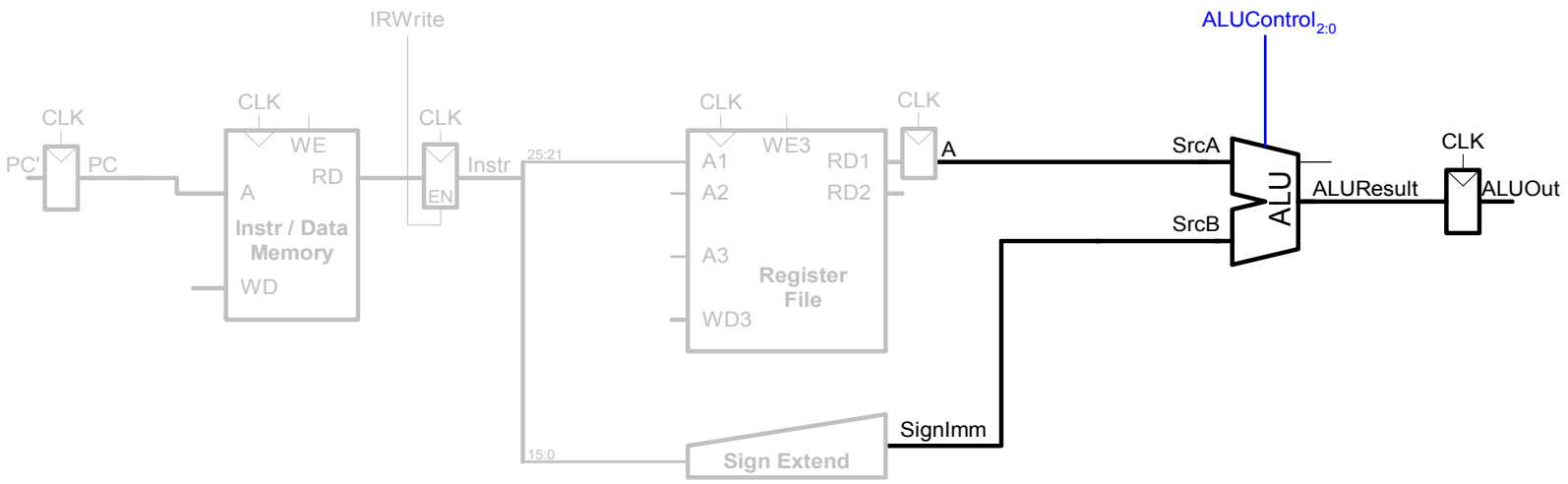
Multicycle Datapath: `lw` Immediate

STEP 2b: Sign-extend the immediate



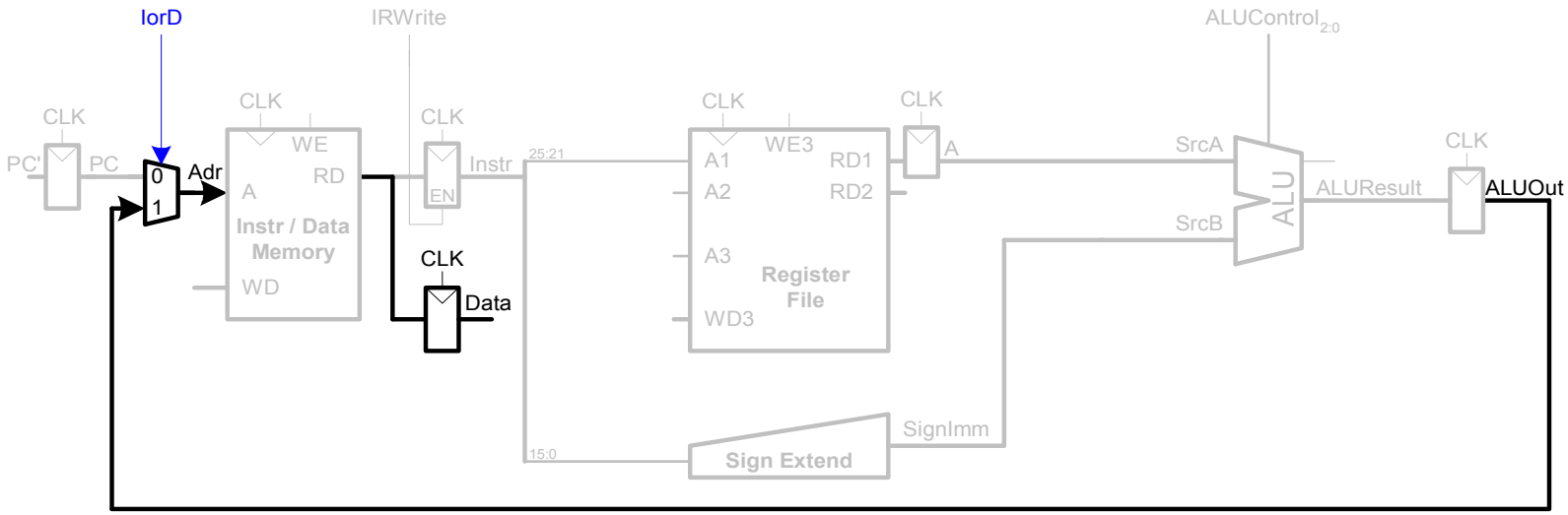
Multicycle Datapath: `lw` Address

STEP 3: Compute the memory address



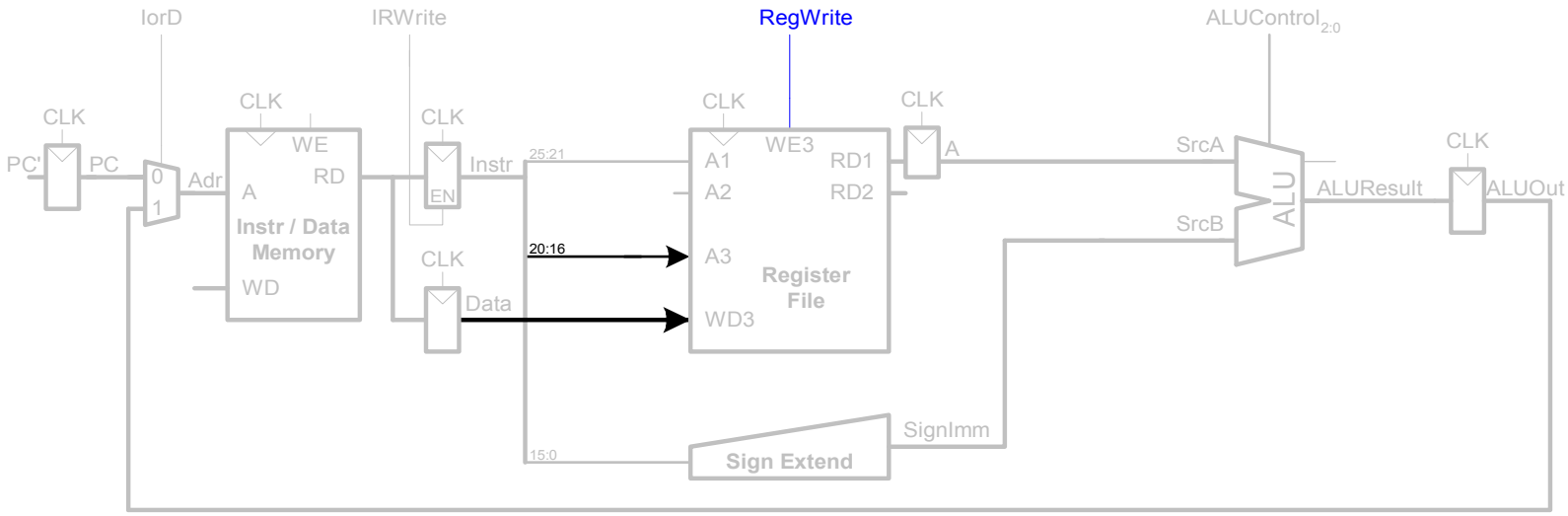
Multicycle Datapath: `lw` Memory Read

STEP 4: Read data from memory



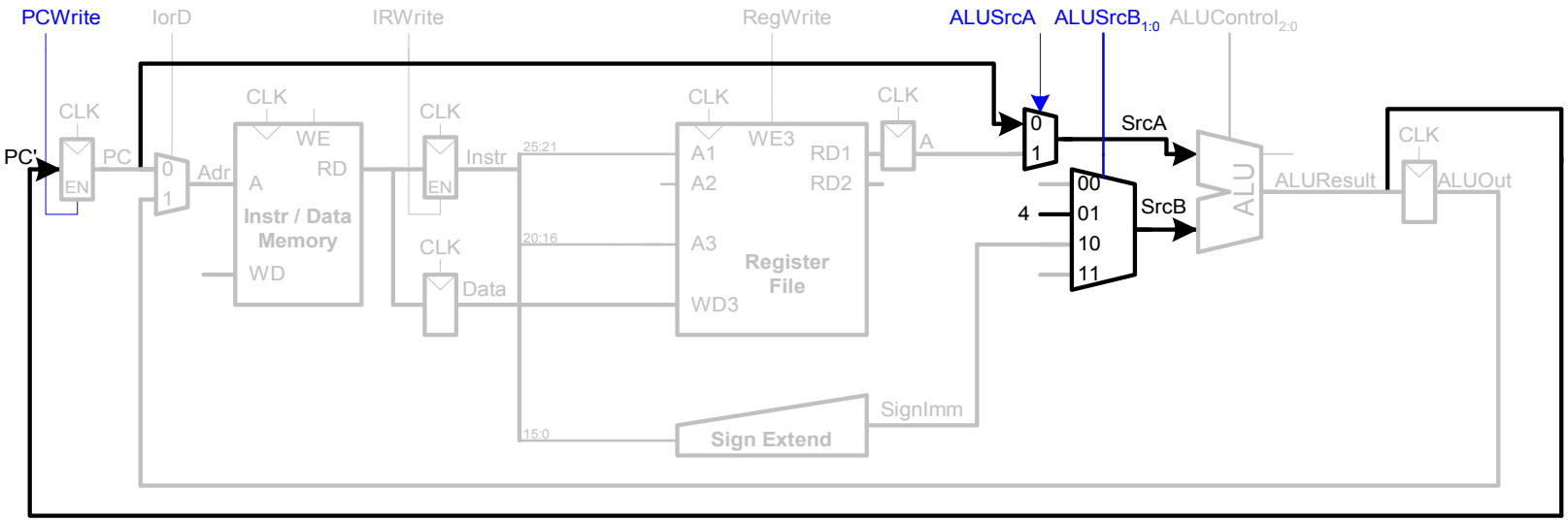
Multicycle Datapath: `lw` Write Register

STEP 5: Write data back to register file



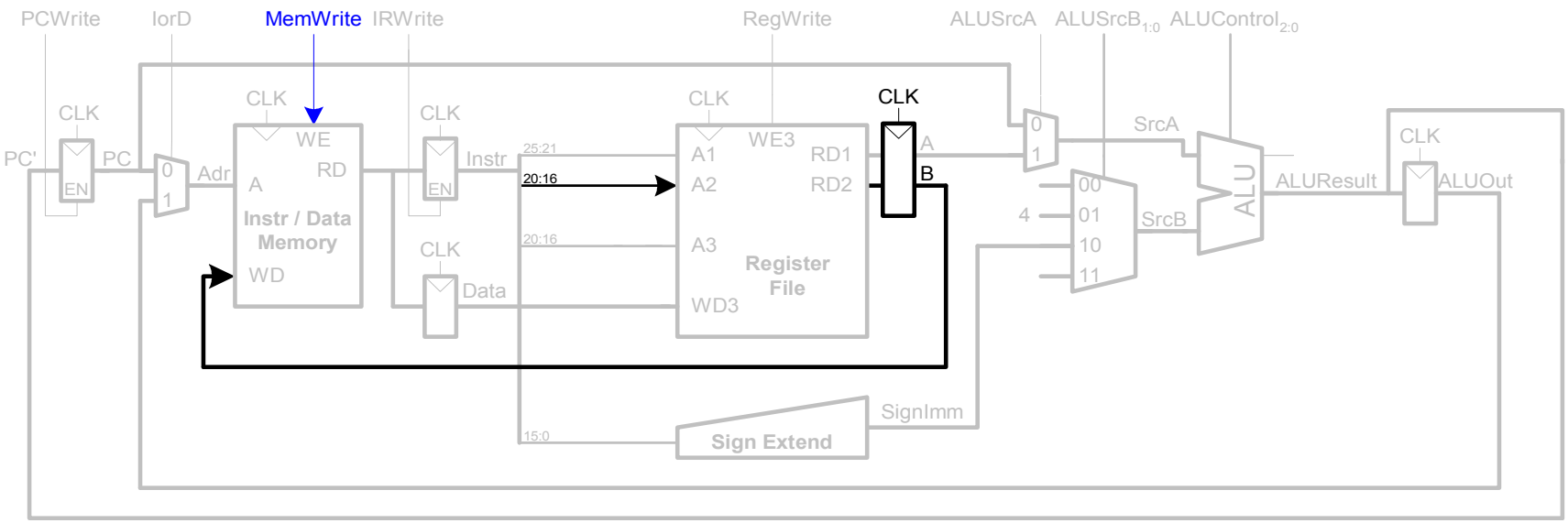
Multicycle Datapath: Increment PC

STEP 6: Increment PC



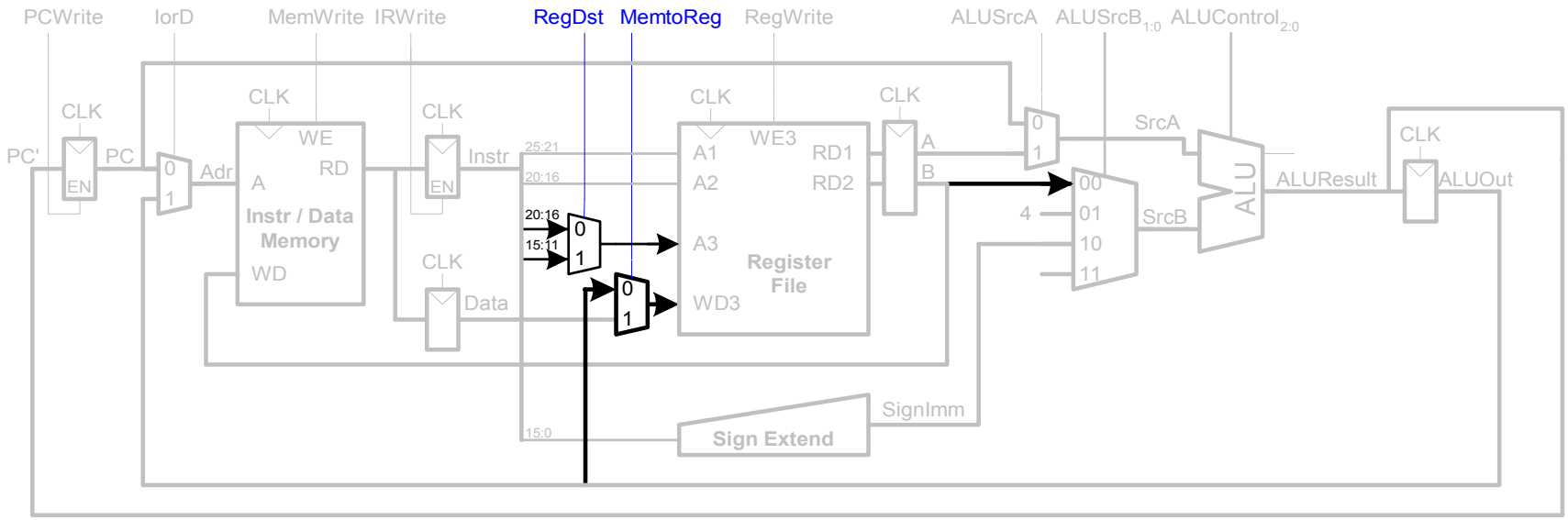
Multicycle Datapath: `sw`

Write data in `rt` to memory



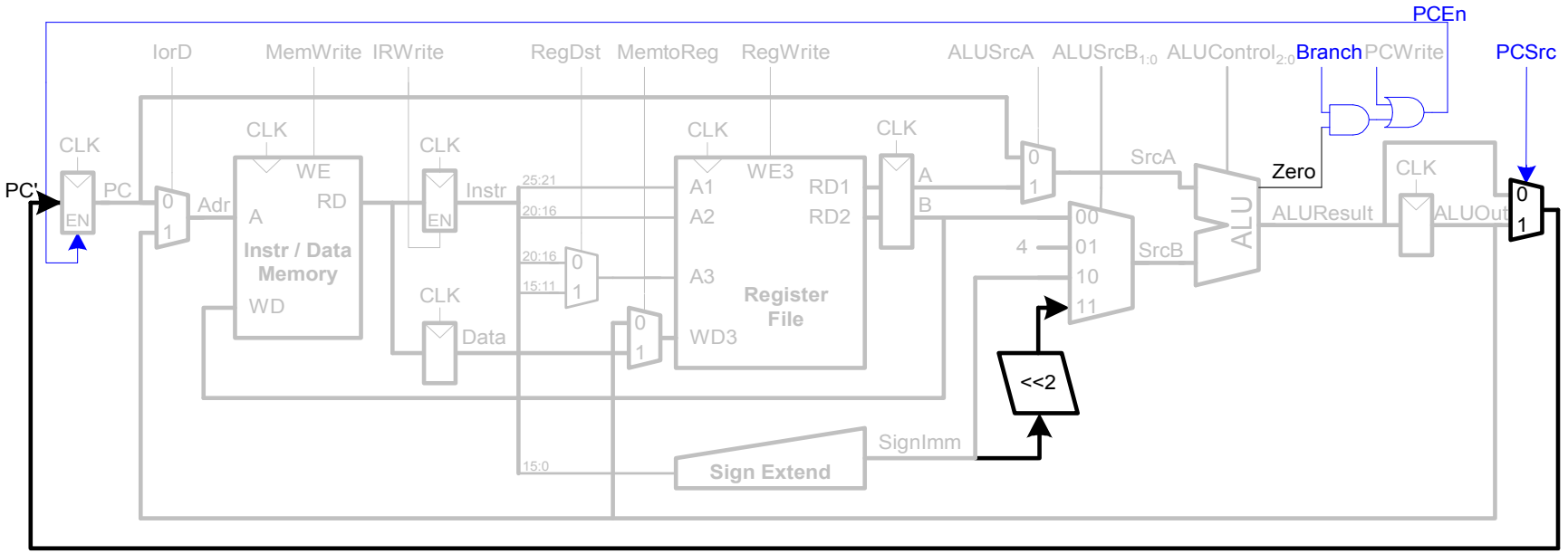
Multicycle Datapath: R-Type

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

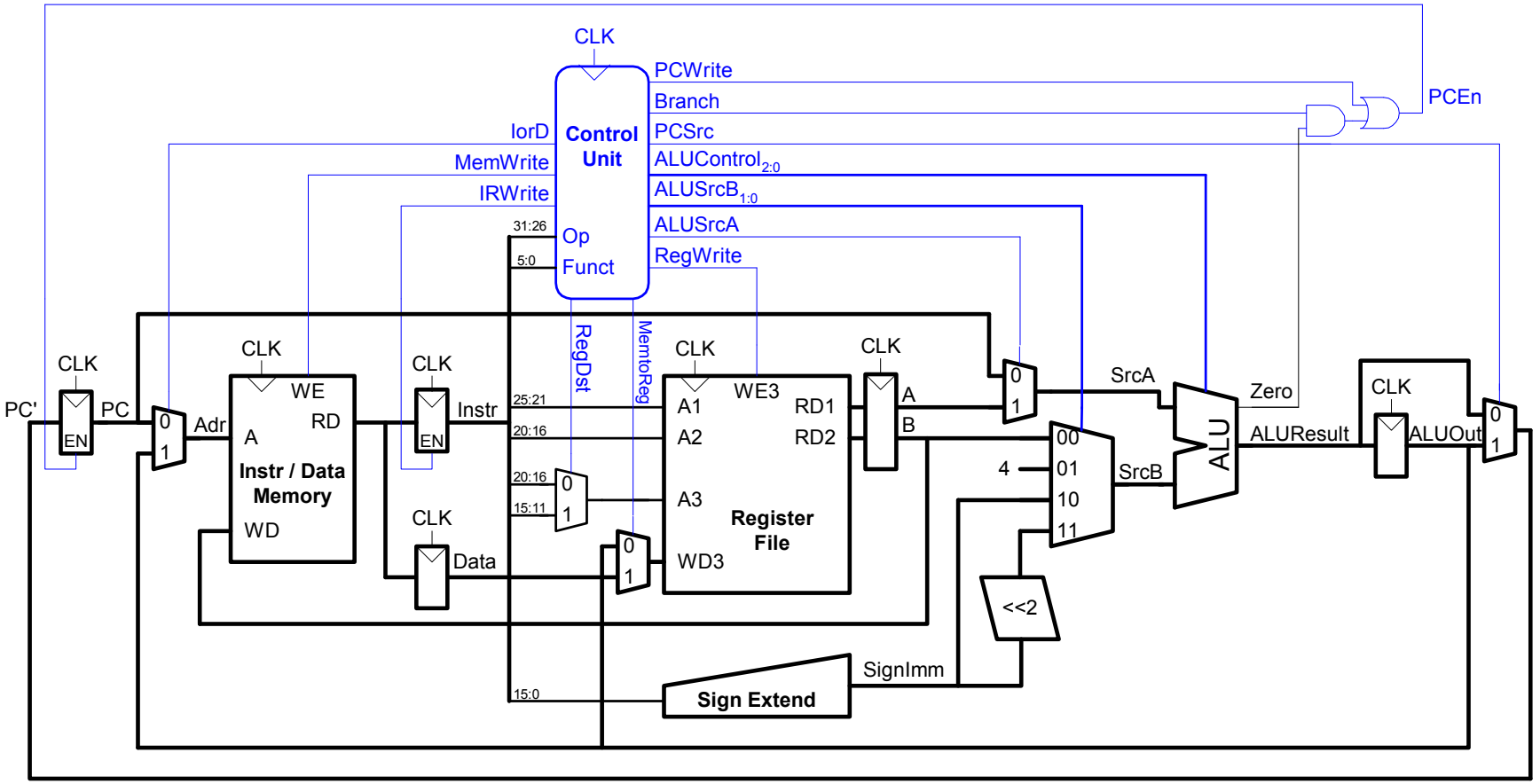


Multicycle Datapath: `beq`

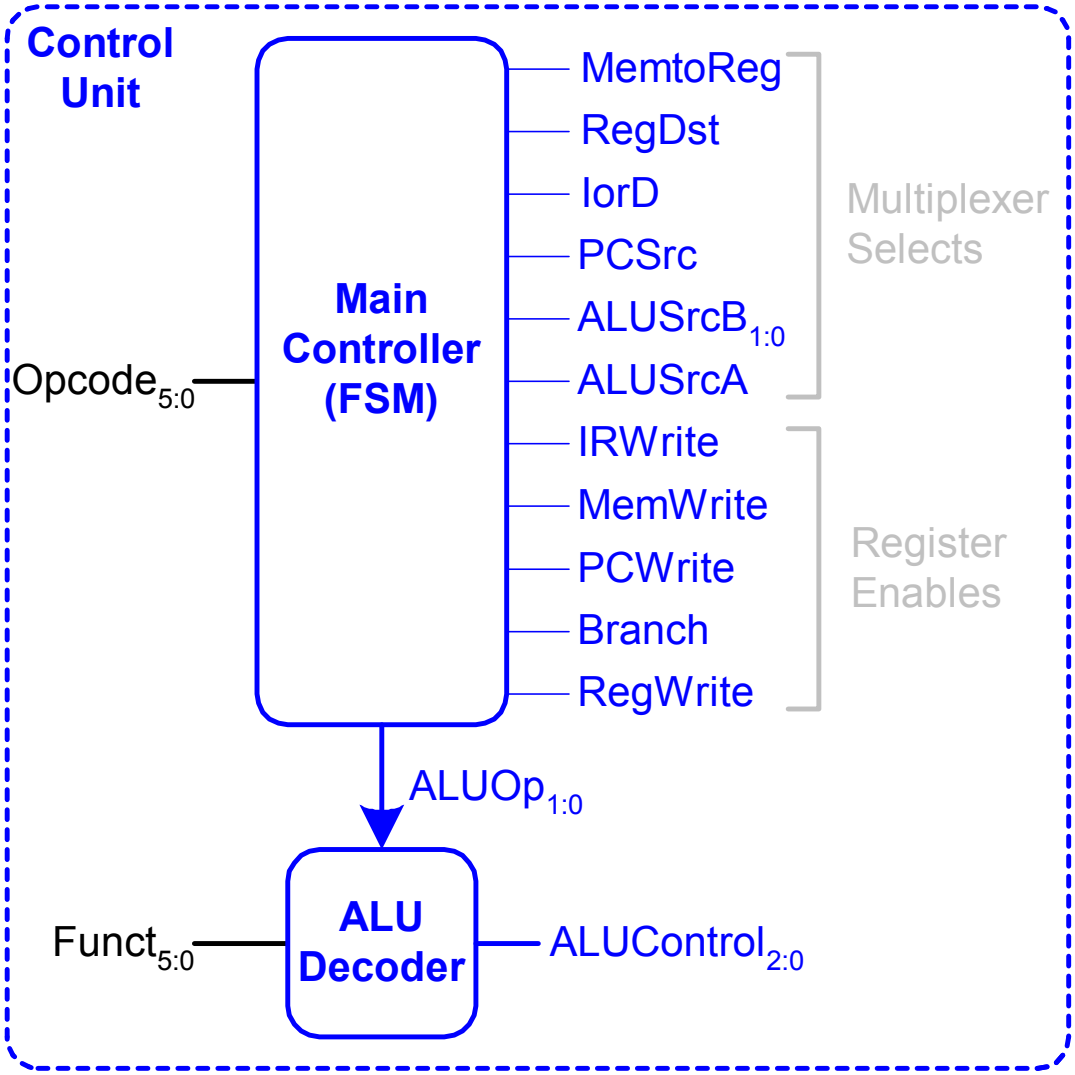
- `rs == rt`?
- BTA = (sign-extended immediate << 2) + (PC+4)



Multicycle Processor



Multicycle Control



Multicycle Processor Performance

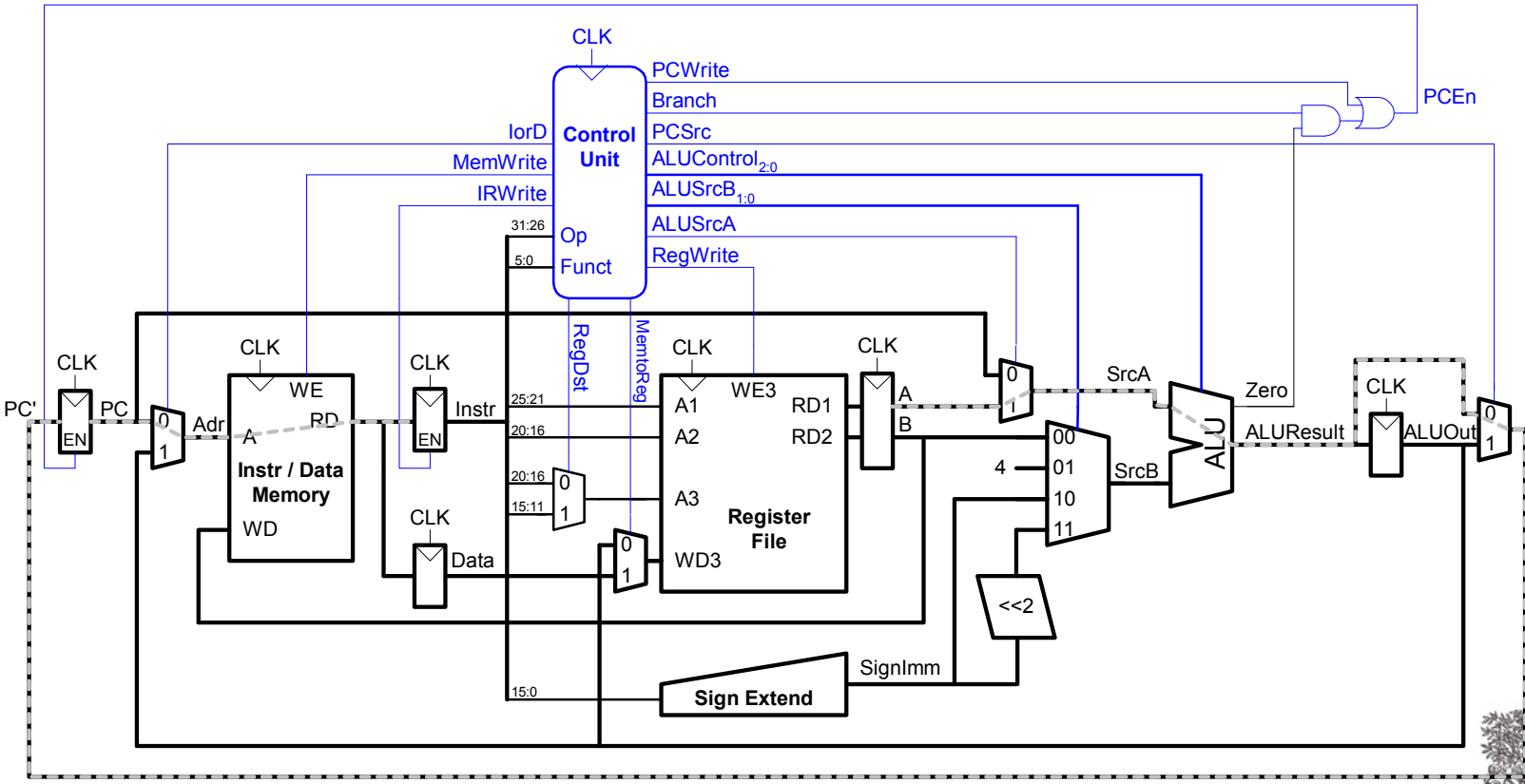
- Instructions take different number of cycles:
  - 3 cycles: `beq`, `j`
  - 4 cycles: R-Type, `sw`, `addi`
  - 5 cycles: `lw`
- CPI is weighted average
- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type

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

Multicycle Processor Performance

Multicycle critical path:

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



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

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

Chapter 7 :: Topics

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

Execution Time =

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

Execution Time = (# instructions) × CPI × T_c

$= (100 \times 10^9)(4.12)(325 \times 10^{-12})$

= 133.9 seconds

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

Multicycle Performance Example

Program with 100 billion instructions

Execution Time = (# instructions) × CPI × T_c

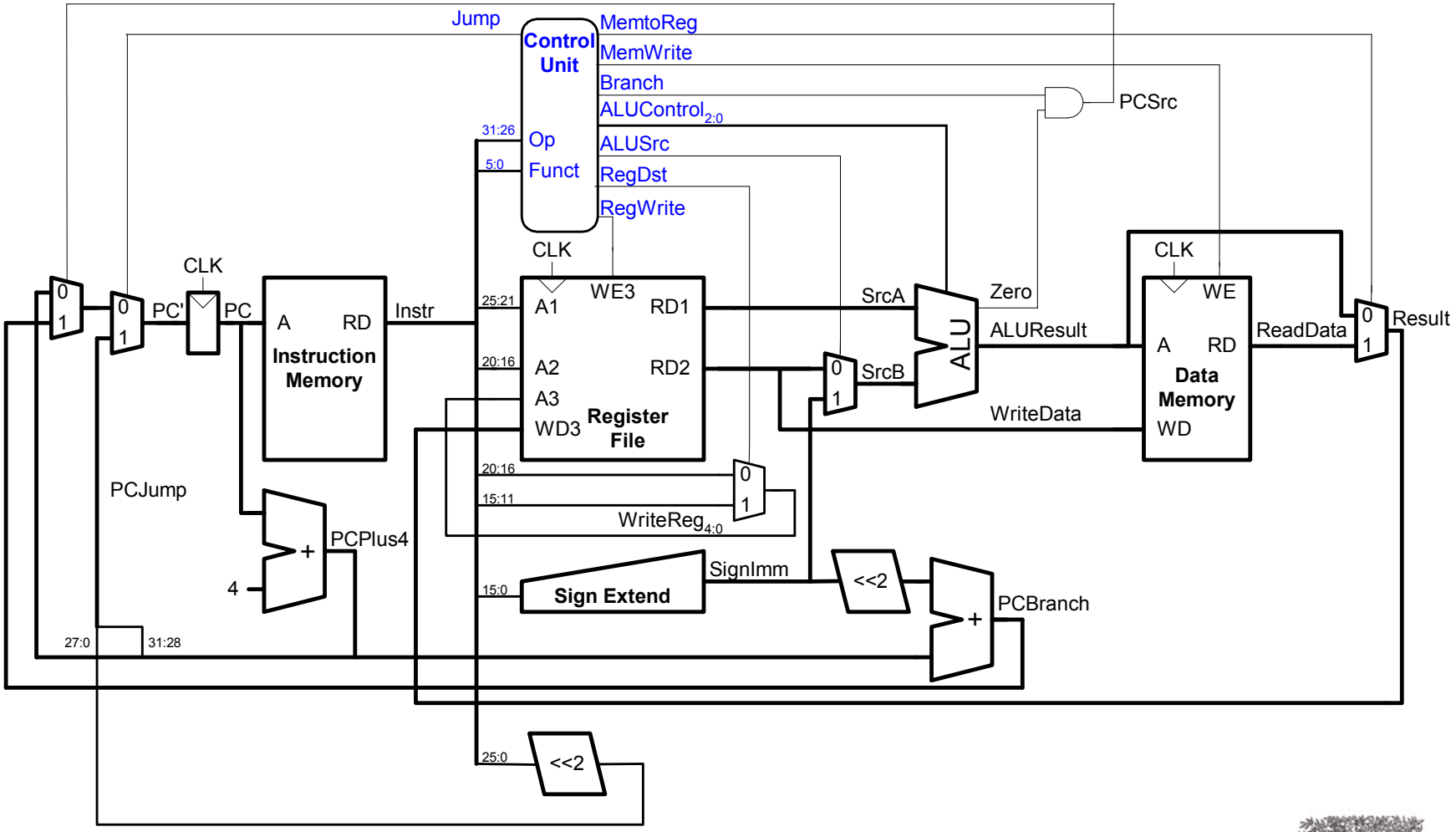
$= (100 \times 10^9)(4.12)(325 \times 10^{-12})$

$=$ **133.9 seconds**

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

- Not all steps same length
- Sequencing overhead for each step ($t_{pcq} + t_{setup} = 50$ ps)

Review: Single-Cycle Processor



Review: Multicycle Processor