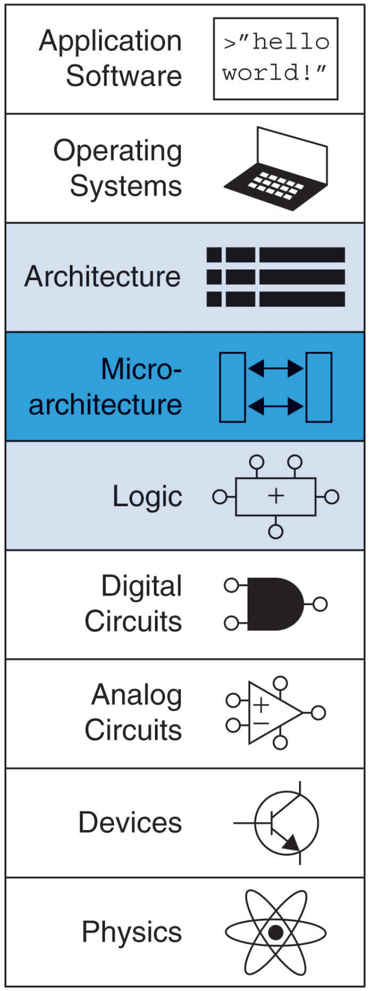
Digital Design and Computer Architecture, RISC-V Edition

David M. Harris and Sarah L. Harris

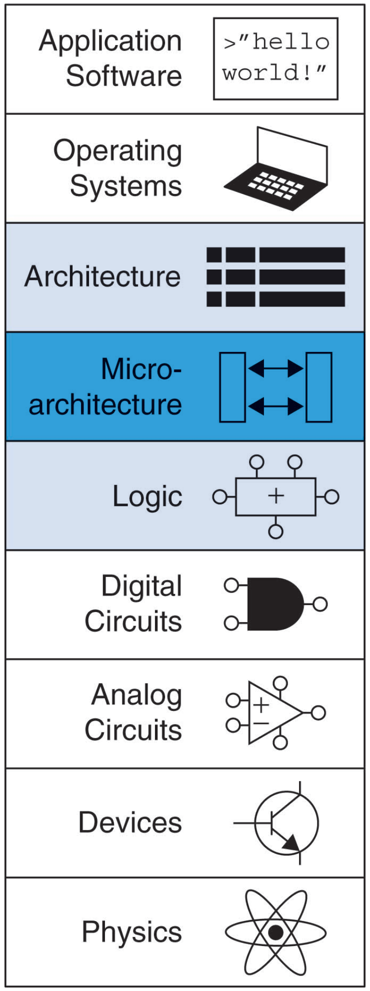
Chapter 7 :: Microarchitecture

- **Introduction**
- **Performance Analysis**
- **Single-Cycle Processor**
- **Multicycle Processor**
- **Pipelined Processor**
- **Advanced Microarchitecture**



Introduction

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



Microarchitecture

- Multiple implementations for a single architecture:
  - **Single-cycle:** Each instruction executes in a single cycle
  - **Multicycle:** Each instruction is broken up into series of shorter steps
  - **Pipelined:** Each instruction broken up into series of steps & multiple instructions execute at once

Processor Performance

- **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

RISC-V Processor

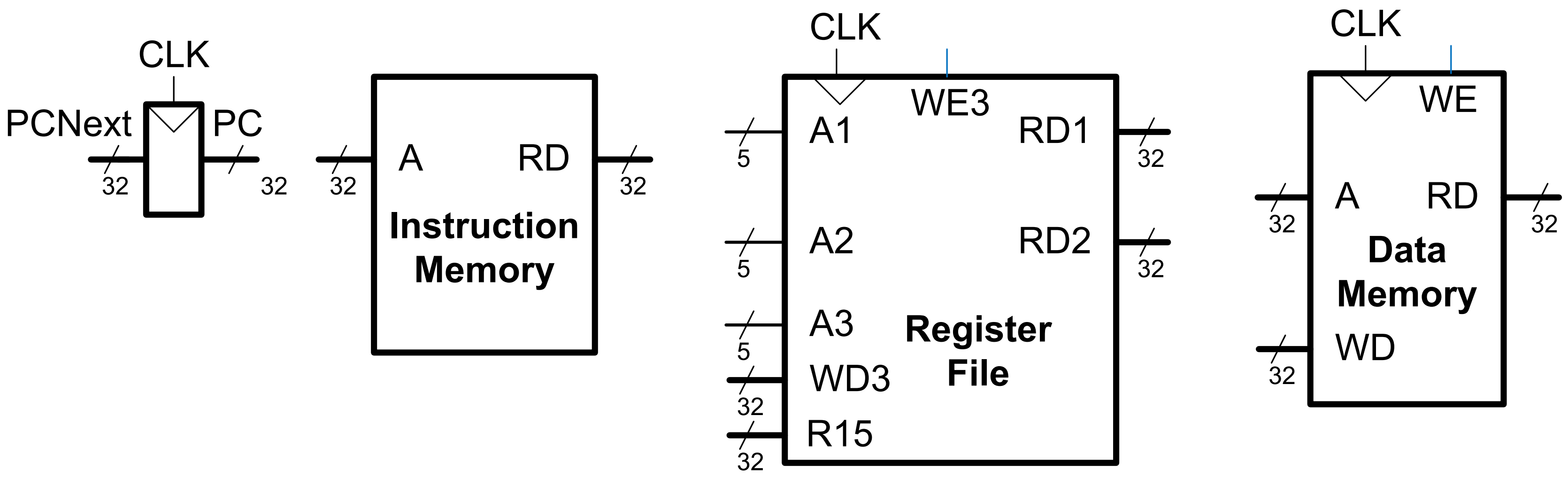
- Consider **subset** of RISC-V instructions:
  - **R-type instructions:**
    - `add`, `sub`, `and`, `or`, `slt`
  - **I-type instruction:**
    - `lw`
  - **S-type instruction:**
    - `sw`
  - **B-type instructions:**
    - `beq`

Architectural State Elements

Determines everything about a processor:

- Architectural state:
  - 32 registers
  - PC
  - Memory

RISC-V Architectural State Elements



Single-Cycle RISC-V Processor

- Datapath
- Control

Single-Cycle RISC-V Processor

- **Datapath**
- Control

RISC-V Processor

- **R-type instructions:**
  - `add, sub, and, or, slt`
- **I-type instruction:**
  - `lw`
- **S-type instruction:**
  - `sw`
- **B-type instructions:**
  - `beq`

R-Type

31:25	24:20	19:15	14:12	11:7	6:0
funct7	rs2	rs1	funct3	rd	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

I-Type

31:20	19:15	14:12	11:7	6:0
$imm_{11:0}$	rs1	funct3	rd	op
12 bits	5 bits	3 bits	5 bits	7 bits

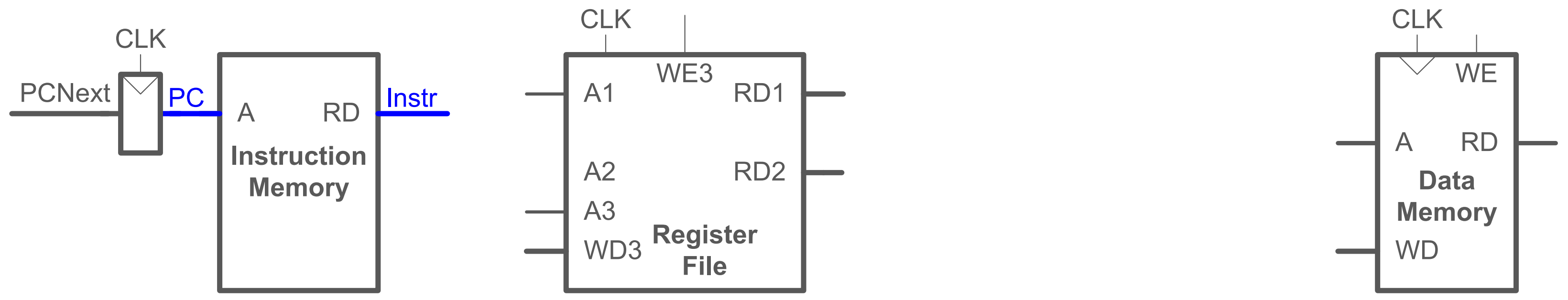
S-Type

31:25	24:20	19:15	14:12	11:7	6:0
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

B-Type

31:25	24:20	19:15	14:12	11:7	6:0
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Single-Cycle RISC-V Datapath



Single-Cycle RISC-V Processor

- **Datapath:** start with `lw` instruction
- **Example:** `lw t2, -8(s3)`

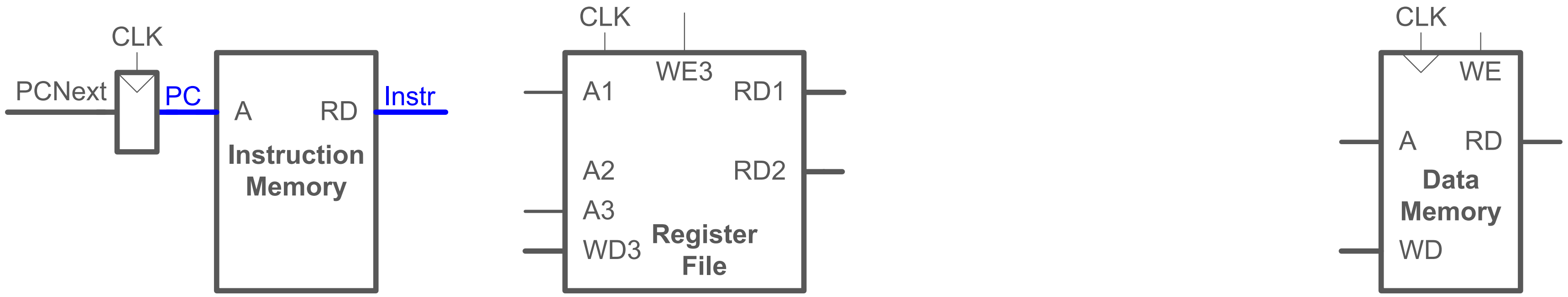
  `lw rd, imm(rs1)`

I-Type

31:20	19:15	14:12	11:7	6:0
$imm_{11:0}$	rs1	funct3	rd	op
12 bits	5 bits	3 bits	5 bits	7 bits

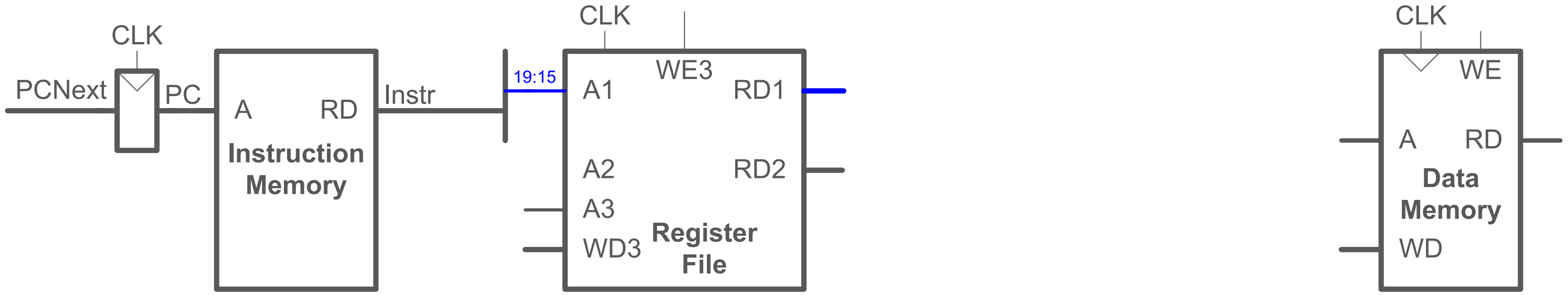
Single-Cycle Datapath: `lw` fetch

STEP 1: Fetch instruction



Single-Cycle Datapath: `lw` Reg Read

STEP 2: Read source operand (**rs1**) from RF



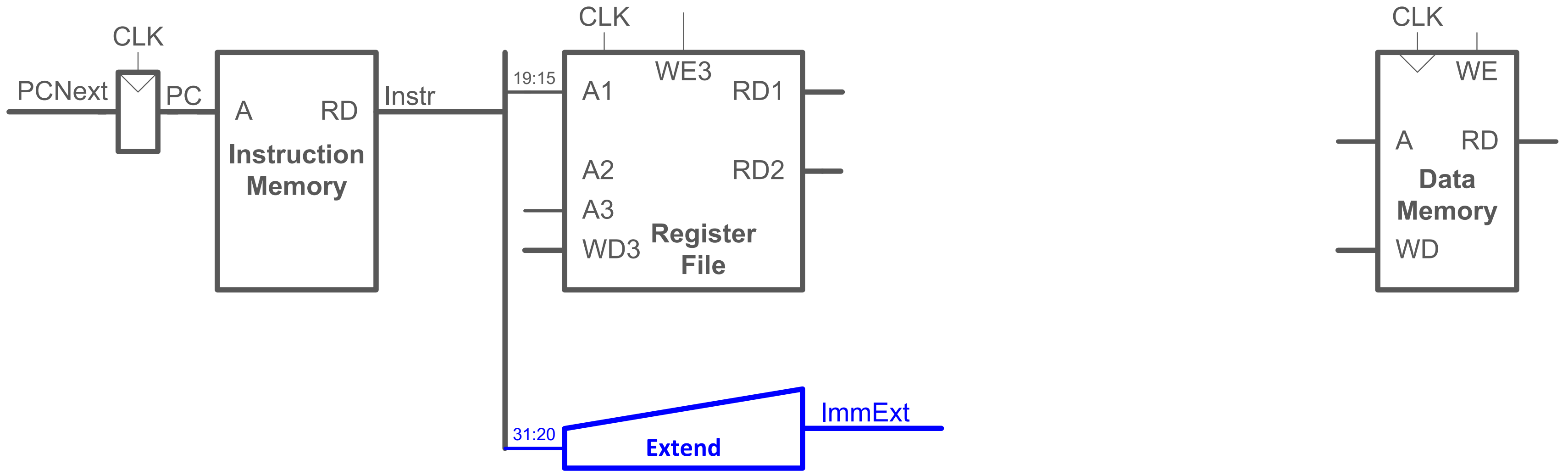
I-Type

31:20	19:15	14:12	11:7	6:0
$imm_{11:0}$	rs1	funct3	rd	op
12 bits	5 bits	3 bits	5 bits	7 bits

`lw rd, imm(rs1)`

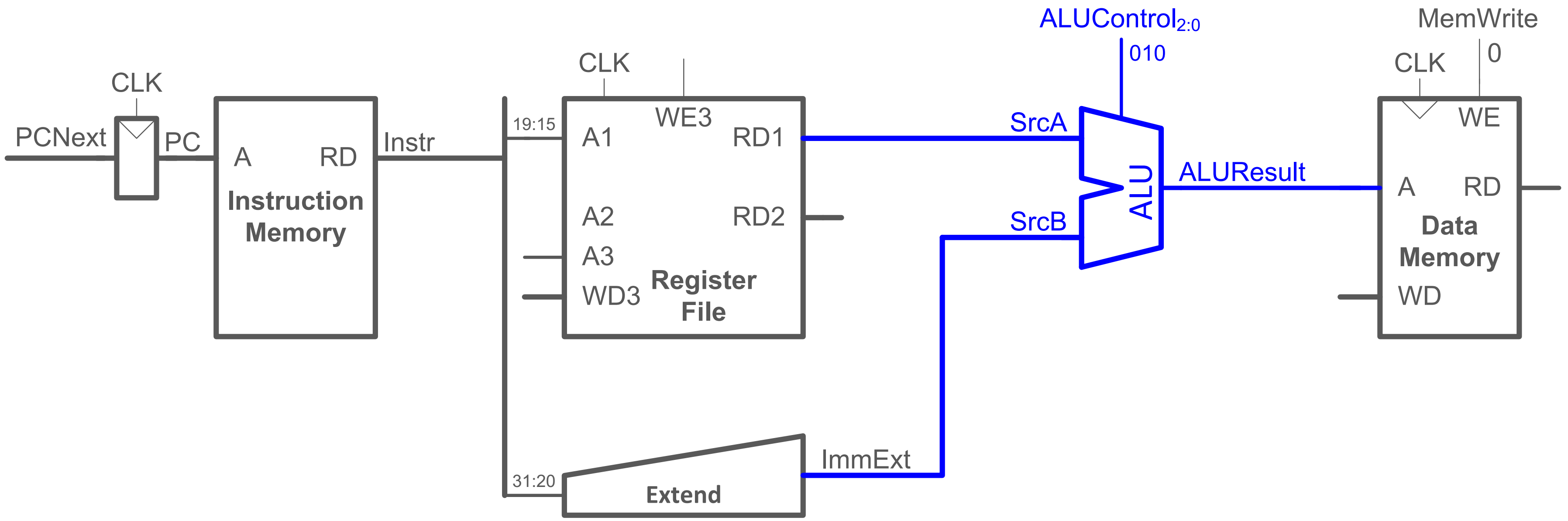
Single-Cycle Datapath: `lw` Immediate

STEP 3: Extend the immediate



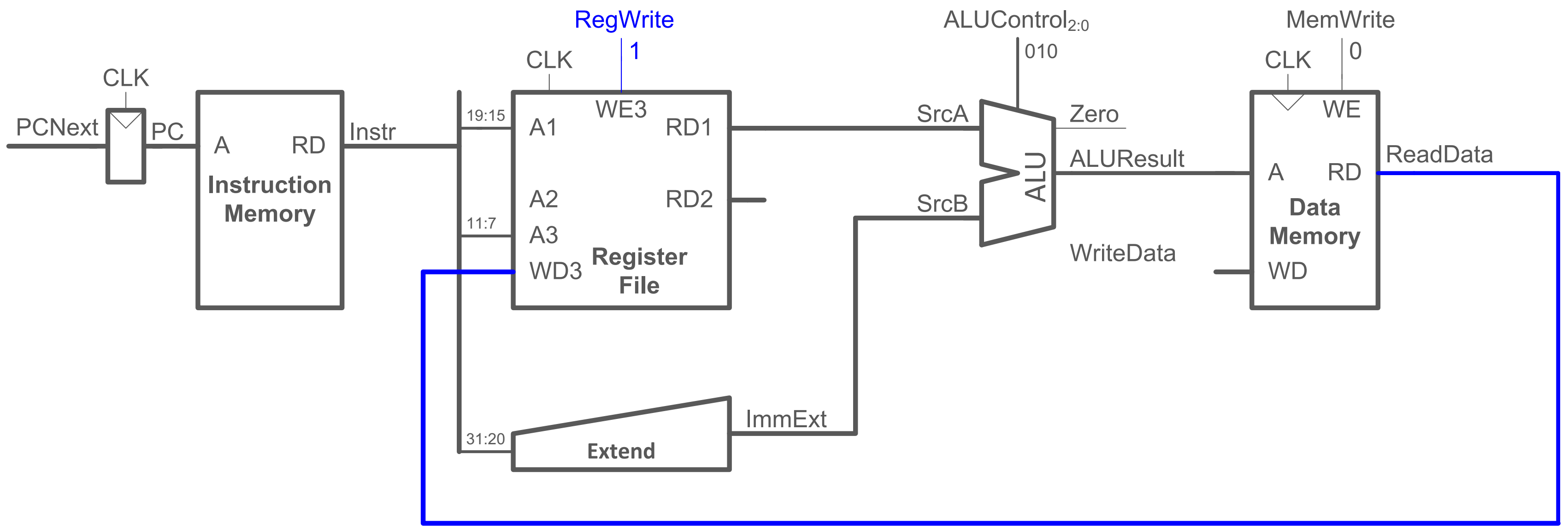
Single-Cycle Datapath: `lw` Address

STEP 4: Compute the memory address



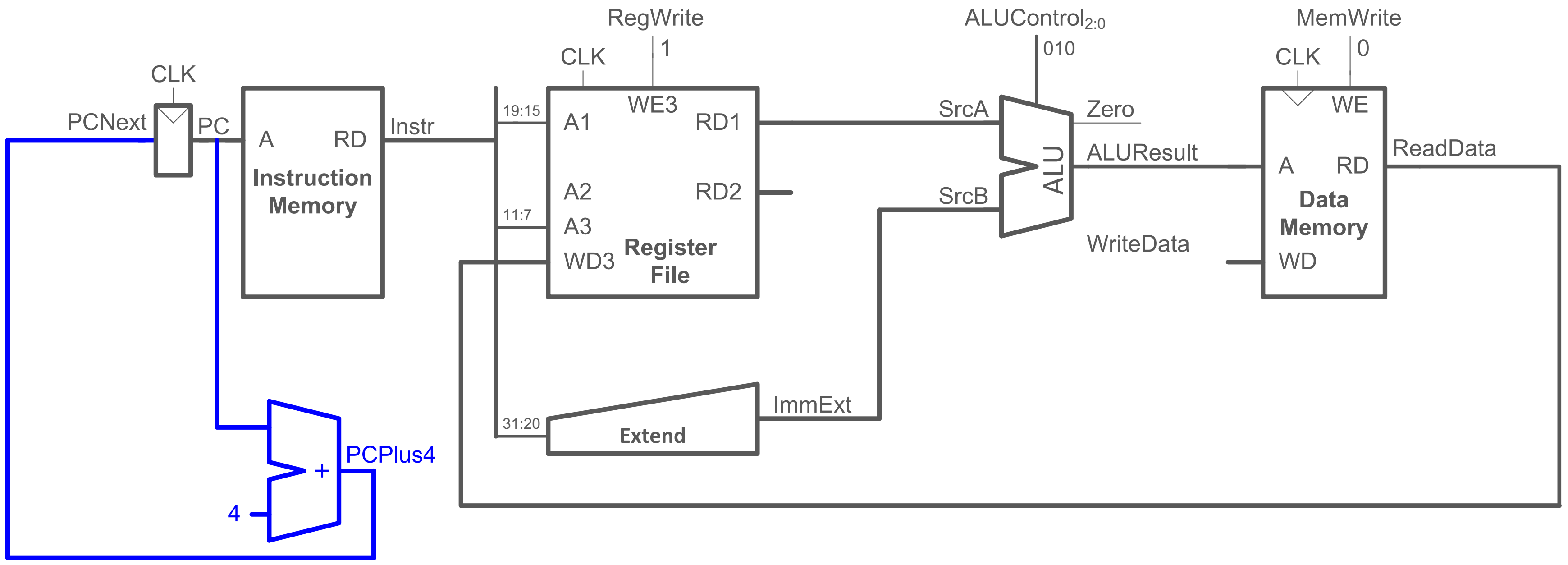
Single-Cycle Datapath: LDR Mem Read

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



Single-Cycle Datapath: PC Increment

STEP 6: Determine address of next instruction



Single-Cycle Datapath: `sw`

Expand datapath to handle `sw`:

- Write data in `rs2` to memory
- **Example:** `sw t2, 0xc(s3)`

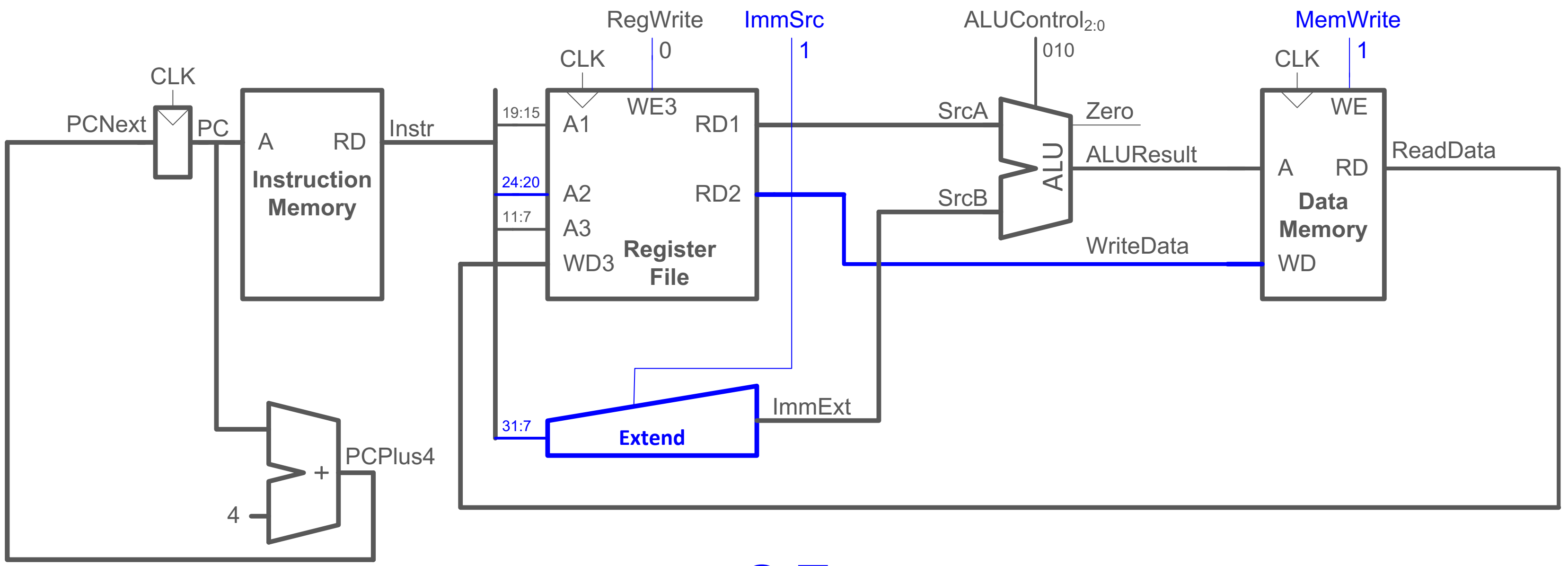
  `sw rs2, imm(rs1)`

S-Type

31:25	24:20	19:15	14:12	11:7	6:0
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Single-Cycle Datapath: Data-processing

- **Immediate:** now in {instr[31:25], instr[11:7]}
- **Add control signals:** ImmSrc, MemWrite



S-Type

31:25	24:20	19:15	14:12	11:7	6:0
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

```
sw rs2, imm(rs1)
```

Single-Cycle Datapath: Immediate

ImmSrc	ImmExt	Instruction Type
0	{{20{instr[31]}}, instr[31:20]}	I-Type
1	{{20{instr[31]}}, instr[31:25], instr[11:7]}	S-Type

I-Type

31:20	19:15	14:12	11:7	6:0
$\text{imm}_{11:0}$	rs1	funct3	rd	op
12 bits	5 bits	3 bits	5 bits	7 bits

S-Type

31:25	24:20	19:15	14:12	11:7	6:0
$\text{imm}_{11:5}$	rs2	rs1	funct3	$\text{imm}_{4:0}$	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Single-Cycle Datapath: R-Type

- **Instructions:** `add`, `sub`, `and`, `or`, `slt`,
- **Example:** `add s1, s2, s3`

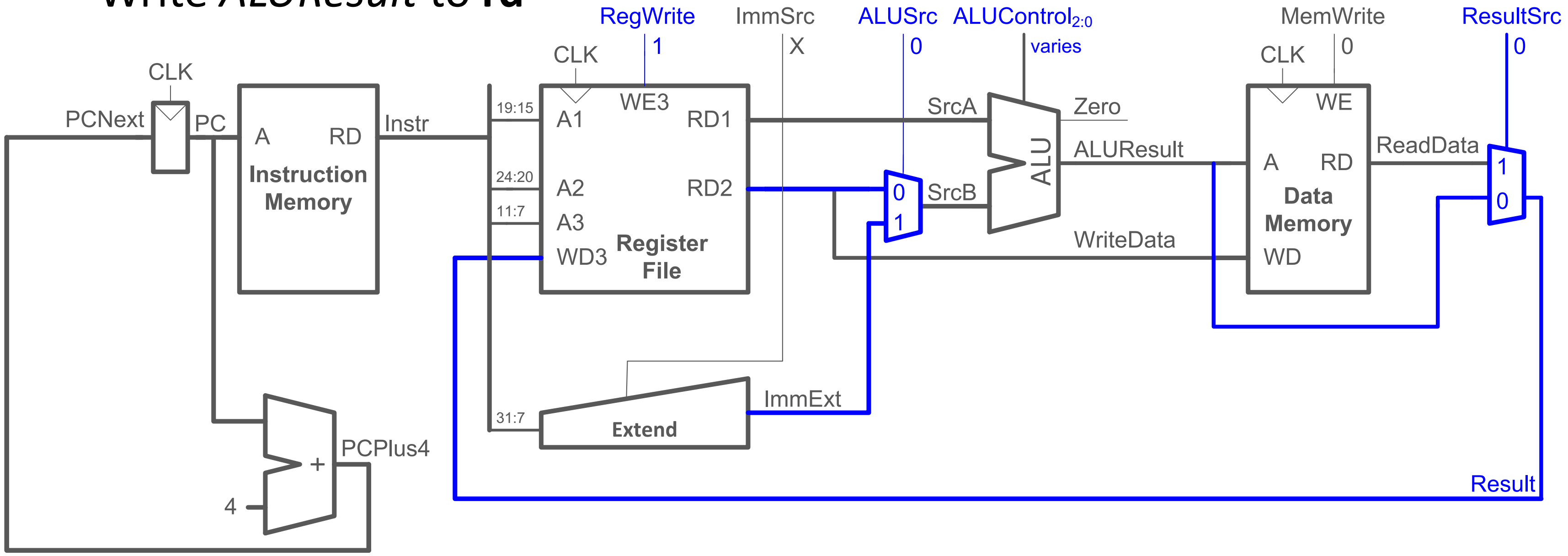
  `op rd, rs1, rs2`

R-Type

31:25	24:20	19:15	14:12	11:7	6:0
funct7	rs2	rs1	funct3	rd	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Single-Cycle Datapath: R-Type

- Read from **rs1** and **rs2** (instead of **imm**)
- Write *ALUResult* to **rd**



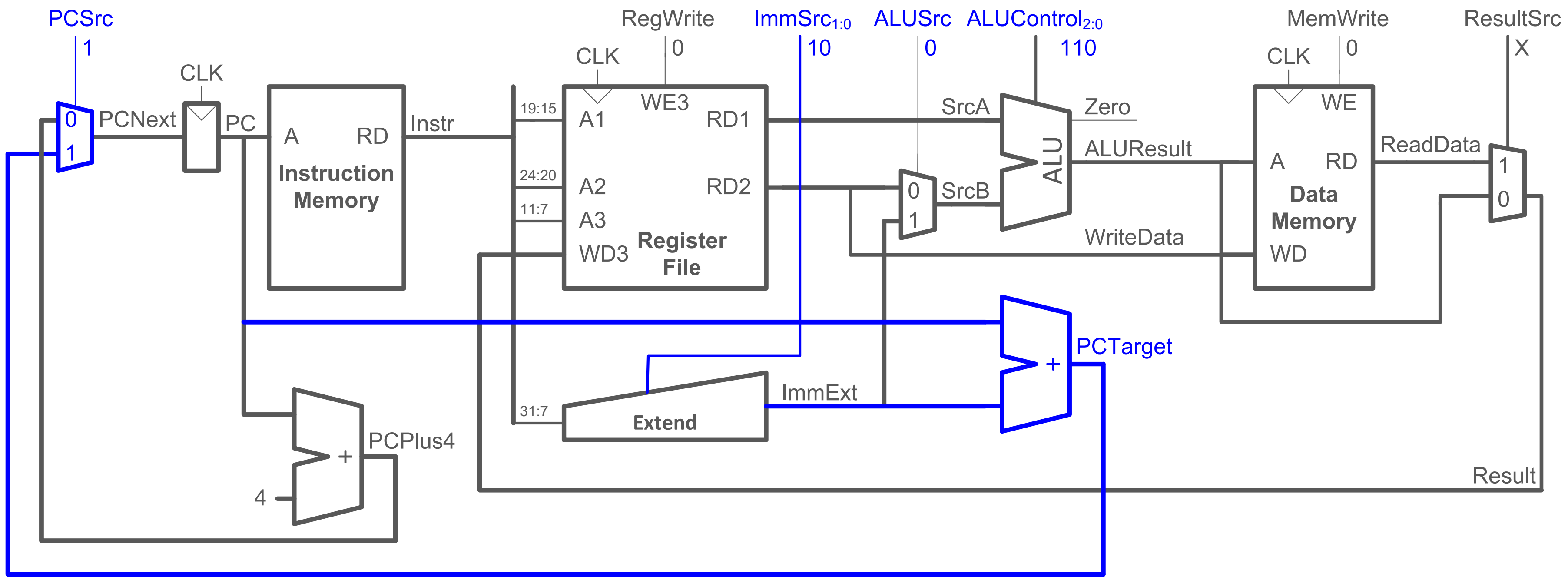
R-Type

31:25	24:20	19:15	14:12	11:7	6:0
funct7	rs2	rs1	funct3	rd	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

```
add rd, rs1, rs2
```

Single-Cycle Datapath: `beq`

Calculate branch target address: PCTarget = PC + imm



B-Type

31:25	24:20	19:15	14:12	11:7	6:0
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

`beq rs1, rs2, Label`

Single-Cycle Datapath: ImmExt

$ImmSrc_{1:0}$	ImmExt	Instruction Type
00	{{20{instr[31]}}, instr[31:20]}	I-Type
01	{{20{instr[31]}}, instr[31:25], instr[11:7]}	S-Type
10	{{19{instr[31]}}, instr[31], instr[7], instr[30:25], instr[11:8], 1'b0}	B-Type

I-Type

31:20	19:15	14:12	11:7	6:0
$imm_{11:0}$	rs1	funct3	rd	op
12 bits	5 bits	3 bits	5 bits	7 bits

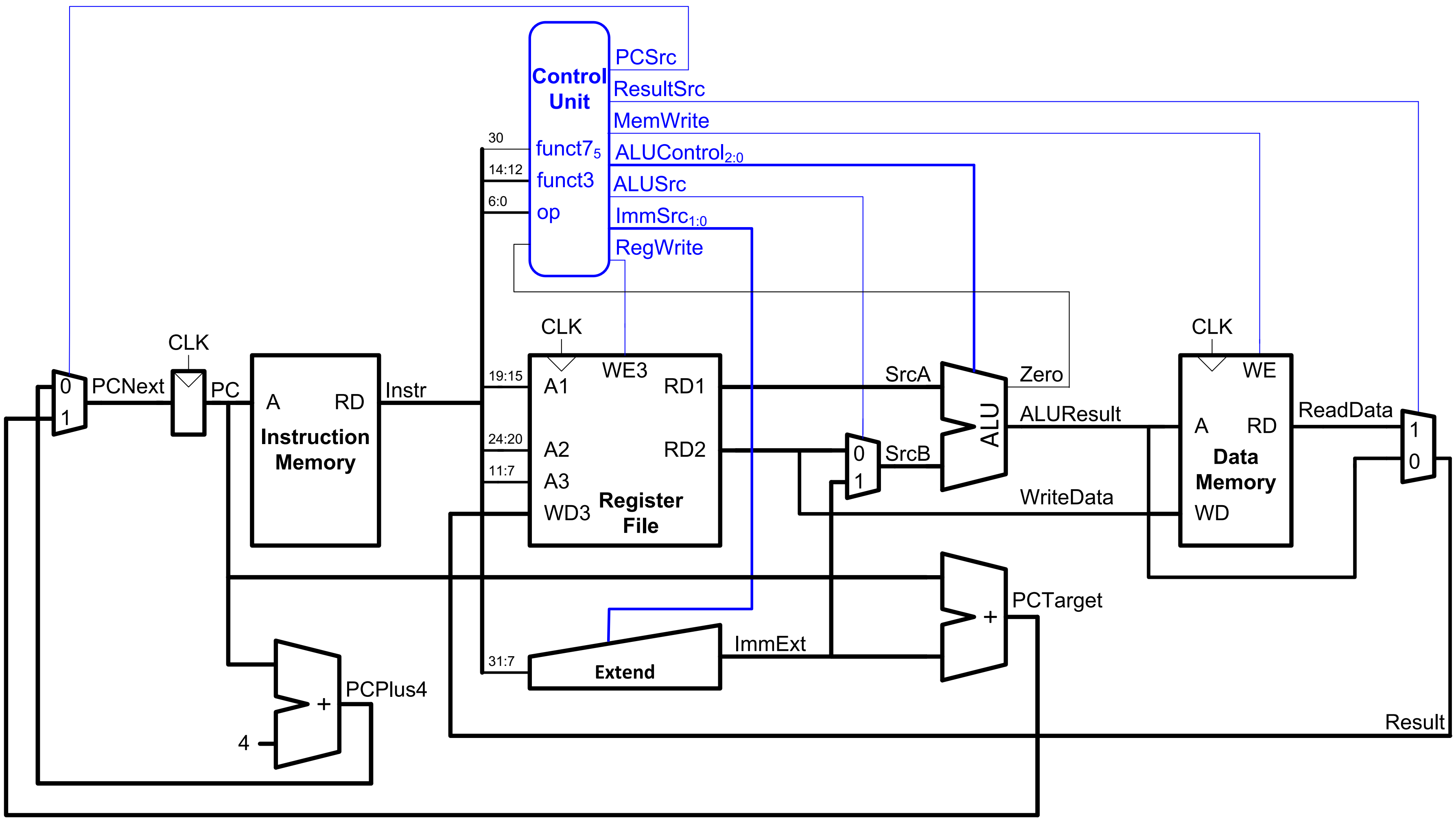
S-Type

31:25	24:20	19:15	14:12	11:7	6:0
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

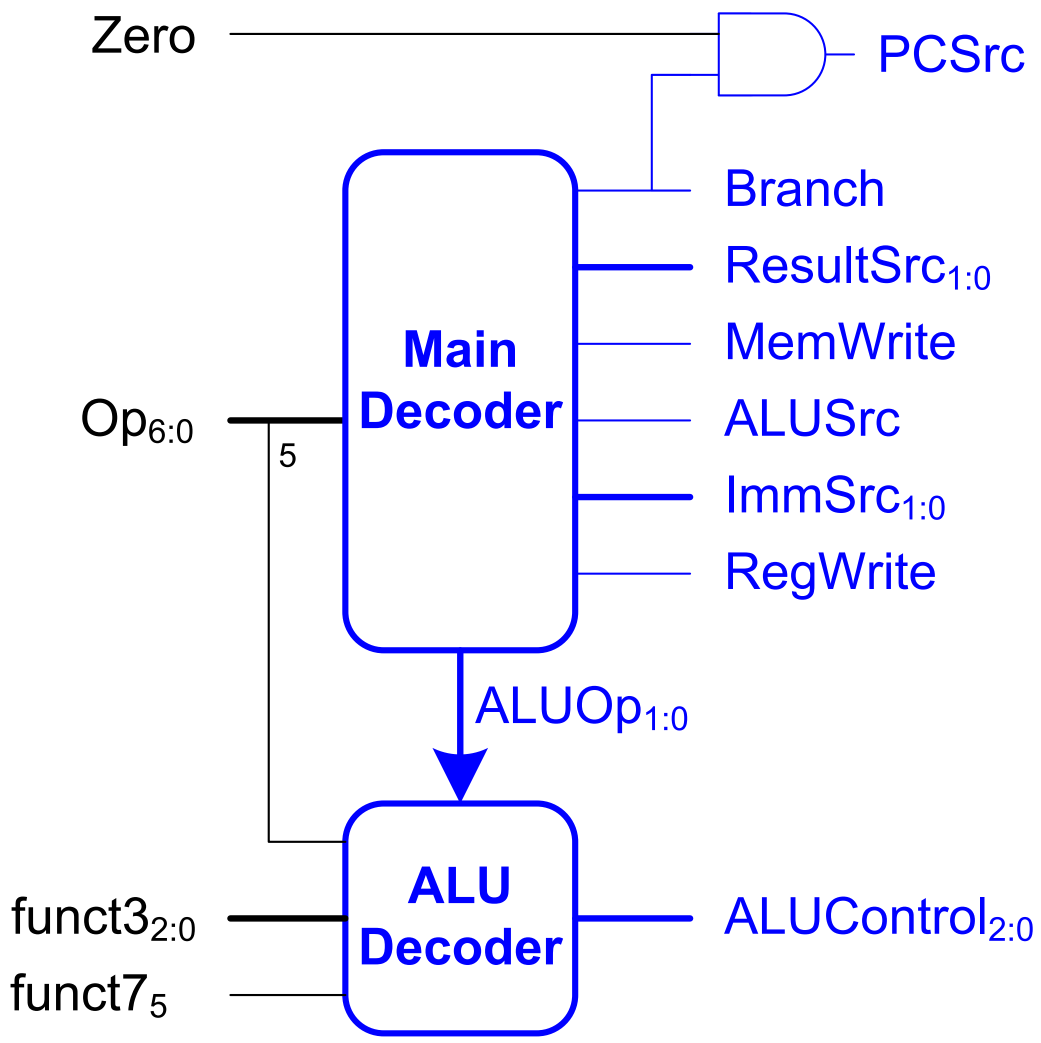
B-Type

31:25	24:20	19:15	14:12	11:7	6:0
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Single-Cycle RISC-V Processor

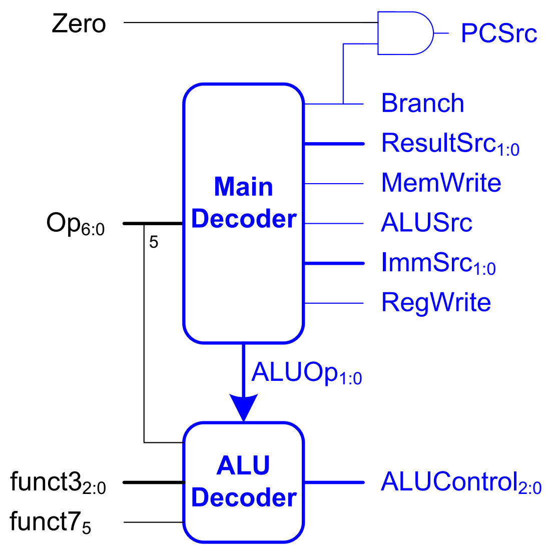


Single-Cycle Control

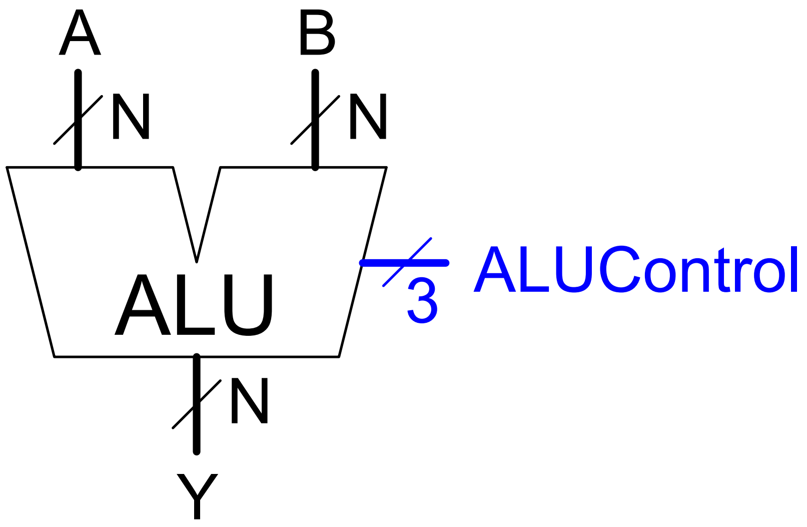


Control Unit: Main Decoder

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	`lw`	1	00	1	0	1	0	00
35	`sw`	0	01	1	1	X	0	00
51	R-type	1	XX	0	0	0	0	10
99	`beq`	0	10	0	0	X	1	01

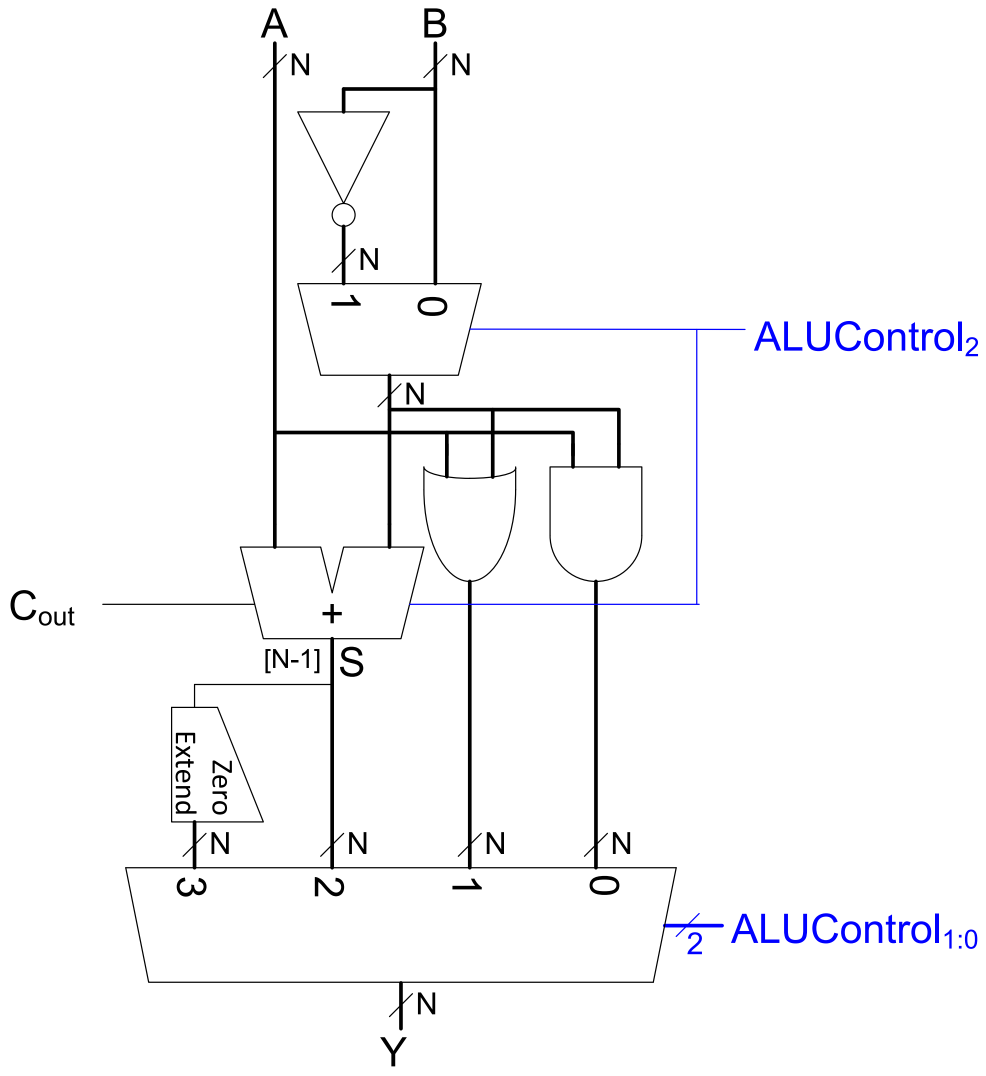


Review: ALU



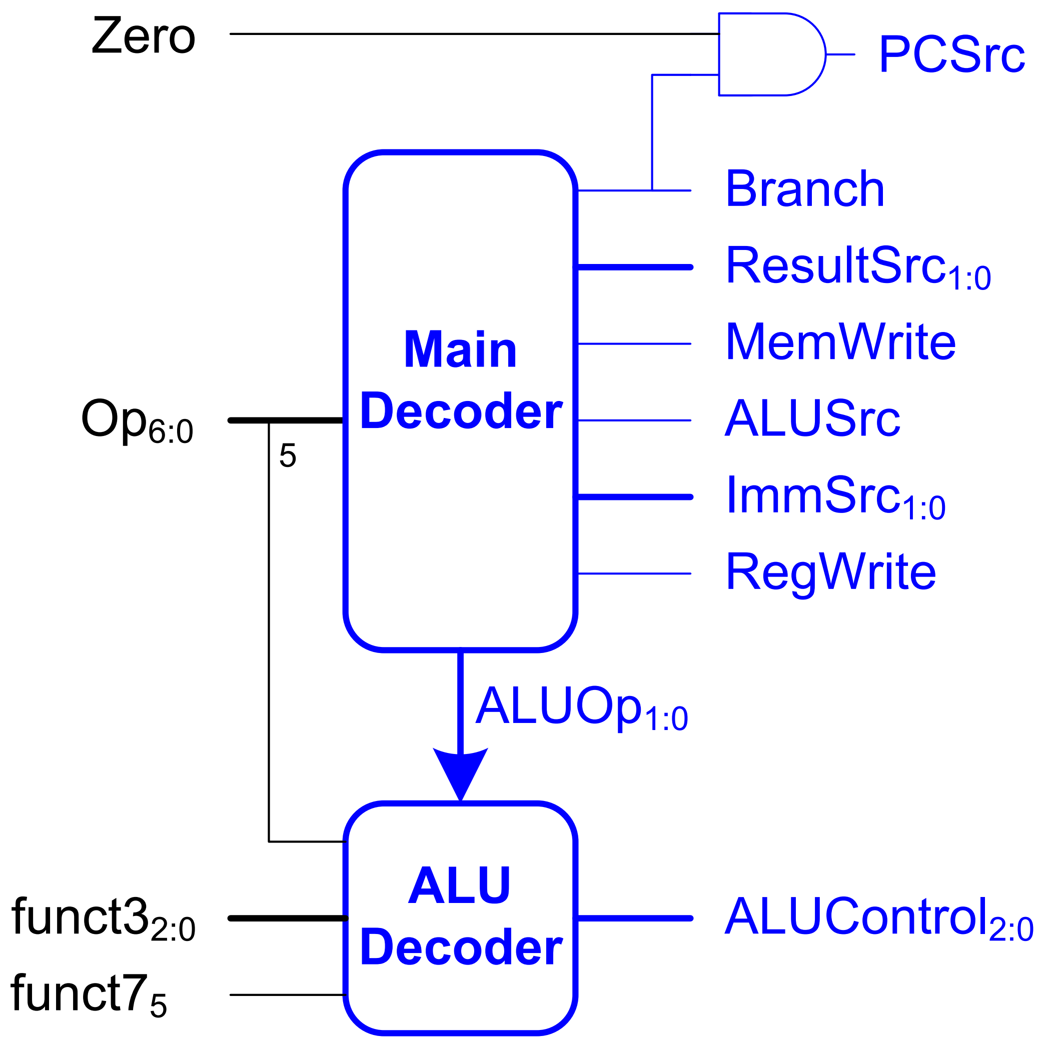
$ALUControl_{2:0}$	Function
000	A & B
001	A | B
010	A + B
110	A - B
111	SLT

Review: ALU



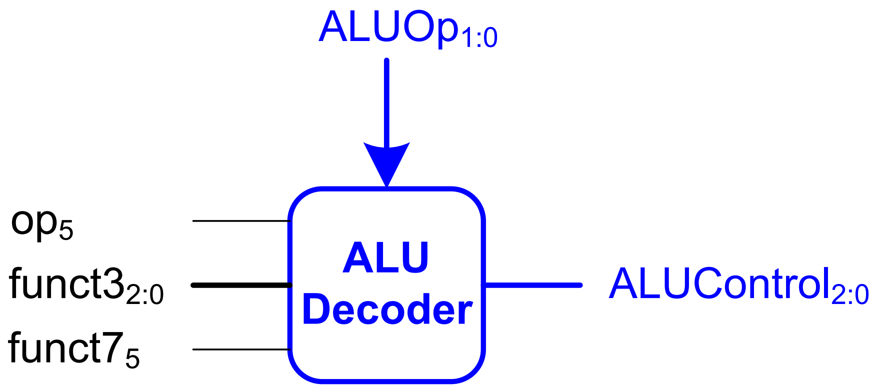
$ALUControl_{2:0}$	Function
000	A & B
001	A | B
010	A + B
110	A - B
111	SLT

Single-Cycle Control: ALU Decoder



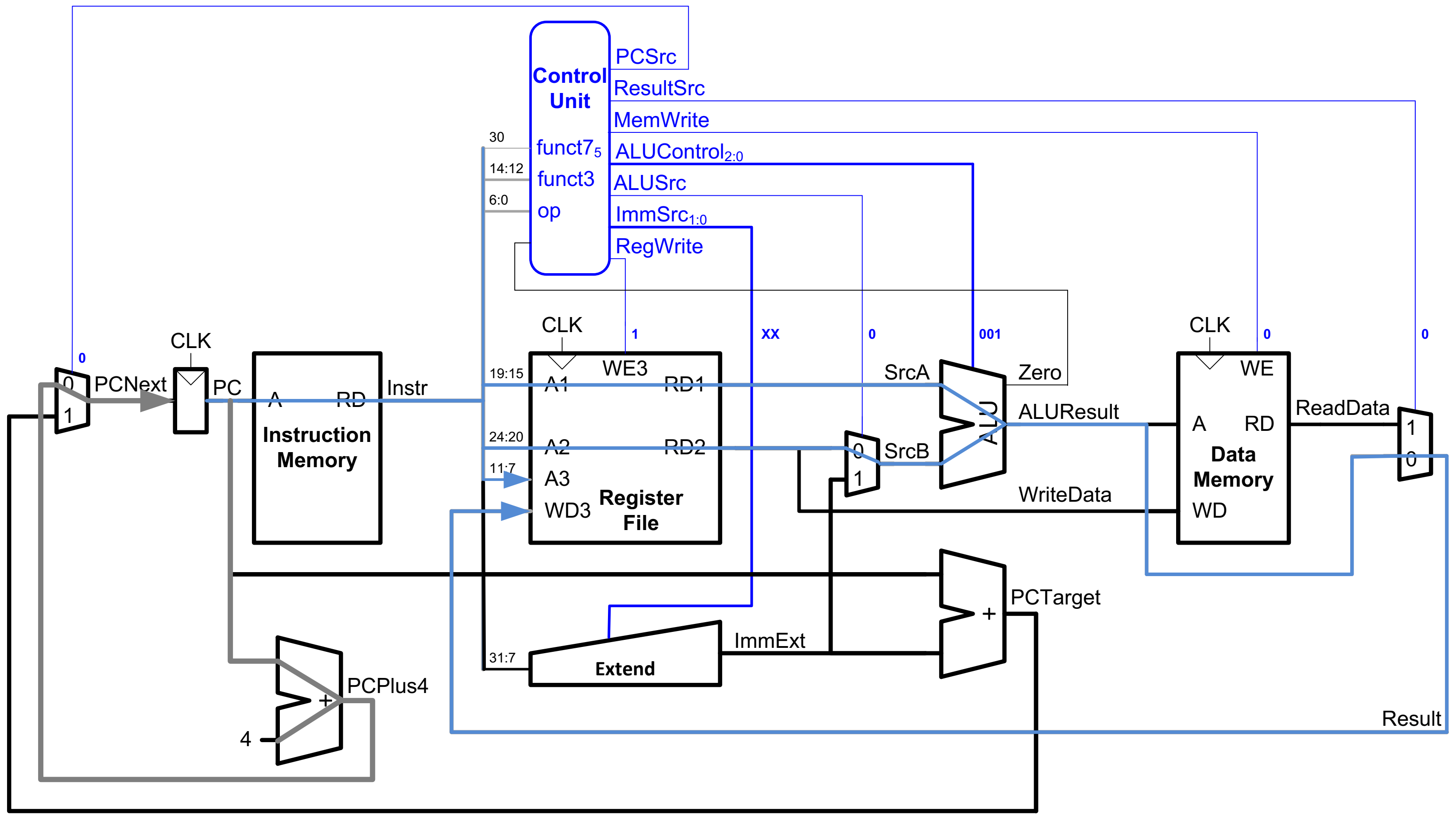
Control Unit: ALU Decoder

ALUOp	op_5	funct3	$funct7_5$	Instruction	$ALUControl_{2:0}$
00	X	X	X	`lw, sw`	010 (add)
01	X	X	X	`beq`	110 (subtract)
10	X	000	0	`add`	010 (add)
	1	000	1	`sub`	110 (subtract)
	X	010	0	`slt`	111 (set less than)
	X	110	0	`or`	001 (or)
	X	111	0	`and`	000 (and)

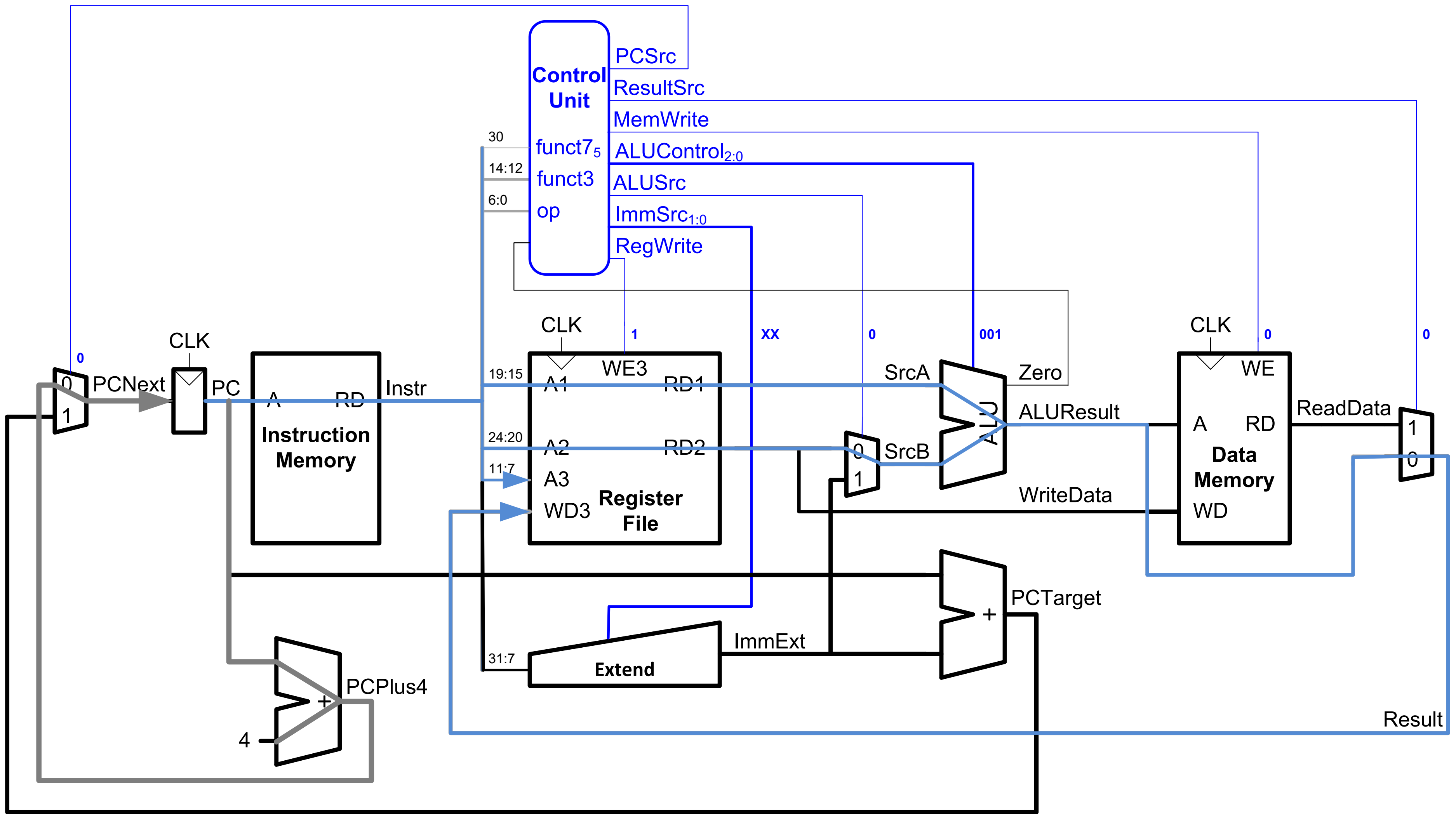


Example: `or`

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
51	R-type	1	XX	0	0	0	0	10



Example: `or`



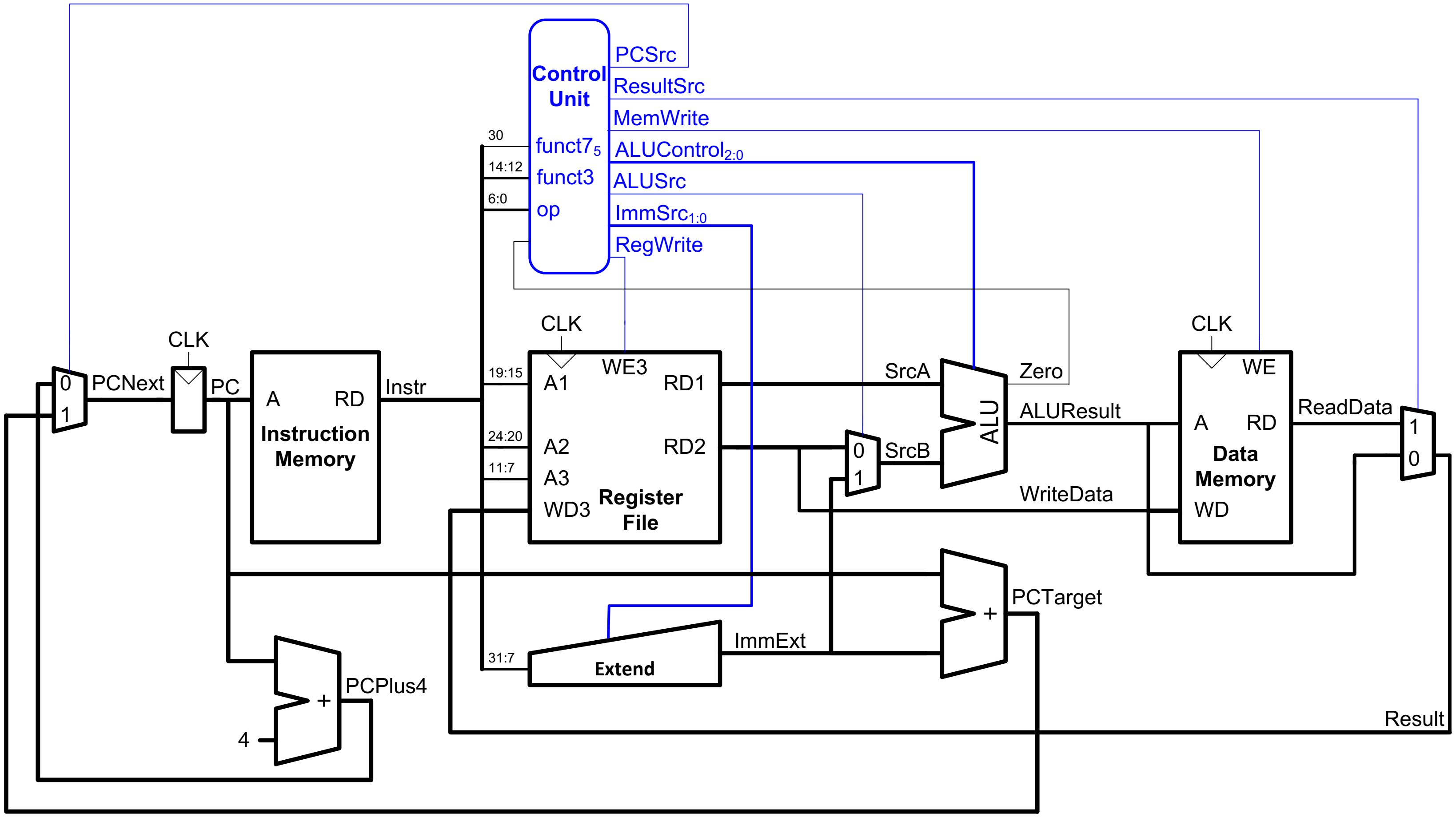
Control Unit
PCSrc
ResultSrc
MemWrite
ALUControl$_{2:0}$
ALUSrc
ImmSrc$_{1:0}$
RegWrite
funct7$_5$
funct3
op
30
14:12
6:0
CLK
1
XX
0
001
0
0
0
PCNext
PC
CLK
A
RD
Instruction Memory
Instr
19:15
24:20
11:7
31:7
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
CLK
WE
A
RD
WD
Data Memory
ReadData
Result
Extend
ImmExt
PCTarget
PCPlus4
4

Extended Functionality: addi

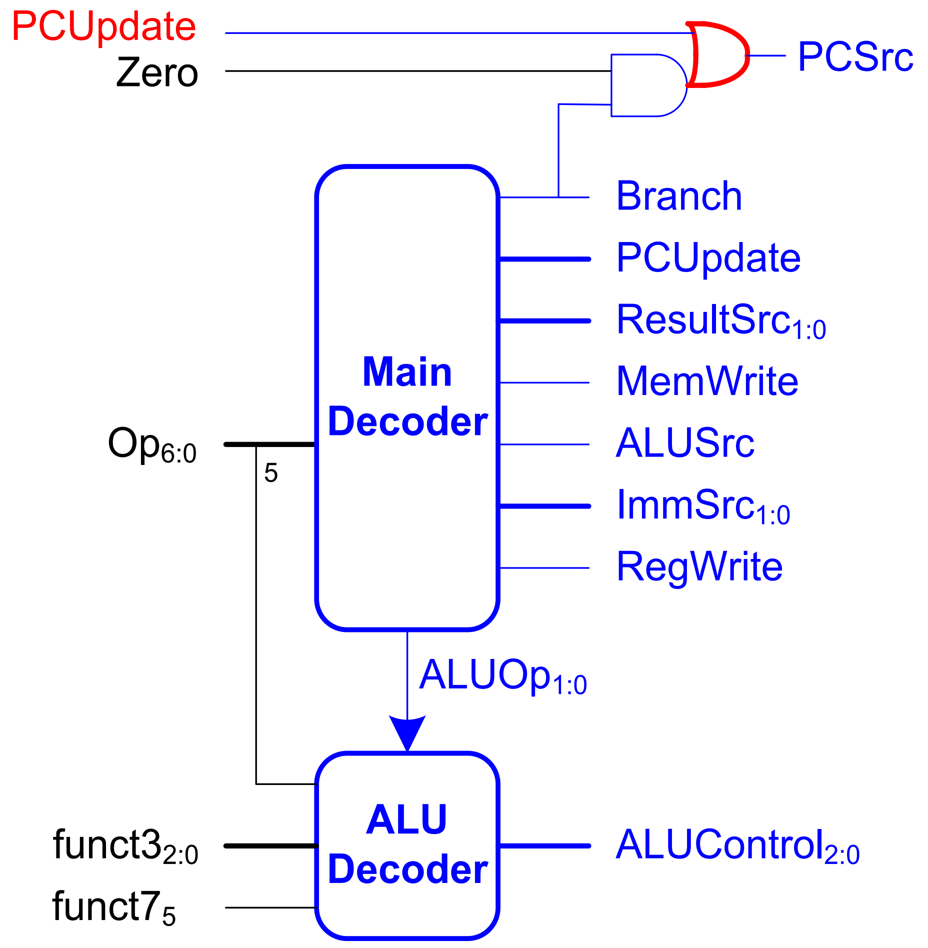
op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	lw	1	00	1	0	1	0	00
35	sw	0	01	1	1	X	0	00
51	R-type	1	XX	0	0	0	0	10
99	beq	0	10	0	0	X	1	01
19	**addi**	**1**	**00**	**1**	**0**	**0**	**0**	**10**

Extended Functionality: `addi`

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
19	addi	1	00	1	0	0	0	10



Extended Functionality: `jal`



Single-Cycle Datapath: ImmExt

$ImmSrc_{1:0}$	ImmExt	Instruction Type
00	{{20{instr[31]}}, instr[31:20]}	I-Type
01	{{20{instr[31]}}, instr[31:25], instr[11:7]}	S-Type
10	{{19{instr[31]}}, instr[31], instr[7], instr[30:25], instr[11:8], 1'b0}	B-Type
11	**{{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0}**	**J-Type**

I-Type

31:20	19:15	14:12	11:7	6:0
$imm_{11:0}$	rs1	funct3	rd	op
12 bits	5 bits	3 bits	5 bits	7 bits

B-Type

31:25	24:20	19:15	14:12	11:7	6:0
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

S-Type

31:25	24:20	19:15	14:12	11:7	6:0
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

J-Type

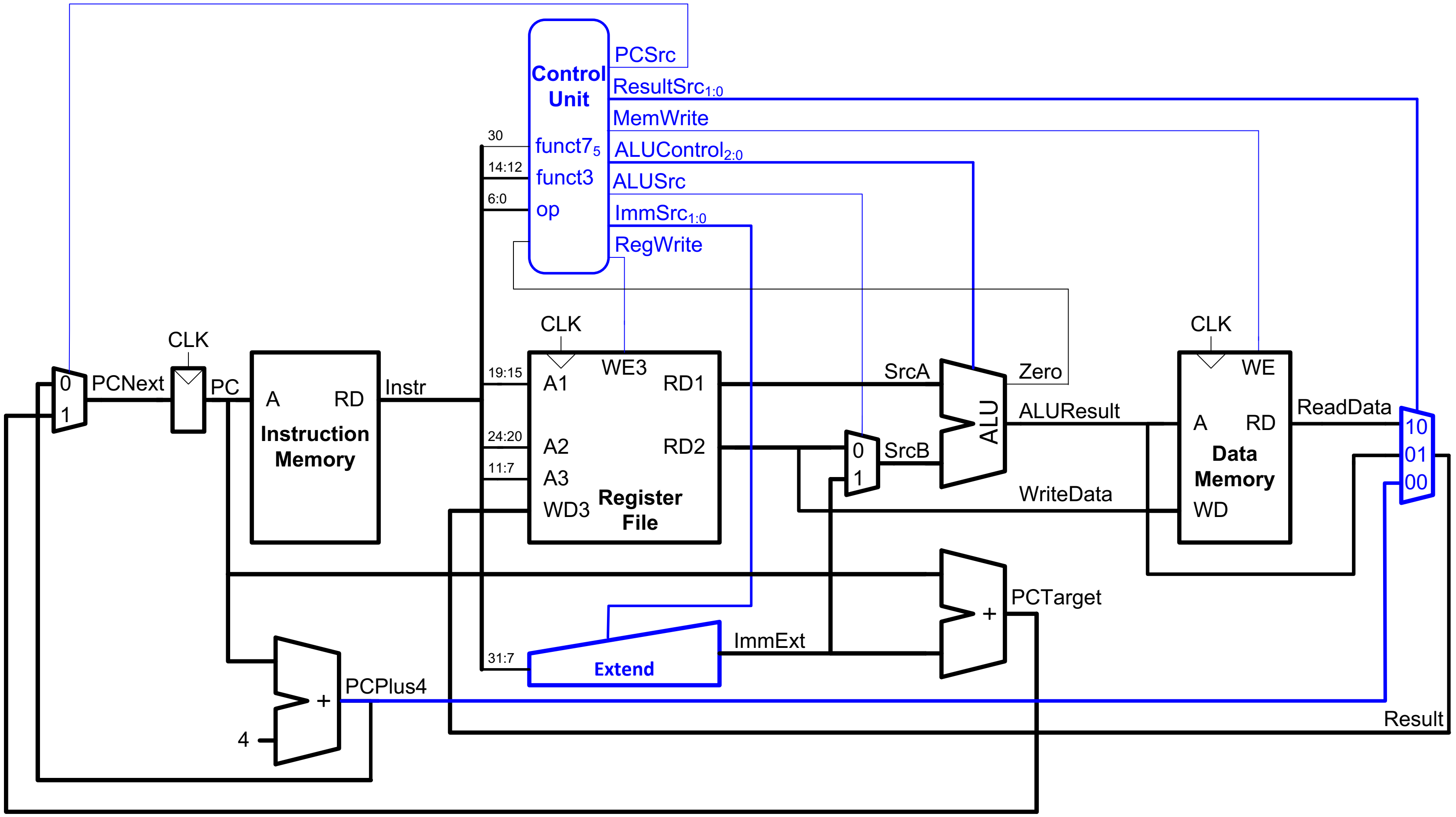
31:12	11:7	6:0
$imm_{20,10:1,11,19:12}$	rd	op
20 bits	5 bits	7 bits

Extended Functionality: `jal`

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	PCUpdate
3	`lw`	1	00	1	0	10	0	00	**0**
35	`sw`	0	01	1	1	XX	0	00	**0**
51	**R-type**	1	XX	0	0	01	0	10	**0**
99	`beq`	0	10	0	0	XX	1	01	**0**
19	`addi`	1	00	1	0	01	0	10	**0**
111	`jal`	**0**	**11**	**X**	**0**	**00**	**0**	**XX**	**1**

Extended Functionality: `jal`

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	PCUpdate
111	`jal`	0	11	X	0	00	0	XX	1



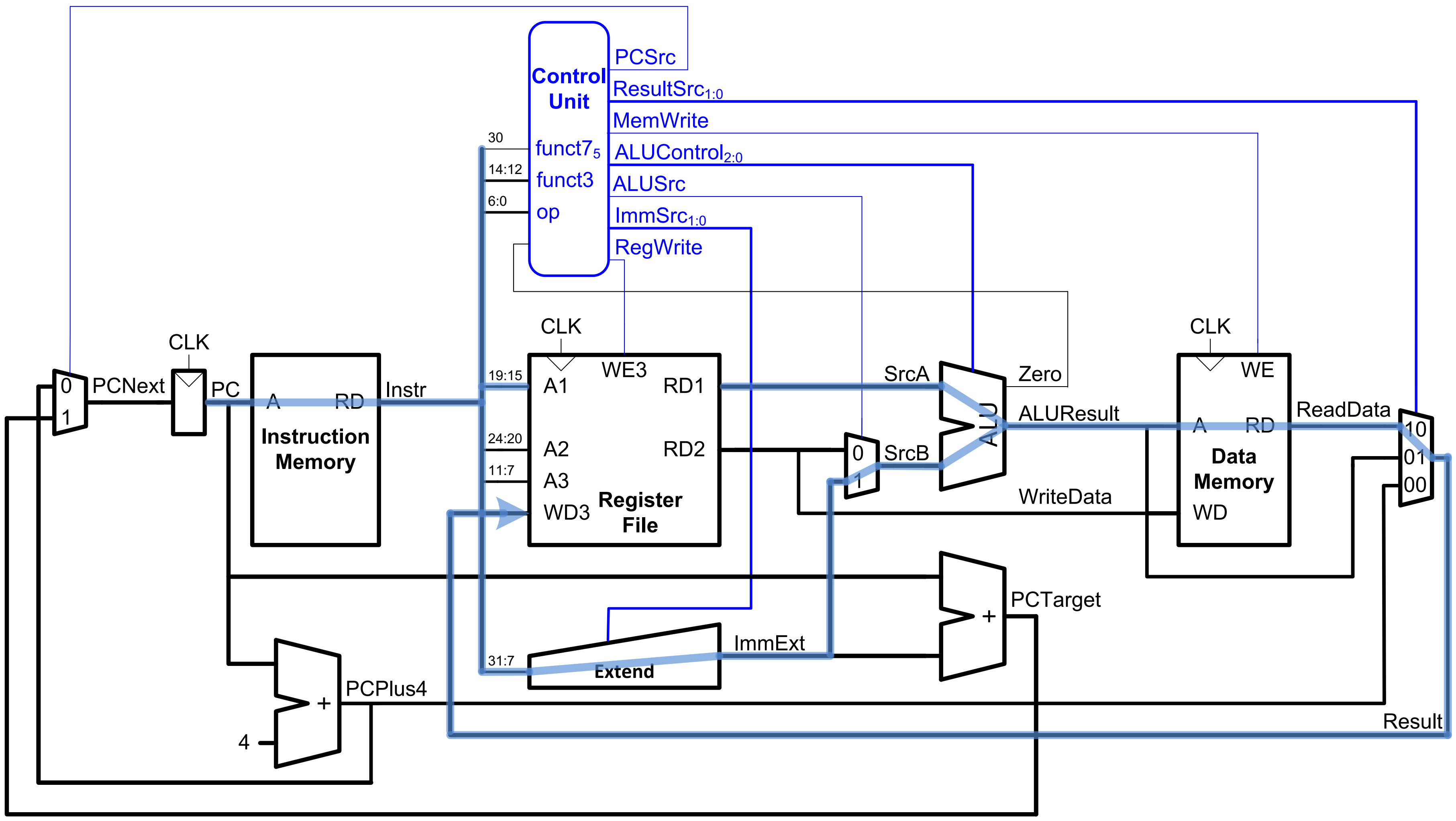
Processor Performance

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:**

$$T_{c1} = t_{pcq\_PC} + t_{\text{mem}} + \max[t_{mux} + t_{RF\text{read}}, t_{ext} + t_{\text{mux}}] + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$$

- **Typically, limiting paths are:**
  - memory, ALU, register file
  - $T_{c1} = t_{pcq\_PC} + 2t_{\text{mem}} + t_{RF\text{read}} + t_{\text{ALU}} + 2t_{\text{mux}} + t_{RF\text{setup}}$

Single-Cycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	25
ALU	t_{ALU}	120
Decoder (Control Unit)	t_{dec}	70
Memory read	t_{mem}	200
Register file read	$t_{RF\mathrm{read}}$	100
Register file setup	$t_{RF\mathrm{setup}}$	60

$T_{c1} = ?$

Single-Cycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	25
ALU	t_{ALU}	120
Decoder (Control Unit)	t_{dec}	70
Memory read	t_{mem}	200
Register file read	$t_{RF\text{read}}$	100
Register file setup	$t_{RF\text{setup}}$	60

$$T_{c1} = t_{pcq\_PC} + 2t_{\text{mem}} + t_{dec} + t_{RF\text{read}} + t_{\text{ALU}} + 2t_{\text{mux}} + t_{RF\text{setup}}$$
$$= [50 + 2(200) + 70 + 100 + 120 + 2(25) + 60] \text{ ps}$$
$$= \mathbf{840\ ps}$$

Single-Cycle Performance Example

Program with 100 billion instructions:

Execution Time = # instructions x CPI x T_C

= $(100 \times 10^9)(1)(840 \times 10^{-12}\ \text{s})$

= **84 seconds**

Multicycle RISC-V Processor

- **Single-cycle:**
  - + simple
  - - cycle time limited by longest instruction (`lw`)
  - - separate memories for instruction and data
  - - 3 adders/ALUs
- **Multicycle processor addresses these issues by breaking instruction into shorter steps**
  - o shorter instructions take fewer steps
  - o can re-use hardware
  - o cycle time is faster

Multicycle RISC-V Processor

- **Single-cycle:**
  - + simple
  - - cycle time limited by longest instruction (`LDR`)
  - - separate memories for instruction and data
  - - 3 adders/ALUs
- **Multicycle:**
  - + higher clock speed
  - + simpler instructions run faster
  - + reuse expensive hardware on multiple cycles
  - - sequencing overhead paid many times

Multicycle RISC-V Processor

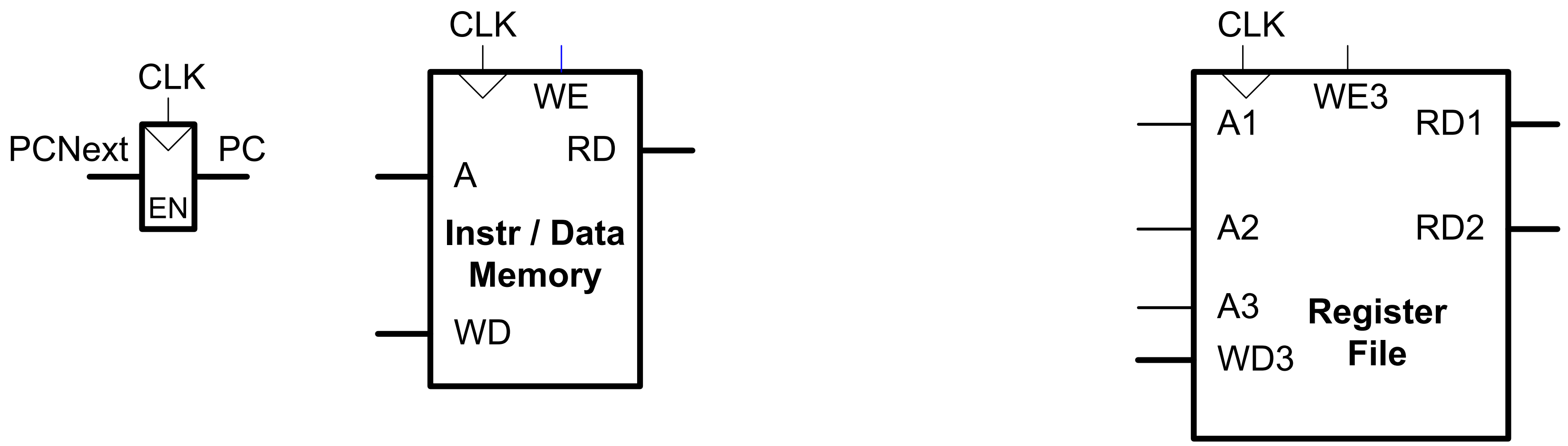
- **Single-cycle:**
  - + simple
  - - cycle time limited by longest instruction (`LDR`)
  - - separate memories for instruction and data
  - - 3 adders/ALUs
- **Multicycle:**
  - + higher clock speed
  - + simpler instructions run faster
  - + reuse expensive hardware on multiple cycles
  - - sequencing overhead paid many times

Same design steps as single-cycle:

- **first datapath**
- **then control**

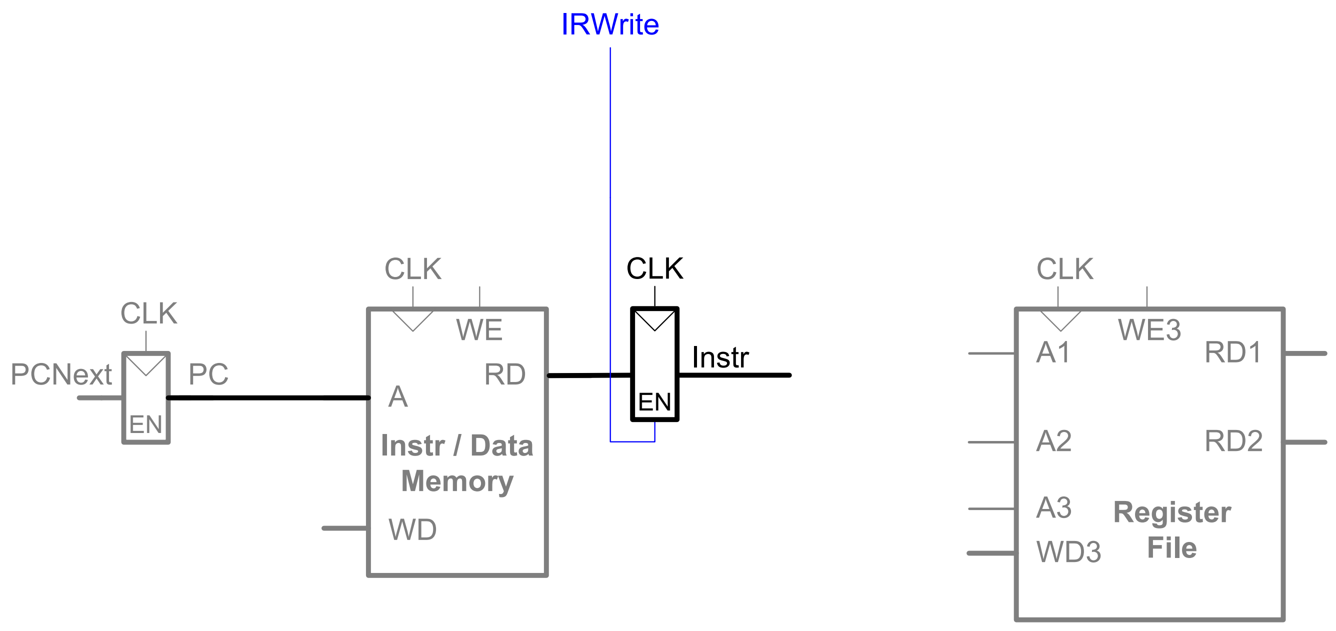
Multicycle State Elements

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



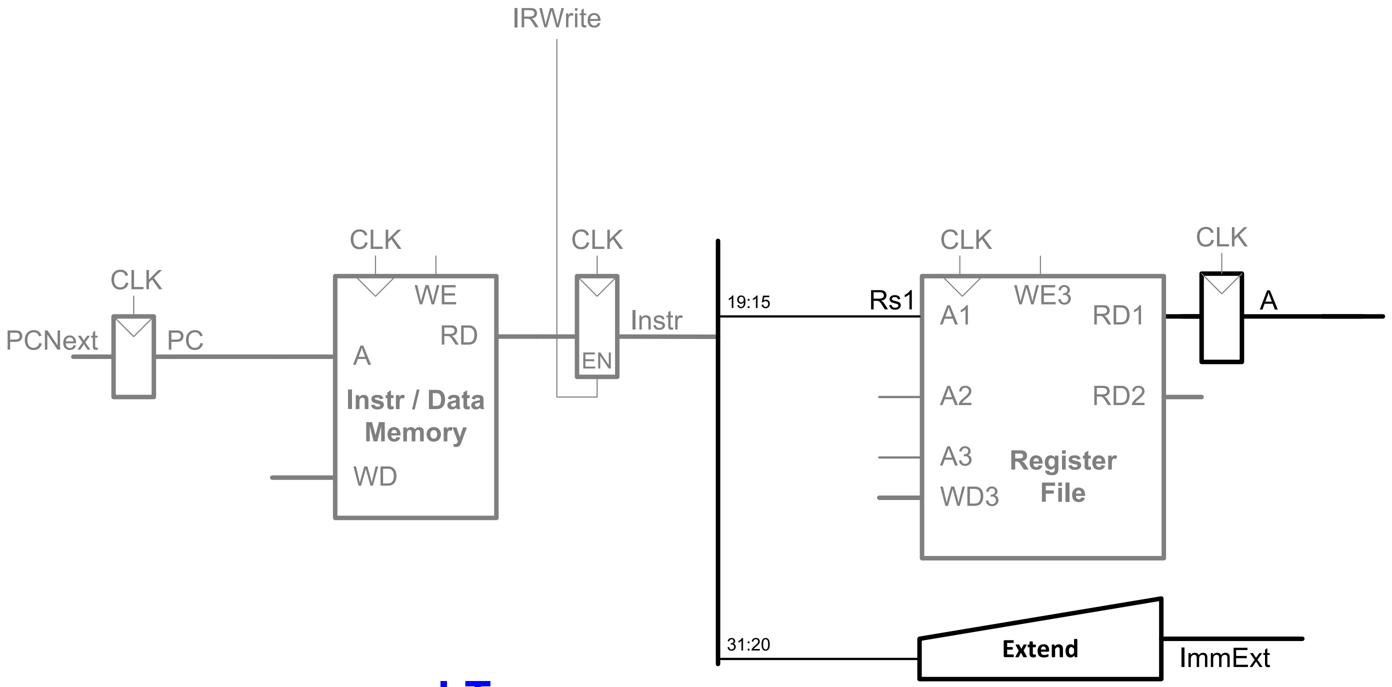
Multicycle Datapath: Instruction Fetch

STEP 1: Fetch instruction



Multicycle Datapath: `lw` get sources

STEP 2: Read source operand from RF and extend immediate



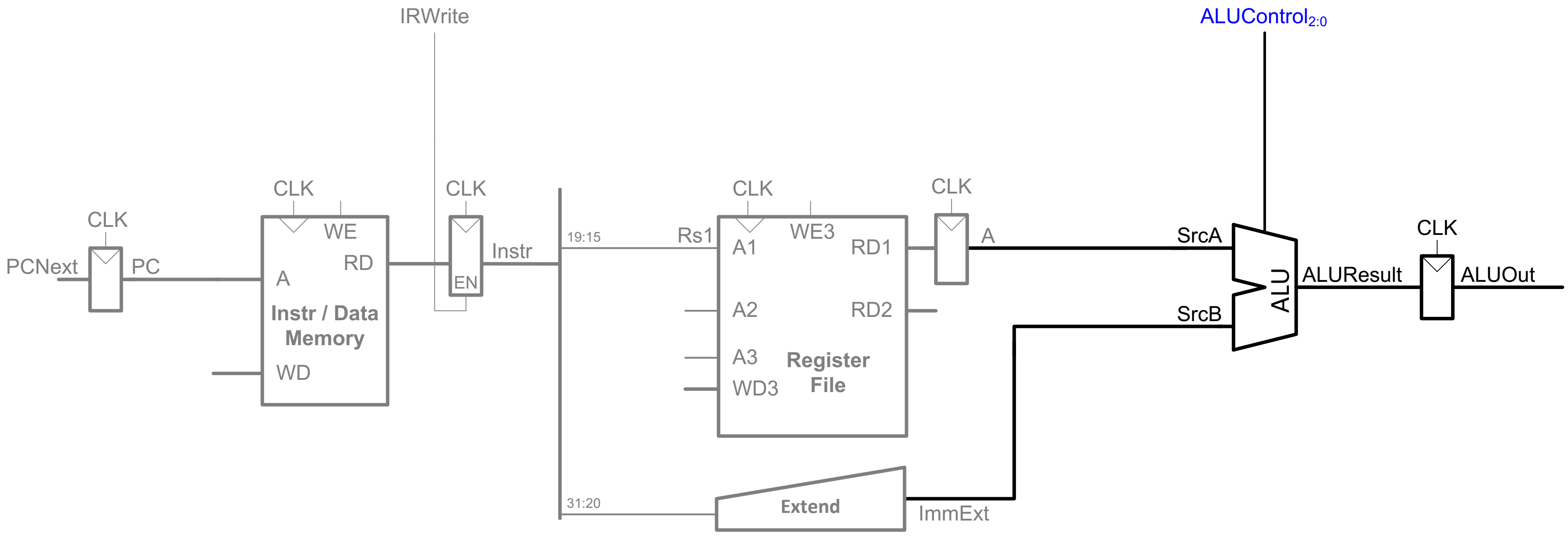
I-Type

31:20	19:15	14:12	11:7	6:0
$imm_{11:0}$	rs1	funct3	rd	op
12 bits	5 bits	3 bits	5 bits	7 bits

```
lw rd, imm(rs1)
```

Multicycle Datapath: `lw` Address

STEP 3: Compute the memory address



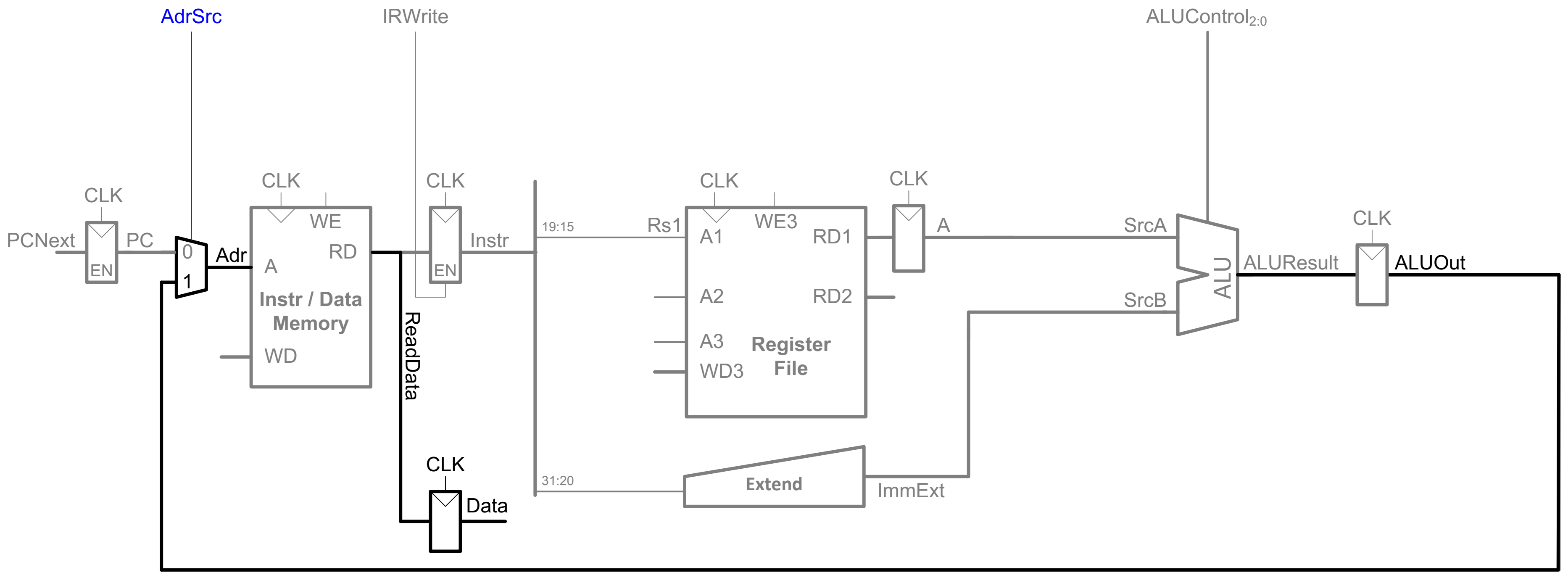
I-Type

31:20	19:15	14:12	11:7	6:0
$imm_{11:0}$	rs1	funct3	rd	op
12 bits	5 bits	3 bits	5 bits	7 bits

`lw rd, imm(rs1)`

Multicycle Datapath: `lw` Memory Read

STEP 4: Read data from memory



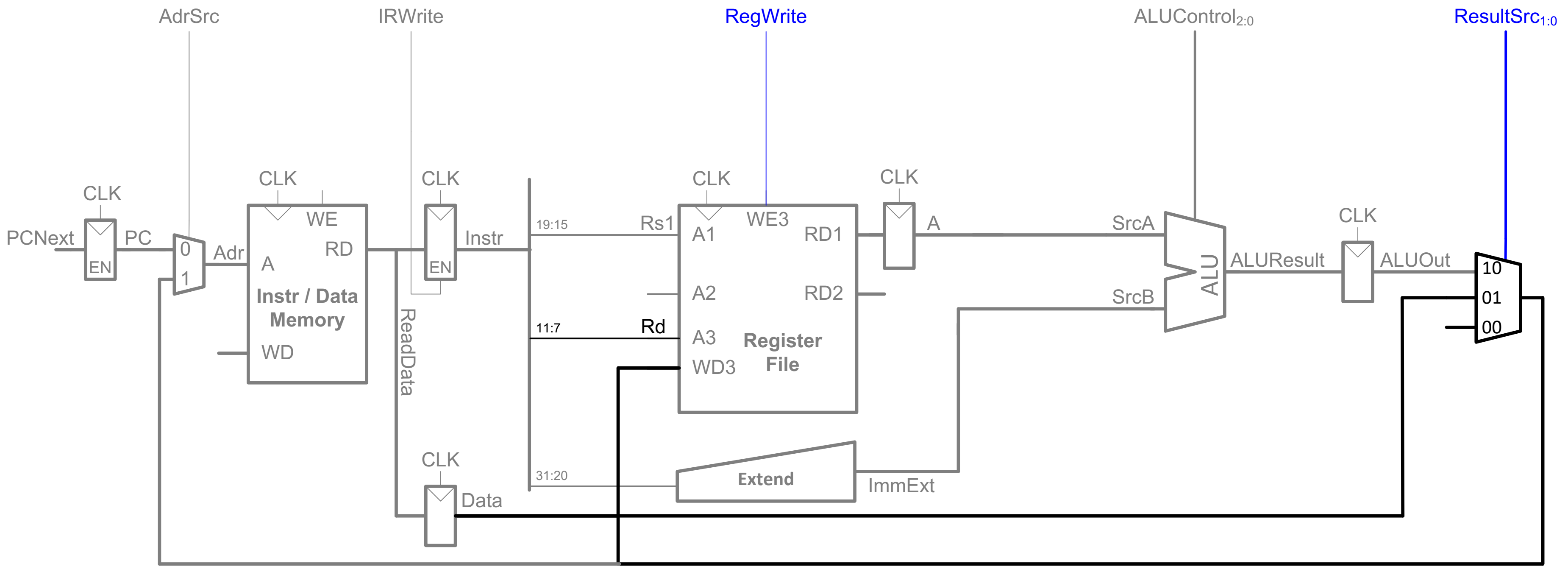
I-Type

31:20	19:15	14:12	11:7	6:0
$imm_{11:0}$	rs1	funct3	rd	op
12 bits	5 bits	3 bits	5 bits	7 bits

```
lw rd, imm(rs1)
```

Multicycle Datapath: `lw` Write Register

STEP 5: Write data back to register file



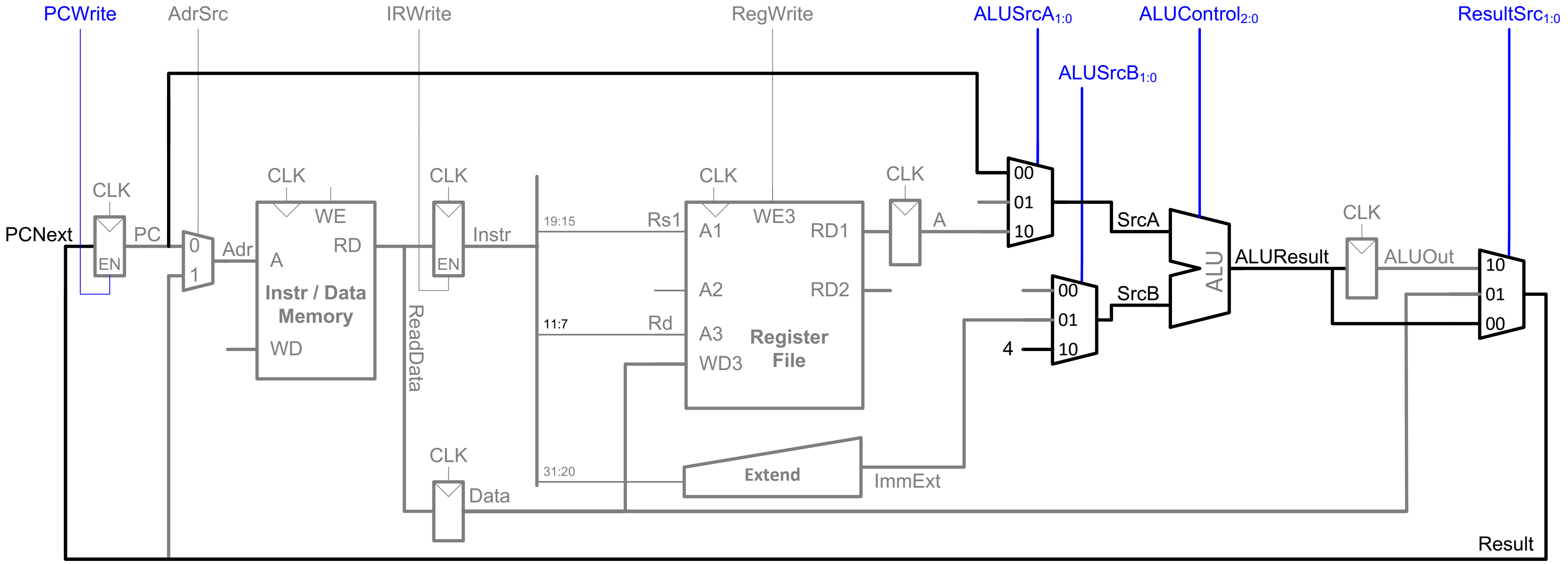
I-Type

31:20	19:15	14:12	11:7	6:0
imm$_{11:0}$	rs1	funct3	rd	op
12 bits	5 bits	3 bits	5 bits	7 bits

`lw rd, imm(rs1)`

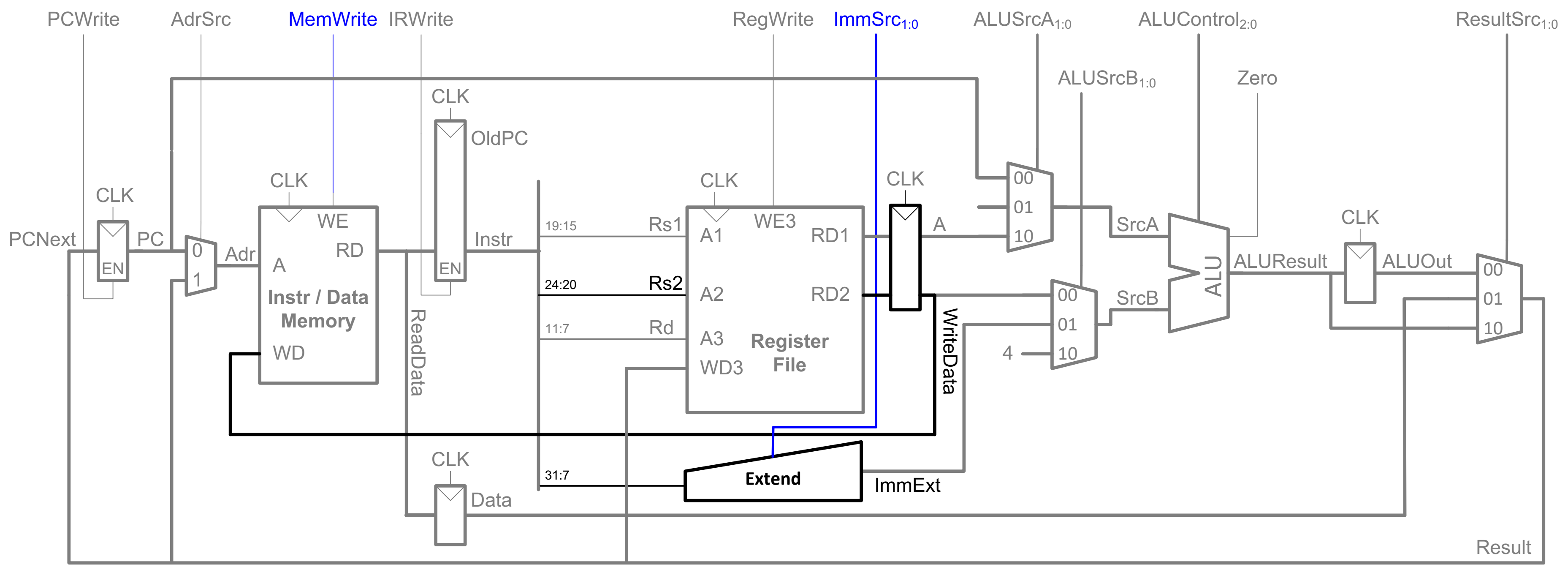
Multicycle Datapath: Increment PC

STEP 6: Increment PC: PC = PC+4



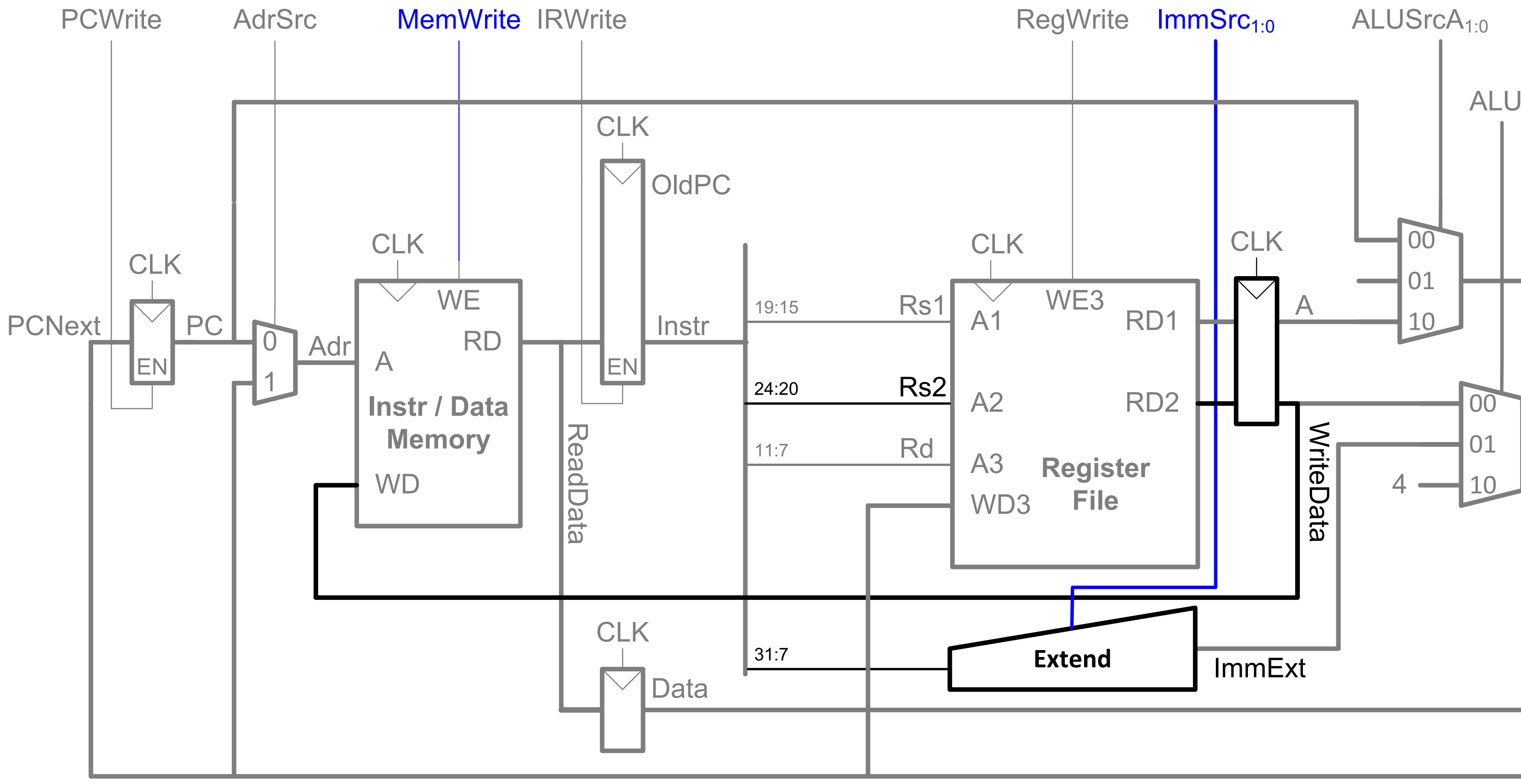
Multicycle Datapath: `sw`

Write data in `rs2` to memory



Multicycle Datapath: `sw`

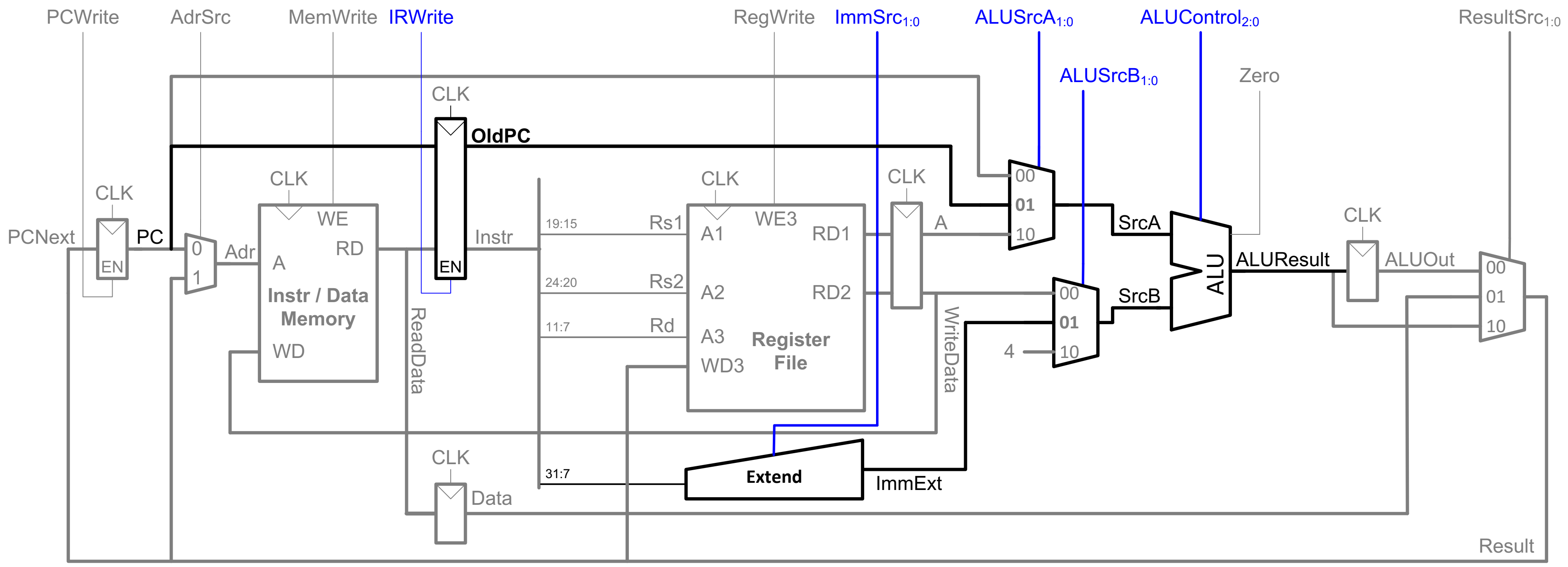
Write data in `rs2` to memory



Multicycle Datapath: beq

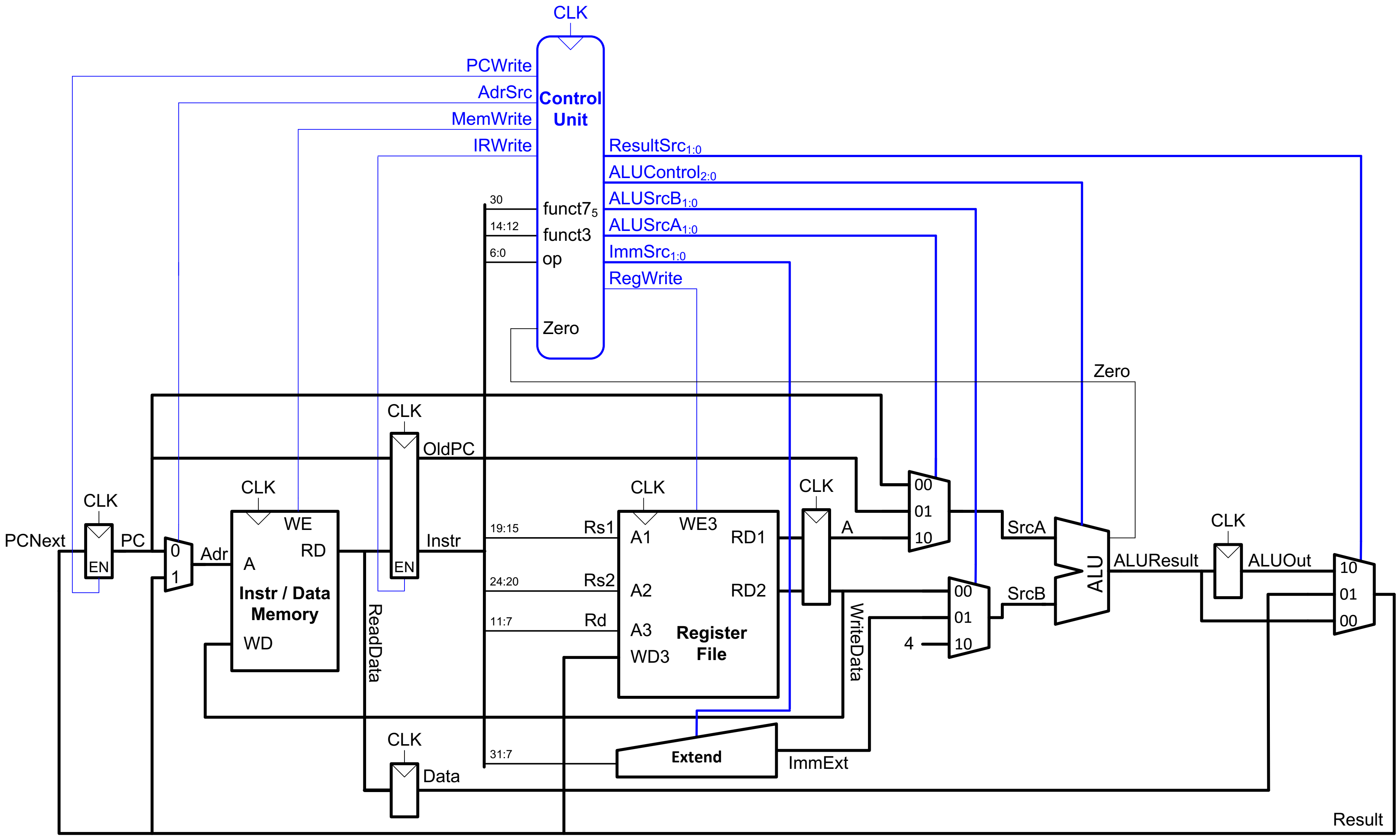
Calculate branch target address:

BTA = PC + imm

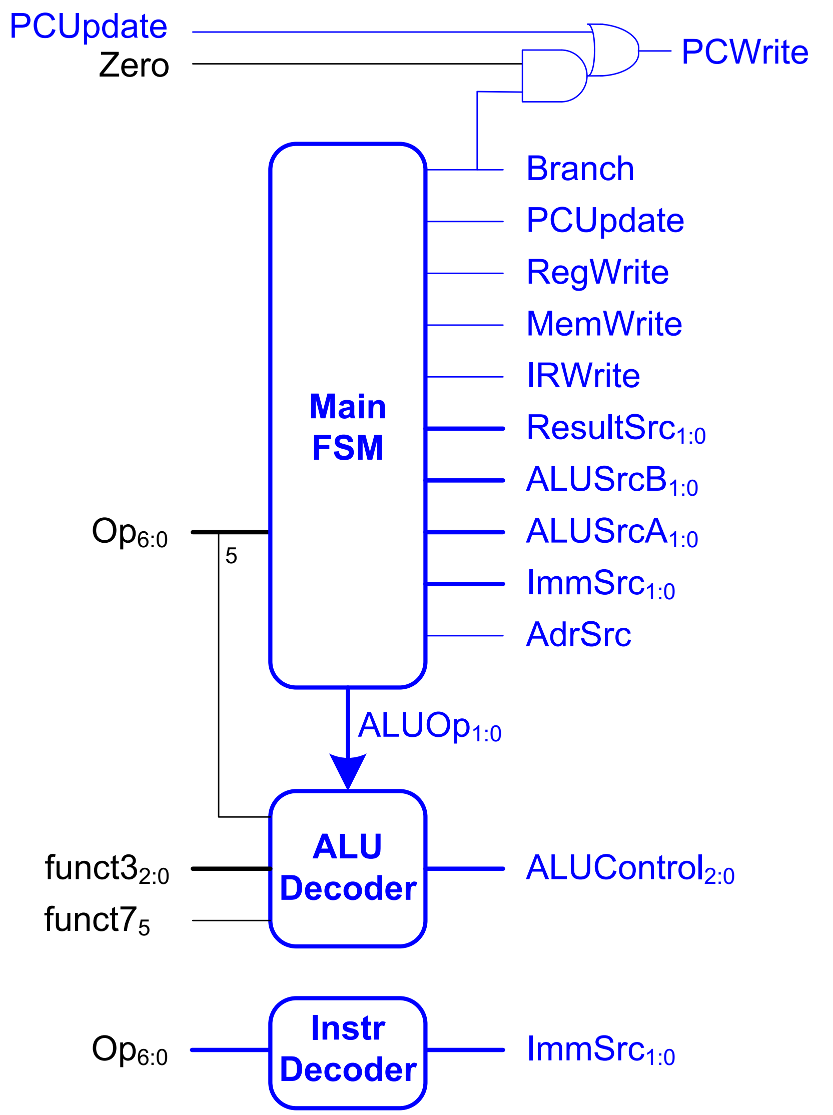


PC was already updated in Fetch stage, so need to save **old PC**

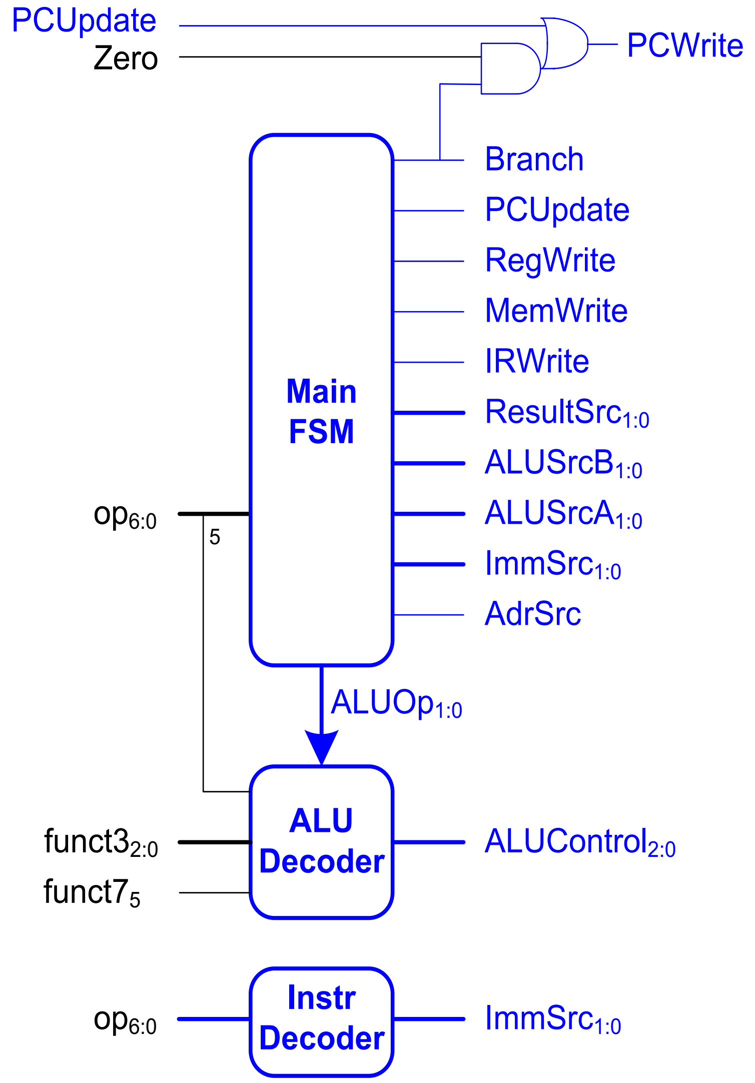
Multicycle RISC-V Processor



Multicycle Control

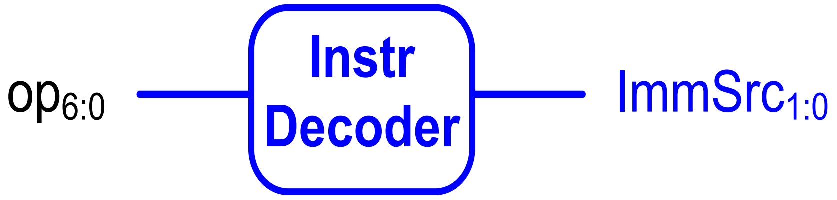


Multicycle Control



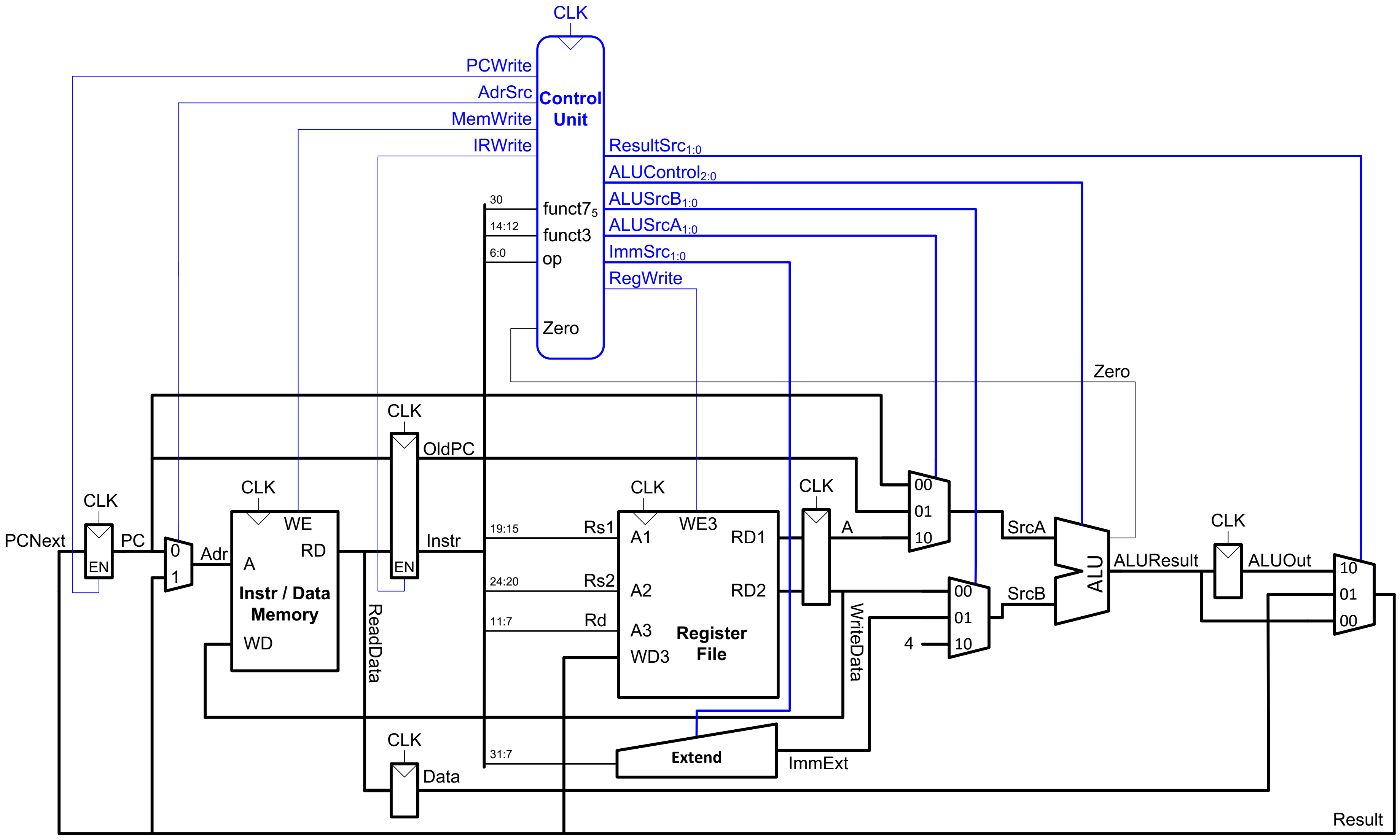
ALU Decoder **same as single-cycle**

Multicycle Control: Instr Decoder

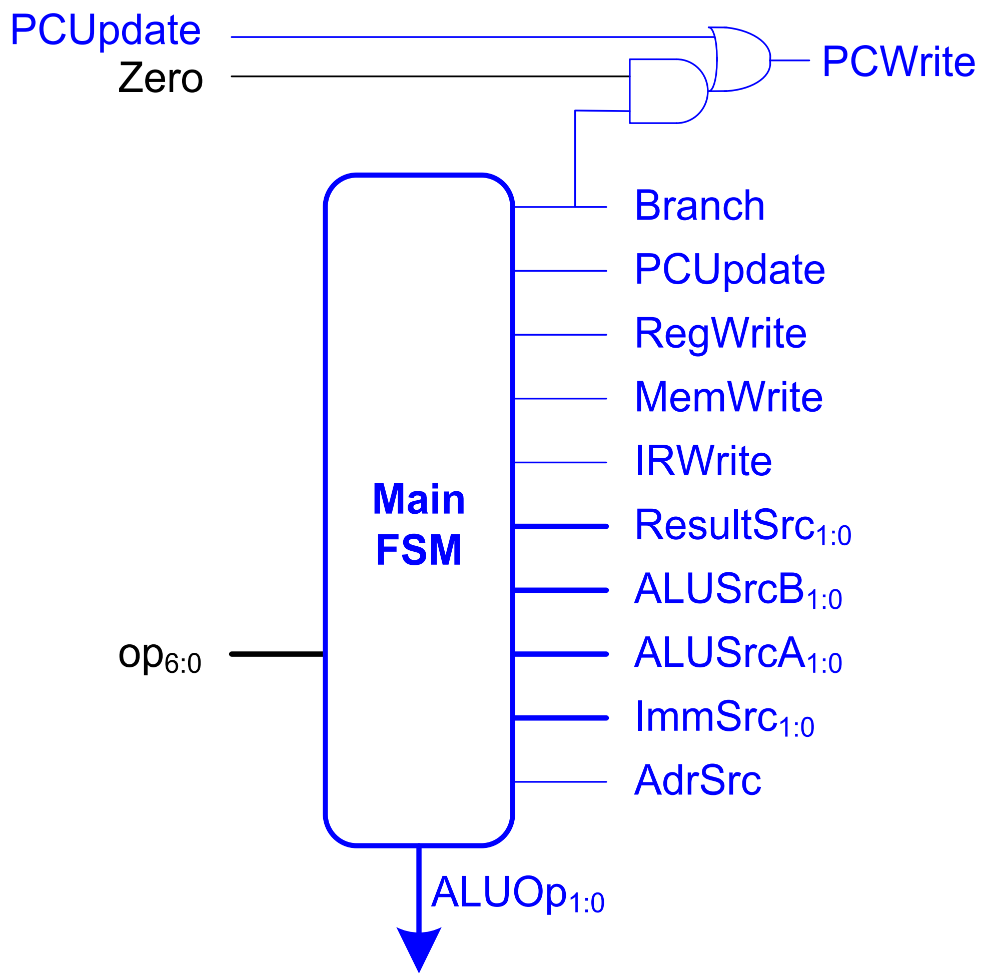


op	Instruction	ImmSrc
3	`lw`	00
35	`sw`	01
51	R-type	XX
99	`beq`	10

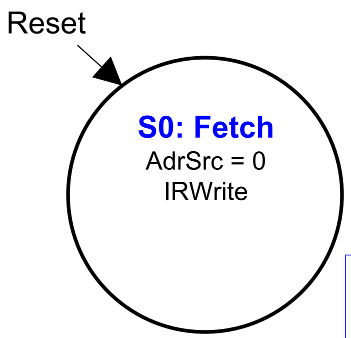
Multicycle RISC-V Processor

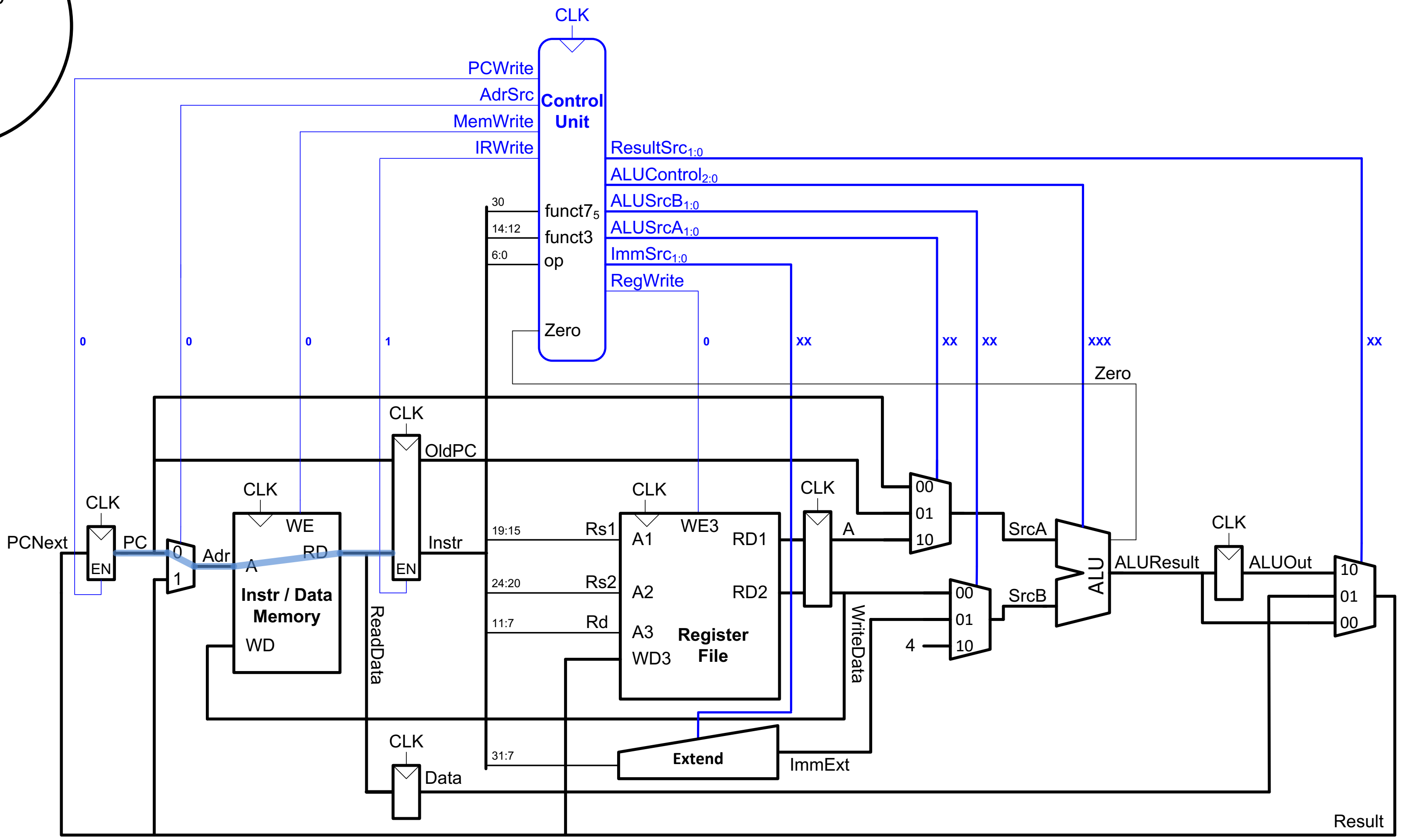


Multicycle Control: Main FSM

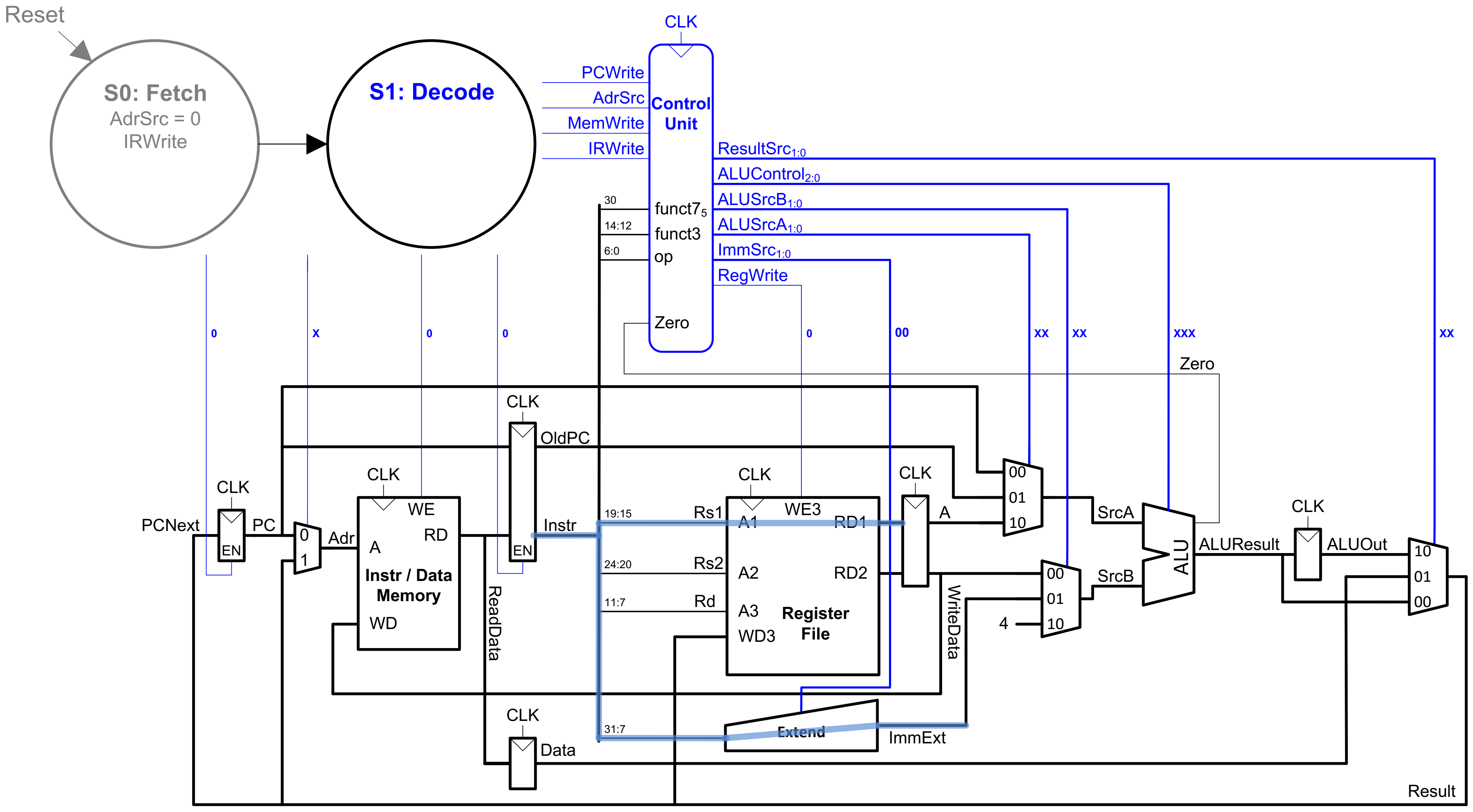


Main Controller FSM: Fetch

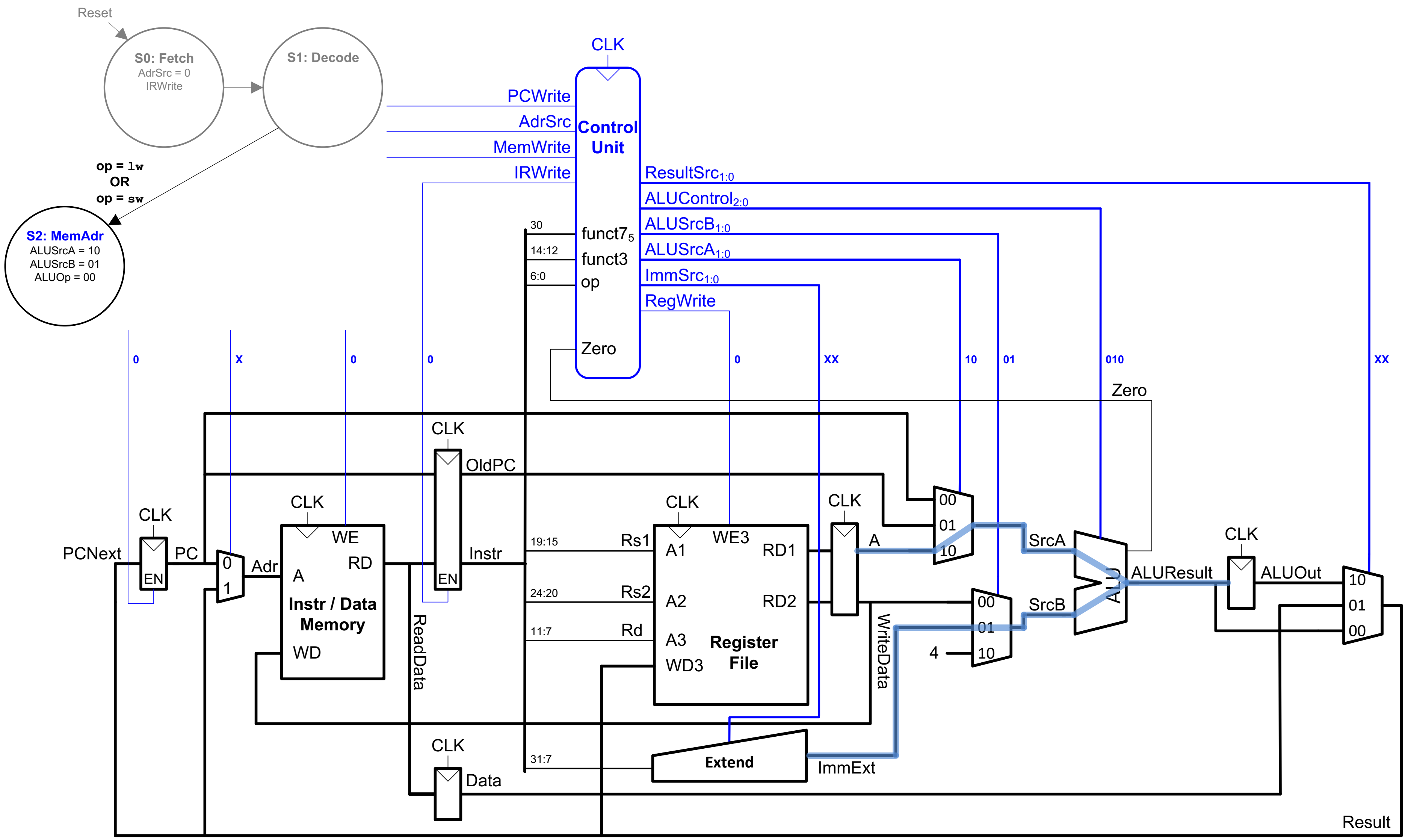




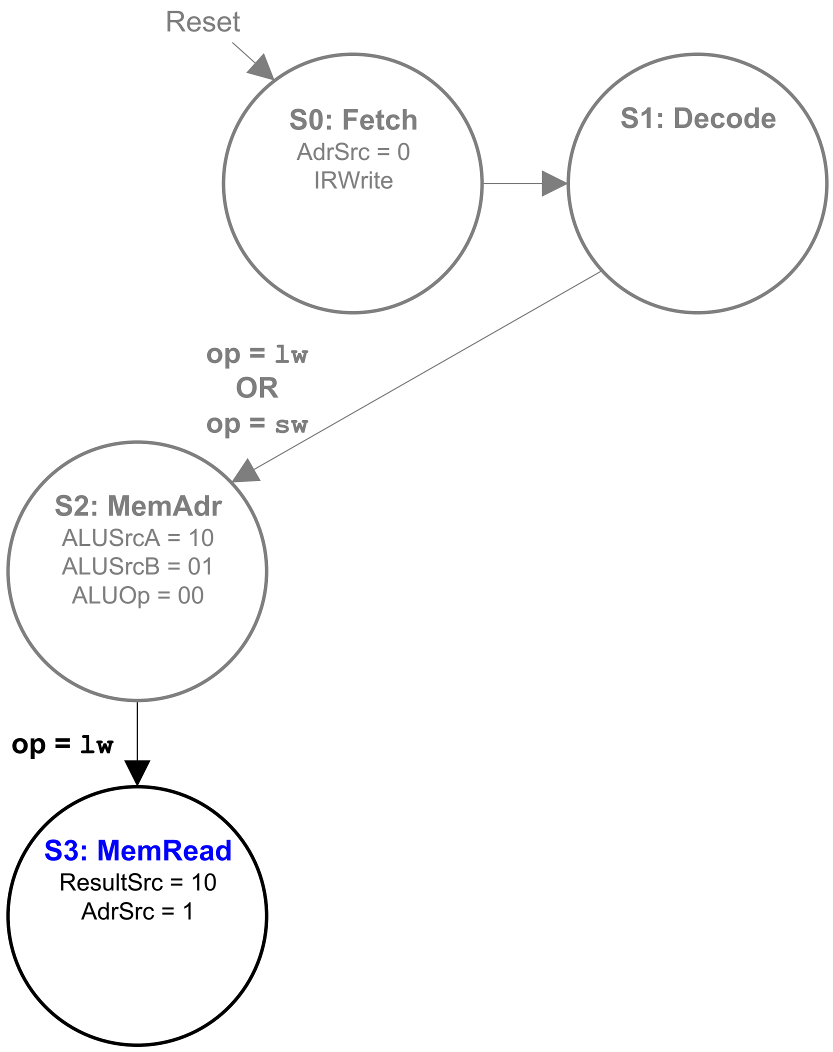
Main Controller FSM: Decode



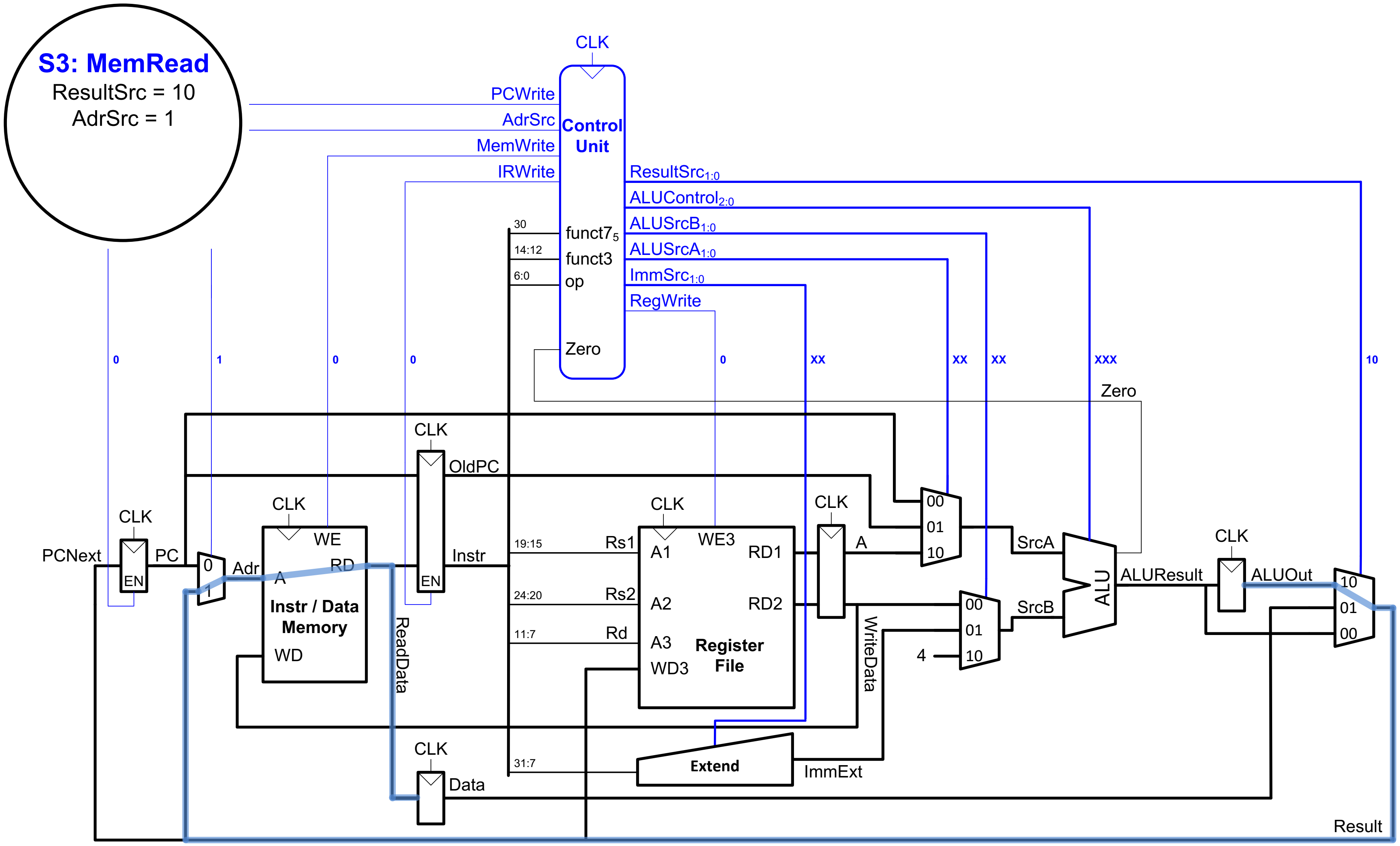
Main Controller FSM: Address



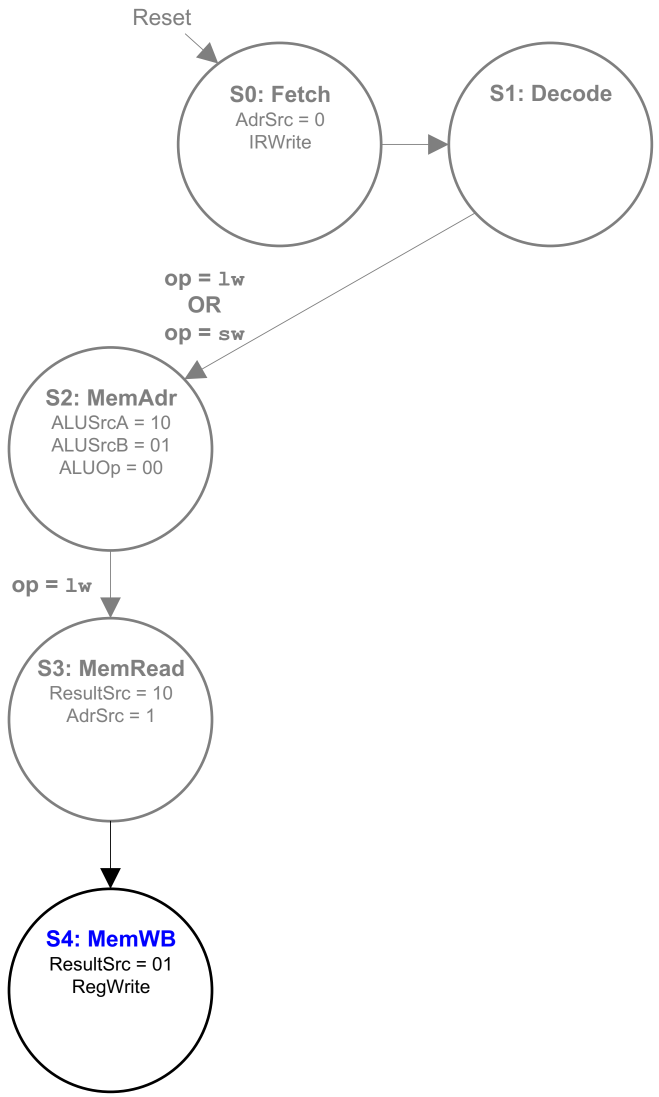
Main Controller FSM: Read Memory



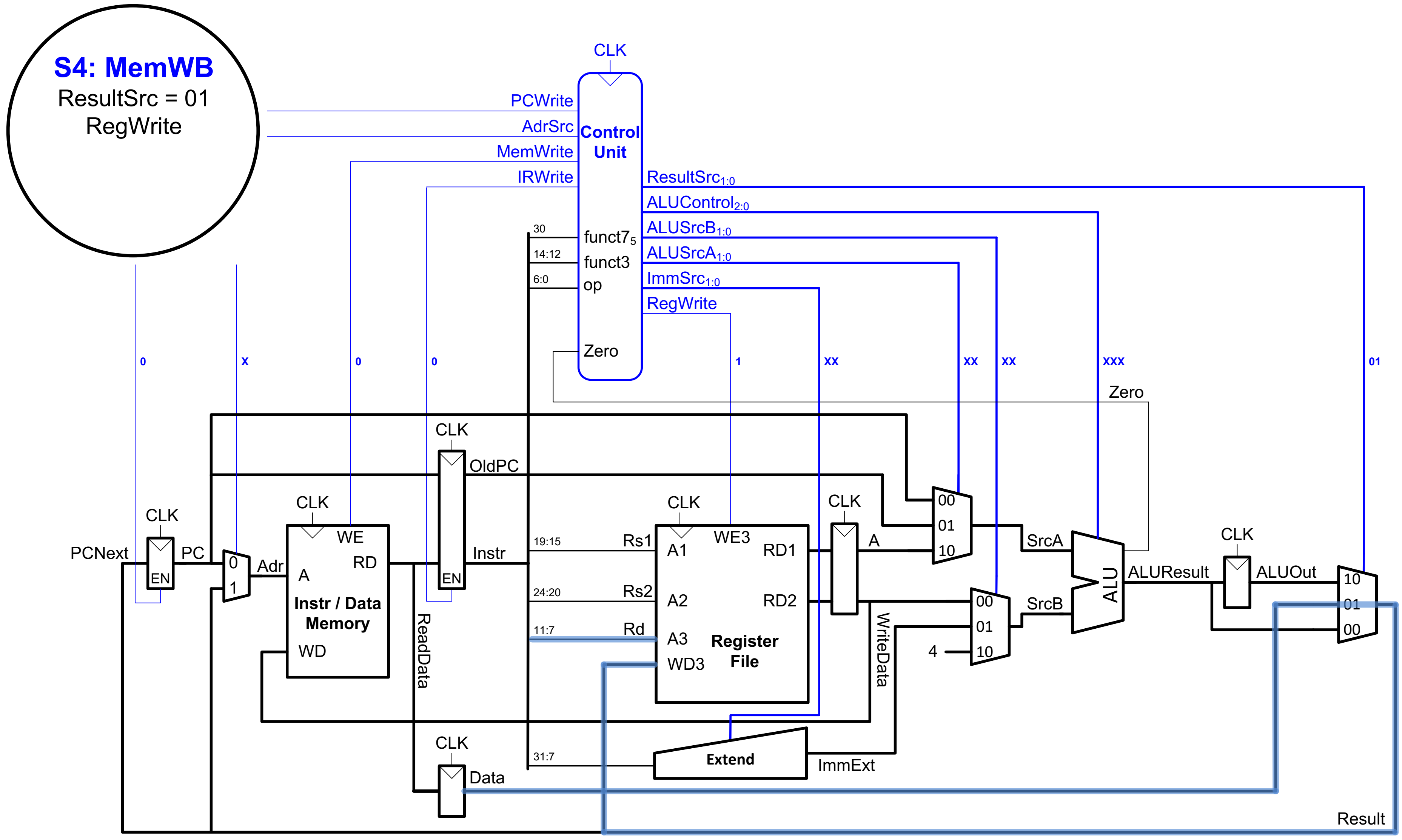
Multicycle RISC-V Processor



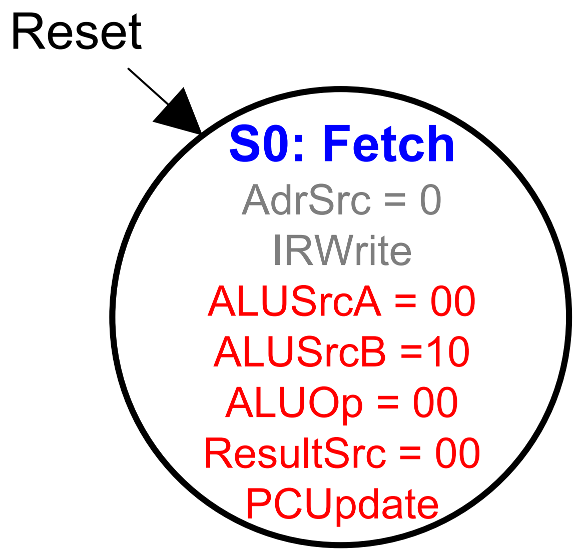
Main Controller FSM: Write RF



Main Controller FSM: Write RF

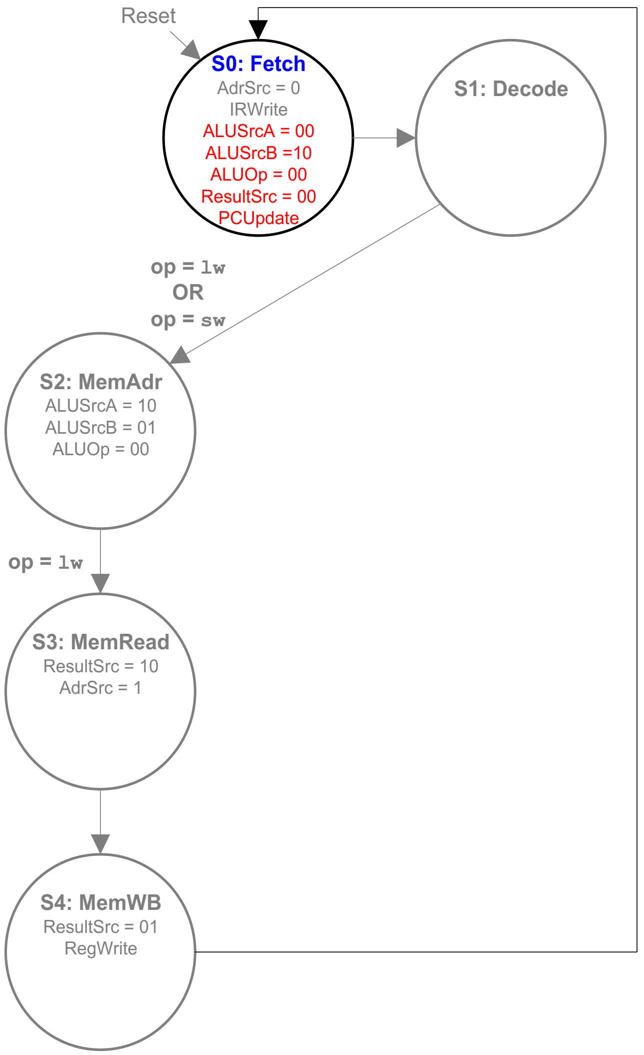


Main Controller FSM: Fetch Revisited



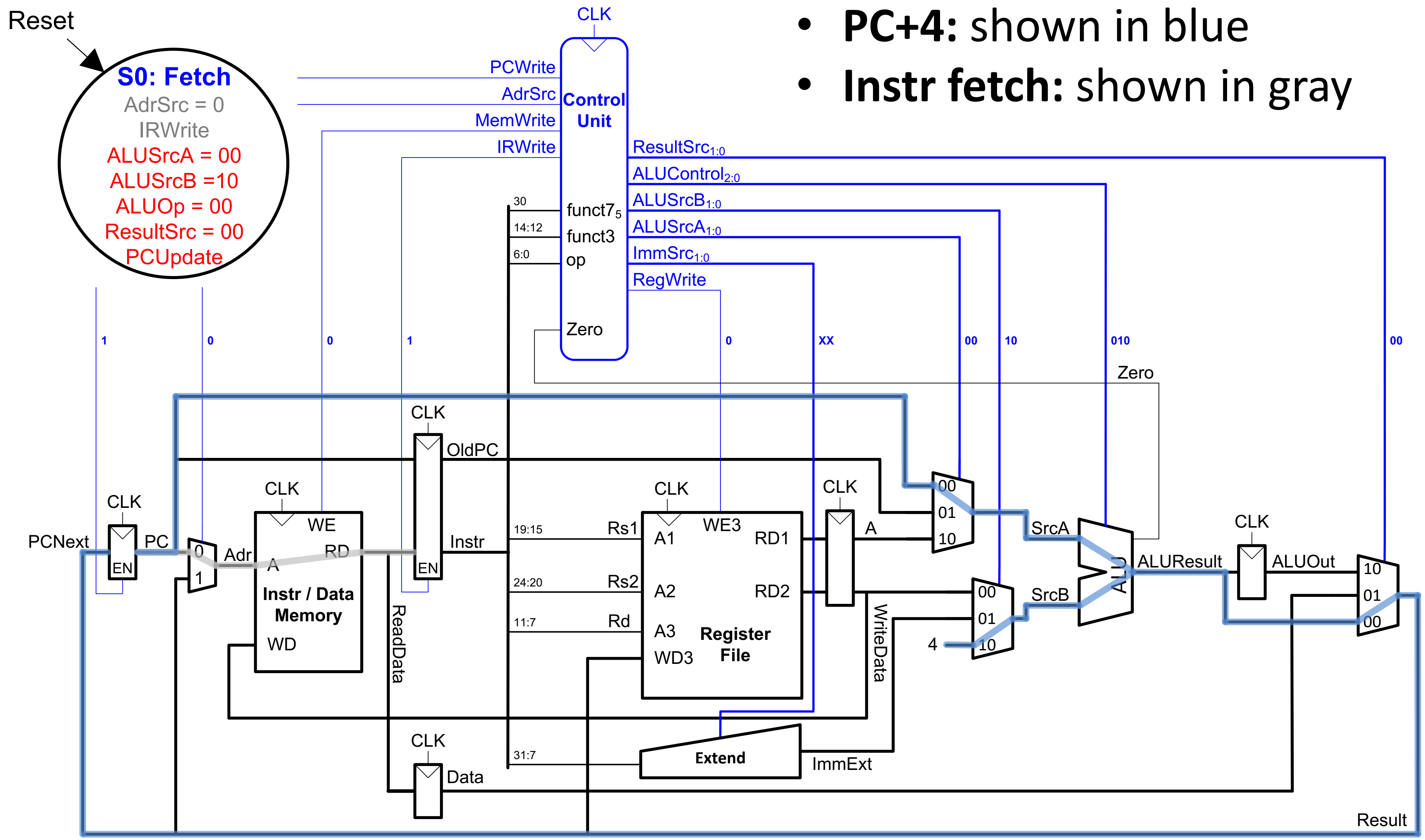
- **ALU** isn't being used
- Use ALU to calculate **PC+4**

Main Controller FSM: Fetch Revisited



Main Controller FSM: Fetch Revisited

- **PC+4:** shown in blue
- **Instr fetch:** shown in gray


Reset
S0: Fetch
AdrSrc = 0
IRWrite
ALUSrcA = 00
ALUSrcB =10
ALUOp = 00
ResultSrc = 00
PCUpdate
CLK
Control Unit
PCWrite
AdrSrc
MemWrite
IRWrite
ResultSrc1:0
ALUControl2:0
ALUSrcB1:0
ALUSrcA1:0
ImmSrc1:0
RegWrite
funct75
funct3
op
Zero
30
14:12
6:0
1
0
0
1
0
XX
00
10
010
00
Zero
CLK
OldPC
Instr
EN
PCNext
PC
EN
Adr
A
RD
WE
Instr / Data Memory
WD
ReadData
CLK
Data
19:15
24:20
11:7
31:7
Rs1
Rs2
Rd
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
Extend
ImmExt
A
WriteData
SrcA
SrcB
4
ALU
ALUResult
ALUOut
Result