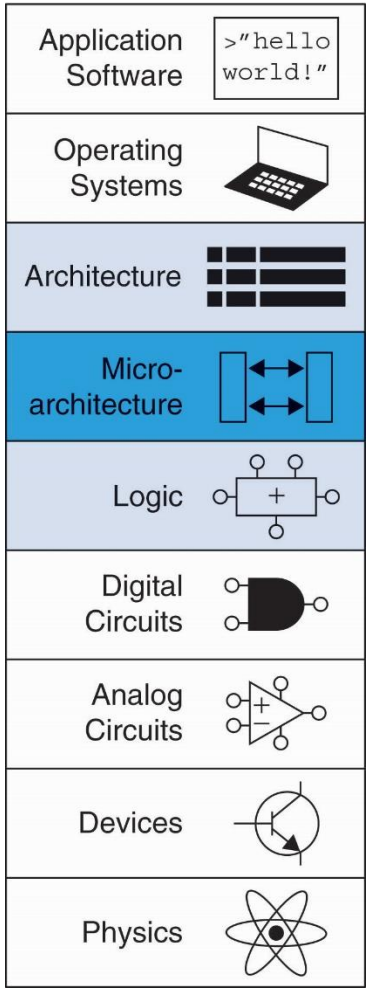
Digital Design & Computer Architecture

Sarah Harris & David Harris

Chapter 7: Microarchitecture

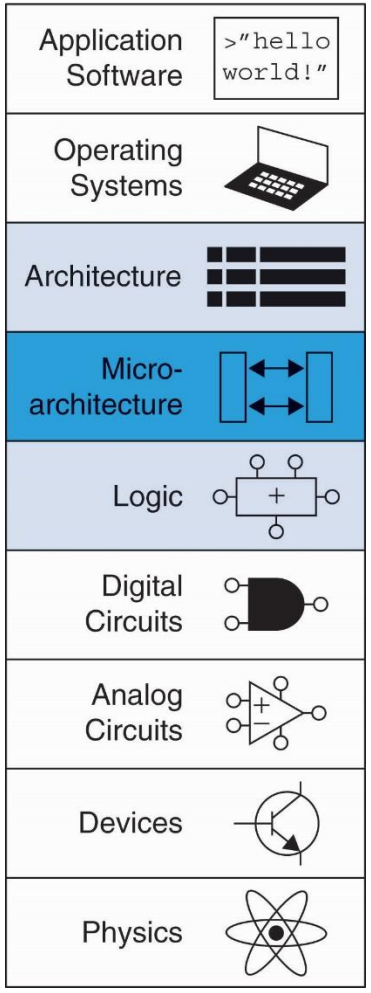
Chapter 7 :: Topics

- **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

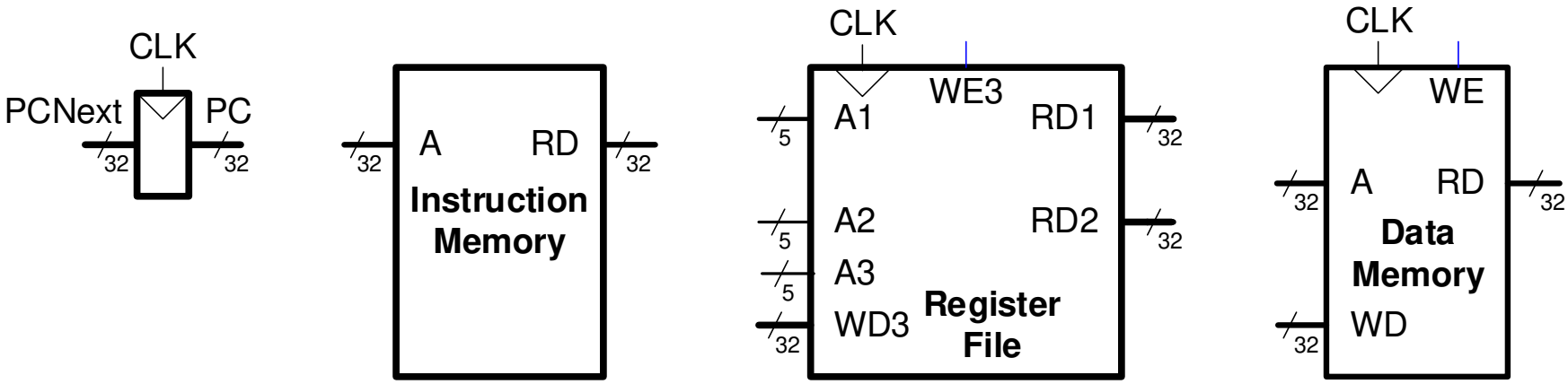
- Consider **subset** of RISC-V instructions:
  - **R-type ALU instructions:**
    - `add`, `sub`, `and`, `or`, `slt`
  - **Memory instructions:**
    - `lw`, `sw`
  - **Branch instructions:**
    - `beq`

Architectural State Elements

Determines everything about a processor:

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

RISC-V Architectural State Elements



Chapter 7: Microarchitecture

Single-Cycle RISC-V Processor

Single-Cycle RISC-V Processor

- Datapath
- Control

Example Program

- Design datapath
- View example program executing

Example Program:

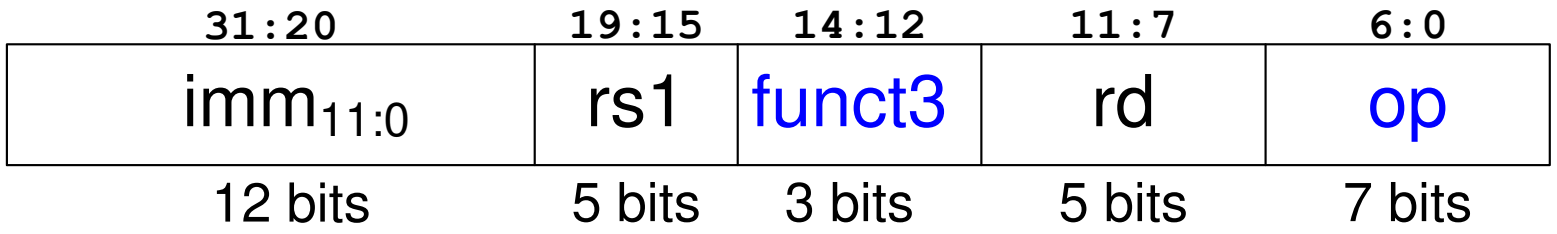
Address	Instruction	Type	Fields						Machine Language
			$\mathbf{imm_{11:0}}$		**rs1**	**f3**	**rd**	**op**	
0x1000	L7: lw x6, -4(x9)	I	111111111100		01001	010	00110	0000011	FFC4A303
			$\mathbf{imm_{11:5}}$	**rs2**	**rs1**	**f3**	$\mathbf{imm_{4:0}}$	**op**	
0x1004	sw x6, 8(x9)	S	0000000	00110	01001	010	01000	0100011	0064A423
			funct7	**rs2**	**rs1**	**f3**	**rd**	**op**	
0x1008	or x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233
			$\mathbf{imm_{12,10:5}}$	**rs2**	**rs1**	**f3**	$\mathbf{imm_{4:1,11}}$	**op**	
0x100C	beq x4, x4, L7	B	1111111	00100	00100	000	10101	1100011	FE420AE3

Single-Cycle RISC-V Processor

- **Datapath:** start with `lw` instruction
- **Example:** `lw x6, -4(x9)`

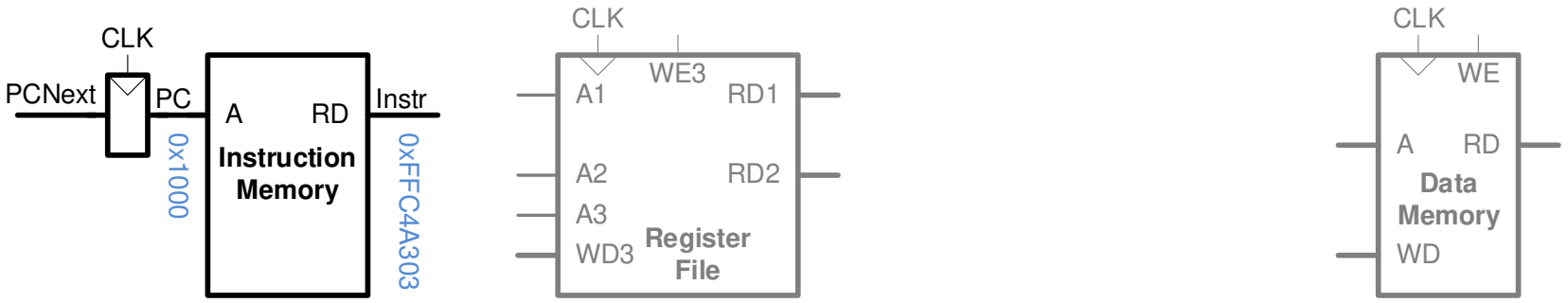
  `lw rd, imm(rs1)`

I-Type



Single-Cycle Datapath: `lw` fetch

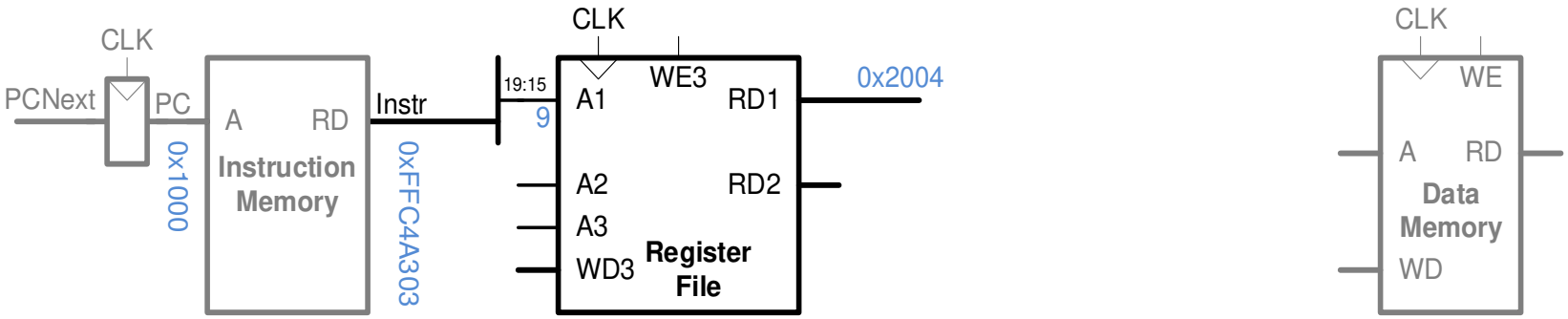
STEP 1: Fetch instruction



Address	Instruction	Type	Fields				Machine Language
			$\mathbf{imm_{11:0}}$	**rs1**	**f3**	**rd**	**op**	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Single-Cycle Datapath: `lw` Reg Read

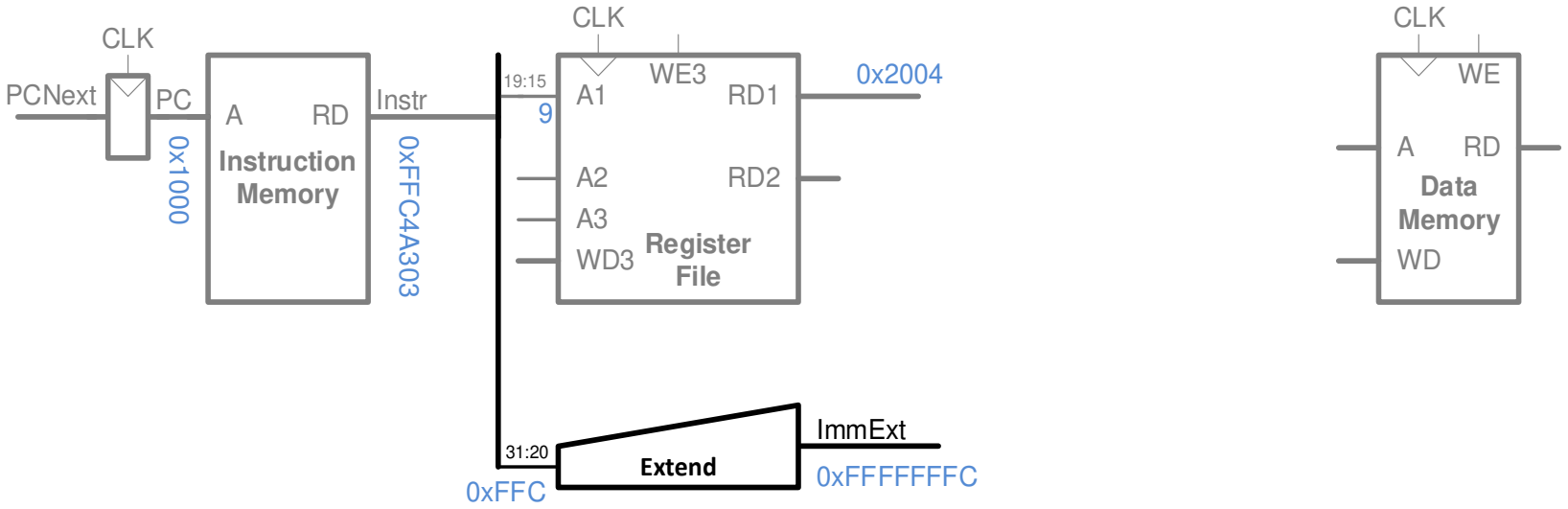
STEP 2: Read source operand (**rs1**) from RF



Address	Instruction	Type	Fields				Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Single-Cycle Datapath: `lw` Immediate

STEP 3: Extend the immediate

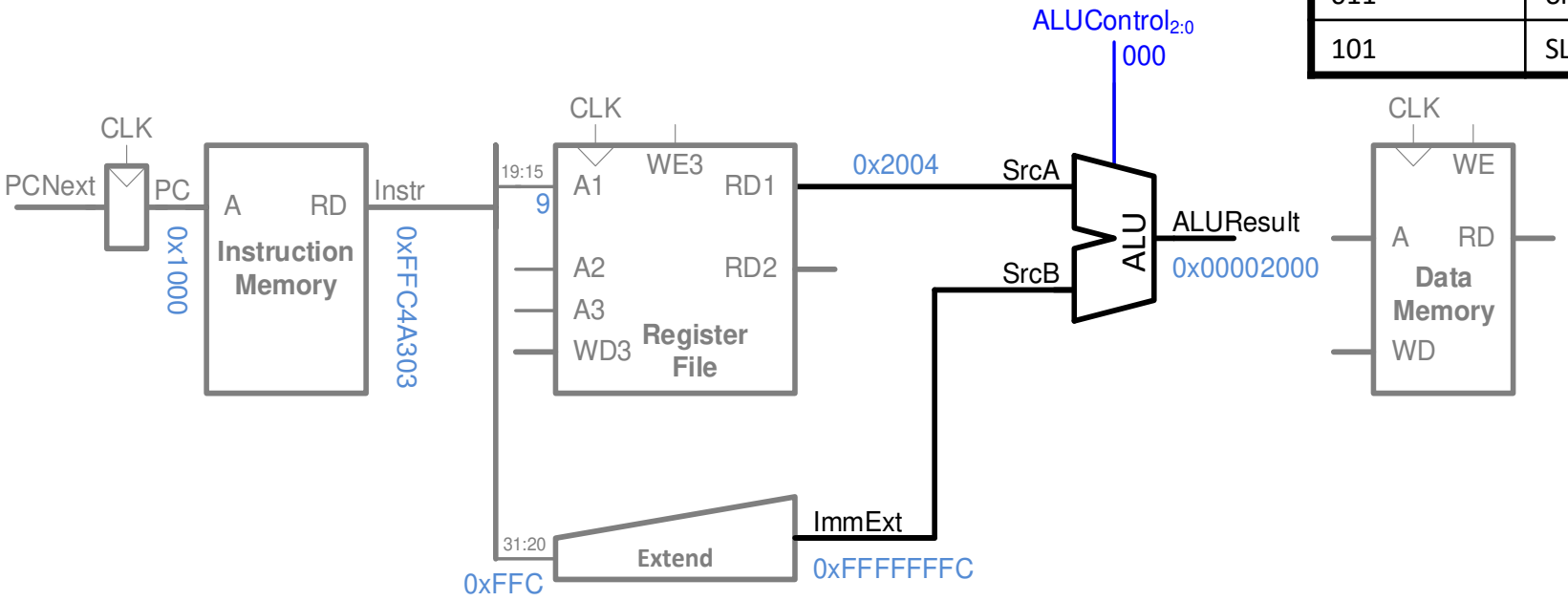


Address	Instruction	Type	Fields					Machine Language
			$\mathbf{imm_{11:0}}$	**rs1**	**f3**	**rd**	**op**	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Single-Cycle Datapath: `lw` Address

STEP 4: Compute the memory address

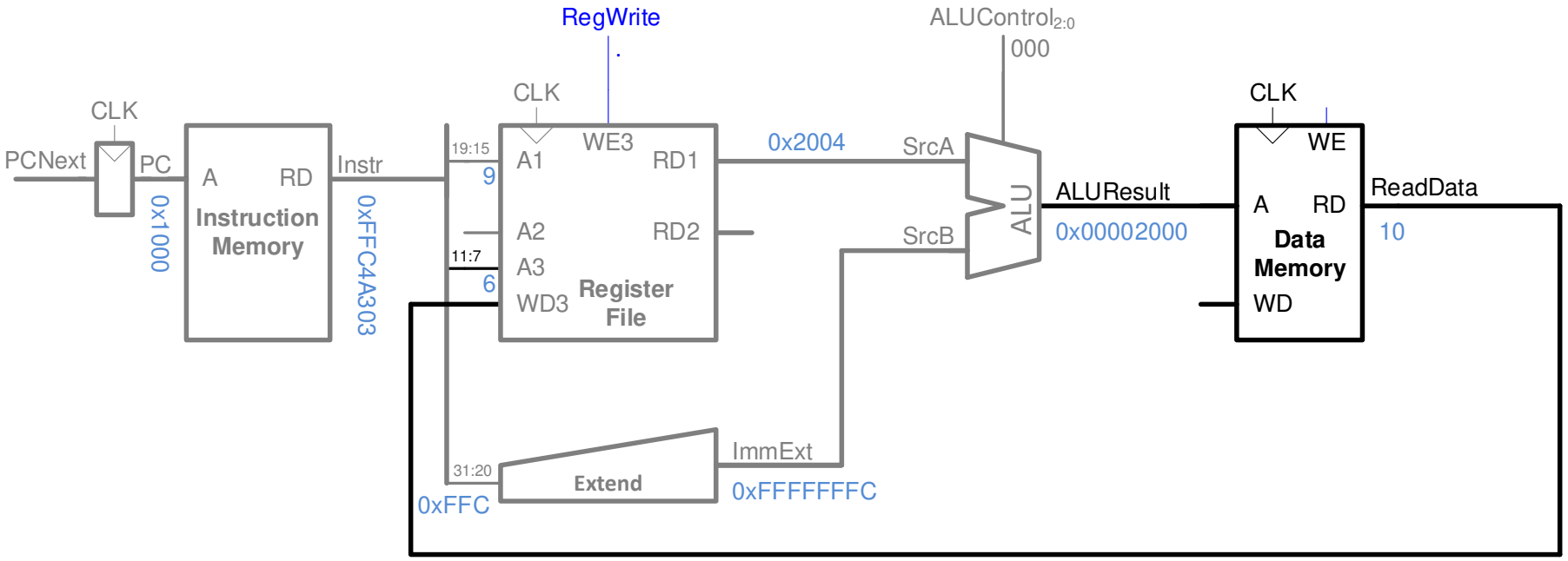
$ALUControl_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT



Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Single-Cycle Datapath: `lw` Mem Read

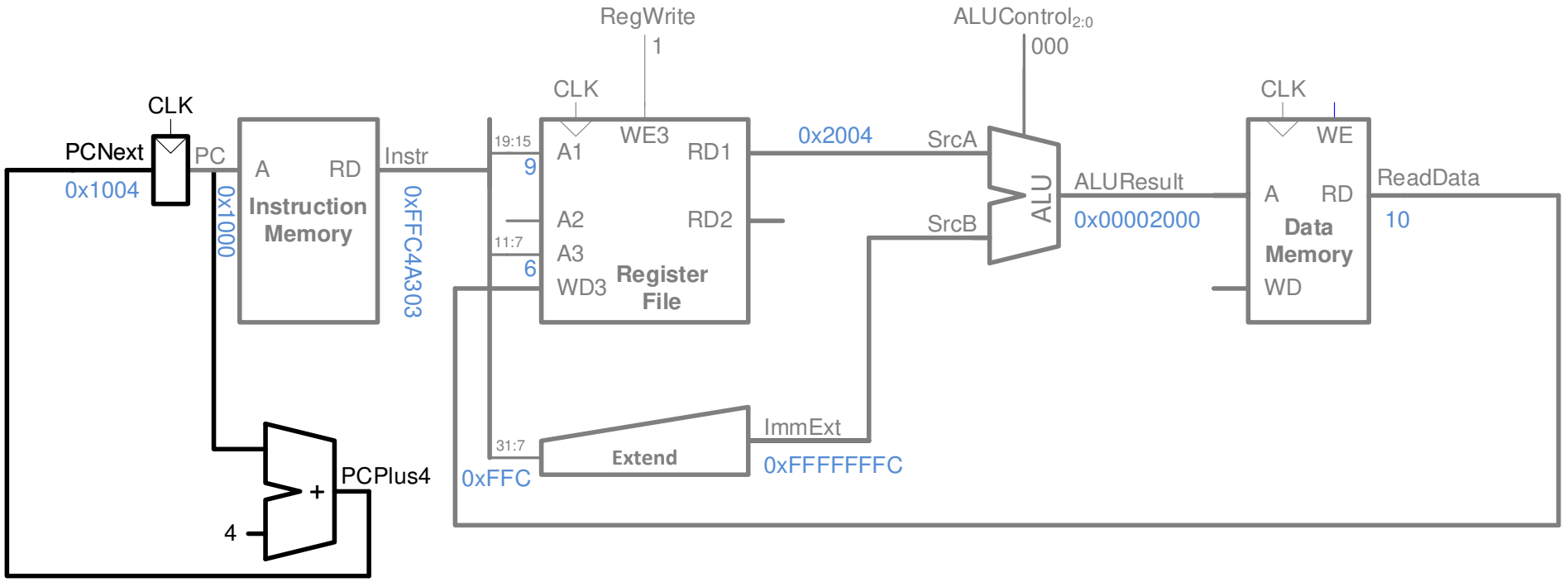
STEP 5: Read data from memory and write it back to register file



Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Single-Cycle Datapath: PC Increment

STEP 6: Determine address of next instruction



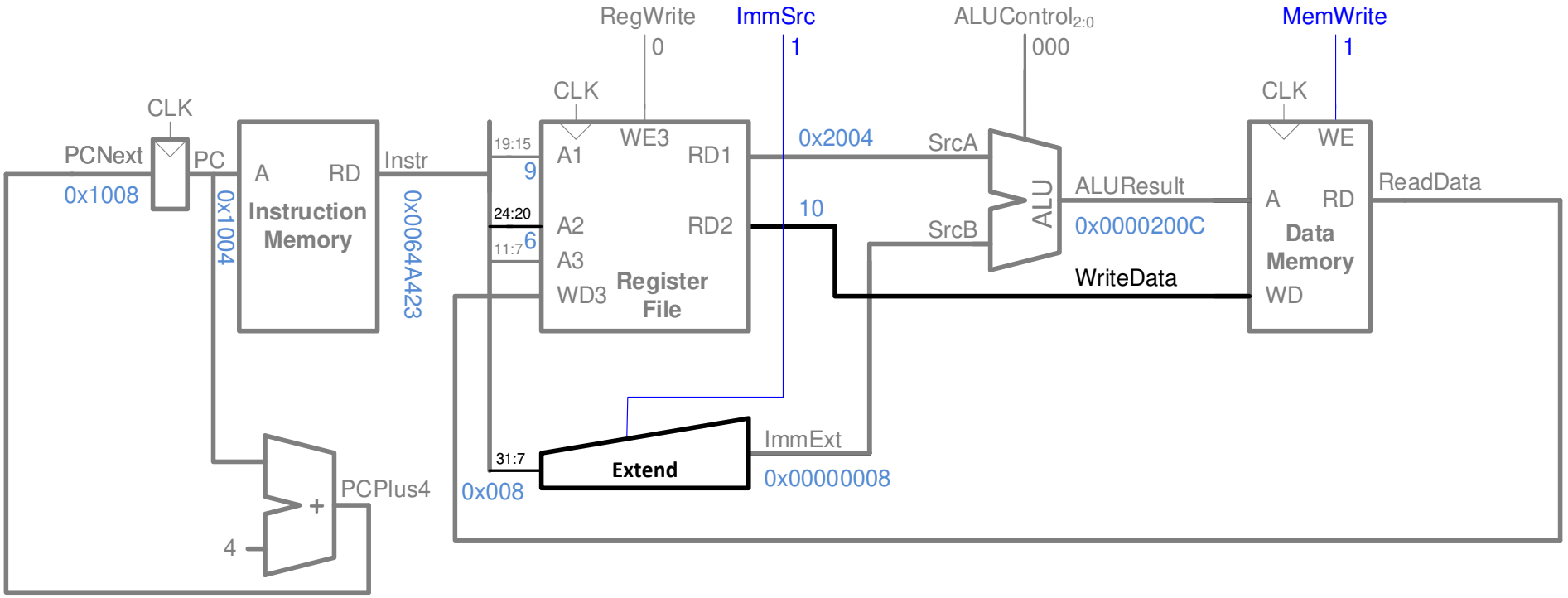
Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Chapter 7: Microarchitecture

Single-Cycle Datapath: Other Instructions

Single-Cycle Datapath: sw

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

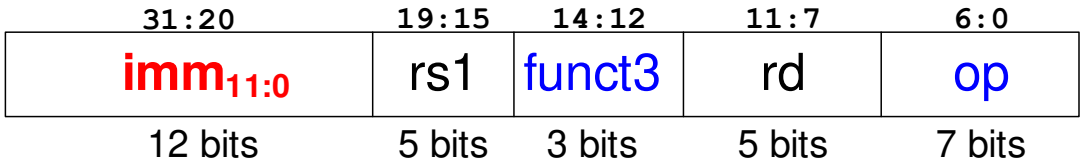


Address	Instruction	Type	Fields						Machine Language
			$imm_{11:5}$	rs2	rs1	f3	$imm_{4:0}$	op	
0x1004	sw x6, 8(x9)	S	0000000	00110	01001	010	01000	0100011	0064A423

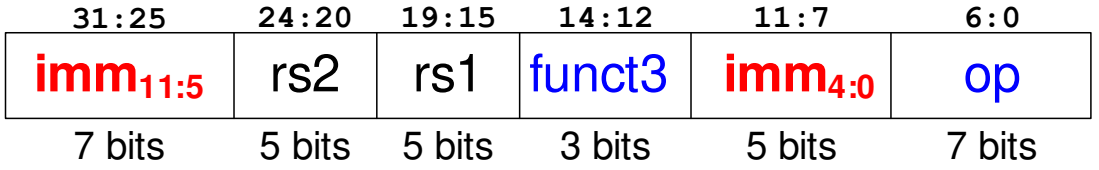
Single-Cycle Datapath: Immediate

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

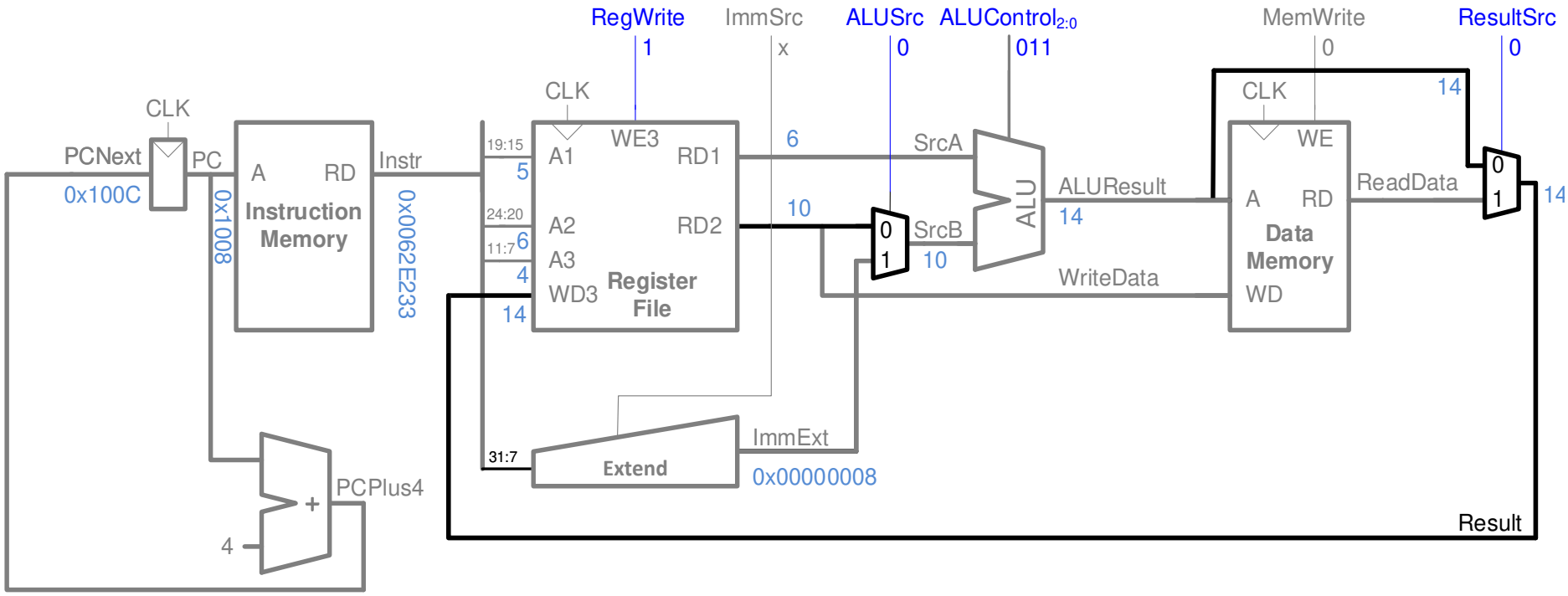


S-Type



Single-Cycle Datapath: R-type

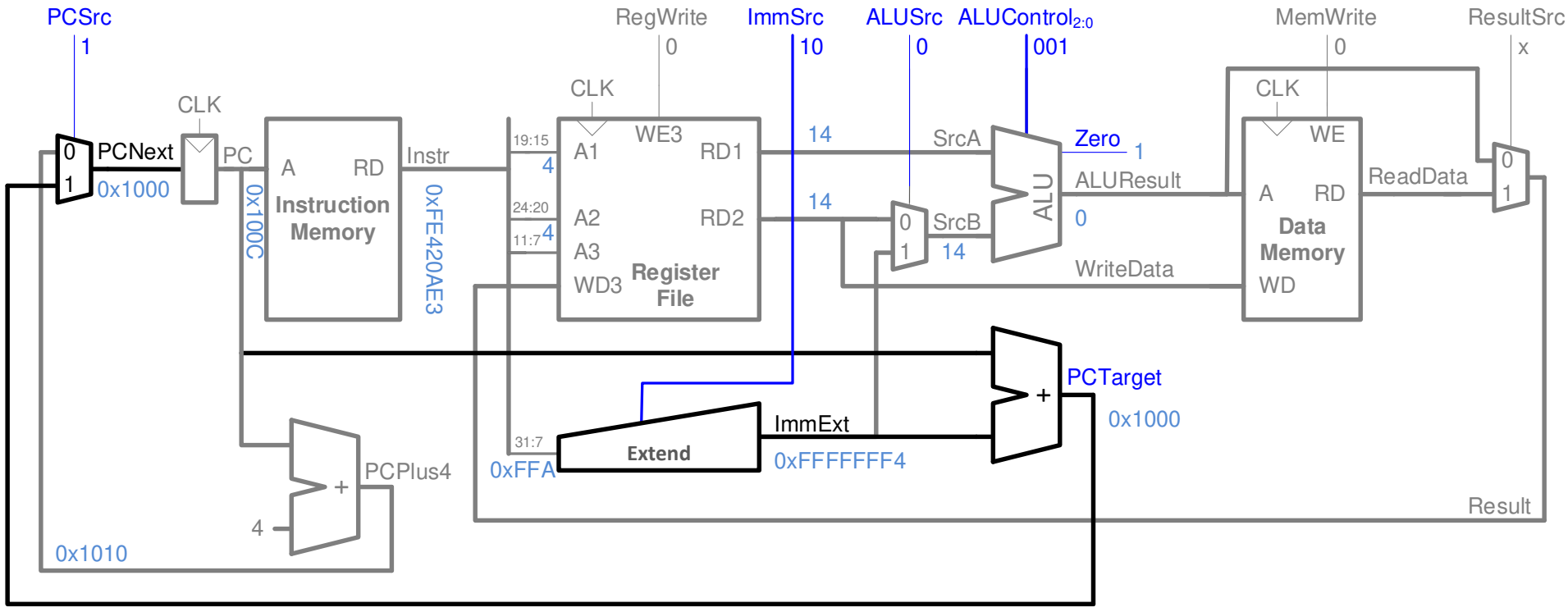
- Read from **rs1** and **rs2** (instead of **imm**)
- Write *ALUResult* to **rd**



Address	Instruction	Type	Fields					Machine Language
			funct7	rs2	rs1	f3	rd	op	
0x1008	or x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233

Single-Cycle Datapath: beq

Calculate **target address:** PCTarget = PC + imm

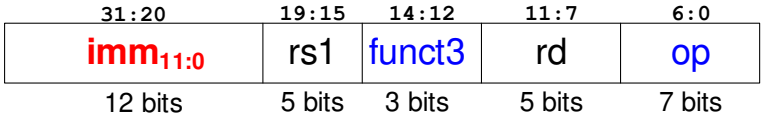


Address	Instruction	Type	Fields						Machine Language
			$imm_{12,10:5}$	rs2	rs1	f3	$imm_{4:1,11}$	op	
0x100C	beq x4, x4, L7	B	1111111	00100	00100	000	10101	1100011	FE420AE3

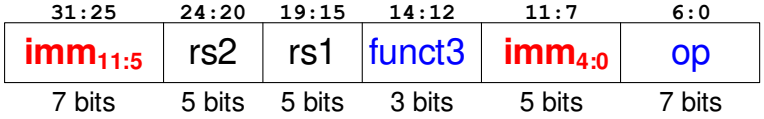
Single-Cycle Datapath: ImmExt

$ImmSrc_{1:0}$	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

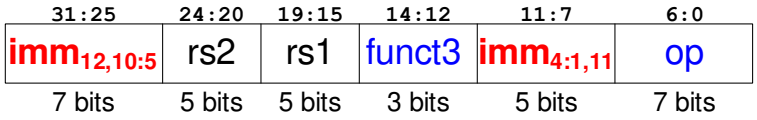
I-Type



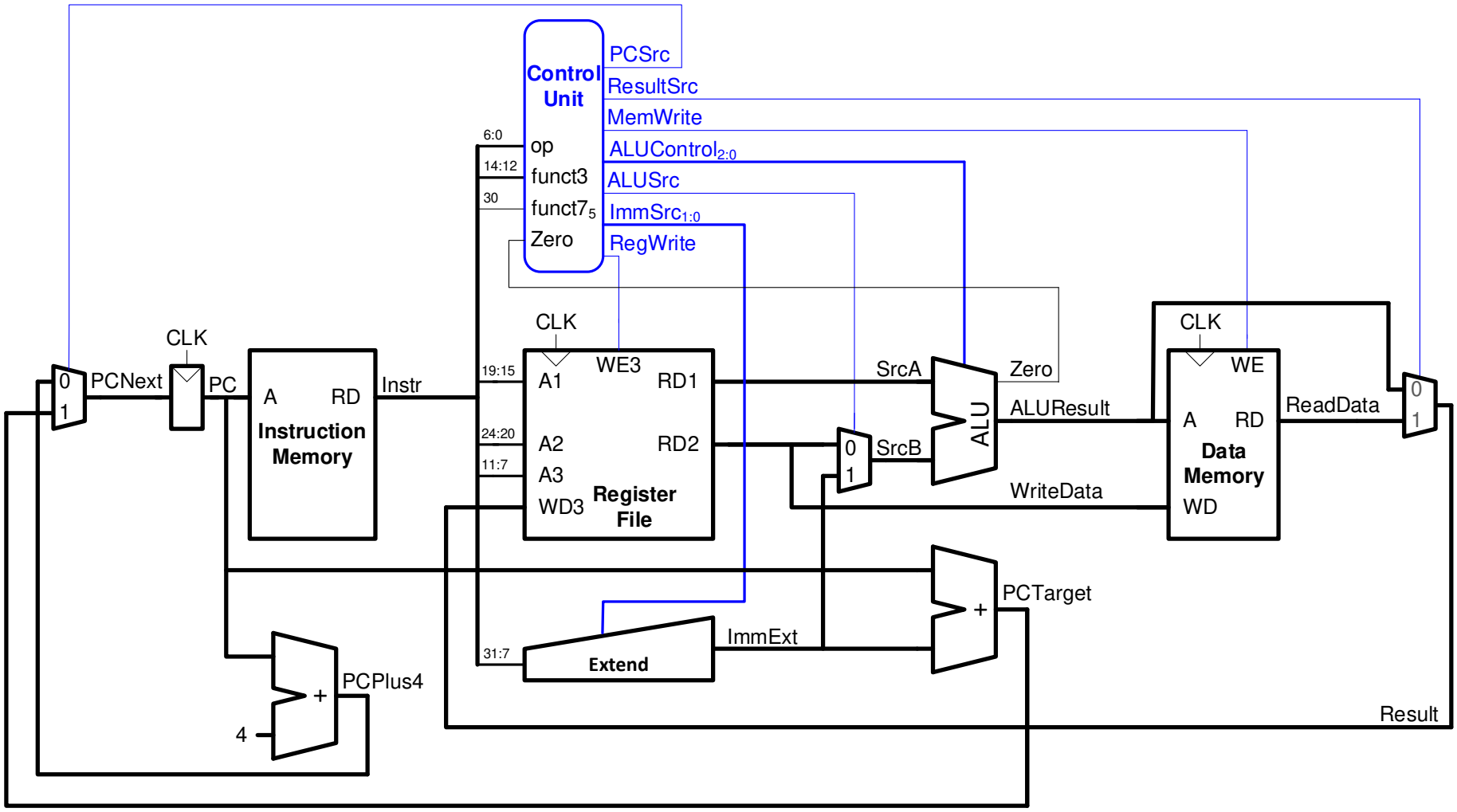
S-Type



B-Type



Single-Cycle RISC-V Processor

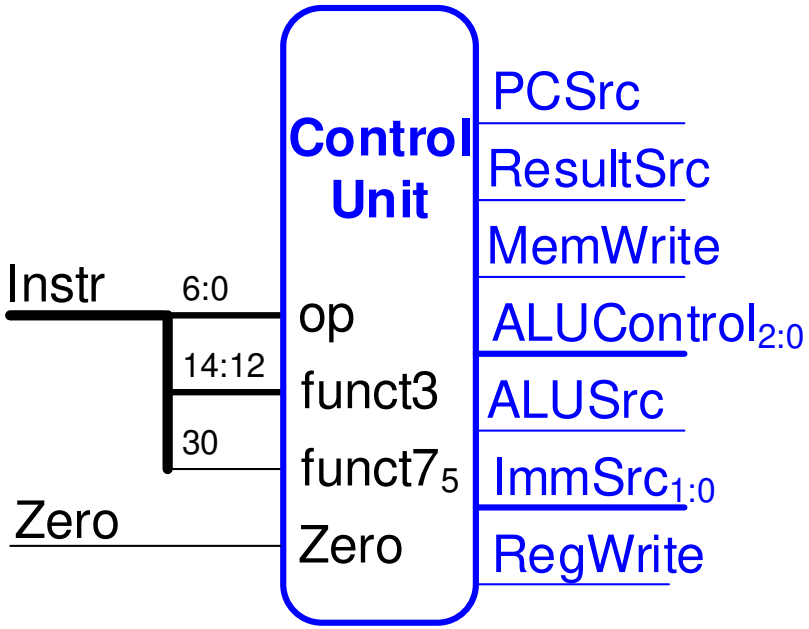


Chapter 7: Microarchitecture

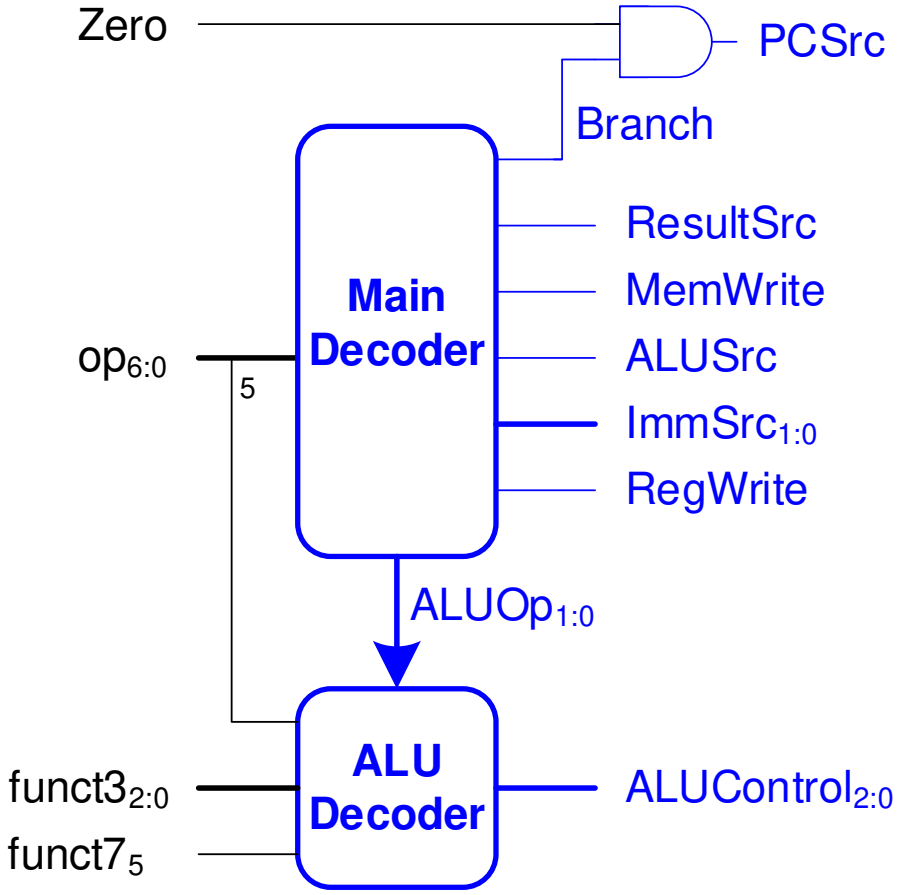
Single-Cycle Control

Single-Cycle Control

High-Level View

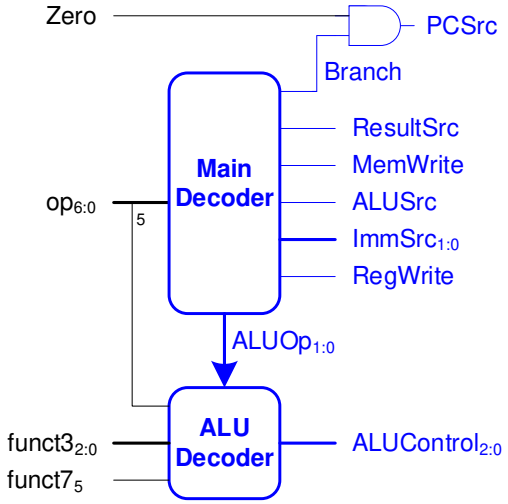


Low-Level View



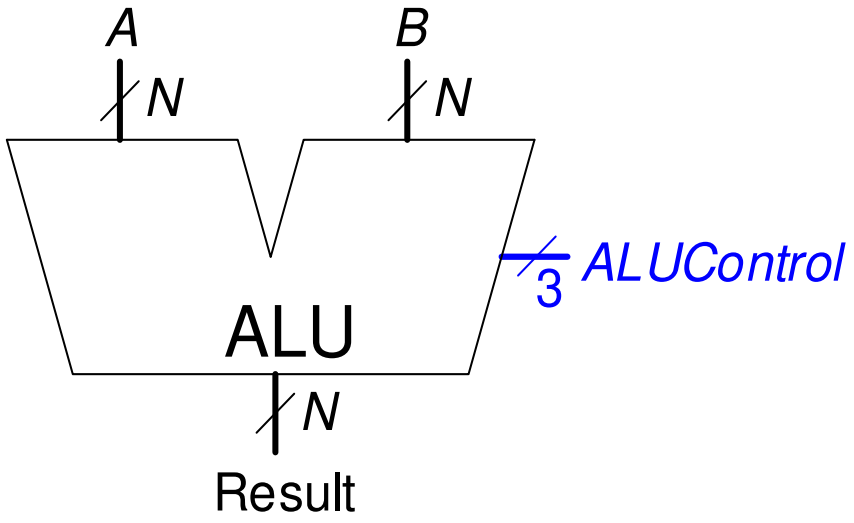
Single-Cycle Control: Main Decoder

op	Instr.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	`lw`							
35	`sw`							
51	R-type							
99	`beq`							



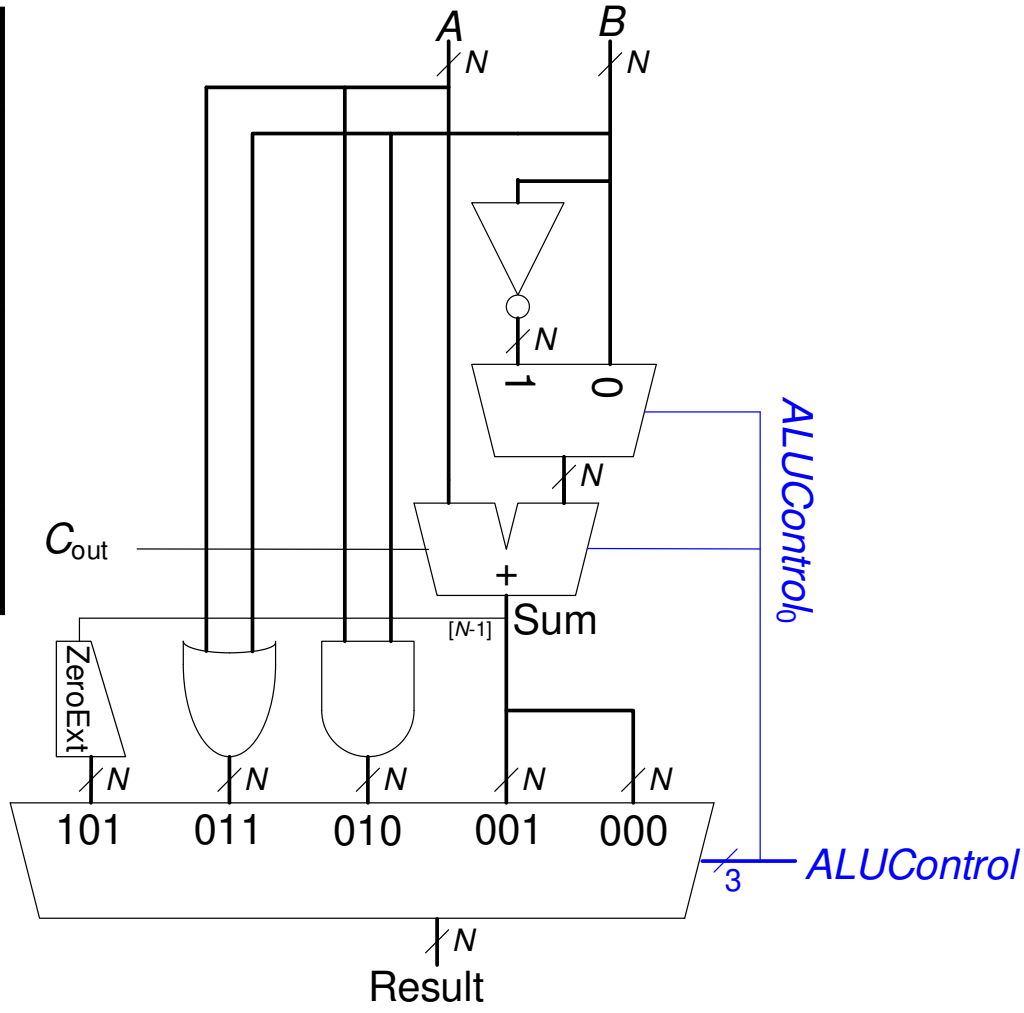
Review: ALU

$ALUControl_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT

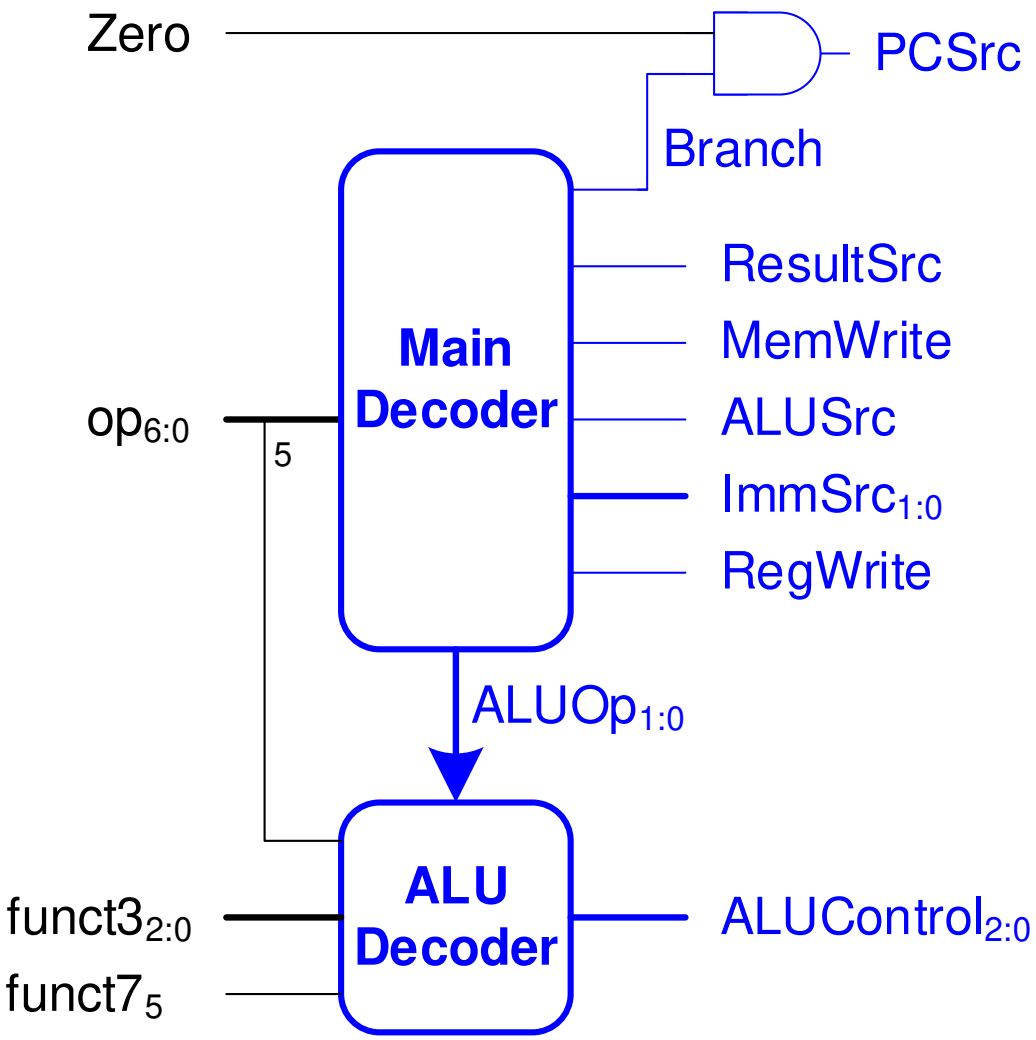


Review: ALU

$ALUControl_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT

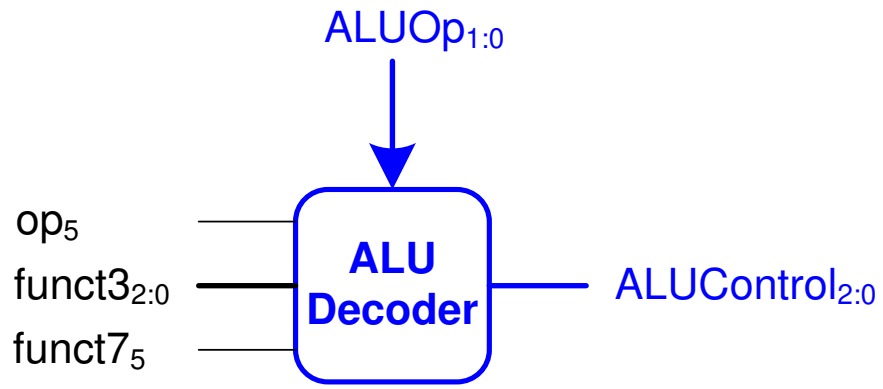


Single-Cycle Control: ALU Decoder



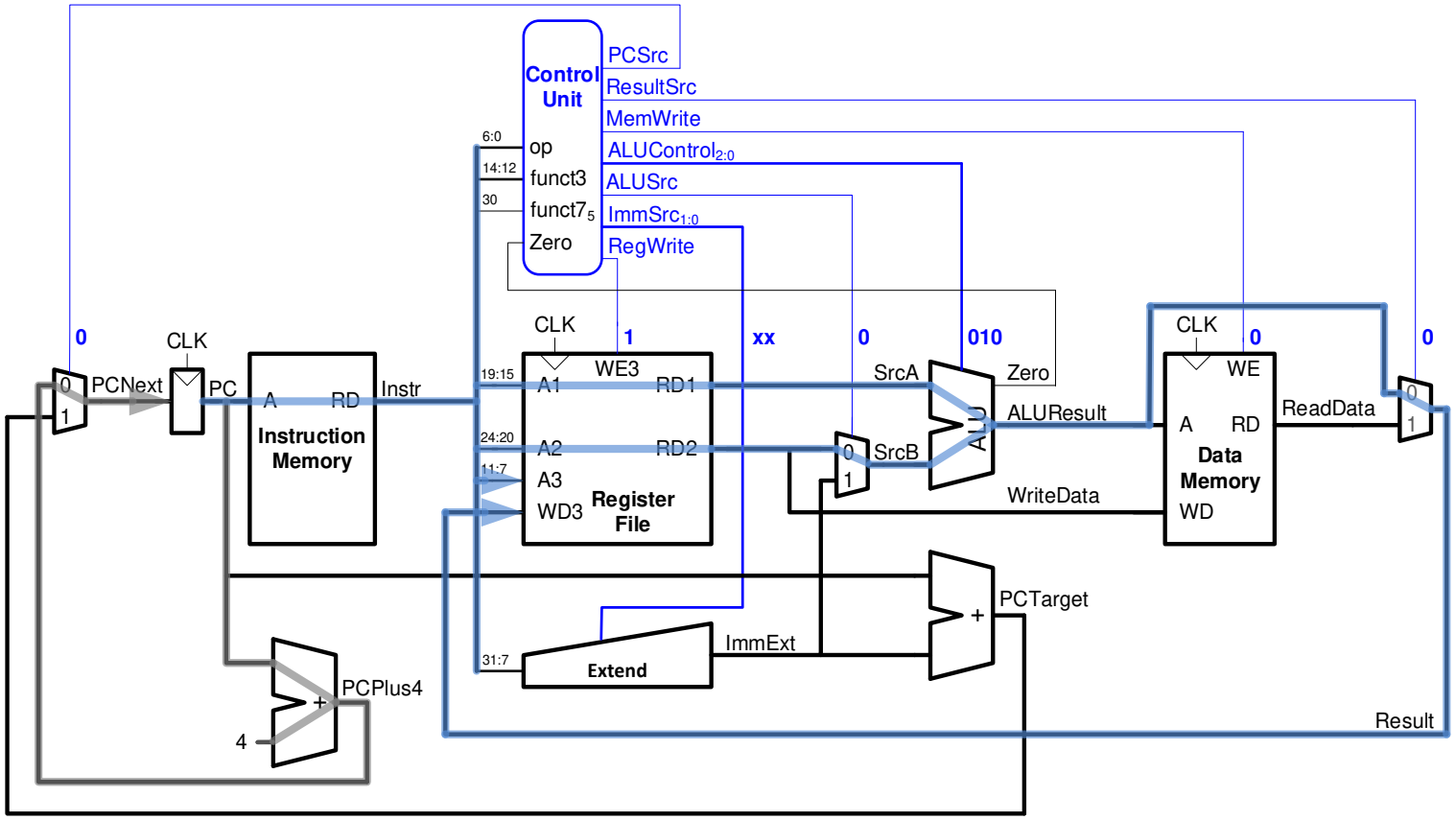
Single-Cycle Control: ALU Decoder

ALUOp	funct3	op_5, $funct7_5$	Instruction	$ALUControl_{2:0}$
00	x	x	`lw, sw`	000 (add)
01	x	x	`beq`	001 (subtract)
10	000	00, 01, 10	`add`	000 (add)
	000	11	`sub`	001 (subtract)
	010	x	`slt`	101 (set less than)
	110	x	`or`	011 (or)
	111	x	`Slt`	010 (and)



Example: and

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



and x5, x6, x7

Chapter 7: Microarchitecture

Extending the Single-Cycle Processor

Extended Functionality: I-Type ALU

Enhance the single-cycle processor to handle **I-Type ALU instructions**: `addi`, `andi`, `ori`, and `slti`

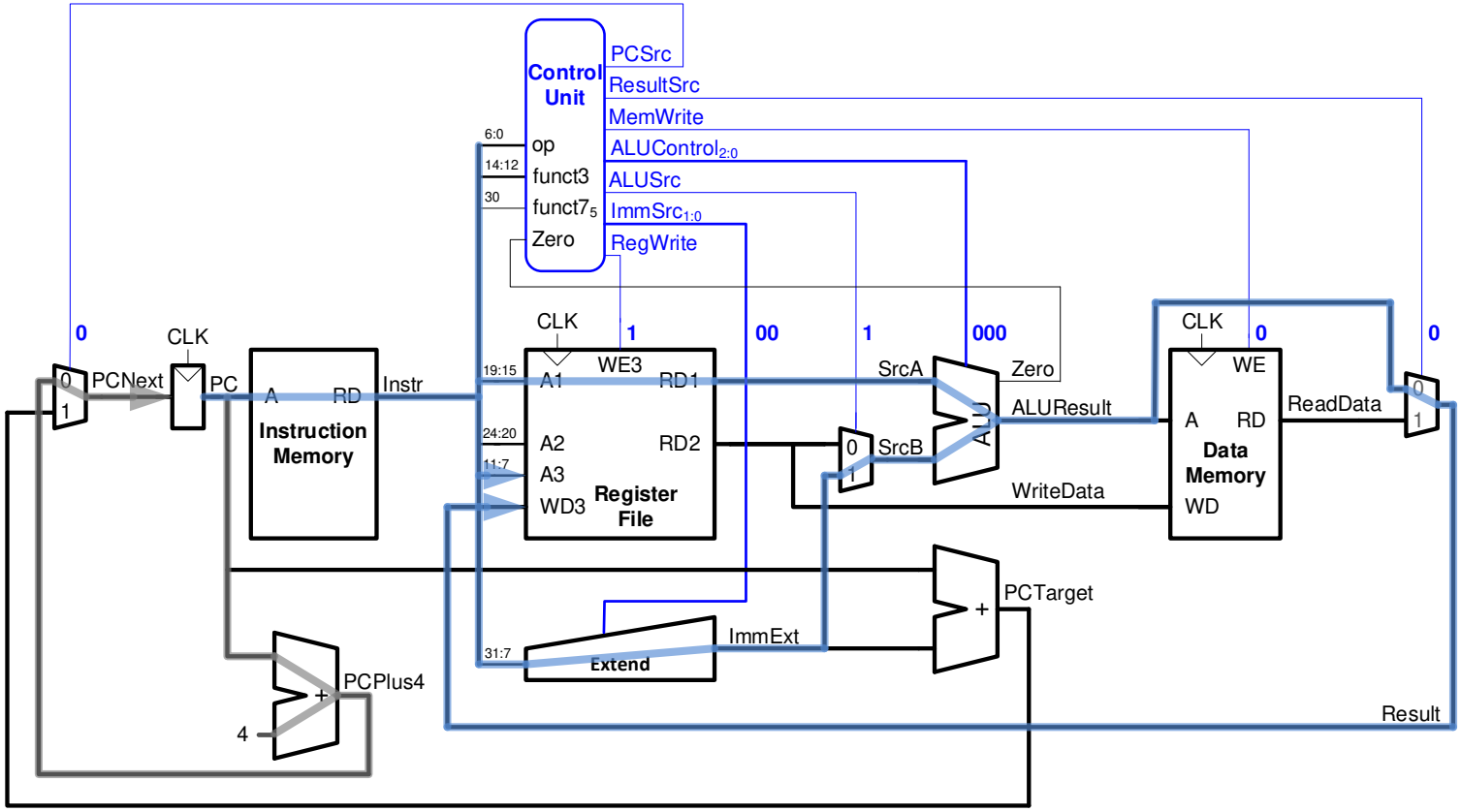
- **Similar to R-type** instructions
- But **second source** comes from **immediate**
- Change ***ALUSrc*** to select the immediate
- And ***ImmSrc*** to pick the correct immediate

Extended Functionality: I-Type ALU

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	**I-type**	**1**	**00**	**1**	**0**	**0**	**0**	**10**

Extended Functionality: addi

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
19	I-type	1	00	1	0	0	0	10



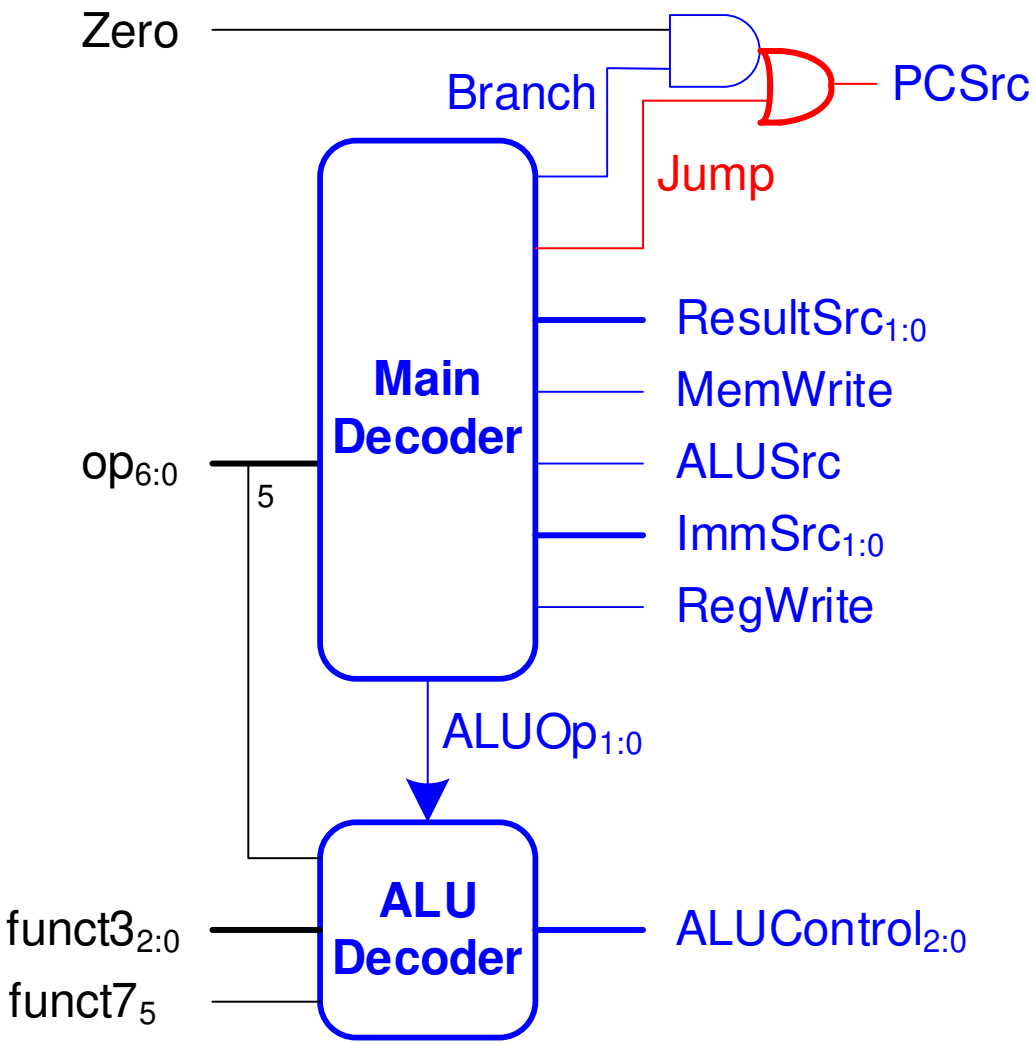
addi x5, x6, -33

Extended Functionality: `jal`

Enhance the single-cycle processor to handle `jal`

- **Similar to `beq`**
- But jump is **always taken**
  - *PCSrc* should be 1
- **Immediate format** is different
  - Need a new *ImmSrc* of 11
- And `jal` must **compute PC+4** and **store in `rd`**
  - Take PC+4 from adder through ResultMux

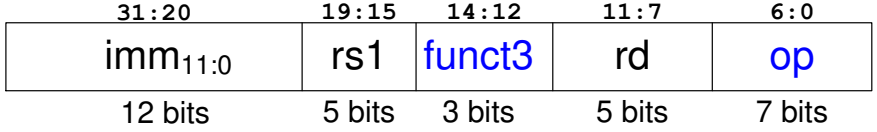
Extended Functionality: jal



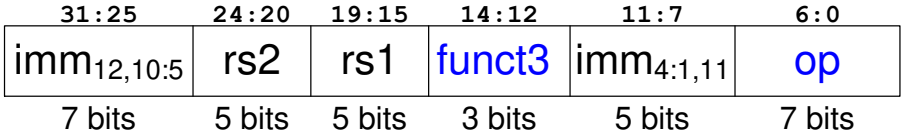
Extended Functionality: *ImmExt*

$ImmSrc_{1:0}$	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**

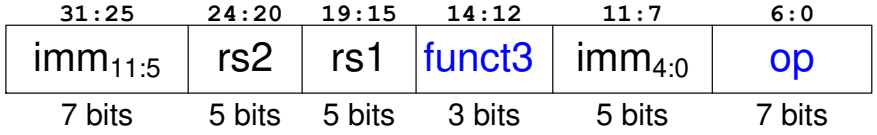
I-Type



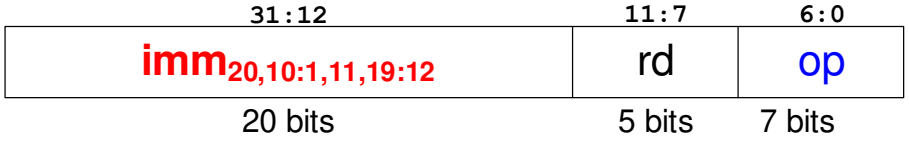
B-Type



S-Type



J-Type

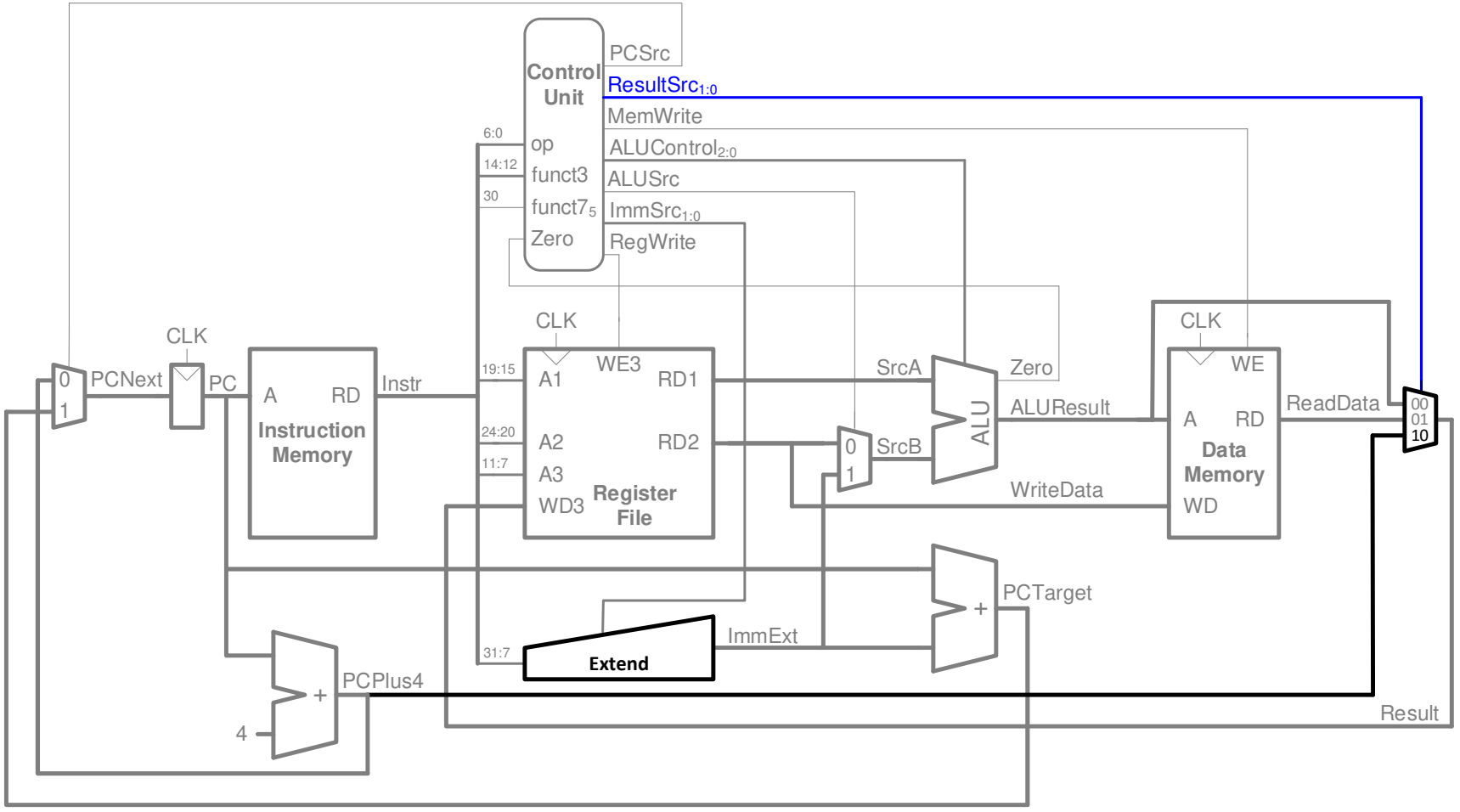


Extended Functionality: jal

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
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	I-type	1	00	1	0	01	0	10	0
111	jal	1	11	X	0	10	0	XX	1

Extended Functionality: `jal`

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
111	jal	1	11	X	0	10	0	XX	1



Chapter 7: Microarchitecture

Single-Cycle Performance

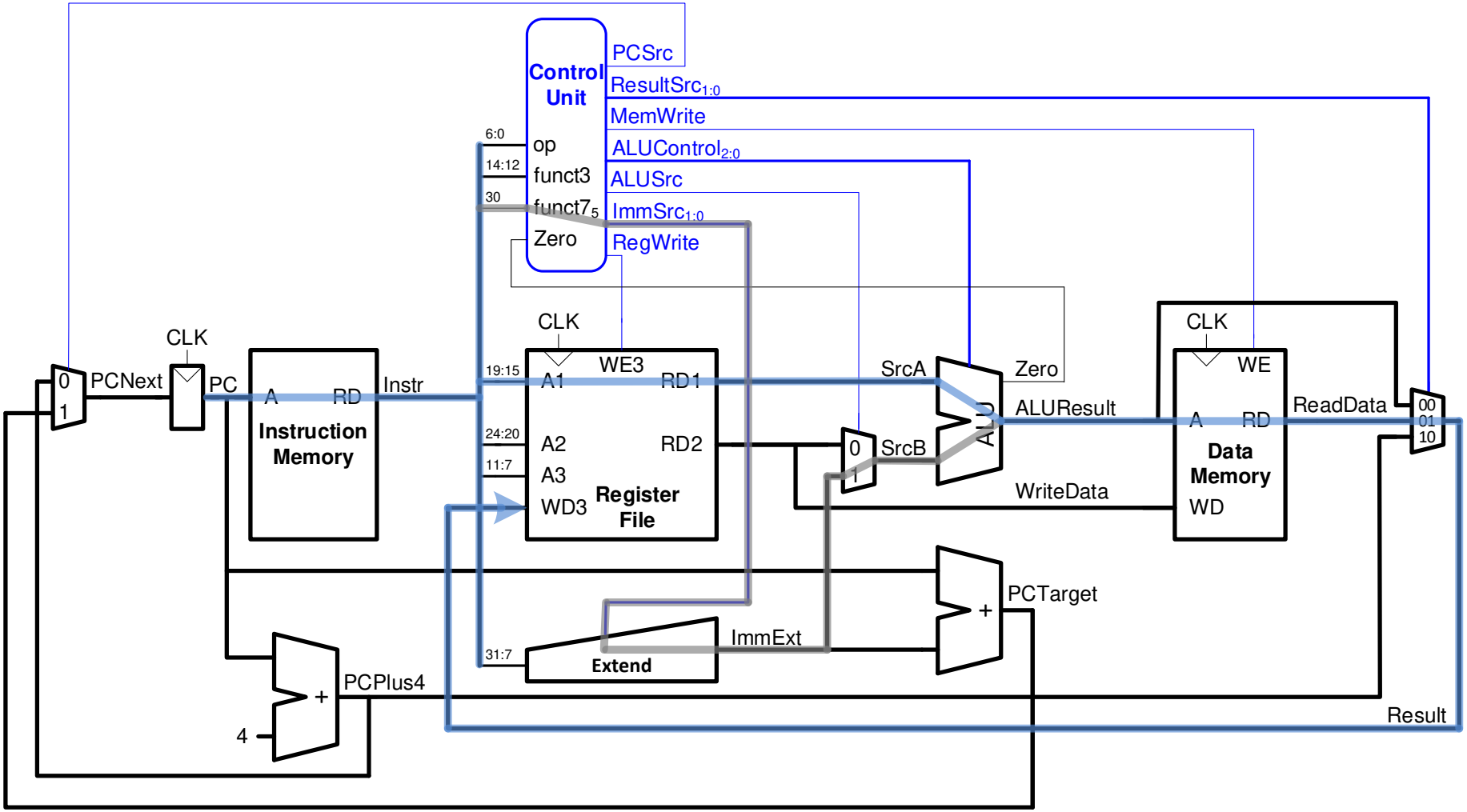
Processor Performance

Program Execution Time

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

= # instructions x CPI x T_C

Single-Cycle Processor Performance



T_c limited by critical path (`lw`)

Single-Cycle Processor Performance

- **Single-cycle critical path:**

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

- **Typically, limiting paths are:**
  - memory, ALU, register file
  - *So,*

$$T_{c\_single} = t_{pcq\_PC} + t_{\text{mem}} + t_{RF\text{read}} + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$$
$$= t_{pcq\_PC} + 2t_{\text{mem}} + t_{RF\text{read}} + t_{\text{ALU}} + t_{\text{mux}} + t_{RF\text{setup}}$$

Single-Cycle Performance Example

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

$$T_{c\_single} = t_{pcq\_PC} + 2t_{\text{mem}} + t_{RF\text{read}} + t_{\text{ALU}} + t_{\text{mux}} + t_{RF\text{setup}}$$

$$=$$

Single-Cycle Performance Example

Program with 100 billion instructions:

Execution Time = # instructions x CPI x T_C