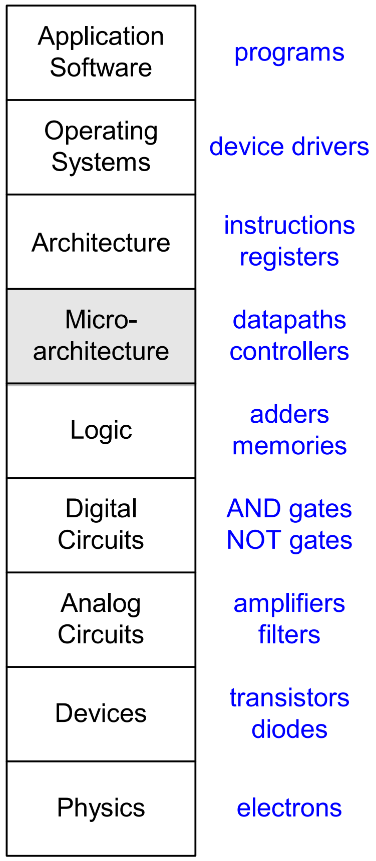
Microarchitecture
(Single Cycle Processor)

Acknowledgment: Slides are adapted from Harris and Harris textbook instructor's material

Introduction

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



Microarchitecture

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

RISC-V Processor

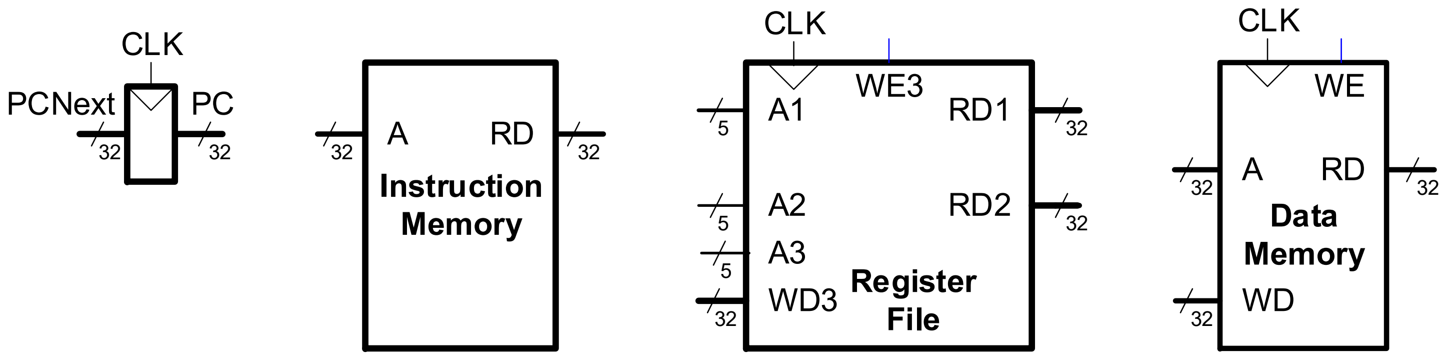
- Consider **subset** of RISC-V instructions:
  - **R-type ALU instructions:**
    - **`add`, `sub`, `and`, `or`, `slt`**
  - **Memory instructions:**
    - **`lw`, `sw`**
  - **Branch instructions:**
    - **`beq`**

Architectural State

- Defines the state of the program during execution
- Determines everything about a processor
- RISC-V elements include
  - PC
  - 32 registers
  - Memory
  - + specific control registers / flags

- Core dump file records architectural state at certain point in time

RISC-V Architectural State Elements



- Asynchronous read / synchronous write for memories
- Multi-ported register file
- Separate memory for instructions and data for now (will discuss how it fits store-program concept later)

Single-Cycle RISC-V Processor

- Datapath: Functional units where data are stored and processed

- Control: What generates control inputs to functional units

Example Program

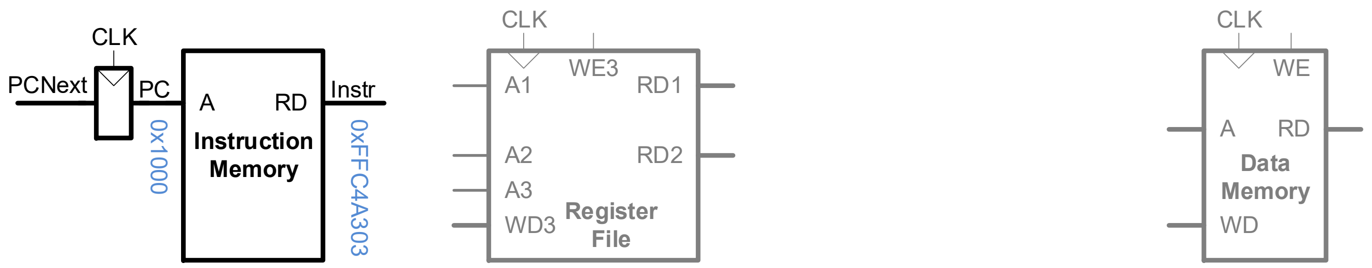
- Design datapath
- View example program executing

Example Program:

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

Single-Cycle Datapath: lw fetch

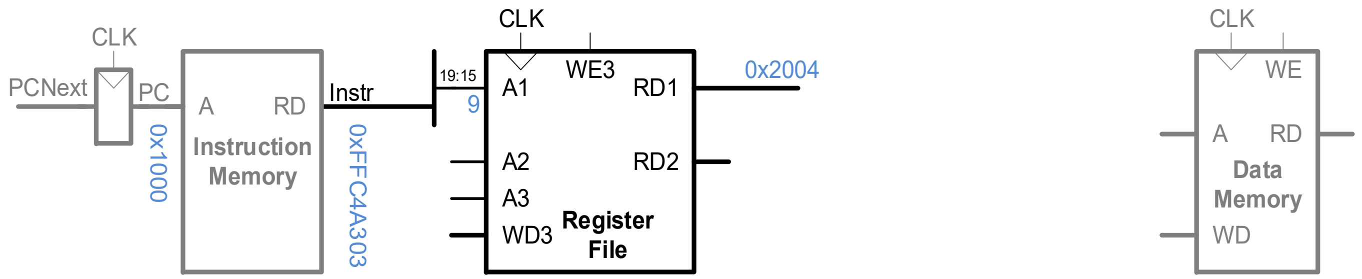
STEP 1: Fetch instruction



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

Single-Cycle Datapath: lw Reg Read

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

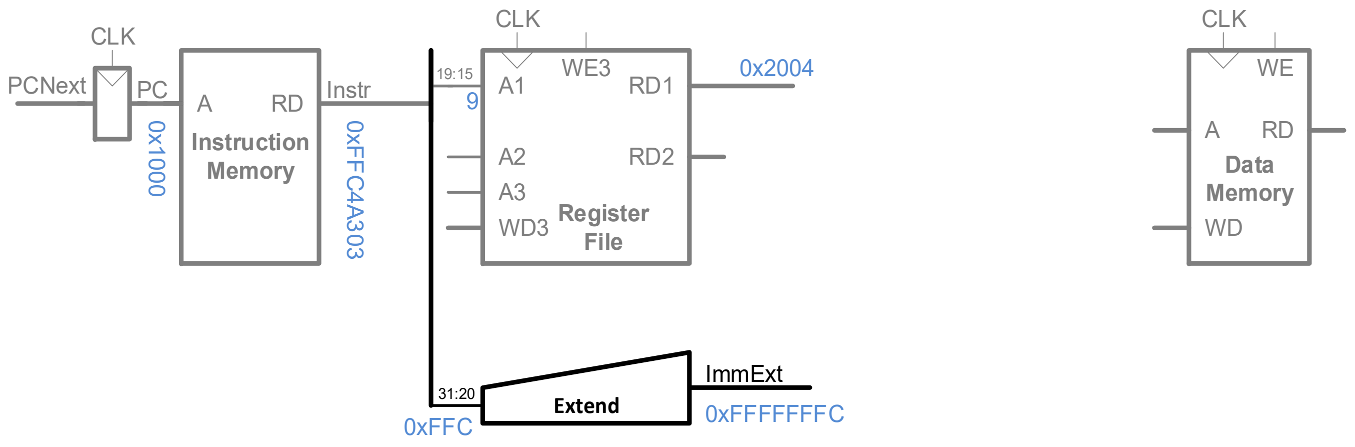


Address	Instruction	Type	Fields					Machine Language
			$\text{imm}_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Single-Cycle Datapath: lw immediate

STEP 3: Extend the immediate

"Steps" are not necessarily sequential. Step 3 is concurrent with Step 2

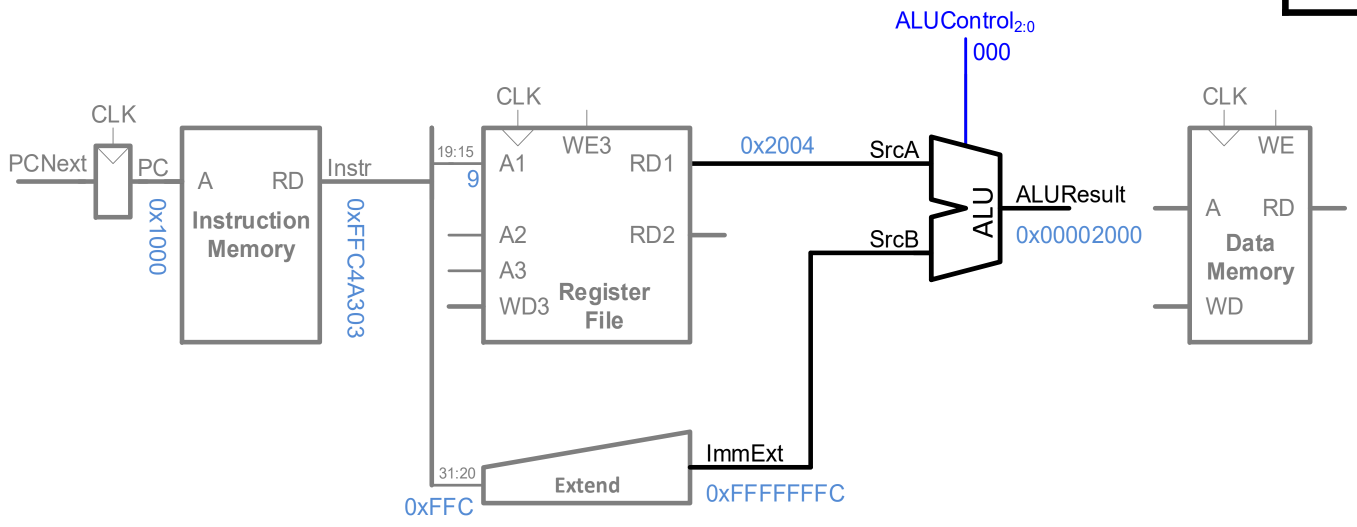


Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Single-Cycle Datapath: lw Address

STEP 4: Compute the memory address

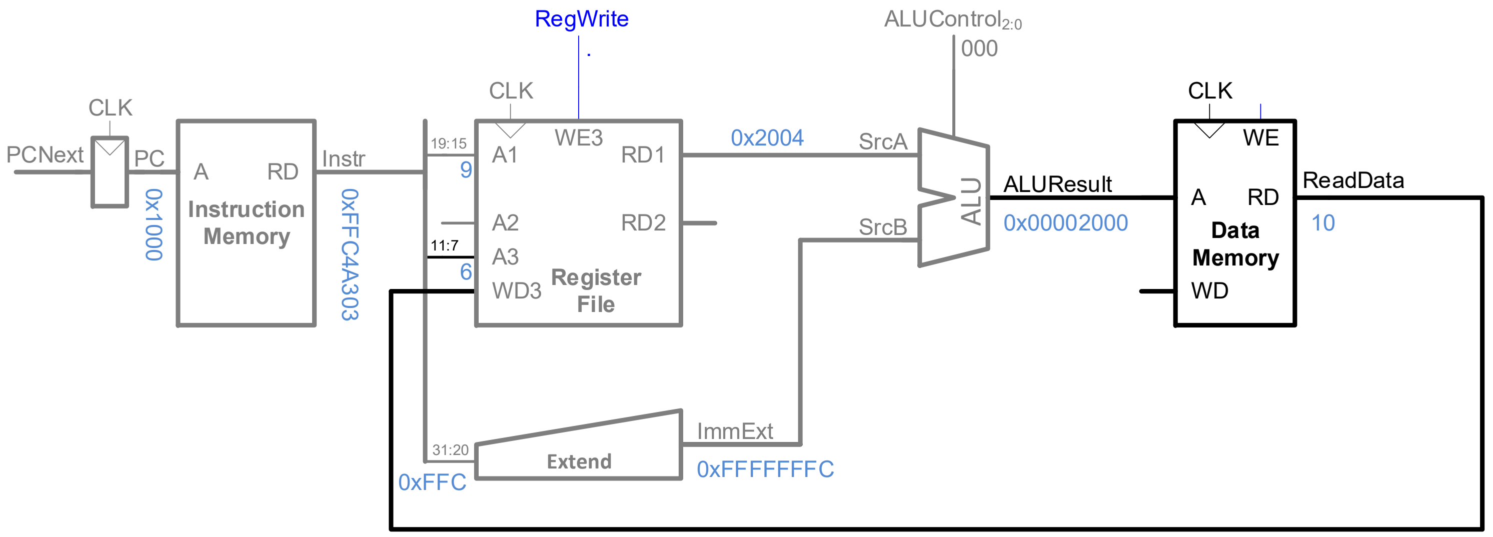
ALUControl$_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT



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

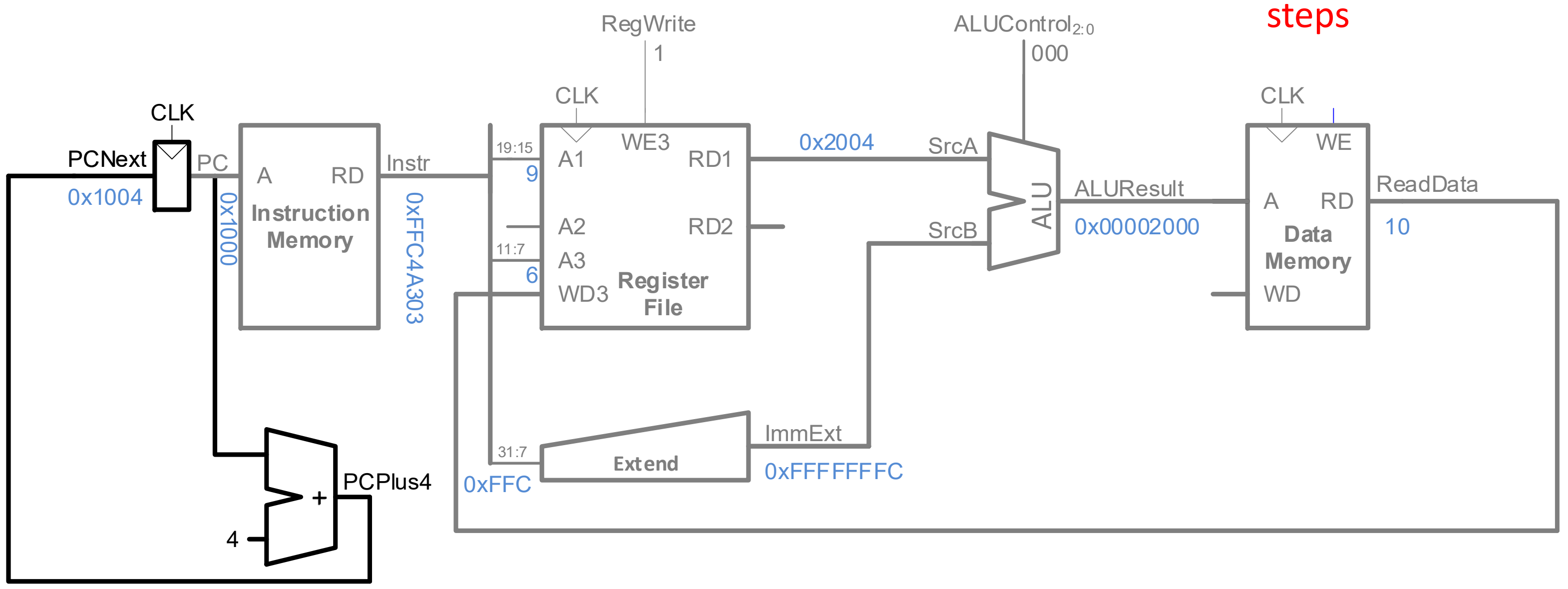


Address	Instruction	Type	Fields					Machine Language
			imm$_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Single-Cycle Datapath: lw Increment

STEP 6: Determine address of next instruction

Step 6 is concurrent with other steps



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

Example Program

Let’s extend the datapath to enable execution of other (core) instructions

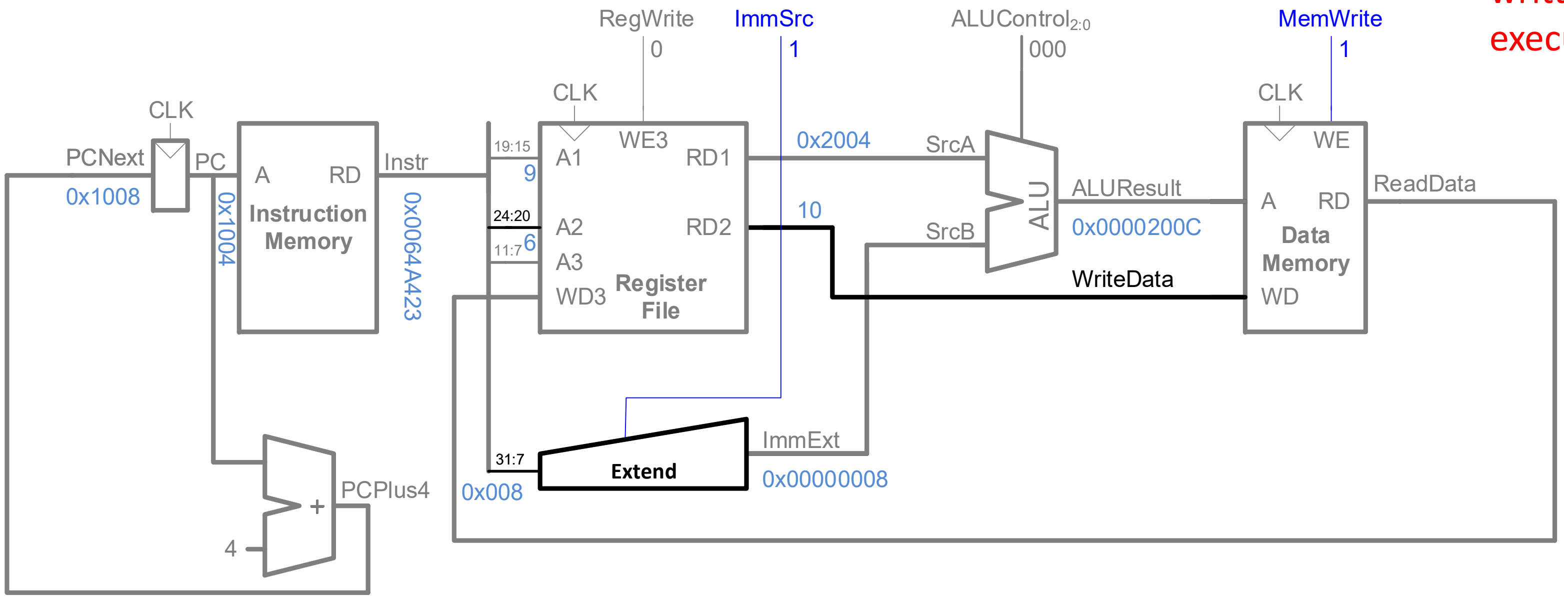
Example Program:

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
			imm$_{11:5}$	**rs2**	**rs1**	**f3**	**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
			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: sw

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

- What is slower, RF or Extend unit?
- Is Rs2 read when executing lw? Is it written to memory? Is something written to RF when executing sw?



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

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

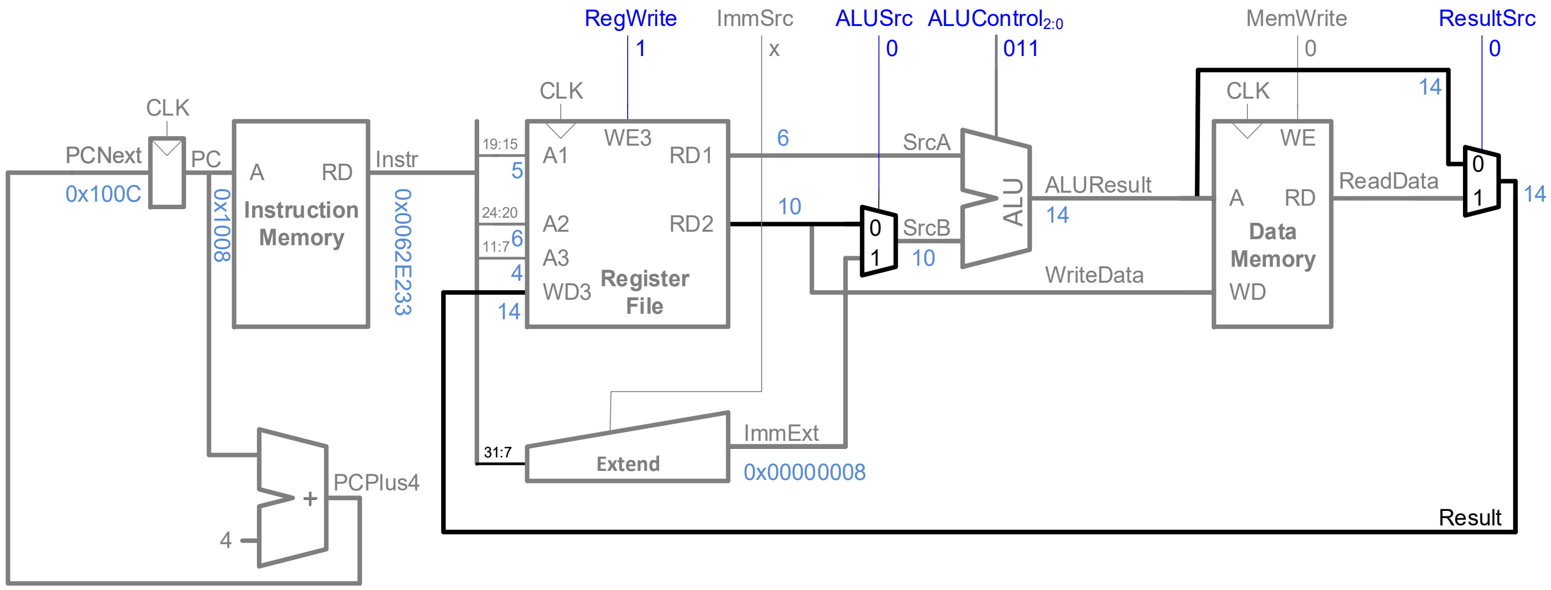
31:25	24:20	19:15	14:12	11:7	6:0
$imm_{11:5}$	rs2	rs1	funct3	**$imm_{4:0}$**	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Single-Cycle Datapath: R-type

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

- What does the output of Extend unit represent?
- Same rs1, rs2, rd field positions help

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

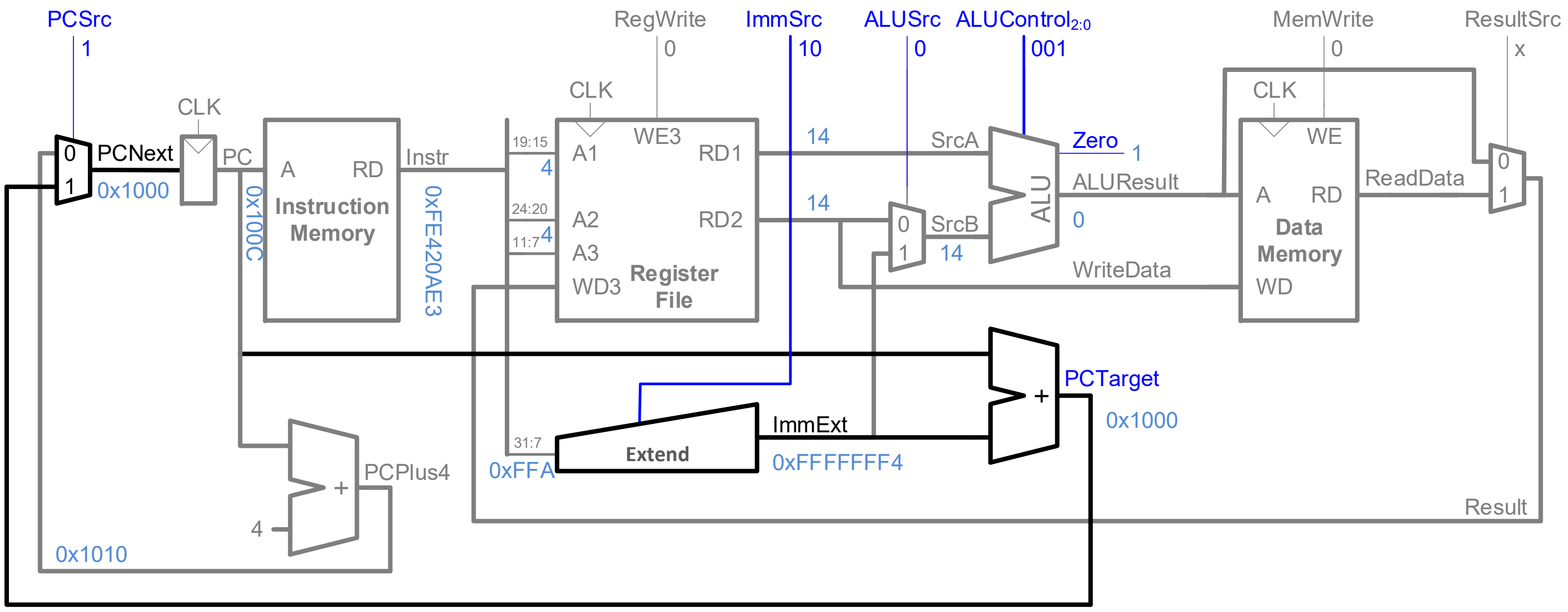


Address	Instruction	Type	Fields						Machine Language
			funct7	rs2	rs1	f3	rd	op	
0x1008	or x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233

Single-Cycle Datapath: beq

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

$\text{ALUControl}_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT



Address	Instruction	Type	Fields						Machine Language
			$\text{imm}_{12,10:5}$	rs2	rs1	f3	$\text{imm}_{4:1,11}$	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

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

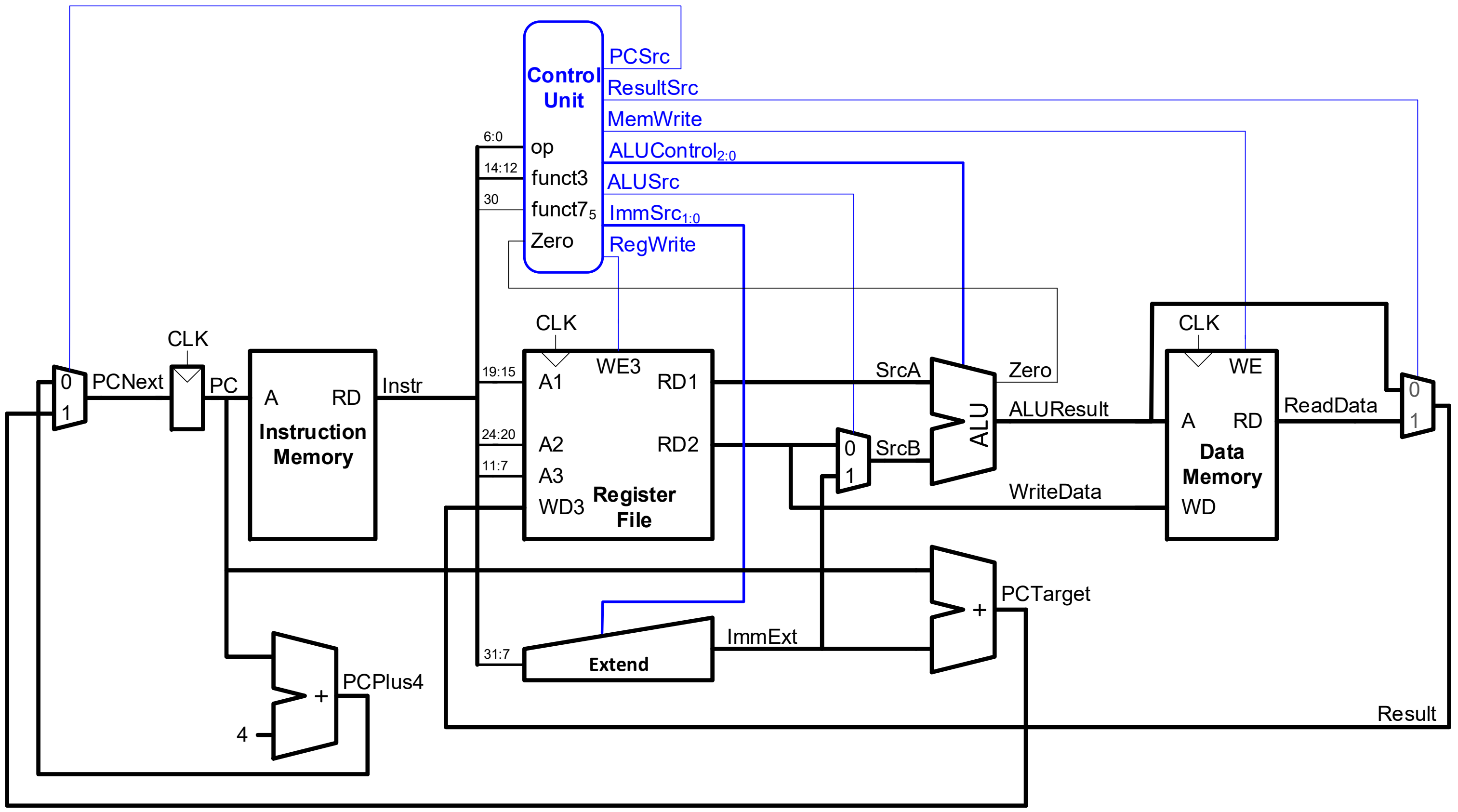
S-Type

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

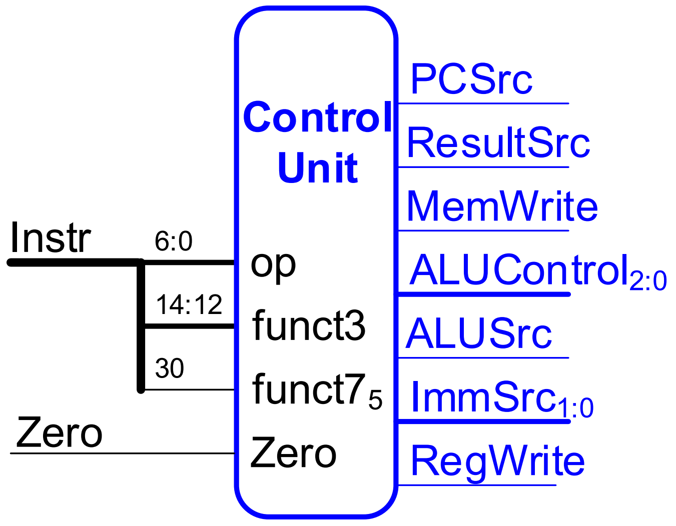
31:25	24:20	19:15	14:12	11:7	6:0
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Single-Cycle RISC-V Processor

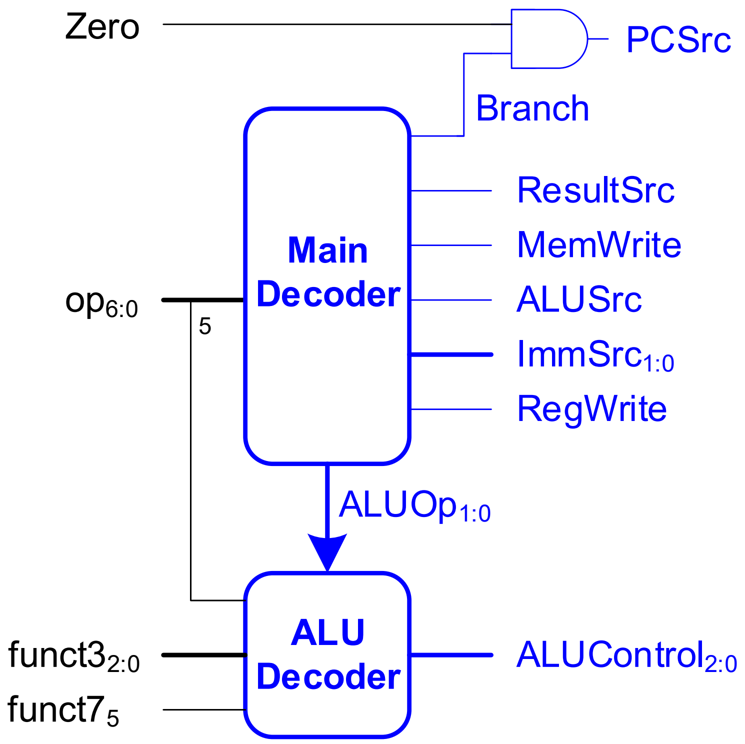


Single-Cycle Control

High-Level View

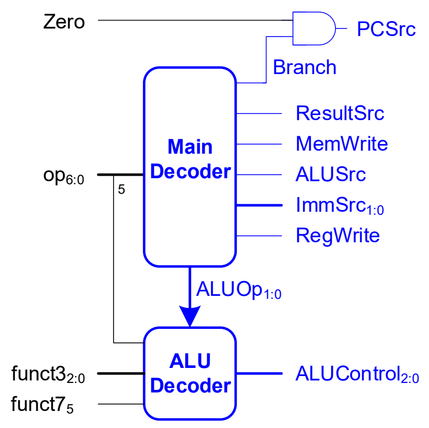


Low-Level View



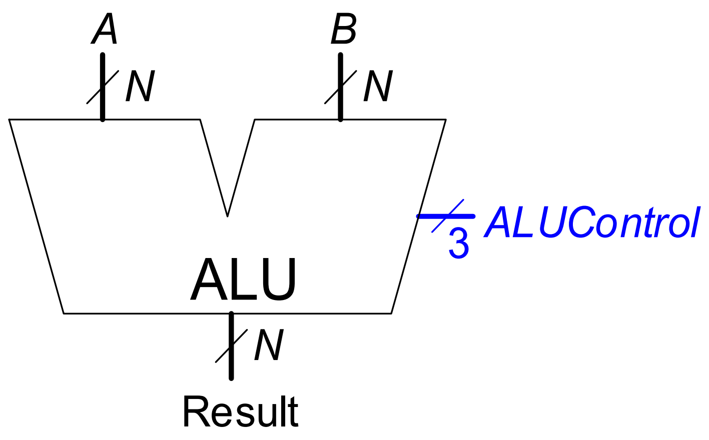
Single-Cycle Control: Main Decoder

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



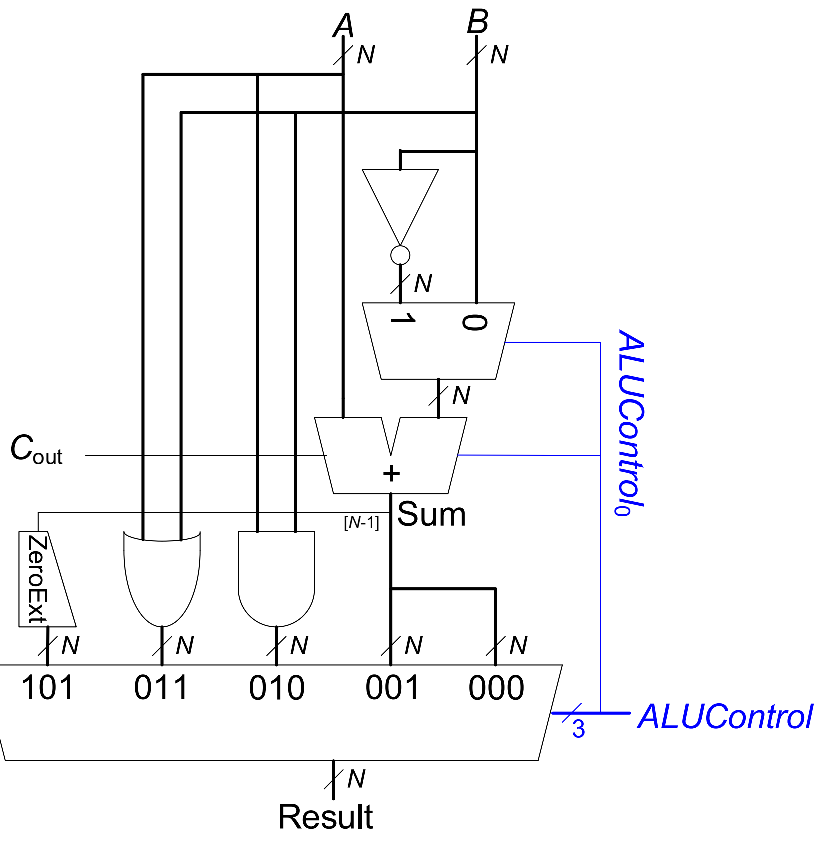
Review: ALU

$\text{ALUControl}_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT

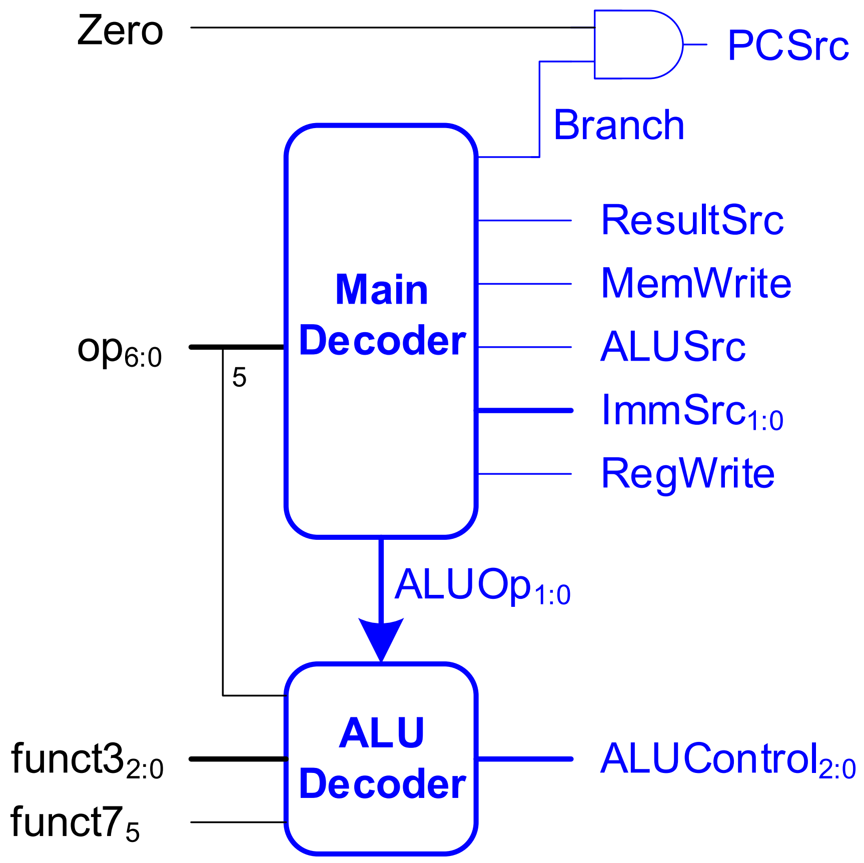


Review: ALU

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

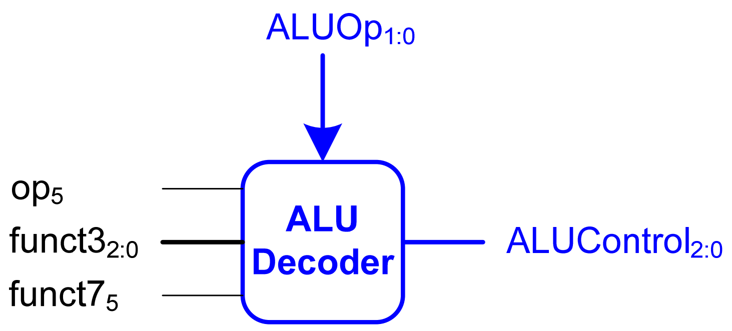


Single-Cycle Control: ALU Decoder



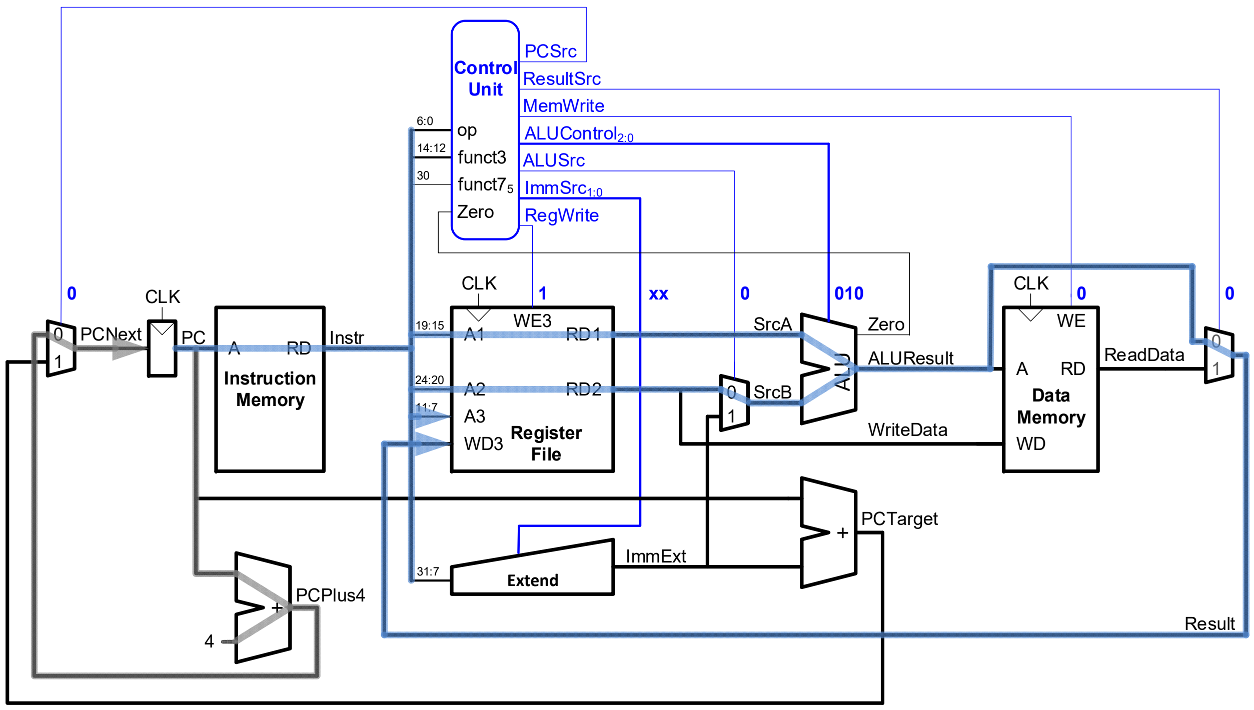
Single-Cycle Control: ALU Decoder

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

Extended Functionality: I-Type ALU

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

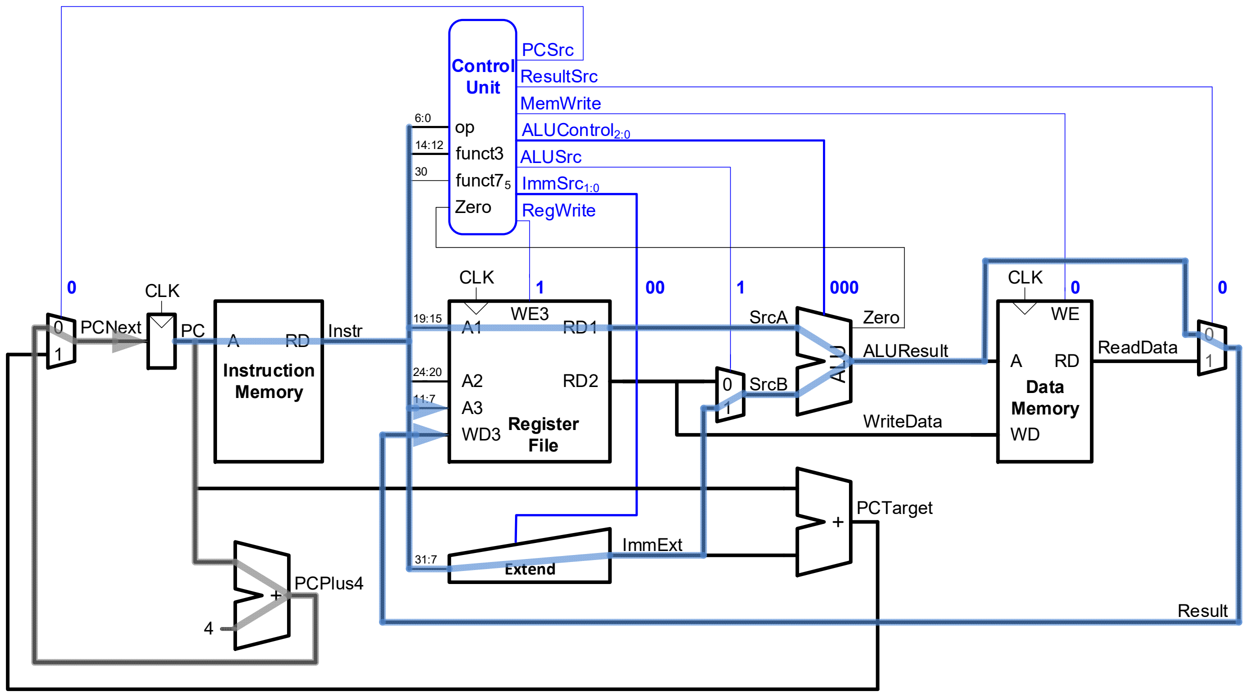
- **Similar to R-type** instructions
- But **second source** comes from **immediate**
- Change ***ALUSrc*** to select the immediate
- And ***ImmSrc*** to pick the correct immediate

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



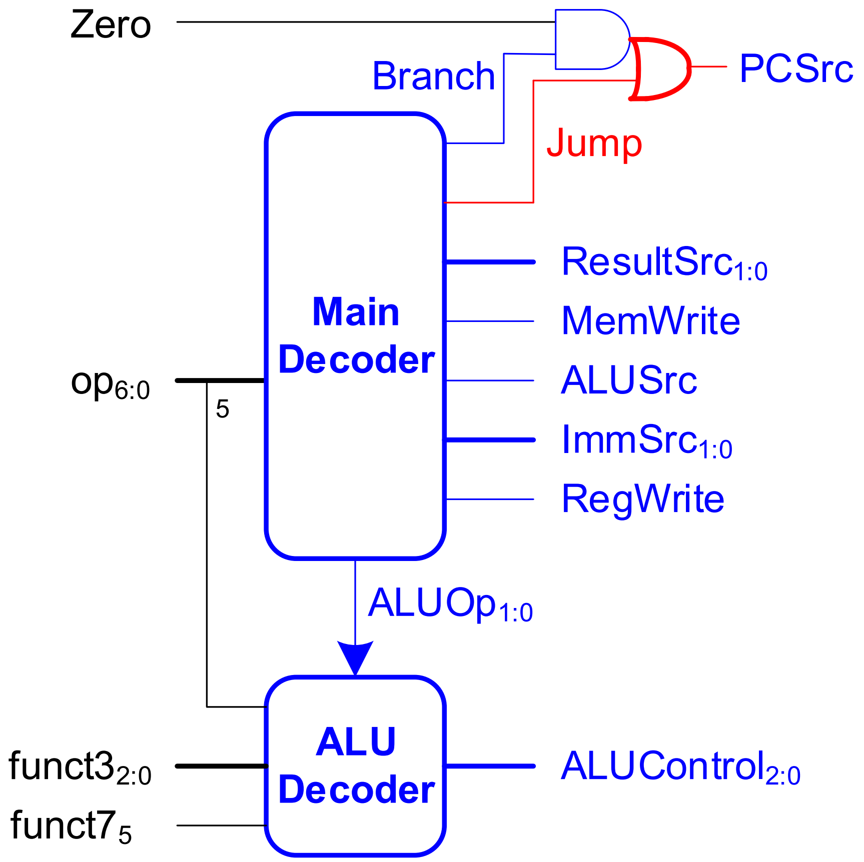
addi x5, x6, -33

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

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

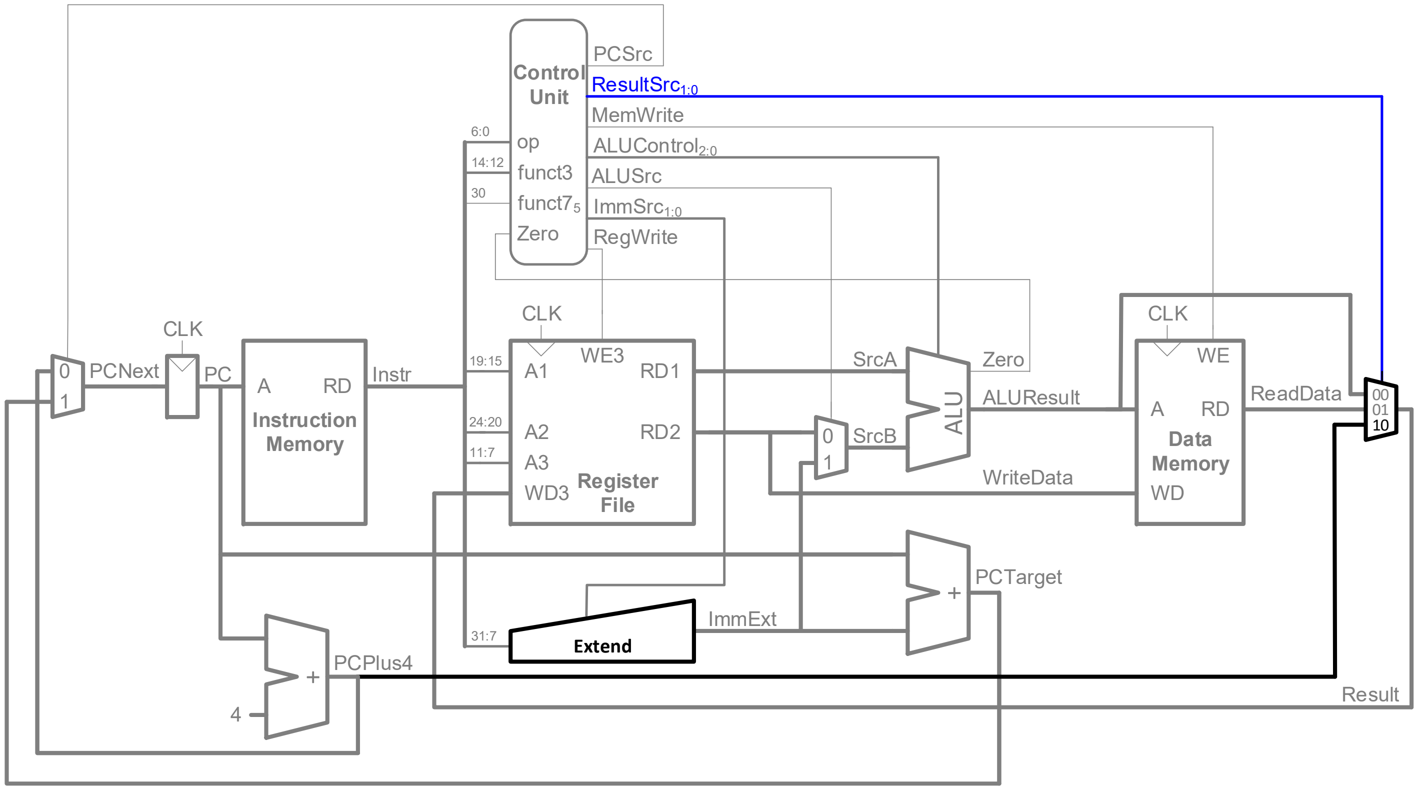
31:12	11:7	6:0
$imm_{20,10:1,11,19:12}$	rd	op
20 bits	5 bits	7 bits

Extended Functionality: jal

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

Extended Functionality: jal

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



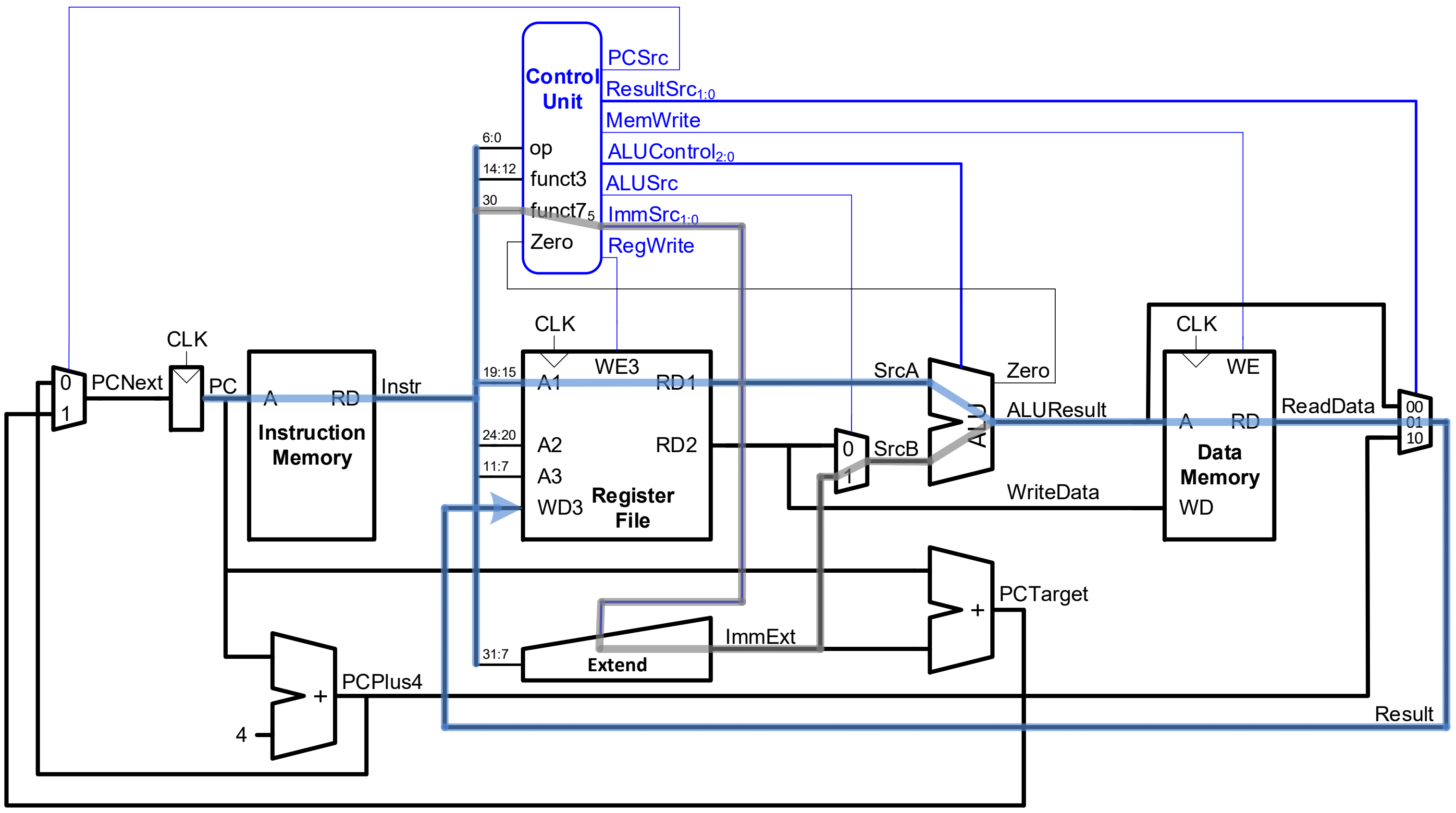
Processor Performance

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`**)

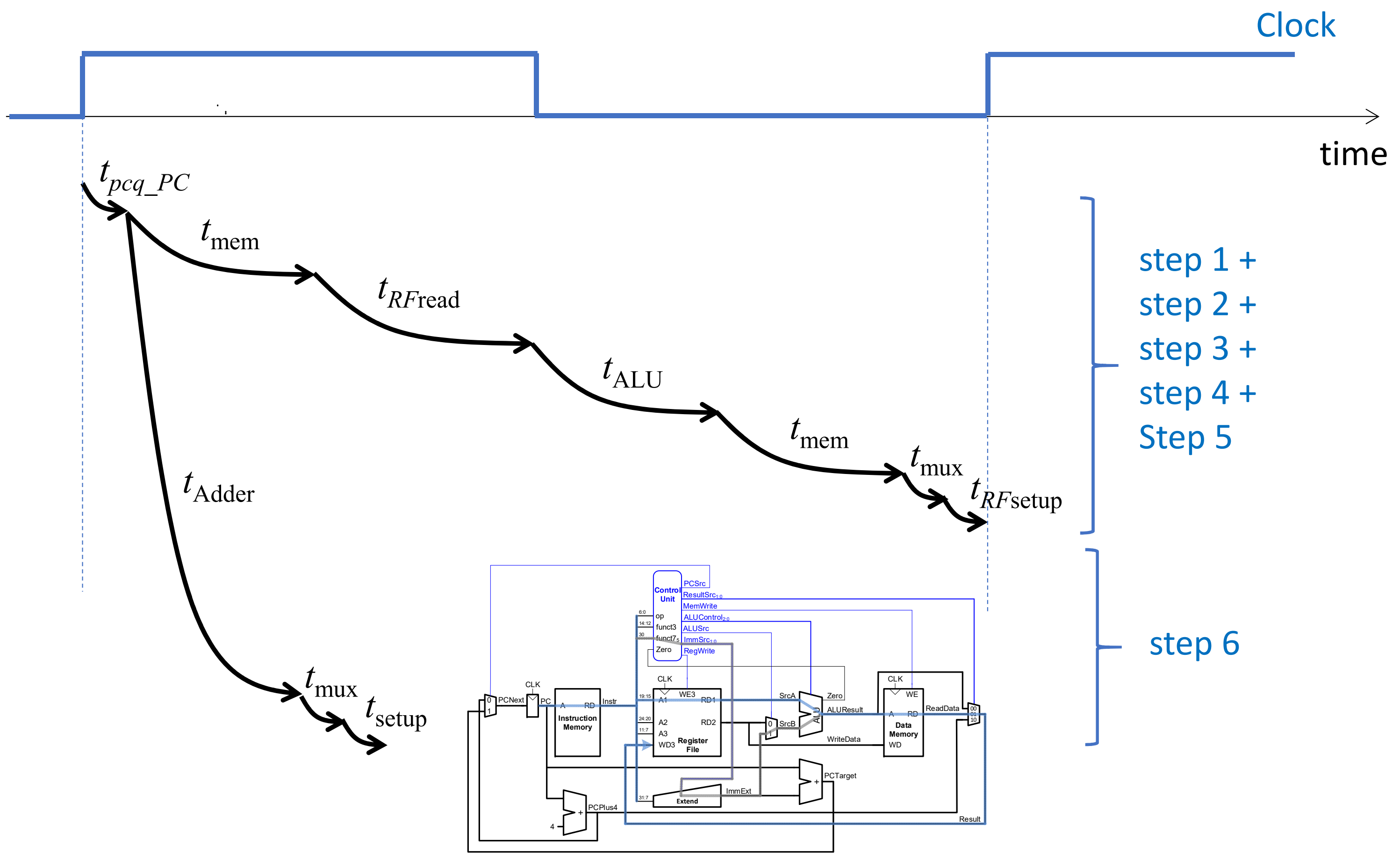
Single-Cycle Processor Performance

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

Timing for LW Instruction



Single-Cycle Processor Performance

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 Processor Example

Program with 100 billion instructions:

Execution Time = # instructions x CPI x T_C