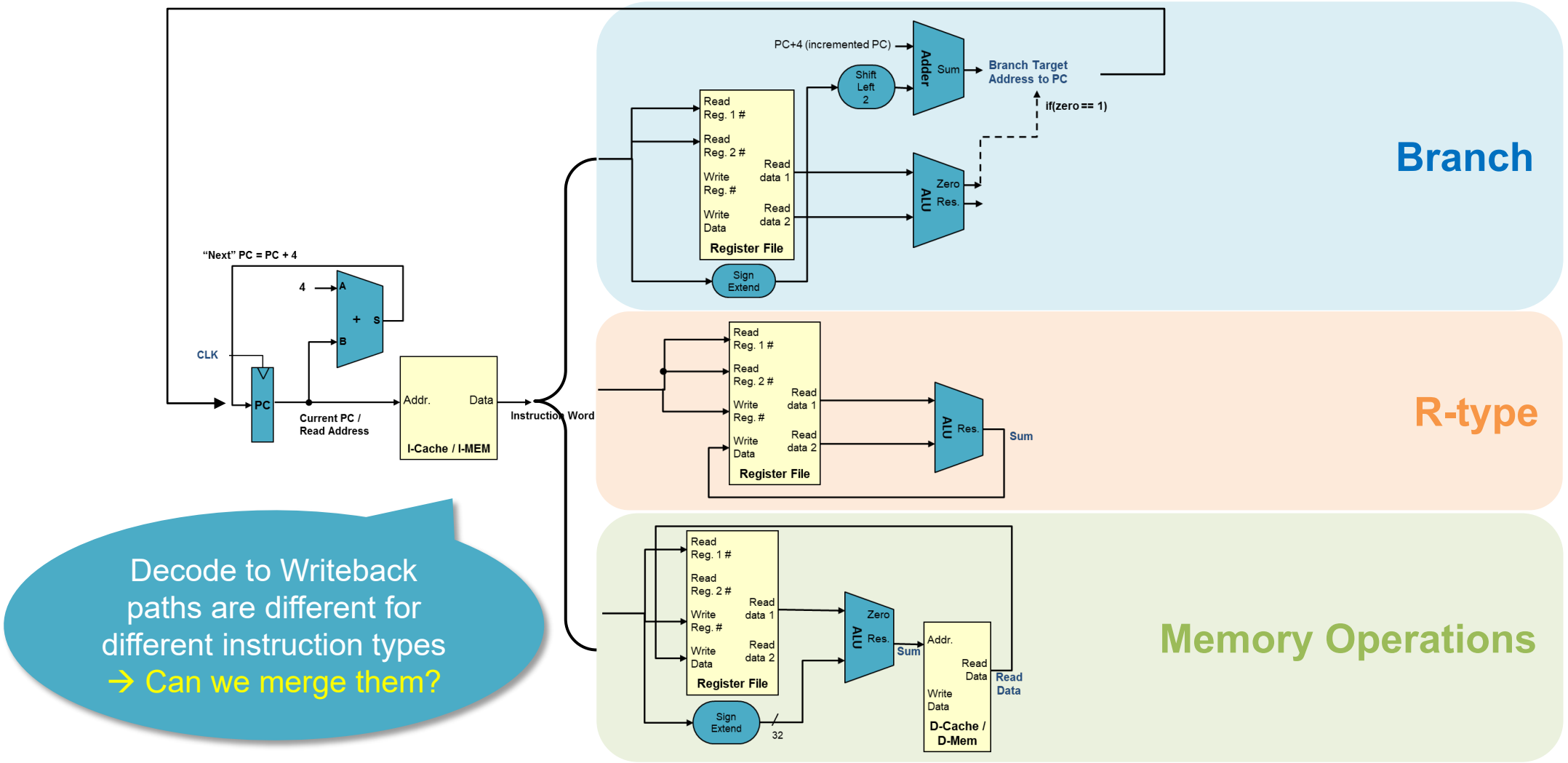
CMPE 200
Computer Architecture & Design

Lecture 3.
Processor Microarchitecture and Design (2)

Haonan Wang

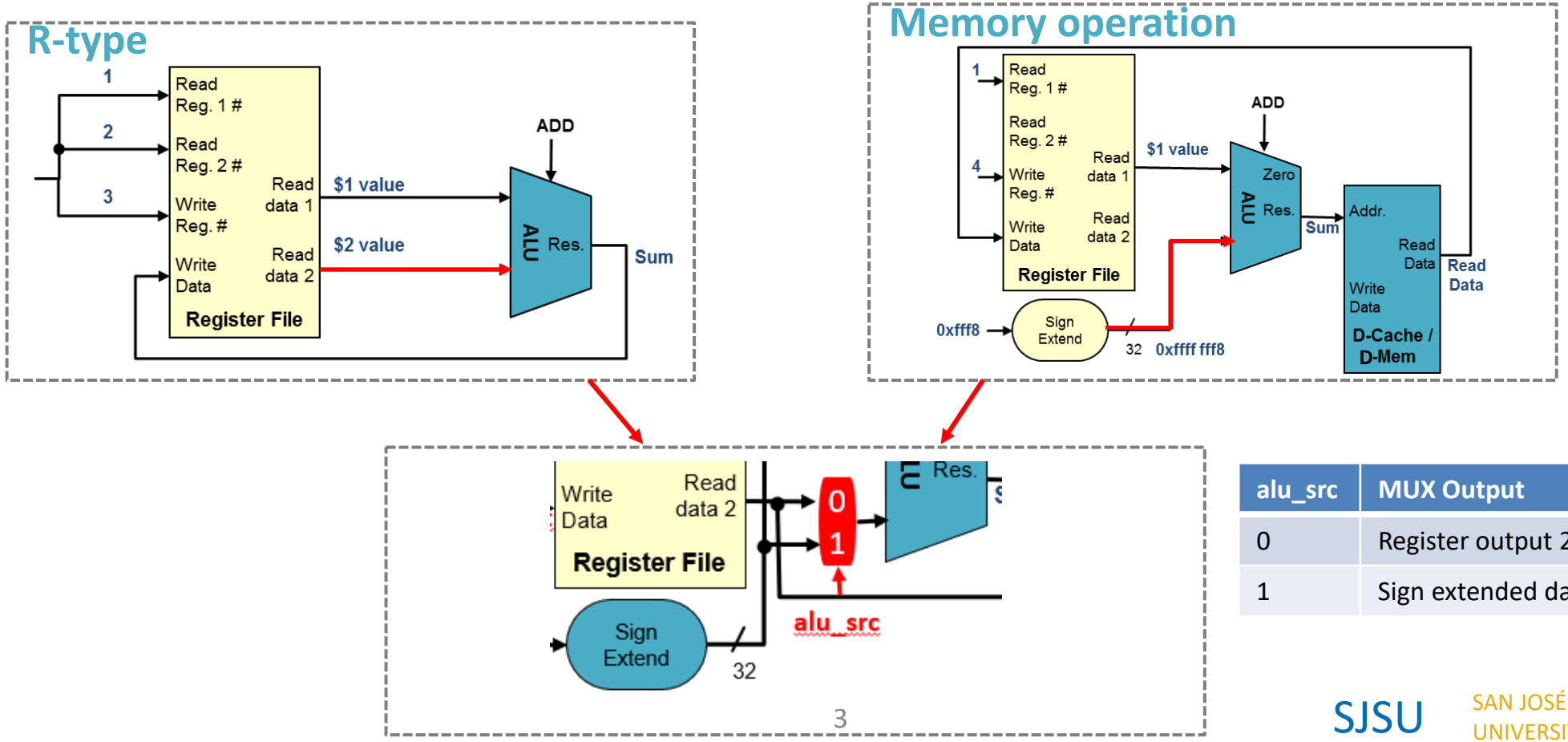
SAN JOSÉ STATE
UNIVERSITY

Single-Cycle Datapath



R-Type + LW/SW (1): alu_src Mux

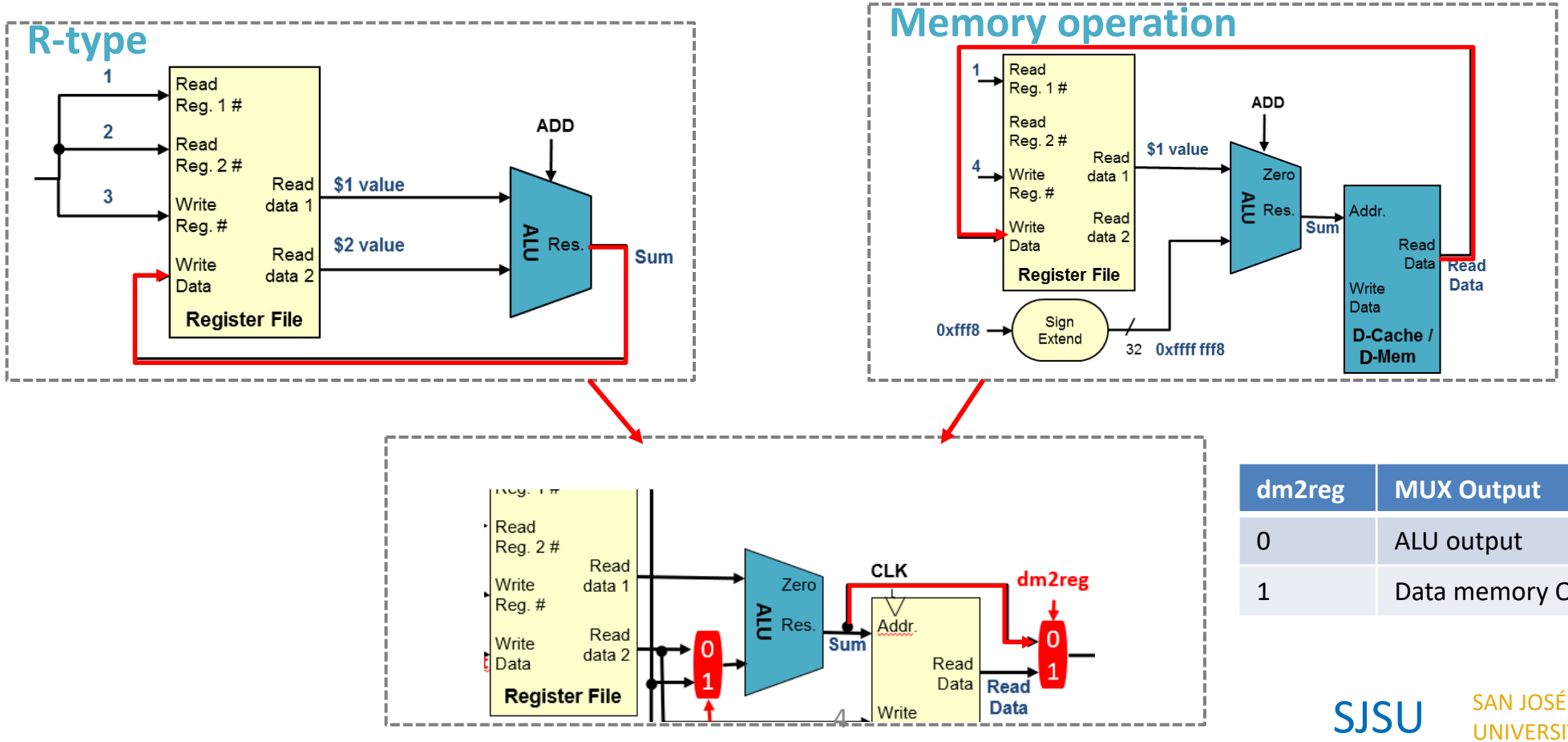
- **Mux controlling second input to ALU**
  - ALU instruction provides Read Register 2 data to the 2nd input of ALU
  - LW/SW uses 2nd input of ALU as an offset to form effective address



alu_src	MUX Output
0	Register output 2
1	Sign extended data

R-Type + LW/SW (2): dm2reg Mux

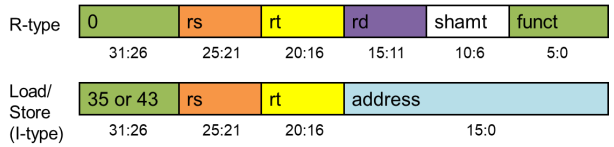
- **Mux controlling writeback value to register file**
  - ALU instructions use the result of the ALU
  - LW uses the read data from data memory

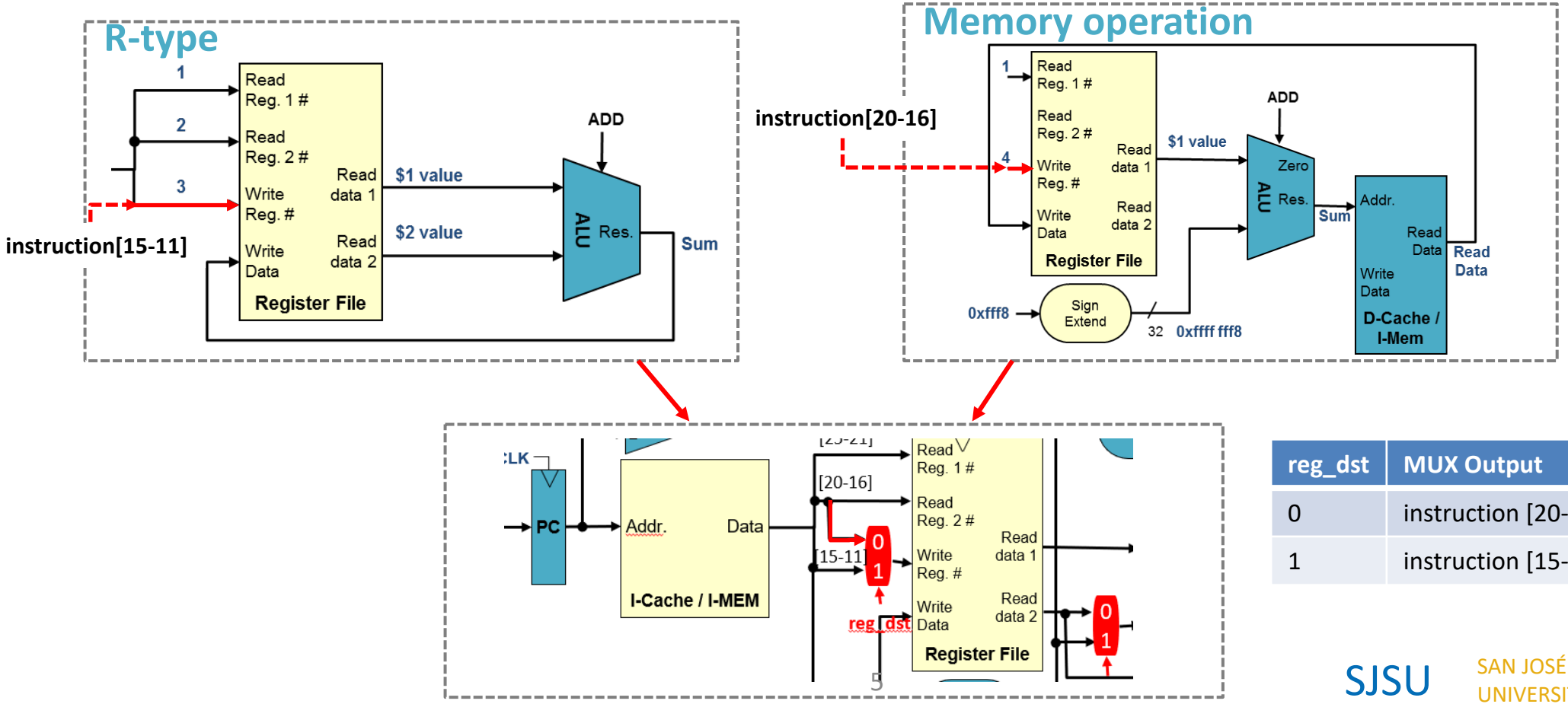


dm2reg	MUX Output
0	ALU output
1	Data memory Output

R-Type + LW: reg_dst Mux

- **Different destination register ID fields for ALU and LW instructions**



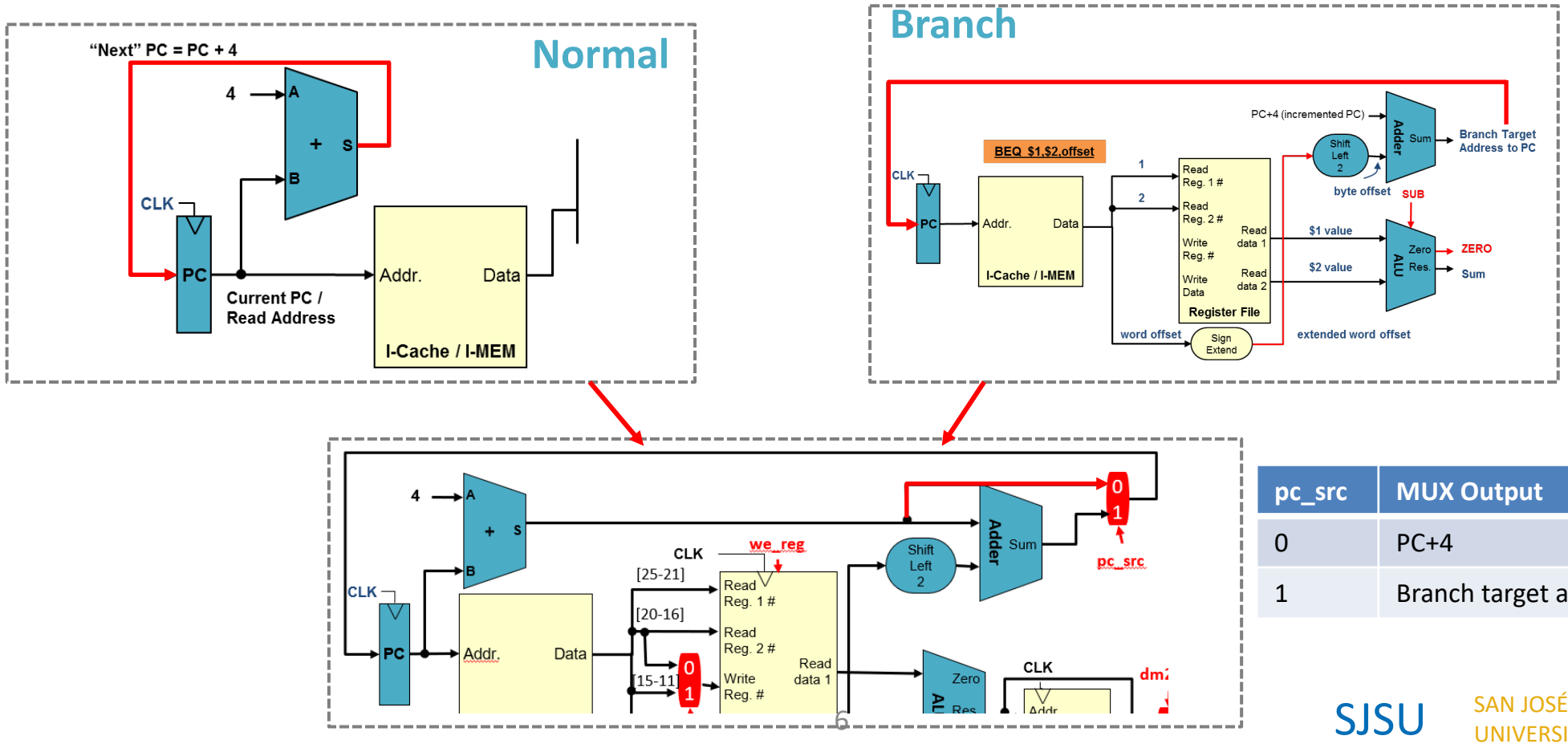


reg_dst	MUX Output
0	instruction [20-16]
1	instruction [15-11]

Branch + Others: pc_src Mux

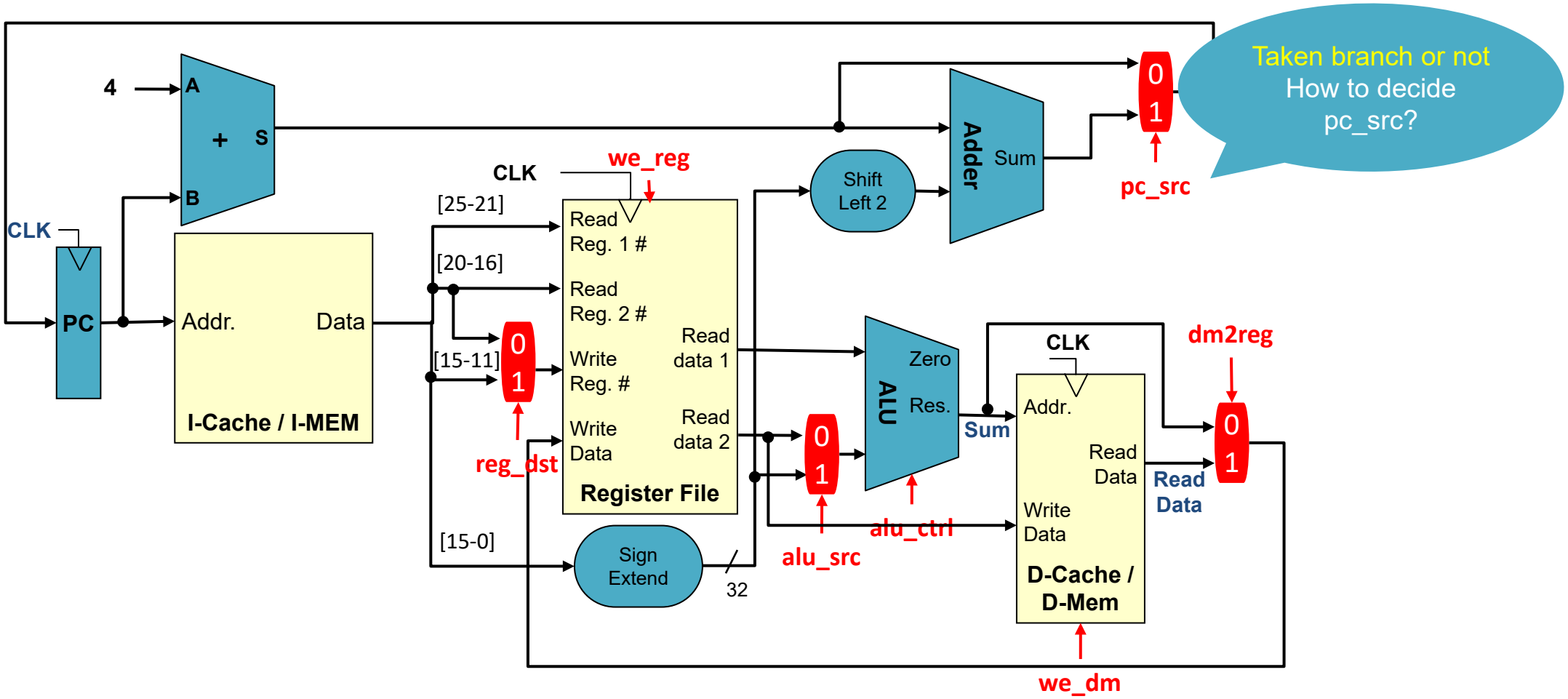
- Next instruction can either be at the next sequential address (PC+4) or the branch target address (PC+offset)

Question: where should we apply mux(es)?

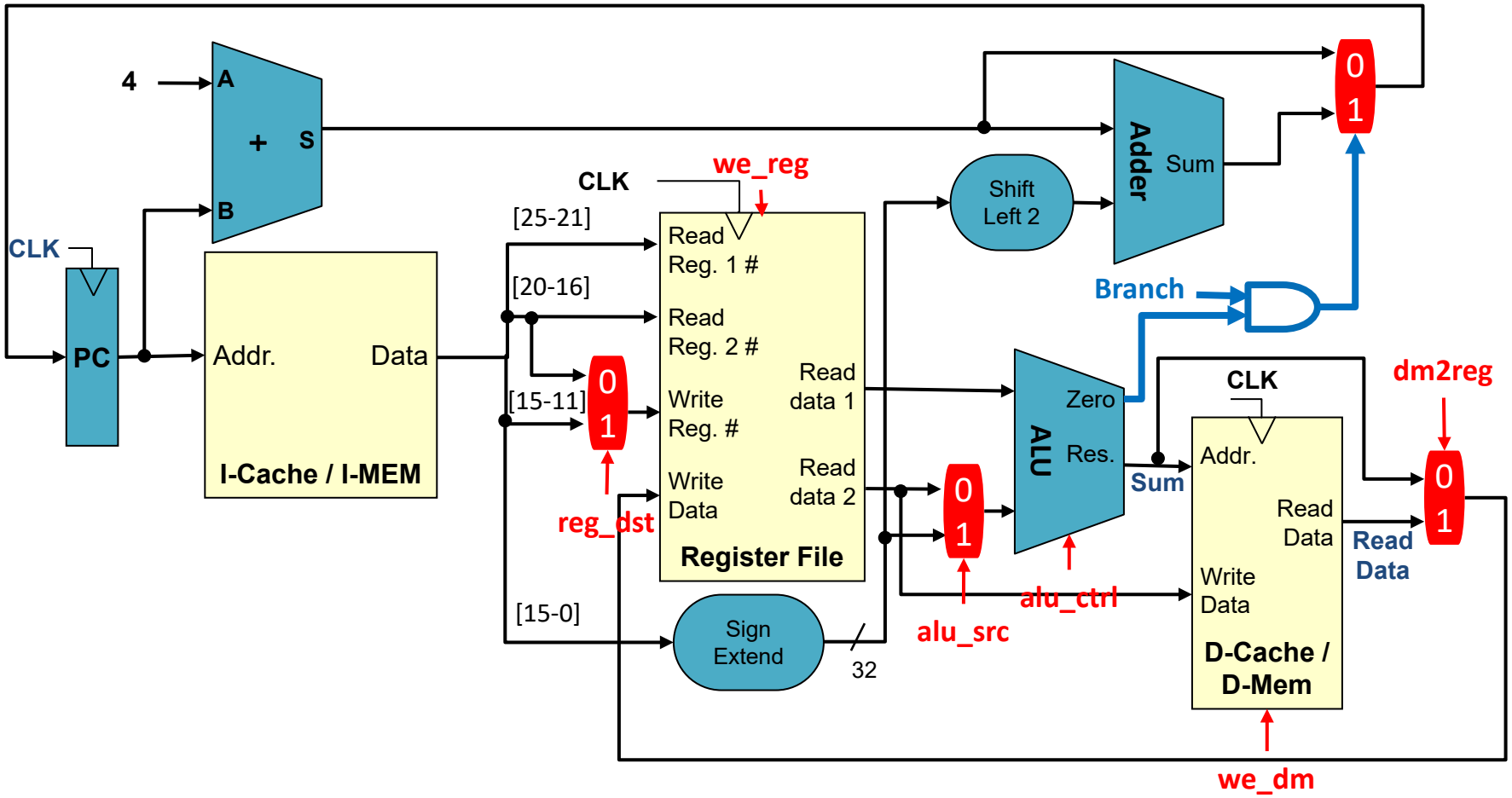


pc_src	MUX Output
0	PC+4
1	Branch target address

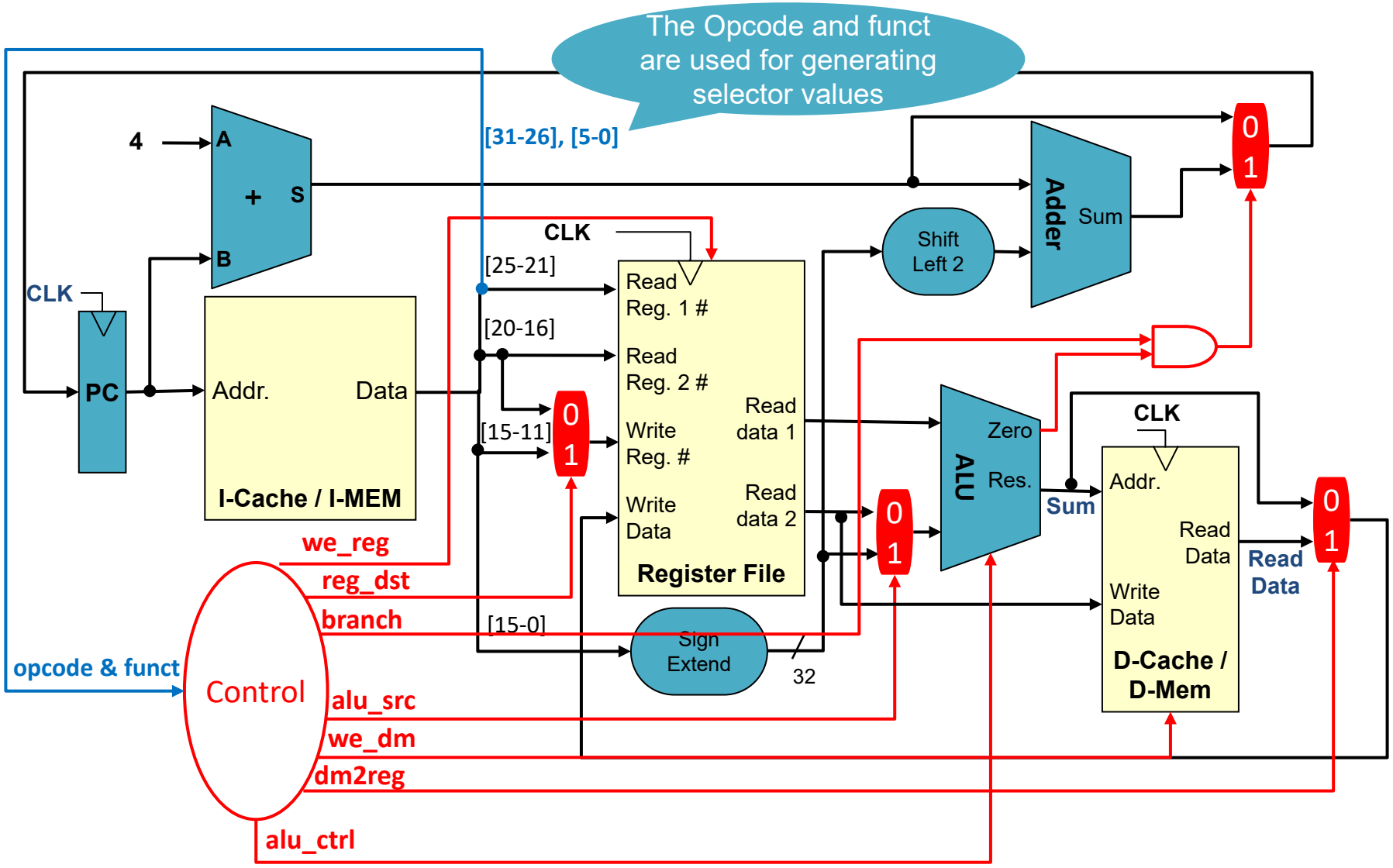
Single-cycle CPU Datapath



Single-cycle CPU Datapath

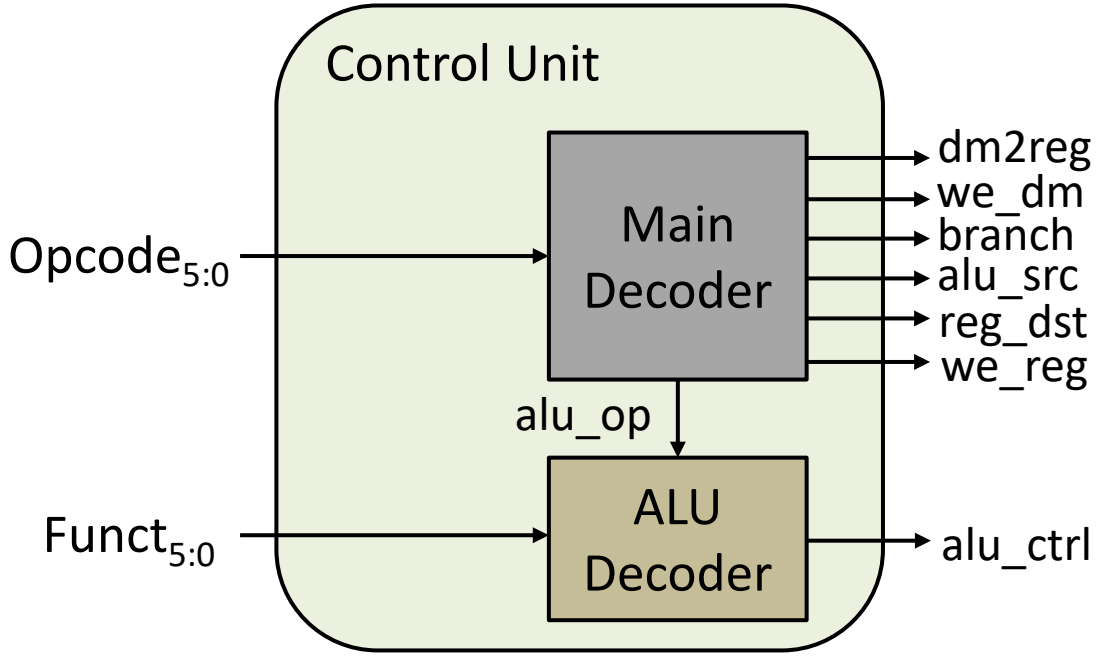


Single-cycle CPU Datapath



Single-cycle CPU Control Unit

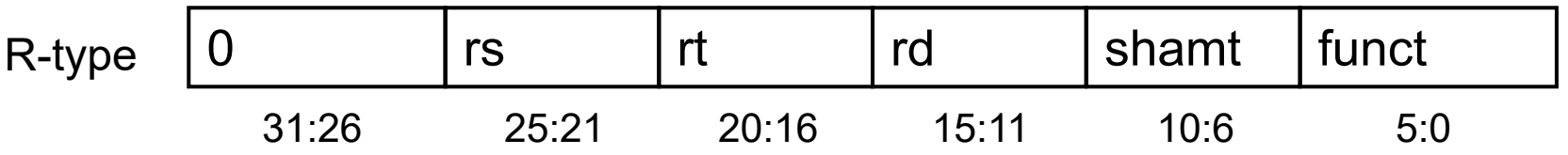
- **The control unit is a combinational decoder for the single-cycle CPU**
- **The CU contains**
  - A main decoder for Opcode
  - An ALU decoder for Funct



ALU Usage

- **Load/Store:**
  - Memory address = Base address + offset → add
- **Branch:**
  - Branch if two values are equal/not equal → subtract
- **R-type:**
  - Opcode & Funct field together indicates an ALU operation

These three instruction types can be distinguished by Opcode



Main Decoder

- **Main Decoder is used to generate instruction type indicator & mux control**
  - To cover 3 types of instructions, we need a **2-bit indicator, alu_op**
  - alu_op will be the input of ALU Decoder

- **For R-type instructions, the ALU Decoder will also check the Funct field**

	Input	Output						
Instruction	Opcode	we_reg	reg_dst	alu_src	branch	we_dm	dm2reg	$alu\_op_{1:0}$
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

ALU Decoder

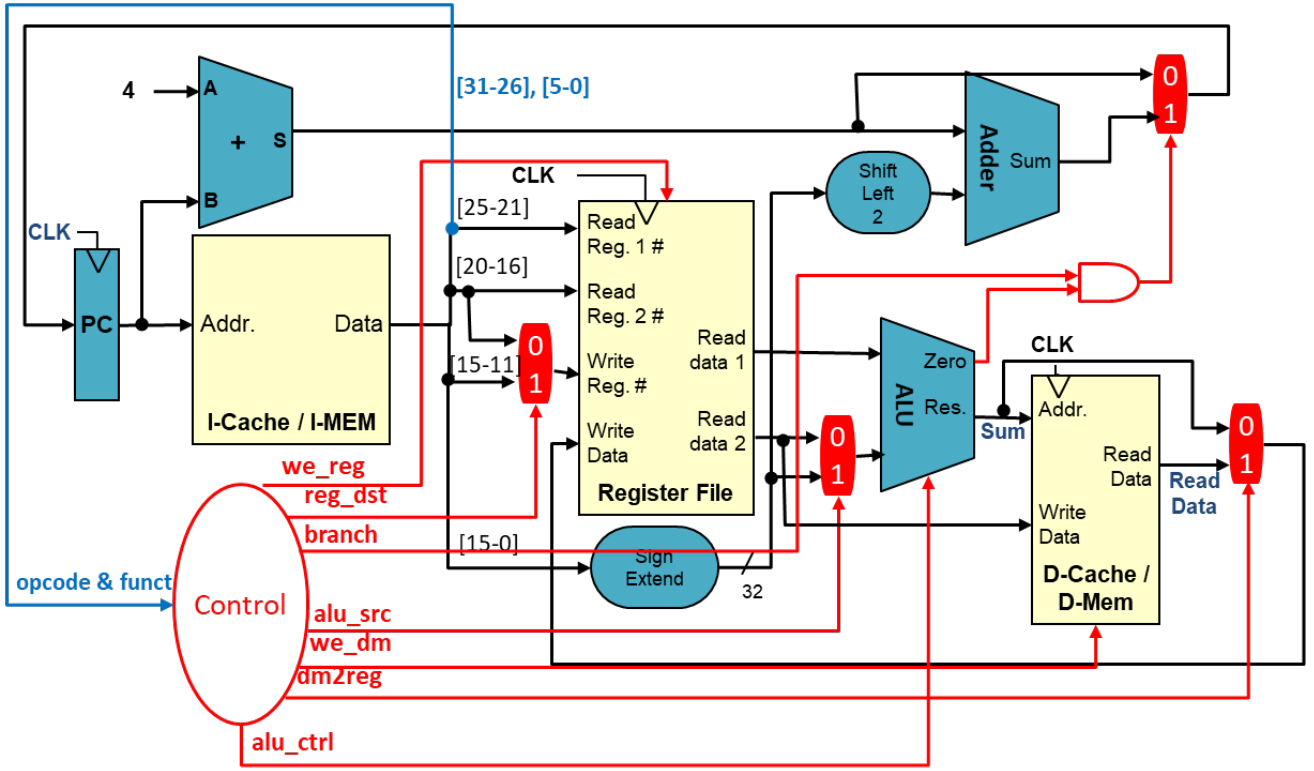
opcode	$alu\_op_{1:0}$ (Input)	Operation	funct (Input)	ALU function	$alu\_ctrl_{2:0}$ (Output)
lw	00	load word	XXXXXX	ADD	010
sw	00	store word	XXXXXX	ADD	010
beq	01	branch equal	XXXXXX	SUB	110
R-type	10	ADD	100000	ADD	010
		SUB	100010	SUB	110
		AND	100100	AND	000
		OR	100101	OR	001
		SLT	101010	SLT	111

So Far…

- **Our design supports the following instructions**
  - Memory Reference Instructions:
    - **LW, SW**
  - Arithmetic and Logic Instructions:
    - **ADD, SUB, AND, OR, SLT**
  - Branch Instructions:
    - **BEQ**

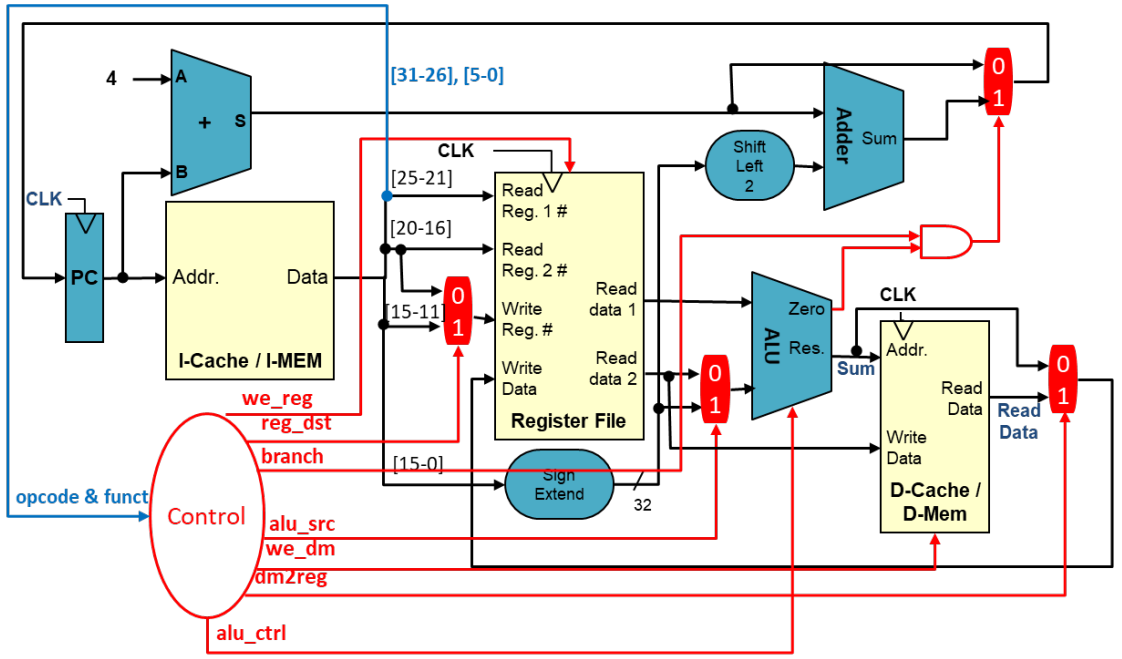
- **How to extend the design to support more MIPS instructions?**

Extension to Support addi



- No need to extend the current datapath
- Need to extend the control unit, specifically only the **Main Decoder**

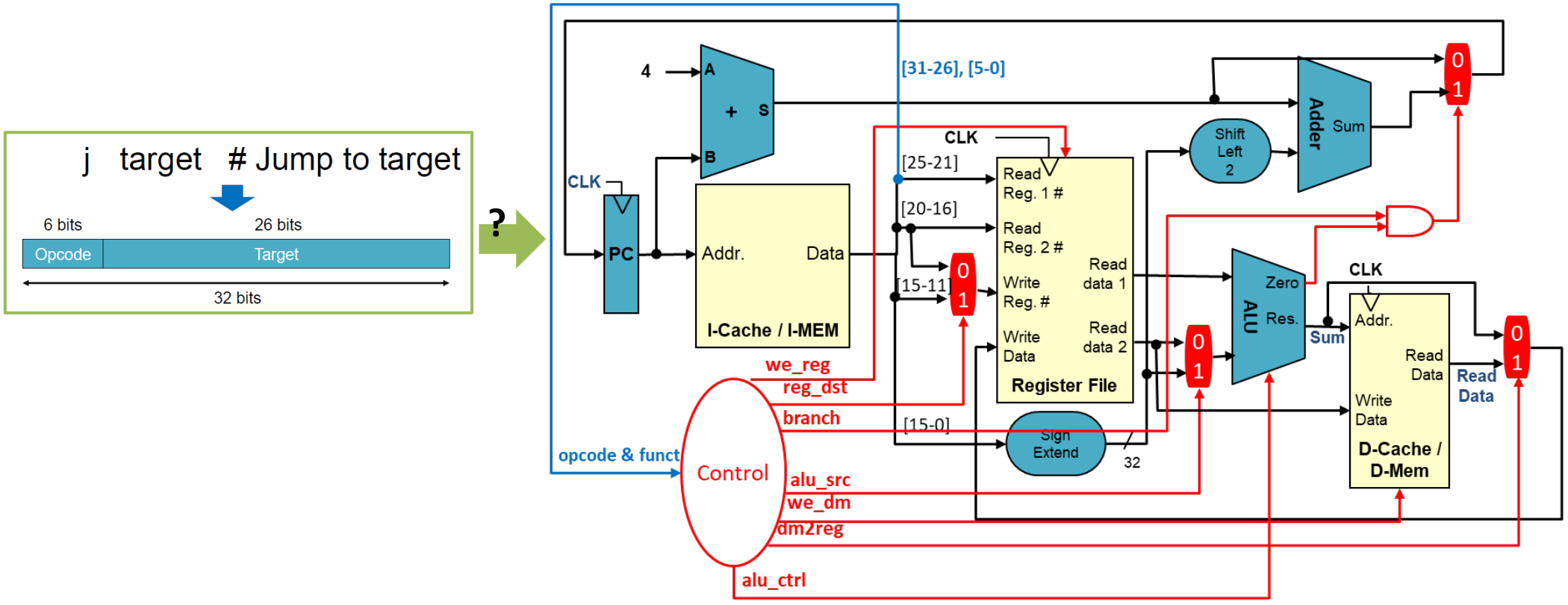
Main Decoder for addi



$alu\_op_{1:0}$	Main Decoder's Message
00	Do "ADD" operation for lw and sw
01	Do "SUB" operation for beq
10	Check Funct field for R-type
11	Not used

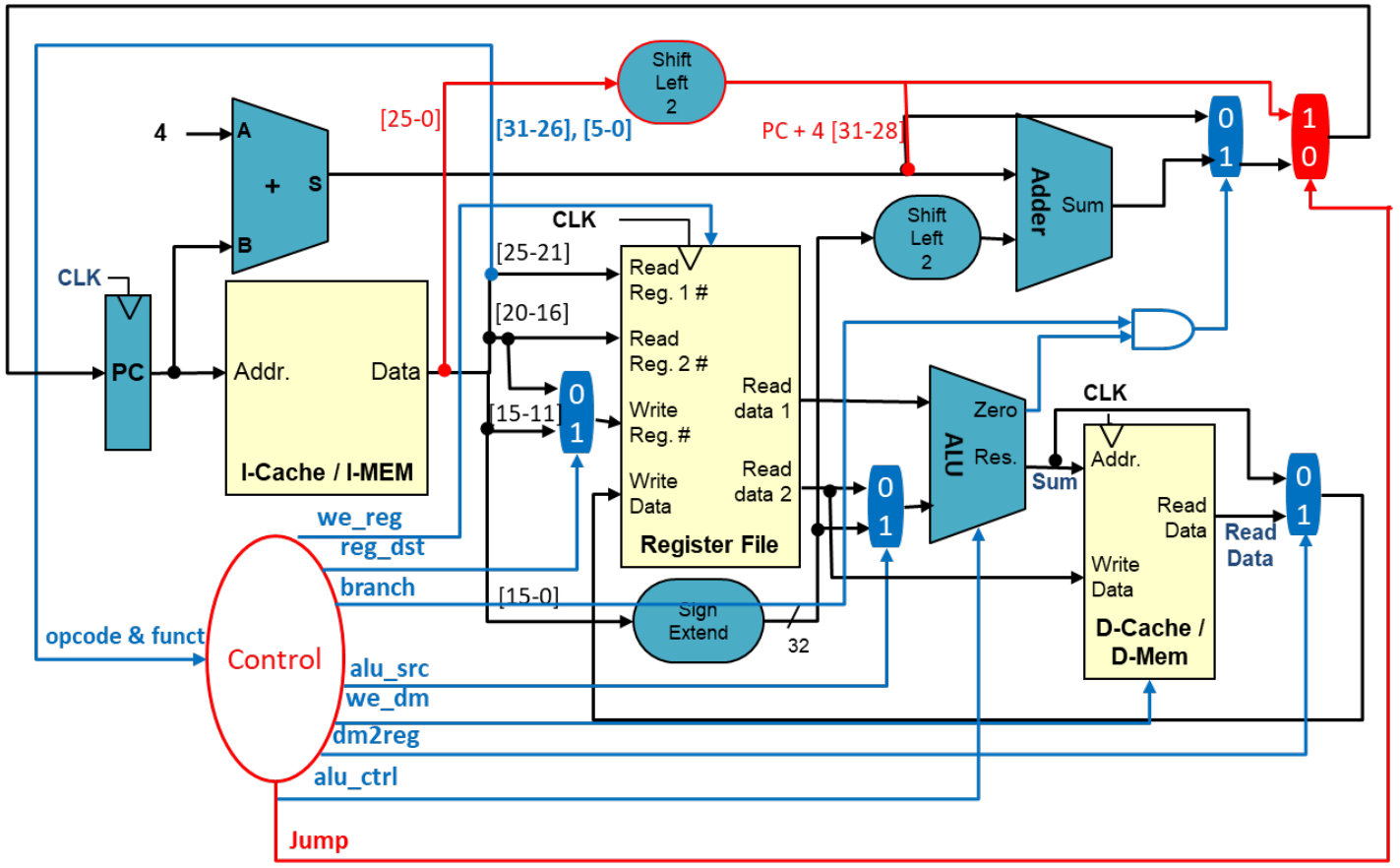
Instruction	Opcode	we_reg	reg_dst	alu_src	branch	we_dm	dm2reg	$alu\_op_{1:0}$
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01
addi	**001000**							

Extension to Support `j`



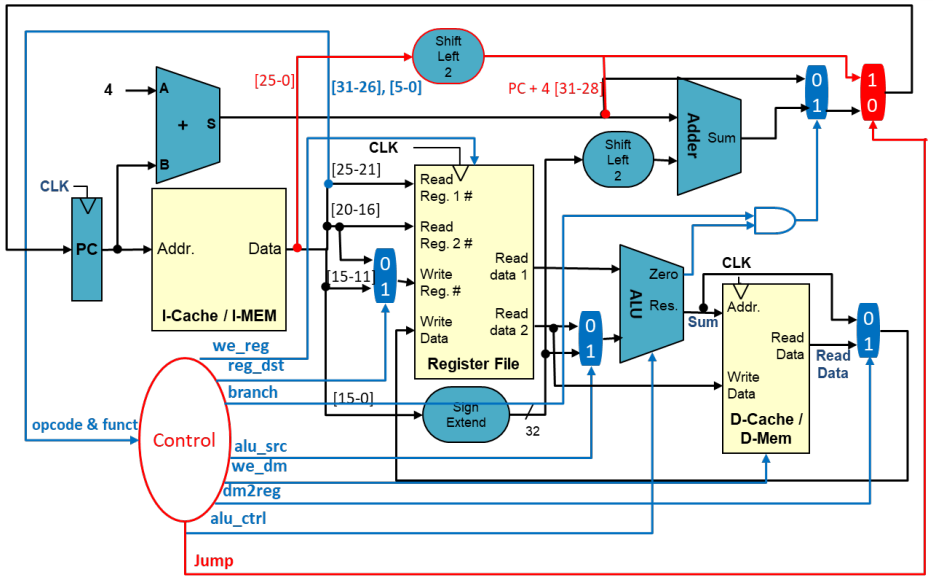
- Can we support `j` with this datapath?

Extension to Support `j`



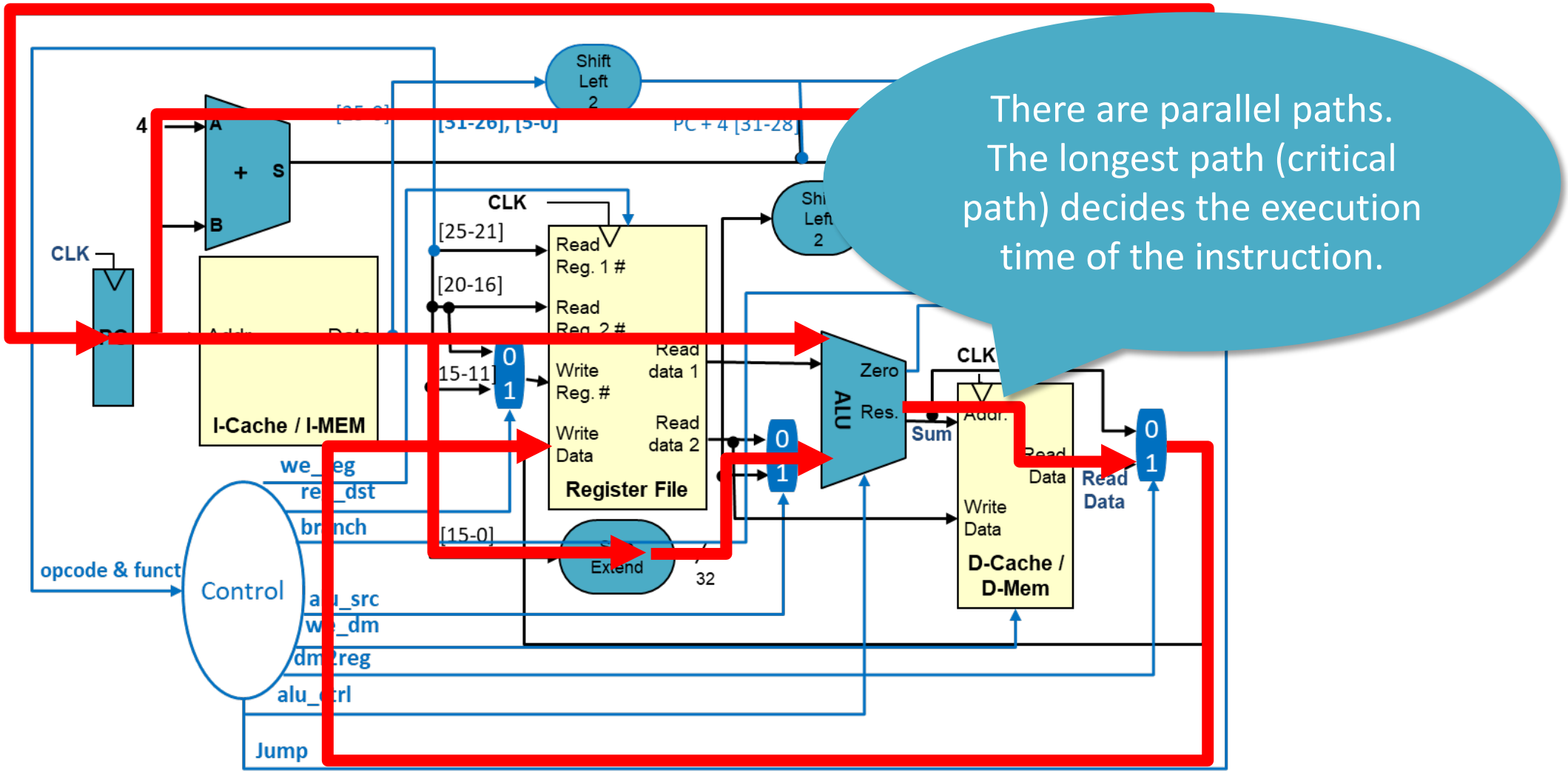
- **Both datapath and control unit (main decoder) need to be extended**

Main Decoder for `j`



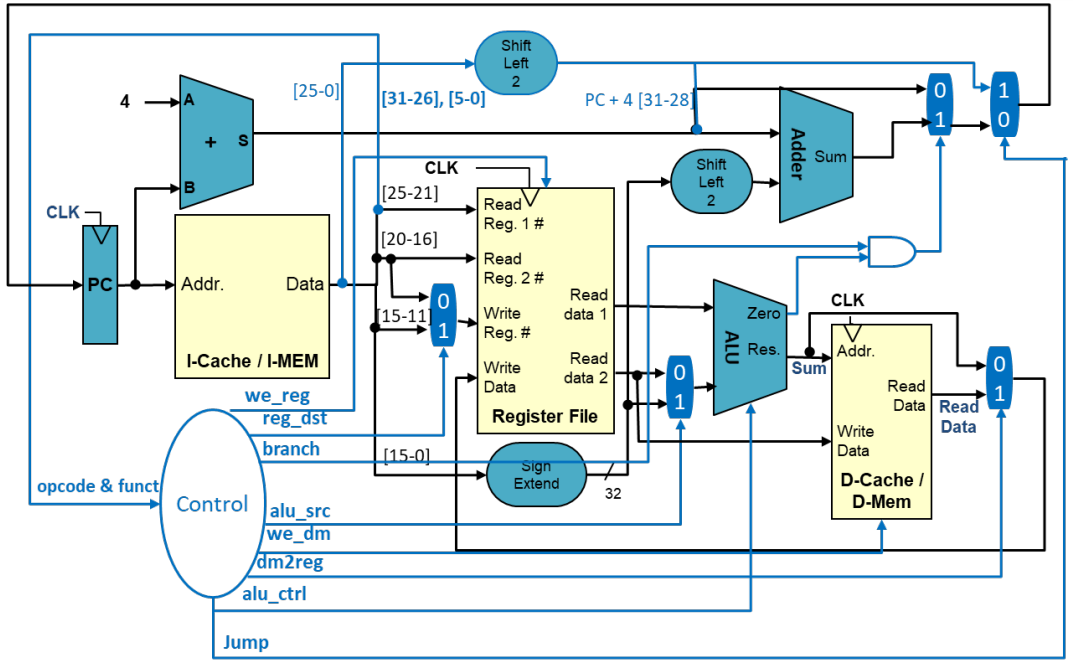
Instruction	Opcode	we_reg	reg_dst	alu_src	branch	we_dm	dm2reg	$alu\_op_{1:0}$	Jump
R-type	000000	1	1	0	0	0	0	10	0
lw	100011	1	0	1	0	0	1	00	0
sw	101011	0	X	1	0	1	X	00	0
beq	000100	0	X	0	1	0	X	01	0
addi	001000	1	0	1	0	0	0	00	0
j	**000010**	**0**	**X**	**X**	**X**	**0**	**X**	**XX**	**1**

Datapath of LW instruction



Single-Cycle CPU Performance Analysis

Function Unit	Parameter	Delay (ps)
Register clock-to-Q	T_{pcq_PC}	30
MUX	T_{MUX}	25
Sign-extend	T_{s_ext}	25
ALU	T_{ALU}	200
Mem read	T_{mem}	250
Register file read	T_{RFread}	150
Register file write	$T_{RFwrite}$	20



Critical path for `LW` =

= (30 + 250 + 150 + 200 + 250 + 25 + 20) ps

= 925 ps

Single-Cycle CPU Performance Analysis

- **The longest instruction's execution time is 925 ps.**
- **If we run 100 billion instructions, what is the total execution time?**

The clock cycle period should be set to 925 ps

CPU time = # instructions x CPI x cycle period

$= 100 \times 10^{9} \times 1 \times 925 \times 10^{-12}$

= 92.5 seconds

Performance Issues

- **Longest delay determines clock period**
  - Critical path: load instruction
  - Instruction memory → register file → ALU → data memory → register file

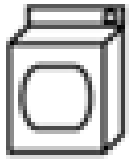
- **Violates design principle**
  - Making the common case fast
  - More arithmetic operations than memory operations

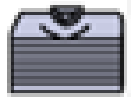
- **Most modern CPUs use multi-cycle processors**
  - Each instruction takes multiple cycles
  - Multiple instructions can execute concurrently (parallelism → pipelining)

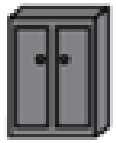
Pipelining Analogy

- **Laundry tasks**


Place one dirty load of clothes in the washer

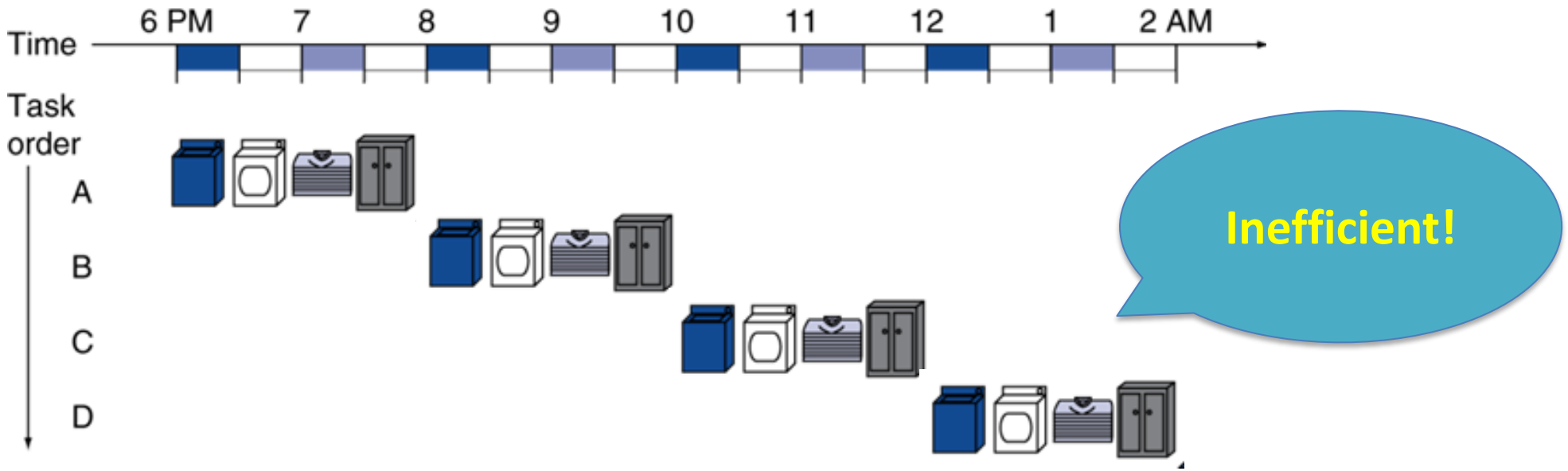

Place the wet load in the dryer


Place the dry load on a table and fold


Ask your roommate to put the clothes away

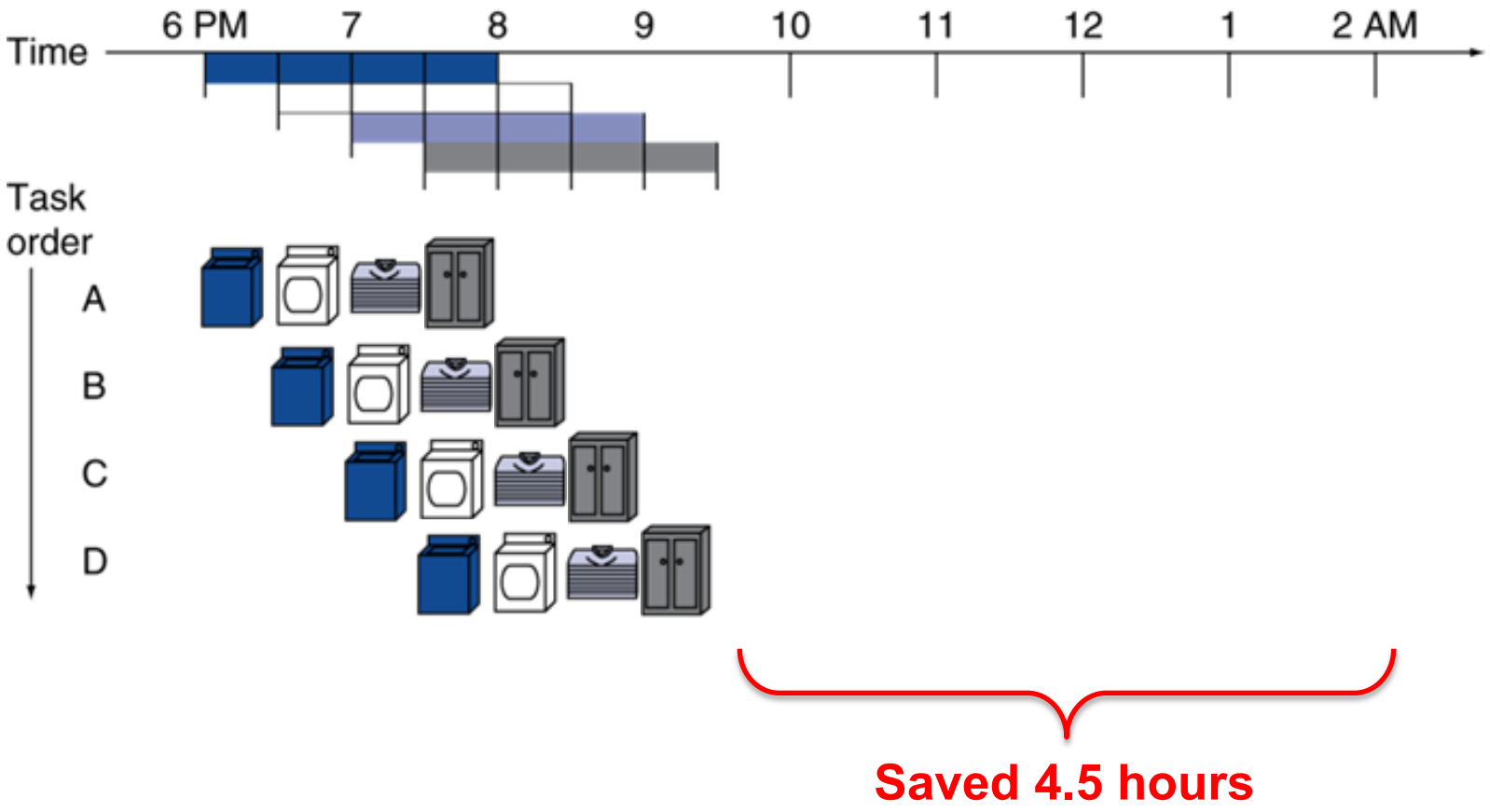
Pipelining Analogy

- **Lots of dirty clothes -- need to wash them over four separate loads**
- **Assume each task takes 30 minutes**



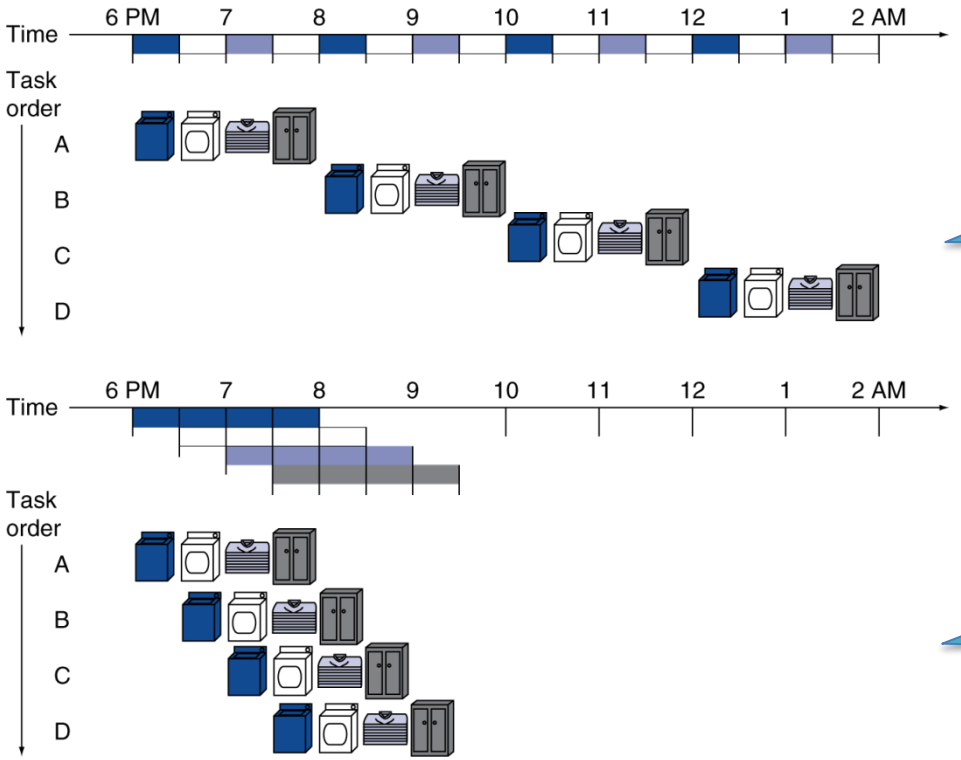
Pipelining Analogy

- **You can start the next load as soon as the washer finishes washing the previous load**



Pipelining Analogy

- **We can parallelize the single-cycle CPU tasks in a similar way**
- **Pipelined laundry → overlapping execution**

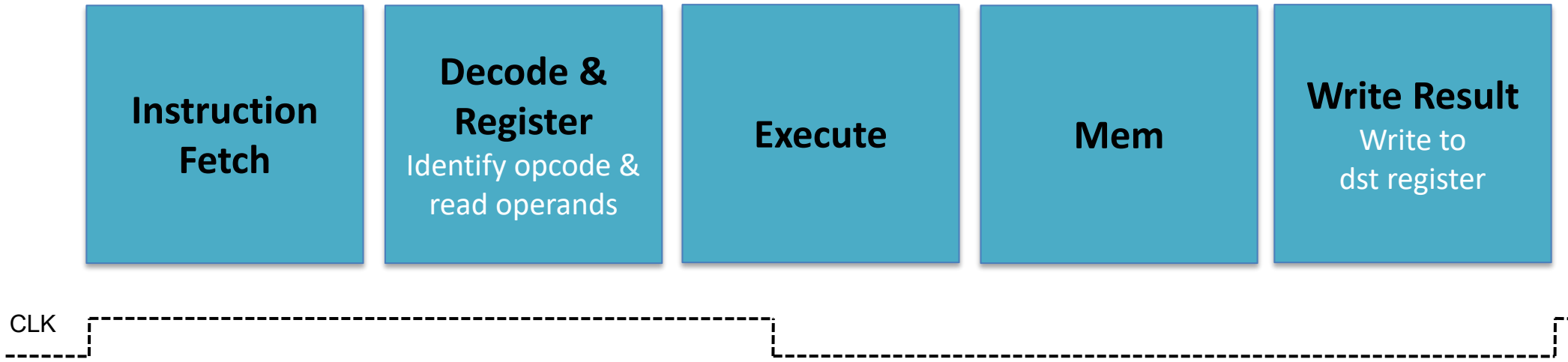


Single-cycle CPU
clock cycle period = 2hrs

Pipelined CPU
clock cycle period = 0.5hrs

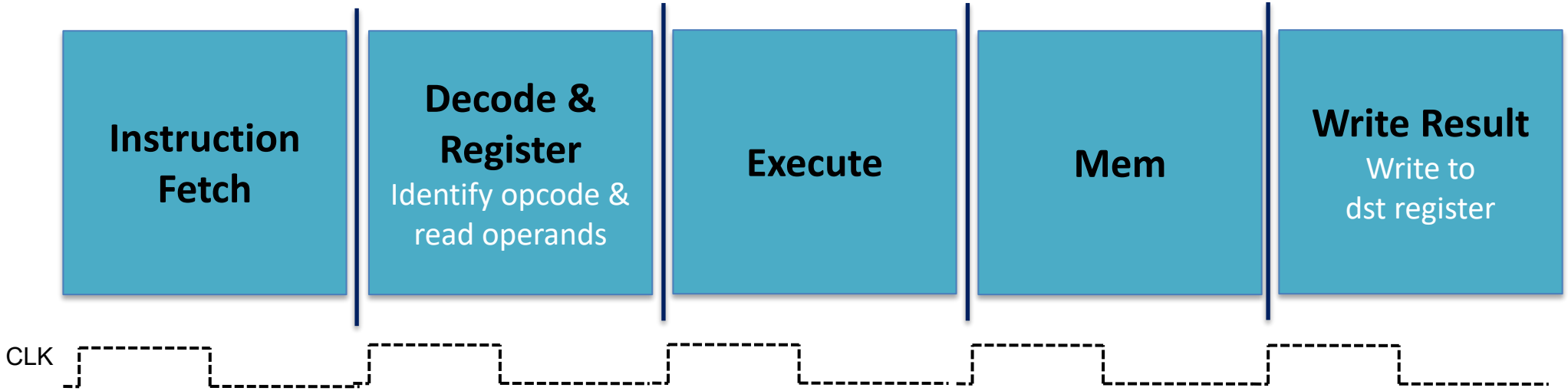
MIPS Pipeline

- **5 steps in one cycle for single-cycle CPU design**
  - With each step using separate resources



MIPS Pipeline

- **Entire datapath can be broken into five stages**
- **Cycle period is changed to the time taken for a stage**
  - Shorter clock cycle
  - Higher CPI (e.g., CPI = 5 in MIPS)



MIPS Pipeline

- **Entire datapath can be broken into five stages**
- **Cycle period is changed to the time taken for a step**
  - Shorter clock cycle
  - Higher CPI (e.g., CPI = 5 in MIPS)
  - System can achieve **CPI = 1** by overlapping Multiple Instructions

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11
Inst1	IF	ID	EXE	MEM	WB						
Inst2		IF	ID	EXE	MEM	WB					
Inst3			IF	ID	EXE	MEM	WB				
Inst4				IF	ID	EXE	MEM	WB			
Inst5					IF	ID	EXE	MEM	WB		

cc: clock cycle

Initial pipeline filling phase

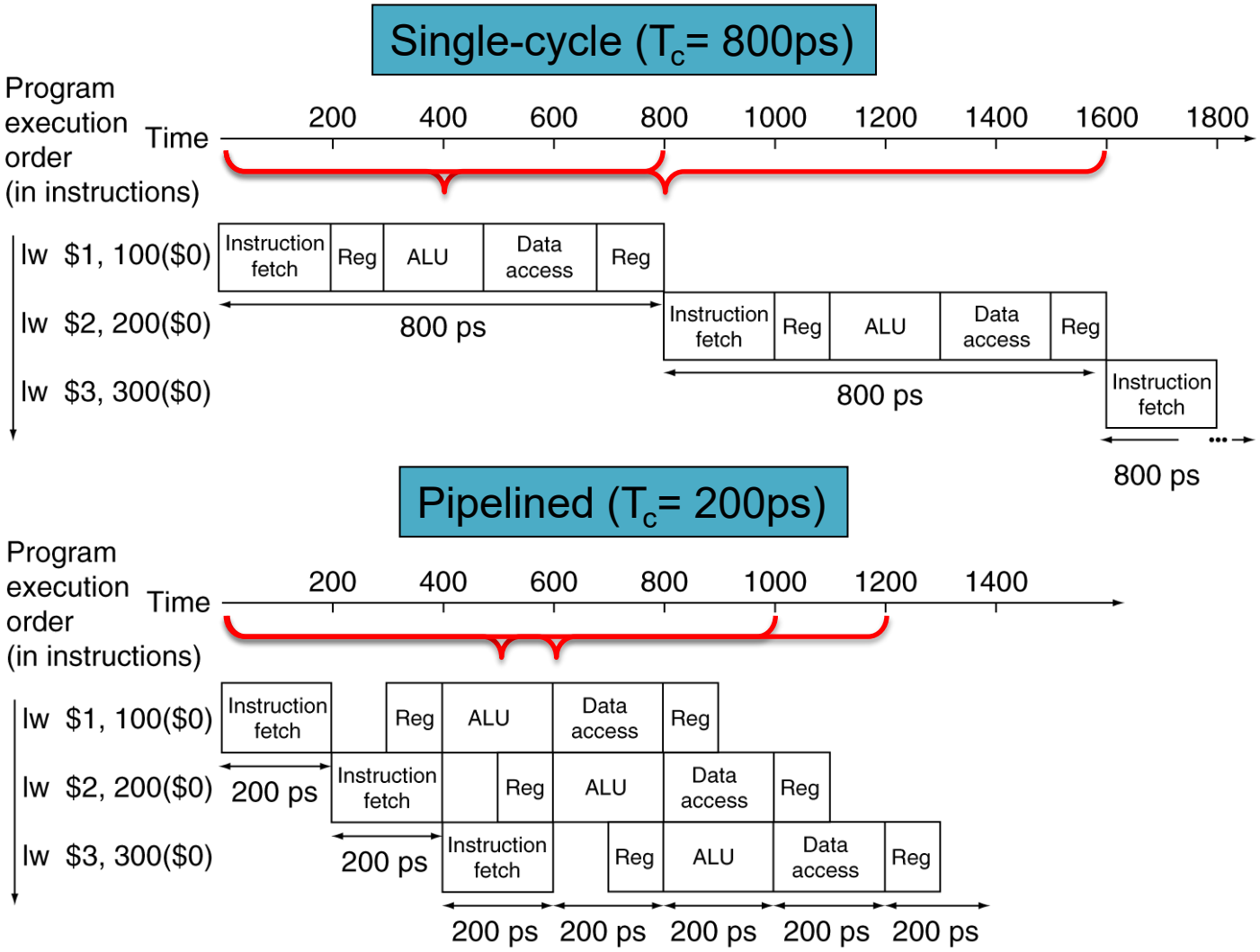
After initial pipeline filling phase, we can finish an instruction every cycle

Pipeline Performance Example

- **100ps for register IOs (i.e., decode, writeback); 200ps for other stages**
- **What would be the cycle period in single-cycle and pipelined datapath?**
- **What would be the execution time of lw instruction?**

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-type	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Pipeline Performance Example



T_c = Longest instruction

T_{lw} = 800ps

2 x T_{lw} = 1600ps

...

T_c = Longest Stage

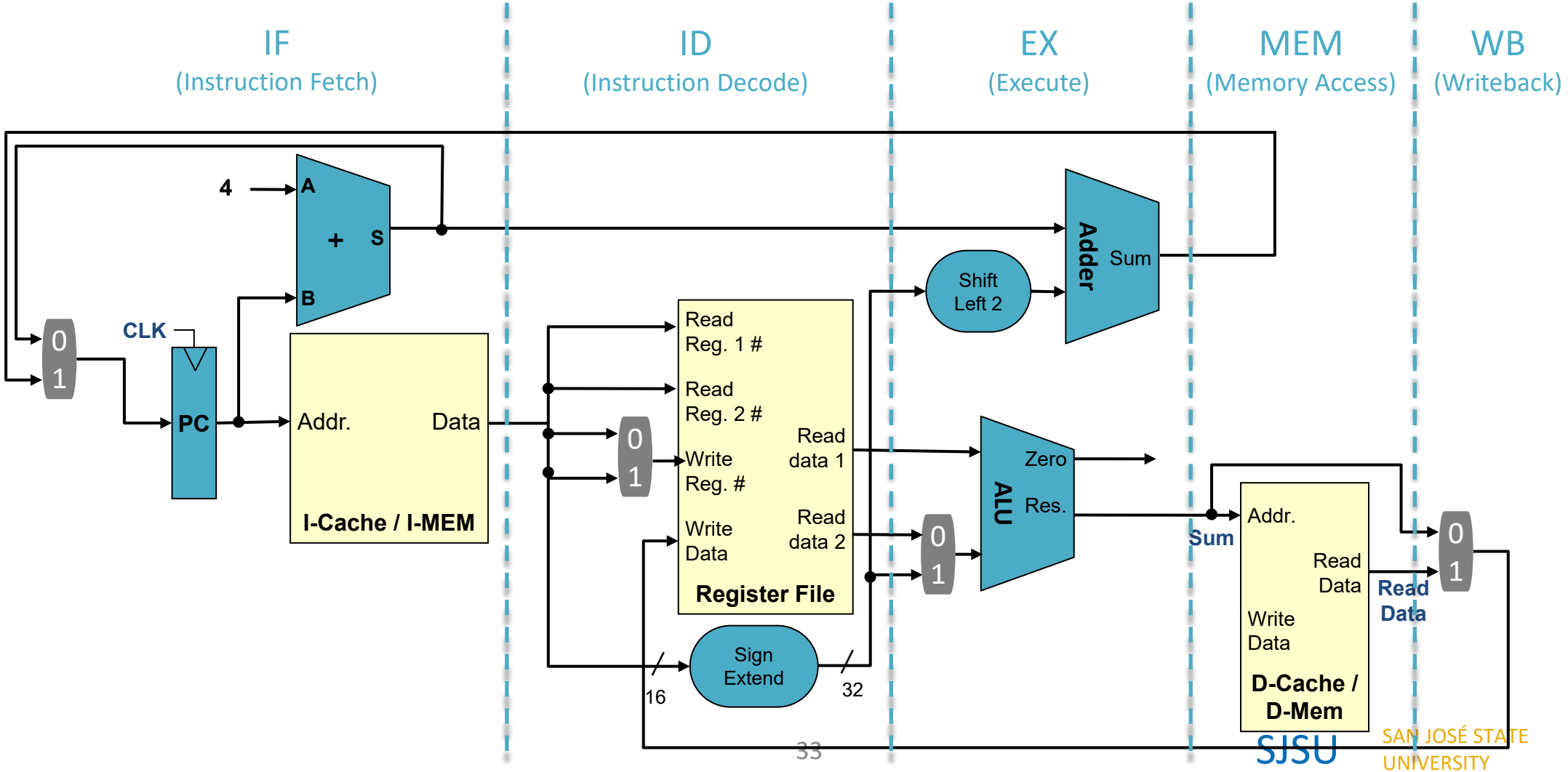
T_{lw} = 1000ps

2 x T_{lw} = 1200ps

...

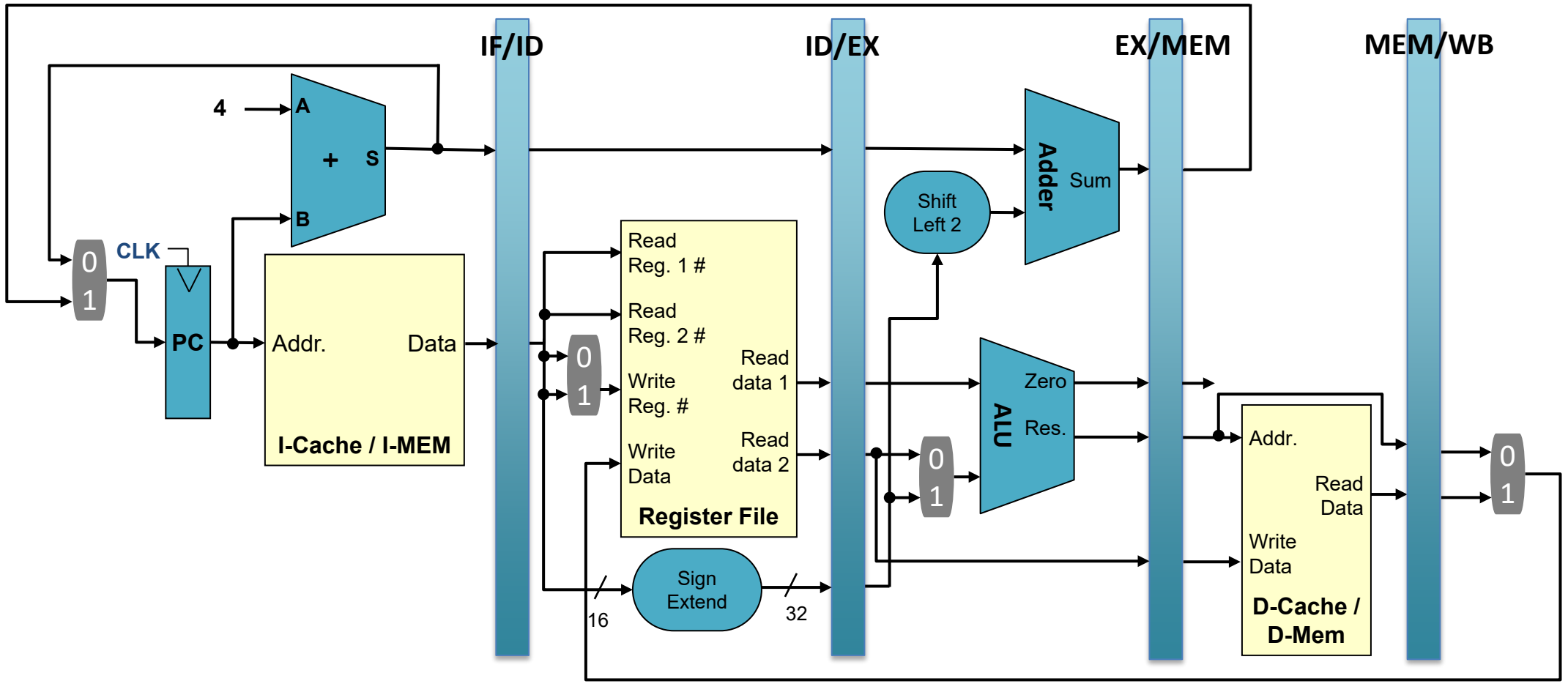
Basic 5 Stage Pipeline

- **Same structure as single cycle but now broken into 5 stages**

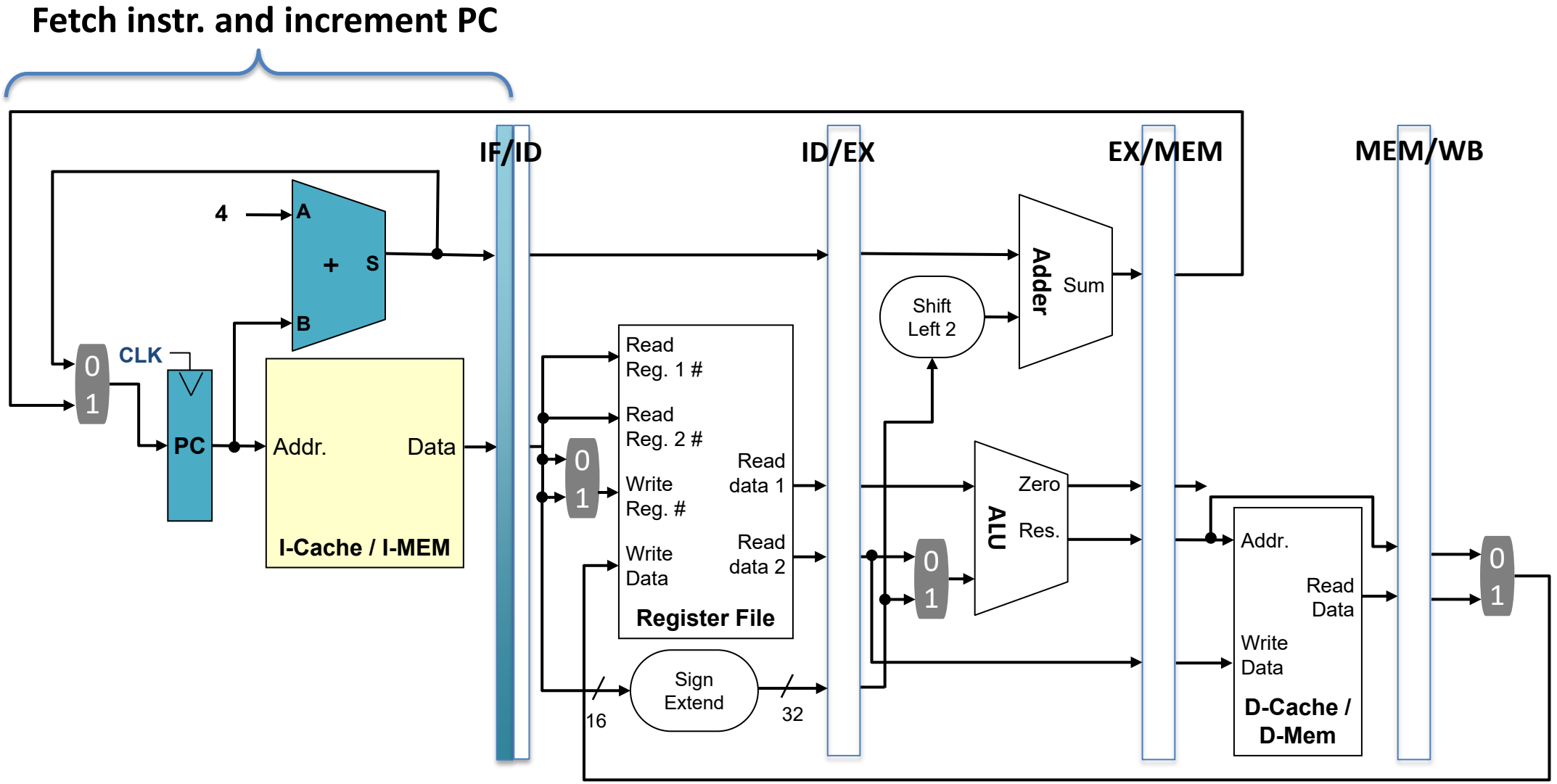


Pipeline Stage Registers

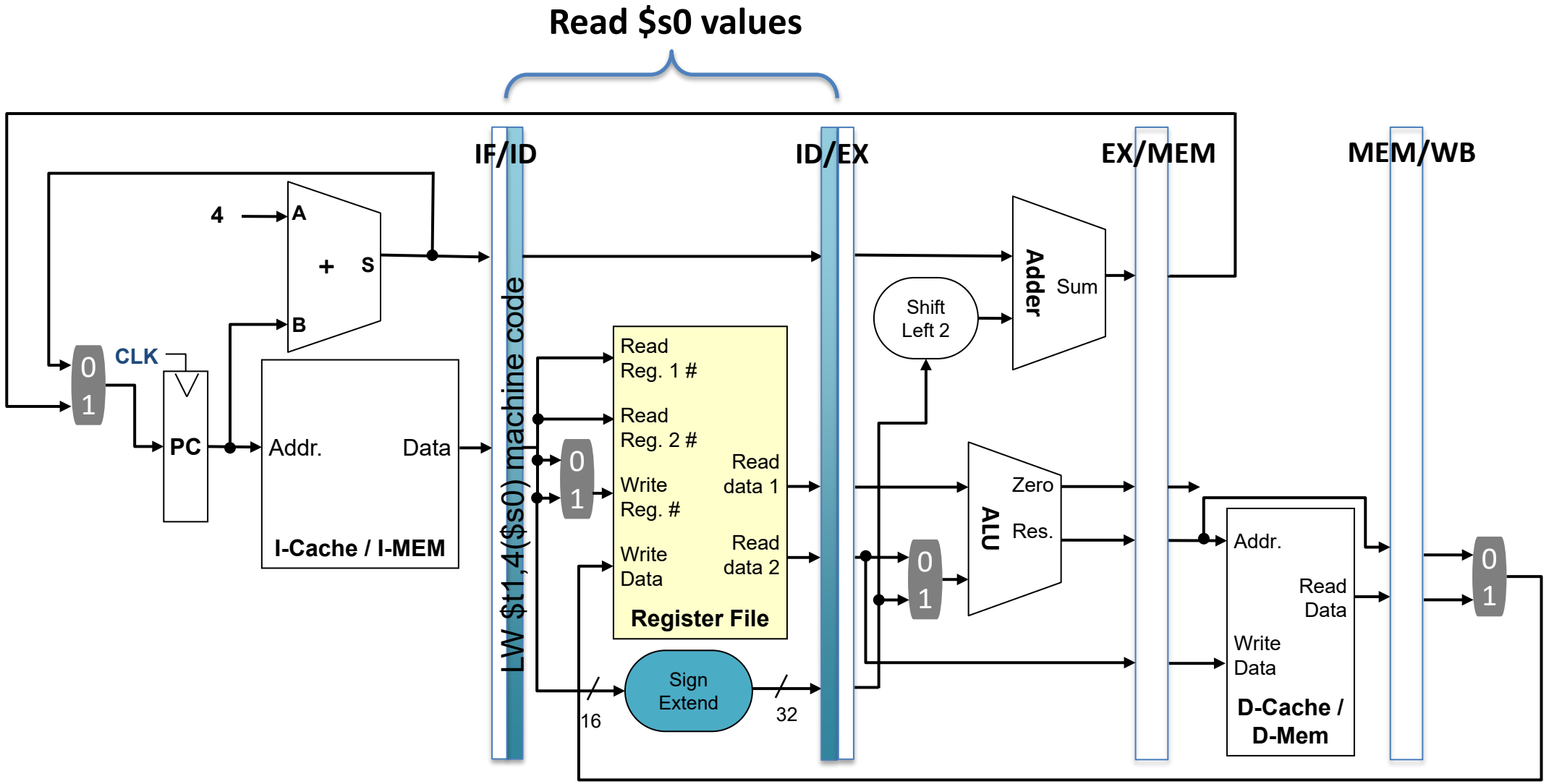
- **Intermediate results need to be stored to allow stage reuse**



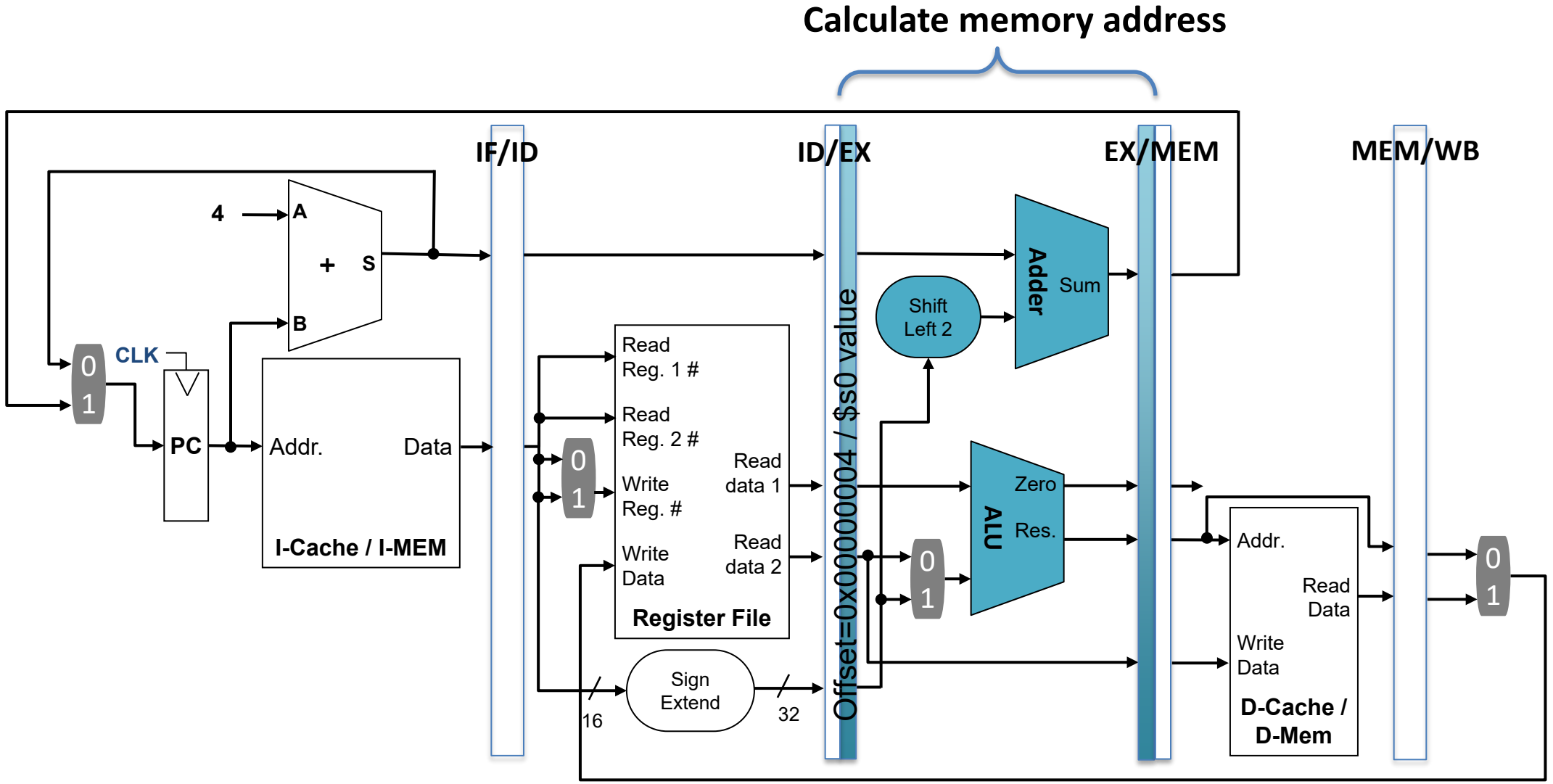
Pipeline Example: LW $t1, 4($s0)



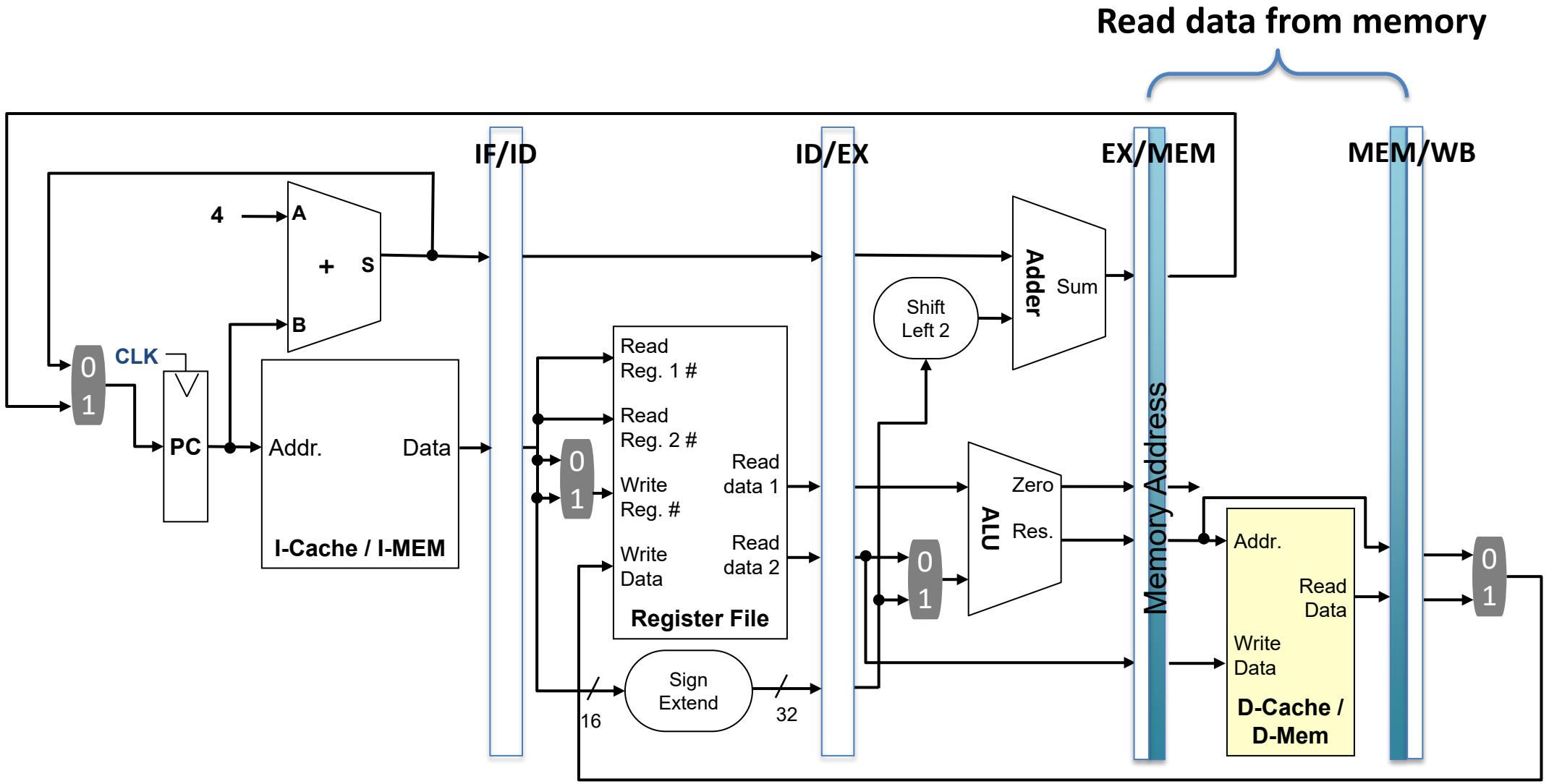
Pipeline Example: LW $t1, 4($s0)



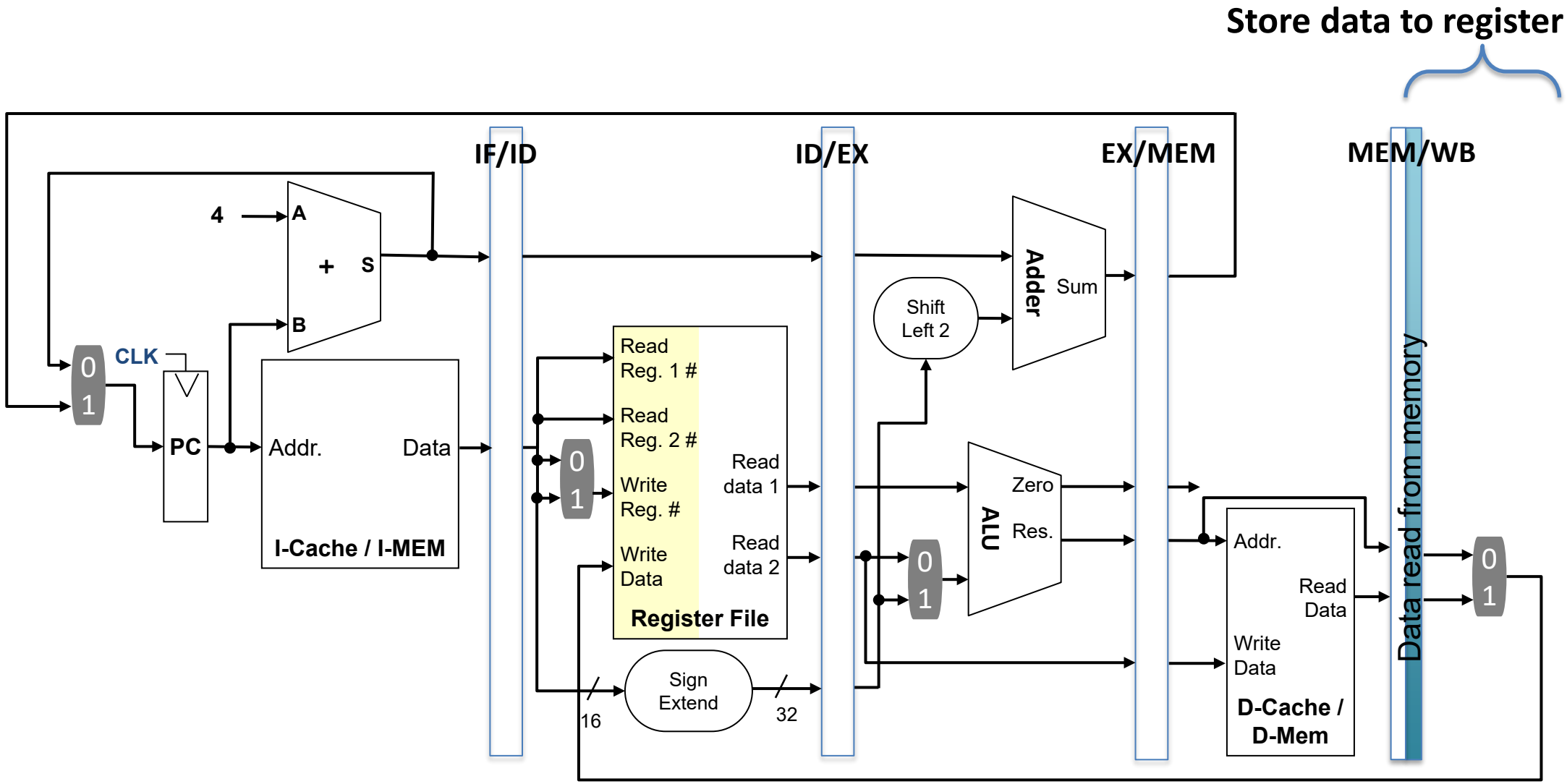
Pipeline Example: LW $t1, 4($s0)



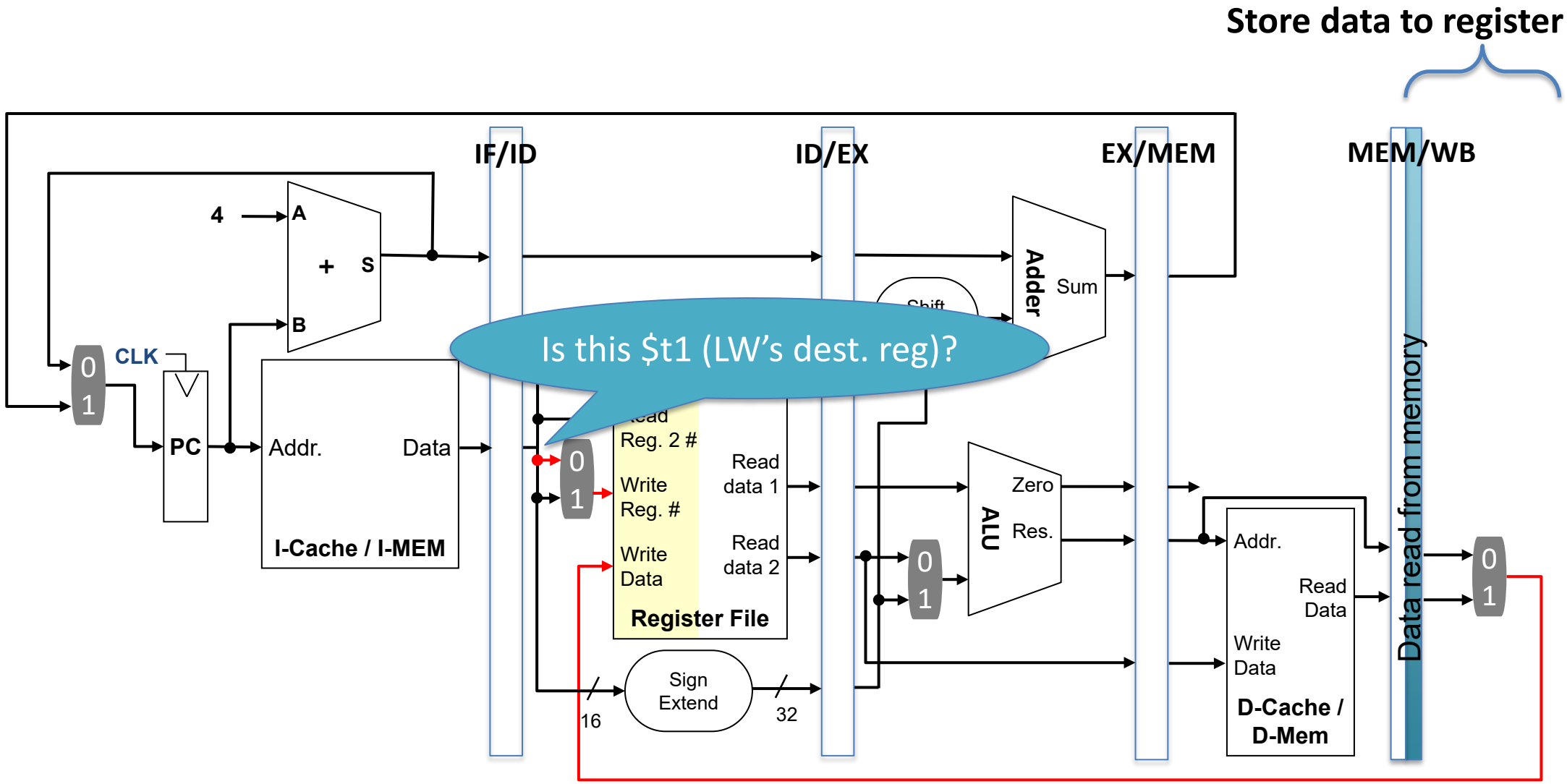
Pipeline Example: LW $t1, 4($s0)



Pipeline Example: LW $t1, 4($s0)

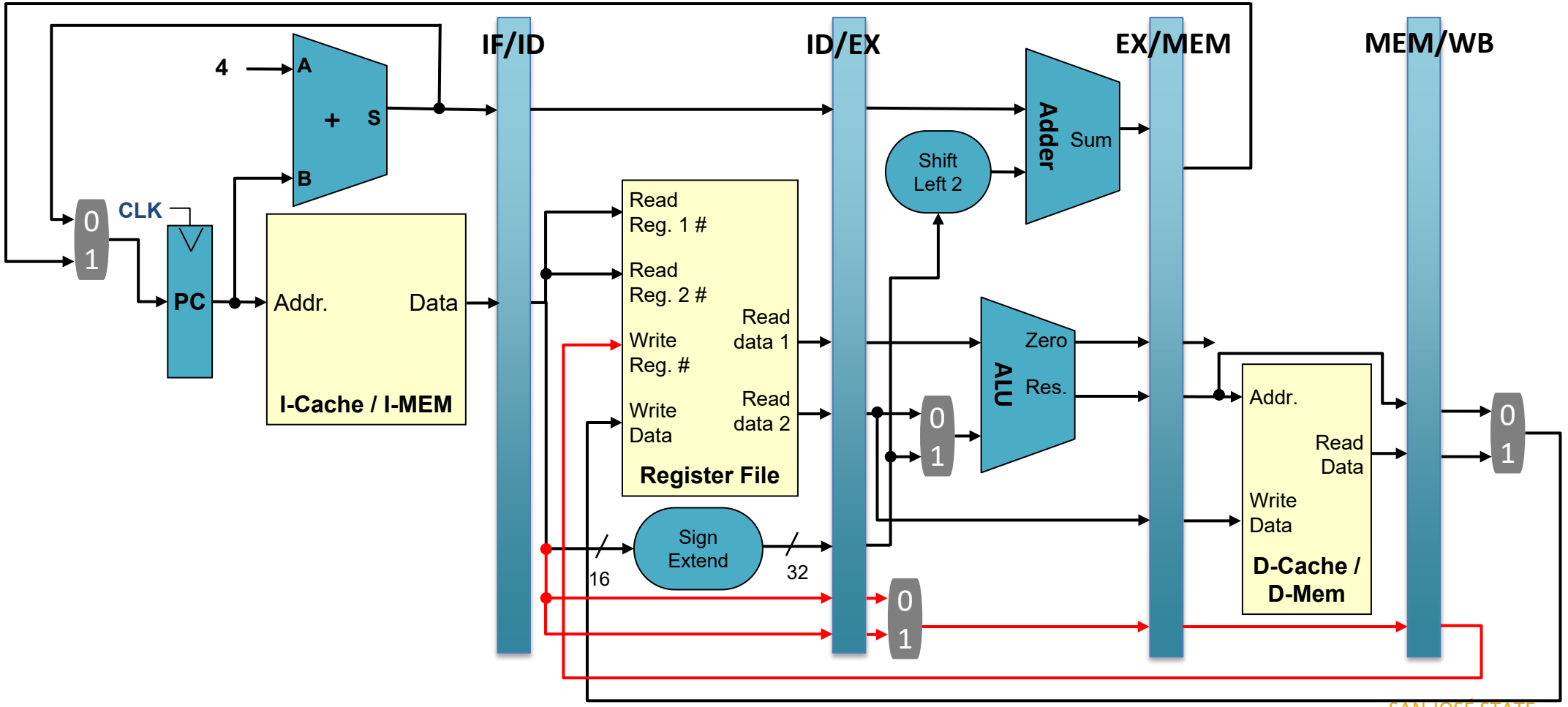


Pipeline Example: LW $t1, 4($s0)



Revised Datapath

- Dest. register number should be stored in the pipeline registers
- The dest. register number is passed together with data during writeback



Conclusion Time

What are the two components of the CPU control unit?

- Main decoder & ALU decoder

Why are both J and Branch instructions based on the next PC?

- Add 4 is already applied

Do we have subi in MIPS? Why?

- Not needed and complex

Conclusion Time

What is the main issue of single-cycle CPU?

- **Long clock period**

What is the key idea behind pipelined CPUs?

- **Parallelism**

SAN JOSÉ STATE UNIVERSITY *powering* SILICON VALLEY