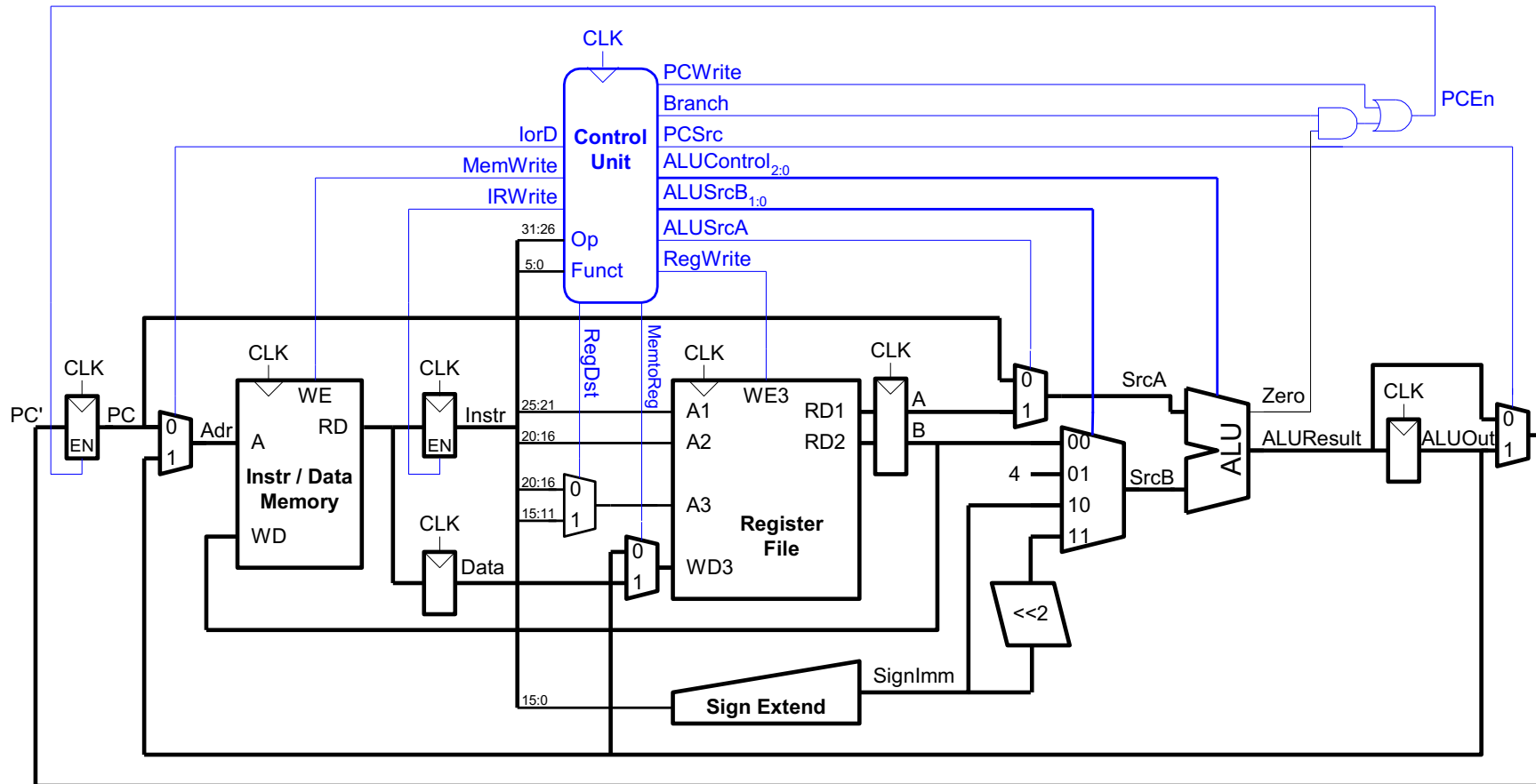


# ***MIPS microarchitecture***

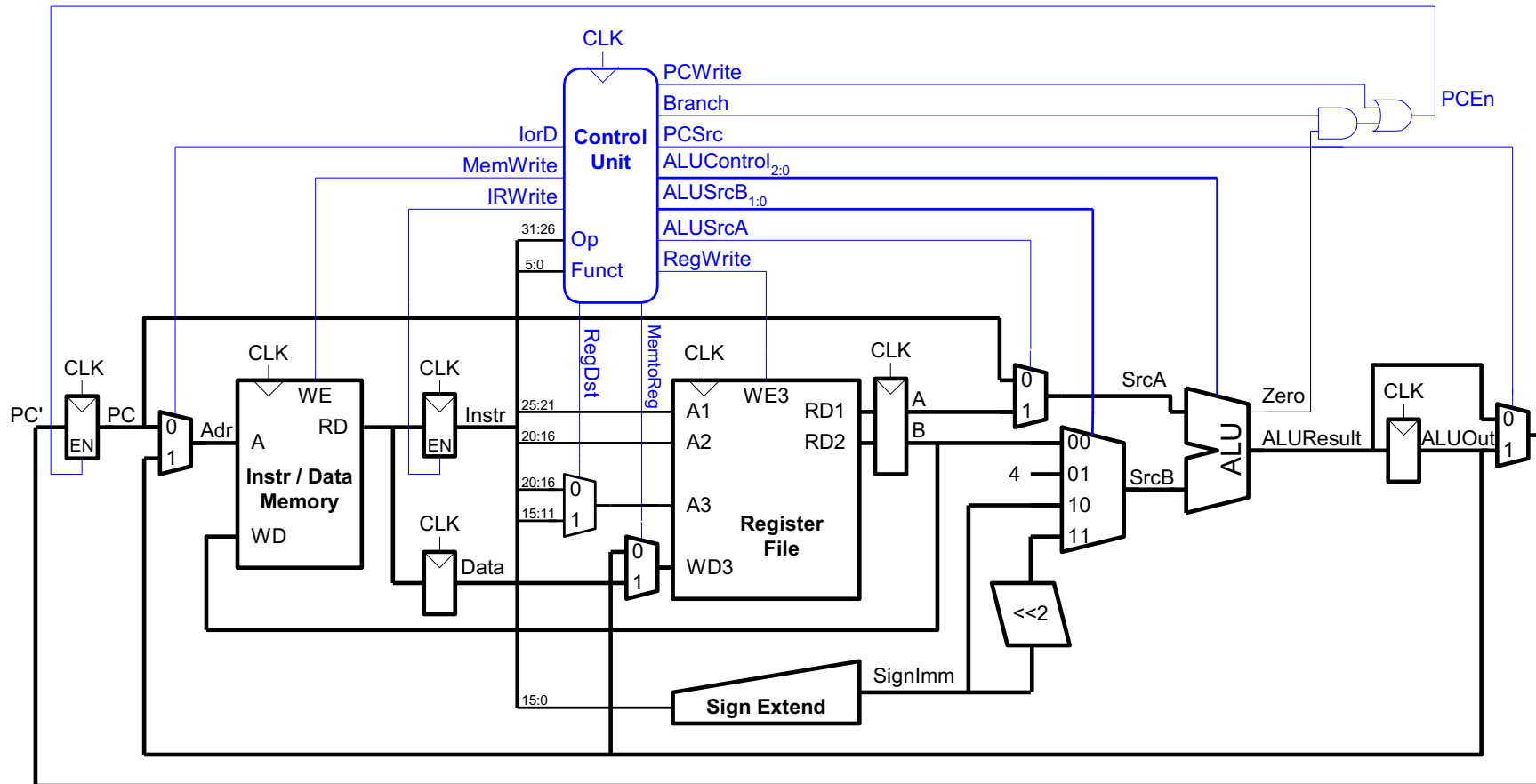
---

These transparencies are based on those provided with the following book:  
David Money Harris and Sarah L. Harris, “Digital Design and Computer Architecture”,  
2nd Edition, 2012, Elsevier

# Multicycle Processor Architecture



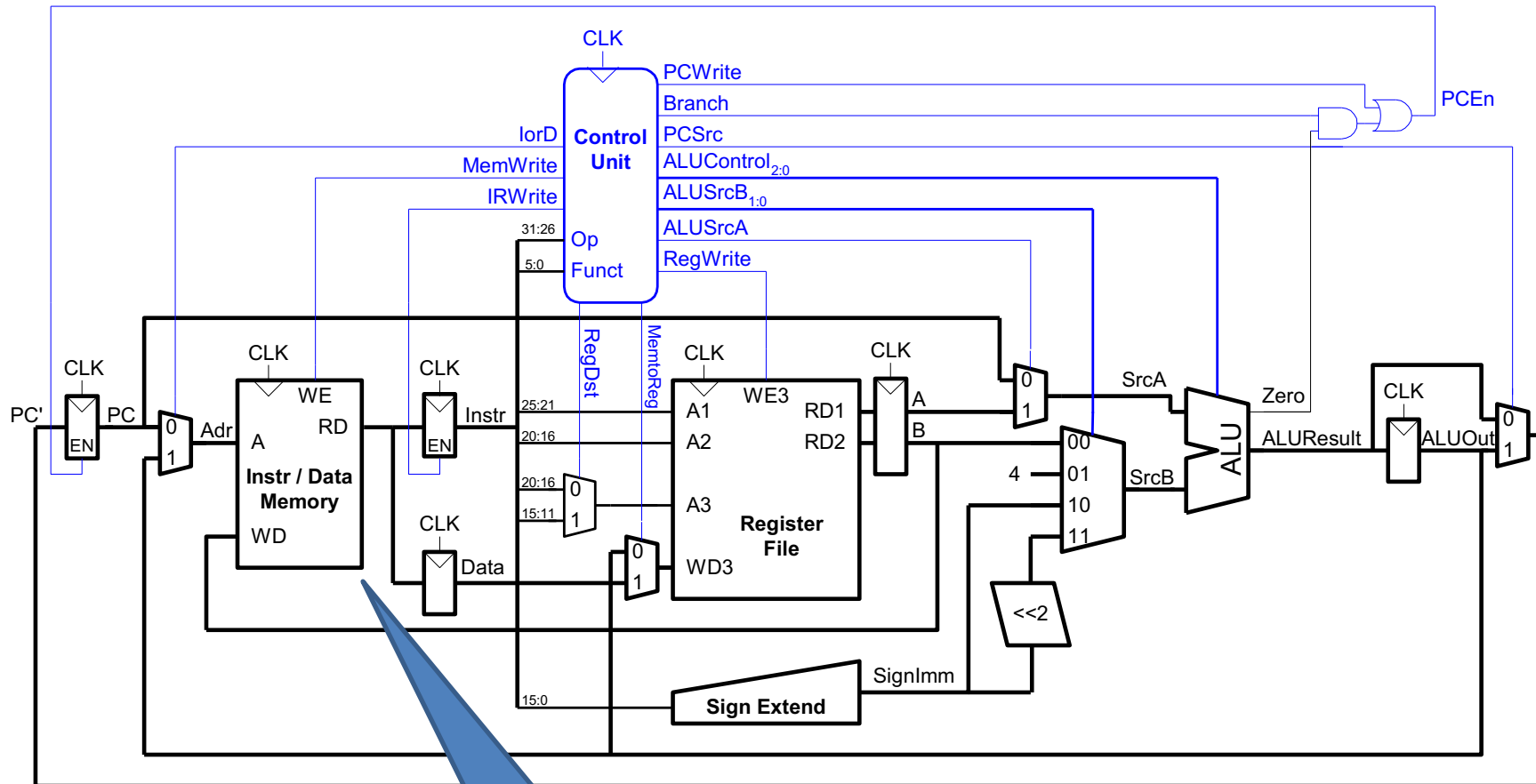
# Multicycle Processor Architecture



The processor is organized in 2 parts:

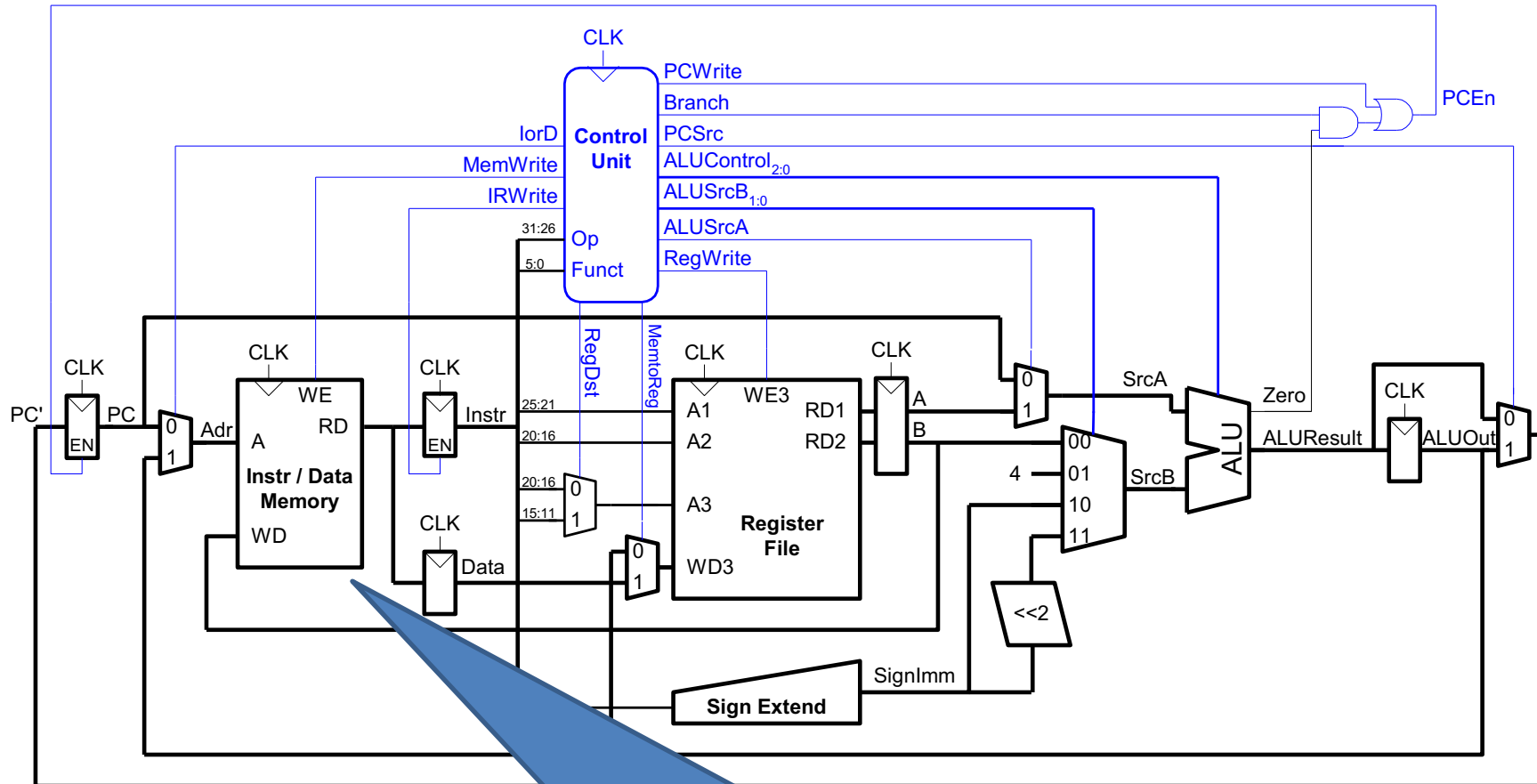
- Data Path (in black)
- Control Unit (in blue)

# Multicycle Processor Architecture



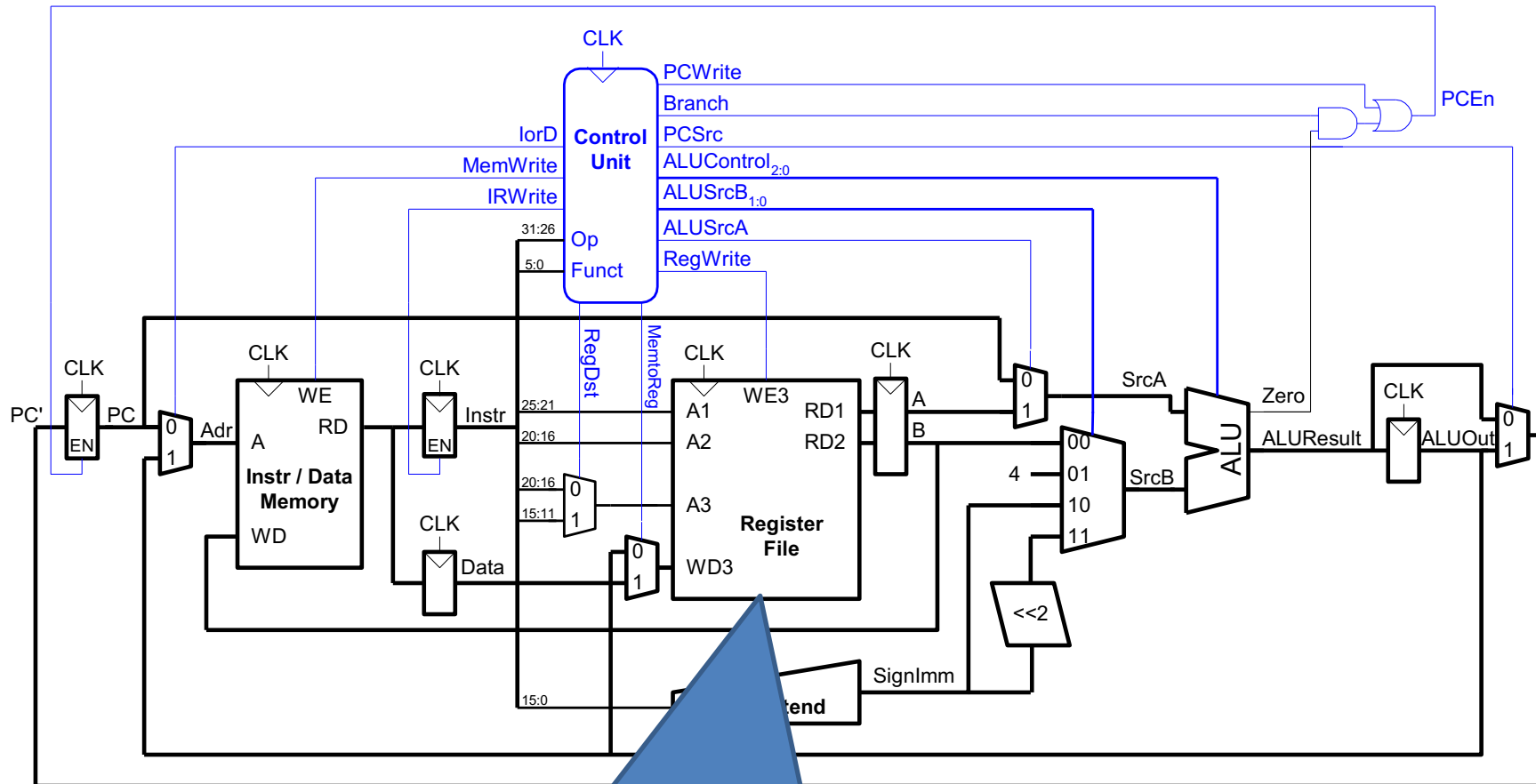
Memory is NOT  
part of the  
processor

# Multicycle Processor Architecture



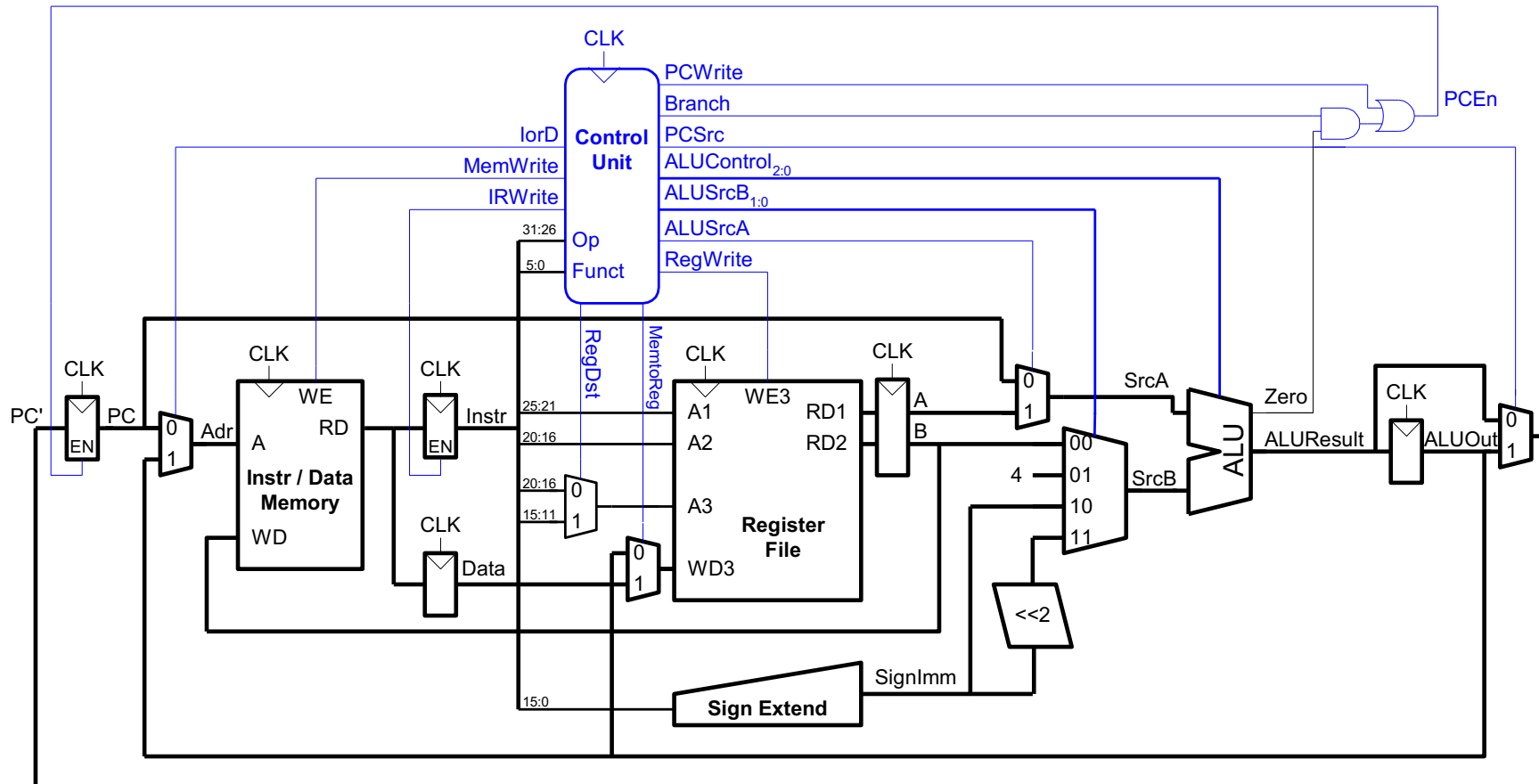
The memory works at each clock cycle, executing a read or write operation, depending on WE.

# Multicycle Processor Architecture



The RF works at each clock cycle, executing a read of two values or a write operation, depending on WE3.

# Multicycle Processor Architecture



All registers work at each clock cycle, storing the value on their inputs, if not disabled.

# MIPS Processor

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

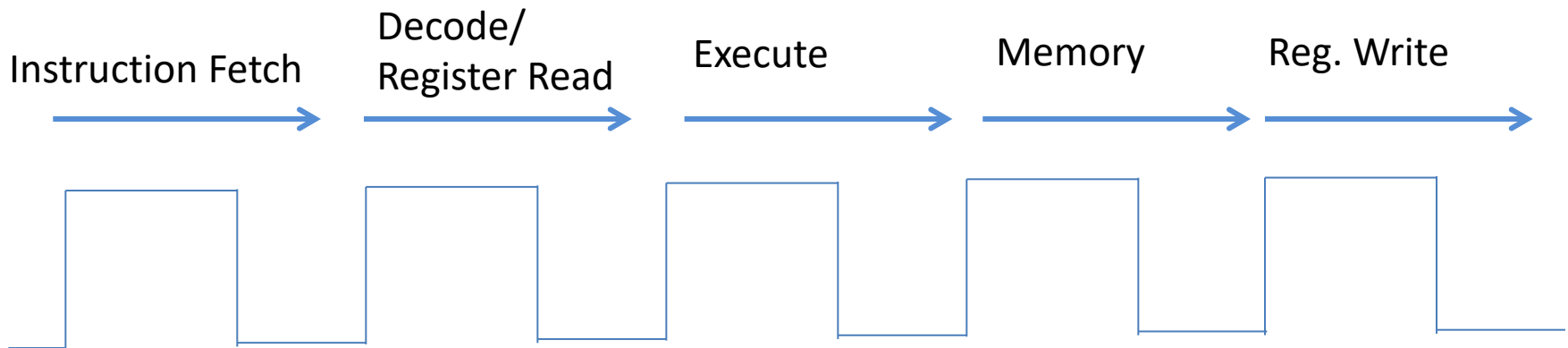


# MIPS Processor - New

- Multiple implementations for a single architecture
- Single-cycle
  - Each instruction executes in a single clock cycle
- Multicycle
  - Each instruction is broken up into a series of shorter steps with one step per clock cycle
- Pipelined (a variant of “multicycle”):
  - Each instruction is broken up into a series of steps with one step per clock cycle
  - Multiple instructions execute at once

# MIPS Processor Multicycle

- Multiple-cycle CPU
  - Only one stage of instruction per clock cycle
  - The clock is made as long as the slowest stage

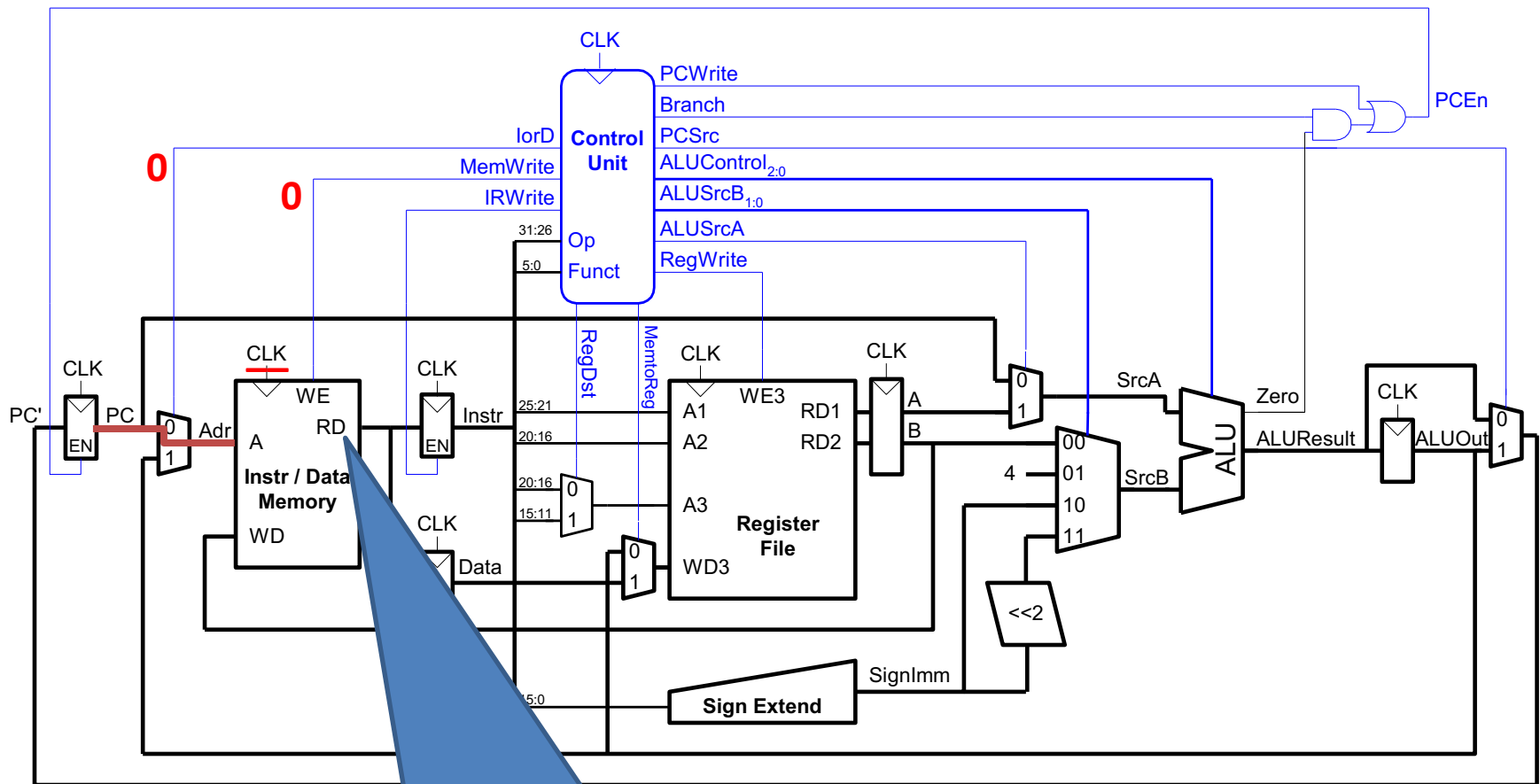


# MIPS State Elements

- Determines everything about the execution status of a processor
  - PC register
  - Instruction and Data memory
  - Instruction and Data registers
  - Register file
  - Operand registers
  - Result register
- For these state elements, clock is used for write but not for read
  - Asynchronous read, Synchronous write

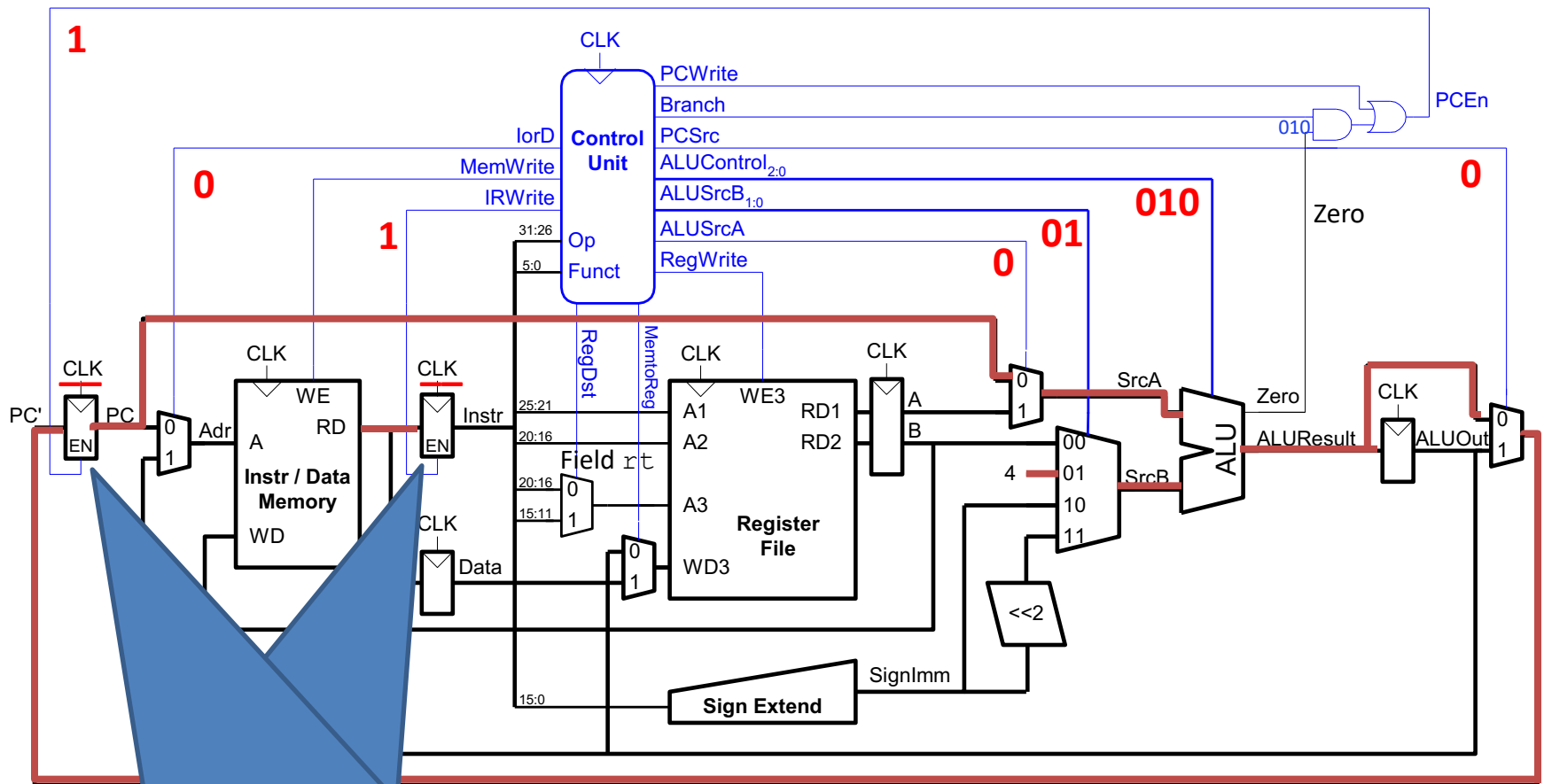
# Multicycle Datapath: Instruction Fetch

## STEP 1: Send address to memory



CLK: writing data on RD.

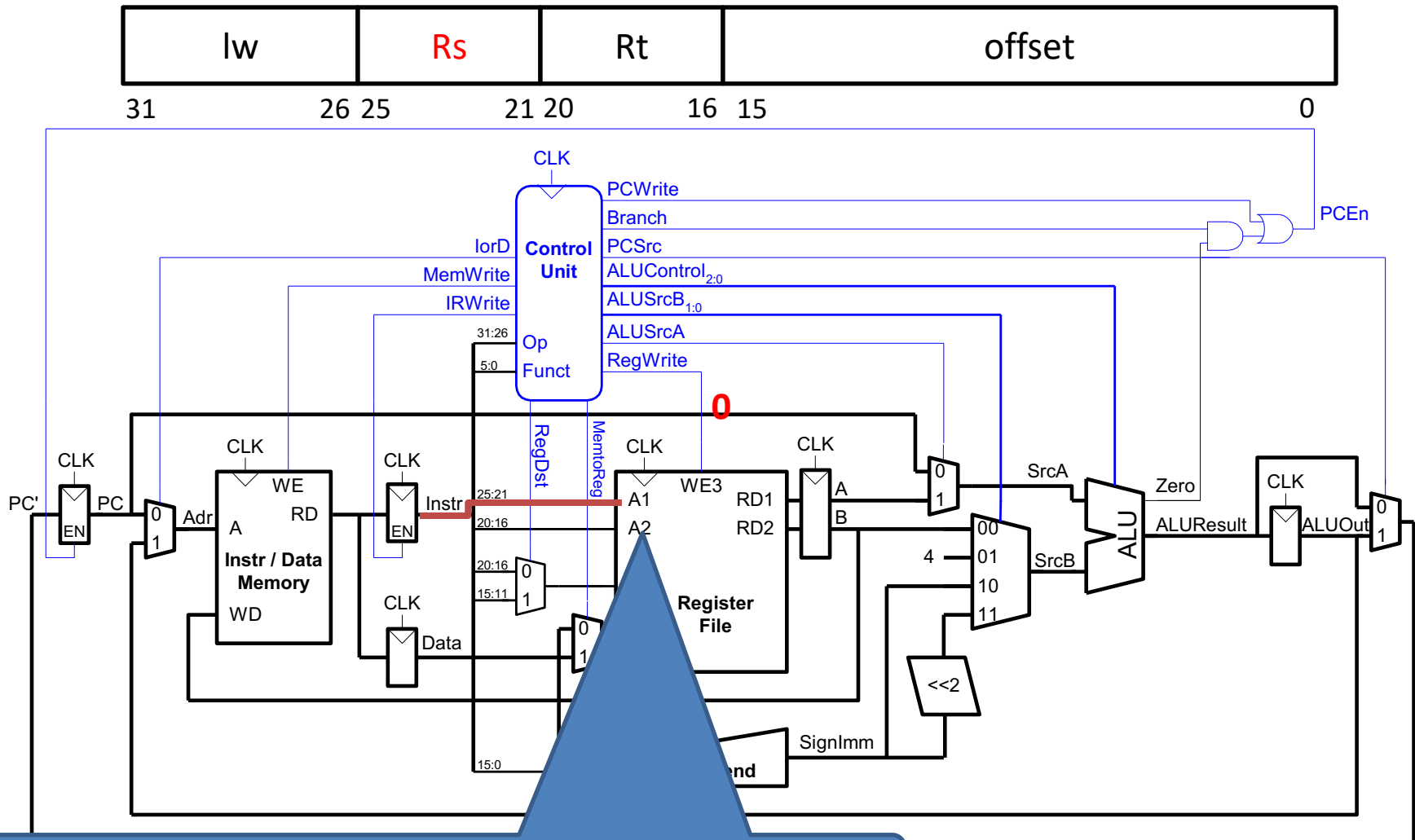
## STEP 2: Write instruction into IR and update PC



CLK: writing RD on IR. Update PC

# Multicycle Datapath: $lw$

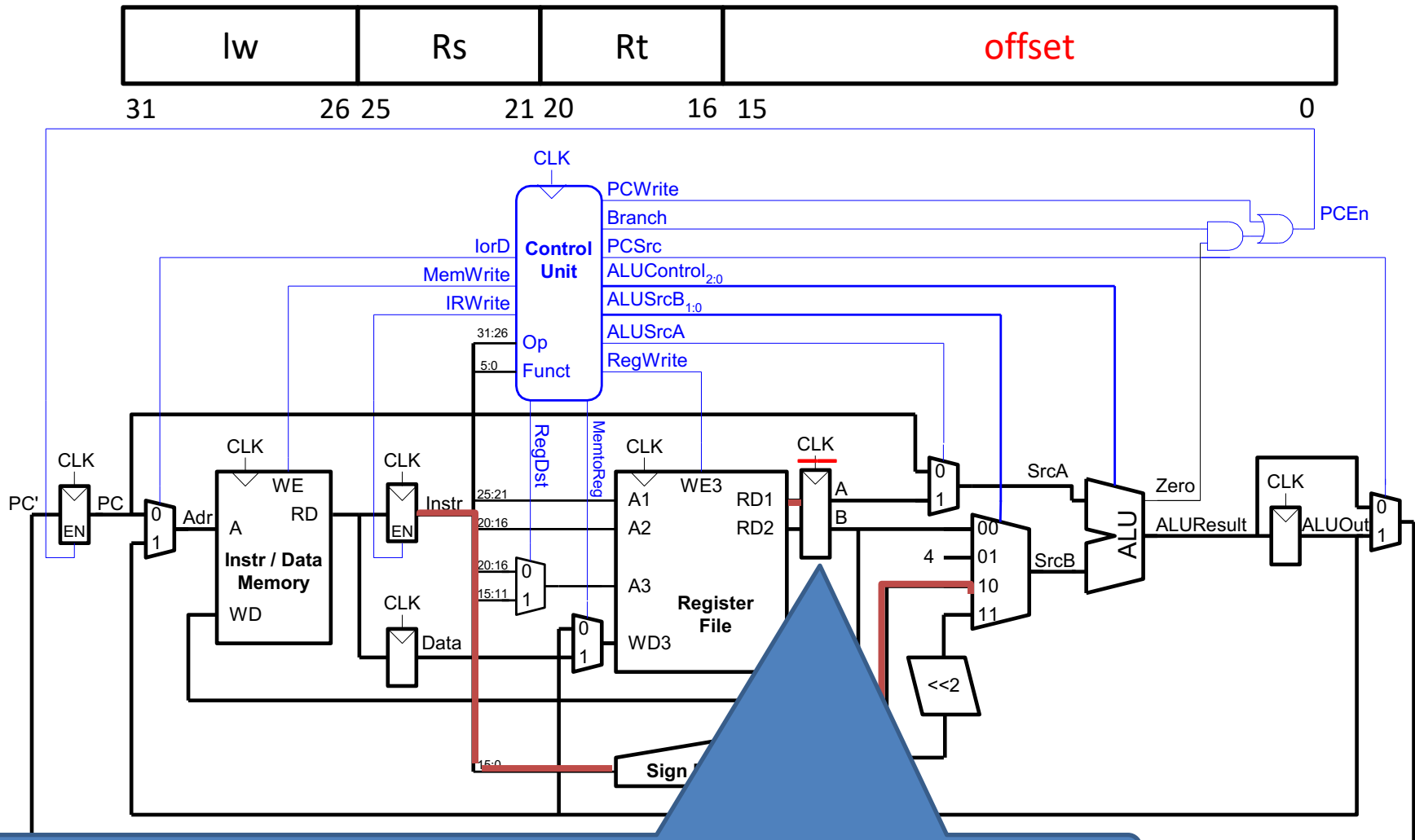
## STEP 3a: Read $rs$ from Register File



Asynchronous read of the register file

# Multicycle Datapath: $lw$

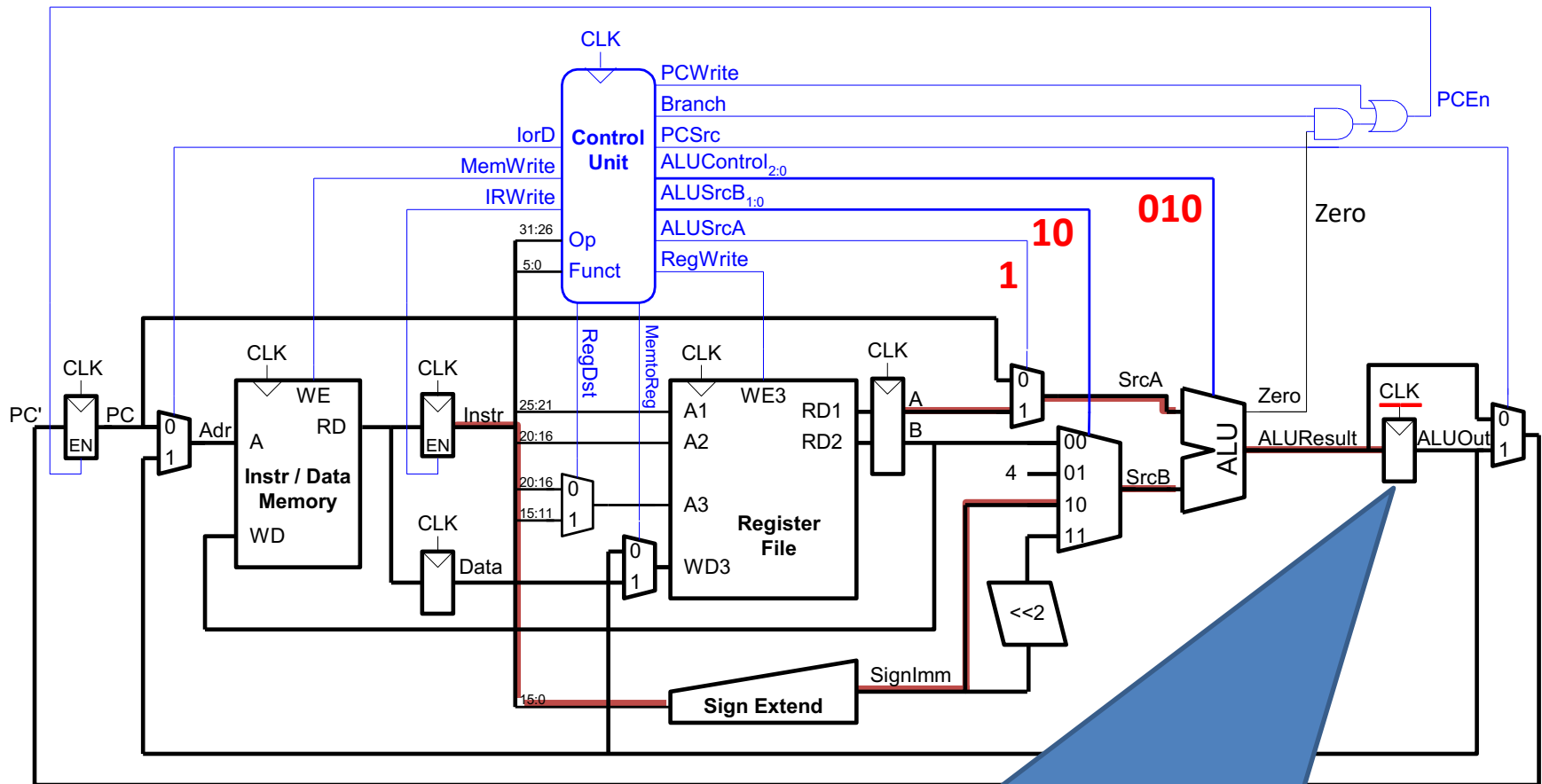
## STEP 3b: Sign-extend the immediate



CLK: writing RD1 into the operand register A

# Multicycle Datapath: $1_w$

## STEP 4: Compute the memory address

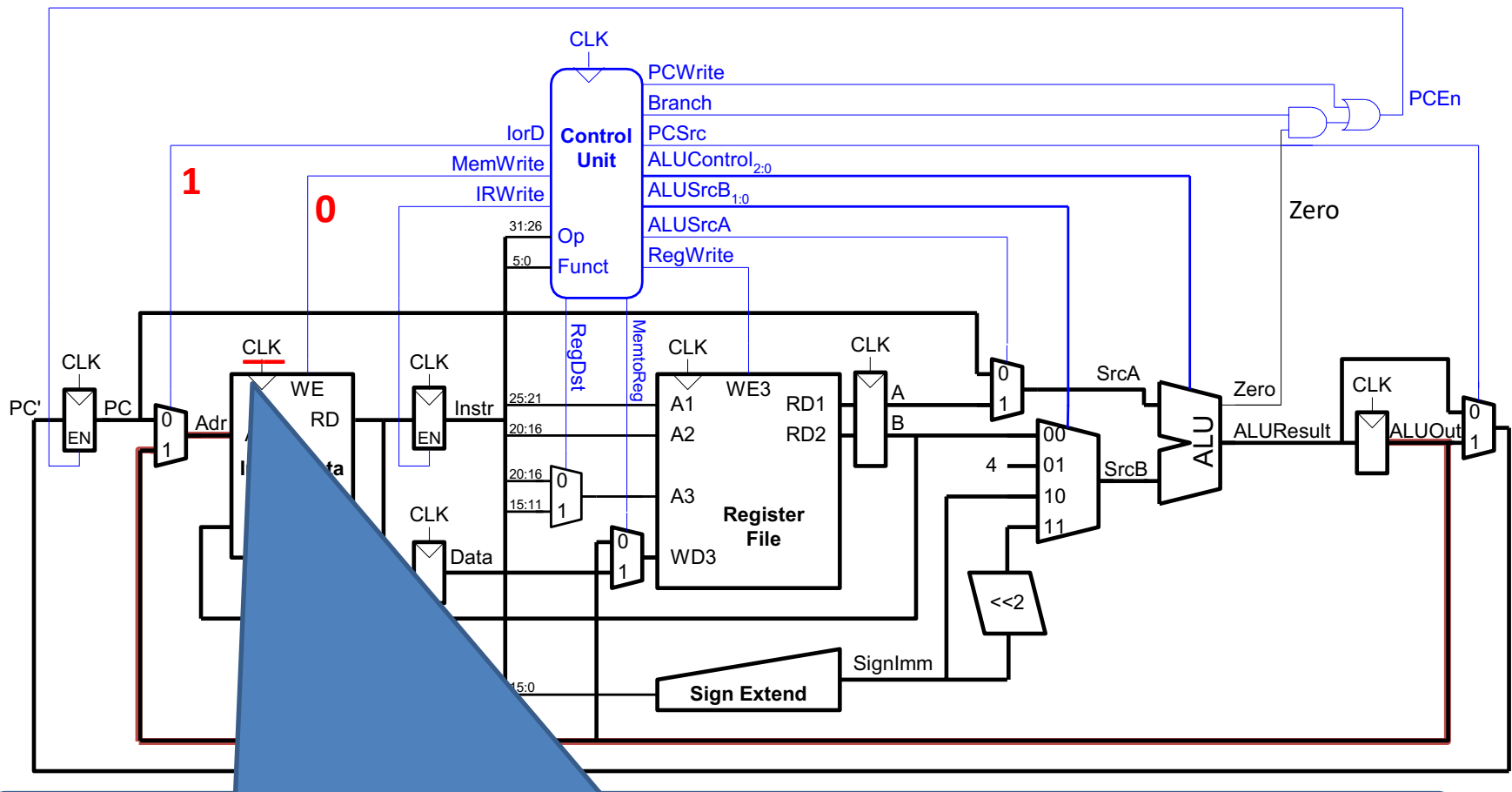


$CLK$ : writing into the result register



# Multicycle Datapath: $1_w$

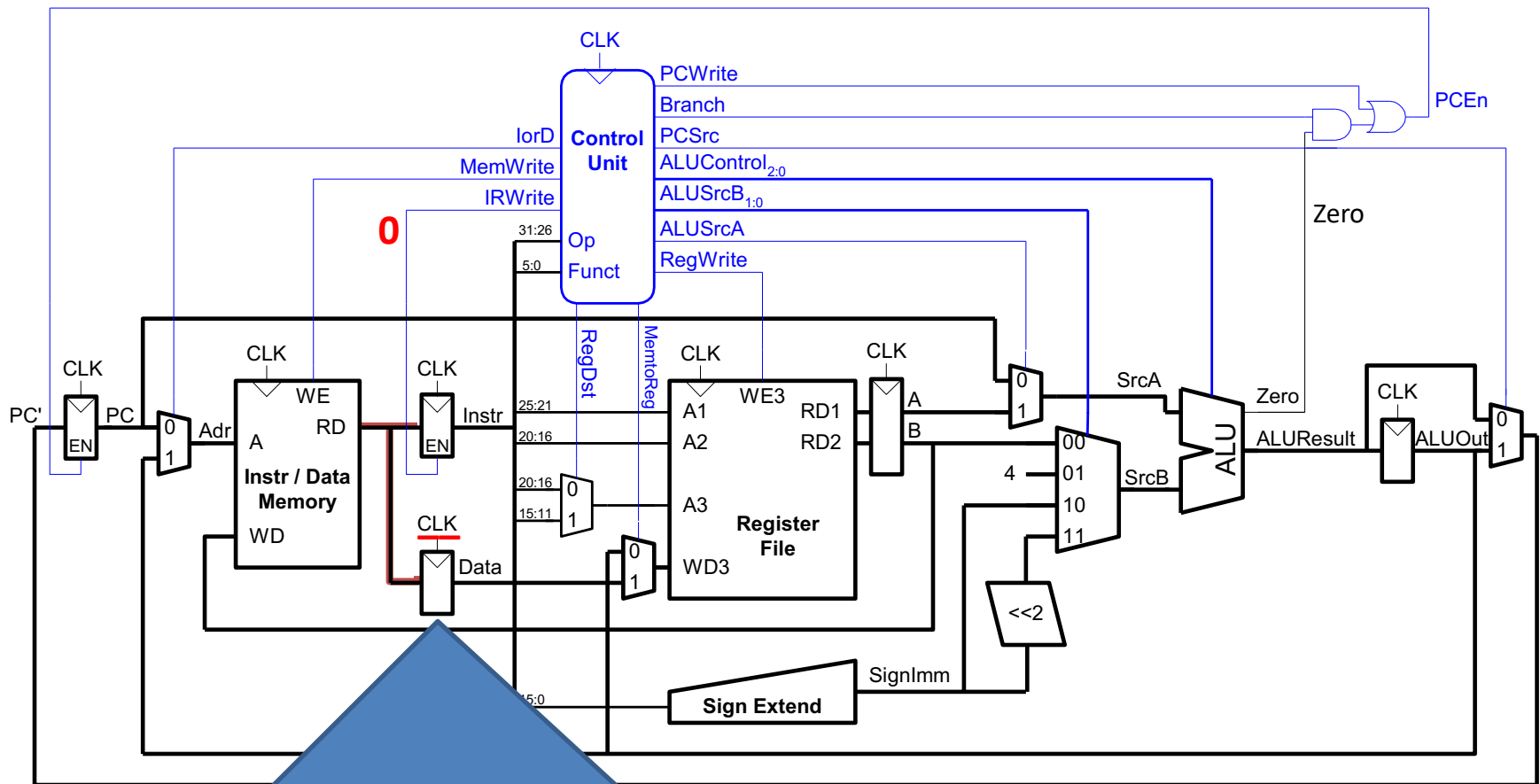
## STEP 5: Send address to memory



$CLK$ : writing the computed address into the memory interface Address Register.

# Multicycle Datapath: $1_W$

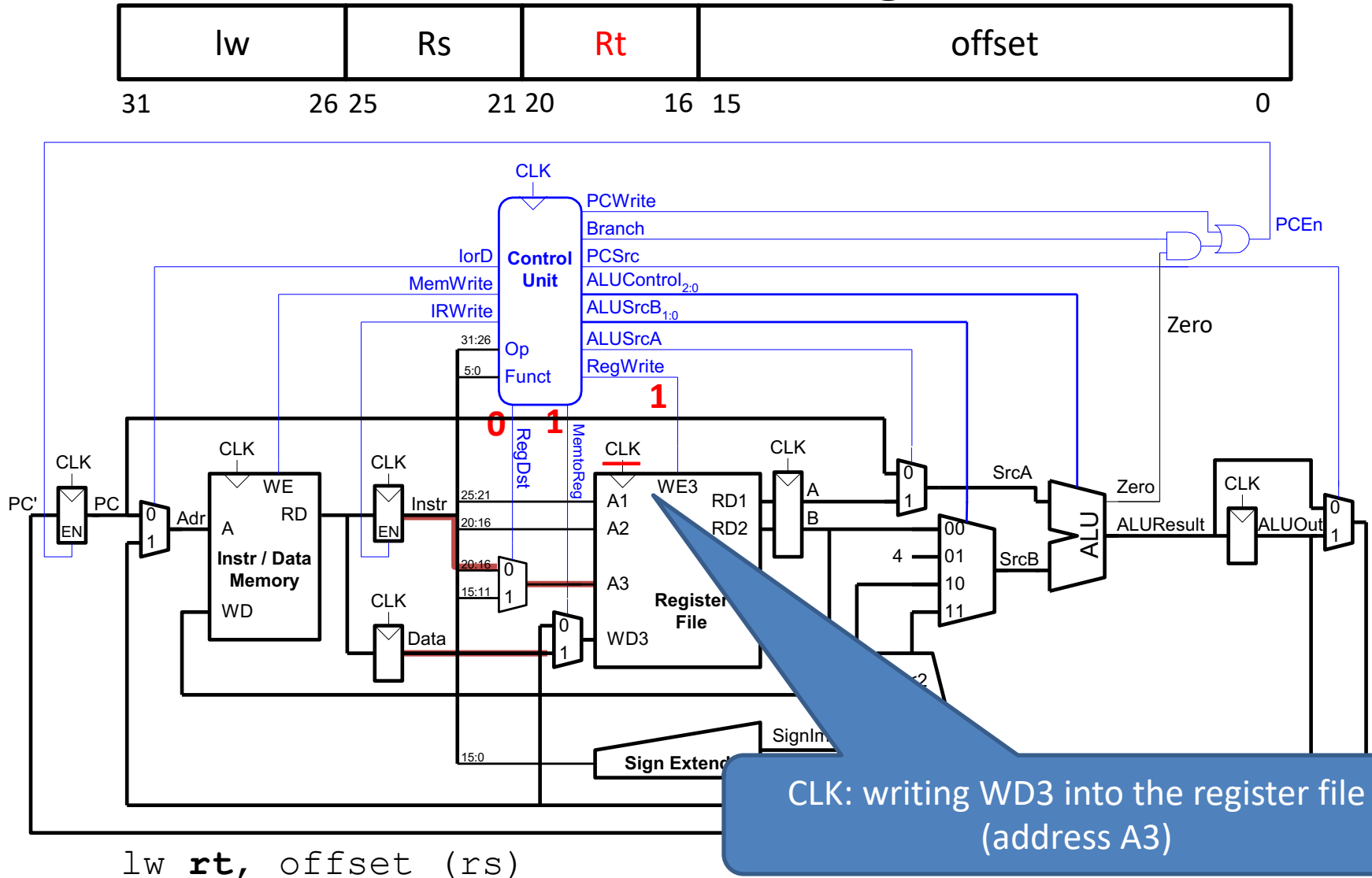
## STEP 6: Read data from memory



CLK: writing the RD value into the Data Register.

# Multicycle Datapath: $\perp_w$

## STEP 7: Write data back to register file



# Multicycle Datapath: $sw$

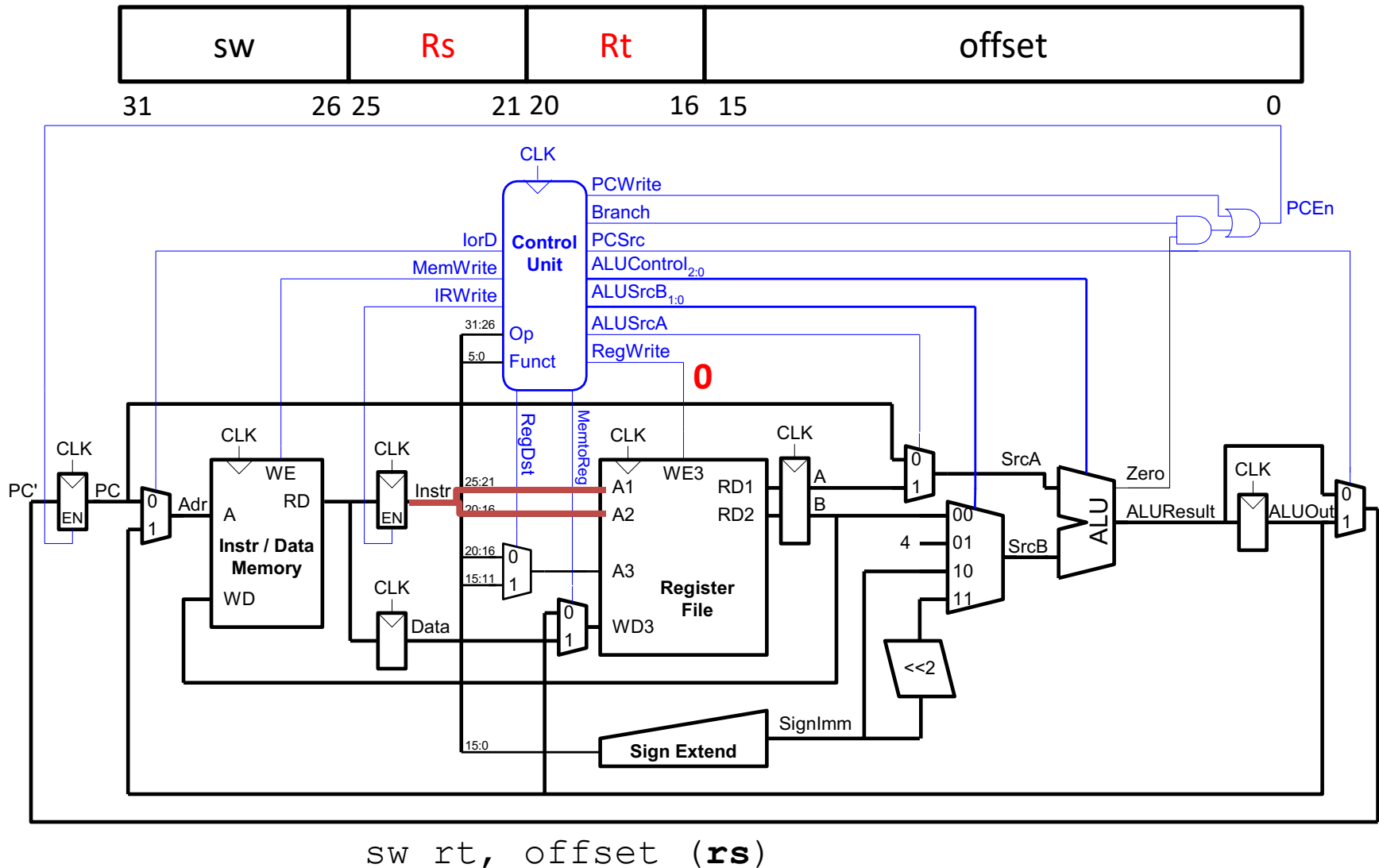
Write data in  $rt$  to memory

**`sw  $rt$ , offset ( $rs$ )`**

Steps 1 and 2 as in previous example

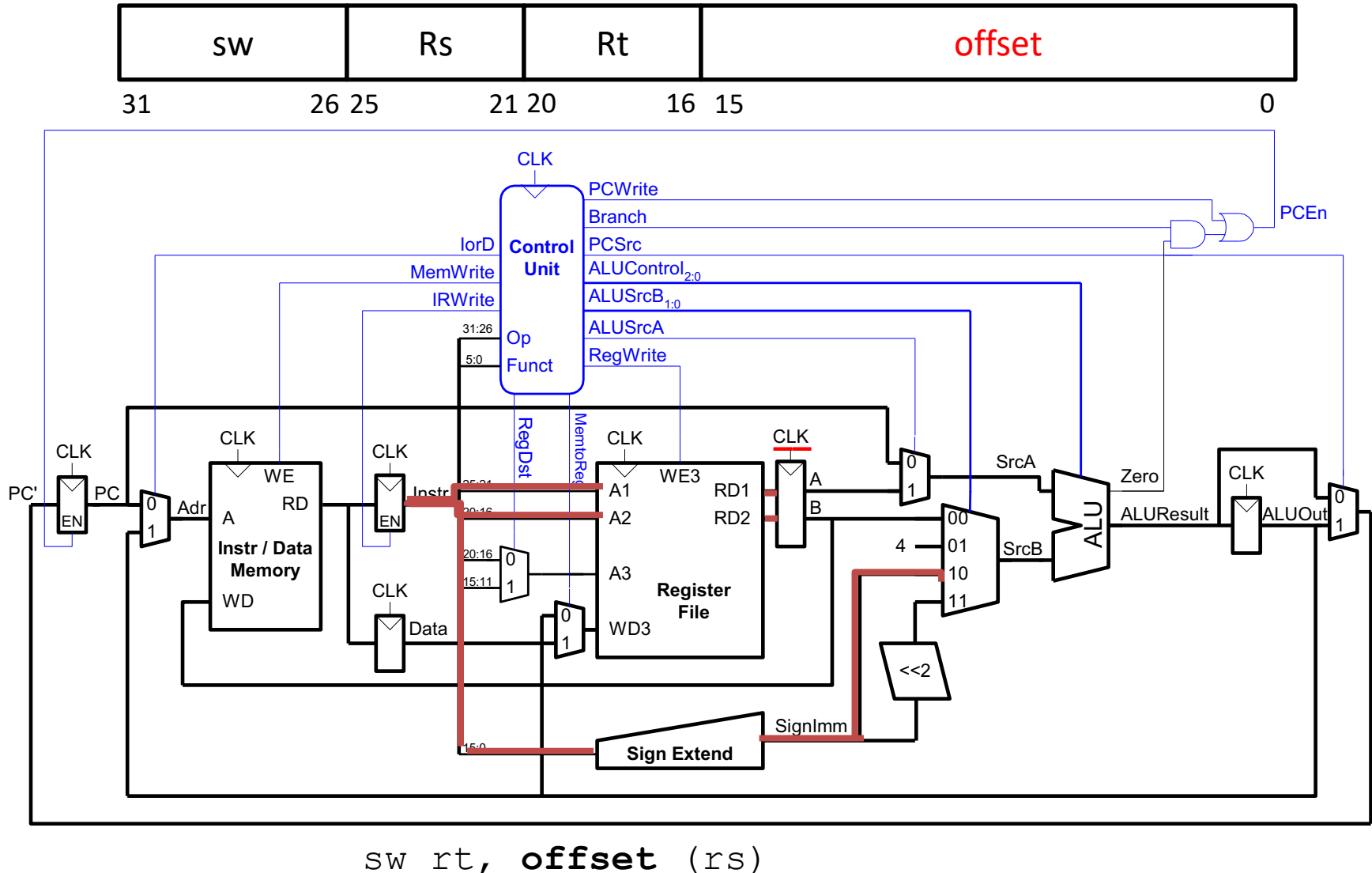
# Multicycle Datapath: SW

## STEP 3a: Read rs and rt from Register File



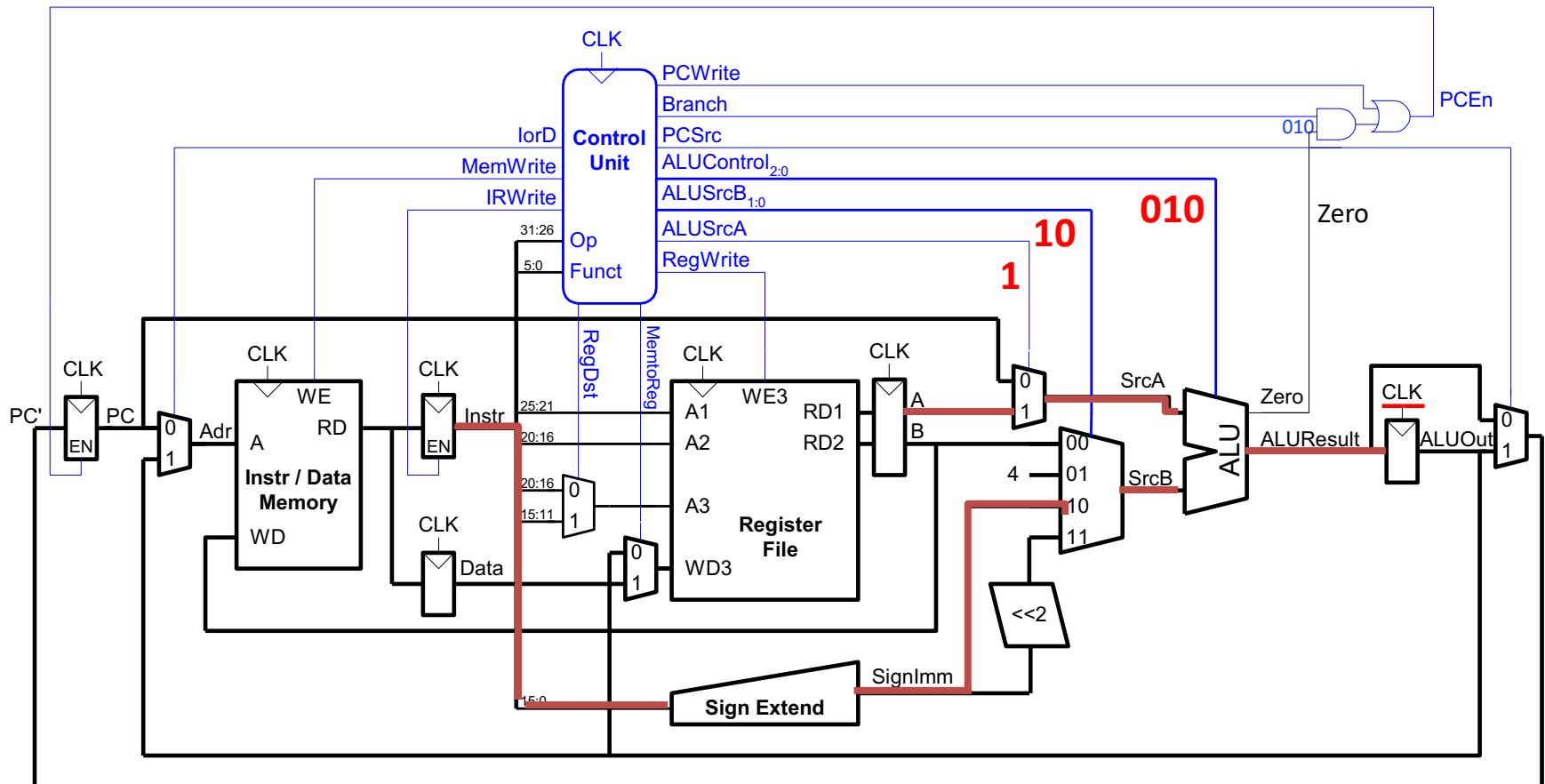
# Multicycle Datapath: SW

## STEP 3b: Sign-extend the immediate



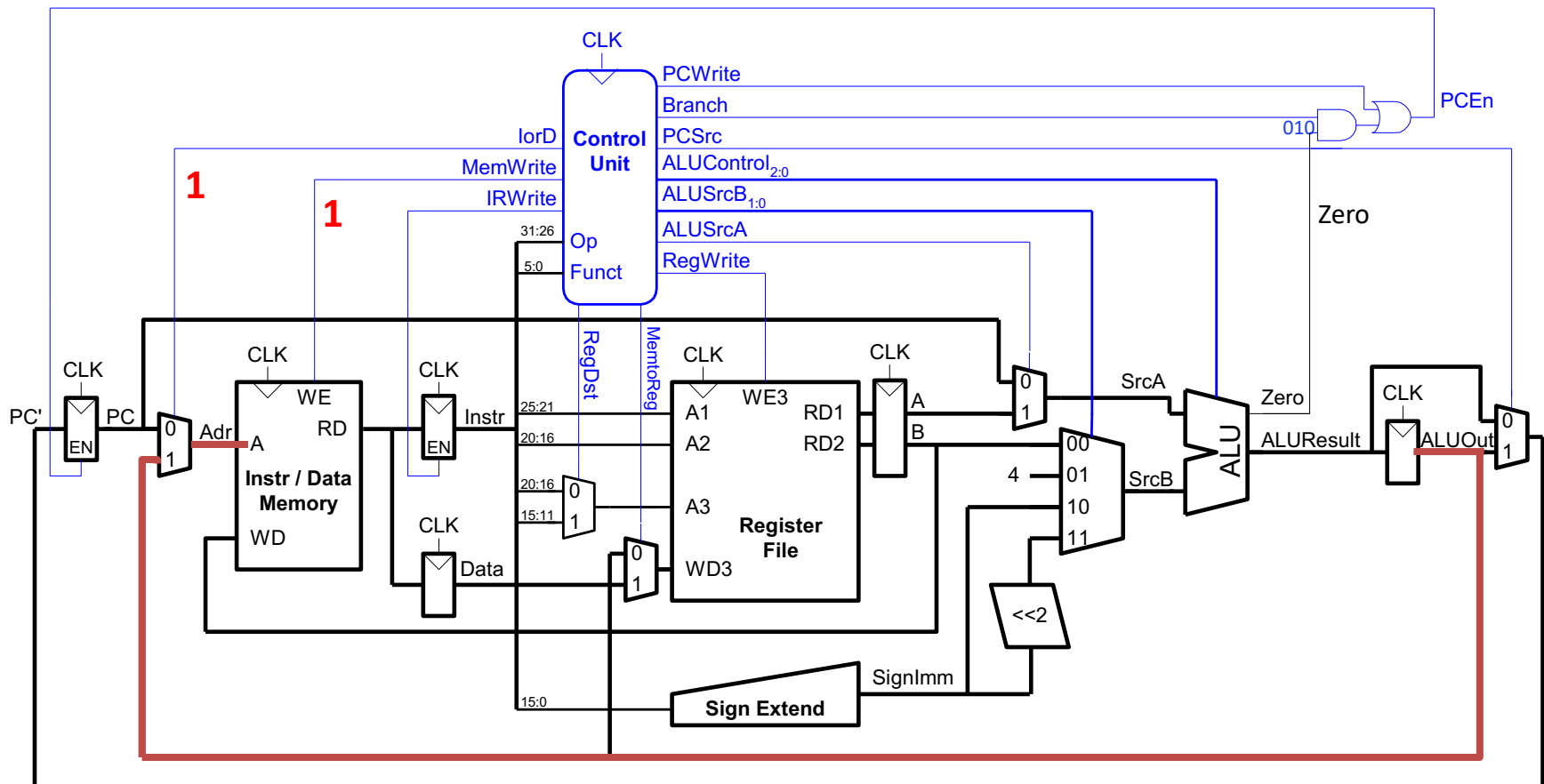
# Multicycle Datapath: SW

## STEP 4: Compute the memory address



# Multicycle Datapath: SW

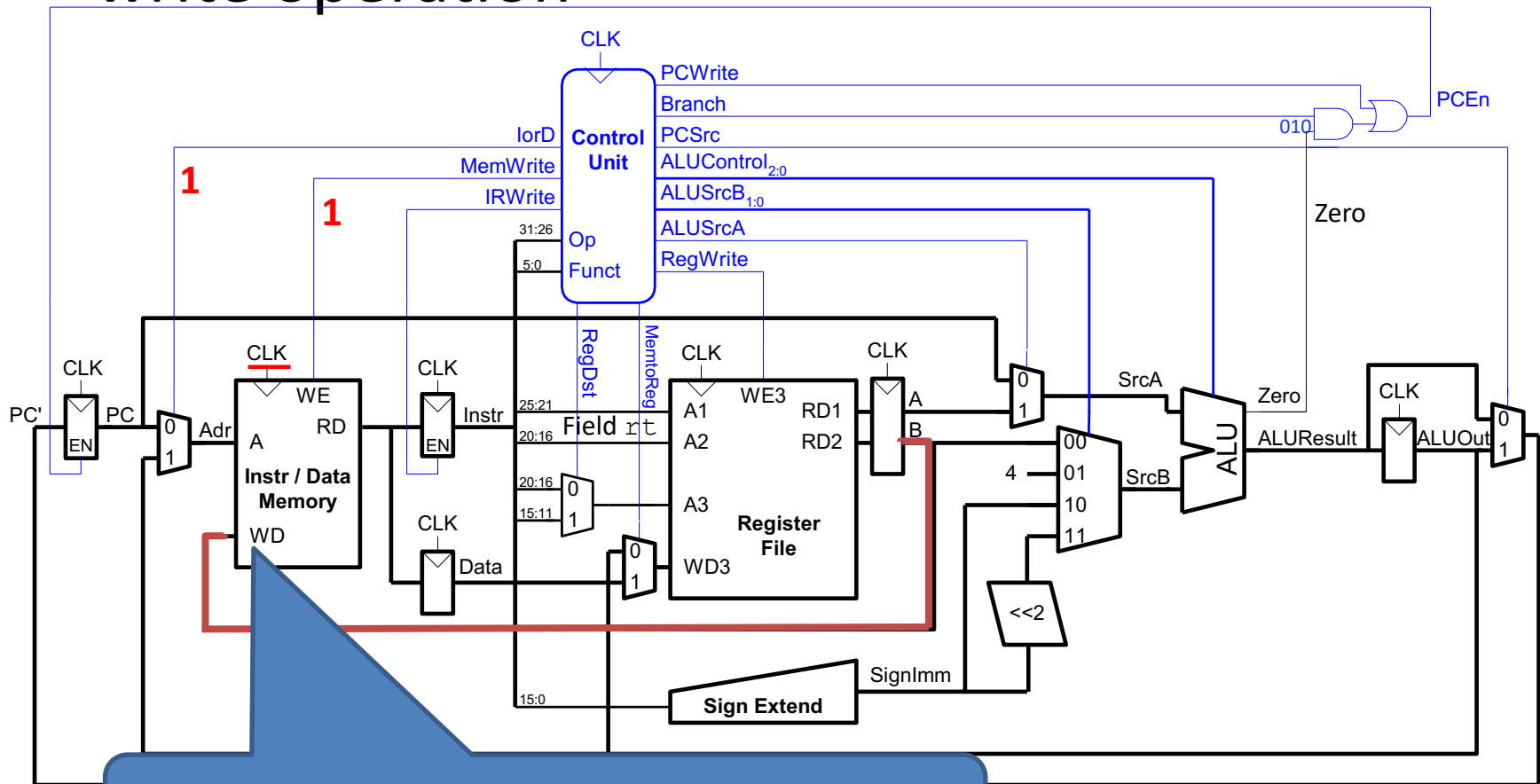
## STEP 5a: Send address to memory





# Multicycle Datapath: SW

**STEP 5b:** Send data to memory and activate write operation

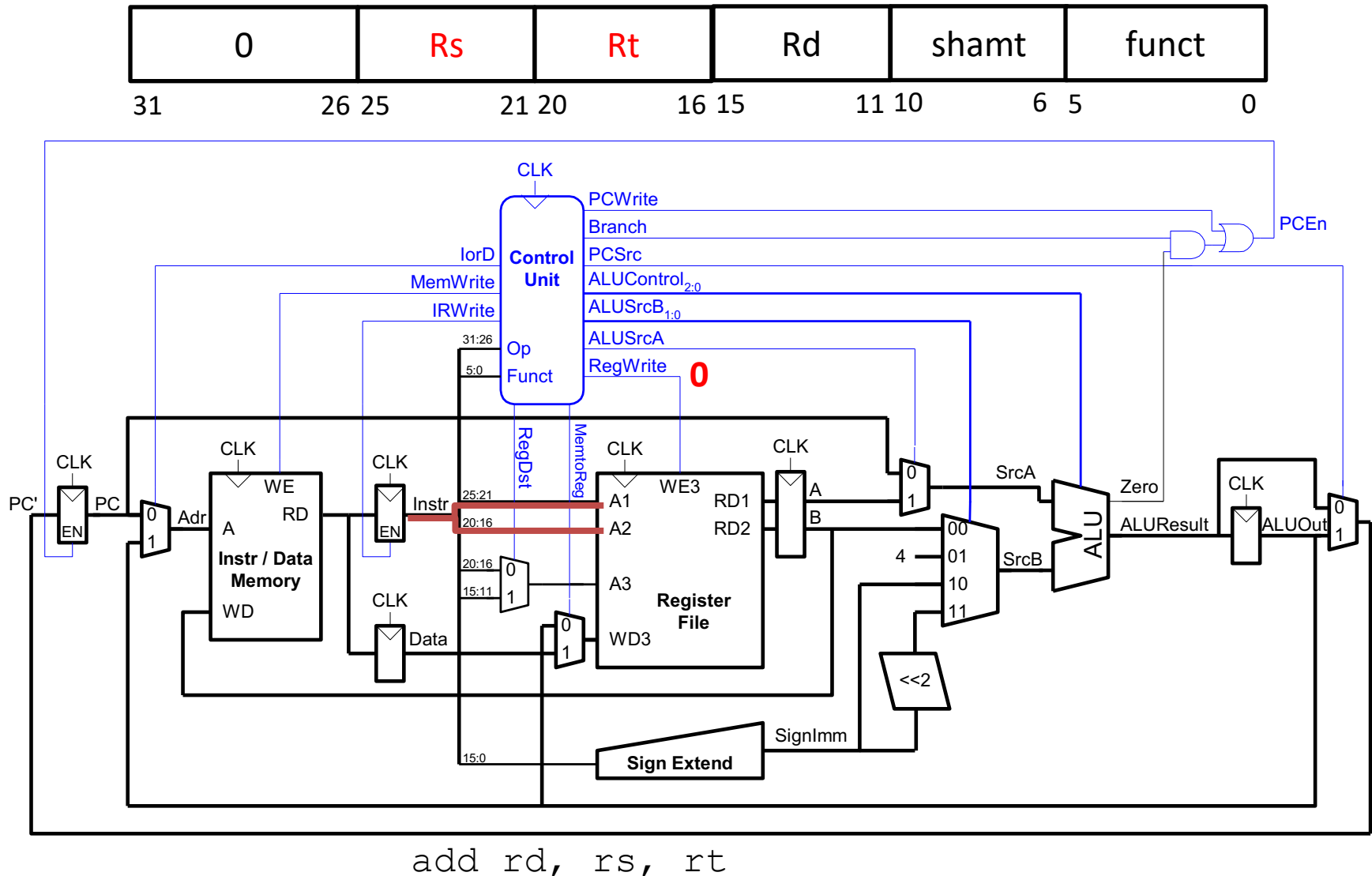


# Multicycle Datapath: R-Type instr

- Read from `rs` and `rt`  
`add rd, rs, rt`
- Write *ALUResult* to register file
- Write to `rd` (instead of `rt`)
- Steps 1 and 2 as in previous examples

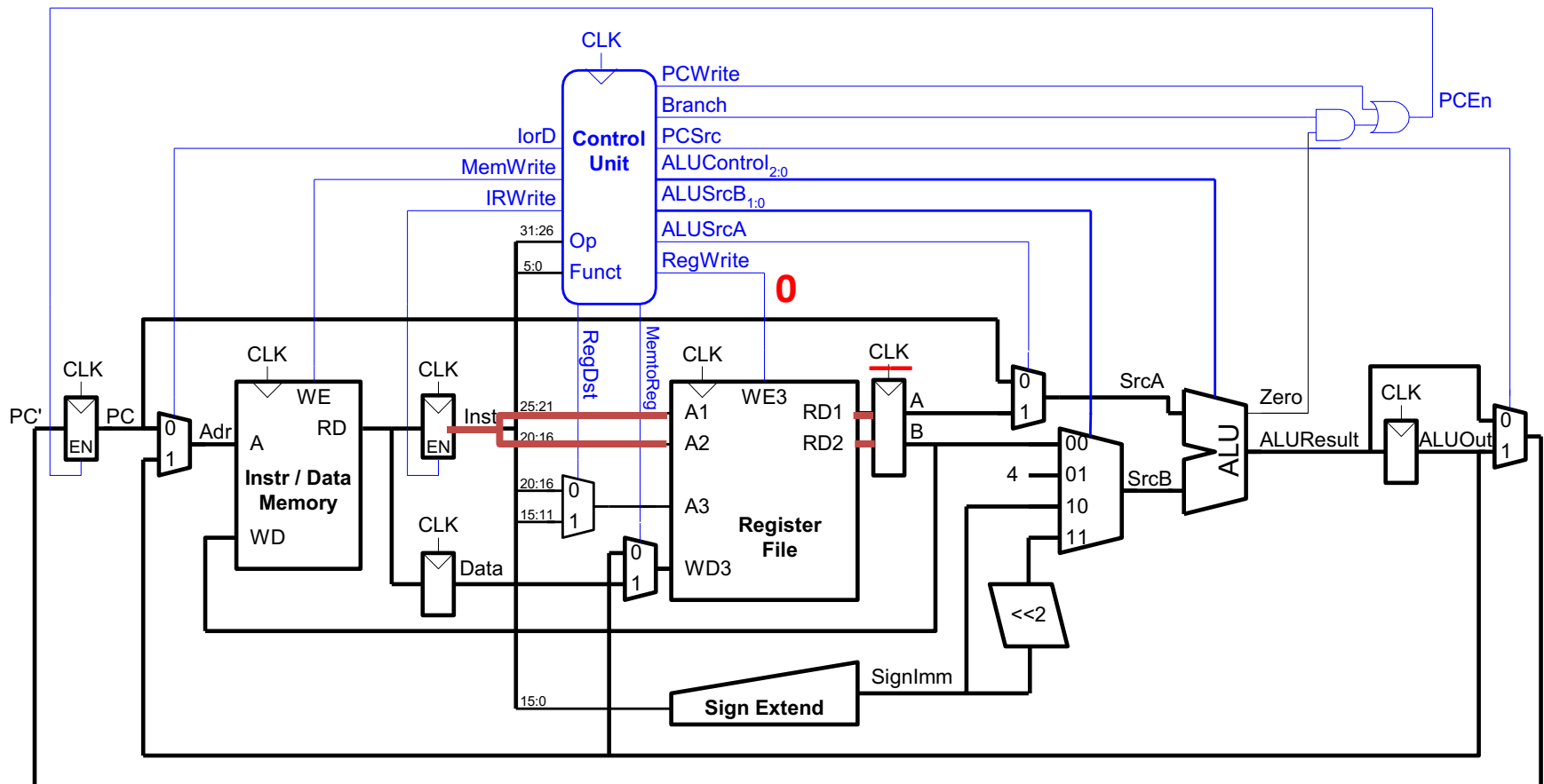
## Multicycle Datapath: R-Type instr

## STEP 3a: Read source operands from Register File



# Multicycle Datapath: R-Type instr

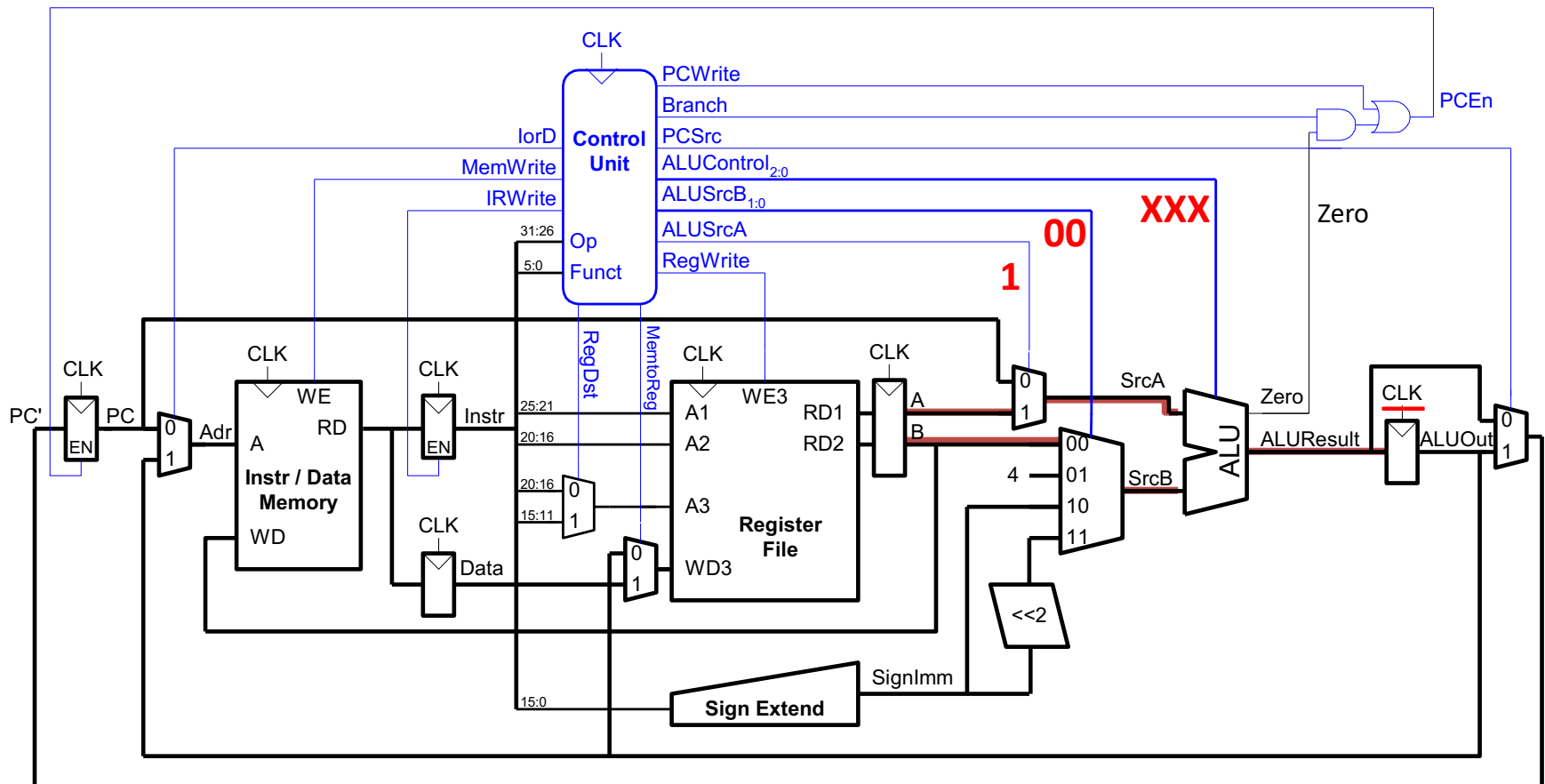
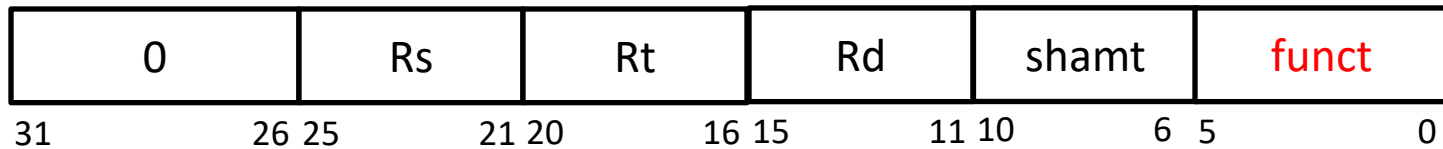
## STEP 3b: Write source operands into Register A/B



add rd, rs, rt

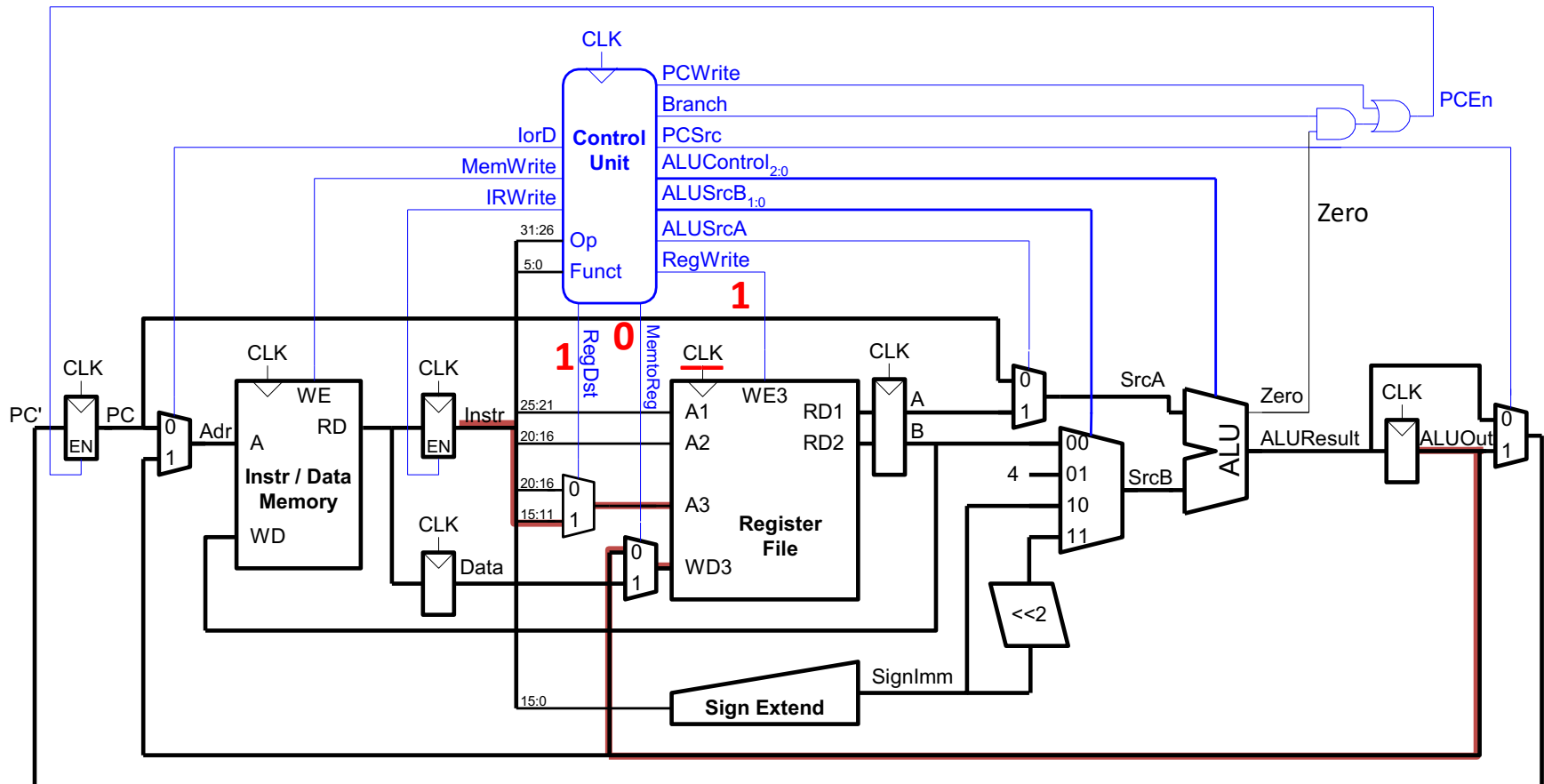
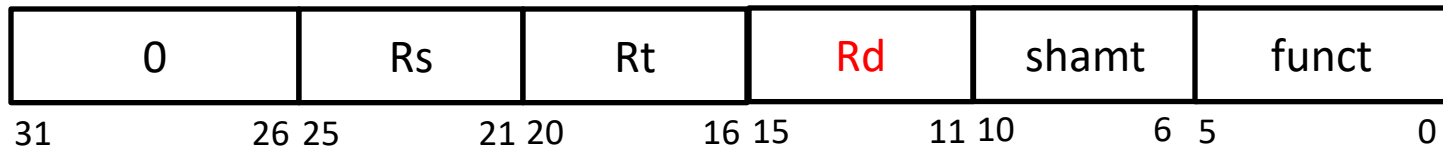
# Multicycle Datapath: R-Type instr

## STEP 4: Perform the required ALU operation



# Multicycle Datapath: R-Type instr

## STEP 5: Send result to Register File (Rd)



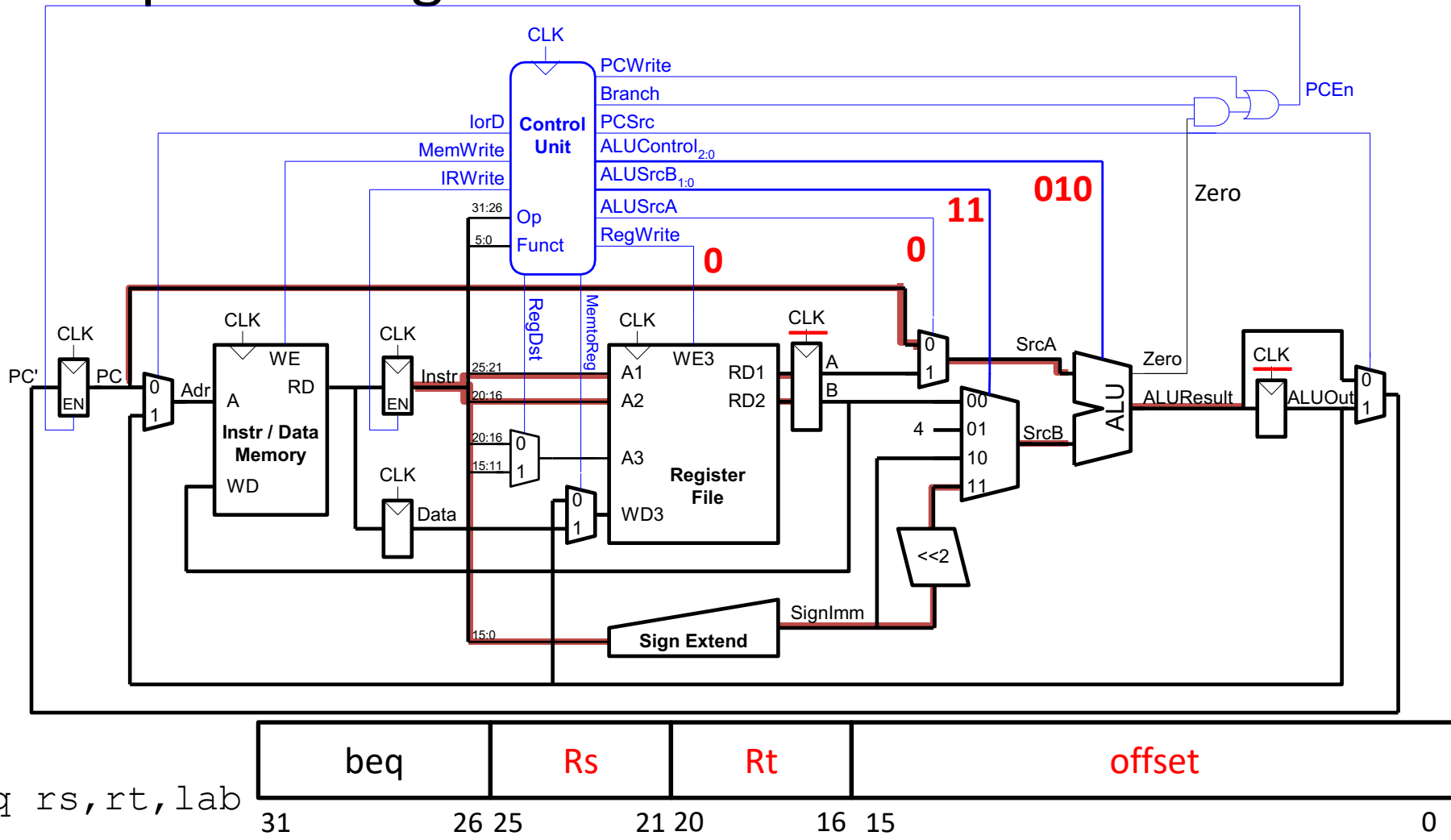
# Multicycle Datapath: beq

**beq rs, rt, lab**

- $rs == rt?$
- $BTA = (\text{sign-extended immediate} \ll 2) + (PC+4)$
- Steps 1 and 2 as in previous examples

# Multicycle Datapath: beq

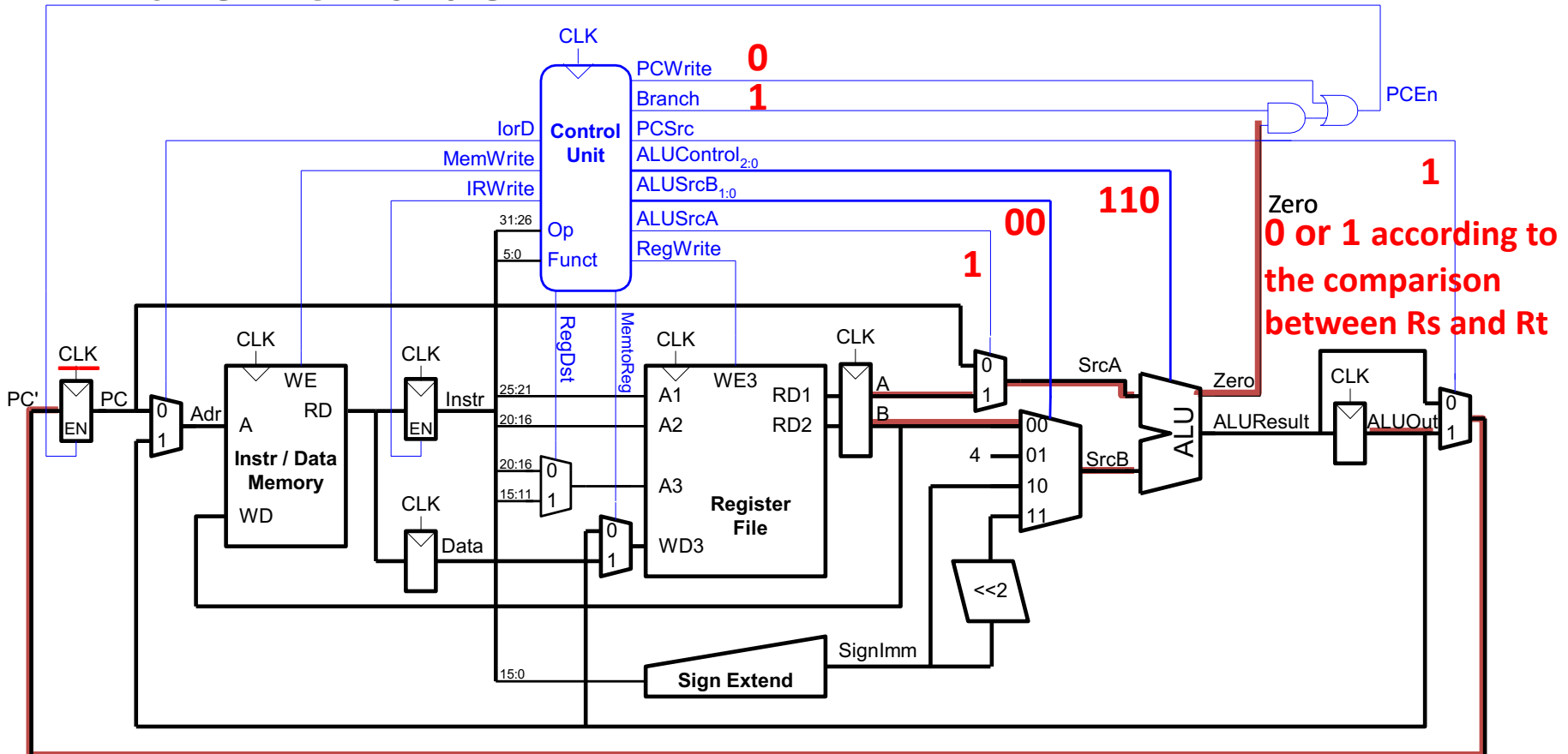
**STEP 3:** compute the new PC value / read the 2 operand registers from RF





# Multicycle Datapath: beq

**STEP 4:** compare the 2 operands and update the PC value

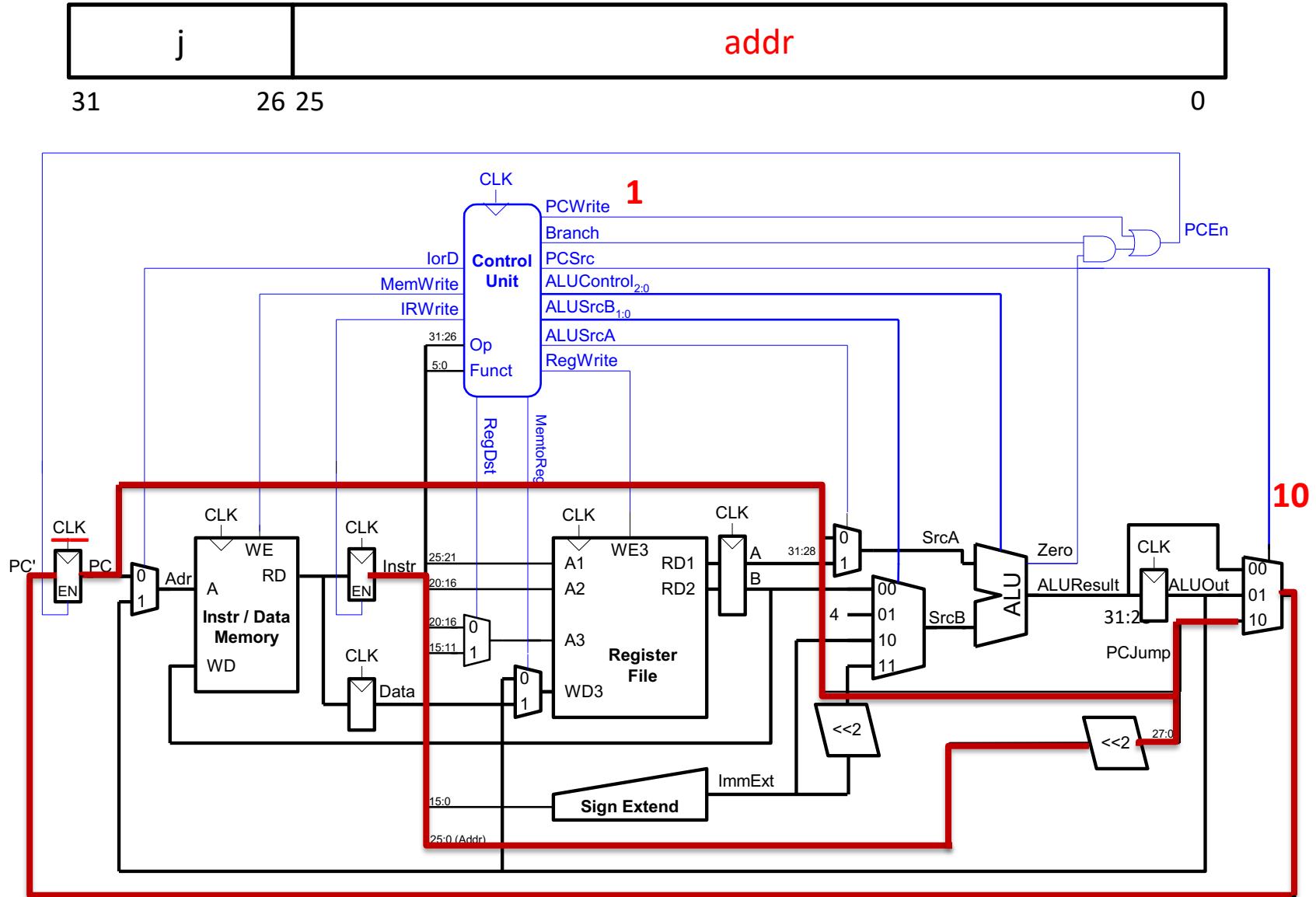


`beq rs, rt, lab`

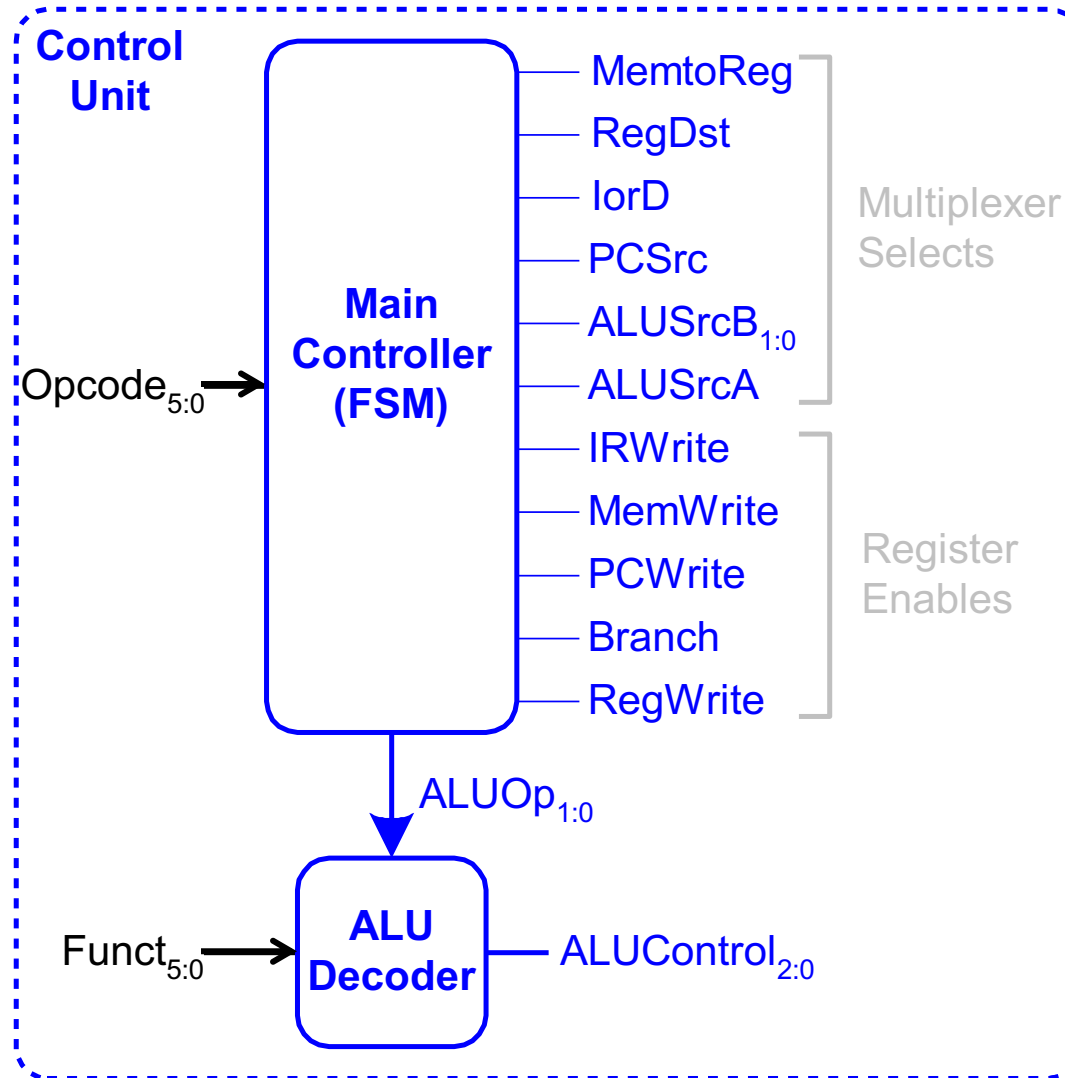
# Multicycle Datapath: j

- $\text{BTA} = (\text{jump address field} \ll 2)_{[25:0]}$  combined with  $(\text{PC}+4)_{[31:26]}$
- Steps 1 and 2 as in previous examples

# Multicycle Datapath+CU: j



# Multicycle Control



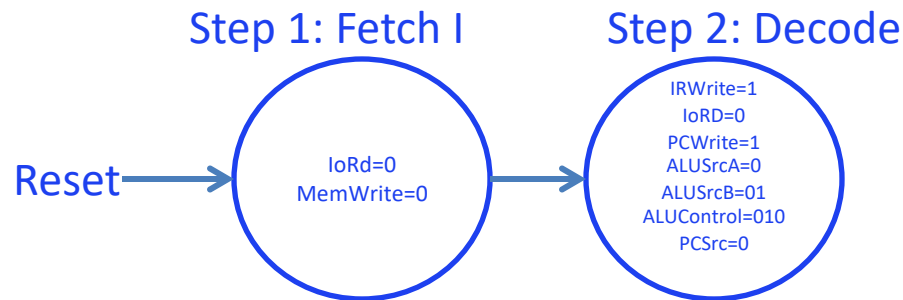
# Control Unit: ALU Decoder

ALUOp <sub>1:0</sub>	Meaning
00	Add
01	Subtract
10	Look at Funct
11	Not Used

ALUOp <sub>1:0</sub>	Funct	ALUControl <sub>2:0</sub>
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)

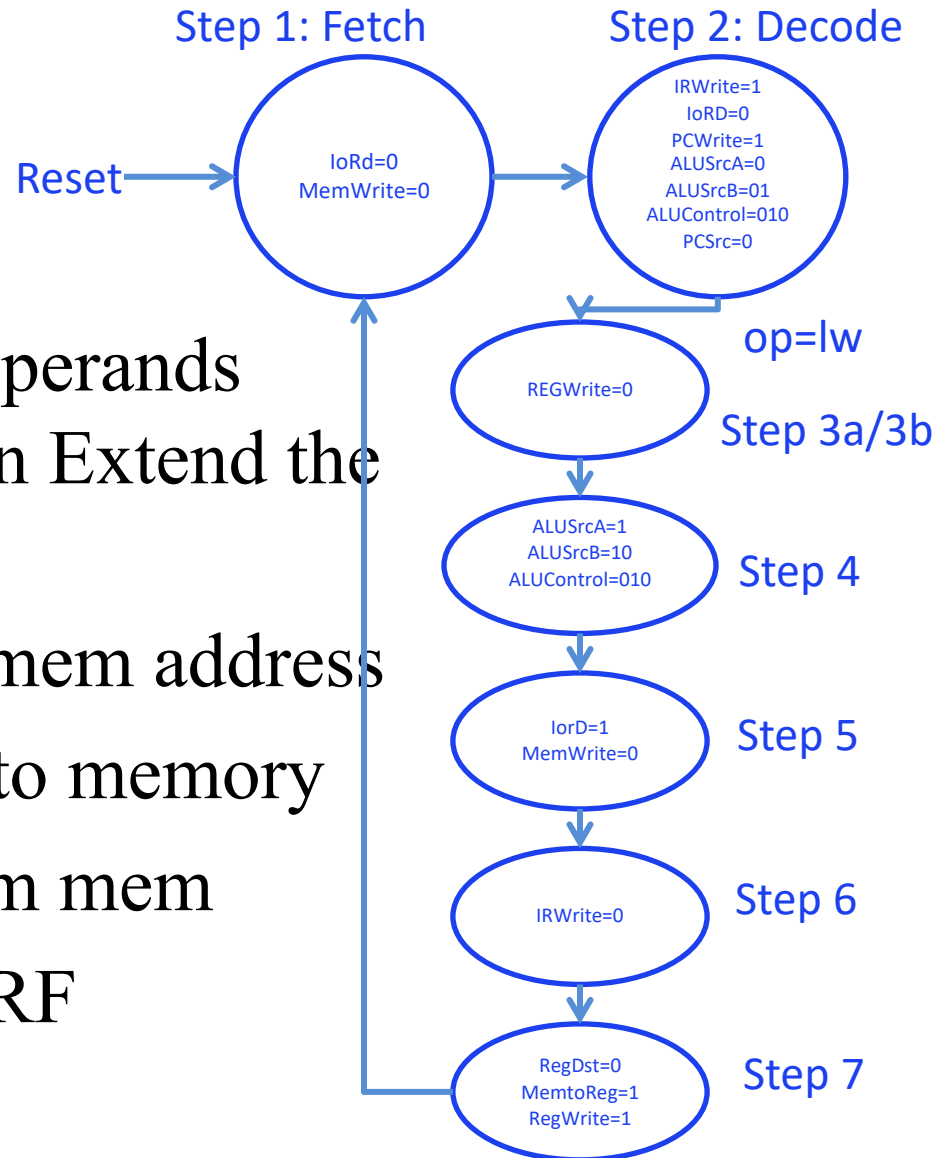
# Main Controller FSM: Fetch

1. Send the PC to memory
2. Read the instruction from memory and update PC



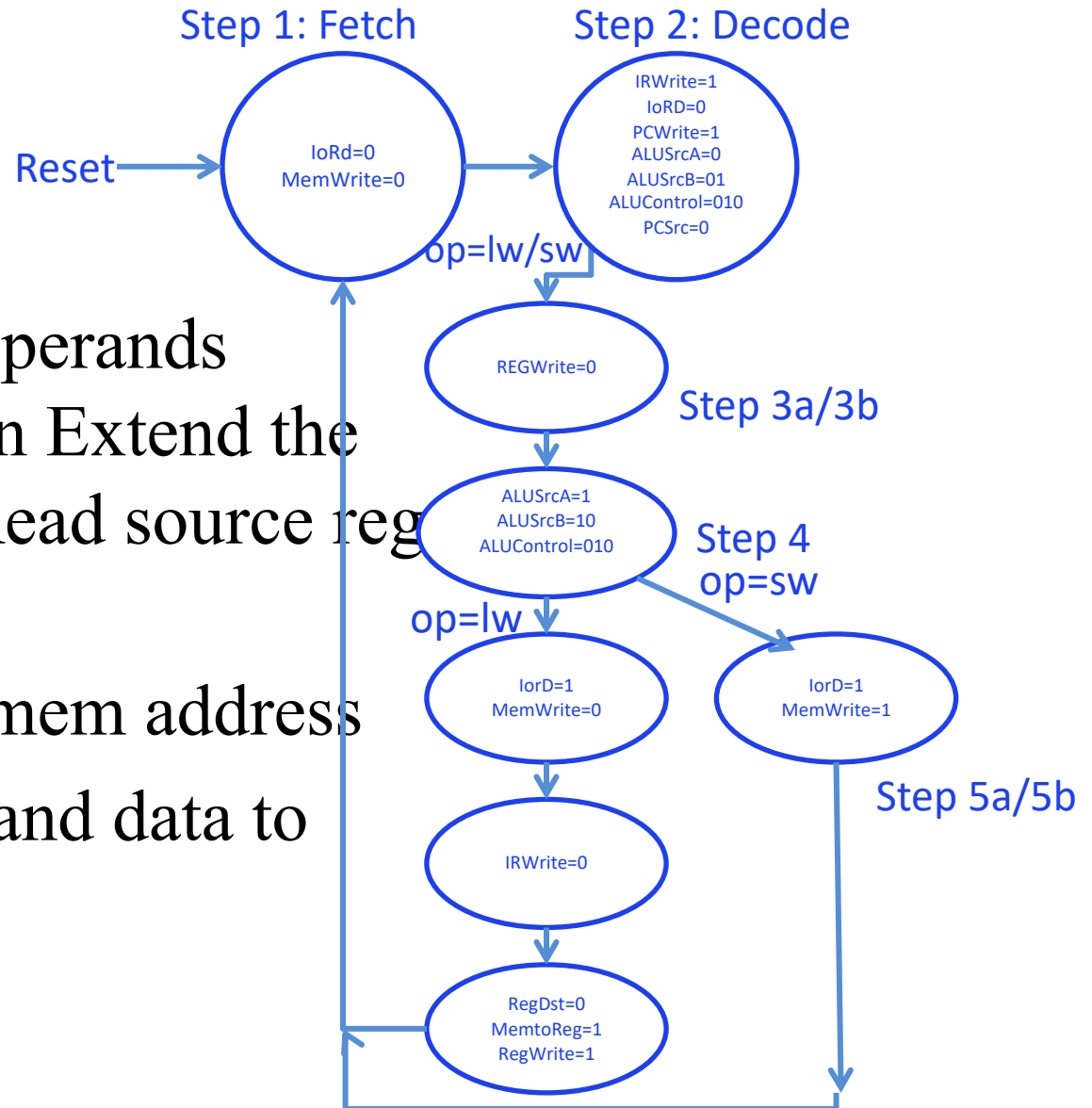
# Main Controller FSM: $lw$

3. Read source operands from RF / Sign Extend the Immediate
4. Compute the mem address
5. Send address to memory
6. Read data from mem
7. Write data to RF



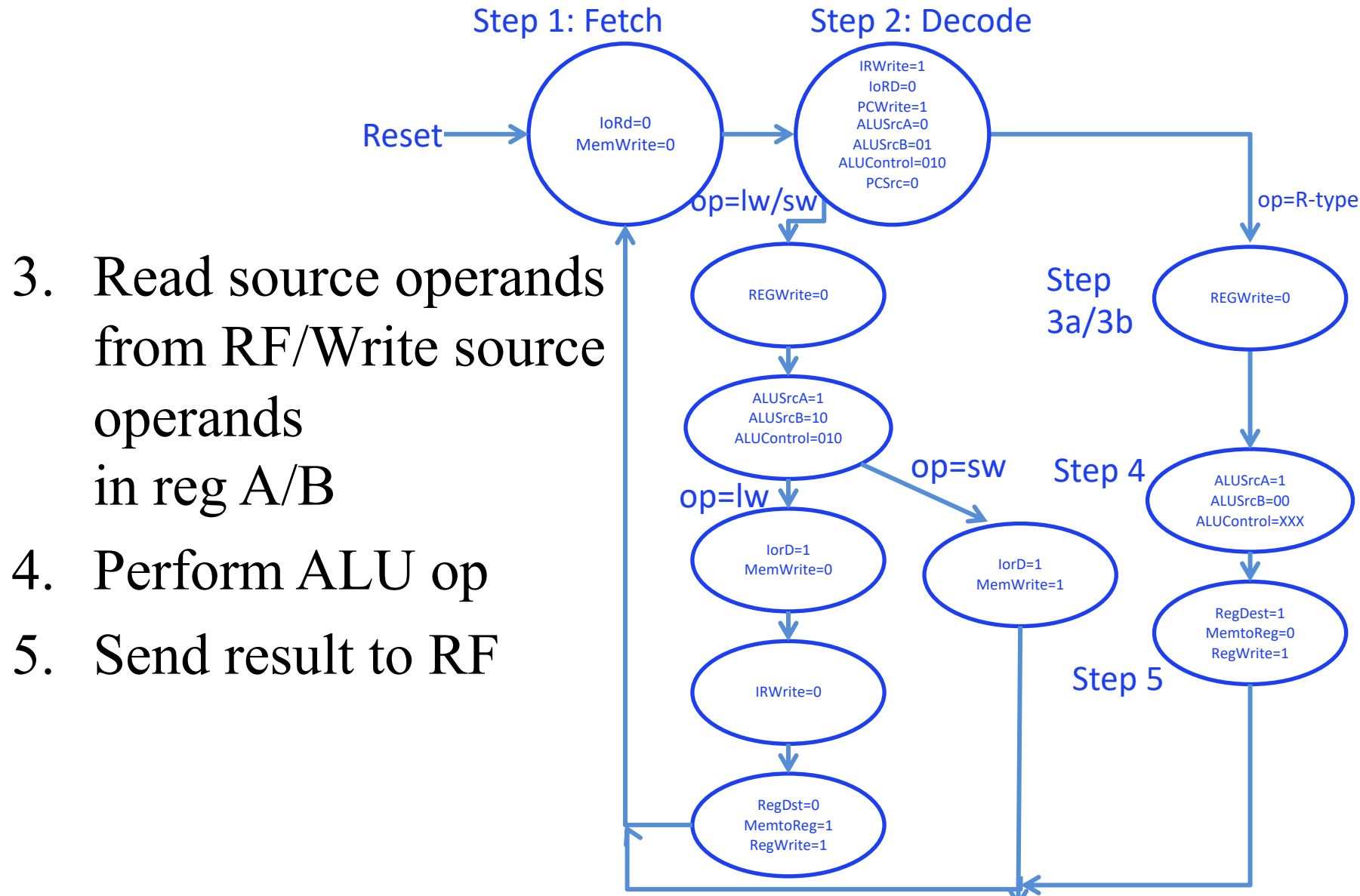
# Main Controller FSM: $SW$

3. Read source operands from RF / Sign Extend the Immediate / Read source reg from RF
4. Compute the mem address
5. Send address and data to memory





# Main Controller FSM: R-type



# Processor Performance

- Program execution time

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

- Definitions:
  - CPI: cycles/instruction
  - clock period: seconds/cycle
- Challenge is to satisfy constraints of:
  - Cost
  - Power
  - Performance

# Multicycle Processor Performance

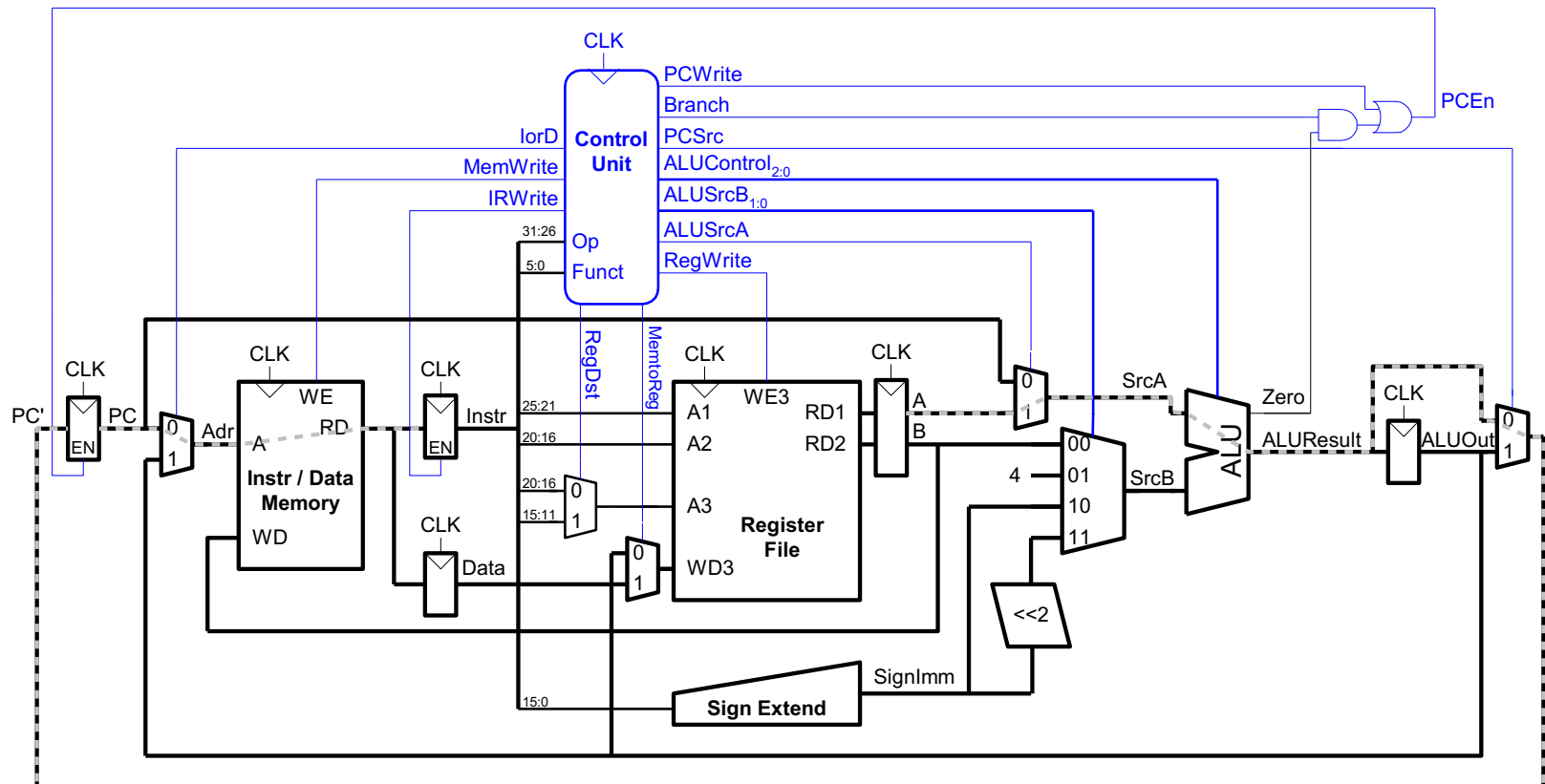
- Instructions take different number of cycles:
  - 3 cycles: beq, j
  - 5 cycles: sw
  - 5 cycles: R-Type
  - ...
  - 7 cycles: lw
- CPI is weighted average
- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type

**Average CPI** =  $(0.11 + 0.02)(3) + (0.10)(5) + \dots + (0.25)(7) = \mathbf{4.12}$

# Multicycle Processor Performance

Multicycle critical path:

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



# 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

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

In this example a Synchronous memory is considered.

# Multicycle Performance Example

Program with 100 billion instructions

$$\begin{aligned}\text{Execution Time} &= (\# \text{ instructions}) \times \text{CPI} \times T_c \\ &= (100 \times 10^9)(4.12)(325 \times 10^{-12}) \\ &= \mathbf{133.9 \text{ seconds}}\end{aligned}$$