CprE 381: Computer Organization and Assembly Level Programming

Multi-Cycle to Pipelining

Henry Duwe
Electrical and Computer Engineering
Iowa State University

Administrative

- Term Project
  - Part 2a check-in due date BEFORE Spring Break
- Midterm Grades due Fri
  - Your Task: On Thursday, double-check the scores that have been entered

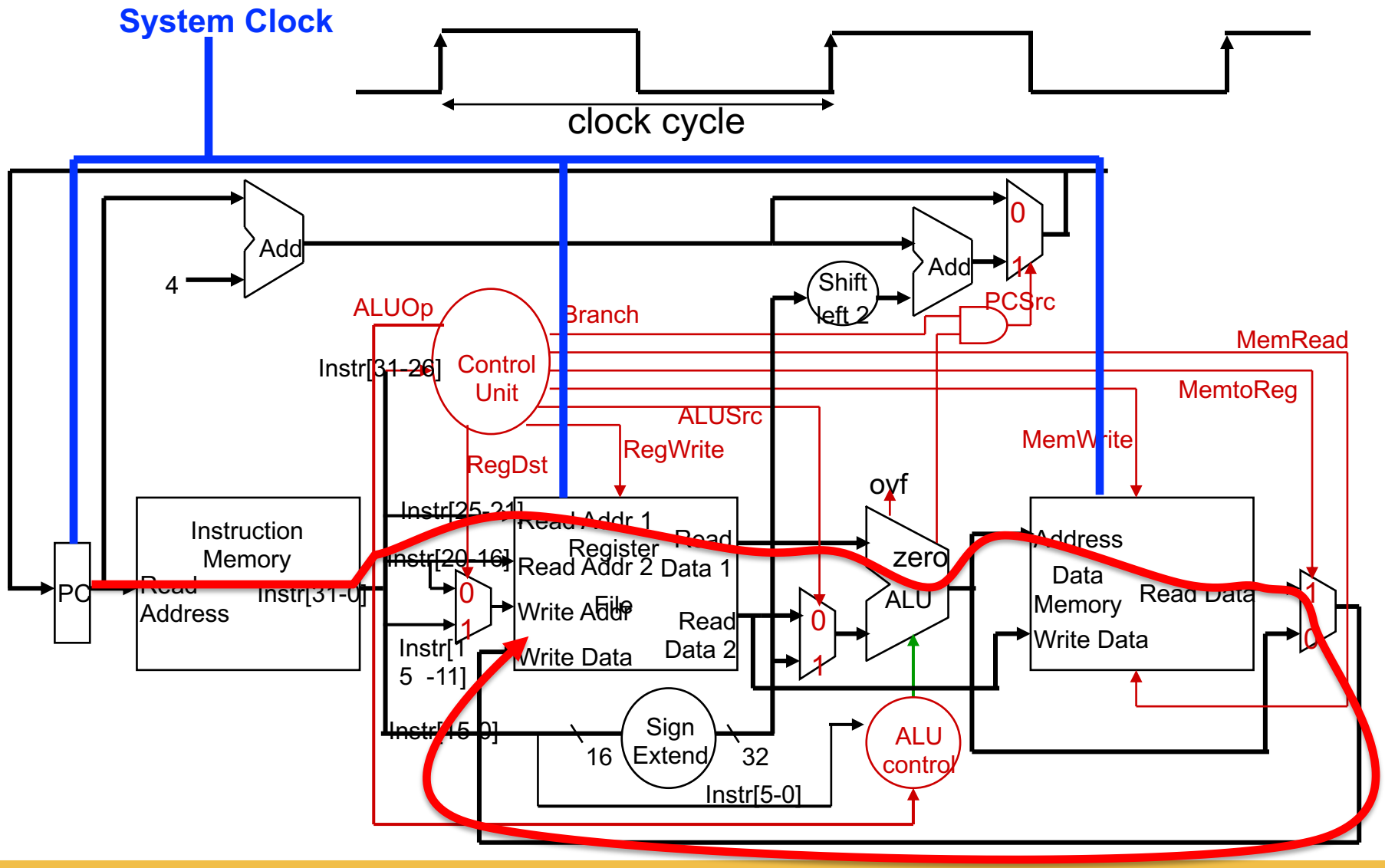
Review: Execution Time

- Drawing on the previous equation:

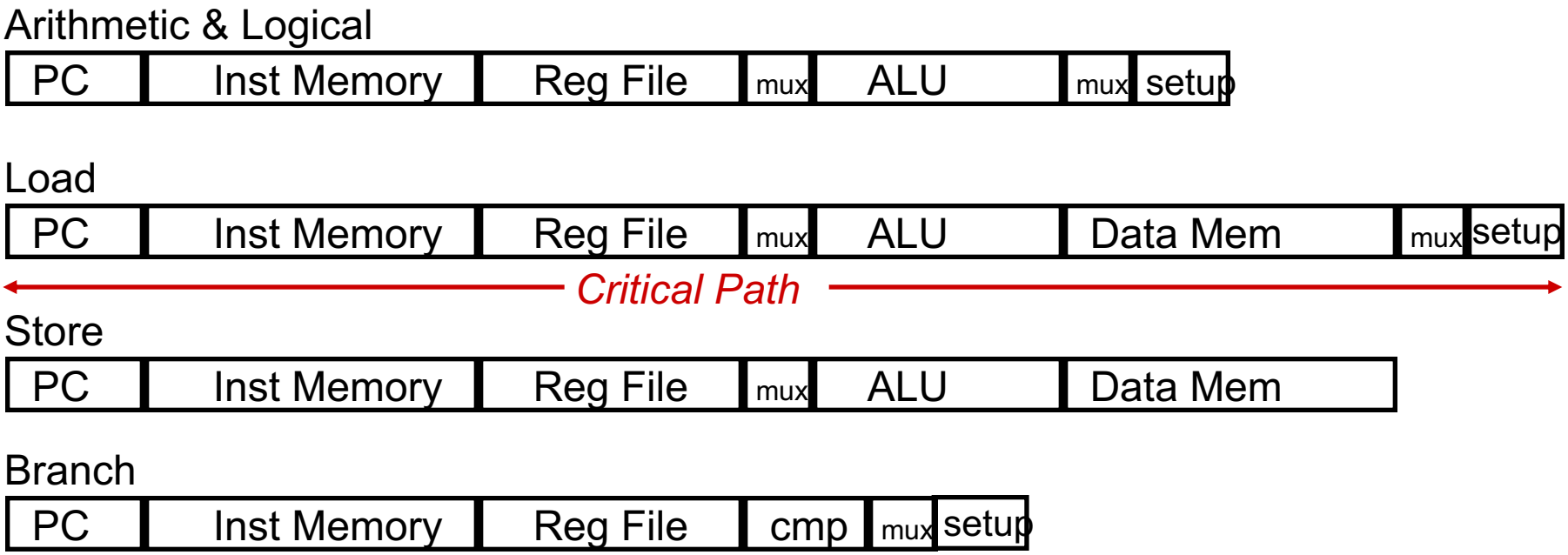
$$\textit{Execution Time} = \#\ \textit{Instructions} \times \frac{\textit{Cycles}}{\textit{Instruction}} \times \frac{\textit{Seconds}}{\textit{Cycle}}$$

- To improve performance (i.e., reduce execution time)
  - Increase clock rate (decrease clock cycle time) OR
  - Decrease CPI OR
  - Reduce the number of instructions
- Designers balance cycle time against the number of cycles required
  - Improving one factor may make the other one worse...

Review: Worst Case Timing (Load Instruction)



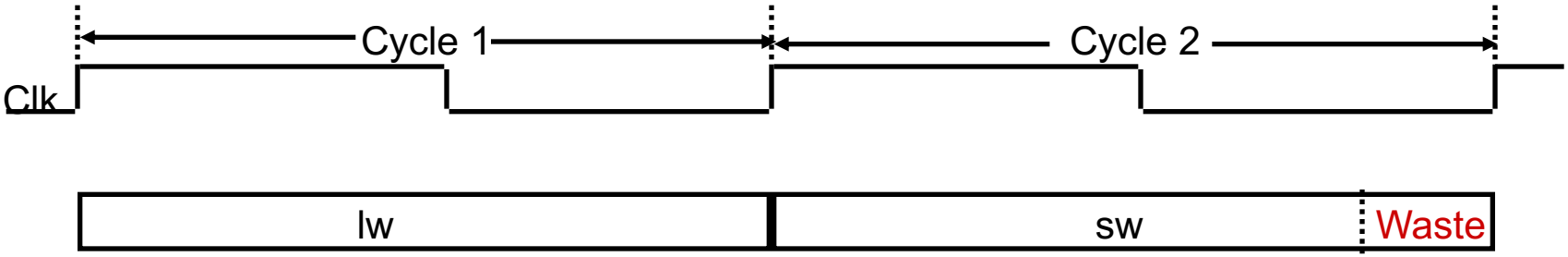
What's wrong with our CPI=1 CPU?



- Long Cycle Time
- All instructions take as much time as the slowest
- Real memory is not so nice as our idealized memory
  - Cannot always get the job done in one (short) cycle

Single Cycle Disadvantages & Advantages

- Uses the clock cycle inefficiently – the clock cycle must be timed to accommodate the slowest instr
  - Especially problematic for more complex instructions like floating point multiply



- May be wasteful of area since some functional units (e.g., adders) must be duplicated since they can not be shared during a clock cycle

But

- It is simple, easy to understand + has CPI of 1

Multicycle Implementation Overview

- Each instruction step takes 1 clock cycle
  - Therefore, an instruction takes more than 1 clock cycle to complete
- Not every instruction takes the same number of clock cycles to complete

- Multicycle implementations allow:
  - faster clock rates
  - different instructions to take a different number of clock cycles

Multicycle Implementation Overview

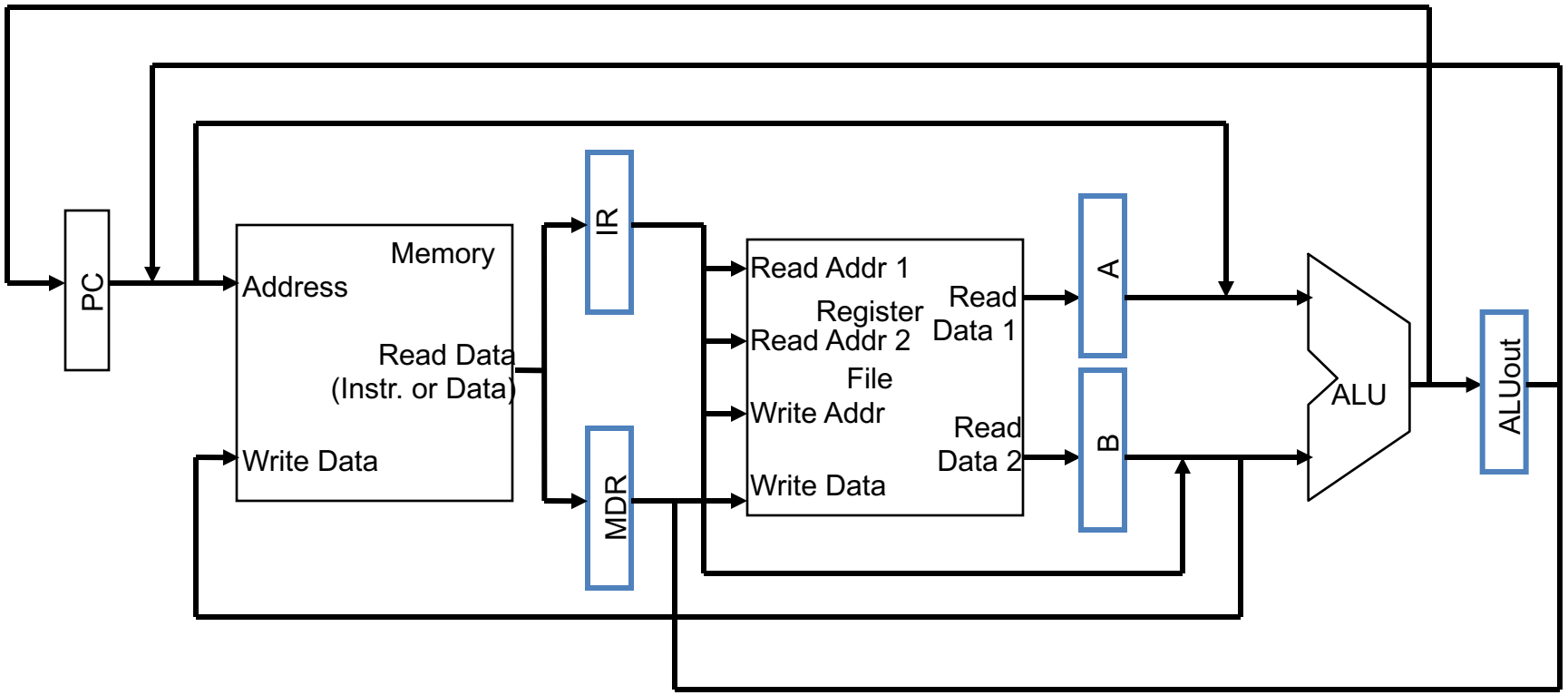
- Each instruction step takes 1 clock cycle
  - Therefore, an instruction takes more than 1 clock cycle to complete
- Not every instruction takes the same number of clock cycles to complete

- Multicycle implementations allow:
  - faster clock rates
  - different instructions to take a different number of clock cycles
  - functional units to be used more than once per instruction as long as they are used on different clock cycles, as a result
    - only need one memory
    - only need one ALU/adder

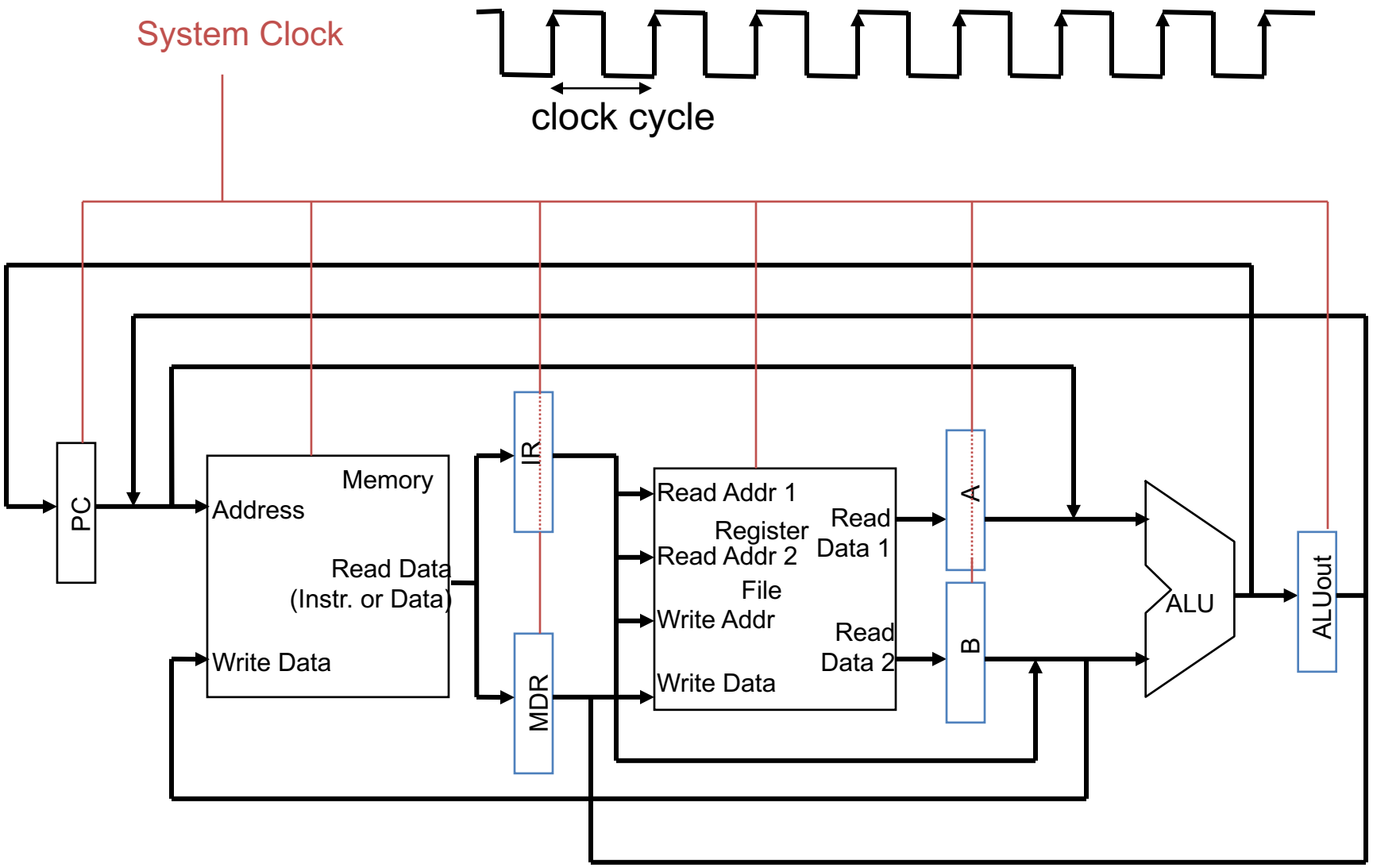
- Challenge: designing control logic

The Multicycle Datapath

- Registers have to be added after every major functional unit to hold the output value until it is used in a subsequent clock cycle



Clocking the Multicycle Datapath



Our Multicycle Approach

- Break up the instructions into steps where each step takes a clock cycle while trying to
  - **Balance** the amount of work to be done in each step
  - Use only one major functional unit per clock cycle
- At the end of a clock cycle
  - Store values needed in a later clock cycle by the current instruction in a state element (internal register not visible to the programmer)

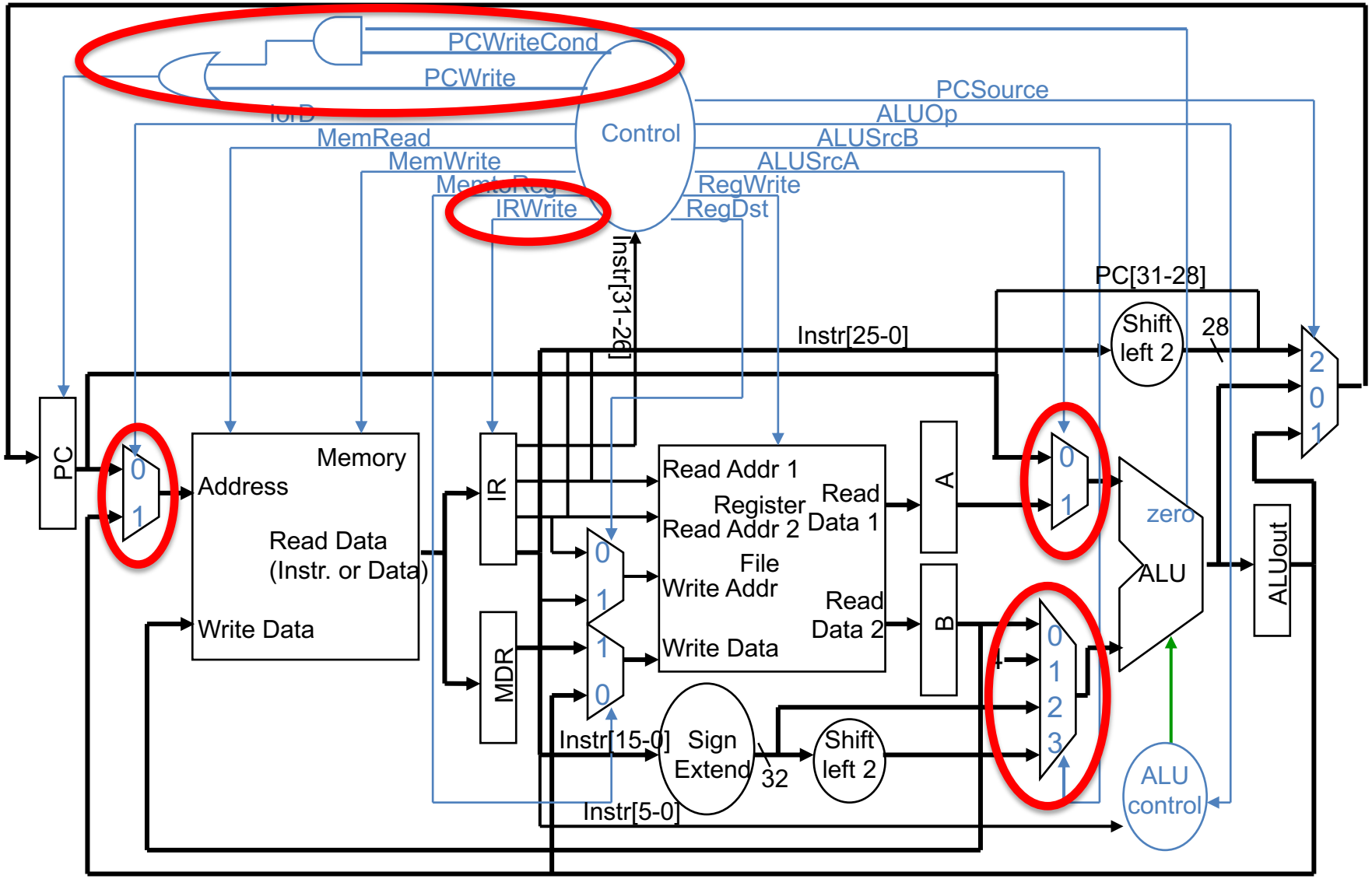
    IR – Instruction Register

    MDR – Memory Data Register

    A and B – Register File read data registers

    ALUout – ALU output register
    - All (except IR) hold data only between a pair of adjacent clock cycles (so they don't need a write control signal)
  - Data used by subsequent instructions are stored in programmer visible state elements (i.e., Register File, PC, or Memory)

The Complete Multicycle Data with Control



Our Multicycle Approach (cont.)

- Reading from or writing to any of the internal registers, Register File, or the PC occurs (quickly) at the beginning (for read) or the end of a clock cycle (for write)

- Reading from the Register File takes ~50% of a clock cycle since it has additional control and access overhead (but reading can be done in parallel with decode)

- Had to add multiplexors in front of several of the functional unit input ports (e.g., Memory, ALU) because they are now shared by different clock cycles and/or do multiple jobs

- **All operations occurring in one clock cycle occur in parallel**
  - This limits us to one ALU operation, one Memory access, and one Register File access per clock cycle

Five Instruction Steps

1. Instruction Fetch
2. Instruction Decode and Register Fetch
3. R-type Instruction Execution, Memory Read/Write Address Computation, Branch Completion, or Jump Completion
4. Memory Read Access, Memory Write Completion or R-type Instruction Completion
5. Memory Read Completion (Write Back)

INSTRUCTIONS TAKE FROM 3 - 5 CYCLES!

Step 1: Instruction Fetch

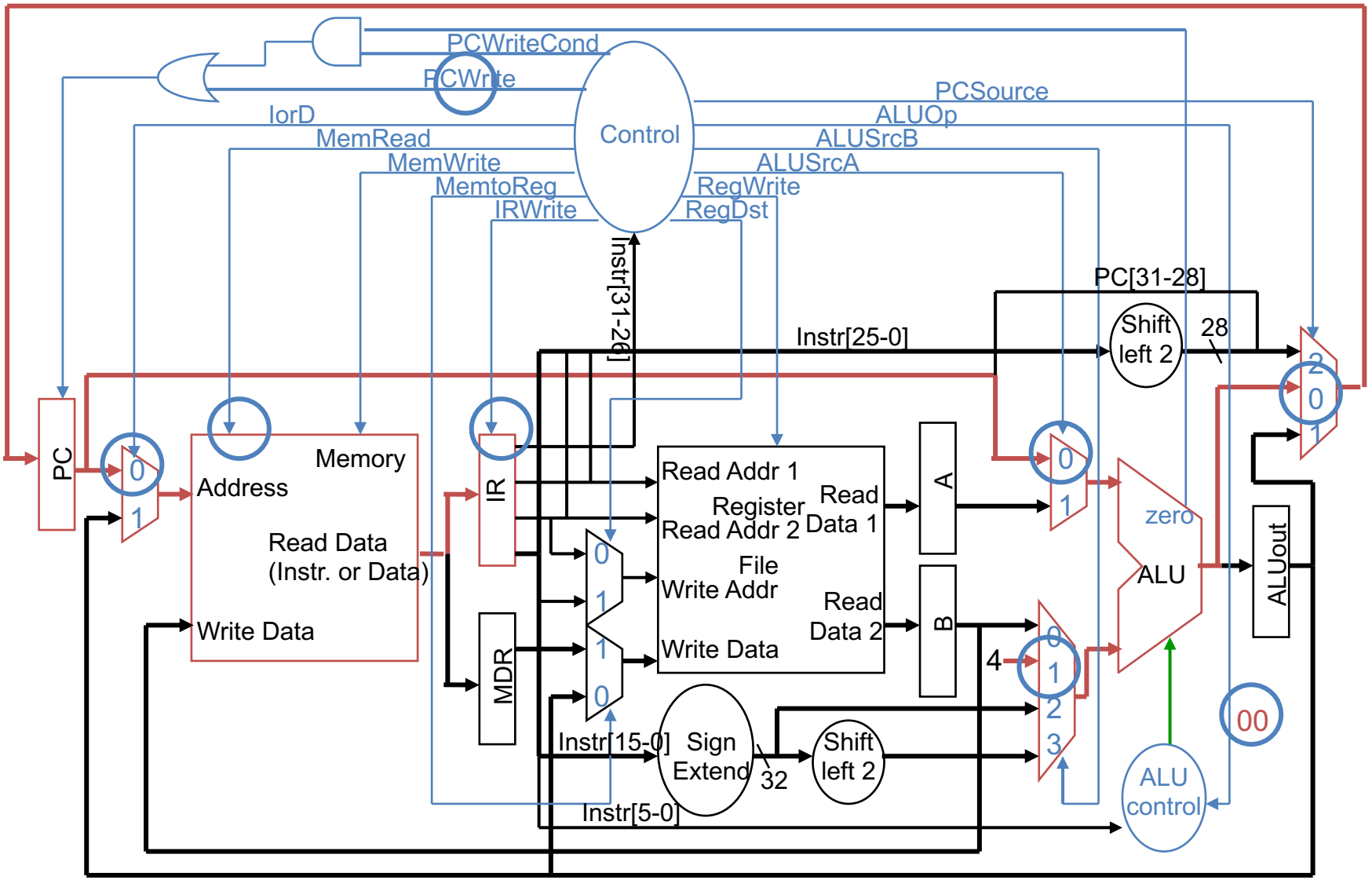
- Use PC to get instruction from the memory and put it in the Instruction Register
- Increment the PC by 4 and put the result back in the PC
- Can be described succinctly using the RTL "Register-Transfer Language"

```
IR = Memory[PC];
PC = PC + 4;
```

What is the advantage of updating the PC now?

Can we figure out the values of the control signals?

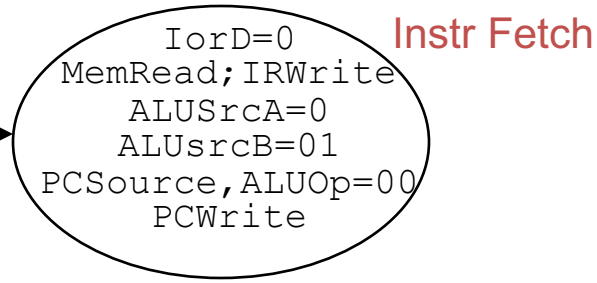
Datapath Activity During Instr Fetch



Fetch Control Signal Settings

Unless otherwise assigned

PCWrite,IRWrite,
MemWrite,RegWrite=0
others=X



Instr Fetch

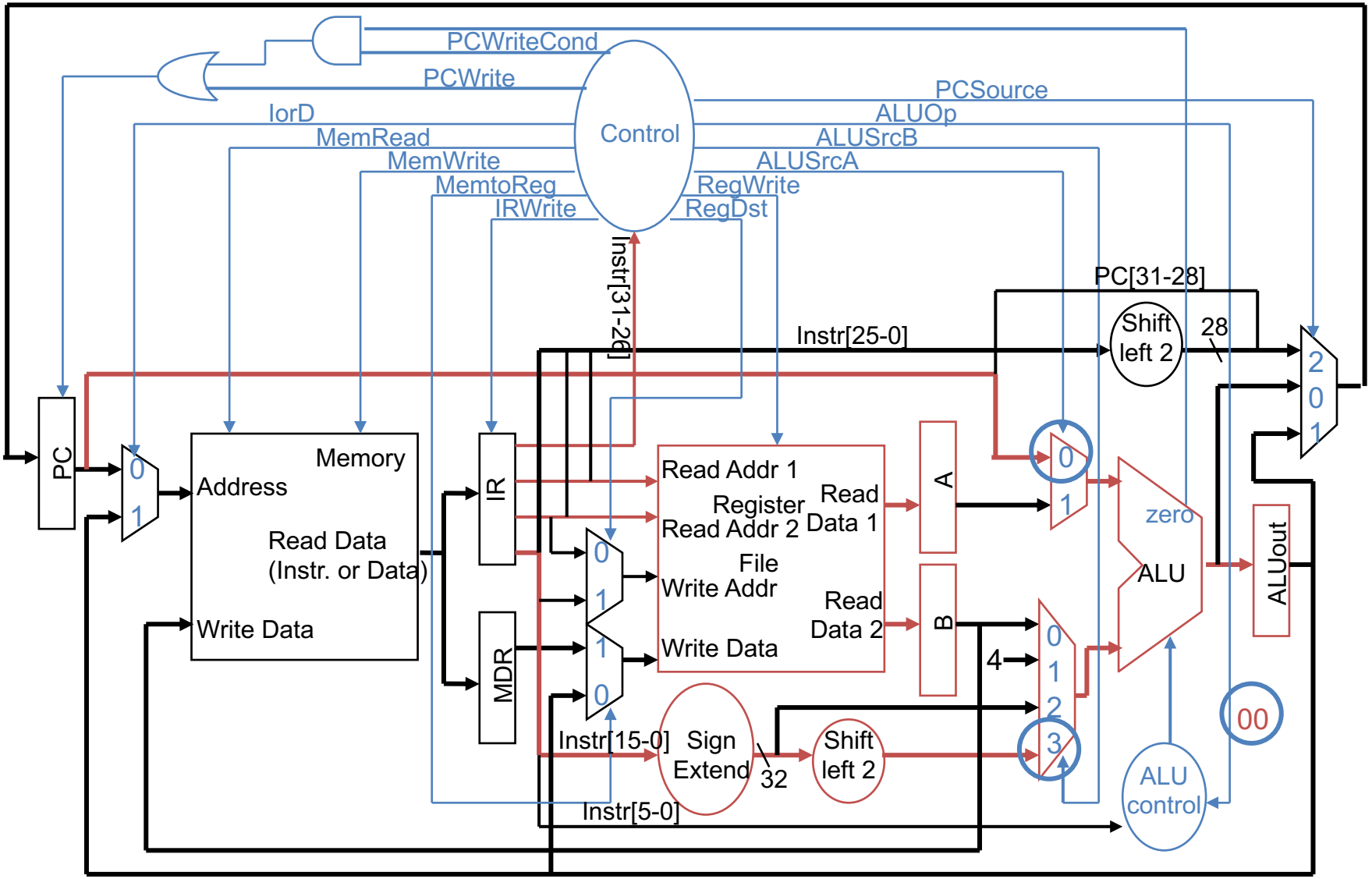
Step 2: Instr Decode and Reg Fetch

- Don’t know what the instruction is yet, so can only
  - Read registers rs and rt in case we need them
  - Compute the branch address in case the instruction is a branch
- The RTL:

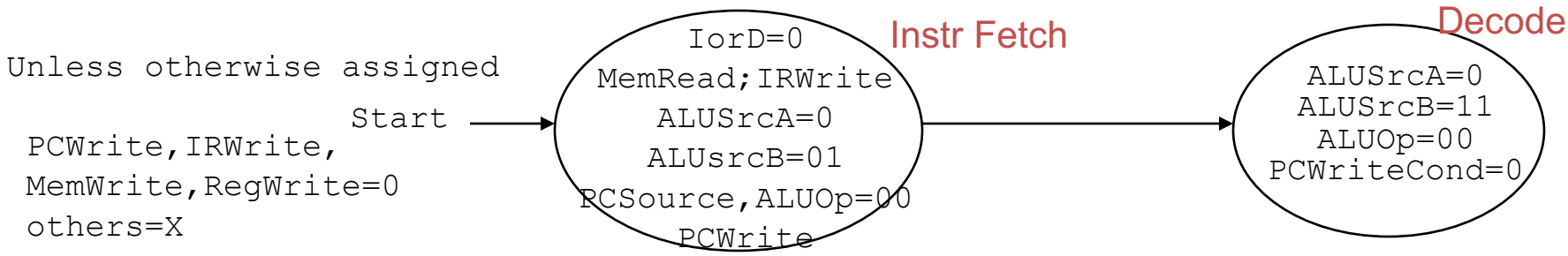
```
A = Reg[IR[25-21]];
B = Reg[IR[20-16]];
ALUOut = PC +(sign-extend(IR[15-0])<< 2);
```

- Note we aren't setting any control lines based on the instruction (since we don’t know what it is (the control logic is busy "decoding" the op code bits))

Datapath Activity During Instr Decode



Decode Control Signals Settings



Step 3 Execute (Instruction Dependent)

- ALU is performing one of four functions, based on instruction type

- Memory reference (**lw** and **sw**):

```
ALUOut = A + sign-extend(IR[15-0]);
```

- R-type:

```
ALUOut = A op B;
```

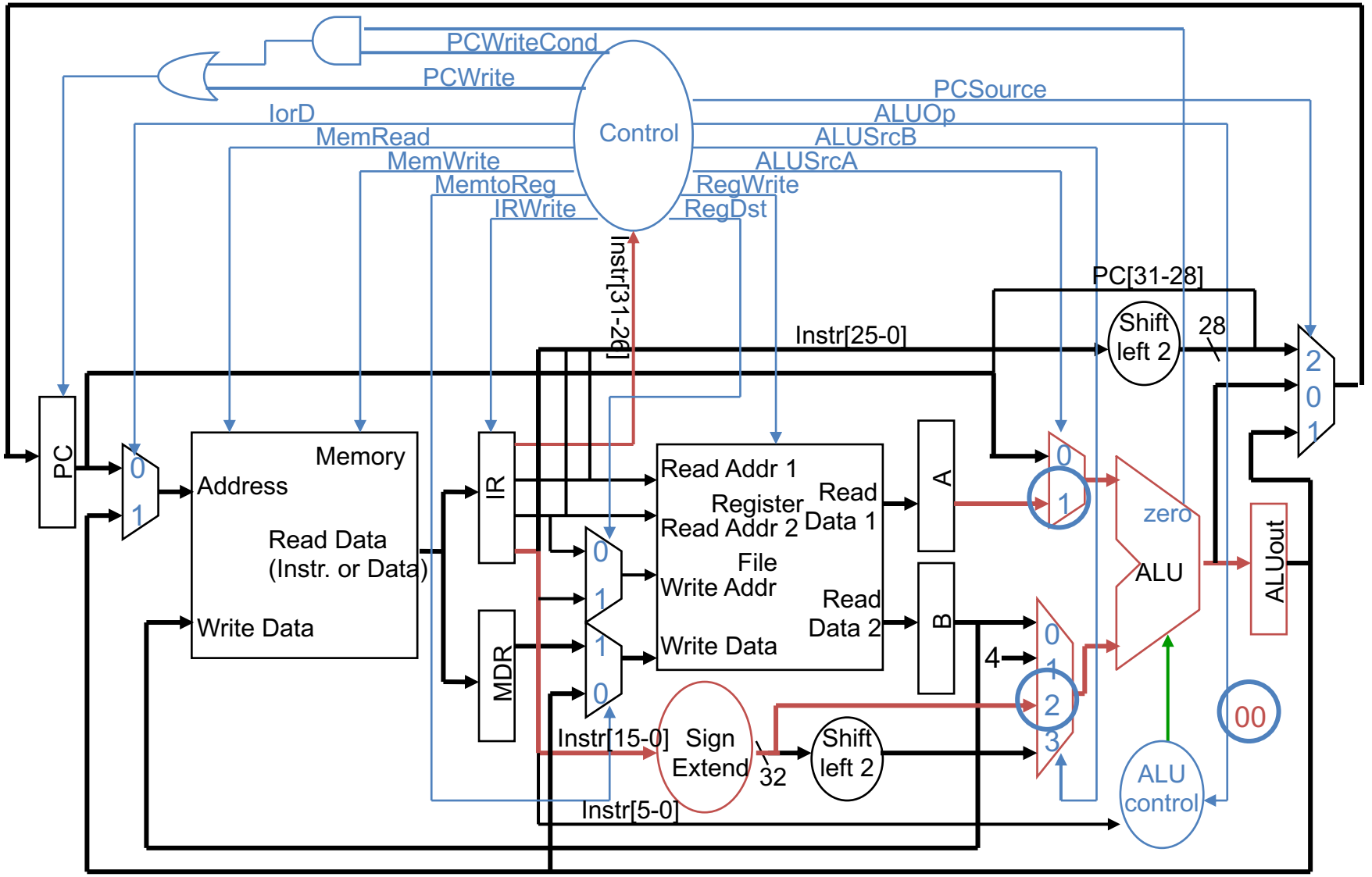
- Branch:

```
if (A==B) PC = ALUOut;
```

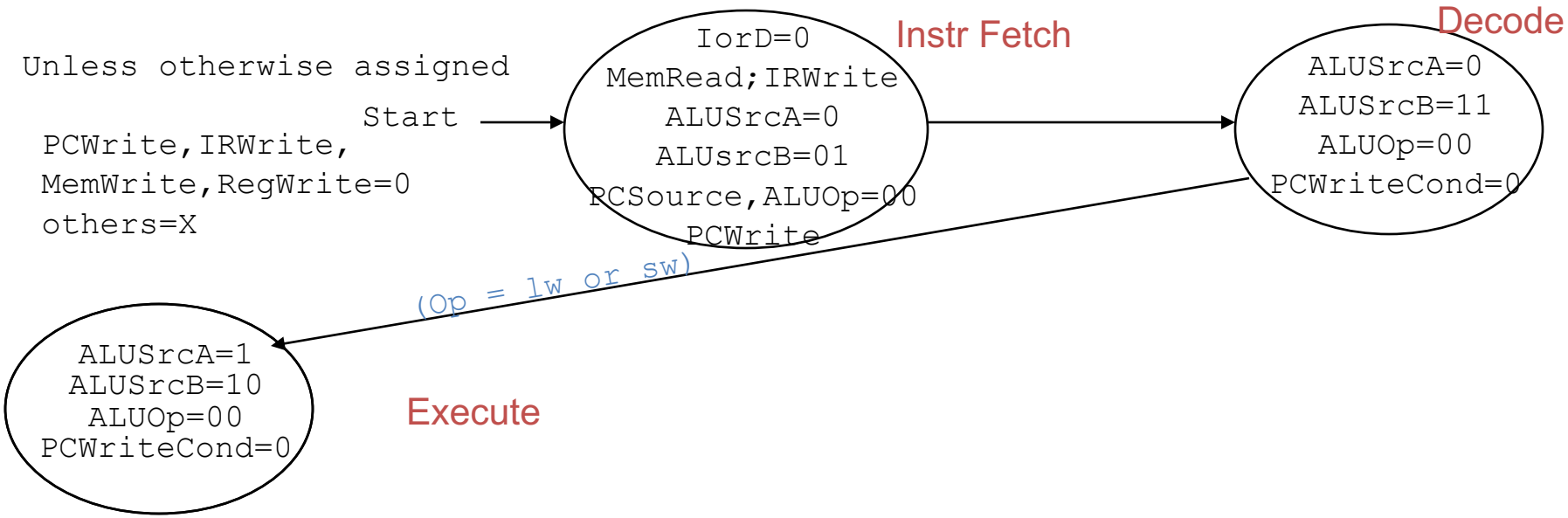
- Jump:

```
PC = PC[31-28] || (IR[25-0] << 2);
```

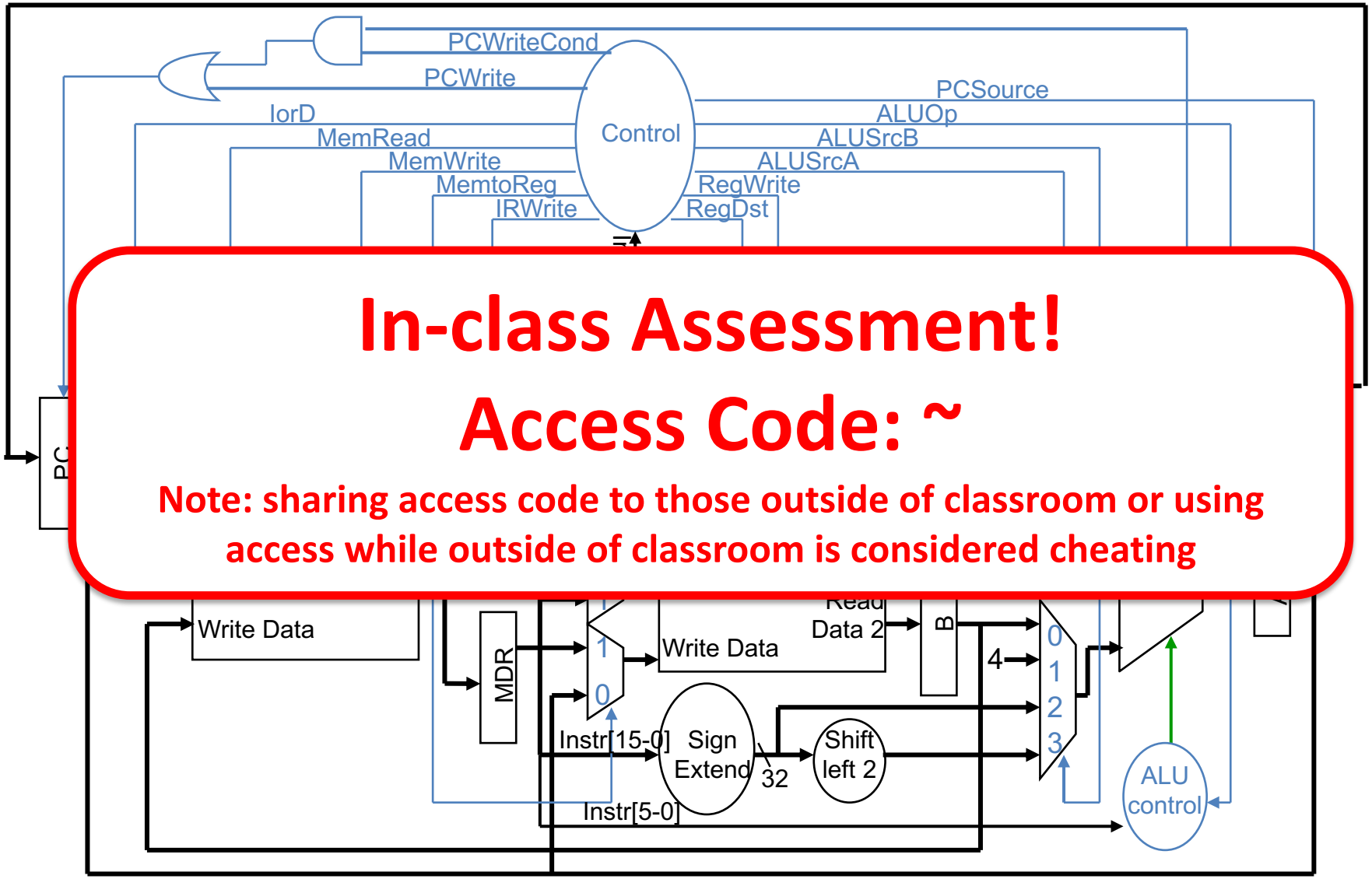
Datapath Activity During `lw/sw` Execute



Execute Control Signals Settings



R-type Execute Control Signals?

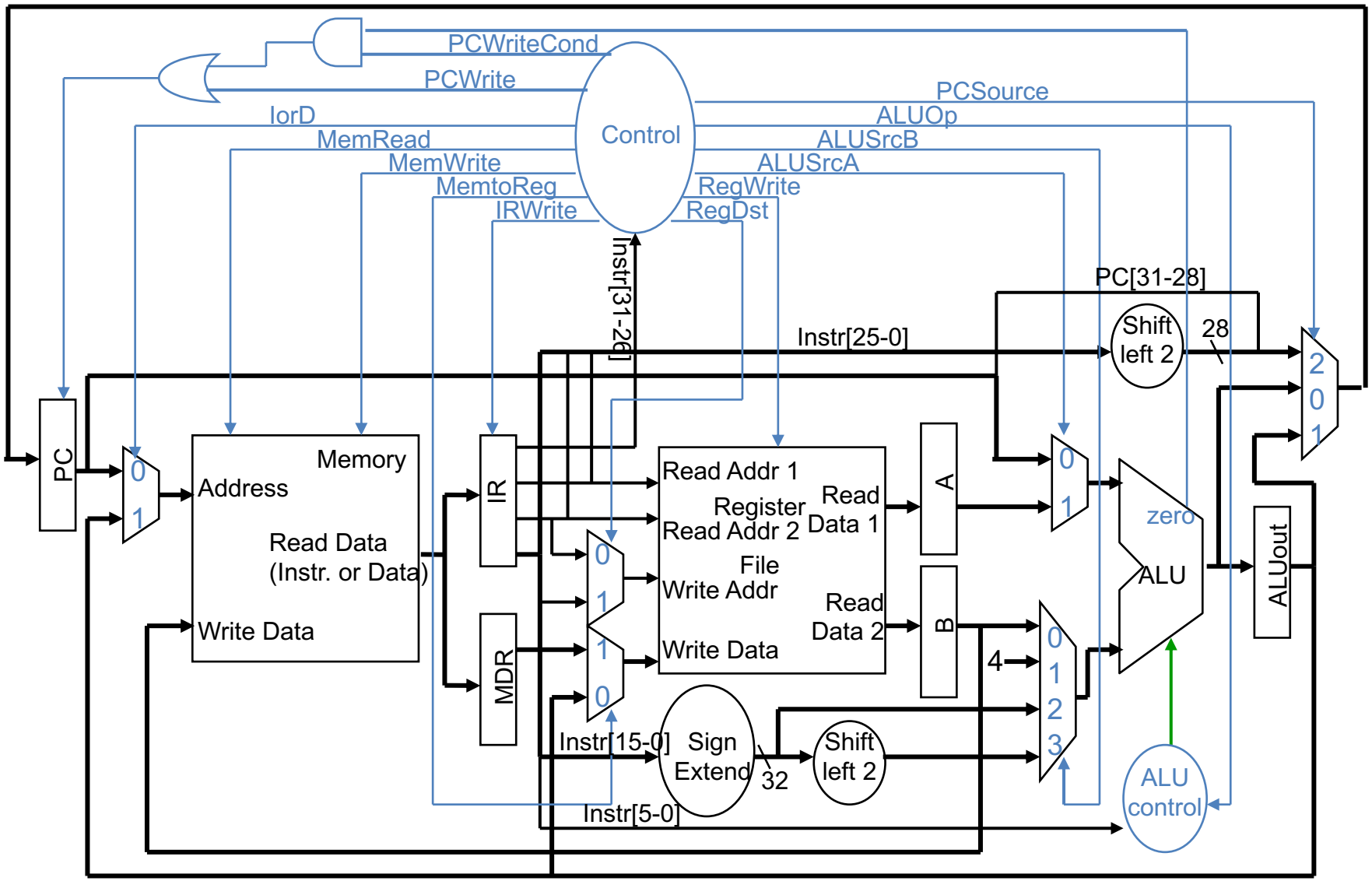


In-class Assessment!

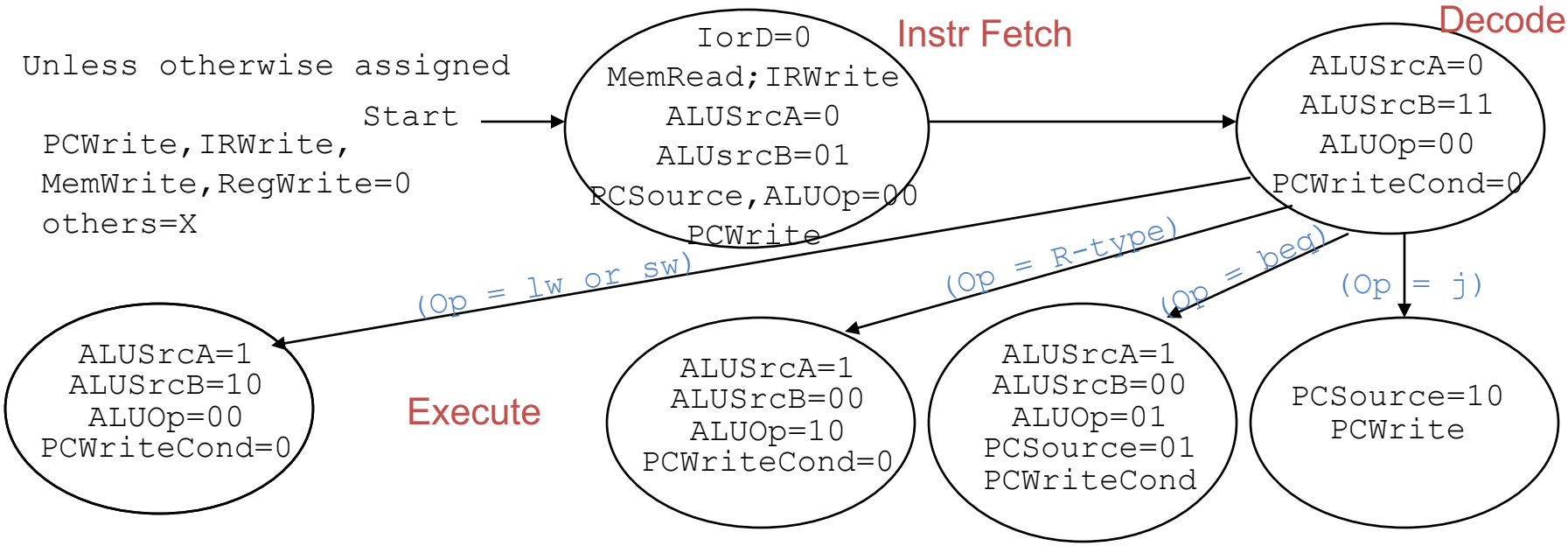
Access Code: ~

Note: sharing access code to those outside of classroom or using access while outside of classroom is considered cheating

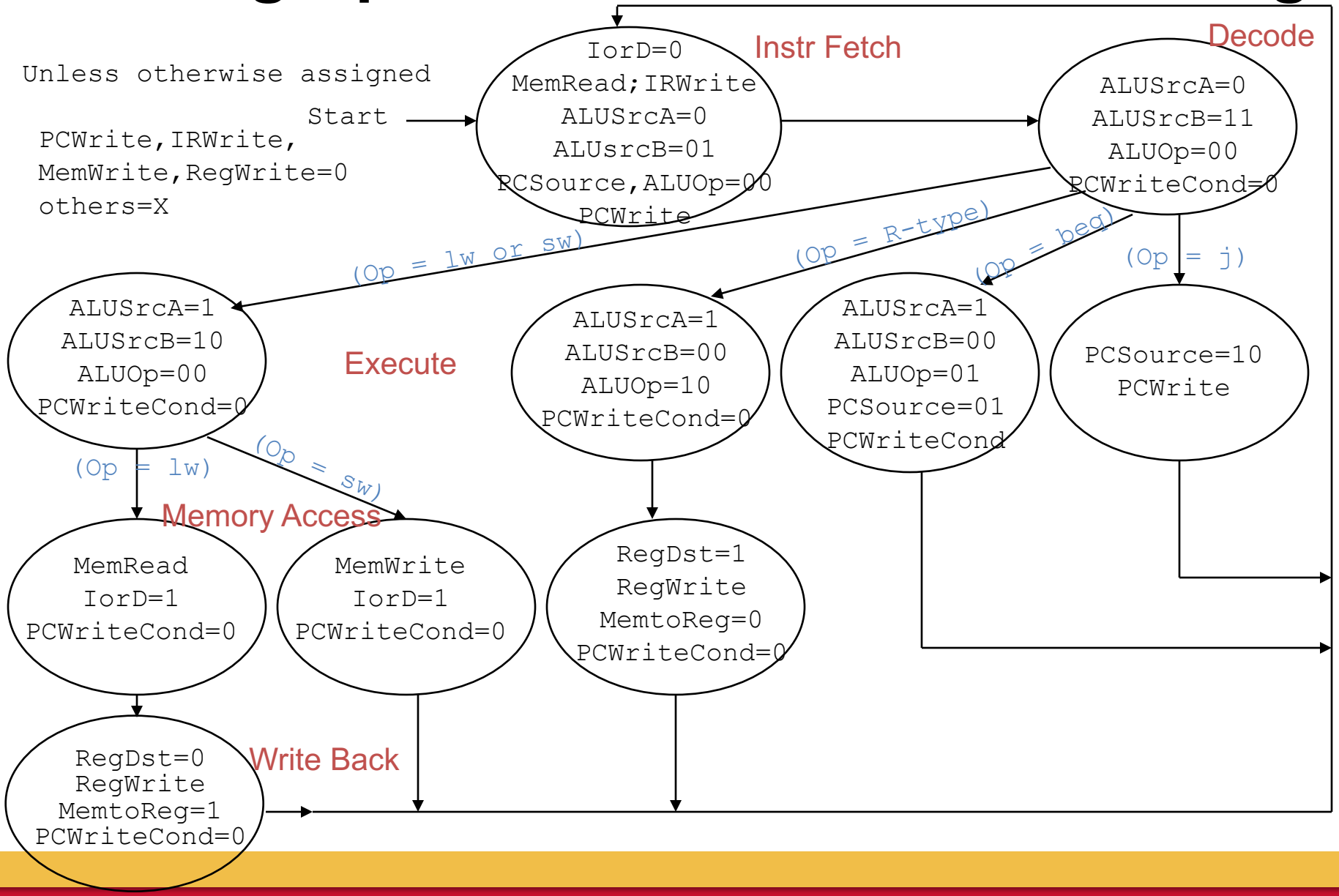
R-type Execute Control Signals



Execute Control Signals Settings



Finishing Up – Write Back Control Settings



Acknowledgments

- These slides contain material developed and copyright by:
  - Joe Zambreno (Iowa State)
  - David Patterson (UC Berkeley)
  - Mary Jane Irwin (Penn State)
  - Christos Kozyrakis (Stanford)
  - Onur Mutlu (Carnegie Mellon)
  - Krste Asanović (UC Berkeley)