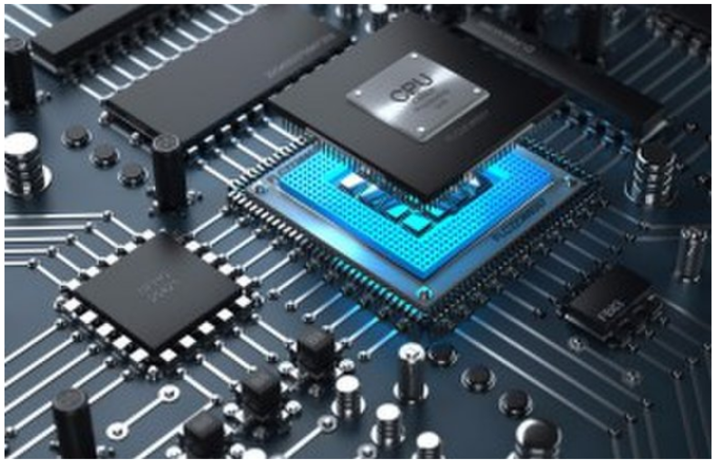


Introduction to CPU Design

CS 154: Computer Architecture
Lecture #10
Winter 2020

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB

Administrative

- Exam on Wednesday, 2/12
- No new lab this week
    - Lab #5 is due on Thursday, 2/13 (by 11:59 PM)

Midterm Exam (**Wed. 2/12**)

What's on It?

- Everything we've done so far from start to Monday, 2/10
- **NO CPU DESIGN MATERIAL IN EXAM!**

What Should I Bring?

- Your pencil(s), eraser, MIPS Reference Card (on **1** page)
- You can bring **1** sheet of hand-written notes (turn it in with exam). 2 sides ok.

What Else Should I Do?

- ***IMPORTANT***: Come to the classroom 5-10 minutes EARLY
- **If you are late, I may not let you take the exam**
- ***IMPORTANT***: Use the bathroom before the exam – once inside, you cannot leave
- Random seat assignments
- Bring your UCSB ID

Lecture Outline

- Some examples using F-P Instructions

- Intro to CPU Design
  - Understanding the Fetch-Execute Cycle in the Hardware

MIPS FP Instructions

	Single-Precision	***Double-Precision***
Addition	`add.s`	`add.d`
Subtraction	`sub.s`	`sub.d`
Multiplication	`mul.s`	`mul.d`
Division	`div.s`	`div.d`
Comparisons Where *xx* can be Example: **c.eq.s**	`c.`*`xx`*`.s` eq, neq, lt, gt,	`c.`*`xx`*`.d` le, ge
Load	`lwc1`	`lwd1`
Store	`swc1`	`swd1`

Also, F-P branch, true (**`bc1t`**) and branch, false (**`bc1f`**)

MIPS FP Instructions

- Programs generally don’t do integer ops on FP data, or vice versa
- FP instructions operate only on FP registers
    - There are 32 FP registers – separate from the “regular” CPU registers
- More registers with minimal code-size impact

The Floating Point Registers

- MIPS has 32 ***separate*** registers for floating point:
    - **`$f0`**, **`$f1`**, etc…

- Paired for double-precision
    - **`$f0/$f1`**, **`$f2/$f3`**, etc…

- Example MIPS assembly code:

```
lwc1 $f4, 0($sp)     # Load 32b F.P. number into F4
lwc1 $f6, 4($sp)     # Load 32b F.P. number into F6
add.s $f2, $f4, $f6  # F2 = F4 + F6 single precision
swc1 $f2, 8($sp)     # Store 32b F.P. number from F2
```

Example Code

C++ code:

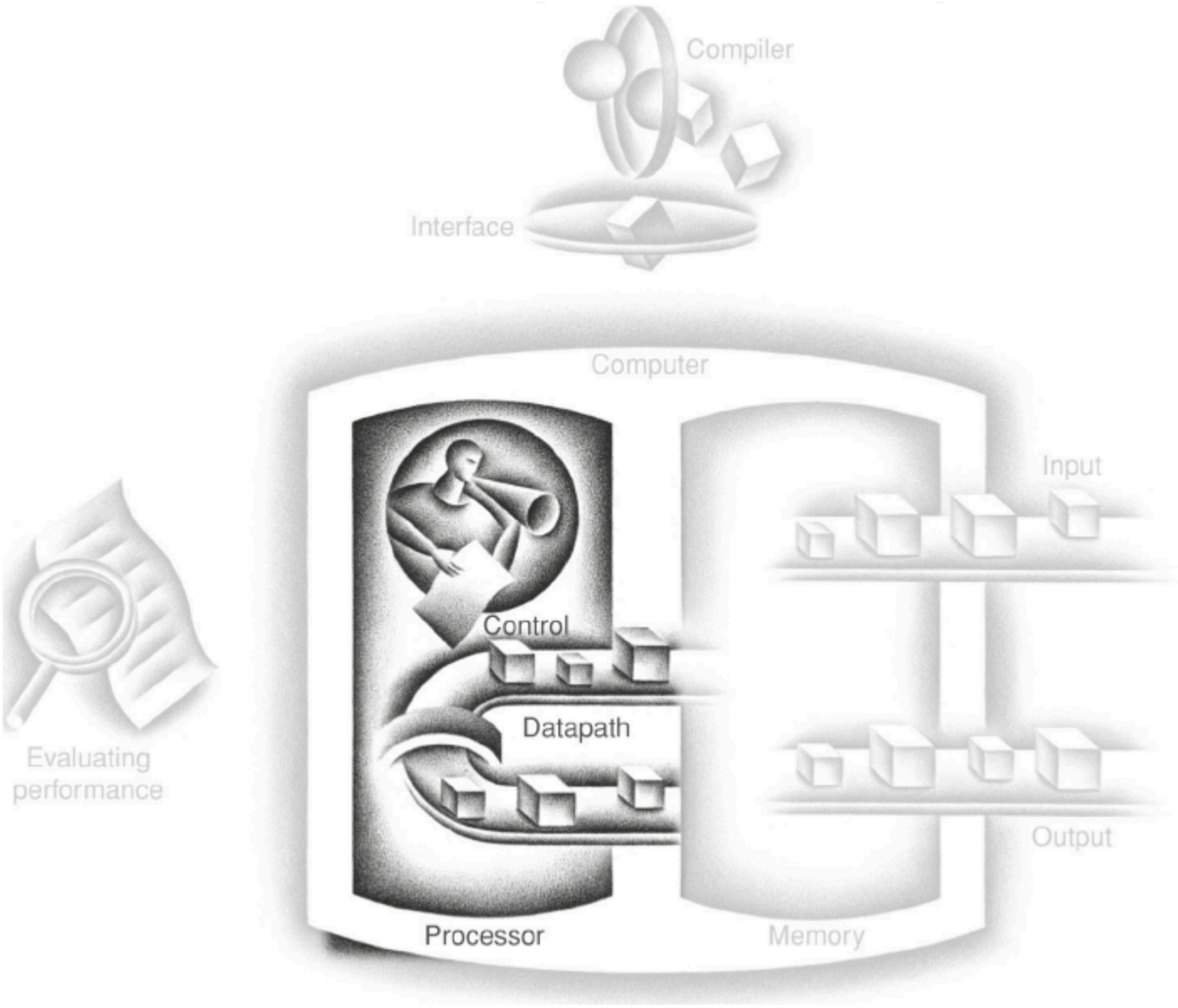
```
  float f2c (float fahr) {
     return ((5.0/9.0)*(fahr - 32.0)); }
```

Assume:

fahr in **$f12**, **result** in **$f0**, constants in global memory space (i.e. defined in **.data**)

Compiled MIPS code:

```
  f2c: lwc1 $f16, const5
       lwc1 $f18, const9
       div.s $f16, $f16, $f18
       lwc1 $f18, const32
       sub.s $f18, $f12, $f18
       mul.s $f0, $f16, $f18
       jr $ra
```


Compiler
Interface
Computer
Evaluating performance
Control
Datapath
Processor
Memory
Input
Output

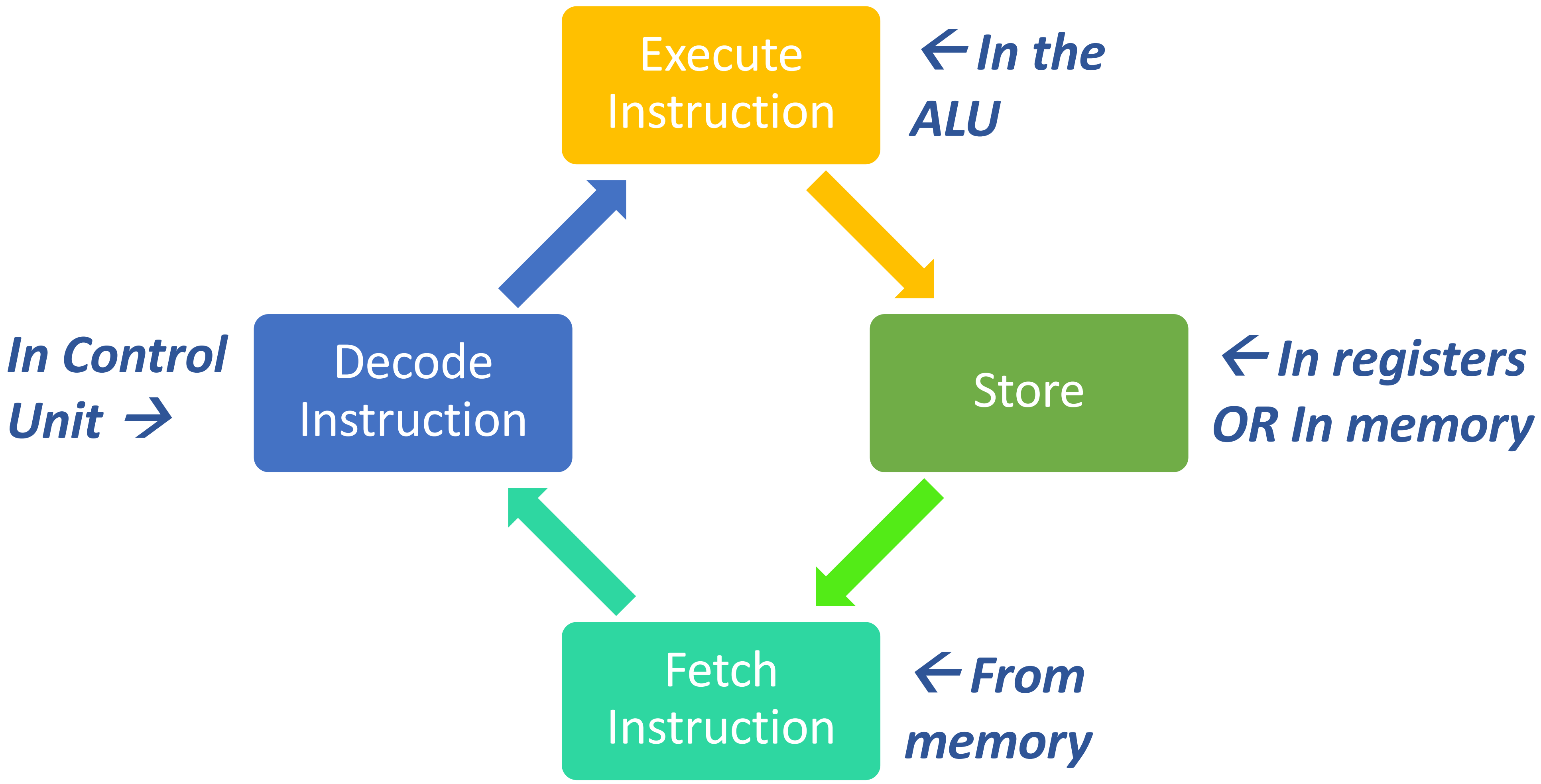
Implementing the Design of a CPU

- CPU performance factors
  - Instruction count: Determined by ISA and compiler
  - CPI and Cycle time: Determined by CPU hardware

- We will examine two MIPS implementations
  - A simplified version
  - A more realistic *pipelined version*

- Simple subset, shows most aspects
  - Memory reference: `lw, sw`
  - Arithmetic/logical: `add, sub, and, or, slt`
  - Control transfer: `beq, j`

The Fetch-Execute Cycle



The Instruction Fetch-Execute Cycle

*For **any** instruction, do these 2 things first:*

1. Send PC to the memory where instruction is & fetch it
2. Read 1 or 2 registers per **rs/rt** codes
   OR
   Read 1 register (for **lw/sw** instructions)

- What happens next depends on the “instruction class”

There are 3 instruction classes:

1. memory-reference
2. arithmetic-logical
3. branches

The Instruction Fetch-Execute Cycle

Depending on instruction class…

- ALU is almost always the next step.
- Use ALU to calculate:
  - Some arithmetic result using Regs
  - Memory address for load/store (again, using Regs)
  - Branch target address (*not* so much using Regs)

- Then, the different instruction classes need different things done…

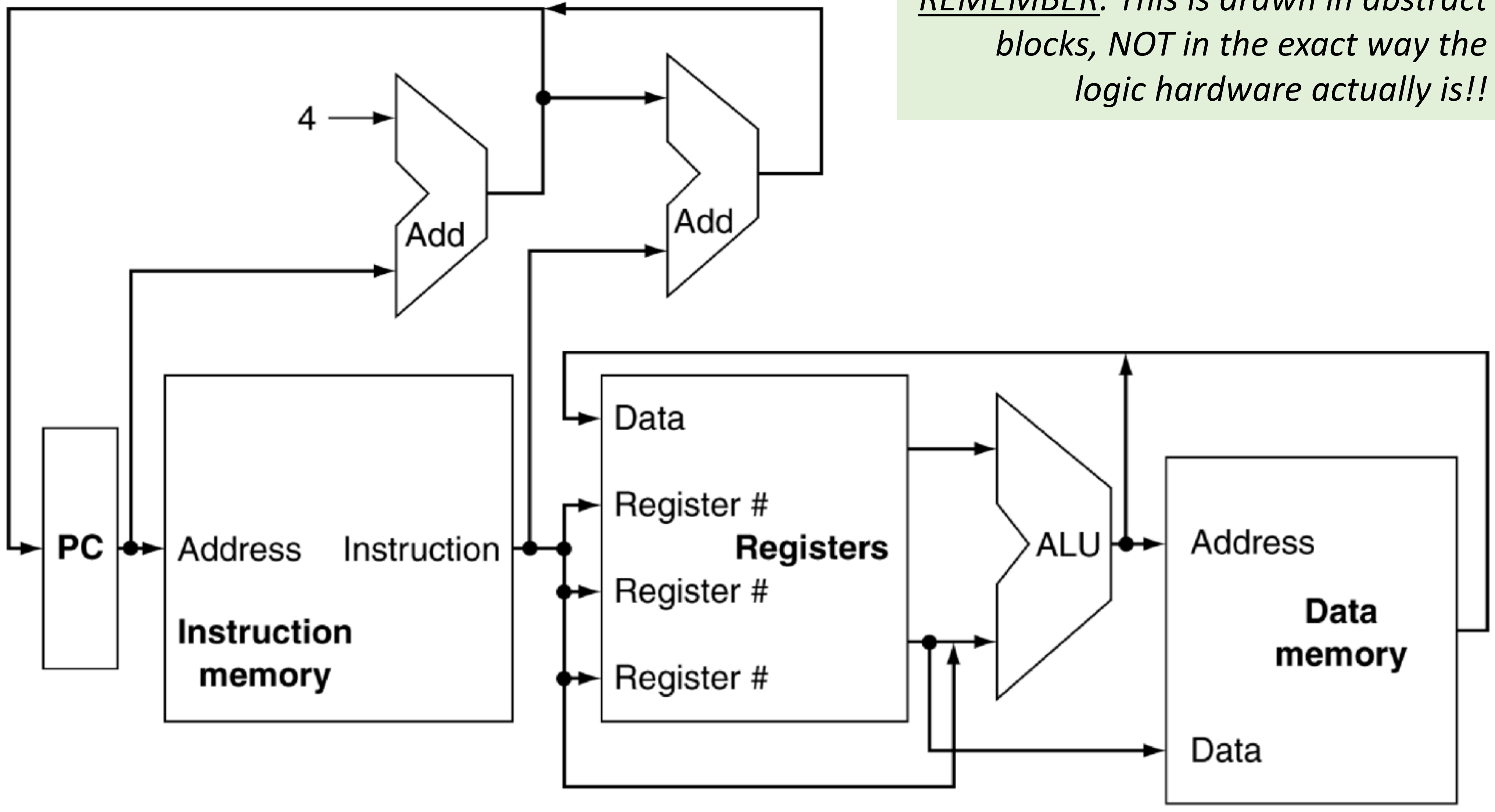
The Instruction Fetch-Execute Cycle

Per the instruction class…

- Memory-reference type:
  - Access data memory for load/store
- Arithmetic-Logical (or load instruction)
  - Write data *from* the ALU *or* memory *back into* a register
- Branching
  - Change next instruction address based on branch outcome
  - Otherwise, the PC = PC + 4

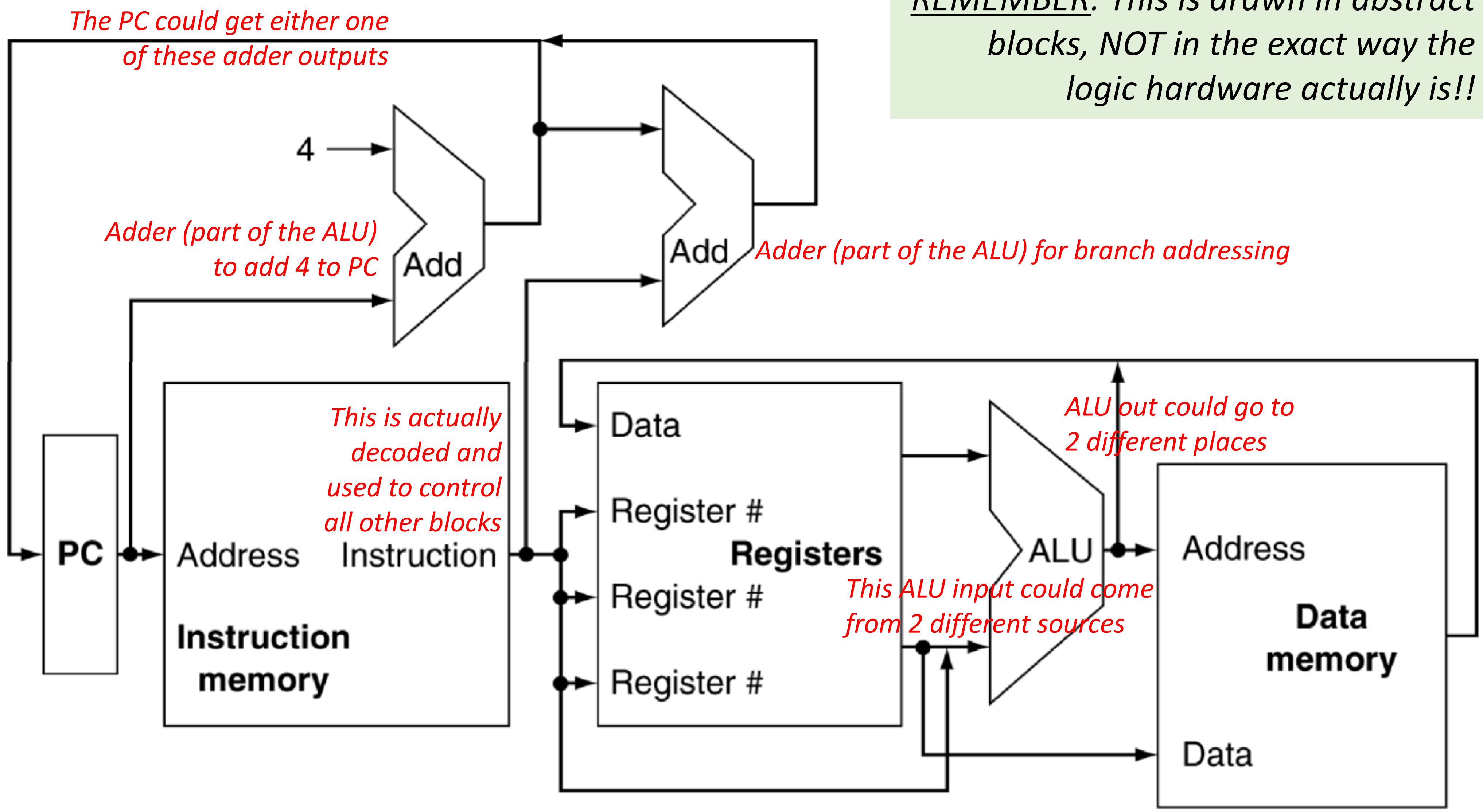
General (and Simplified) CPU Hardware Design

<u>REMEMBER</u>: This is drawn in abstract blocks, NOT in the exact way the logic hardware actually is!!

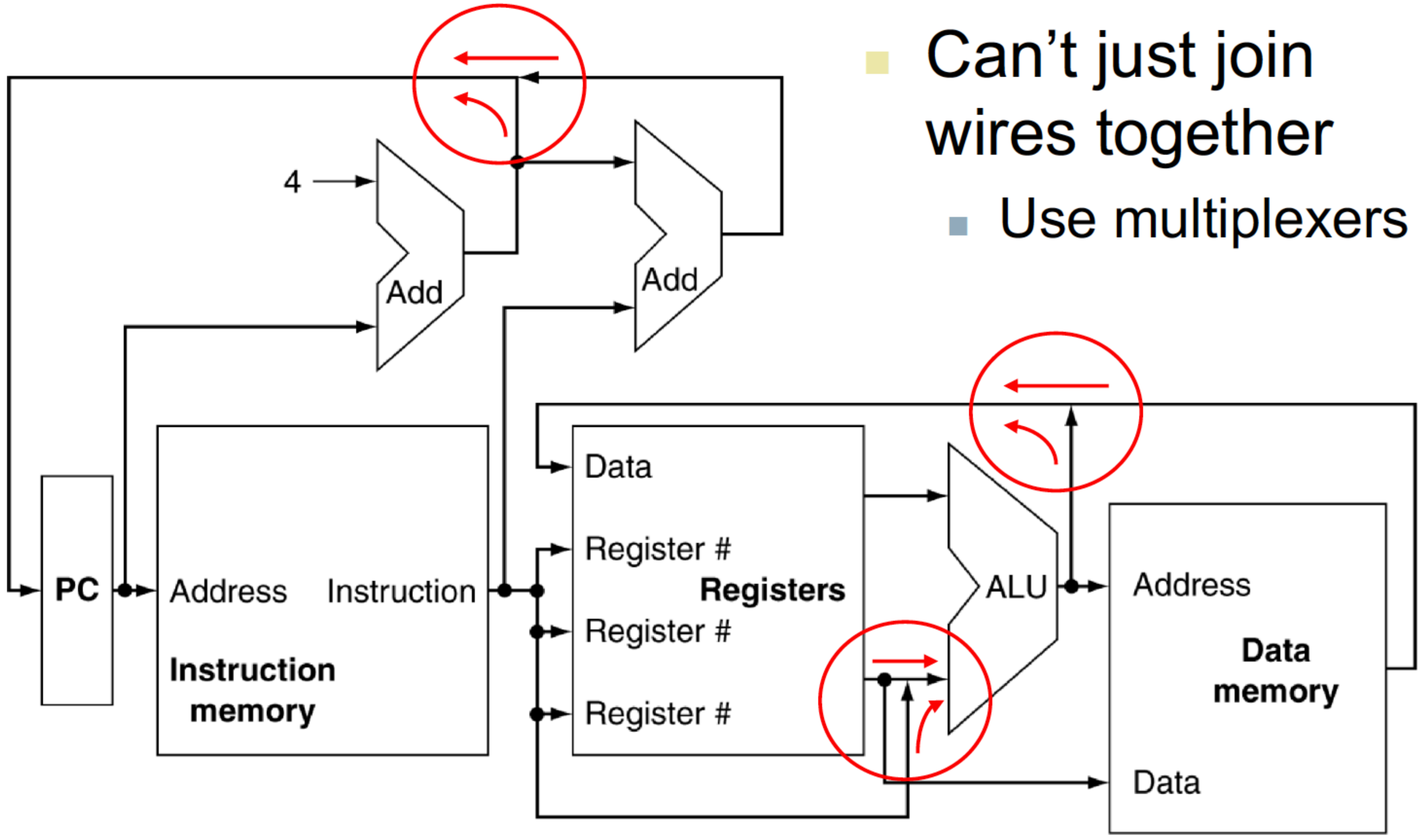


General (and Simplified) CPU Hardware Design

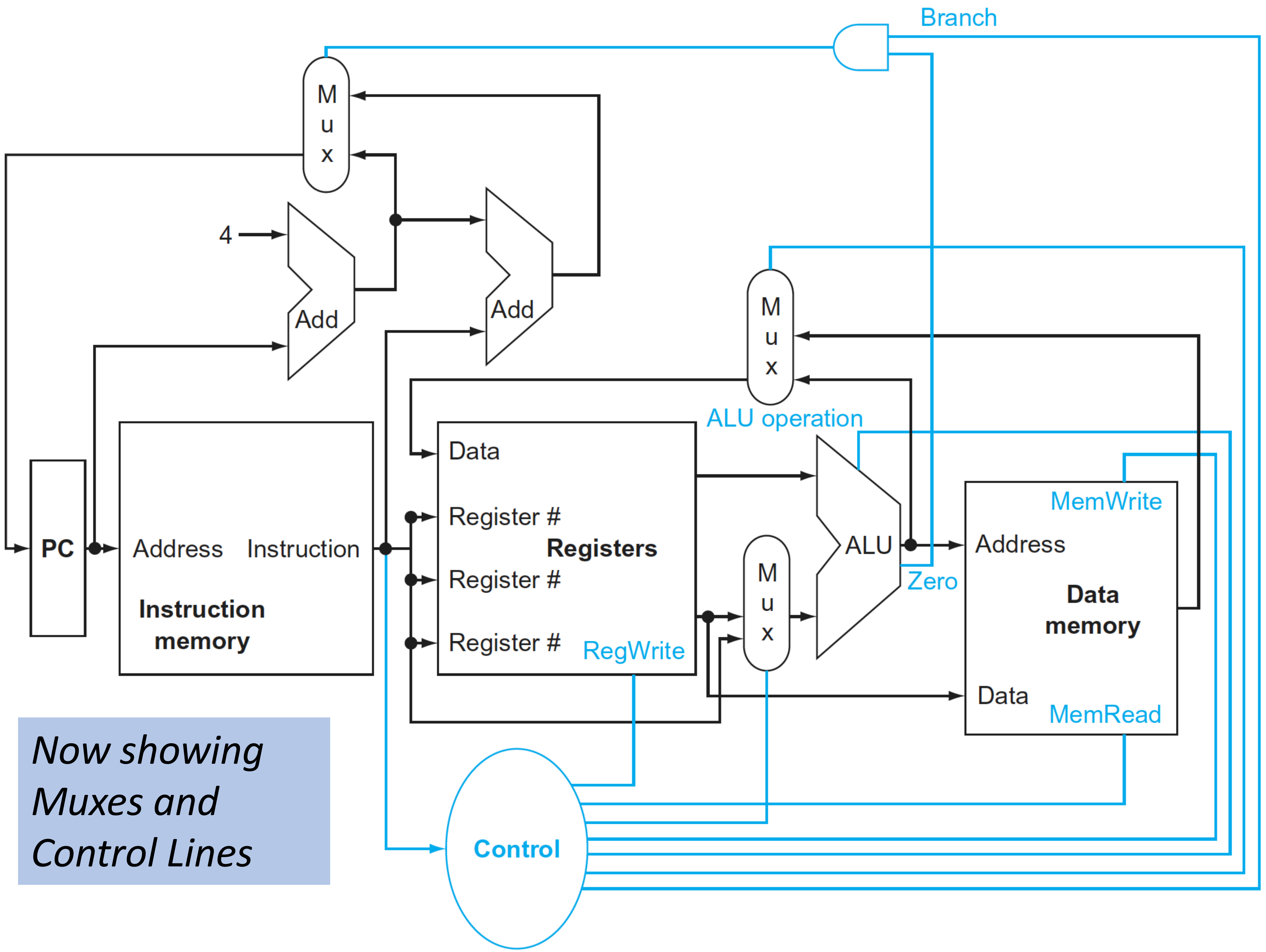
REMEMBER: This is drawn in abstract blocks, NOT in the exact way the logic hardware actually is!!


The PC could get either one of these adder outputs
4
Add
Adder (part of the ALU) to add 4 to PC
Add
Adder (part of the ALU) for branch addressing
PC
Address
Instruction
Instruction memory
This is actually decoded and used to control all other blocks
Data
Register #
Registers
Register #
Register #
ALU
ALU out could go to 2 different places
This ALU input could come from 2 different sources
Address
Data memory
Data

A Little More Detail… (Remember Multiplexers?)



- Can't just join wires together
  - Use multiplexers



Now showing Muxes and Control Lines

YOUR TO-DOs for the Week

- Study for the midterm!

- Current lab due on Thursday
- No new Lab this week!

- Next week:
  - NO CLASS ON MONDAY (University Holiday)
  - Wednesday (2/19) we resume CPU Design (Ch. 4)

</LECTURE>