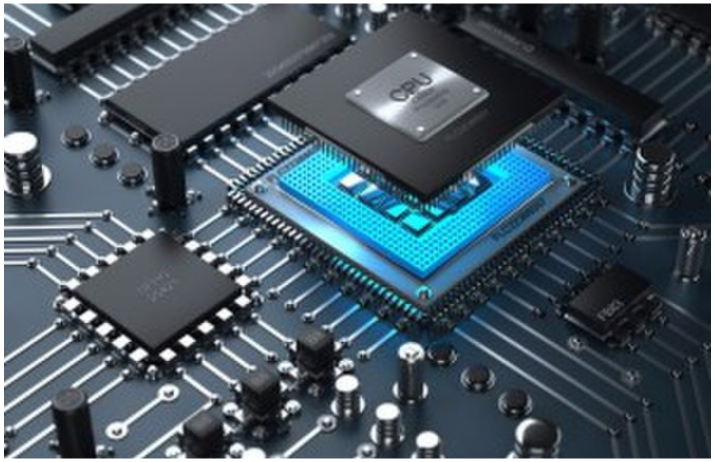


CPU Datapaths 1

CS 154: Computer Architecture

Lecture #11

Winter 2020

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

Administrative

- Exam grades will be announced by the weekend

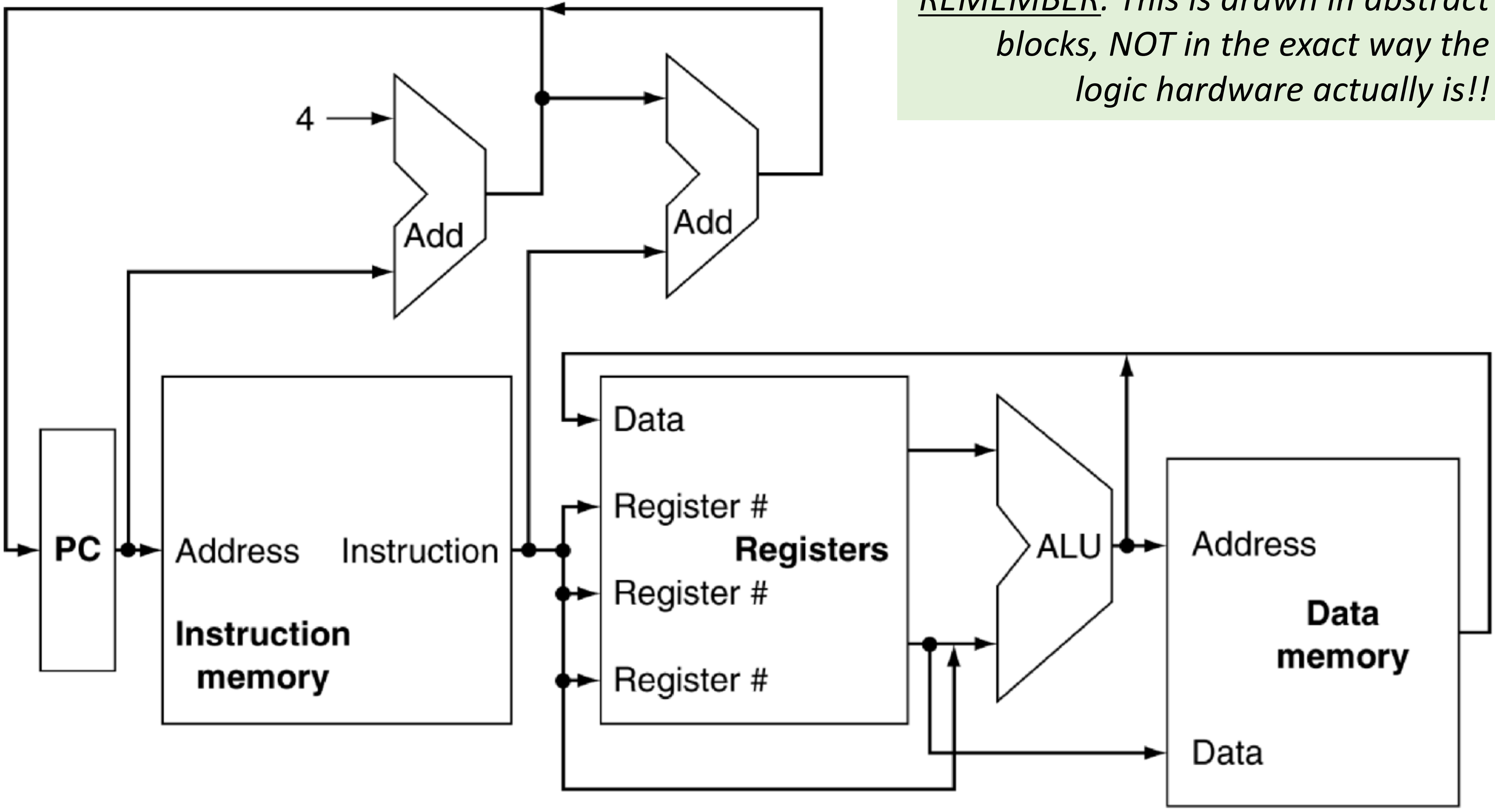
- New lab this week (Lab 6)

Lecture Outline

- Logic Design Refresher
- Datapaths

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!!

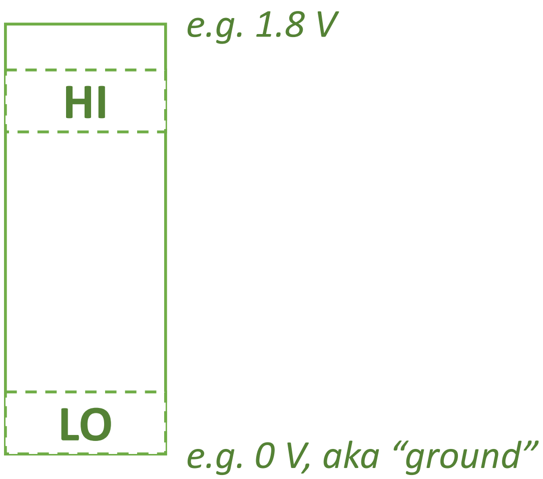


Information Encoding

- At the very basic level, an electronic logic circuit has:
  - Input(s) ***deal with logic levels of "0" and "1"***
  - Output(s) ***and nothing else***
  - Power Supply *it's how the circuits actually work*

- Power supply (i.e. a battery) voltage range allows for 2 distinct "levels" within that range
  - Low voltage ≡ Logic **0**
  - High voltage ≡ Logic **1**

- One wire per bit
- Multi-bit data encoded on multi-wire "**buses**"



e.g. 1.8 V

HI

LO

e.g. 0 V, aka "ground"

The 2 Types of Logic Building Blocks

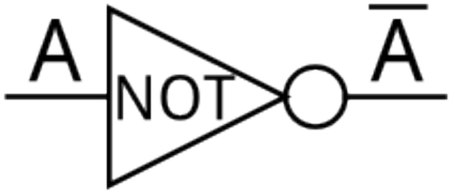
- **Combinatorial Logic**
  - Output is a direct "result" of a combination of inputs
  - Input changes mean output changes *immediately**

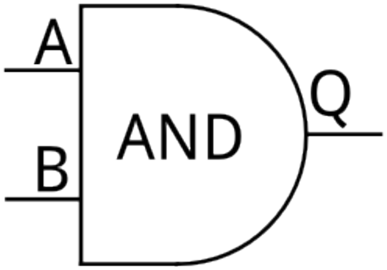
- **Sequential Logic**
  - Output can *optionally*** remain unchanged
  - It means that this kind of logic can keep an old output the same *even if the input(s) change(s)!*
  - In other words, these circuits have "memory" of "old states"

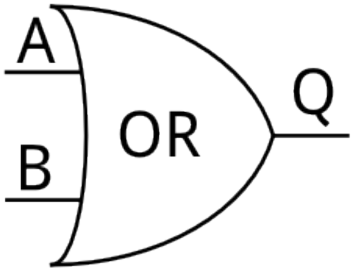
* *In the **physical world**, there is no such thing as "change occurring immediately". The use of this term here just means "practically immediately".*

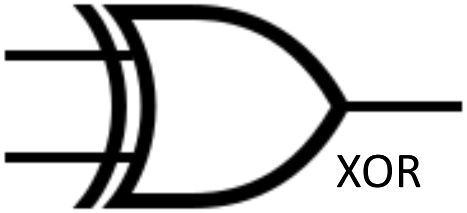
** *This means the output must rely on a "special" input to **enable** the output to change.*

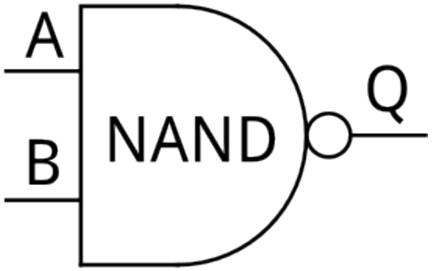
Graphical Symbols of Basic Combinatorial Logic Elements

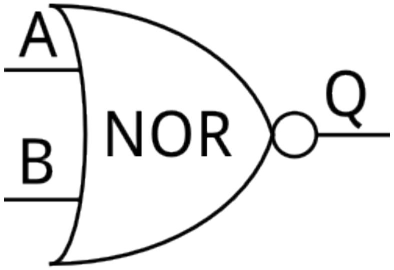


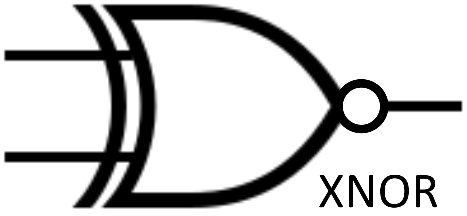






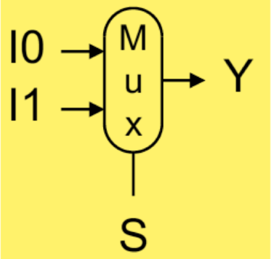




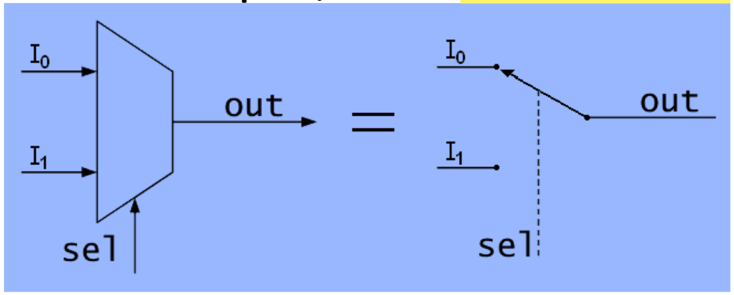


Multiplexers (Muxes)

- Combinatorial circuits who function as a “chooser” between multiple inputs to be “driven to” the output
- Always multiple inputs (N), always ONE output (N-to-1 mux)
  - Can be drawn symbolically in 2 ways (trapezoid vs oval)
    --- there’s NO difference, just a preference in drawing



- 1 of the input data lines gets selected to become the output, based on a 3rd “select” (sel) input
  - If **sel** = 0, then I_0 gets to be the output
  - If **sel** = 1, then I_1 gets to be the output
  - So: **OUTPUT = I_1.sel + $I_0.\overline{\text{sel}}$**



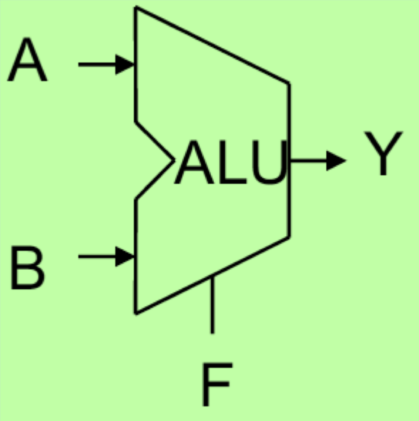
- The opposite of a Mux is called a **Demulitplexer** (or **Demux**)

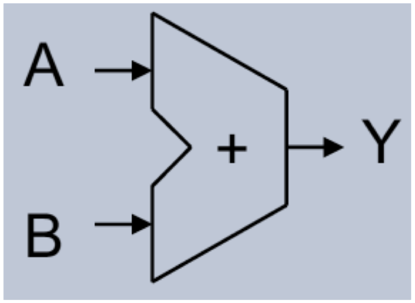
ALUs and Adders

- Combinatorial logic

- ALU takes 2 bus (i.e. multi-bit) inputs and outputs 1 bus, depending on what function (F) is chosen
  - Y = F(A,B)

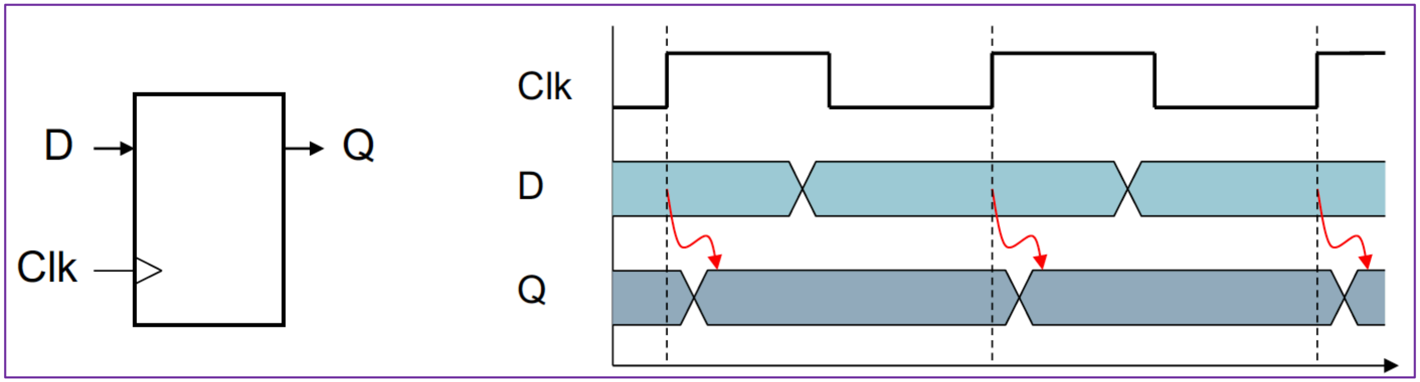
- Adder is really a sub-set of any ALU – we sometimes want to symbolize it separately
  - Y = A bit-add B (i.e. A + B)





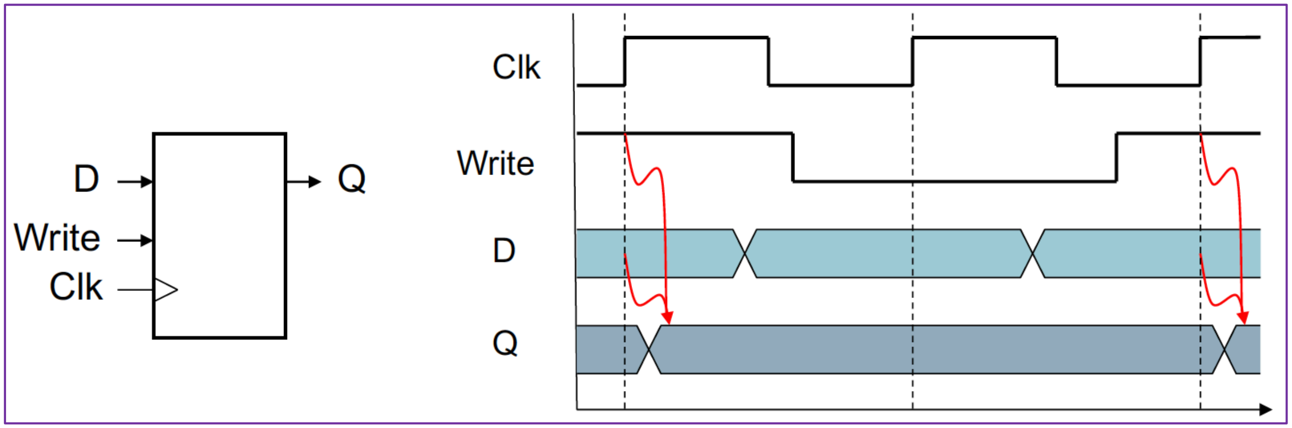
Registers

- Sequential Logic
- Input passes on to the output ONLY IF enabled
    - Otherwise, the output remains in its “old” state
- Typically, registers are D-Flip Flop circuits
    - Uses a ***clock signal*** to determine when to update the stored value
    - ***Edge-triggered***: update when clock signal changes from 0 to 1



Registers with Write Control

- Sometimes registers have an additional enabling mechanism: a WRITE-ENABLE input
- Only updates on clock edge when write-enable control input is 1
- Useful if we don’t need to write something right away
  - You can’t “mute” the clock signal because it is a global one
  - WRITE-ENABLE is a local signal only to *that particular* register

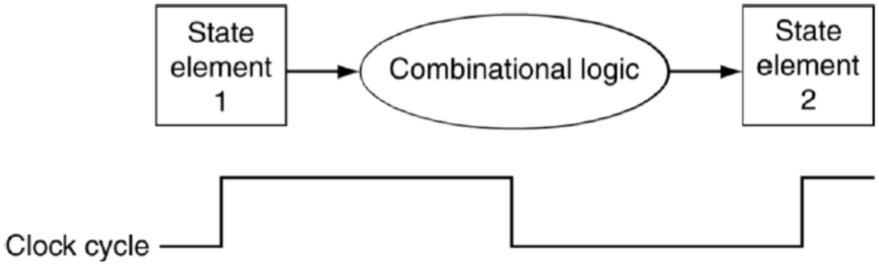


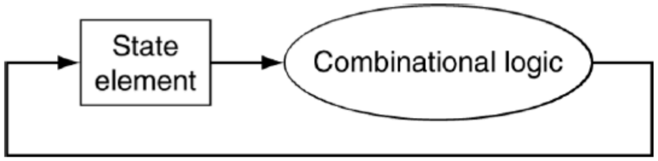
Clocking Methodology

- The same clock signal is propagating to multiple parts of the CPU

- Combinational logic transforms data during clock cycles
  - i.e. *between* clock edges
  - Input from state elements, output to state element

- Longest delay determines clock period
  - Or better said: clock period determines how long the longest delay can be





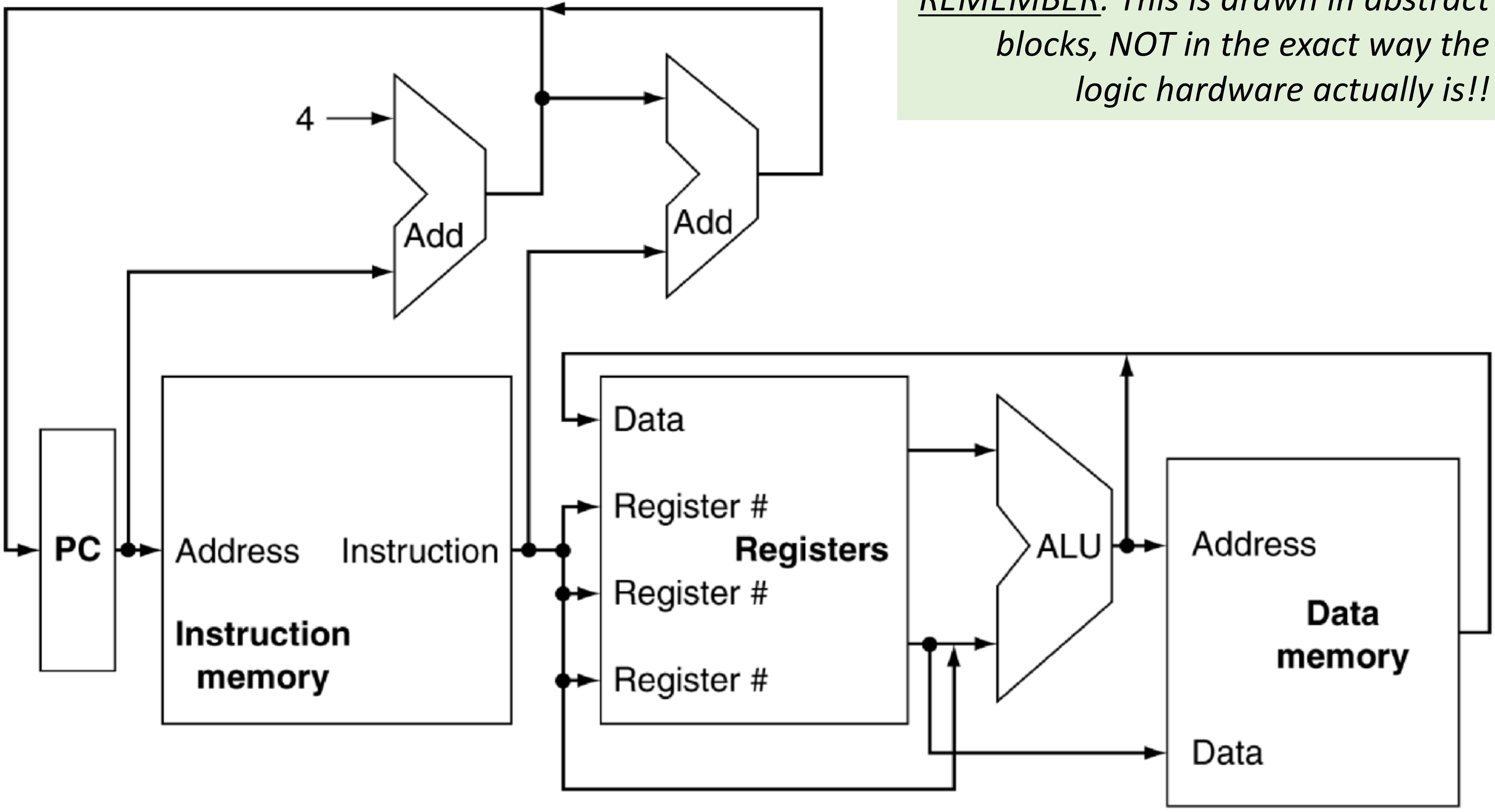
Building a Datapath

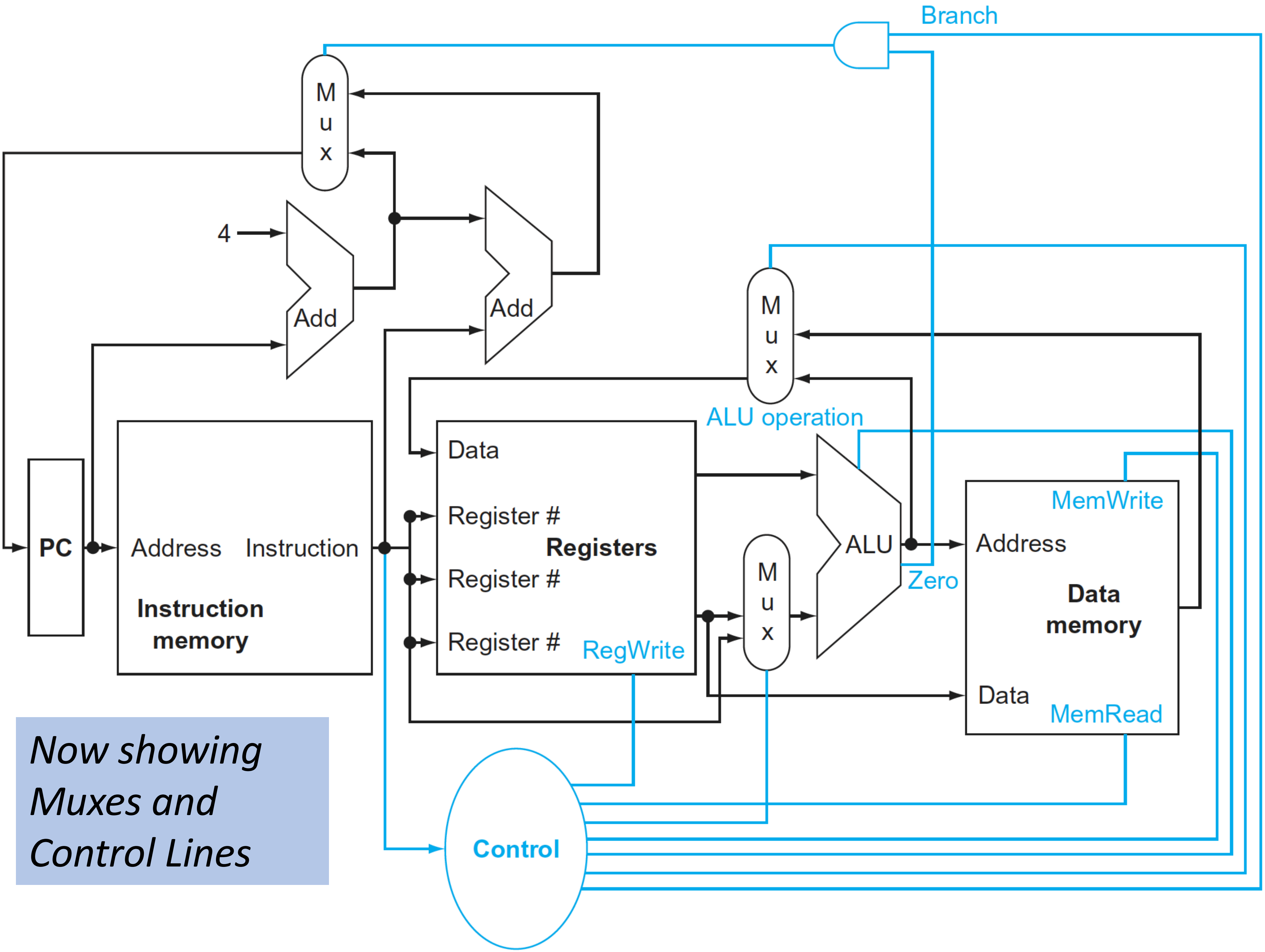
- **Datapath**: Elements that process data and addresses in the CPU
  - Example: Registers, ALUs, muxes, memories, …

- We will build a MIPS datapath incrementally
  - Refine the overview "simple" design

Simplified CPU Design

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





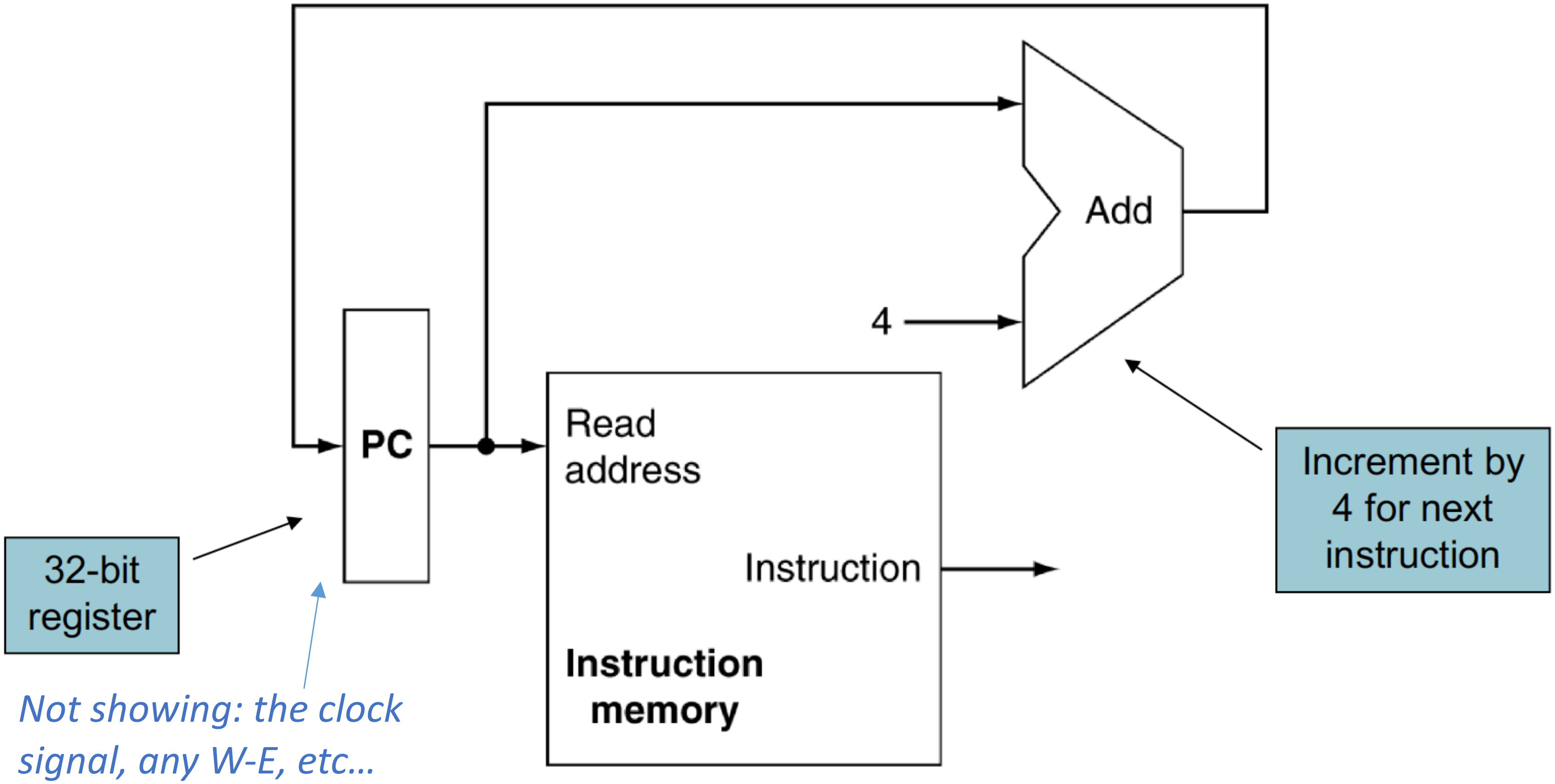
Now showing Muxes and Control Lines

Ok…

Let’s Review it

One Instruction Type at a Time…

First: The Instruction Fetch



32-bit register

Not showing: the clock signal, any W-E, etc…

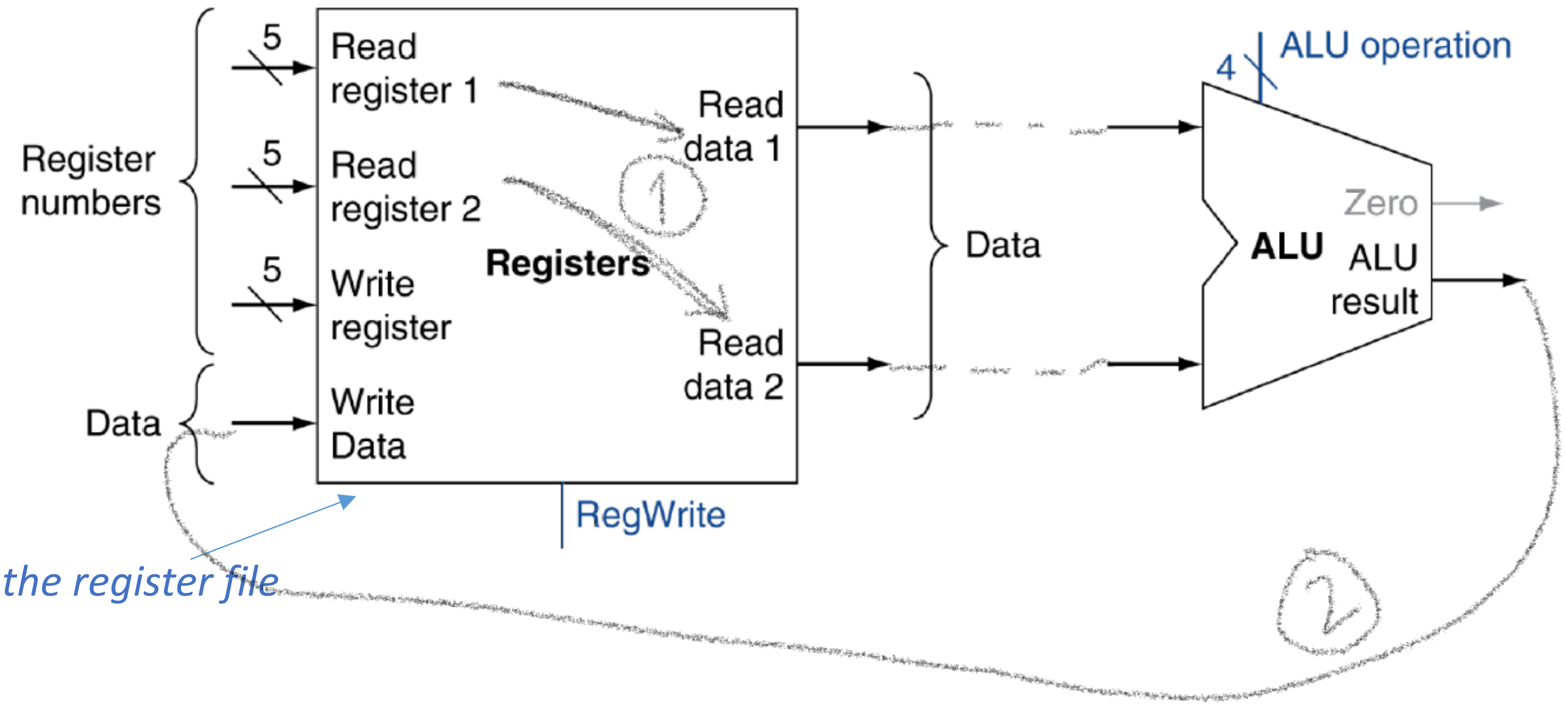
Increment by 4 for next instruction

1. To execute ***any*** instruction, we must start by fetching the instruction from memory.
2. To prepare for executing the *next* instruction, we must also increment the PC so that it points at the next instruction, 4 bytes later.

R-Format Instructions: 3 Steps

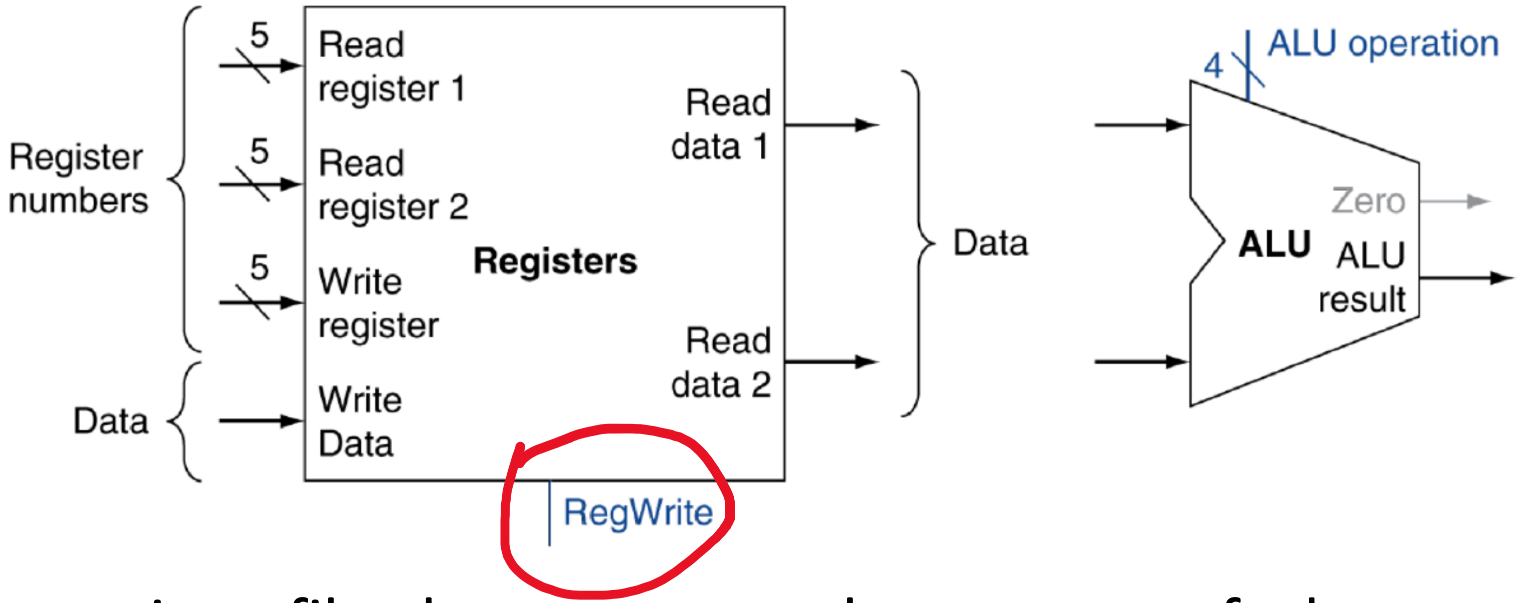
includes add, sub, and, or, slt
e.g.: ***`add $t1, $t2, $t3`***

- Read **two** register operands ($rs, $rt)
- Perform arithmetic/logical operation
- Write register result ($rd)



This is the register file

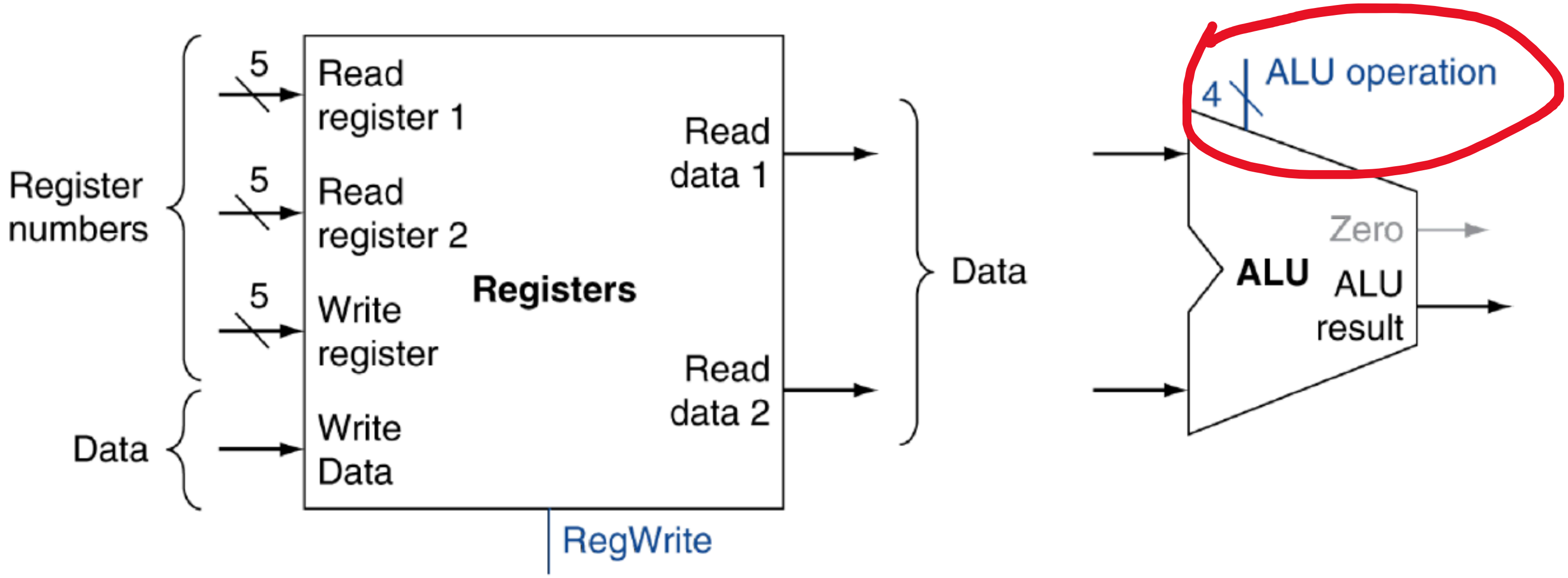
Register File Read and Write



- The register file always outputs the contents of whatever register numbers are on the Read register inputs.

- Writes are controlled by the write control signal, which must be *asserted* for a write to occur at the clock edge.

The ALU Operation



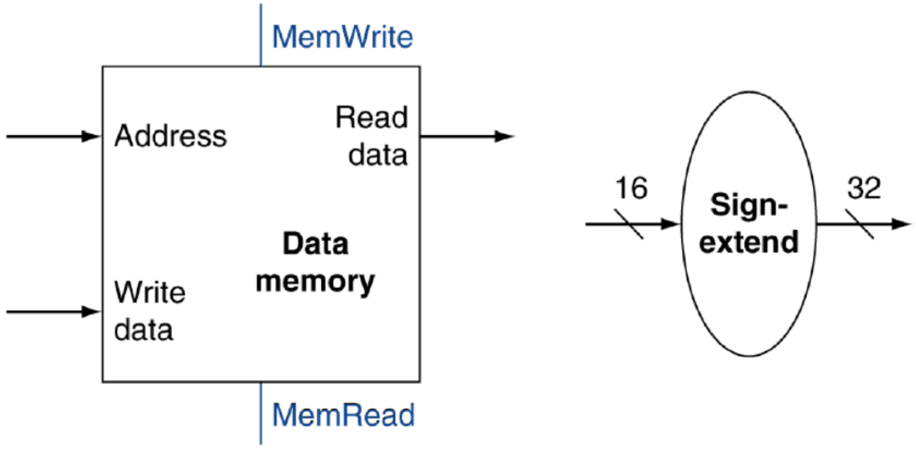
- The ALU takes two 32-bit inputs and produces a 32-bit result
- Also produces a 1-bit signal if the result is 0.
- The 4-bit control signal of the ALU (“ALU operation”) tells it what op it’s performing on the inputs
  - It’s a decoded set of bits… more on those later

Load/Store Instructions

includes lw, sw
e.g.: `lw $t0, 4($sp)`

- Read register operands
- Calculate address using 16-bit offset (immediate)
  - First take the offset and sign-extend it to 32-bits
  - Then use ALU
- Load: Read memory and update register
- Store: Write register value to memory

We'll also need…



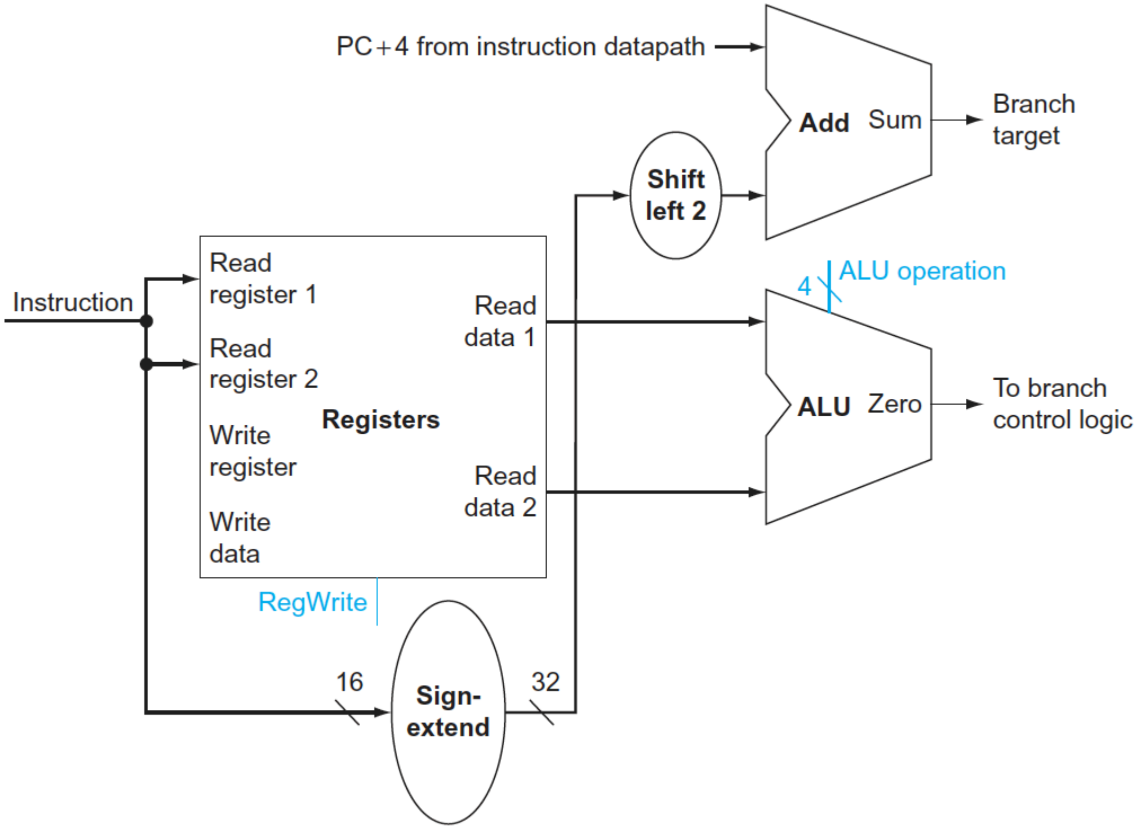
a. Data memory unit

b. Sign extension unit

Branch Instructions

includes beq, bne
e.g.: ***`beq $t1, $t2, Label`***

- Read register operands
- Compare operands
  - Use ALU, subtract
    and check Zero output
- Calculate target address
  - Sign-extend displacement
  - Shift left 2 places
  - Add to PC + 4
    (already calculated
    by instruction fetch)



Next Time… Putting them all Together

YOUR TO-DOs for the Week

- Lab 6 will be up soon…
- Assembly programming assignment

</LECTURE>