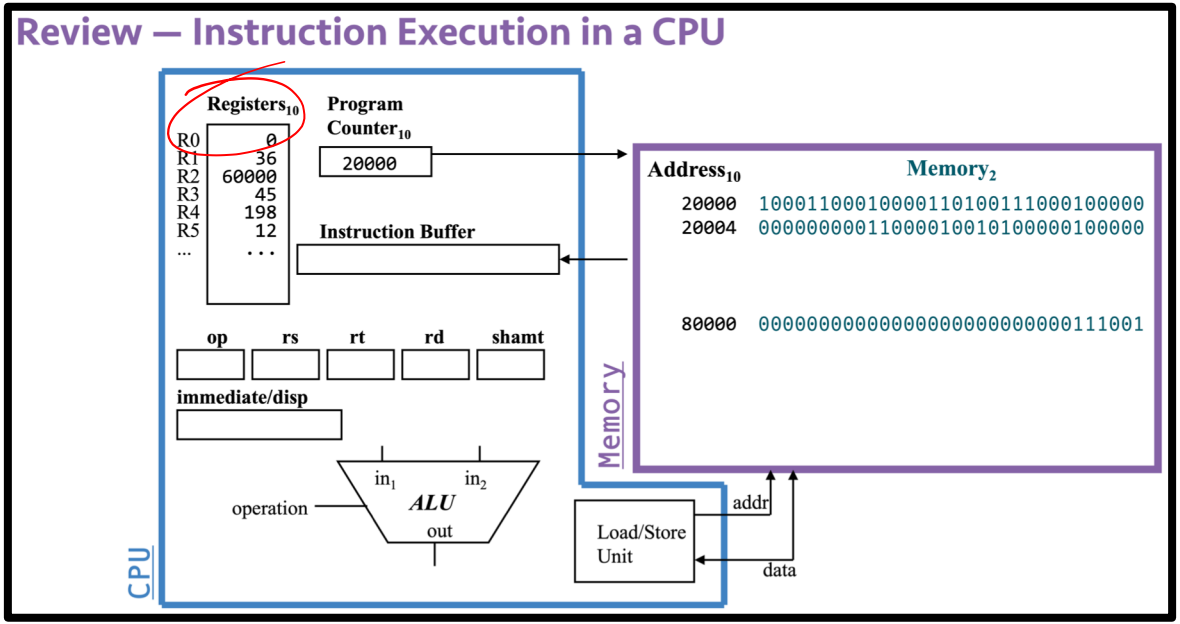
Our previous view of a computer had no organization

- From Part I…



Think about how a MIPS machine executes instructions… Which correctly describes the *order* things must happen in?

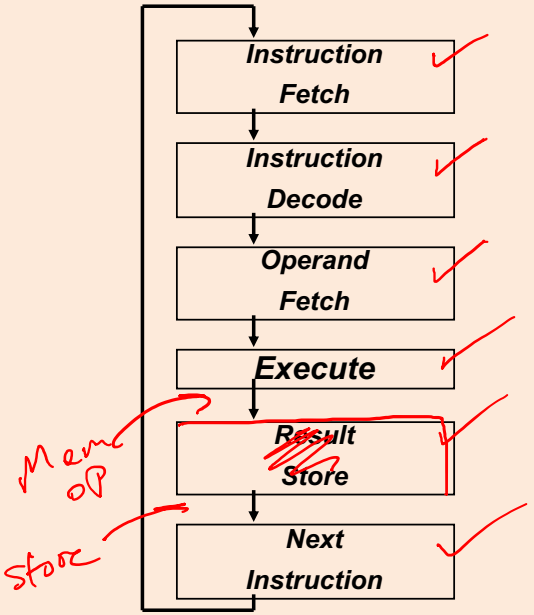
A. The ALU *always* performs an operation before accessing data memory

B. The ALU *sometimes* performs an operation before accessing data memory

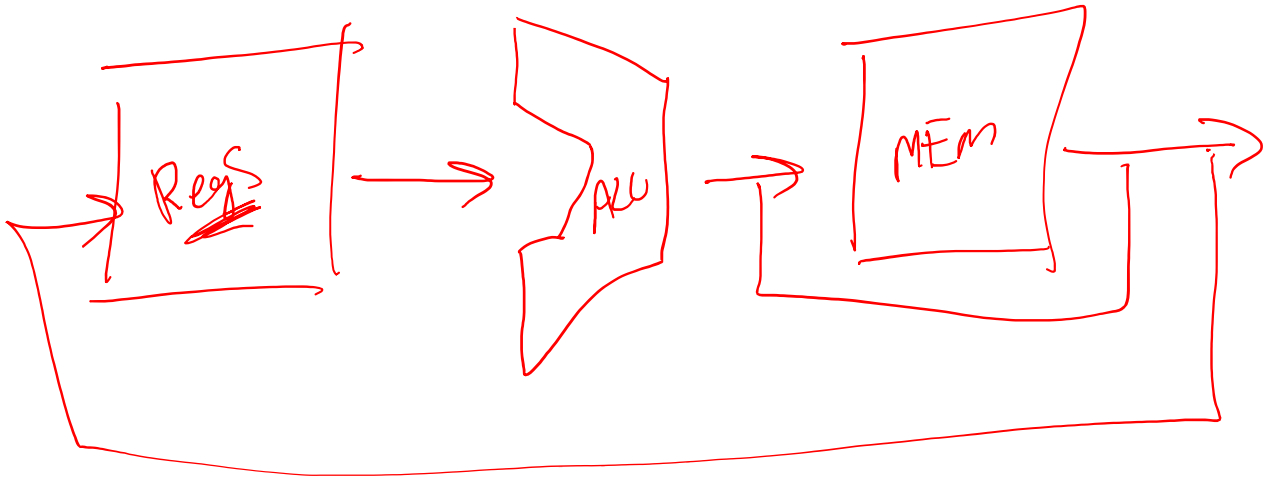
C. Data memory is *always* accessed ~~before~~ after performing an ALU operation

D. Data memory is *sometimes* accessed before performing an ALU operation

E. None of the above.

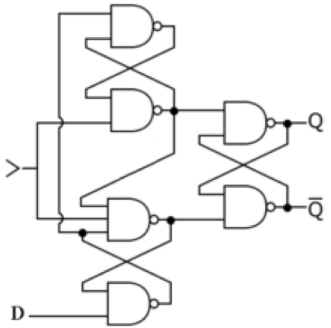


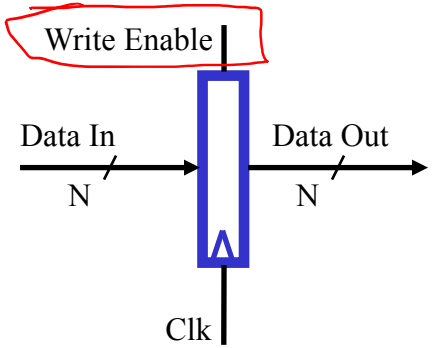
So what does this tell us about what the machine might look like?



Storage Element: Register

- Review: D Flip Flop
- New: Register
  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input
  - Write Enable:
    - 0: Data Out will not change
    - 1: Data Out will become Data In (on the clock edge)

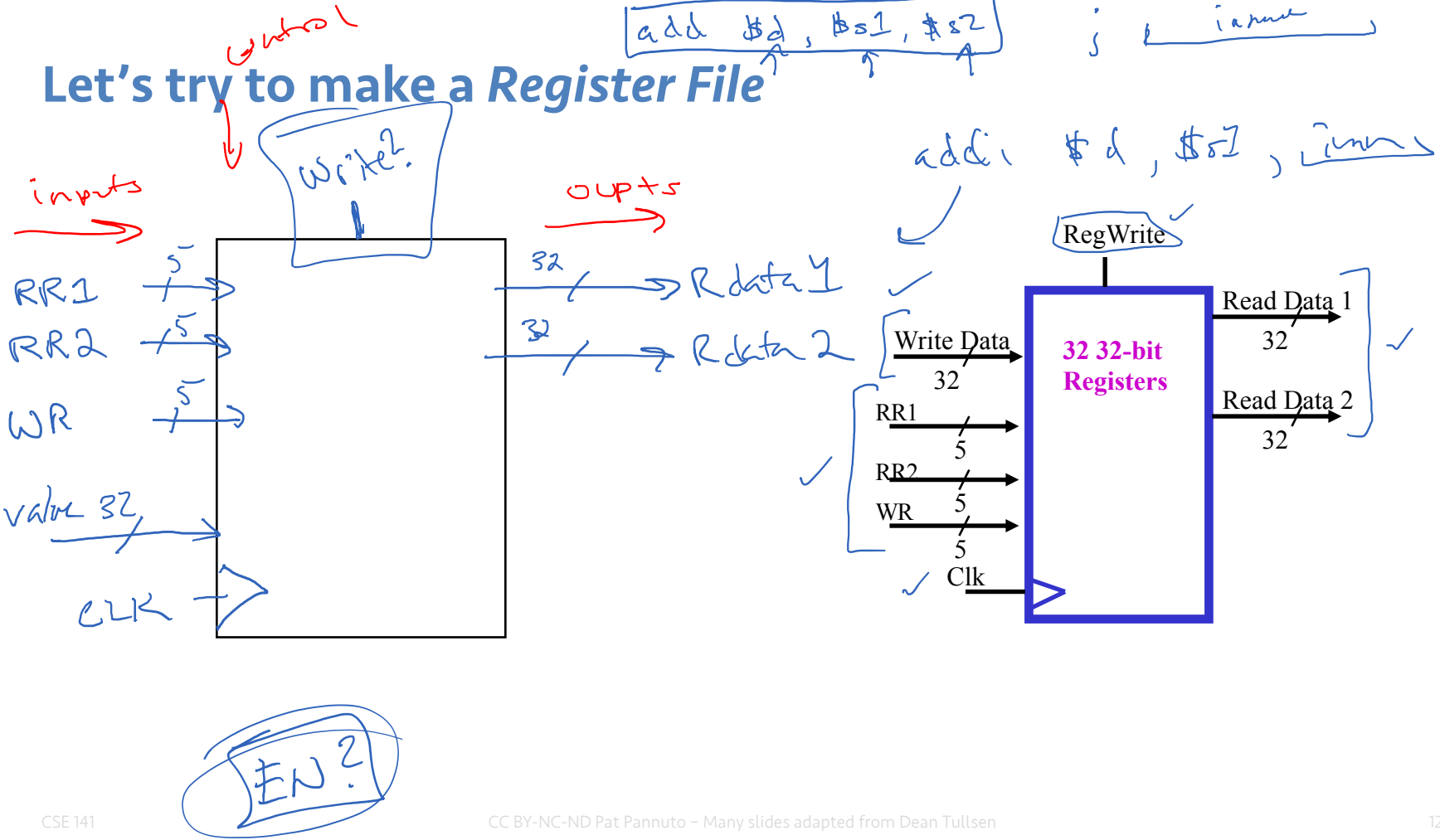




A *register file* is a structure that holds many registers. What kinds of signals will we need for our MIPS register file?

	Number of bits for register output	Number of bits for register selection	Control Inputs?	Control Outputs?
A	5	32	clk	read/write
B	5	5	clk, read/write	clk
C	32	5	clk, read/write	(none)
D	32	32	clk, read/write	clk, read/write
E	32	5	read/write	(none)

Let's try to make a *Register File*

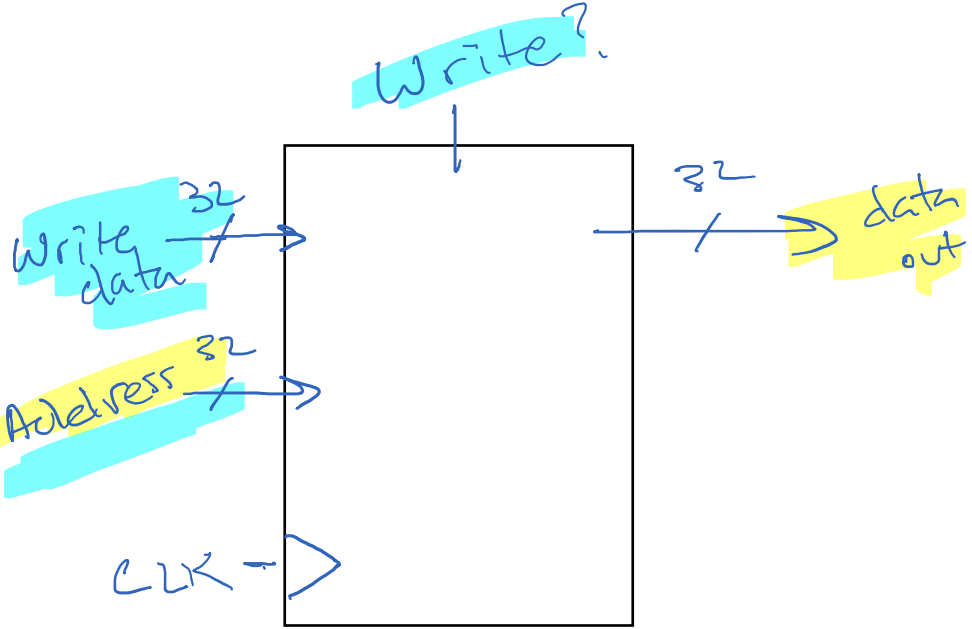


Which of these describes our memory interface (for now)?

A	One 32-bit output	One 5-bit input	One 32-bit input	Clk input	Two 1-bit control inputs
B	One 32-bit output	Two 5-bit inputs		Clk input	Two 1-bit control inputs
C	One 32-bit output		Two 32-bit inputs	Clk input	Two 1-bit control inputs
D	One 32-bit output		One 32-bit input	Clk input	Two 1-bit control inputs
E	*None of these are correct*				

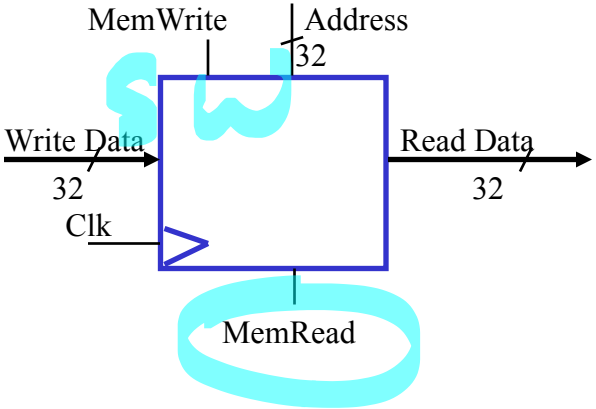
Let's describe the signals to interface to *Memory*

Simple



lw

sw

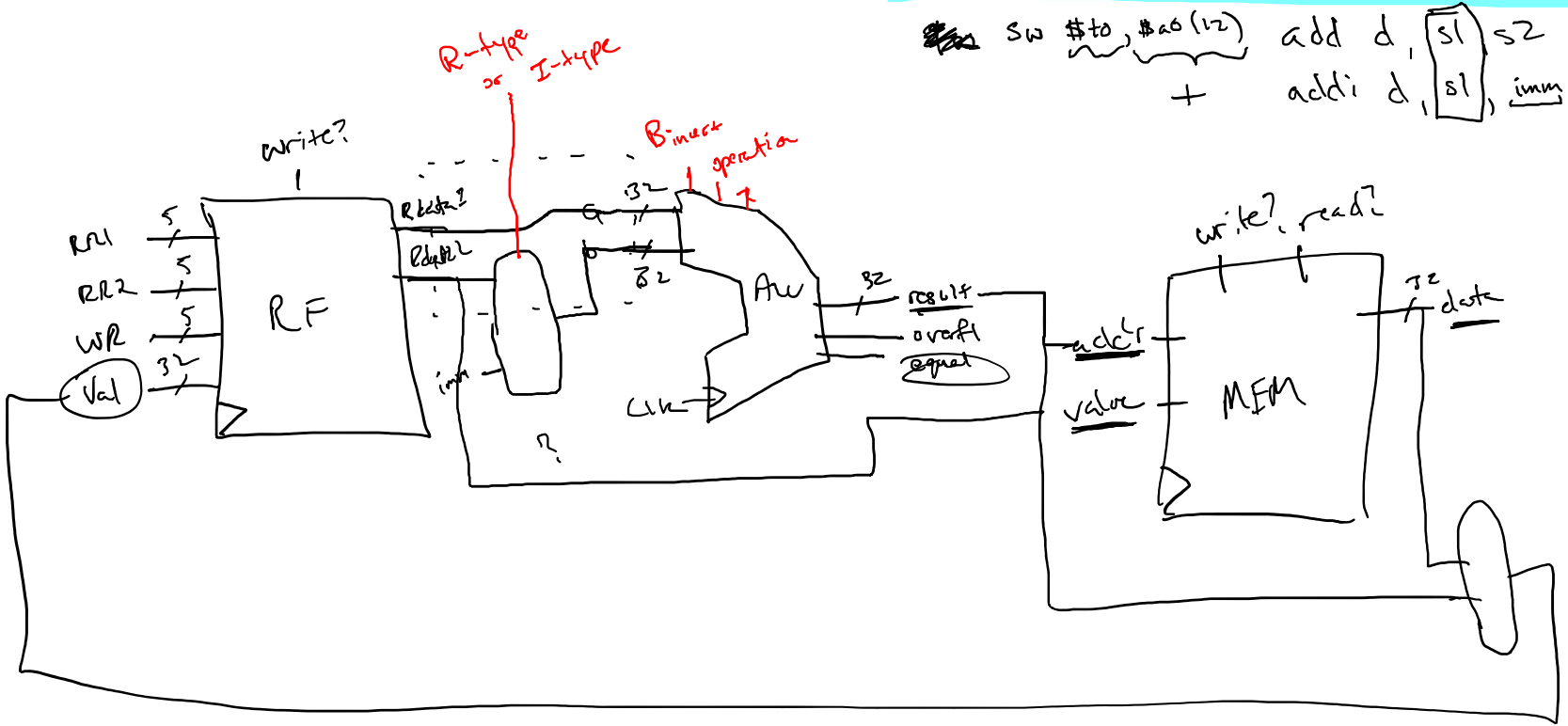


~~eventually~~ eventually: will want EN (Read?)

Can we layout a high-level design to do everything?

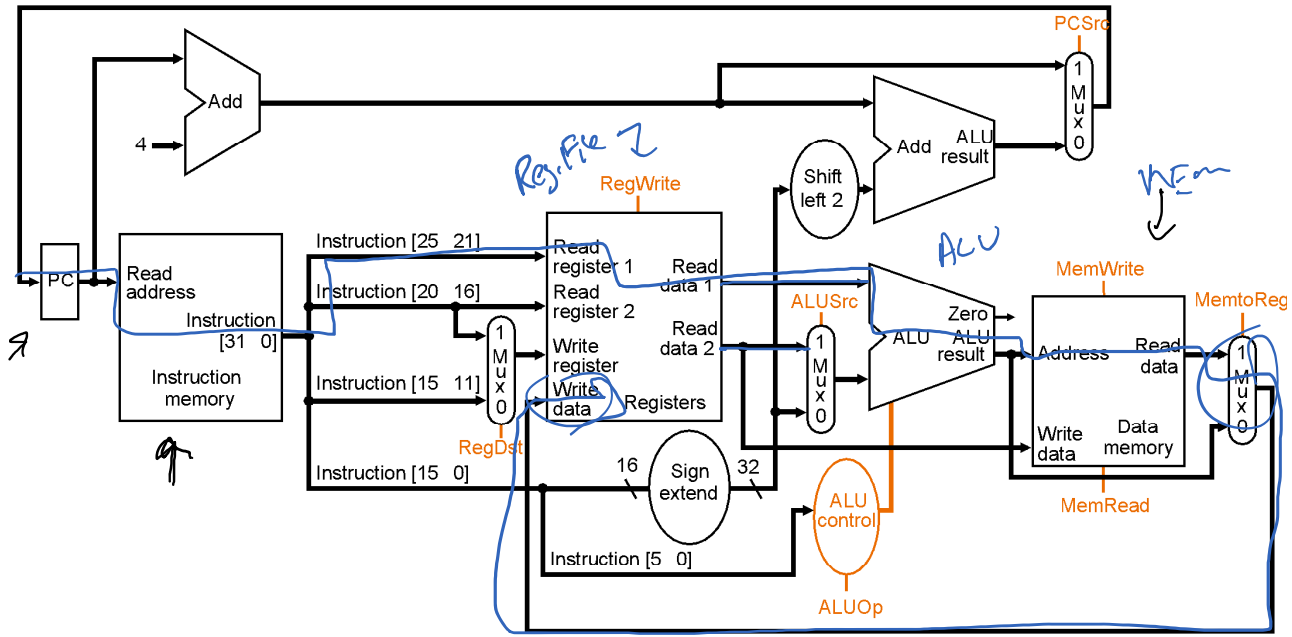
We're ready to look at a simplified MIPS with only:

- memory-reference instructions: `lw, sw`
- arithmetic-logical instructions: `add, sub, and, or, slt`
- control flow instructions: `beq`



Putting it All Together: A Single Cycle Datapath

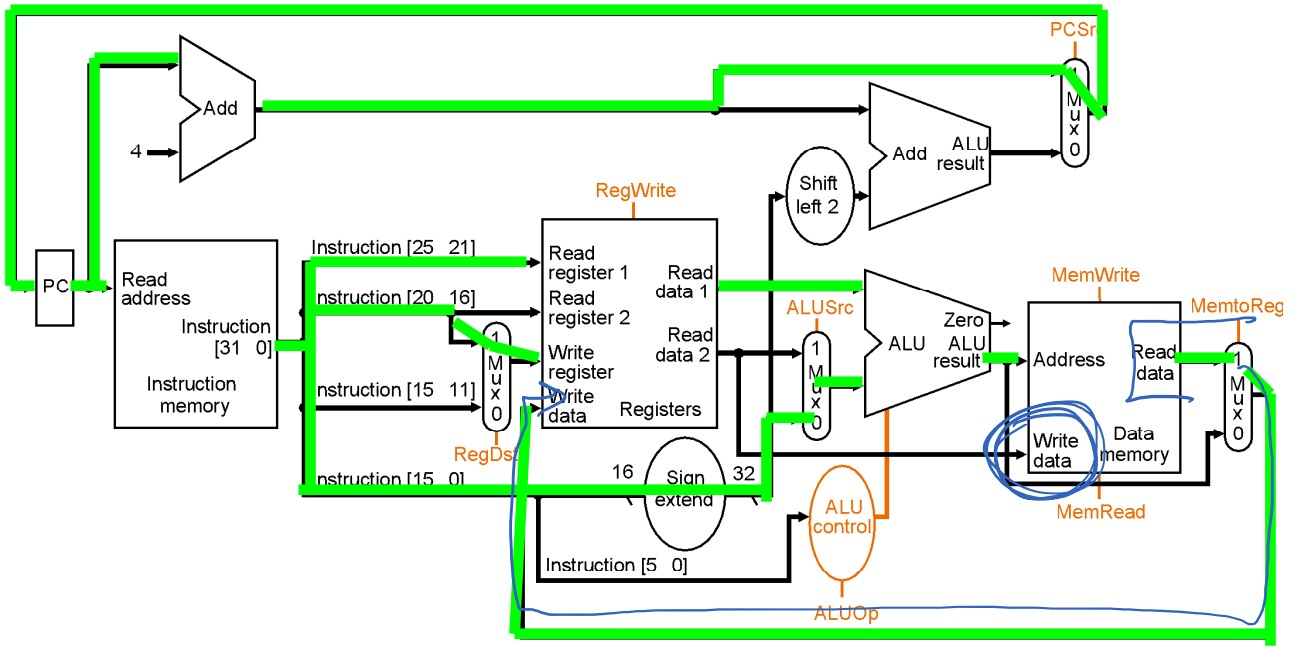
- We have everything except control signals (later)



Active Single-Cycle Datapath

Ignoring control – which instruction does this active datapath represent

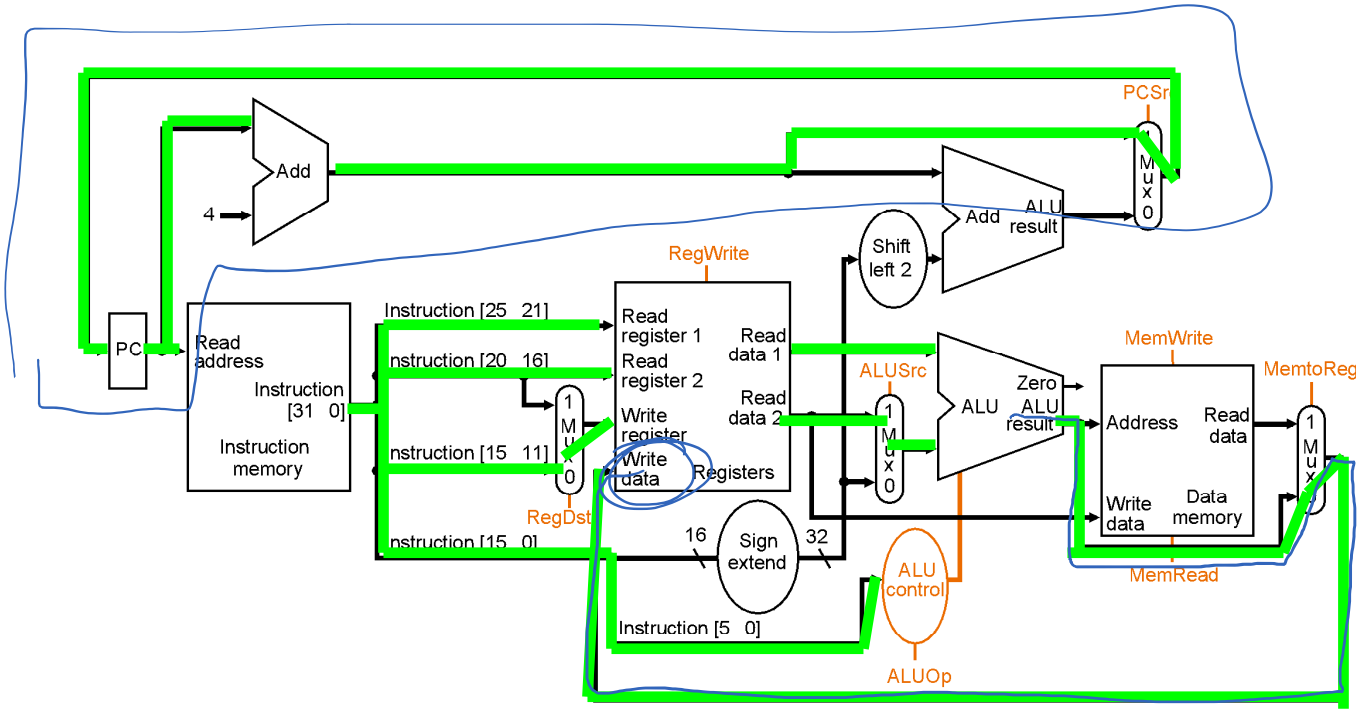
A. R-type
B. lw
C. sw
D. Beq
E. None of the above



Active Single-Cycle Datapath

Ignoring control – which instruction does this active datapath represent

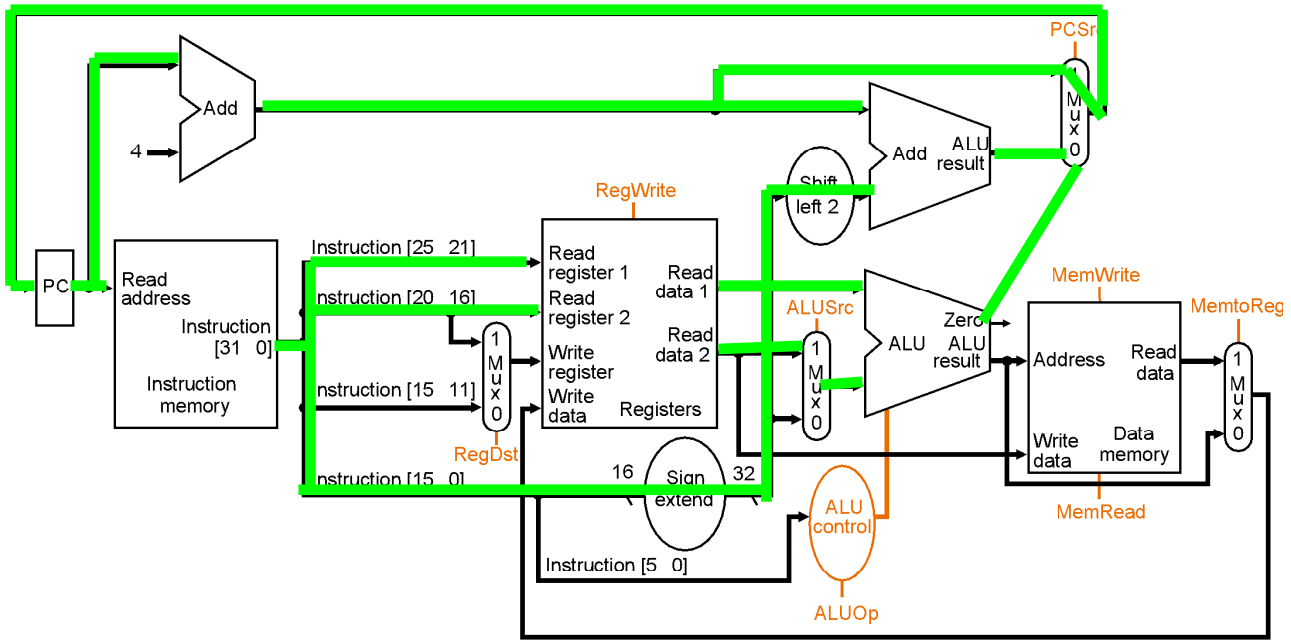
A. R-type
B. lw
C. sw
D. Beq
E. None of the above



Active Single-Cycle Datapath

Ignoring control – which instruction does this active datapath represent

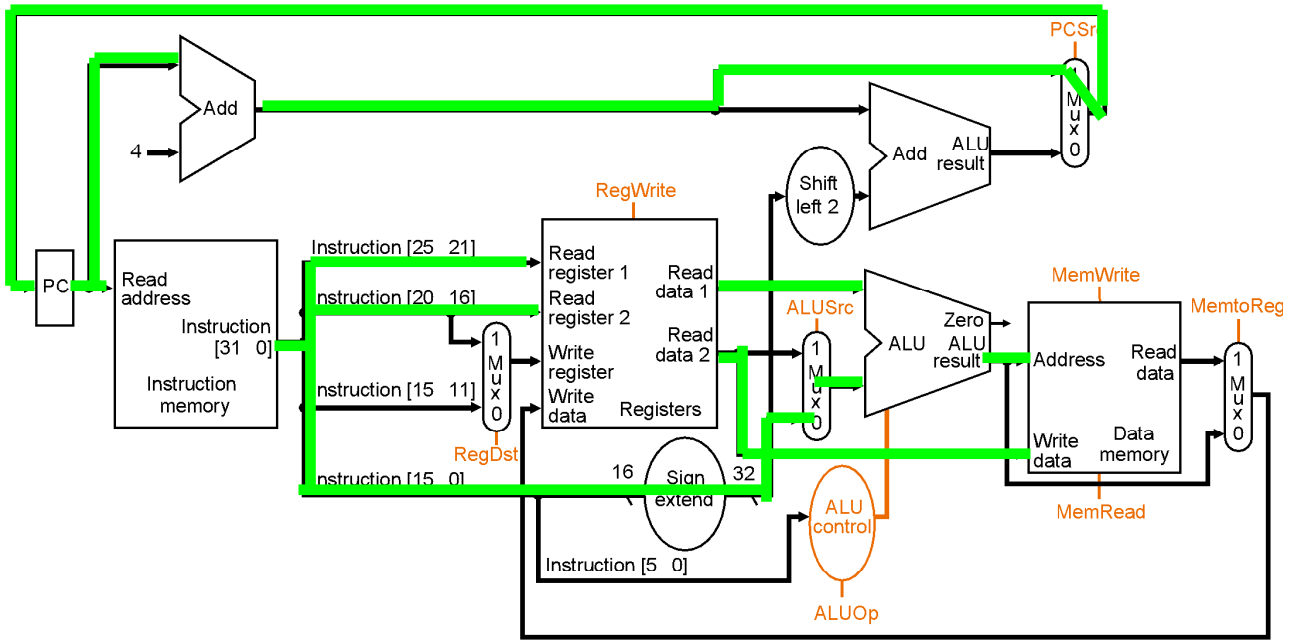
A. R-type
B. lw
C. sw
D. Beq
E. None of the above



Active Single-Cycle Datapath

Ignoring control – which instruction does this active datapath represent

A. R-type
B. lw
C. sw
D. Beq
E. None of the above



Key Points

- CPU is just a collection of state and combinational logic
- We just designed a very rich processor, at least in terms of functionality
- ET = IC * CPI * Cycle Time
  - where does the single-cycle machine fit in?

“The Control Path”

aka, what controls which wires are green?