Building a Processor: Putting the Components Together

Check In 2 on Friday!

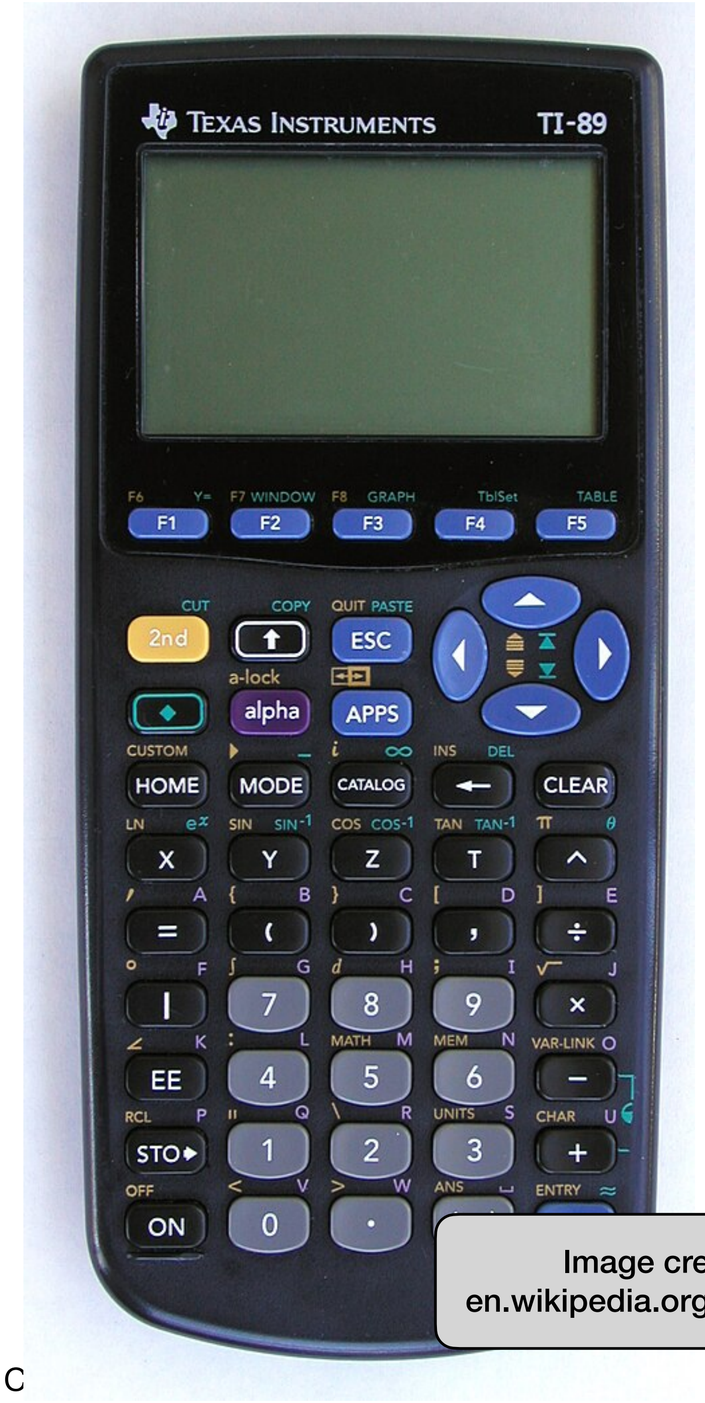
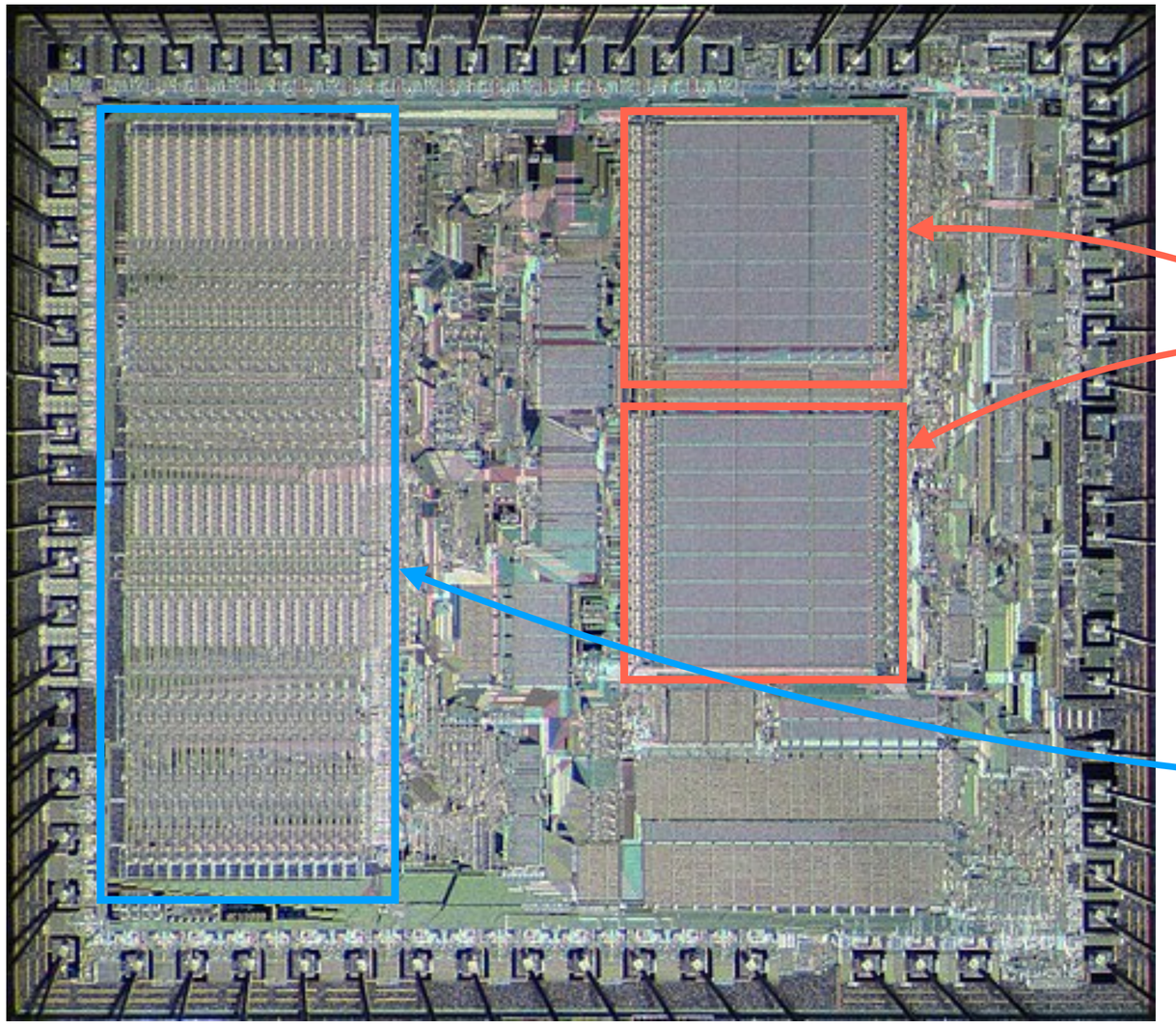


Image credit: https://en.wikipedia.org/wiki/TI-89_series



Register Files

Combinational Arithmetic Logic

Image credit: https://en.wikipedia.org/wiki/Motorola_68000

A Brief Clarification…

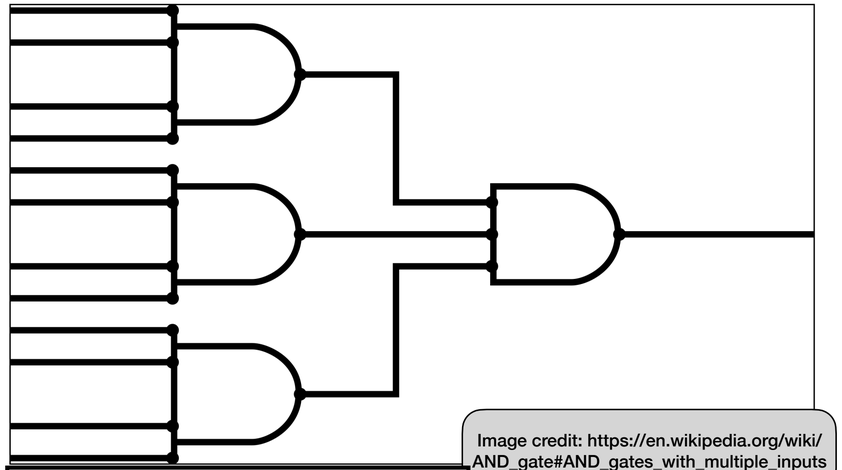
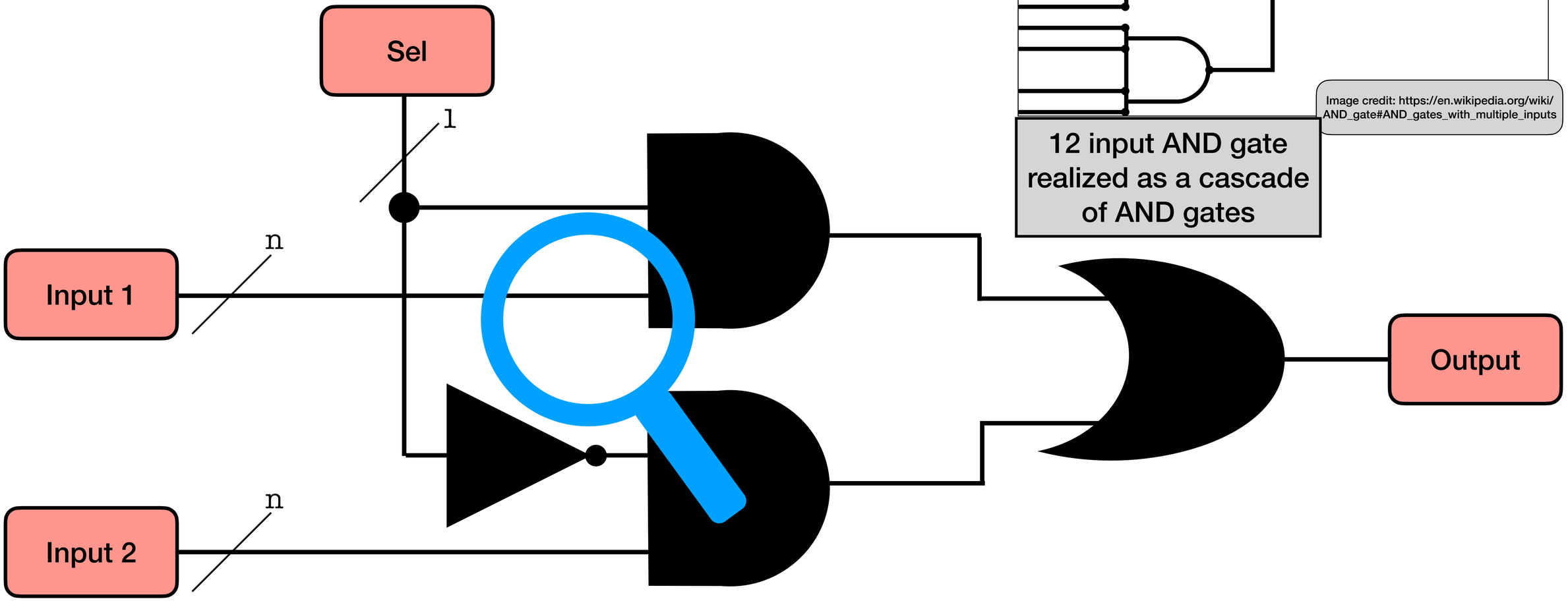


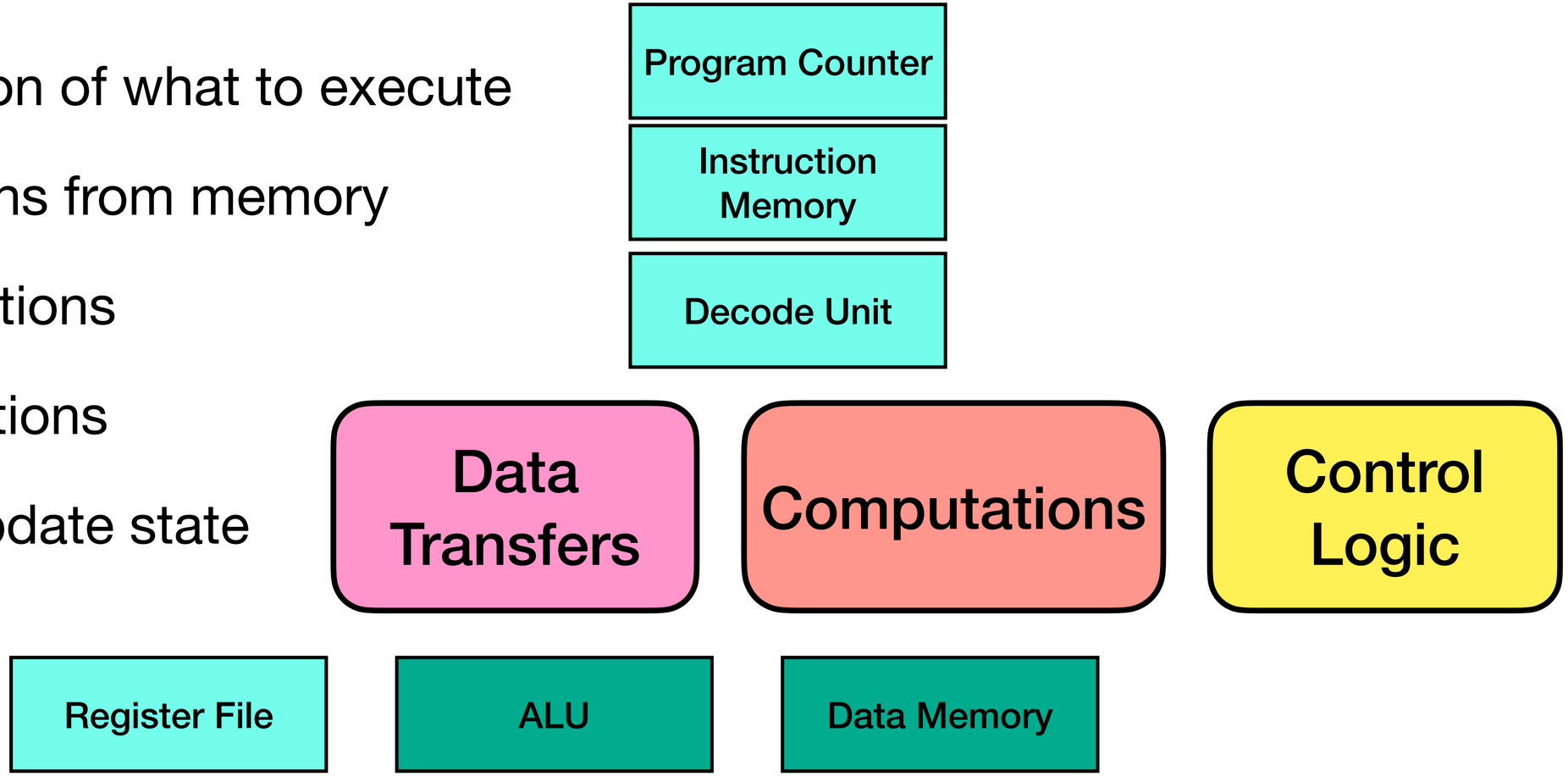
Image credit: https://en.wikipedia.org/wiki/AND_gate#AND_gates_with_multiple_inputs

12 input AND gate realized as a cascade of AND gates



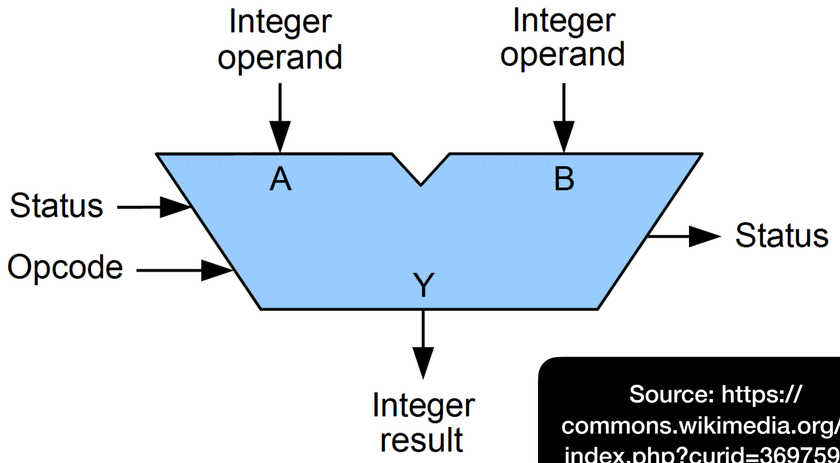
From Friday

- Track the location of what to execute
- Fetch instructions from memory
- Interpret instructions
- Execute instructions
- Maintain and update state



Computational Units

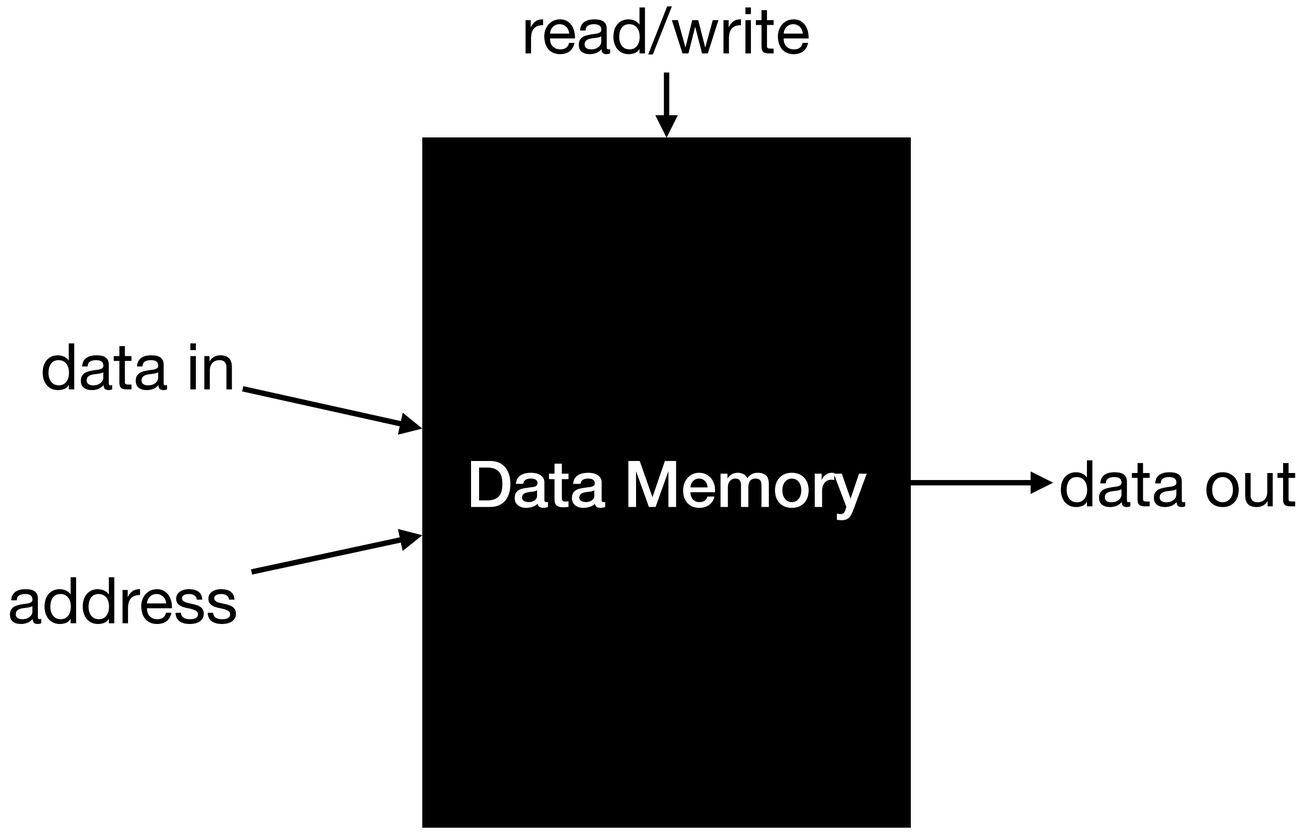
- Arithmetic Logic Unit (ALU): a large combinational logic unit for arithmetic operations
- Integer operands (inputs) passed to the ALU with values to operate on
- Takes “opcode” as input to determine which functionality to use where functionality is implemented as a “pure function” from combinational logic
- Produces integer result; status output used to detect overflows, exceptions, etc.



Source: https://commons.wikimedia.org/w/index.php?curid=36975996

Data Memory

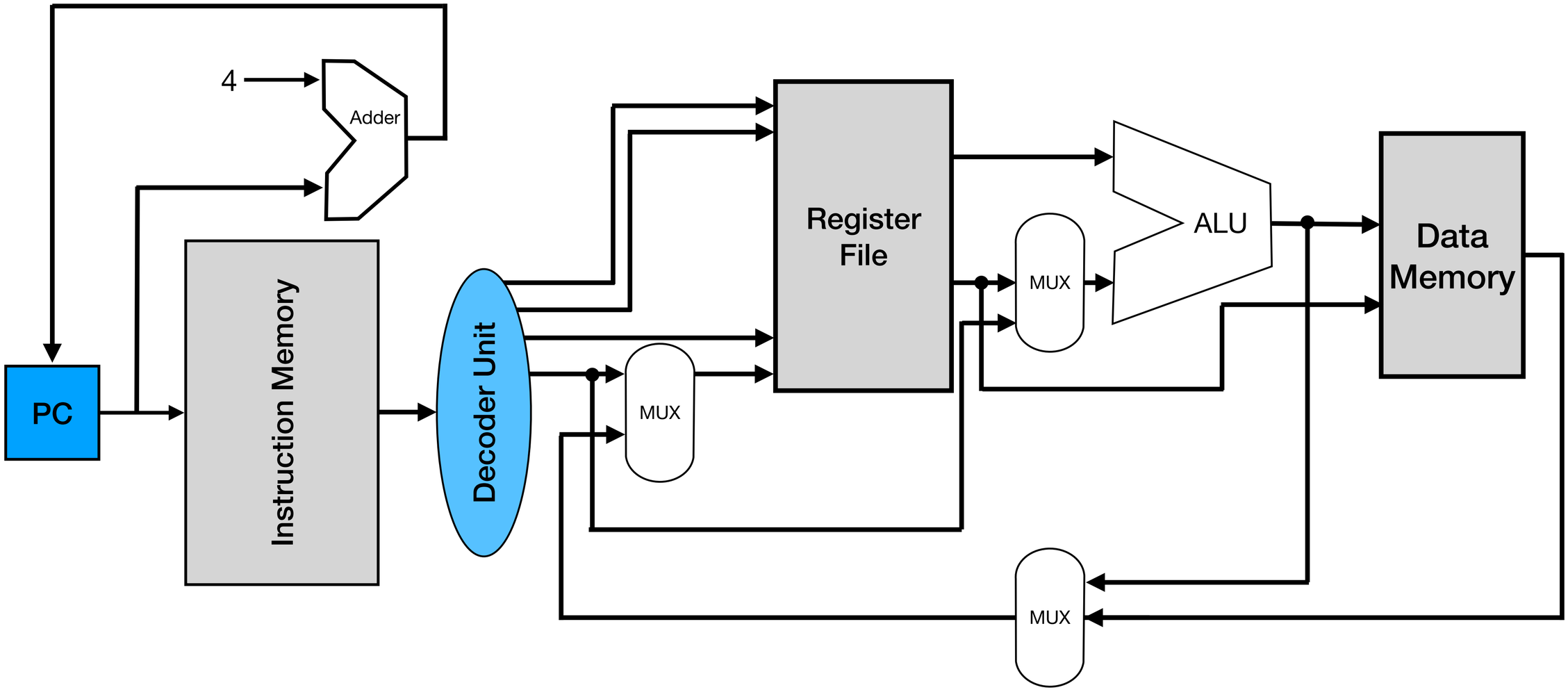
- Much like instruction memory, data memory is composed of transistors/ capacitors
- Data may be read or written, so there is an additional control signal that this component uses as input
- Data out only set if input signal is set to read
- Data in only used if input signal is set to write



Putting it all Together…

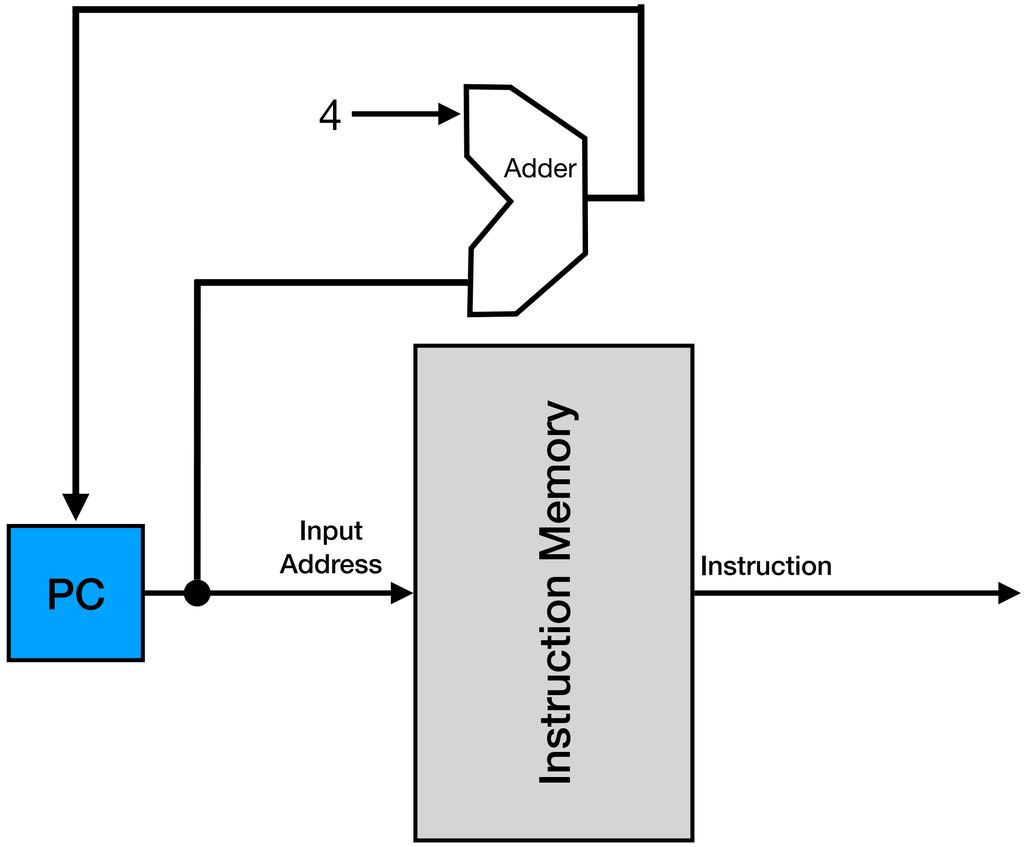
- Now that we have all of the functional units to operate on and hold data in the processor, we need to connect them
- To do so, we will first think about creating pathways for all data to flow through the appropriate execution unit for each possible instruction that we implement
- These connections are referred to as the *data path*
- After our data path is constructed, we will think about how to control the flow of information through the processor relative to the instruction to execute

Where we're going…



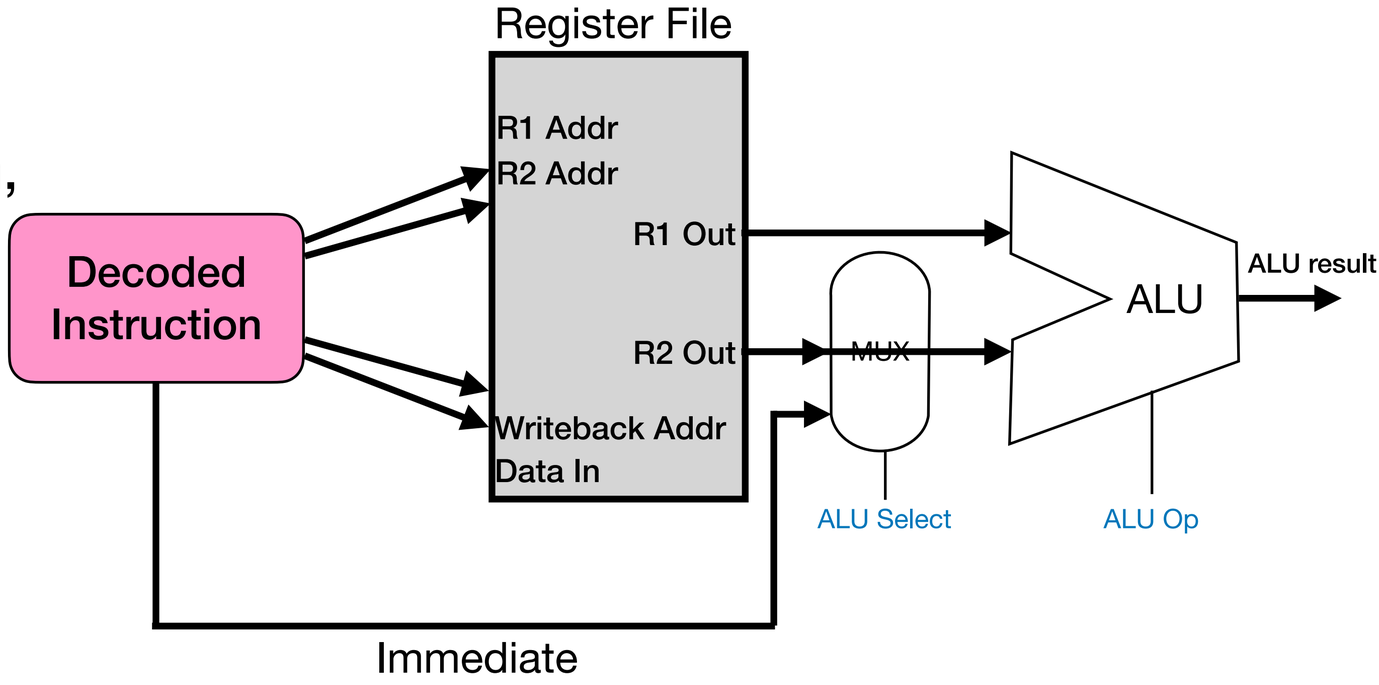
Building an Instruction Fetch Data Path

- We can build an “adder” as a simpler form of the ALU with only one arithmetic functionality
- The PC acts as input to instruction memory and input to the “PC updating logic”
- For now, let’s assume all instructions are four-bytes long and that there are no control instructions… we will come back to this later in the course!



Building an Execution Data Path

- Our decoded instruction will have all sorts of inputs depending on the operation
- Almost all data transfer, computation, and control logic instructions will refer to data in the register state
- Use register outputs as inputs to execution
- But wait… how to handle immediates?

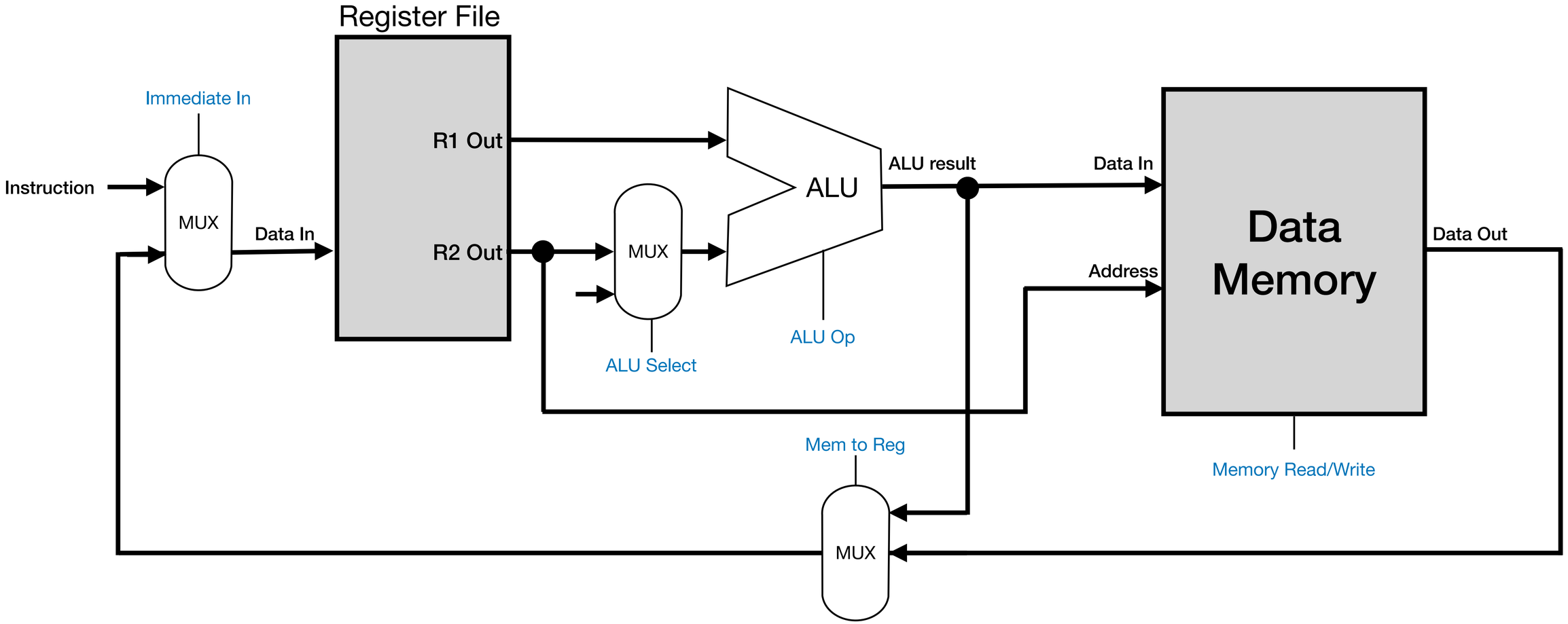


Building a Data Path to Update and Maintain State

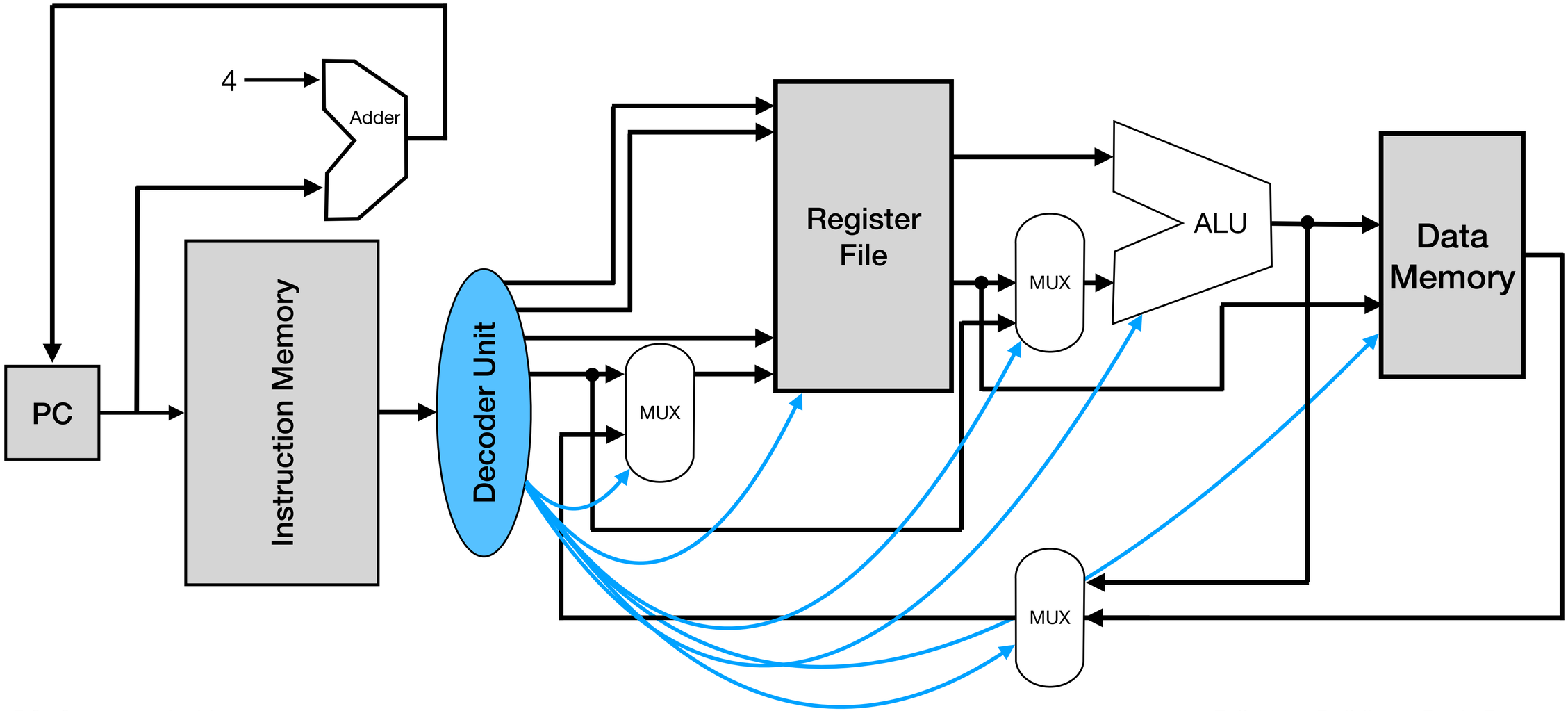
Q0: Where does the input for data memory come from?

Q1: What should our data path do if updated value is ALU result?

Q2: How should our data path handle “load immediate” at data in?



Putting it all Together!



Control Signals

- ALU Select: determines whether the second ALU data input is from the instruction (i.e., an immediate) or from a register source
- ALU Op: determines which operation the ALU will perform
- Memory Read/Write: determines whether the address should be read from or written to
- Memory to Register: determines whether the data to write to the register file should come from the execution units or from memory

All of this information can be inferred from the instruction opcode! The decode unit sends signals to the appropriate fields

What Assumptions Have We Made?

- Instructions always incremented by four… Fixed size and no control instructions!
- Decoder as a black box that handles all control logic…
- Instruction memory and data memory as a lightweight black box…

Takeaways

- Components can be constructed into a processor's *data path*
- The data path must handle the flow of all instructions — different instruction sets will require a different data path!