Computer System Architecture - CSEN 601

Module 4: *Pipelining and Execution Techniques*

Lecture 09: *MIPS Single-Cycle Machines Datapath Implementation Scheme*

Dr. Eng. Catherine M. Elias

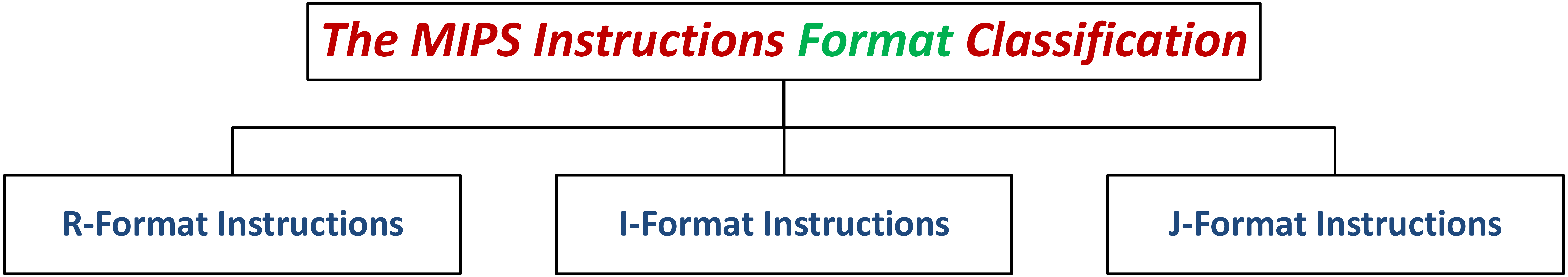
catherine.elias@guc.edu.eg

*Lecturer, Computer Science and Engineering,
Faculty of Media Engineering and Technology, German University in Cairo*

Outline

- The Datapath
- Building a MIPS Datapath

Remember MIPS!



	31 26	25 21	20 16	15 11	10 6	5 0
R-Type	opcode	rs	rt	rd	sa	function

	31 26	25 21	20 16	15 0
I-Type	opcode	rs	rt	immediate

	31 26	25 0
J-Type	opcode	Instr_index

It can be noticed from the illustrated formats that:

- all instruction types have an opcode field, and
- they differ in terms of the other fields that include the needed registers to perform the required operation, immediate, or/and index.

Remember MIPS!

Some Observations!

- Much of what needs to be done to implement these instructions is the same, independent of the exact class of instruction. For every instruction, the first two steps are identical:
  1. Send the program counter (PC) to the memory that contains the code and fetch the instruction from that memory.
  2. Read one or two registers, using fields of the instruction to select the registers to read.
     For the load word instruction, we need to read only one register, but most other instructions require reading two registers.

Remember MIPS!

Some Observations!

- After these two steps, the actions required to complete the instruction depend on the instruction class.
- Fortunately, for each of the three instruction classes (memory-reference, arithmetic-logical, and branches), the actions are largely the same, independent of the exact instruction.
- The simplicity and regularity of the MIPS instruction set simplifies the implementation by making the execution of many of the instruction classes similar.
- For example, all instruction classes, *except jump*, use the arithmetic-logical unit (ALU) after reading the registers.
    - ➢The memory-reference instructions use the ALU for an address calculation.
    - ➢The arithmetic-logical instructions for the operation execution, and branches for comparison.

Remember MIPS!

Some Observations!

- After using the ALU, the actions required to complete various instruction classes differ.
  - ➢A memory-reference instruction will need to access the memory either to read data for a load or write data for a store.
  - ➢An arithmetic-logical or load instruction must write the data from the ALU or memory back into a register.
  - ➢A branch instruction may need to change the next instruction address based on the comparison; otherwise, the PC should be incremented by 4 to get the address of the next instruction.

Very Important Note:

When we design our computer architecture, think generically to fit all the instructions.

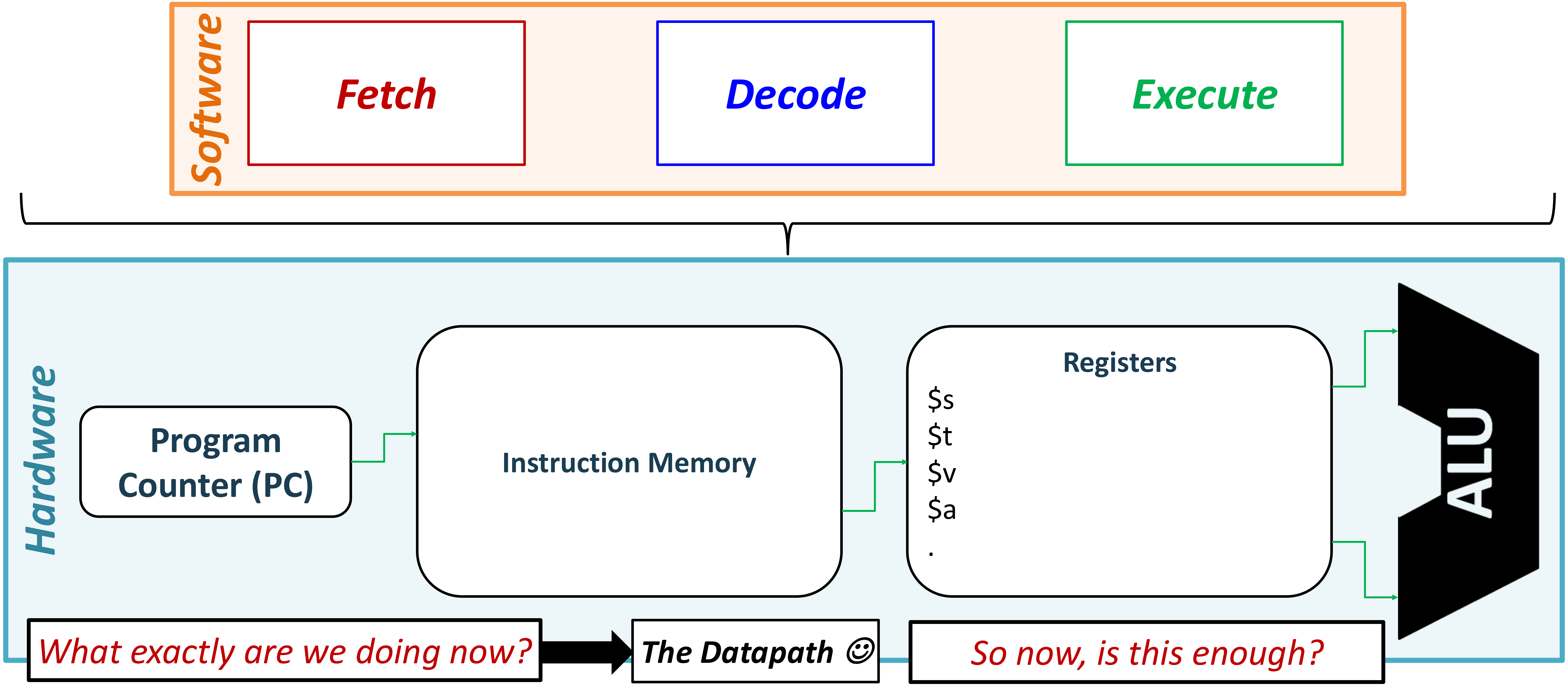
Basic Instruction Execution Stages

What are the steps when executing an instruction?

- Based on the previous observations, we can conclude the three basic execution stages:
  1. Use program counter (PC) to **Fetch** an instruction to execute it.
  2. **Decode** the needed registers to be selected and read using the proper fields
  3. **Execute** the action intended by the instruction

Basic Instruction Execution Stages

What are the steps when executing an instruction?



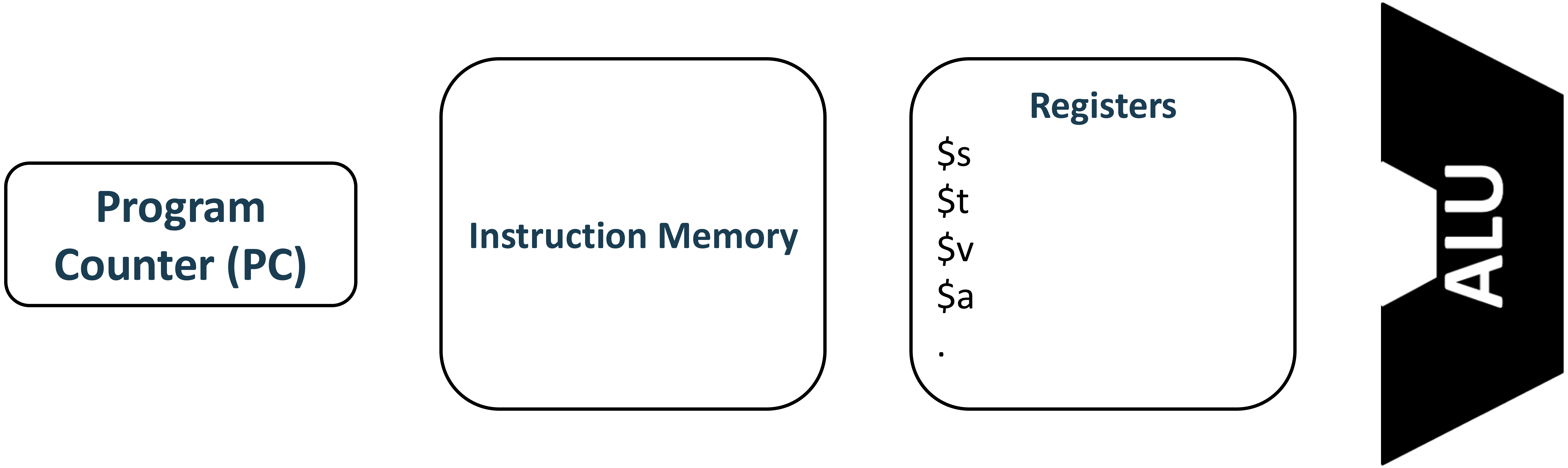
What exactly are we doing now? → ***The Datapath ☺***

So now, is this enough?

The Datapath

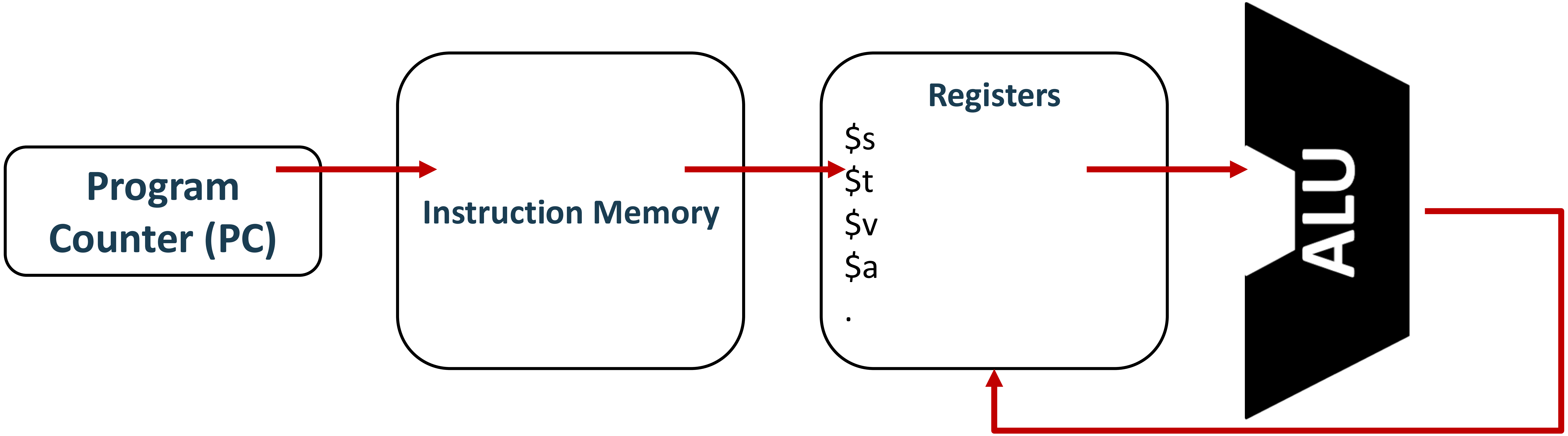
How to build the datapath?

- Imagine the datapath as some pipes where the data flows.
- Therefore, you should consider where and in which direction the data will flow from the fetching till the end of the execution.
- Again, think generically when you design the datapath to fit all the instructions classes.
- In order to do that, we can start by drawing separate datapathes for each instruction class.



The Arithmetic and Logical Instructions

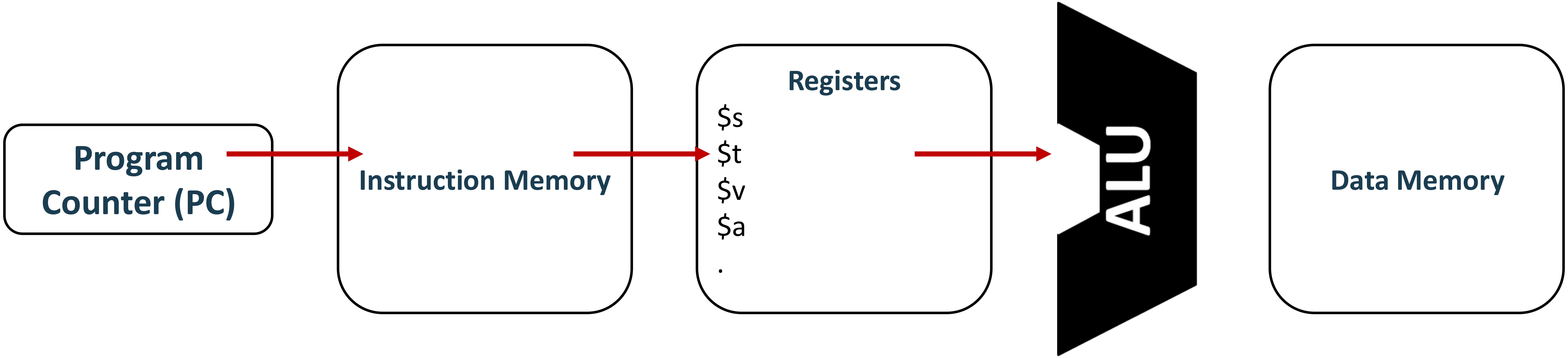
- PC determines the instruction address to be fetched
- Decoded instruction determine the 2 source registers
- These registers are passed to the ALU where the function is executed.
- After finishing the execution, the results should be stored again in the registers (*remember the Reg-Reg machine model*)



The Datapath

Memory-Reference Instructions

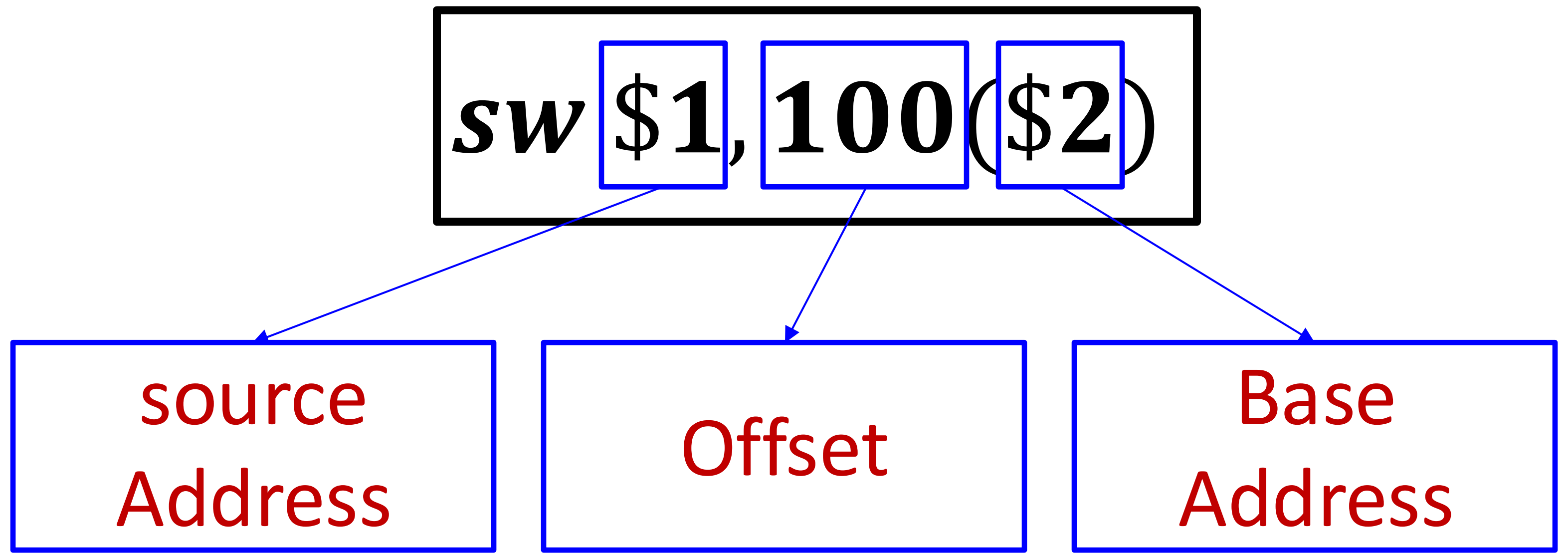
- Here these components are not enough anymore, as we have to load/store our data in the data memory

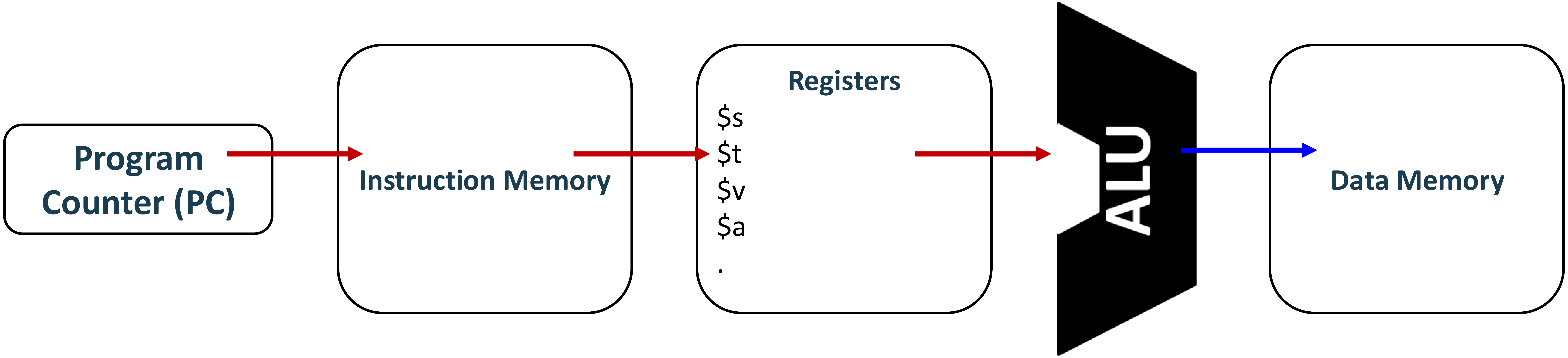


The Datapath

Memory-Reference Instructions: *Store*

- PC determines the instruction address to be fetched
- Decoded instruction determine the 1 source address
- The ALU performs addition to get the new storage location *(base registers + the offset)*
- The results is stored in the data memory

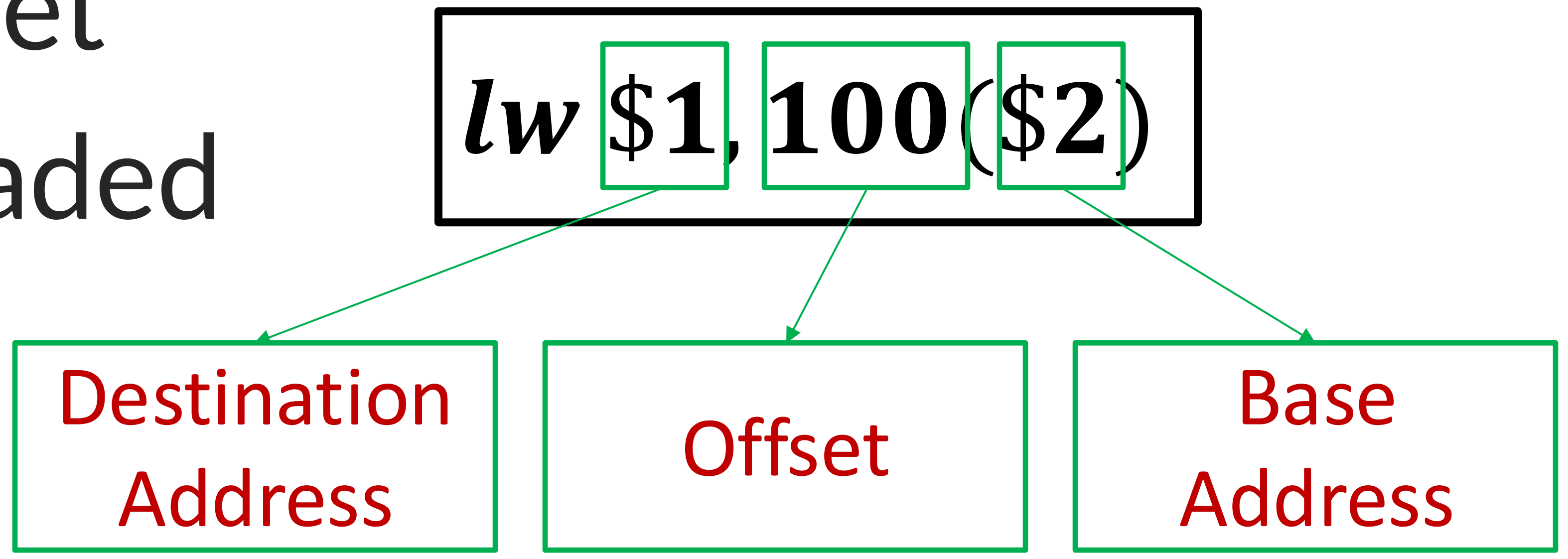


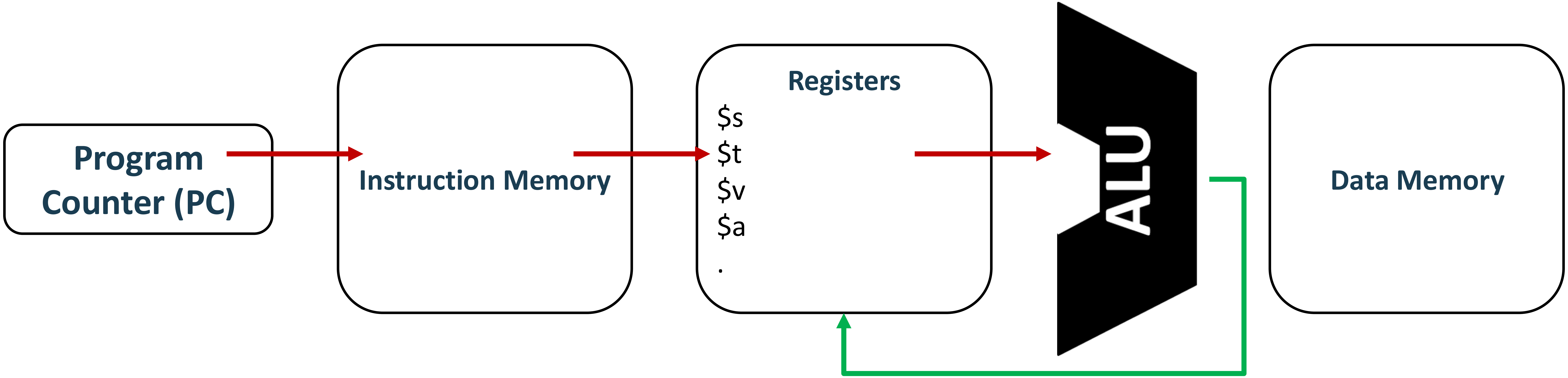


The Datapath

Memory-Reference Instructions: *Load*

- PC determines the instruction address to be fetched
- Decoded instruction determine the 1 base registers and the offset
- The ALU performs addition to the address where data will be loaded
- The results is loaded in the destination register

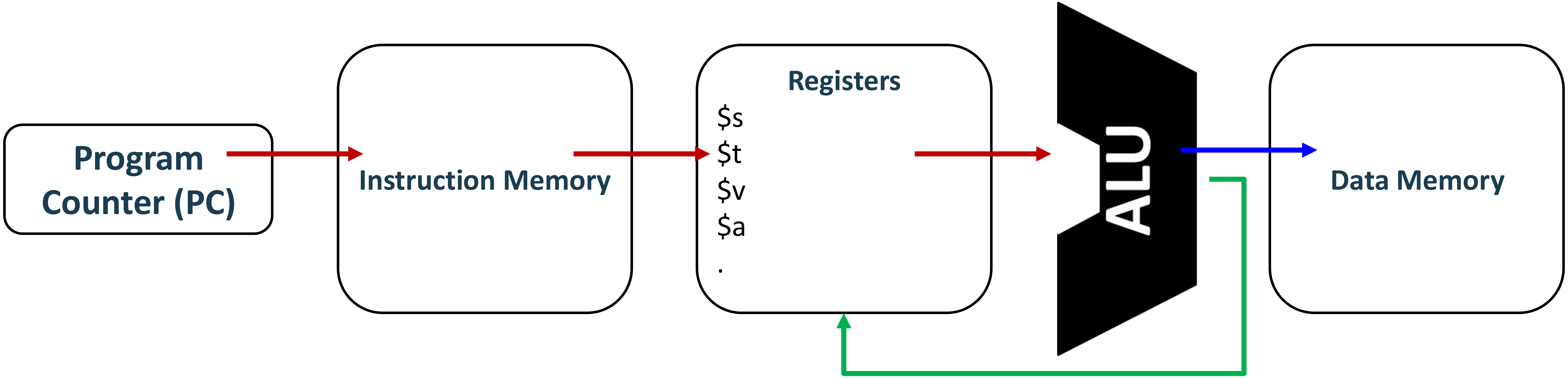




The Datapath

Memory-Reference Instructions

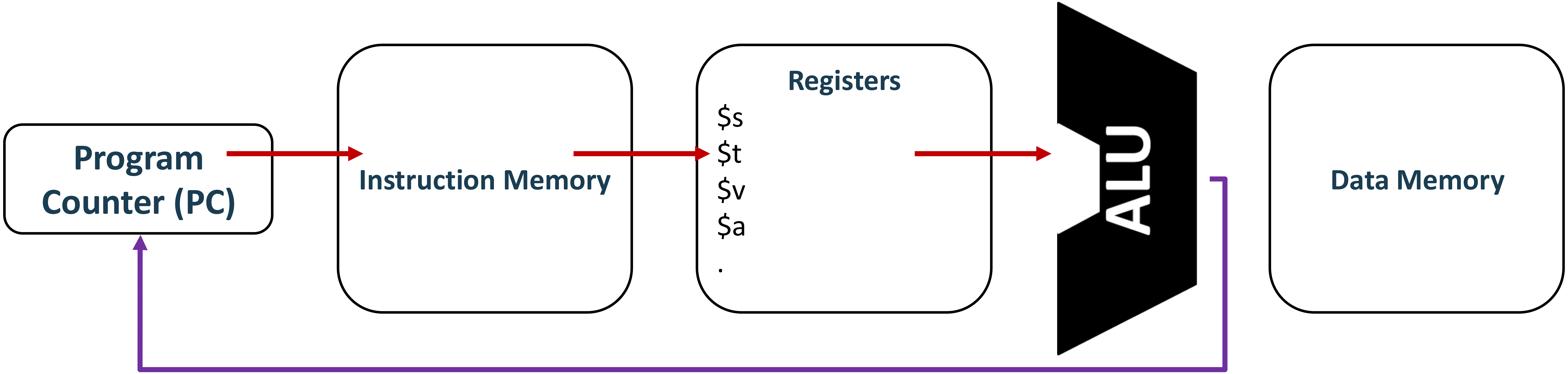
- Combining both the load and store instructions, we will end up with:
  - ➢Common datapath from PC till ALU
  - ➢2 possibilities depends on the operation itseld



The Datapath

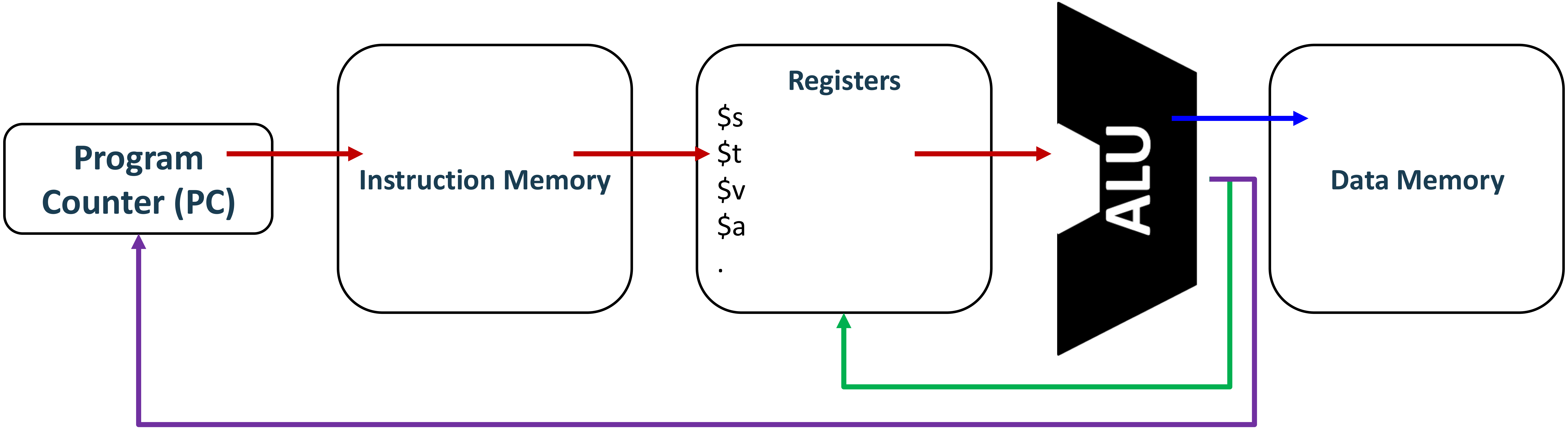
Branches and Jumps Instructions

- PC determines the instruction address to be fetched
- Decoded instruction determine the offset of the jump or the registers to be compared in case of branching
- ALU executes the operation (addition of the offset for $jump$ or subtraction in case of beq)
- Output is used to update the PC value



The Abstracted Datapath

- Here is the big picture combining the 3 instructions classes of MIPS.



Building a Datapath

Datapath Design

- A reasonable way to start a detailed datapath design is to examine **the major components** required to execute each class of MIPS instructions.
- That means we start at the top by looking at which **datapath elements** each instruction needs, and then work our way down through the levels of abstraction.

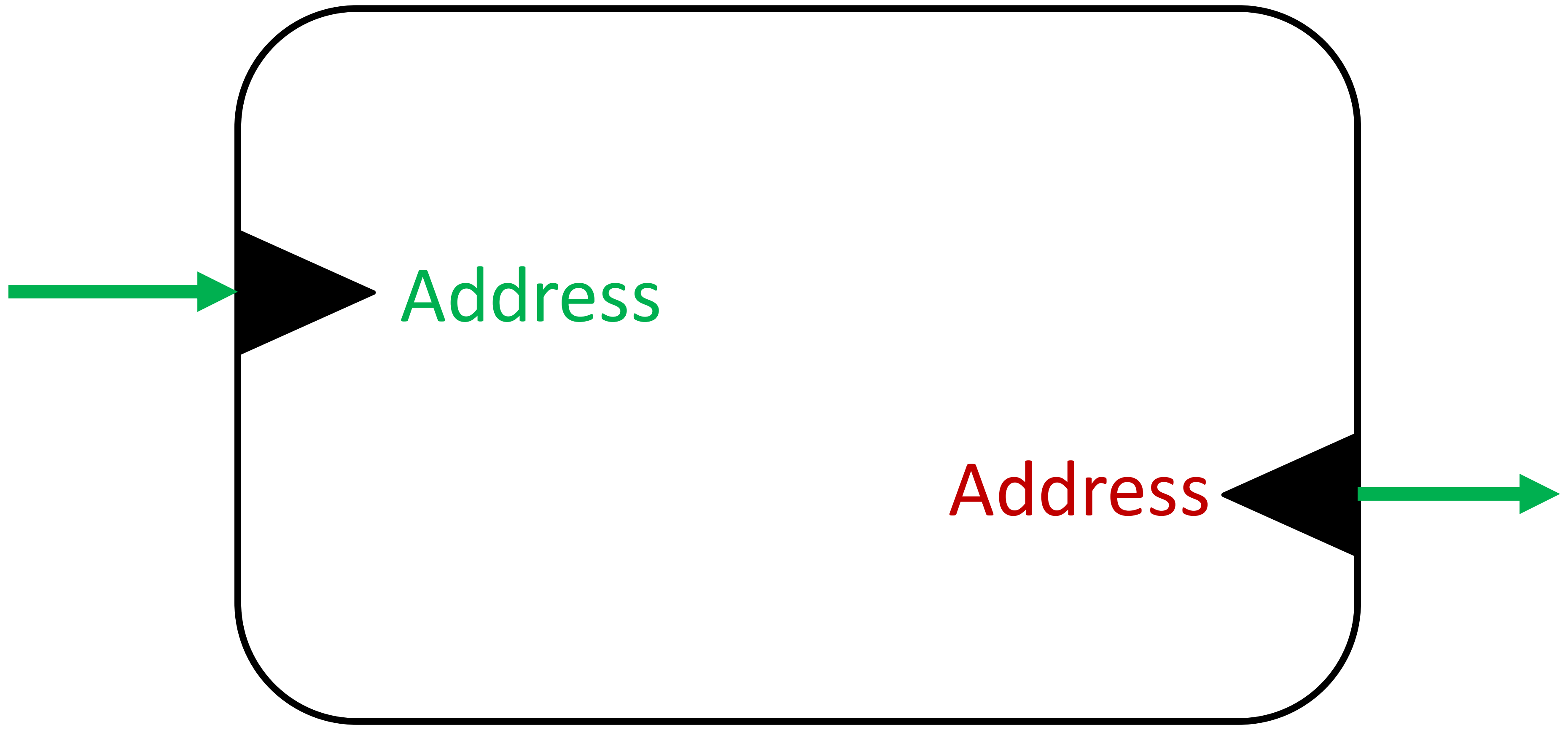
The Datapath Elements

- Datapath element is a unit used to operate on or hold data within a processor.
- In the MIPS implementation, the datapath elements include the instruction and data memories, the register file, the ALU, and adders.

Building a Datapath

The Program Counter

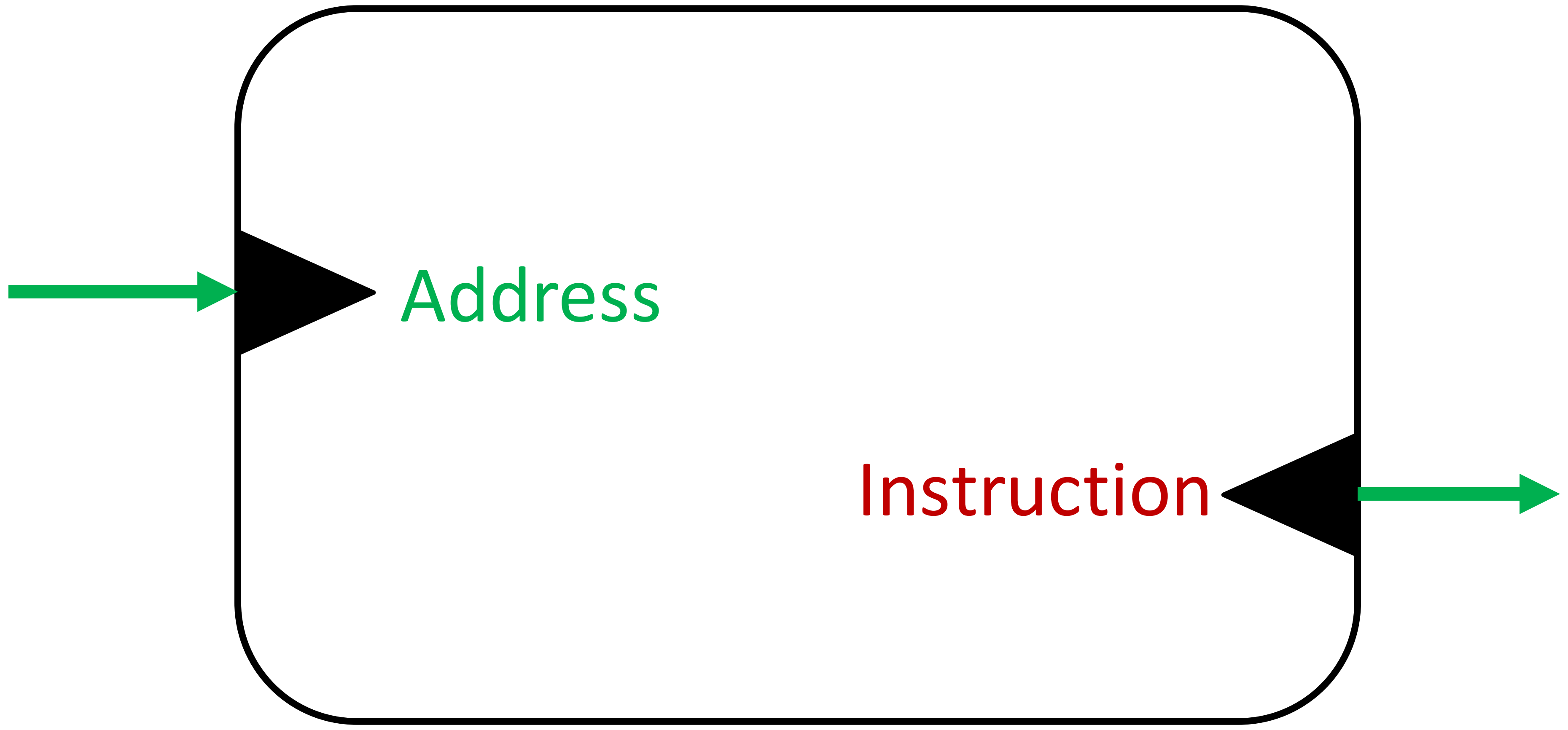
- The program counter is a 32-bit register that is written at the end of every clock cycle and thus does not need a write control signal.



Building a Datapath

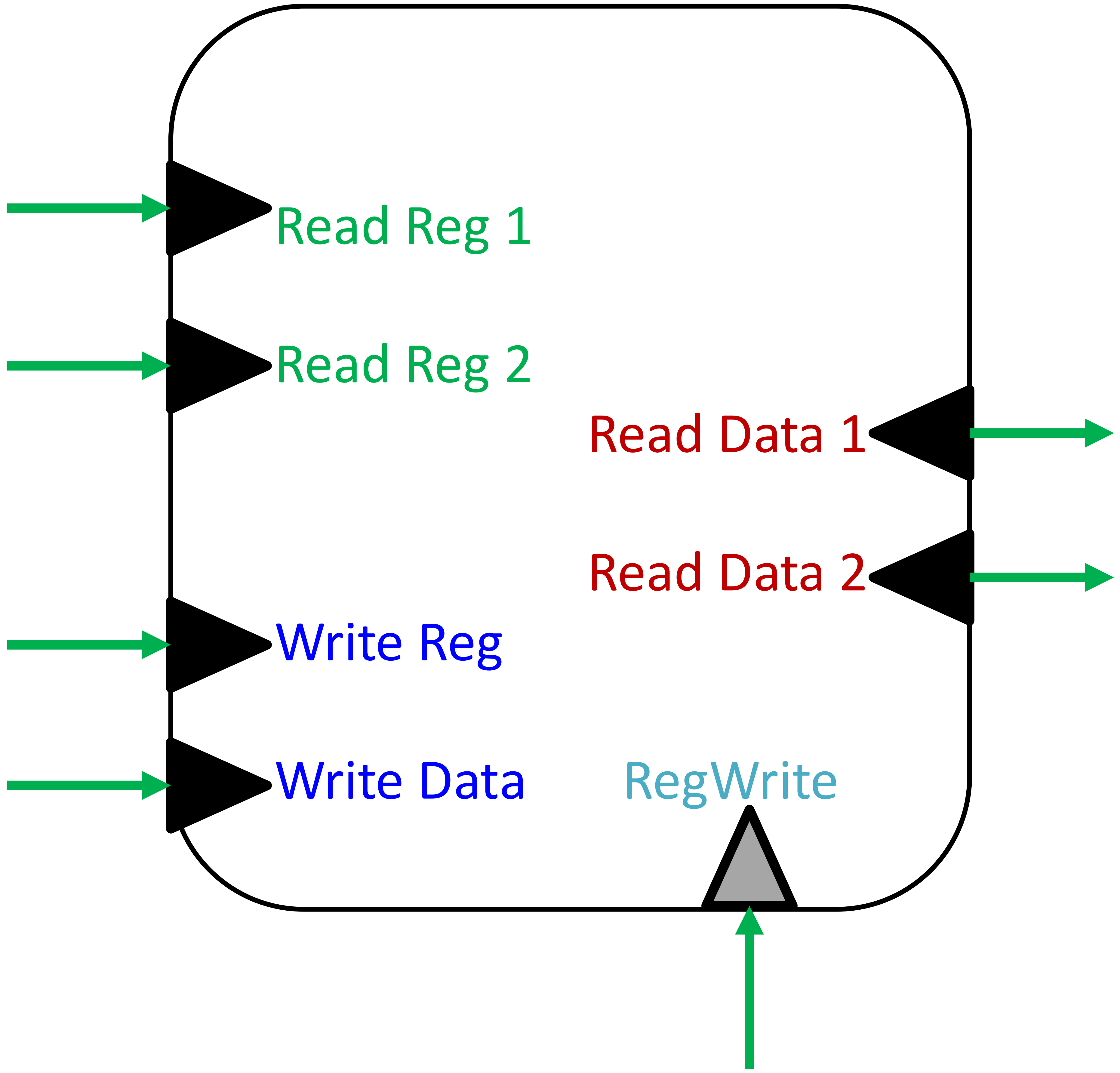
The Instruction Memory

- The instruction memory need only provide read access because the datapath does not write instructions.
- Since the instruction memory only reads, we treat it as combinational logic:
  - ➢the output at any time reflects the contents of the location specified by the address input, and no read control signal is needed.
  - ➢Note that we will need to write the instruction memory when we load the program; this is not hard to add, and we ignore it for simplicity.



The Register File

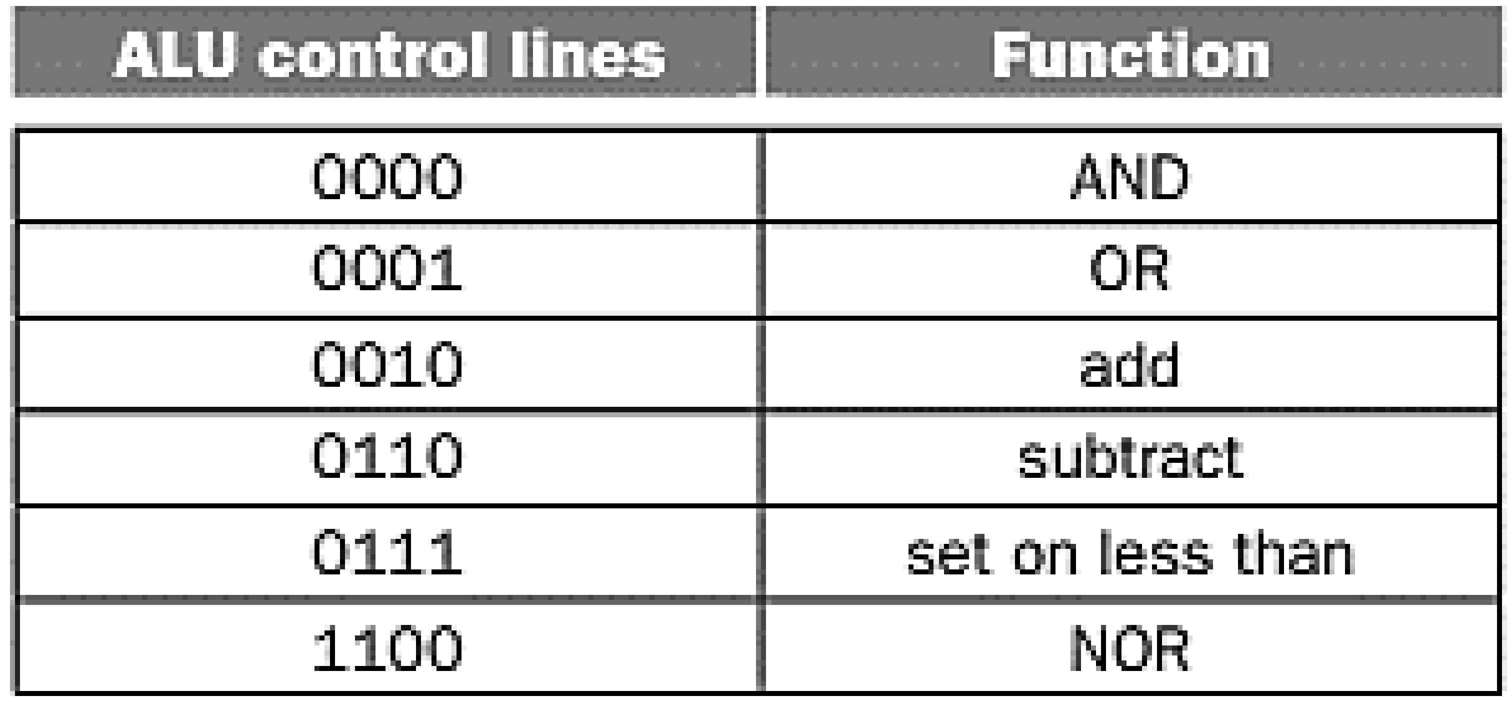
- The register file contains all the registers and has two read ports and one write port.
- The register file always outputs the contents of the registers corresponding to the Read register inputs on the outputs; no other control inputs are needed.
- In contrast, a register write must be explicitly indicated by asserting the write control signal.
- The inputs carrying the register number to the **register file are all 5 bits wide**, whereas the lines carrying **data values are 32 bits wide**. →*Why?*



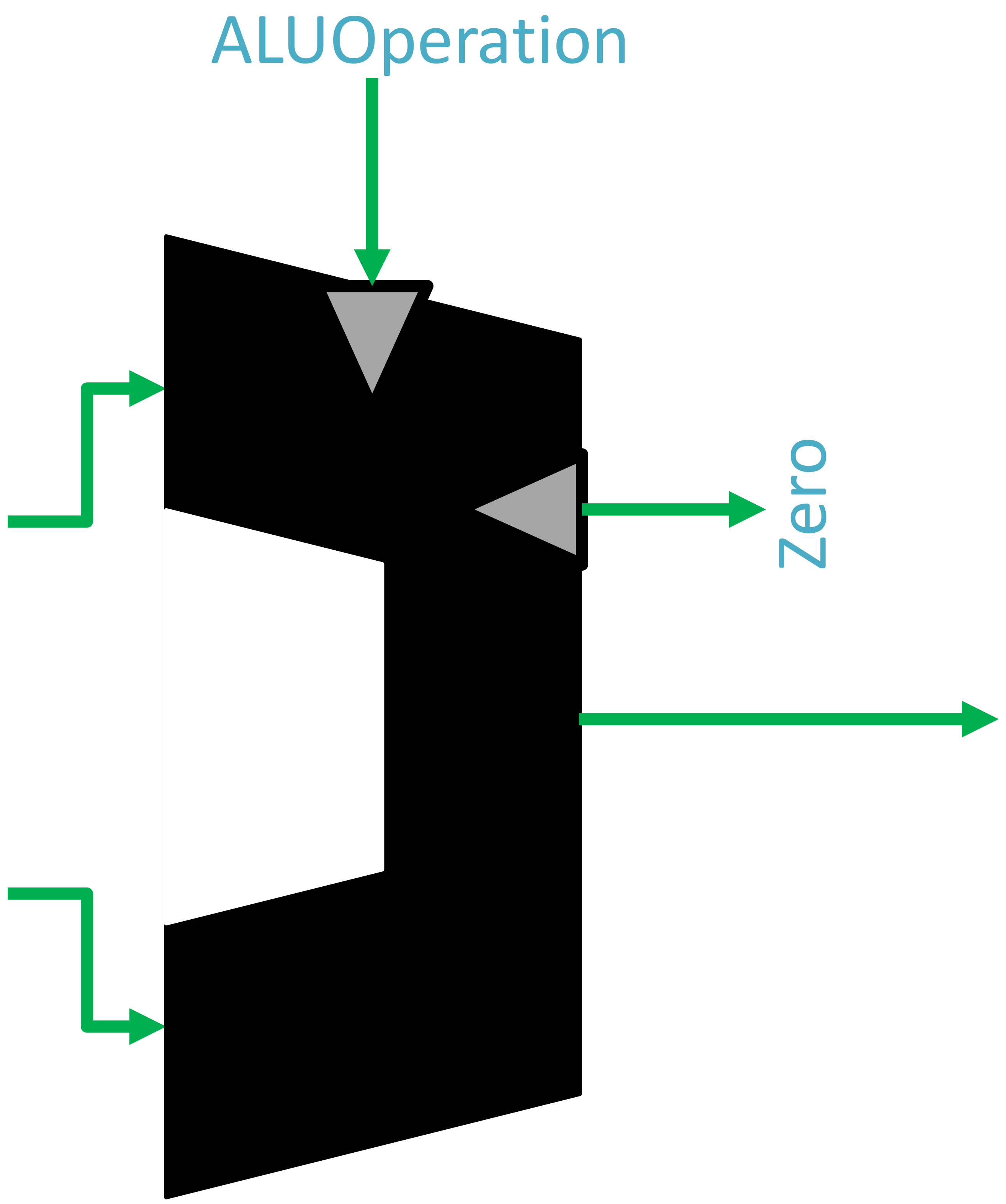
Building a Datapath

The ALU

- The operation to be performed by the ALU is controlled with the ALU operation signal, which will be **4 bits wide**. *→What is that?*
- We will use the Zero detection output of the ALU shortly to implement branches.

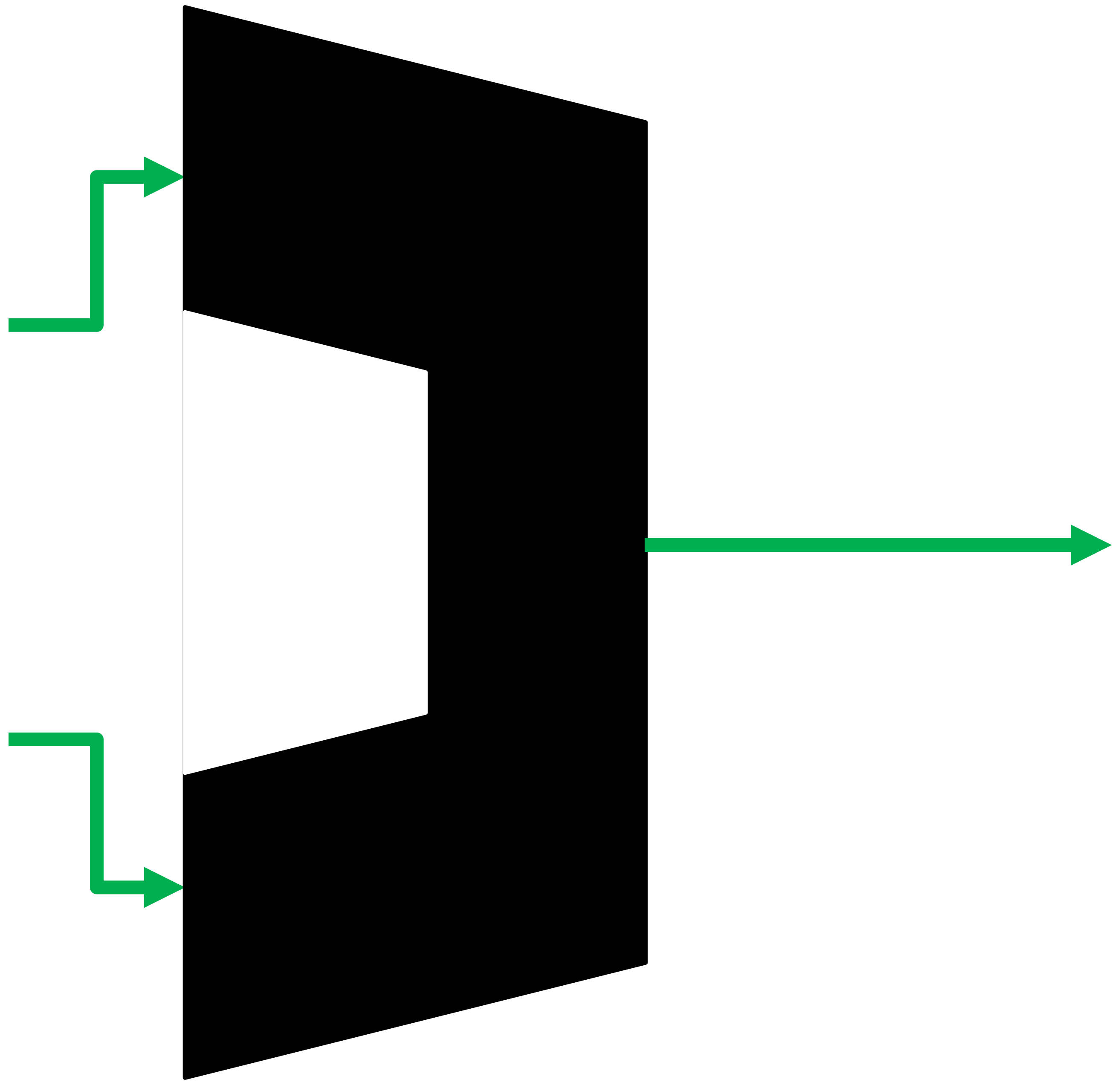


ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR



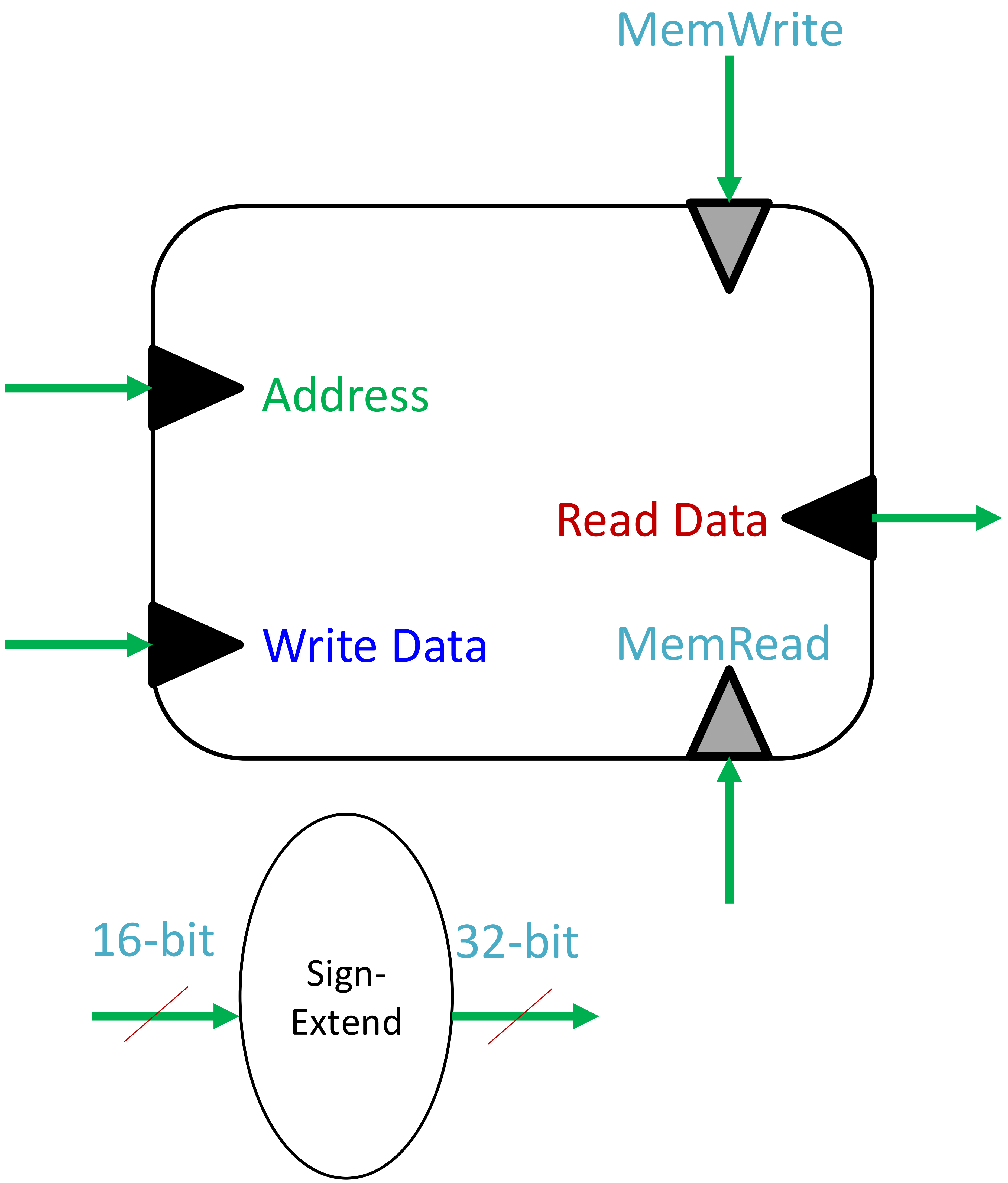
The Adder

- The adder is an ALU wired to always add its two **32-bit inputs** and place the sum on its output. →*the ALU Operation is always set to add 0010*



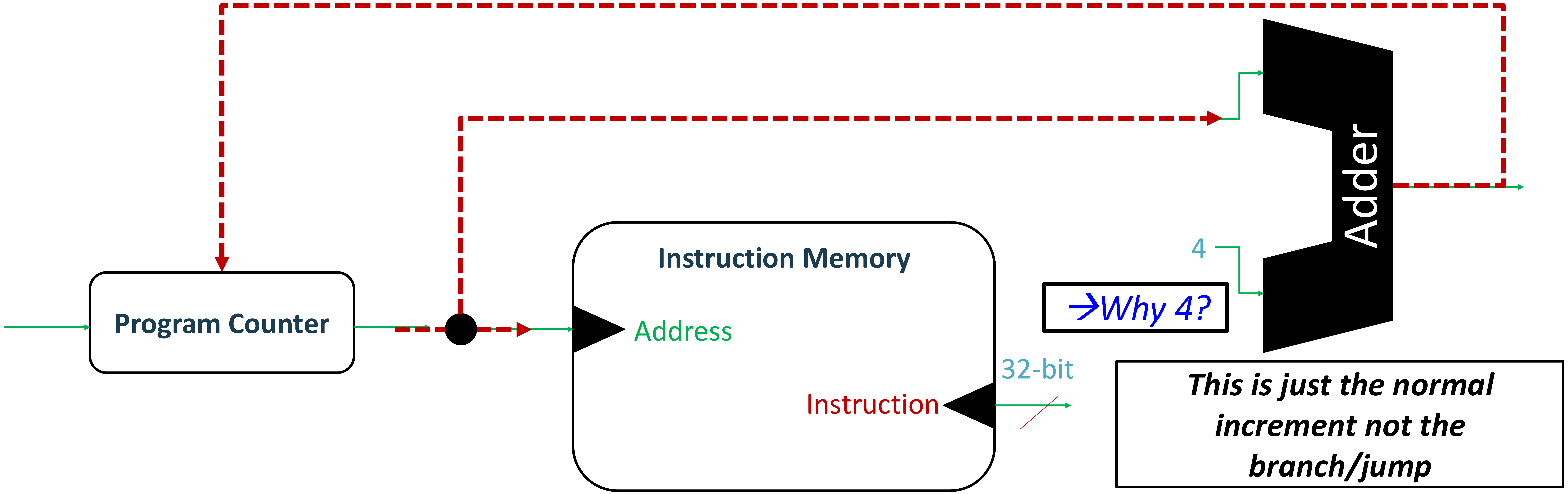
The Data Memory

- The memory unit is a state element with inputs for the address and the write data, and a single output for the read result.
- A sign extension unit is needed with a 16-bit input that is sign-extended into a 32-bit result appearing on the output.
- The sign-extend unit extends the 16-bit offset field in the instruction to a 32-bit signed value, and a data memory unit to read from or write to. The data memory must be written on store instructions; hence, data memory has read and write control signals, an address input, and an input for the data to be written into memory. *→ to balance the bits size for the ALU*



Fetching instructions and incrementing the program counter

- Here we will need the program counter, the instruction memory, and an adder.
- Let’s start the connection

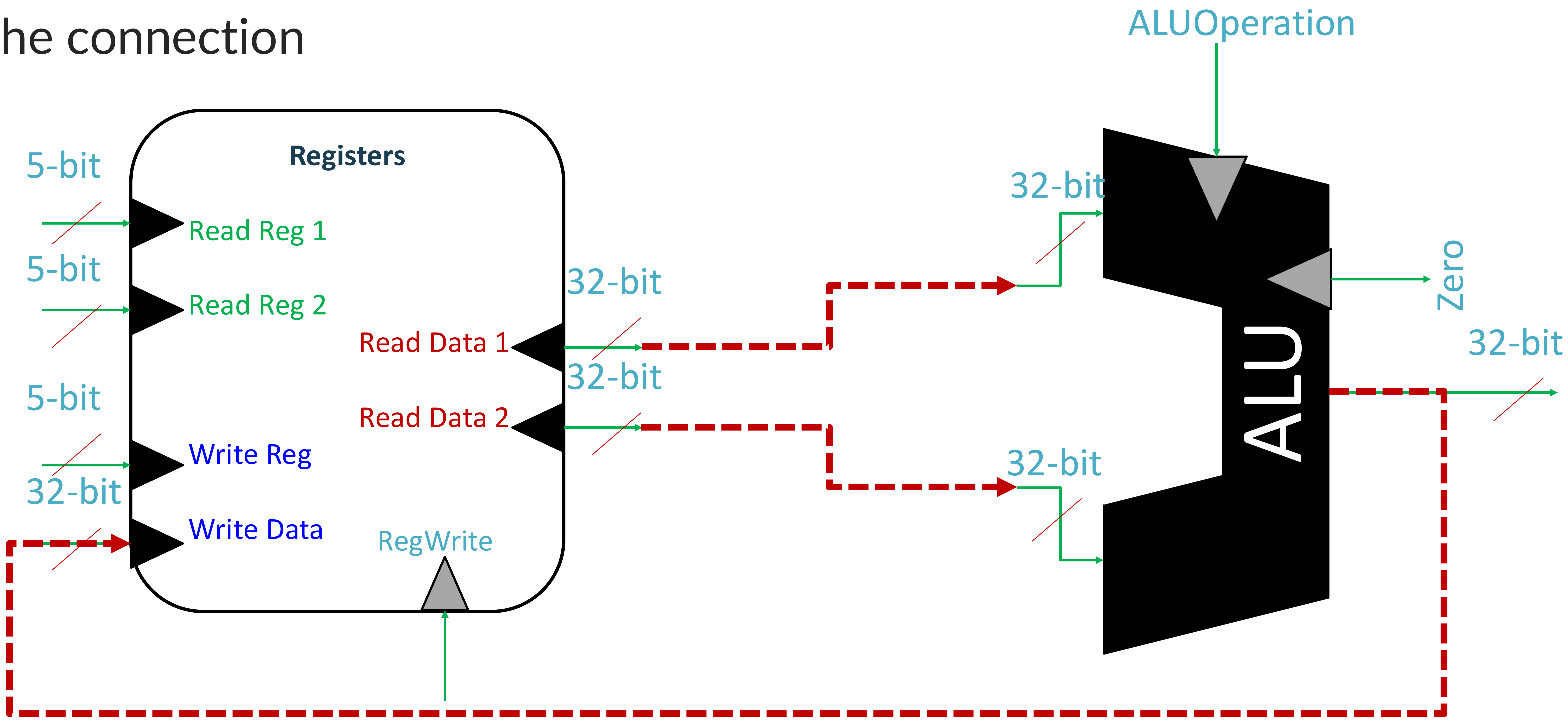


→Why 4?

This is just the normal increment not the branch/jump

Executing *R*-Format Instructions

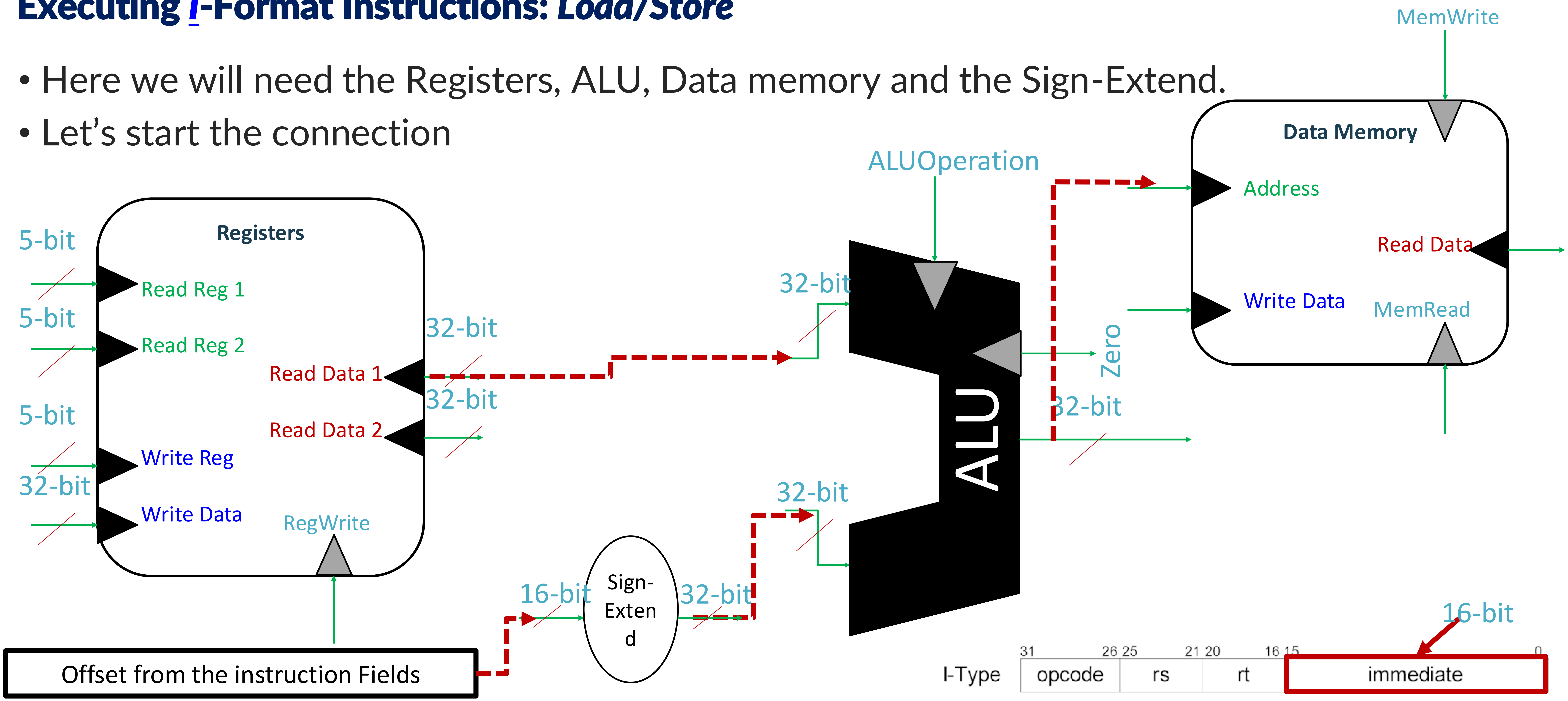
- Here we will need the Registers and the ALU.
- Let’s start the connection



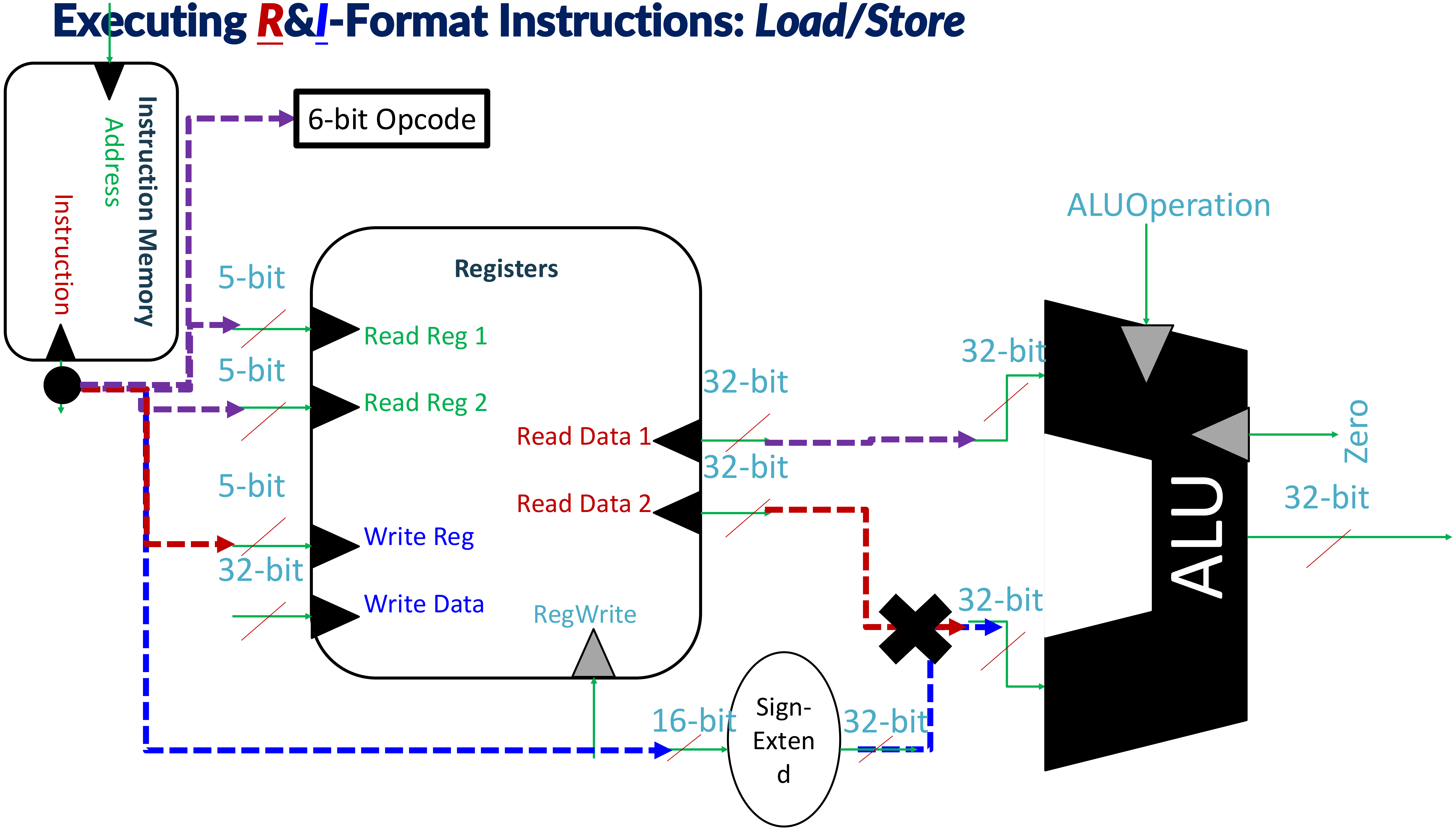
Building a Datapath

Executing *I*-Format Instructions: *Load/Store*

- Here we will need the Registers, ALU, Data memory and the Sign-Extend.
- Let's start the connection

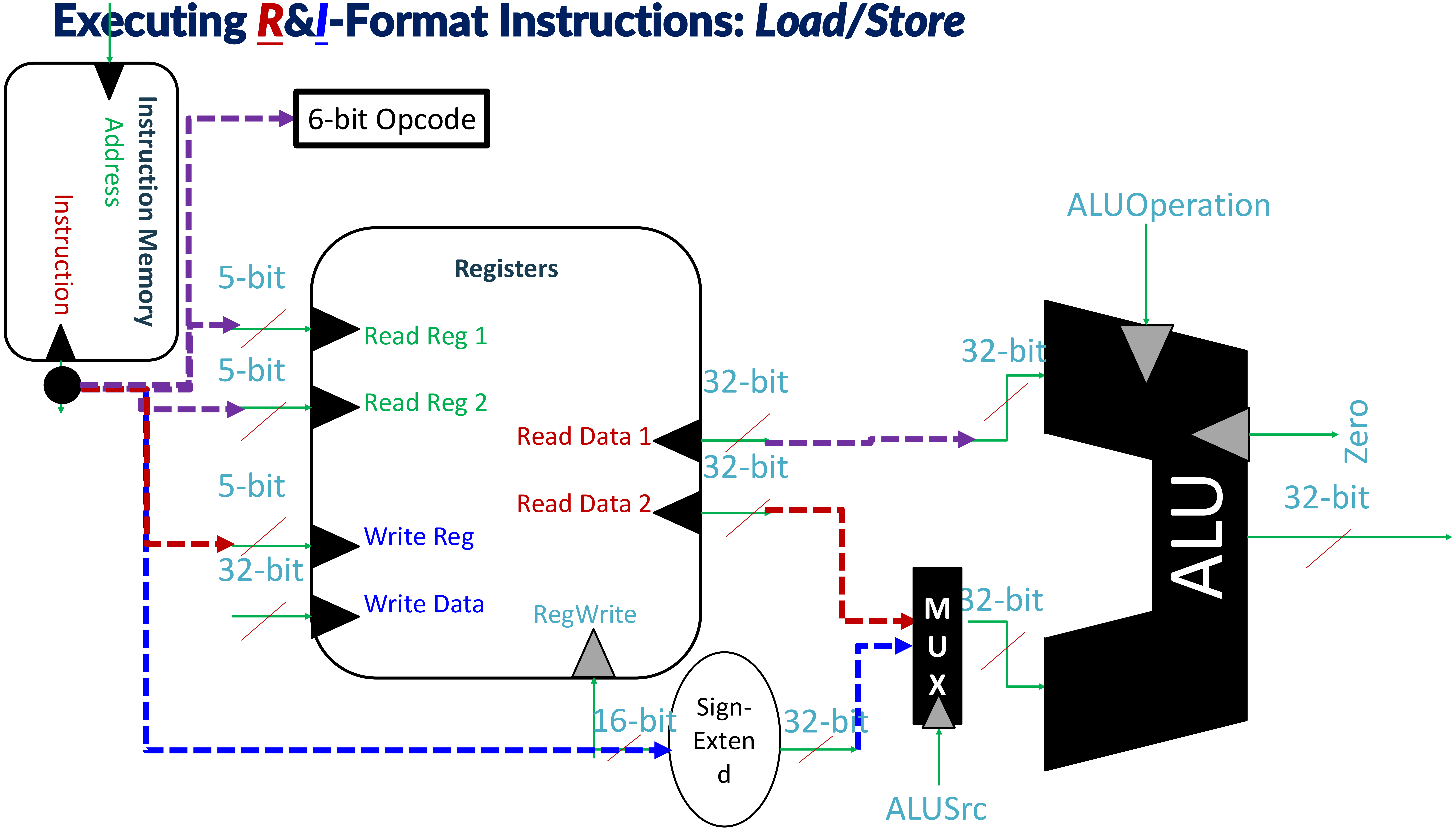


Executing R&I-Format Instructions: *Load/Store*



Building a Datapath

Executing R&I-Format Instructions: *Load/Store*



Building a Datapath

Executing R&I-Format Instructions: *Load/Store*

Instruction Memory
Address
Instruction
6-bit Opcode
5-bit
5-bit
5-bit
32-bit
Registers
Read Reg 1
Read Reg 2
Read Data 1
Read Data 2
Write Reg
Write Data
RegWrite
32-bit
32-bit
16-bit
Sign-Extend
32-bit
MUX
ALUSrc
32-bit
32-bit
ALUOperation
ALU
Zero
32-bit
MemWrite
Data Memory
Address
Read Data
Write Data
MemRead

Executing R&I-Format Instructions: *Load/Store*

MemWrite
Instruction Memory
Address
Instruction
6-bit Opcode
Registers
5-bit
Read Reg 1
5-bit
Read Reg 2
Read Data 1
Read Data 2
5-bit
Write Reg
32-bit
Write Data
RegWrite
32-bit
32-bit
ALUOperation
32-bit
ALU
Zero
32-bit
Data Memory
Address
Read Data
Write Data
MemRead
M
U
X
32-bit
ALUSrc
Sign-
Exten
d
16-bit
32-bit
M
U
X
MemtoReg

Building a Datapath

Executing *Branching* Instruction

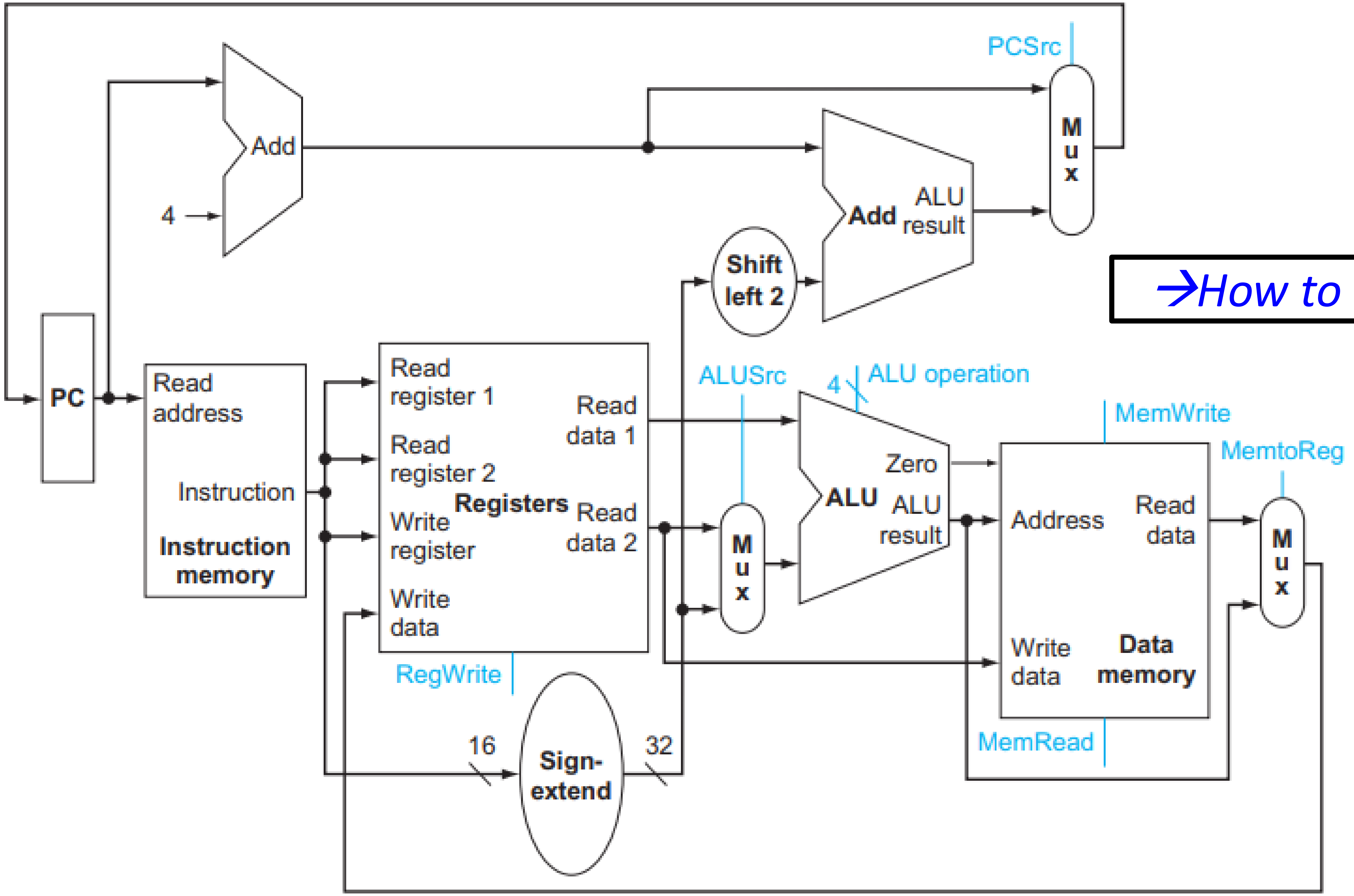
- Here we will need the program counter, the registers, and the ALU.

→Seen it before?
4
Adder
Adder
6-bit Opcode
16-bit offset in words
16-bit
Sign-Extend
32-bit
Left Shift 2
→Why?
5-bit
5-bit
5-bit
32-bit
Registers
Read Reg 1
Read Reg 2
Write Reg
Write Data
Read Data 1
Read Data 2
RegWrite
32-bit
32-bit
Instruction Memory
Address
Instruction
Program Counter
ALUOperation
32-bit
32-bit
ALU
Zero
32-bit

Building a Datapath

Let's put R/I/Branch/PC increment together

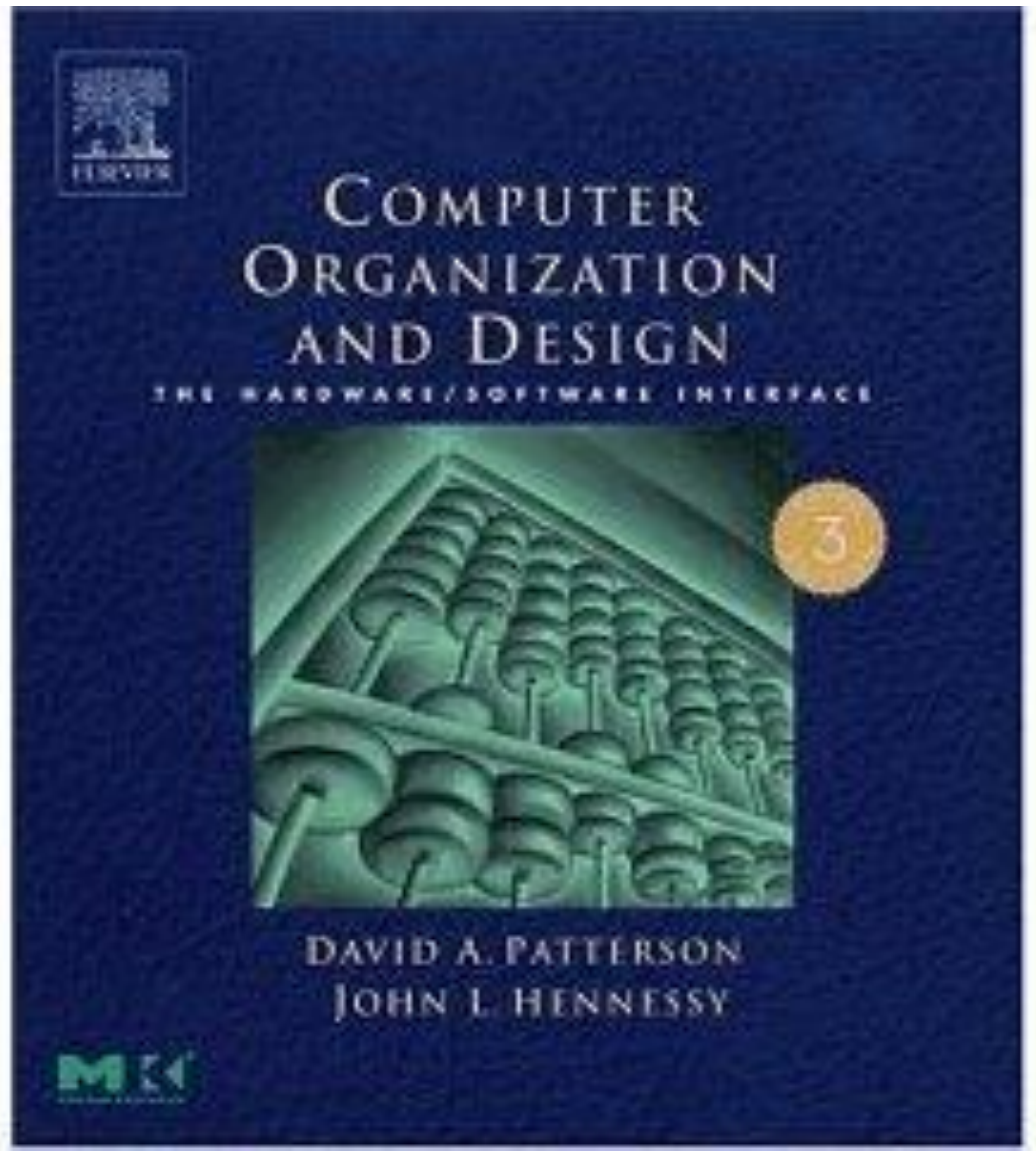
This is called a single-cycle machine



→How to control the ALU?

Reference

- Patterson, D. A., Hennessy, J. L. (2004). Computer Organization and Design: The Hardware/Software Interface. Netherlands: Morgan Kaufmann. (Chapter 4)


COMPUTER ORGANIZATION AND DESIGN
THE HARDWARE/SOFTWARE INTERFACE
3
DAVID A. PATTERSON
JOHN L. HENNESSY
MK

For Further Inquiries, Please

✉ ***send an email***

Catherine.elias@guc.edu.eg,
Catherine.elias@ieee.org

Thank you for your attention!

See you next time ☺