The Processor

Dr. Rajib Ranjan Maiti

CSIS, BITS-Pilani, Hyderabad

Performance of computer

- Parameters to evaluated performance
  - instruction count,
  - clock cycle time, and
  - clock cycles per instruction (CPI)
- The instruction set architecture
  - determines the instruction count required for a given program

Performance of computer

- Parameters to evaluated performance
  - instruction count,
  - clock cycle time, and
  - clock cycles per instruction (CPI)
- The instruction set architecture
  - determines the instruction count required for a given program
- the implementation of the processor
  - determines both the clock cycle time and
  - the number of clock cycles per instruction
- two different implementations of the MIPS instruction set
  - Non-pipelined MIPS implementation
  - pipelined MIPS implementation

Basic MIPS Implementation

- We shall examine the implementation of some of MIPS instructions, namely
  - *load word* (lw) and *store word* (sw)
  - *Arithmetic:* add, sub, AND, OR, and slt
  - *branch equal* (beq) and *jump* (j)
- Our focus
  - the key principles used in creating a datapath and designing the control

Basic MIPS Implementation

- How the instruction set architecture determines different aspects of the implementation?
- How an implementation can affect the clock rate and CPI for the computer?
- Most concepts used to implement the MIPS subset are
  - the same basic ideas that are used to construct a broad spectrum of computers
    - from high-performance servers
    - to general-purpose microprocessors
    - to embedded processors

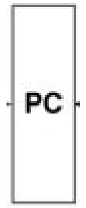
Basic MIPS Implementation

- 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.
- After these two steps,
  - the actions required to complete the instruction depend on the instruction class.

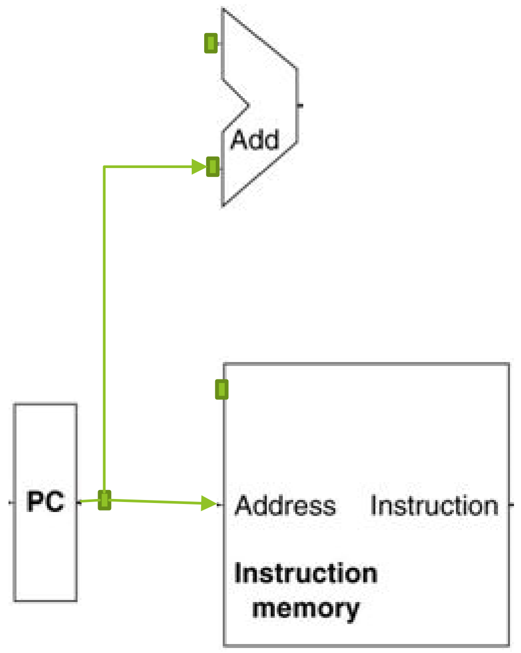
Basic MIPS Implementation

- 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 ALU after reading the registers.
  - ALU is used by
    - the memory-reference instructions to calculate an address calculation,
    - the arithmetic-logical instructions to execute the operation, and
    - Branch instructions to compare
  - A memory-reference instruction will need to access the memory either to read data for a load or write data for a store.
  - An R-type or load instruction must write the data from the ALU or memory back respectively into a register.
  - A branch instruction may 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.

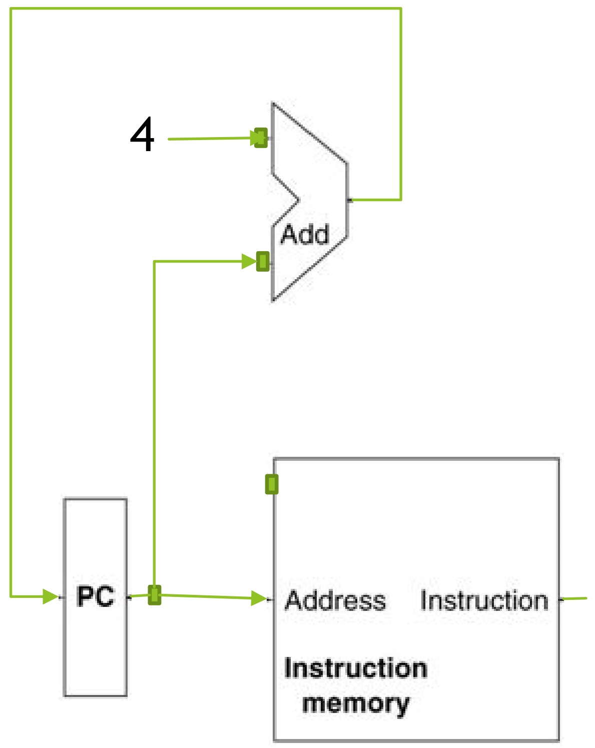
Functional units and their interconnection in MIPS implementation



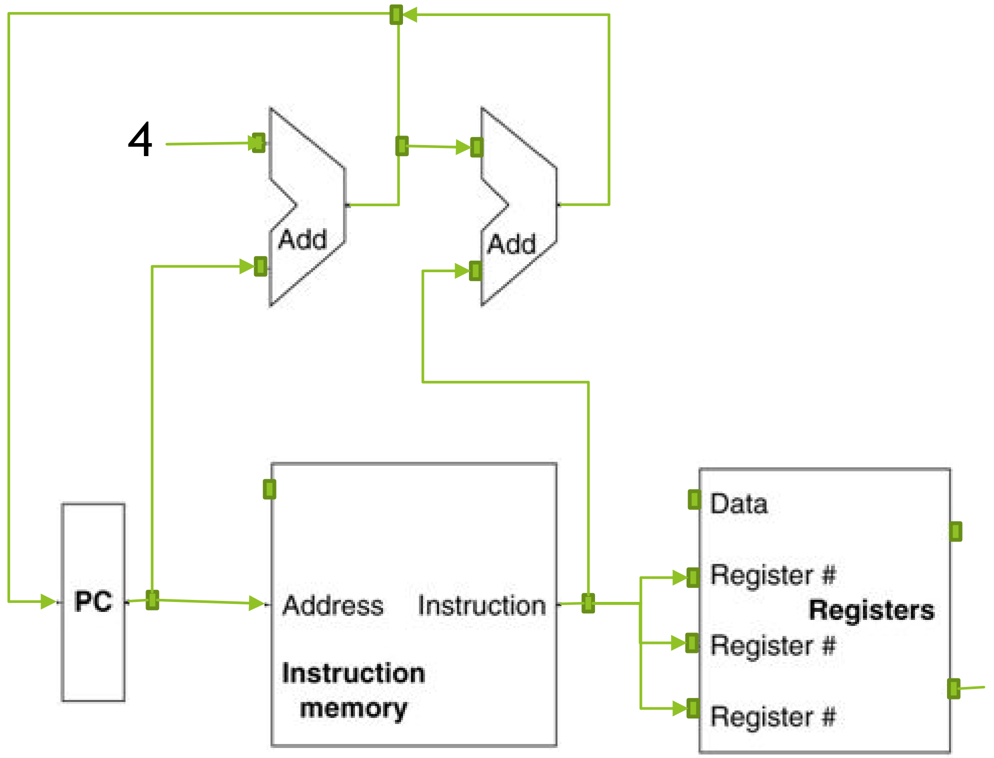
Functional units and their interconnection in MIPS implementation



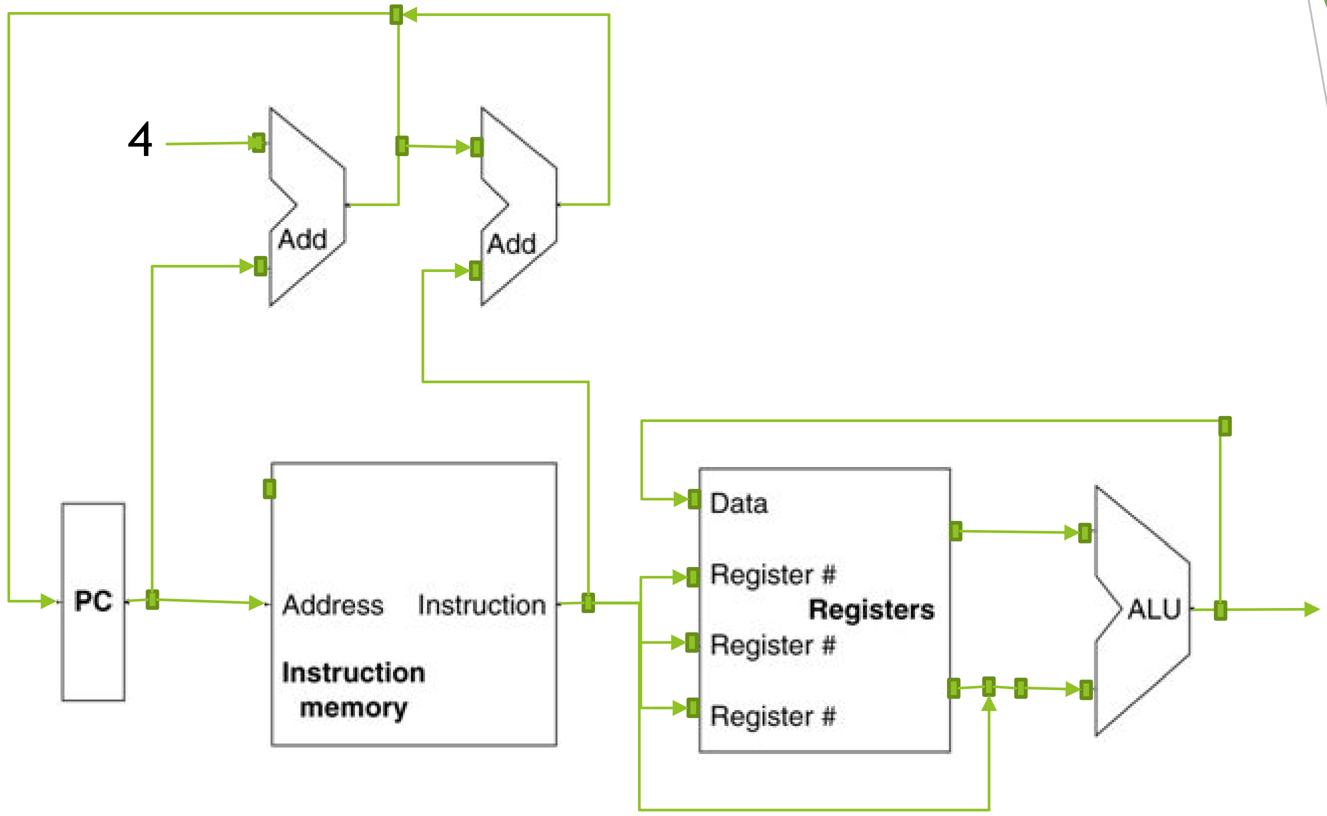
Functional units and their interconnection in MIPS implementation



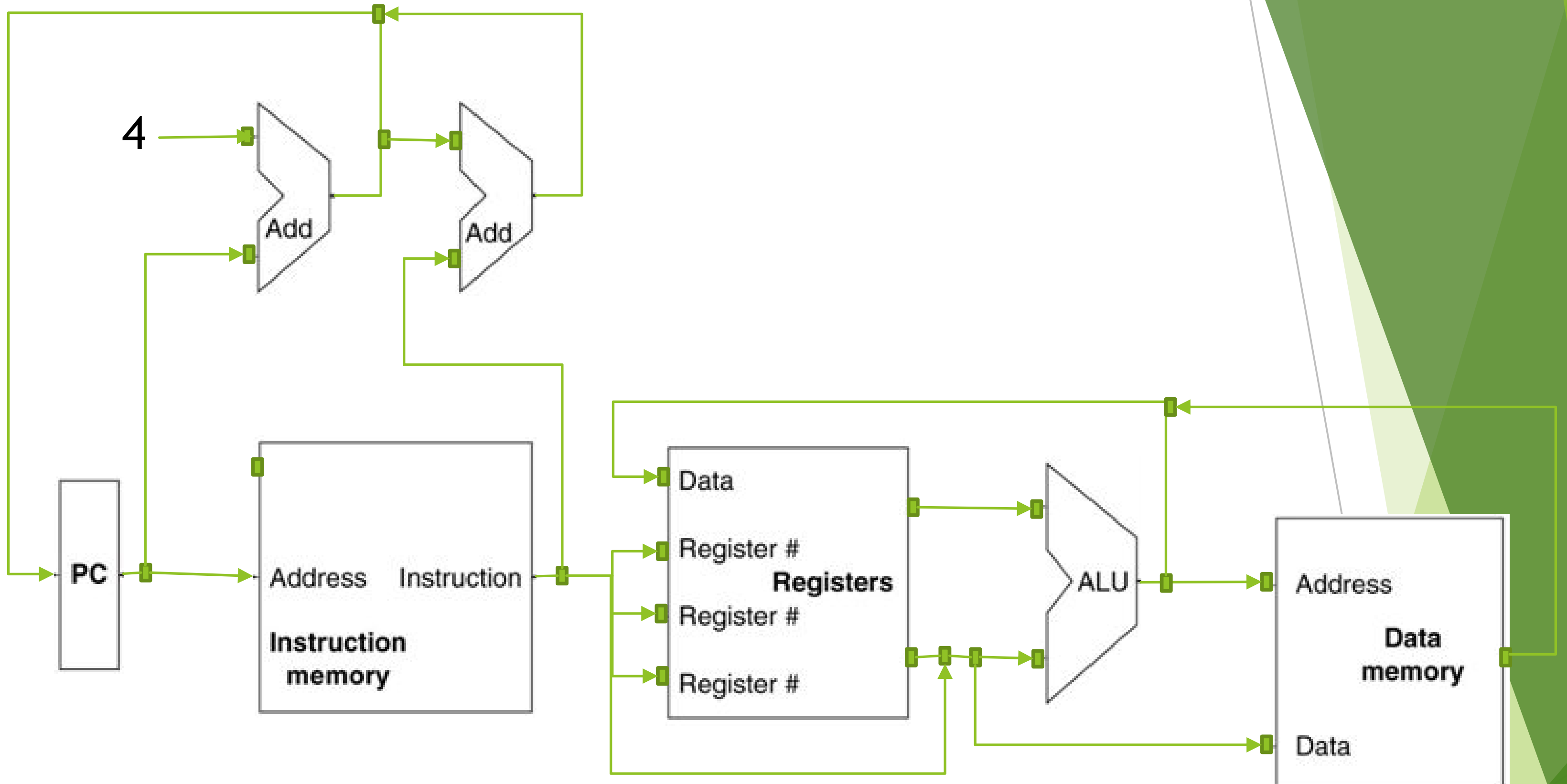
Functional units and their interconnection in MIPS implementation



Functional units and their interconnection in MIPS implementation

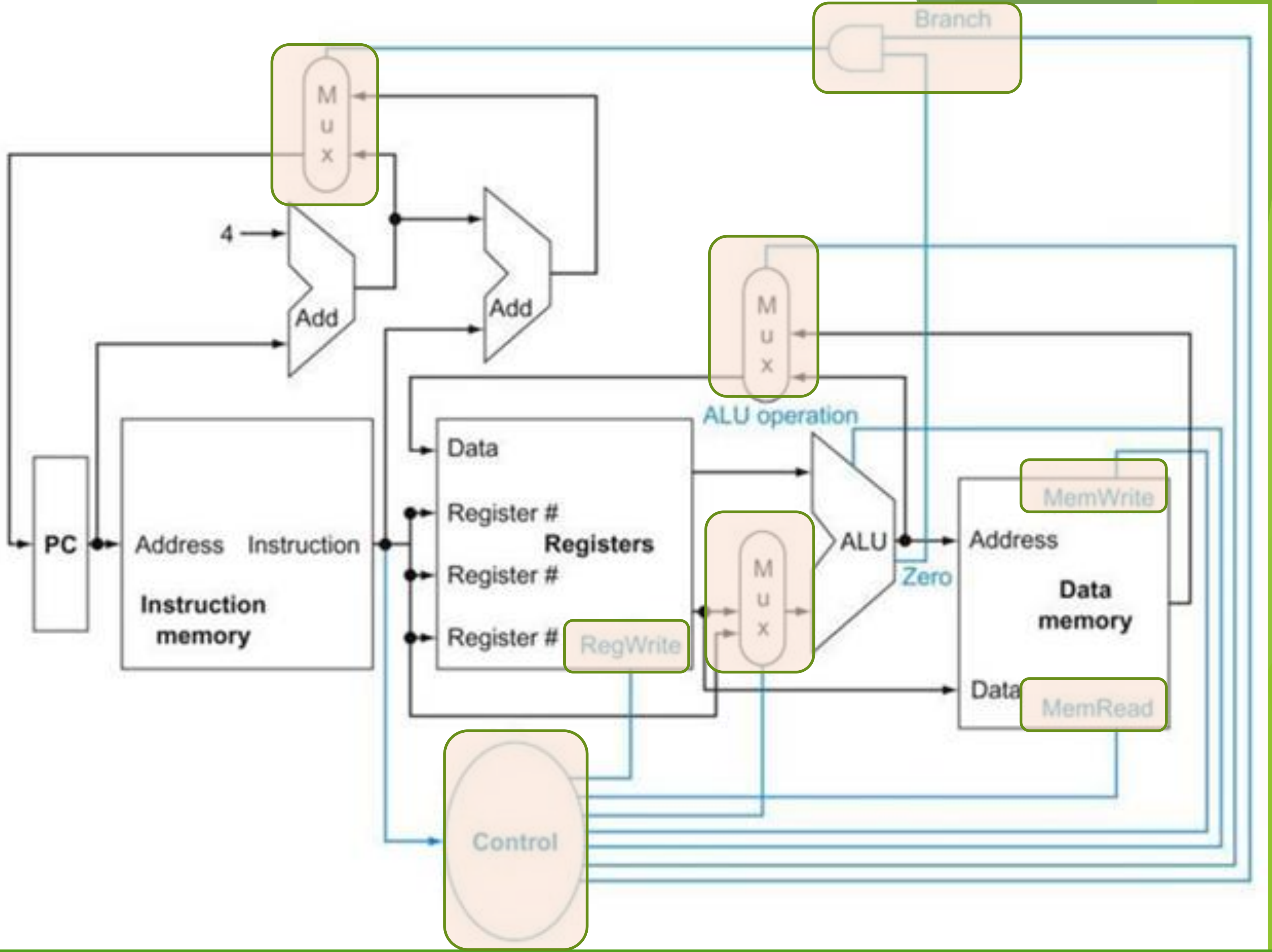


Functional units and their interconnection in MIPS implementation



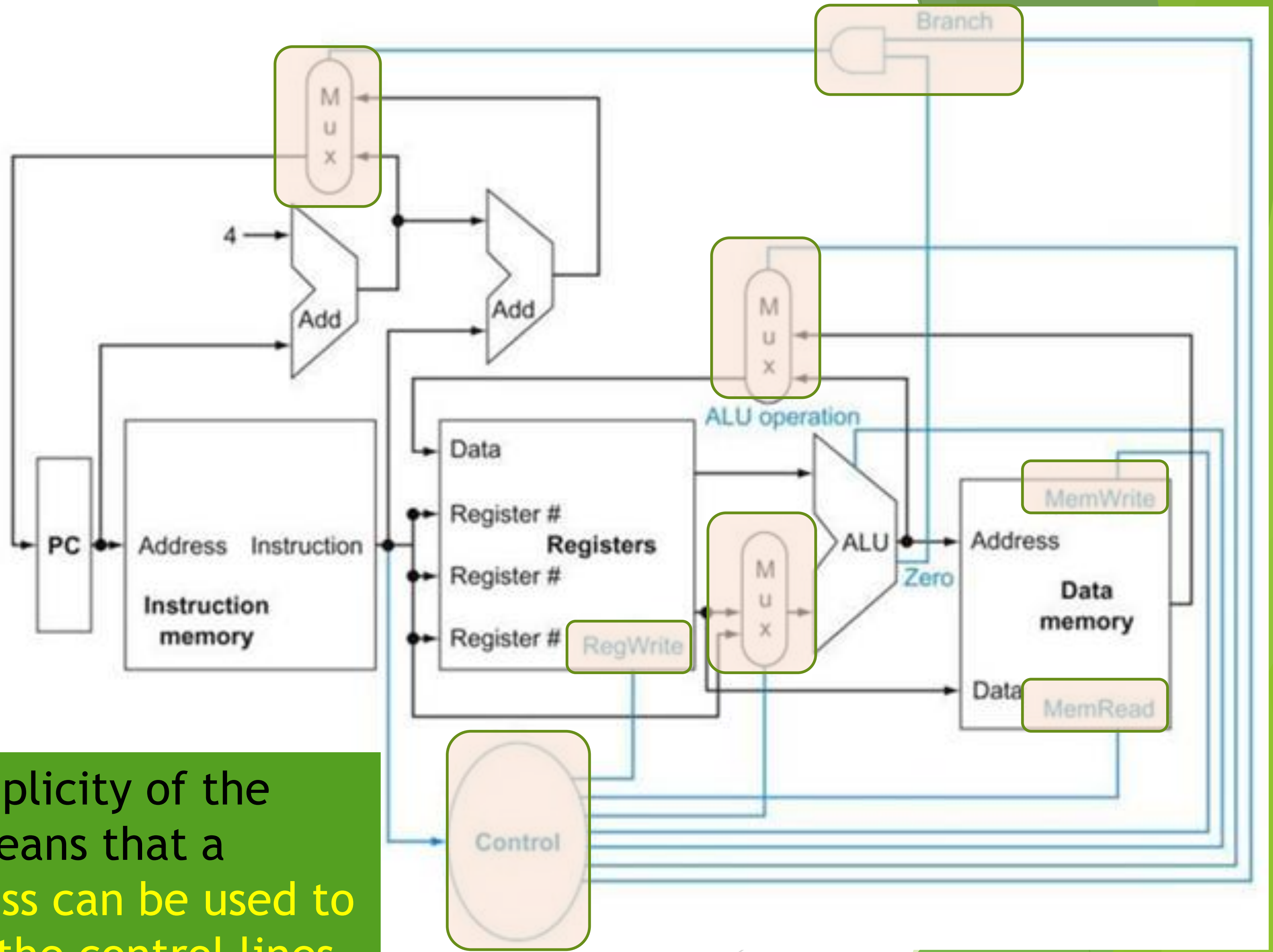
data going to a particular unit as coming from two different sources

The datapath of MIPS implementation



data going to a particular unit as coming from two different sources: use MUX

The datapath of MIPS implementation



The regularity and simplicity of the MIPS instruction set means that a simple decoding process can be used to determine how to set the control lines.

Logic Design Conventions

- Two questions
  - **how the hardware logic will operate?**
  - how the computer is clocked?

- Two different types of logic elements:
  - **Combinational:** the outputs depend only on the current inputs, **e.g., ALU**
    - Given the same input, a combinational element always produces the same output
  - **State elements:** have some internal storage,
    - For example, the instruction and data memories, and the registers
    - A state element has at least two inputs and one output
    - Inputs: the data value to be written and the clock

Logic Design Conventions

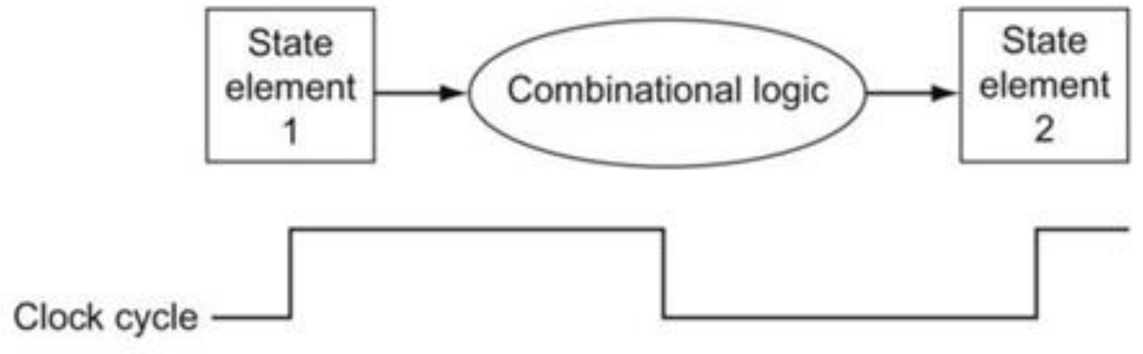
- Two questions
  - how the hardware logic will operate?
  - **how the computer is clocked?**

- At any time, the **clock** is used to **determine**
  - when the **state element** should be **written** or
  - a **state element** can be **read**
- **Clocking Methodology** defines
  - when signals can be read and when they can be written

Logic Design Conventions

- We assume an **edge-triggered clocking** methodology, i.e.,
  - any update in a **sequential logic** takes place **only on a clock edge**
  - Any collection of **combinational logic must have its inputs come from a set of state elements** and its **outputs written into a set of state elements**
  - The **inputs** are values that were **written in a previous clock cycle**,
    - while the **outputs** are values that can be **used in a following clock cycle**

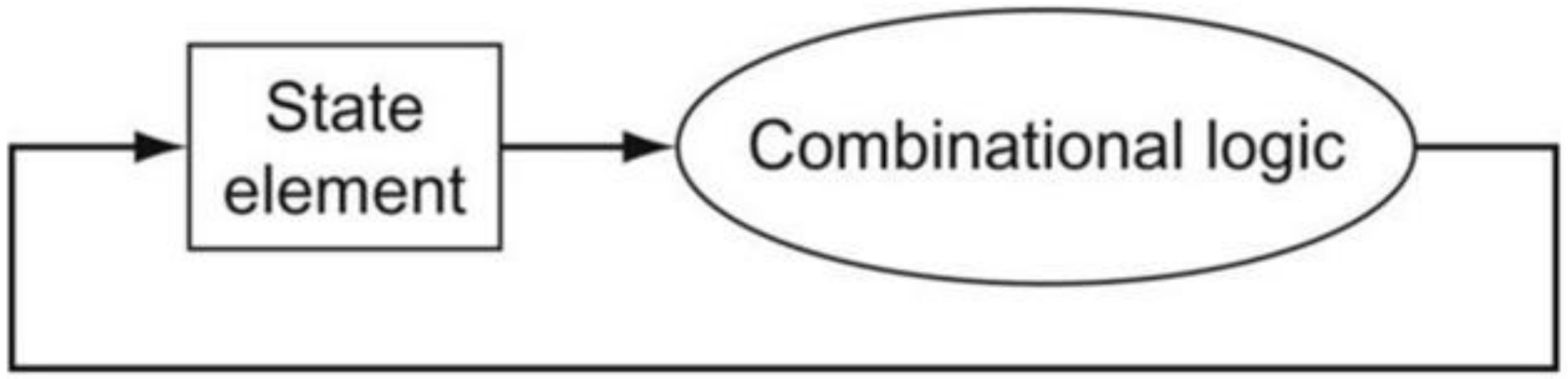
Logic Design Conventions



the two state elements surrounding a block of combinational logic, which operates in a single clock cycle: all signals must propagate from state element 1, through the combinational logic, and to state element 2 in the time of one clock cycle.

The time necessary for the signals to reach state element 2 defines the length of the clock cycle.

Logic Design Conventions



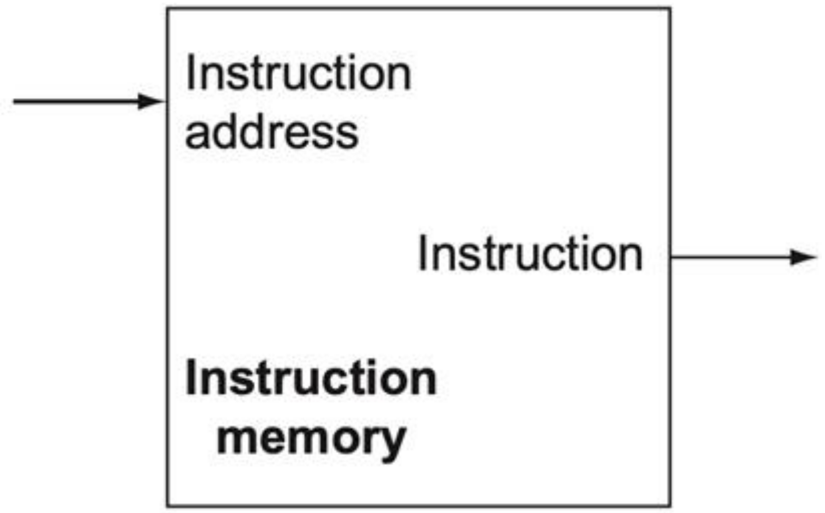
An edge-triggered methodology allows us to read the contents of a register, send the value through some combinational logic, and write that register in the same clock cycle.

Building a Datapath

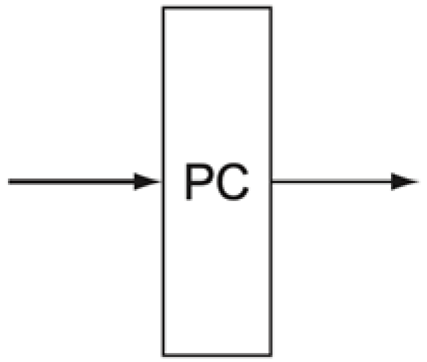
Datapath for Instruction fetch

- **to start** a datapath design
  - examine the **major components required** to execute **each class of MIPS** instructions
- Some Components are required for every instruction :
  - **instruction fetch**
- A **memory** unit
  - stores the instructions of a program
  - supplies an instruction given an address
- The **program counter (PC)**
  - a register that holds the address of the current instruction
- An **adder**
  - to increment the PC to the address of the next instruction

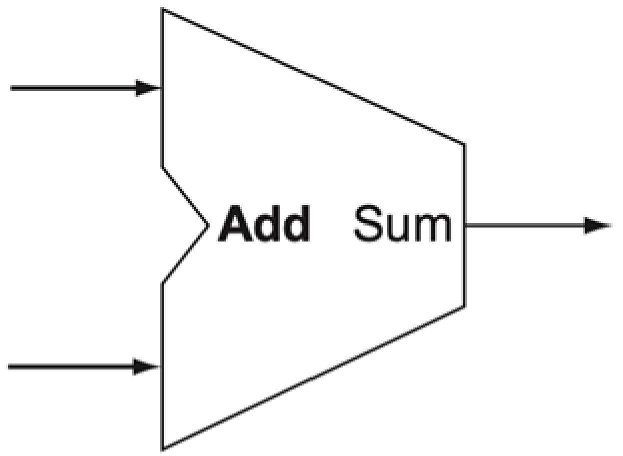
Datapath for Instruction fetch



State element 1



State element 2



State element 2

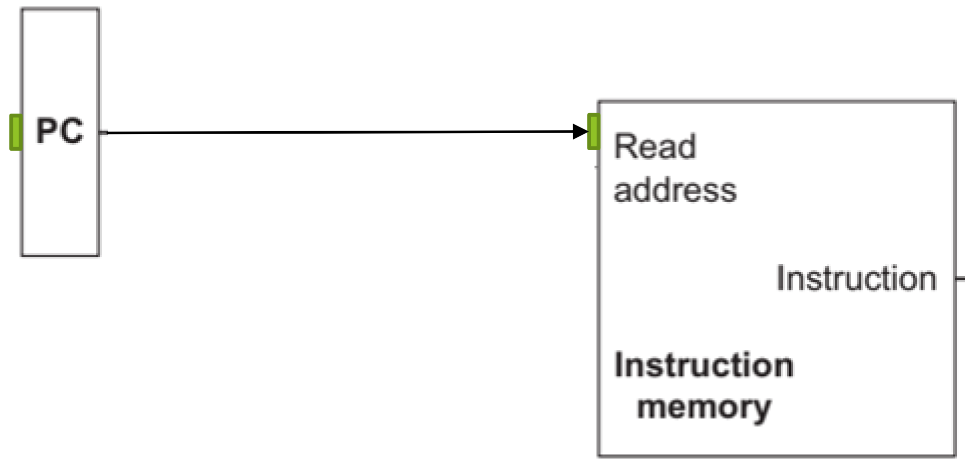
Since the instruction memory only reads, we treat it as combinational logic:
the output at any time reflects the contents of the location specified by the address input, and no read control signal is needed.

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.

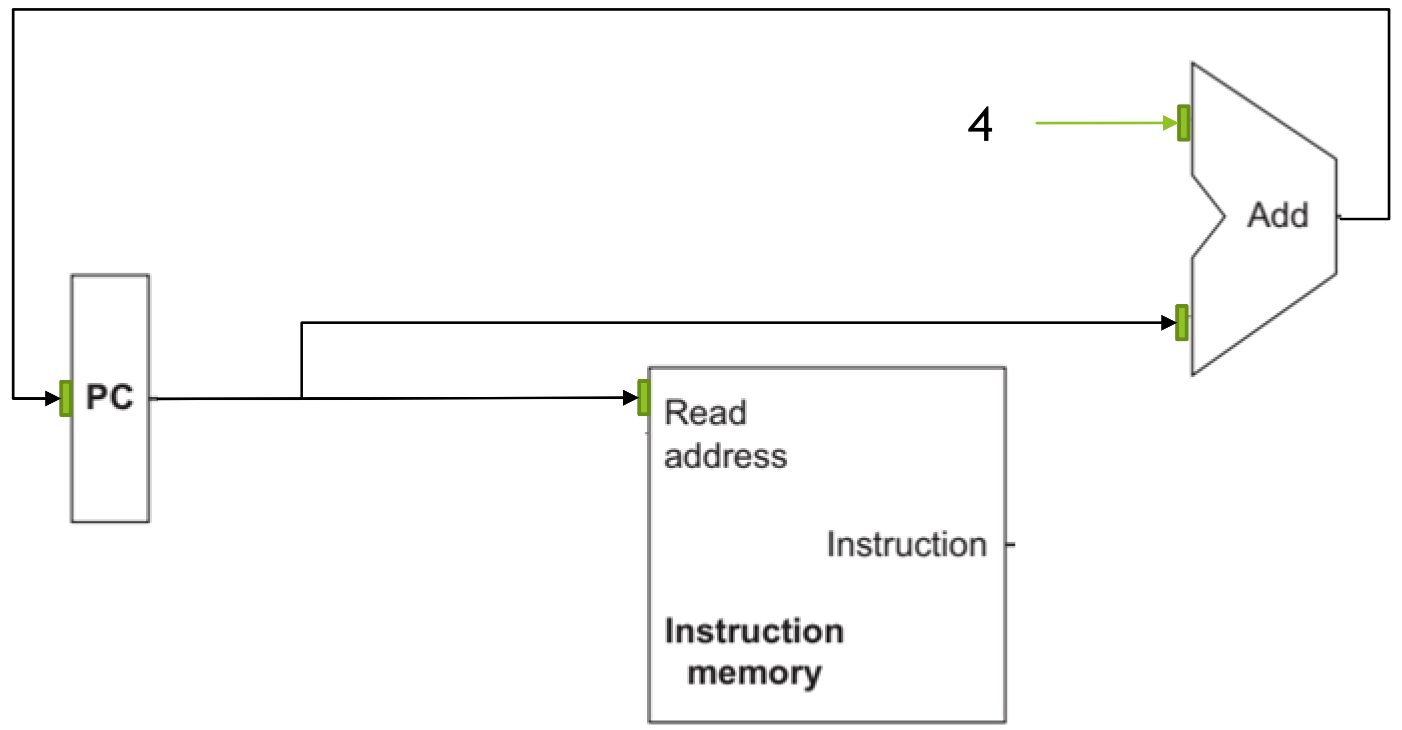
The adder is an ALU wired to always add its two 32-bit inputs and place the sum on its output.

How do we combine these datapath elements?

How do we combine these datapath elements?



How do we combine these datapath elements?



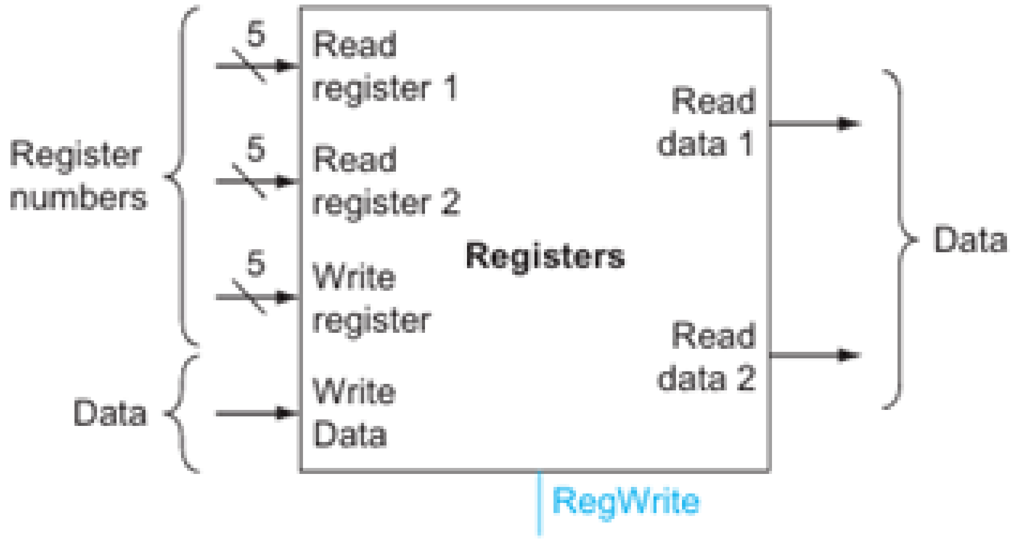
Building a Datapath: Execution of R-Type instruction

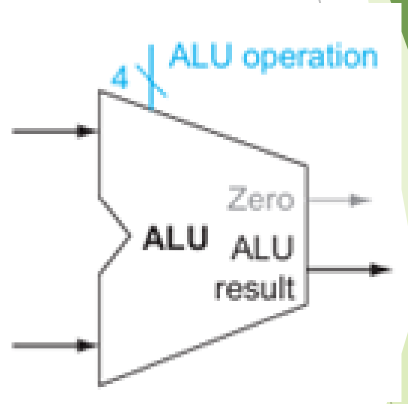
- **The task**
  - **Read two registers**,
  - Perform an **ALU operation** on the contents of the registers, and
  - **Write** the result **to a register**

- Example of R-Type instructions
  - add $t1, $t2, $t3
  - sub $t2, $s0, $t3
  - and $t2, $s0, $t3
  - or $t2, $s0, $t3
  - slt $s1, $s2, $s3   # if ($s2 < $s3) $s1= 1; else $s1 = 0

- MIPS elements
  - 32 general-purpose registers are stored in a structure called a **register file**
  - **An ALU** to operate on the values read from the registers

Building a Datapath: Execution of R-Type instruction





Register file: A state element that consists of a set of registers that can be read and written by supplying a register number

A total of 4 inputs (3 for register numbers + 1 for data)

2 outputs (both for data)

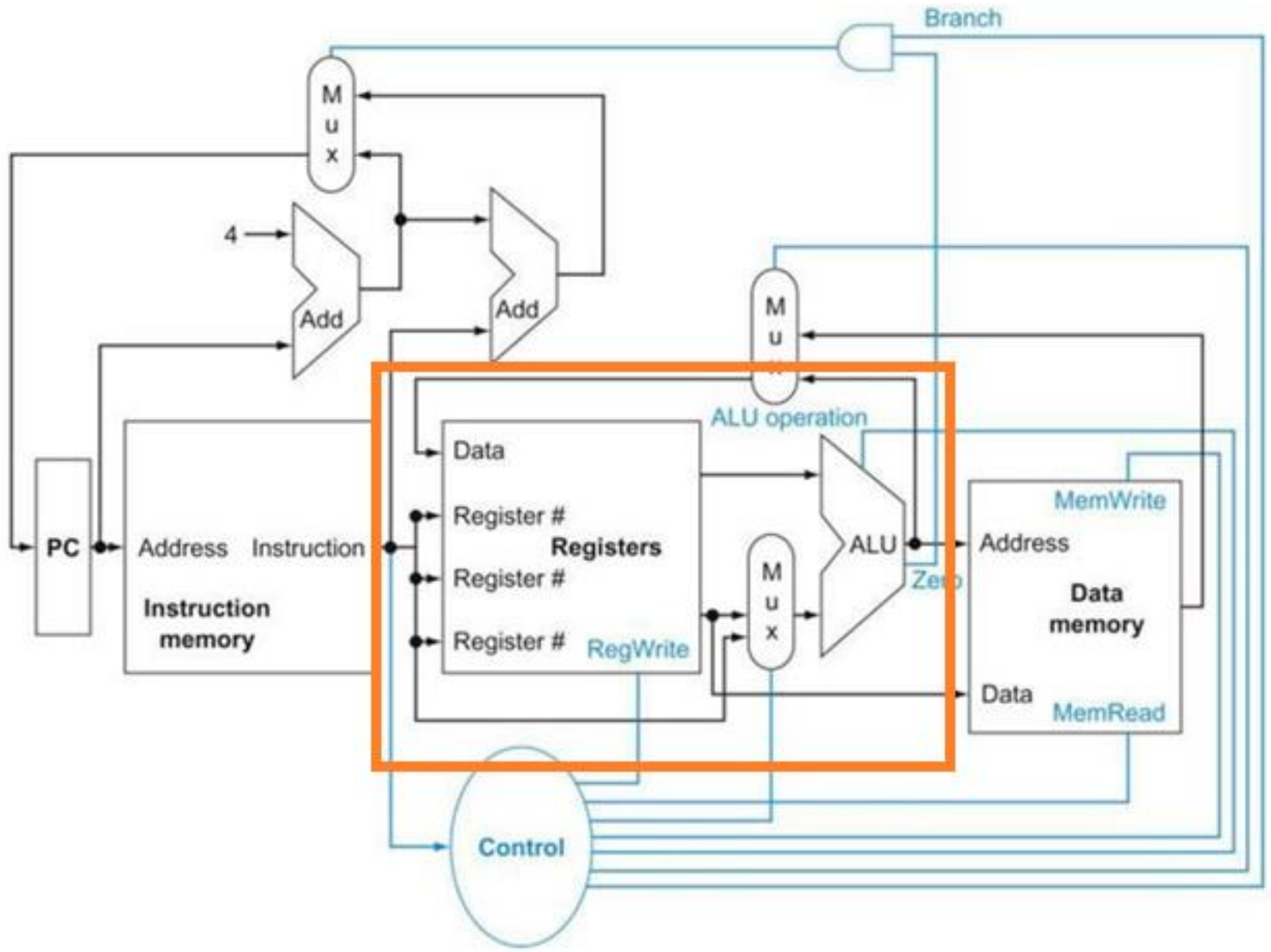
Two 32-bit inputs

A 32-bit output

1-bit signal if the output is 0

The 4-bit control signal

Building a Datapath: Execution of R-Type instruction



Thank You