CCSYA

The Processor
Single-cycle Implementation
Part 1

Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto

Luís Nogueira (lmn@isep.ipp.pt)

Introduction

- **CPU performance factors**
  - Instruction count - Determined by ISA and compiler
  - CPI and Cycle time - Determined by CPU hardware

- **We will examine two MIPS implementations**
  - A simplified version
  - A more realistic pipelined version (next lectures)

- **Simple subset (shows most aspects)**
  - Memory reference: `lw`, `sw`
  - Arithmetic/logical: `add`, `sub`, `and`, `or`, `slt`
  - Control transfer: `beq`, `j`

Introduction

- **In examining the implementation, we will have the opportunity to see:**
  - How the ISA determines many aspects of the implementation
  - How the choice of various implementation strategies affects the clock rate and CPI
  - How many of the key design principles, such as *Simplicity favors regularity*, can be illustrated

- **In addition, most concepts used to implement the MIPS subset are used to construct a broad spectrum of computers**
  - From high-performance servers to general-purpose microprocessors to embedded processors

Logic Design Conventions

- **To discuss the design of a computer, we must decide:**
  - How the hardware logic implementing the computer will operate
  - How the computer is clocked

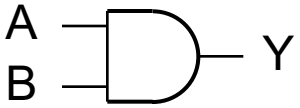
- **The datapath elements in the MIPS implementation consist of two different types of logic elements:**
  - Elements that operate on data values (combinational)
  - Elements that contain state (sequential)

Logic Design Basics

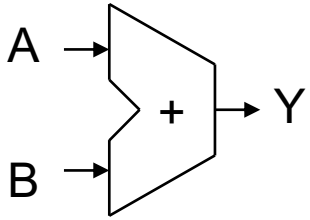
- **Information encoded in binary**
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses
- **Combinational element**
  - Operate on data
  - Output is a function of input
- **State (sequential) elements**
  - Store information
  - Output can depend on both the inputs and the value stored in memory, which is called the state of the block

Combinational Elements

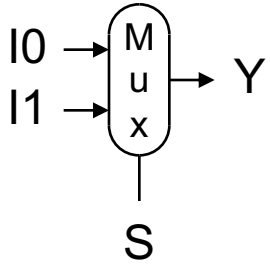
- **AND-gate**
  - Y = A & B



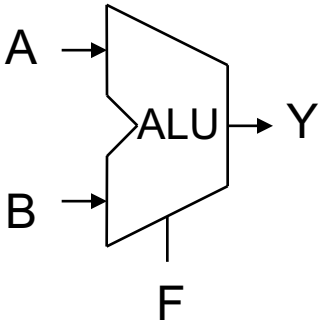
- **Adder**
  - Y = A + B



- **Multiplexer**
  - Y = S ? I1 : I0

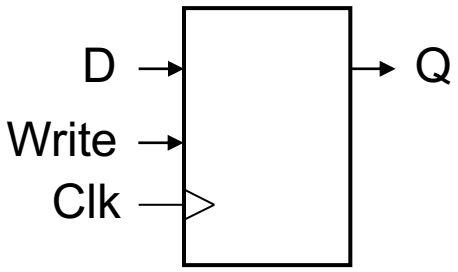


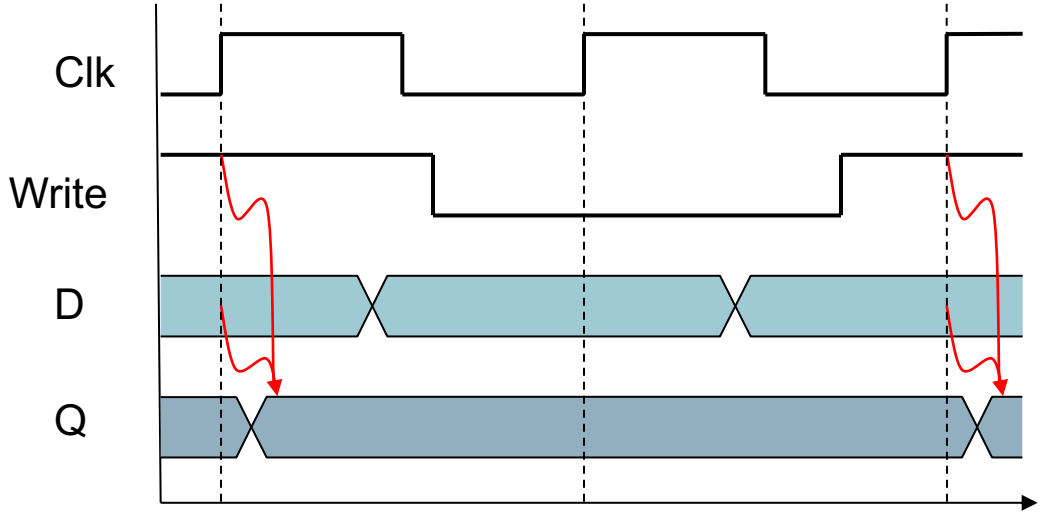
- **Arithmetic/Logic Unit**
  - Y = F(A, B)



Sequential Elements

- **Stores data in a circuit**
  - Uses a clock signal to determine when to update the stored value
  - A clock is simply a free-running signal with a fixed cycle time (or clock period)
- **A clocking methodology is designed to make hardware predictable**
  - **Edge-triggered**: update when Clk changes from 0 to 1

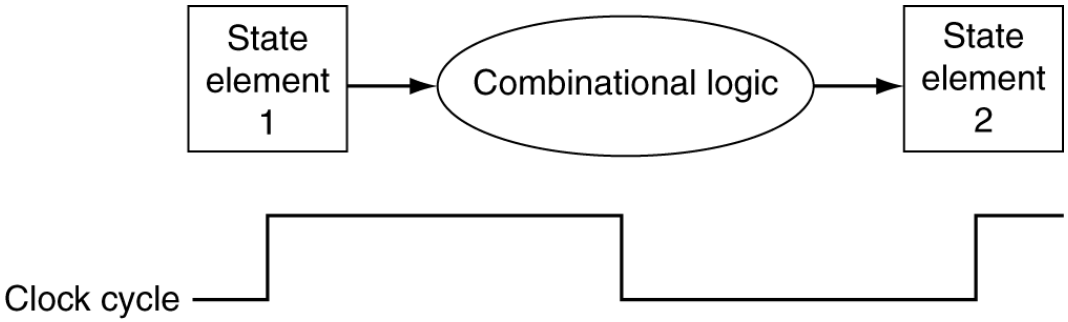




Clocking Methodology

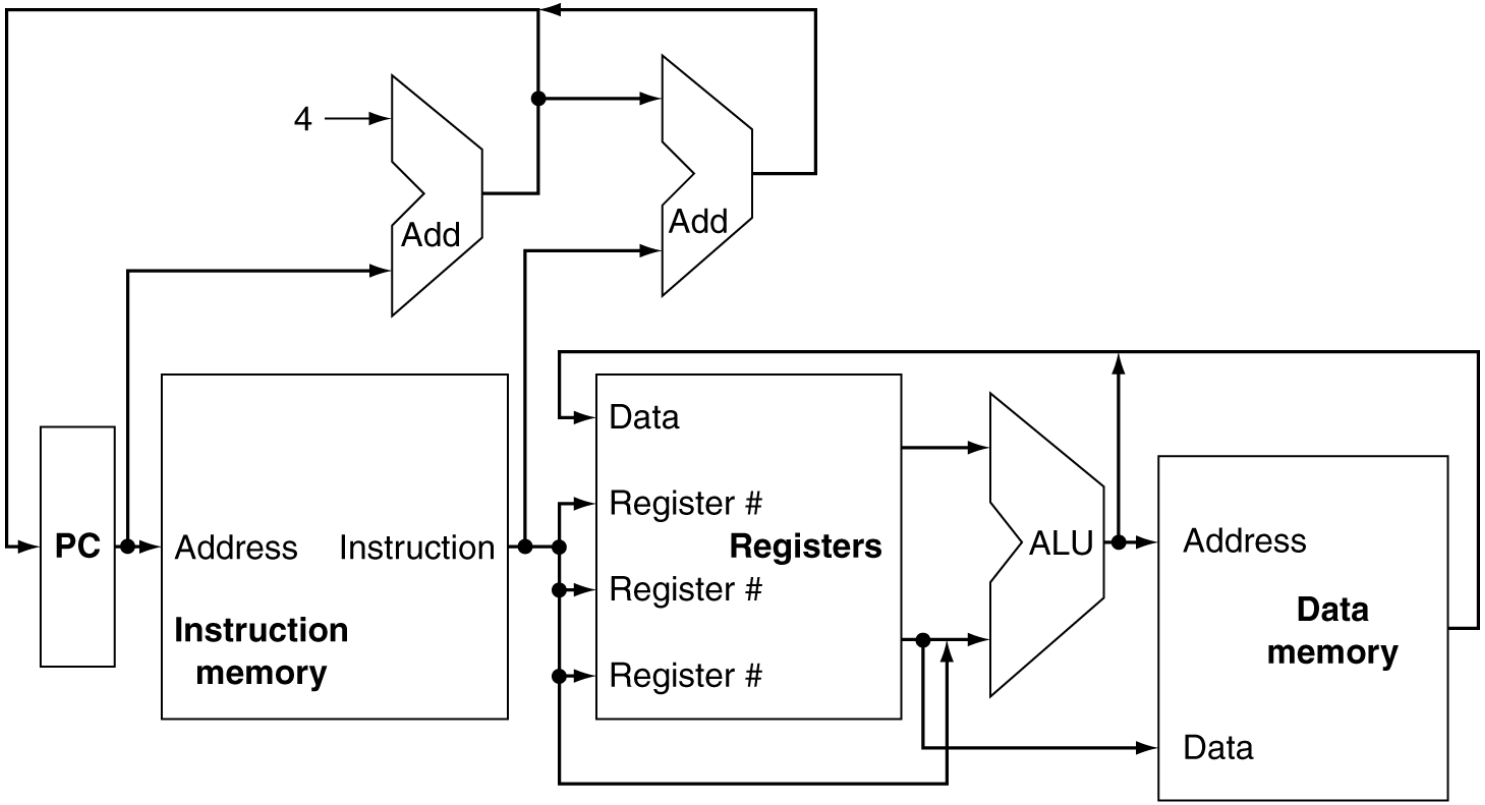
- **The clock must have a long enough period so that all the signals in the combinational logic block stabilize**
  - To ensure that the values written into the state elements on the active clock edge are valid

- **This constraint sets a lower bound on the length of the clock period**
  - Which must be long enough for all state element inputs to be valid



Building a Datapath

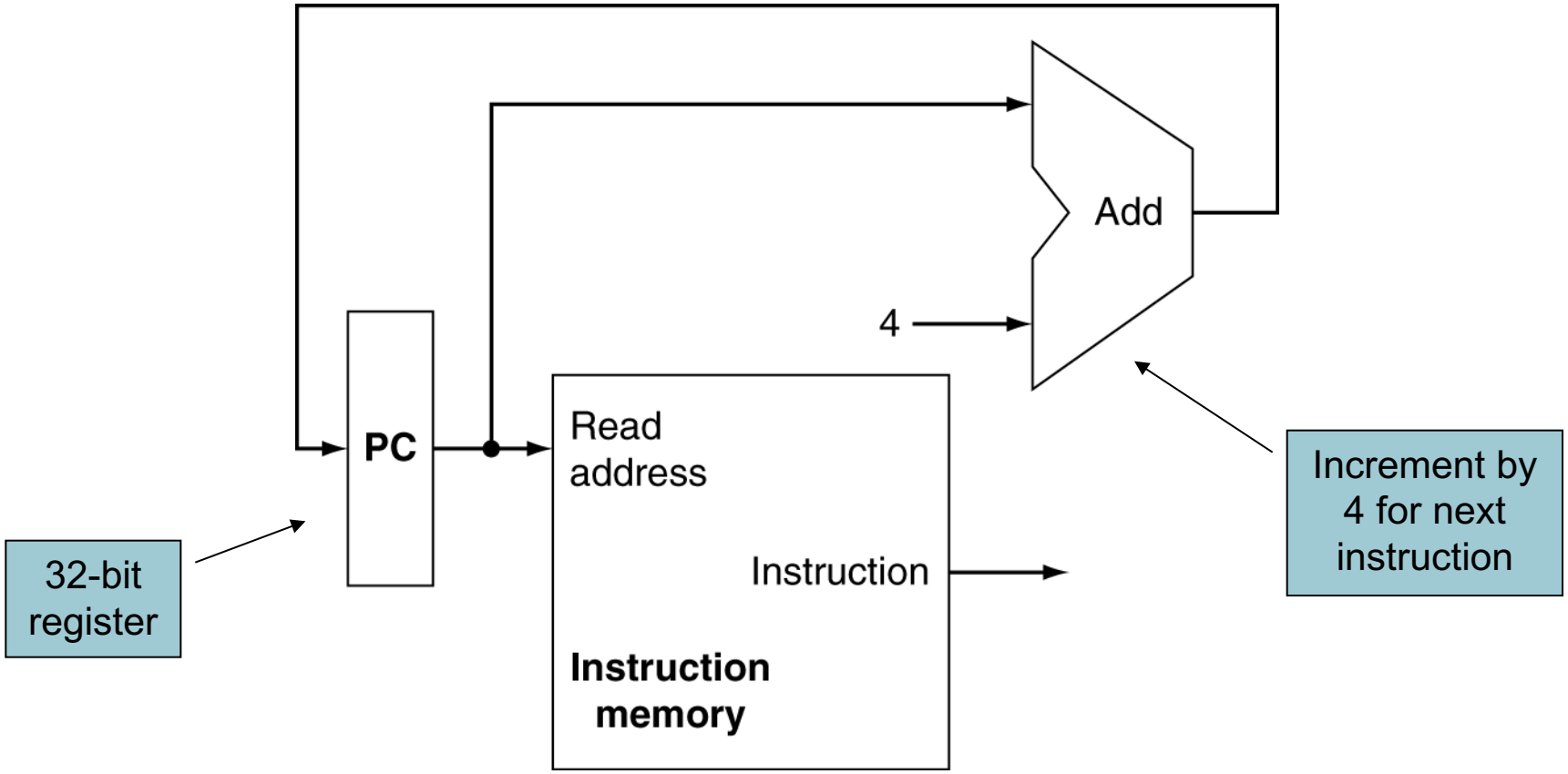
- **Datapath – elements that process data and addresses in the CPU**
  - Registers, ALUs, multiplexers, memories, …



Datapath – Instruction Fetch

- **To execute any instruction, we must start by fetching the instruction from memory**
- **To prepare for executing the next instruction, we must also increment the program counter**
  - So that it points at the next instruction, 4 bytes later
- **Fetching instructions thus requires:**
  - **Memory unit** where instructions are stored
  - **Program counter**(PC) to hold the address of the current/next instruction
  - **Adder** to increment the PC to the address of the next instruction

Instruction Fetch

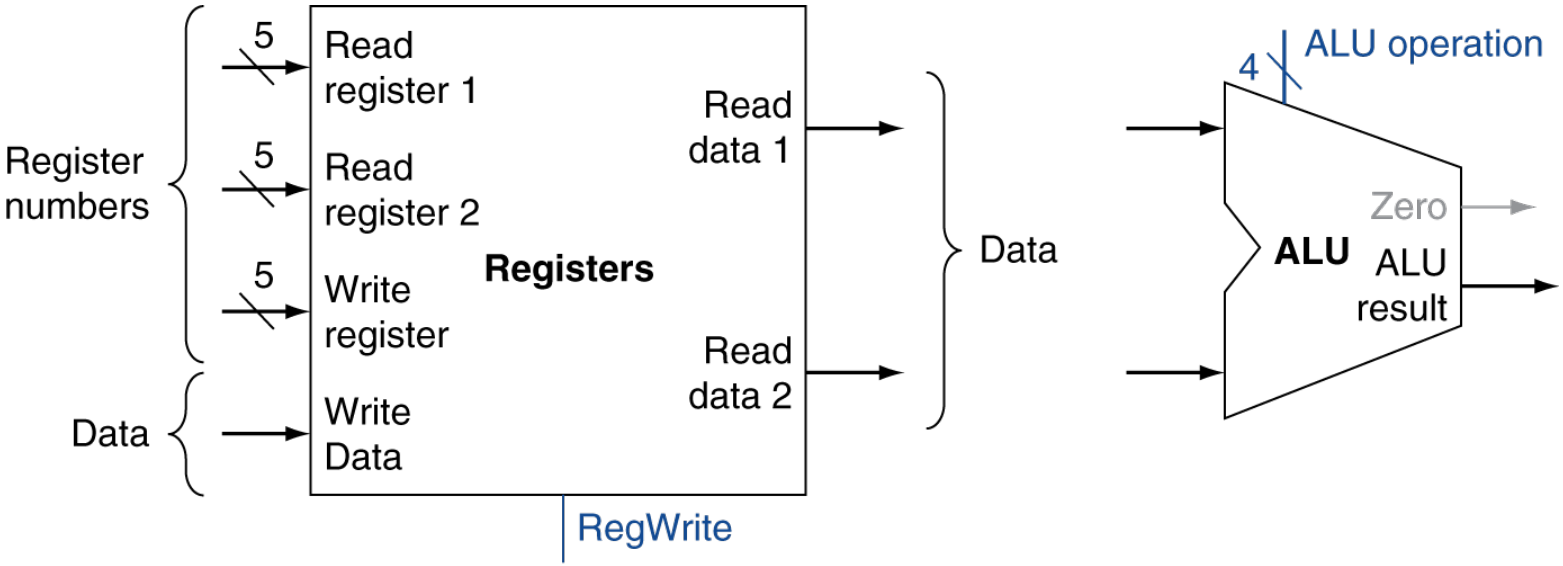


Datapath – R-Type Instructions

- **The R-type instructions (e.g., `add $t1, $t2, $t3`) perform arithmetic-logical operations**
  - Read two registers
  - Perform an ALU operation on the contents of the registers
  - Write the result to a register

- **R-type instructions thus require:**
  - **Register file** where the register's contents are stored
  - **ALU** to operate on the values read from the registers

R-Format Instructions



a. Registers

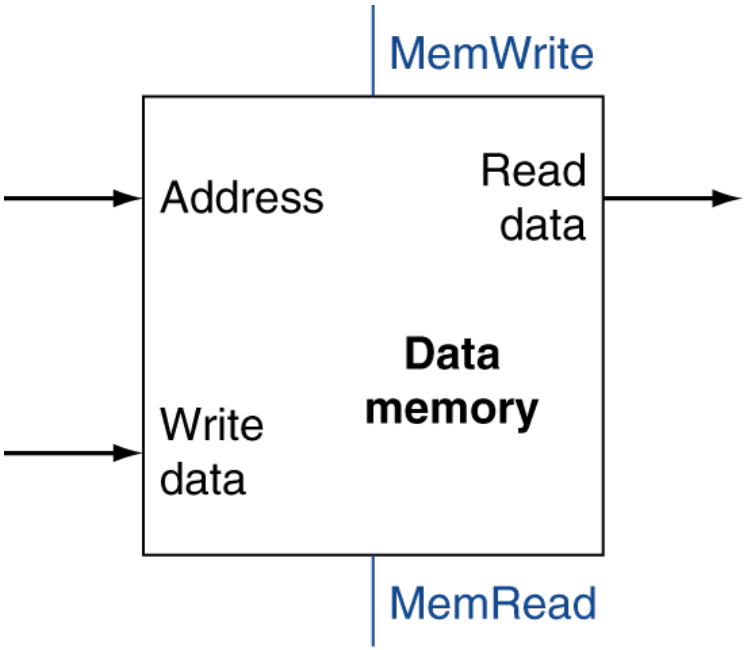
b. ALU

Datapath – Load and Store

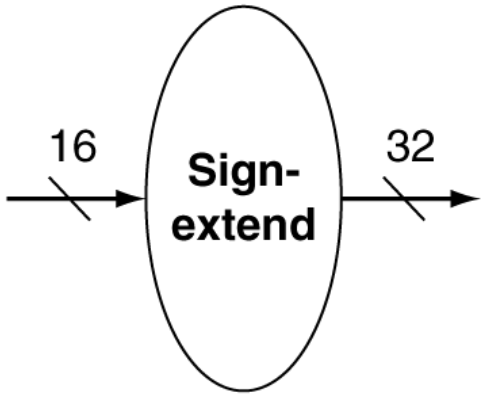
- **Memory reference instructions (e.g., `lw $t1, offset($t2)`)**
  - Compute a memory address
  - Perform a load or store operation, by adding a base register to a 16-bit signed offset field contained in the instruction

- **Memory reference instructions thus require:**
  - **Register file** where the register's contents are loaded/stored
  - **ALU** to operate on the value read from the base register and the offset
  - **Memory unit** where data is loaded/stored
  - **Sign extension unit** to sign-extend the 16-bit offset field to a 32-bit signed value

Load/Store Instructions



a. Data memory unit



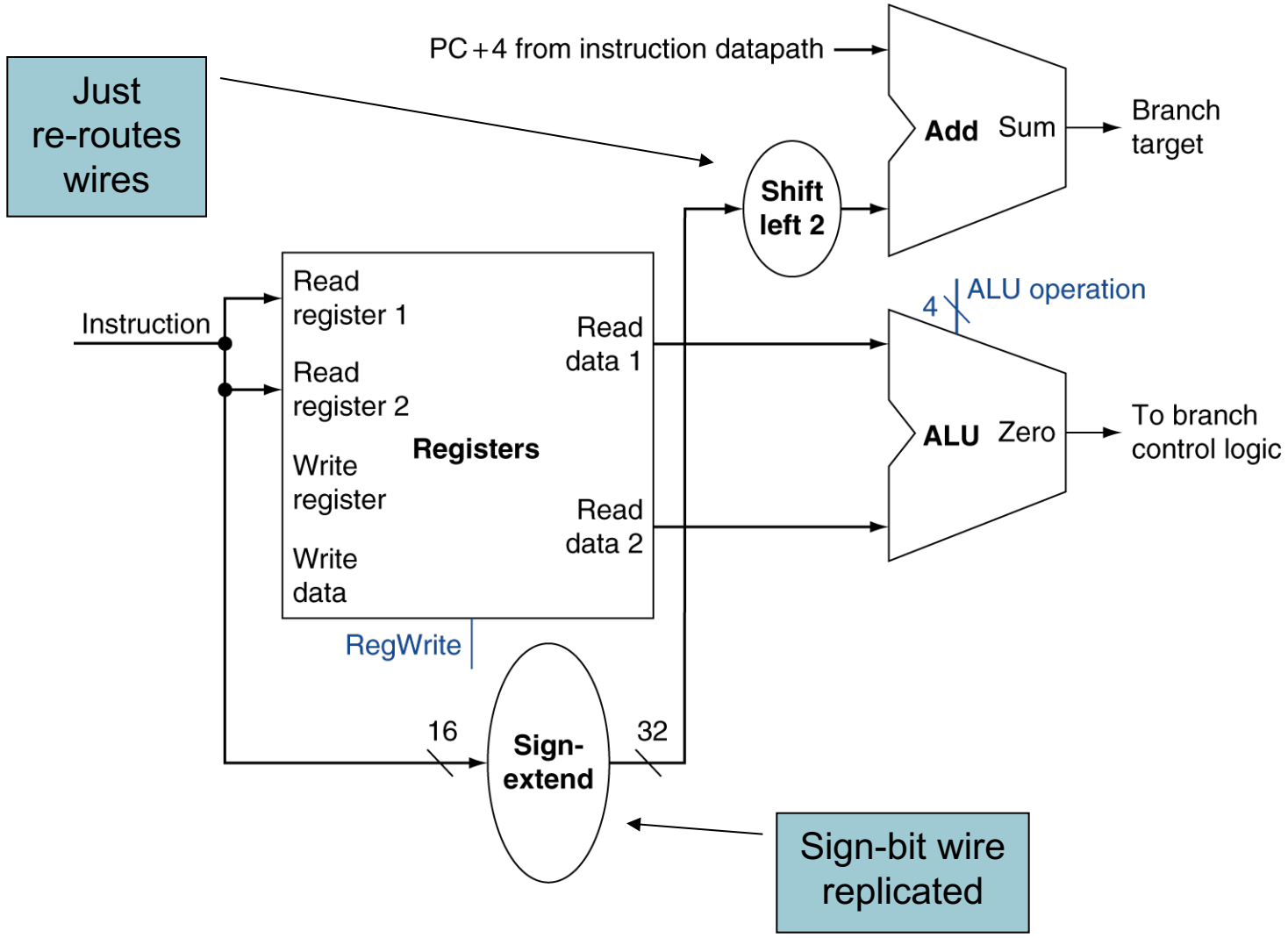
b. Sign extension unit

Datapath - Branches

- **Branch instructions (e.g., `beq $t1, $t2, offset`)**
  - Compare two registers
  - Compute a branch target address, by adding the PC+4 (the base for computing the branch target address) to a 16-bit signed offset field contained in the instruction
  - The offset is shifted left 2 bits so that it is a word offset (this shift increases the offset range by a factor of 4)

- **Branch instructions thus require:**
  - **Register file** where the register's contents are stored
  - **ALU** to compare the values read from the registers
  - **Adder** to compute the branch target address
  - **Sign extension** and **shift left units** to sign-extend and shift left the 16-bit offset field to a 32-bit signed value

Branch Instructions

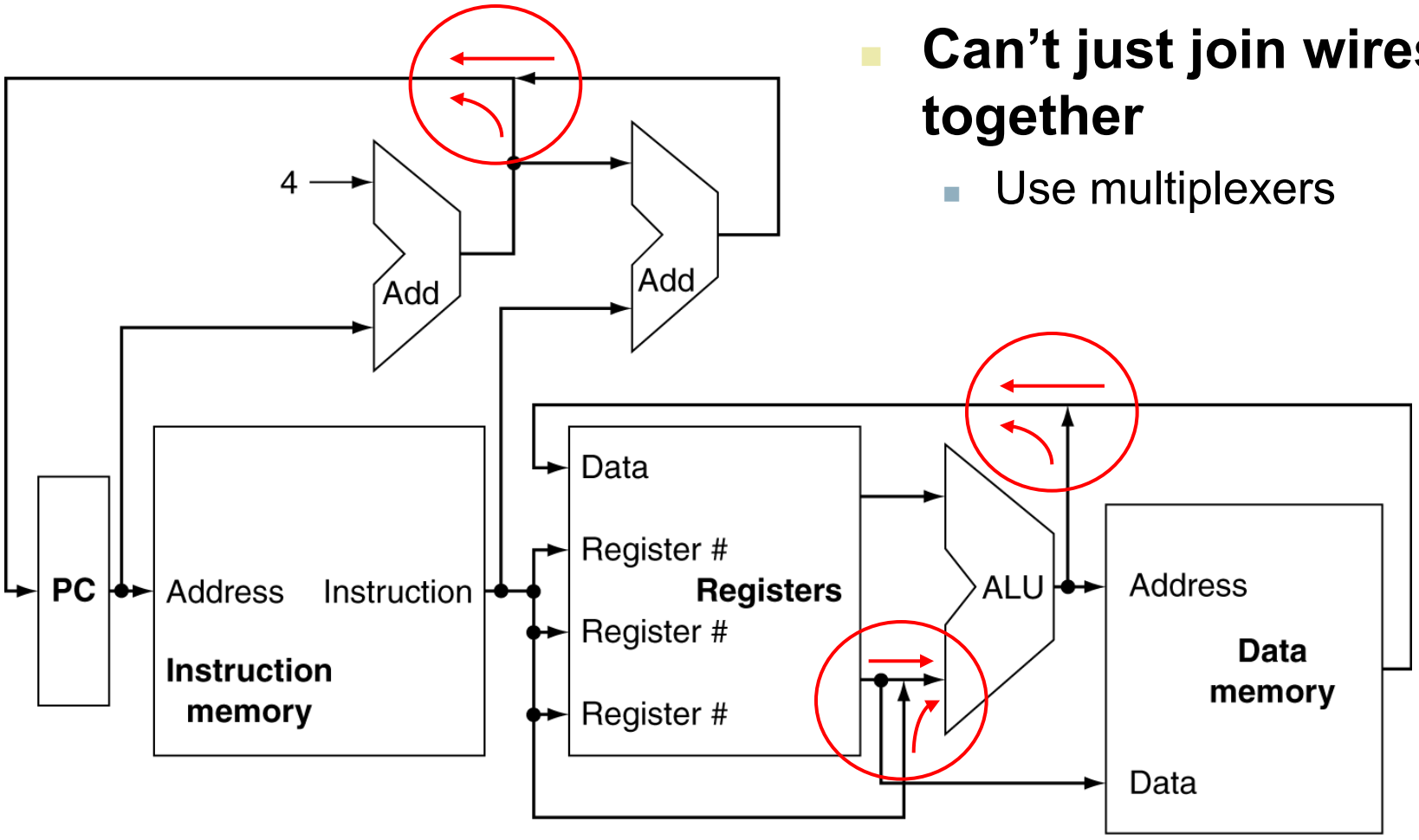


Combining the Elements

- **Our simple datapath executes an instruction in one clock cycle**
  - This means that no datapath resource can be used more than once per instruction, so any element needed more than once must be duplicated
  - Hence, we need **separate instruction and data memories**

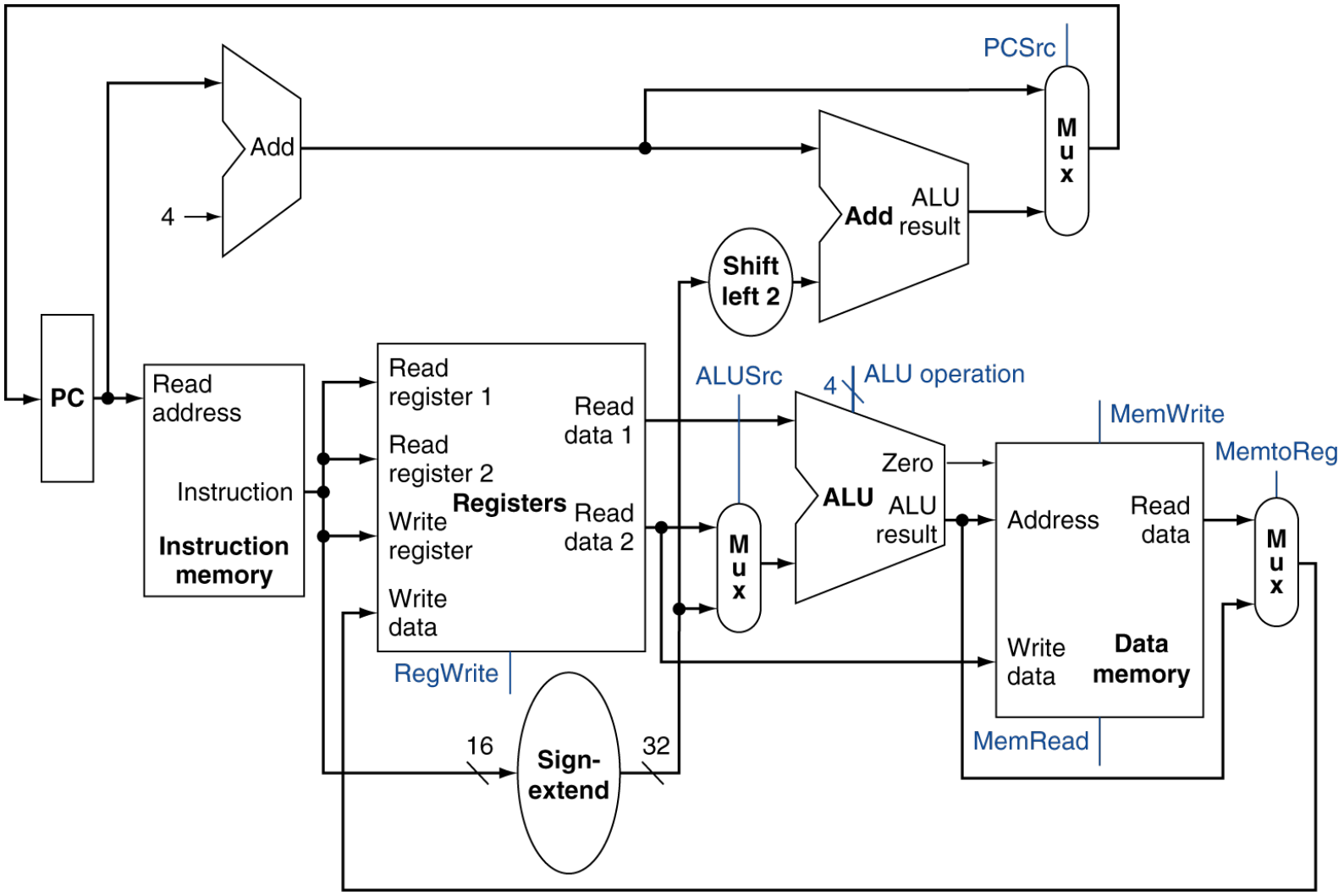
- **Although some of the functional units will need to be duplicated, many of the elements can be shared by different instruction flows**
  - Use **multiplexers** and **control signals** where alternate data sources are used for different instructions

Multiplexers



- **Can't just join wires together**
  - Use multiplexers

Datapath – Full Picture



Concluding remarks

- **This datapath can execute the basic instructions in a single clock cycle**
  - (load-store word, ALU operations, and branches)
  - The support for jumps will be added later

- **In the next lecture, we refine this view to fill in the details**
  - Which requires that we add further functional units, increase the number of connections between units, and, of course, a control unit to control what actions are taken for different instruction classes