Lecture 3
Processor: Datapath and Control

ALU

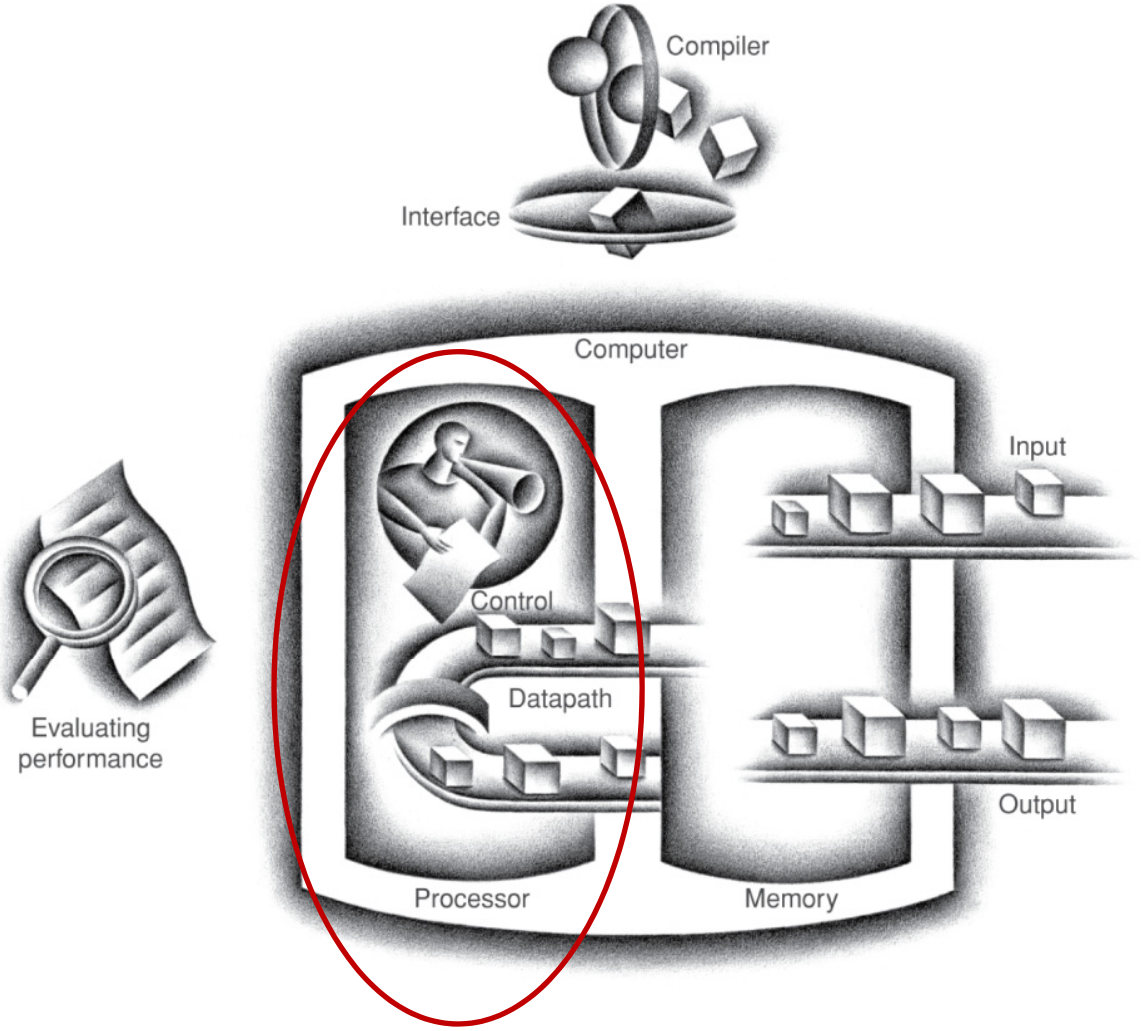
- Arithmetic Logic Unit is the hardware that performs addition, subtraction, AND, OR …

Recap: Performance

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

- CPU performance factors
  - Instruction count
    - Determined by Instruction Set Architecture and compiler
  - CPI and Cycle time
    - Determined by implementation of the processor

Components of a Computer

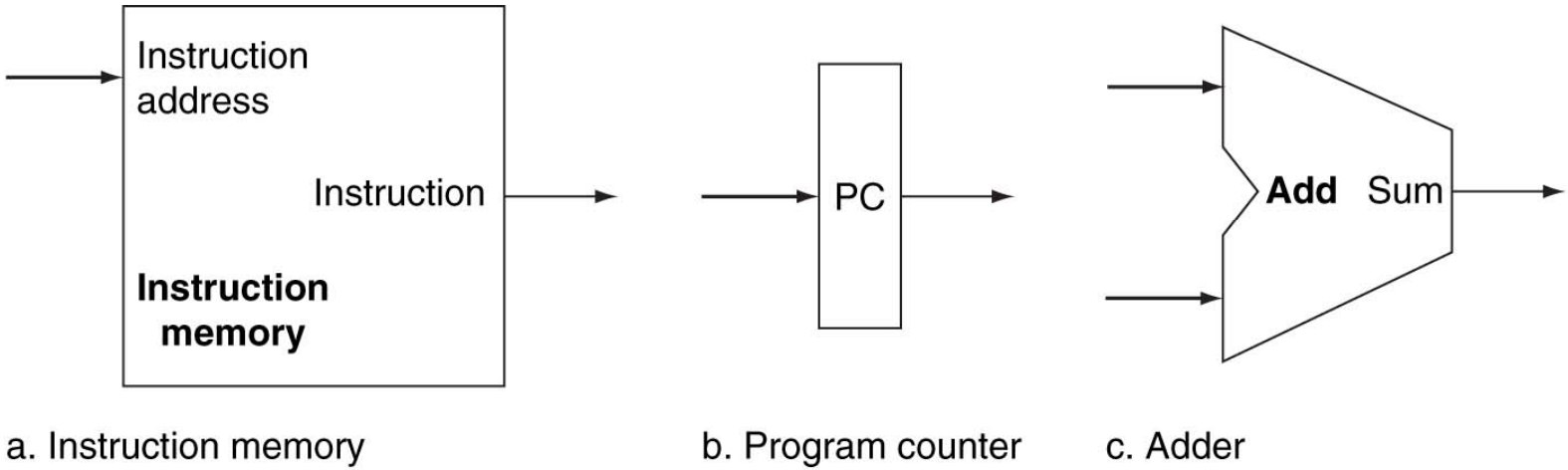


Processor

- Datapath
  - Components of the processor that perform arithmetic operations and holds data
- Control
  - Component of the processor that commands the datapath, memory, I/O devices according to the instructions of the memory

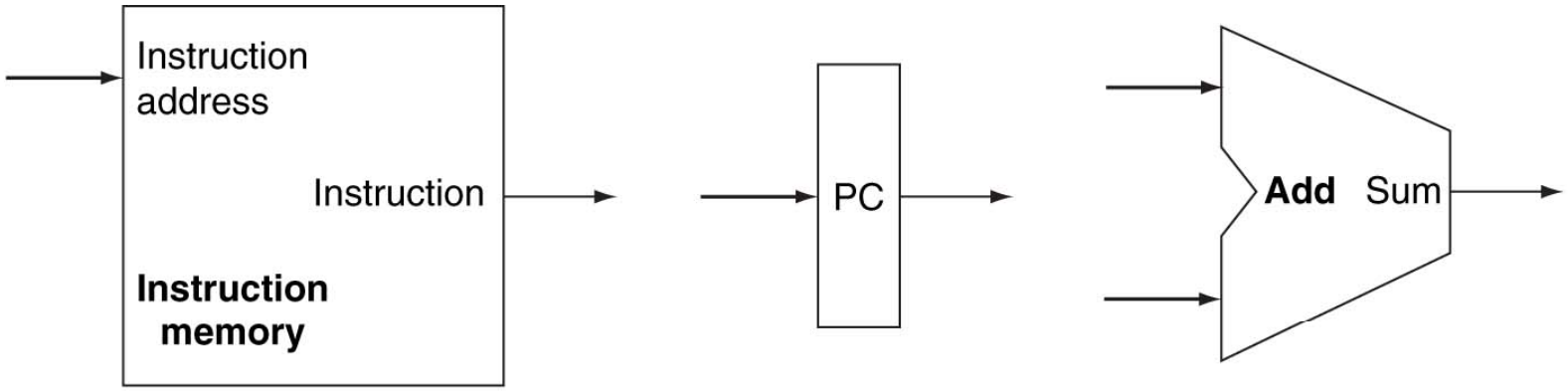
Building a Datapath

- Datapath
  - Elements that process data and addresses in the CPU
    - Memories, registers, ALUs, …
- We will build a MIPS datapath incrementally
  - considering only a subset of instructions
- To start, we will look at 3 elements



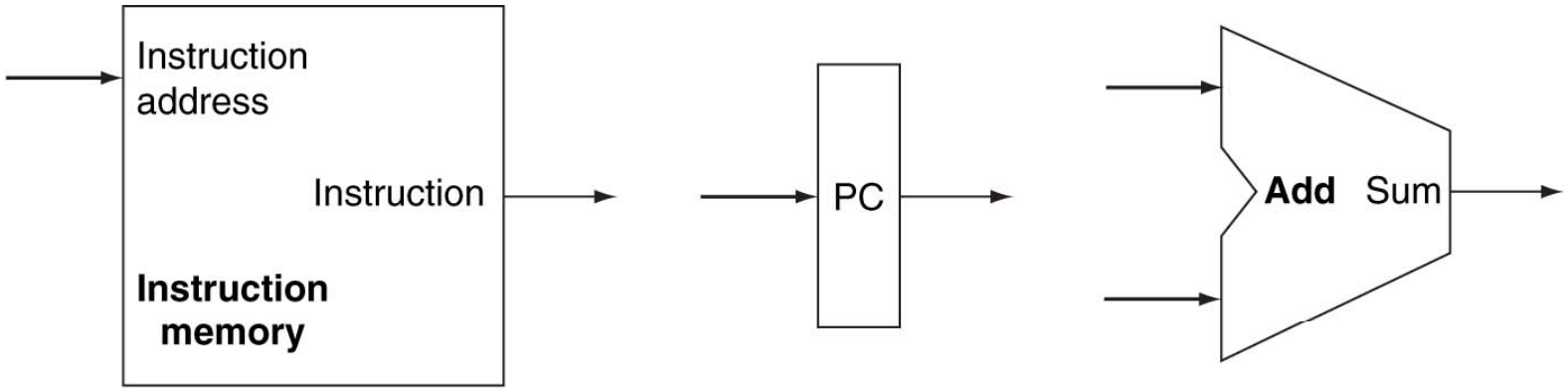
a. Instruction memory b. Program counter c. Adder

- A memory unit to store instructions of a program and supply instructions given an address
- Needs to provide only read access (once the program is loaded).
  - No control signal is need.



a. Instruction memory b. Program counter c. Adder

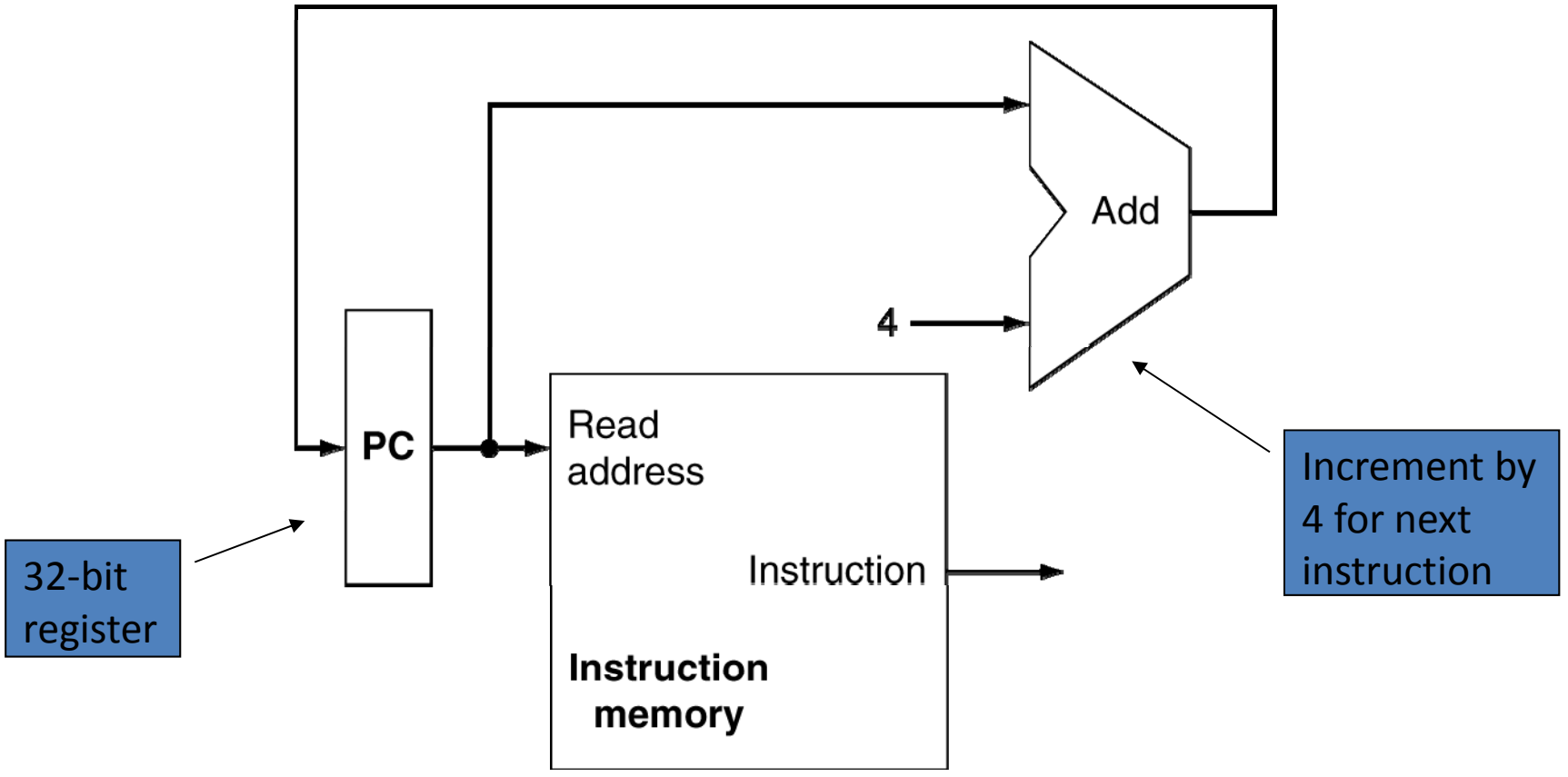
- PC (Program Counter or Instruction address register) is a register that holds the address of the current instruction
- A new value is written to it every clock cycle. No control signal is required to enable write



a. Instruction memory b. Program counter c. Adder

- Adder to increment the PC to the address of the next instruction
- An ALU permanently wired to do only addition. No extra control signal required

Datapath portion for Instruction Fetch



Types of Elements in the Datapath

- State element:
  - A memory element, i.e., it contains a state
  - E.g., program counter, instruction memory

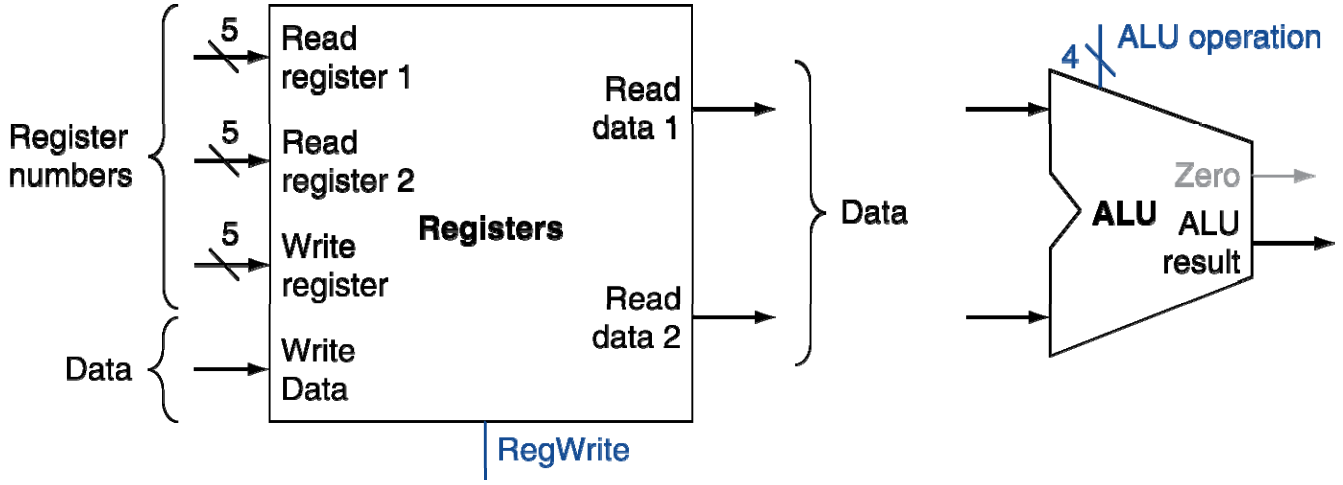
- Combinational element:
  - Elements that operate on values
  - E.g. adder, ALU

- Now, we will look at datapath elements required by the different classes of instructions
  - Arithmetic and logical instructions
  - Data transfer instructions
  - Branch instructions

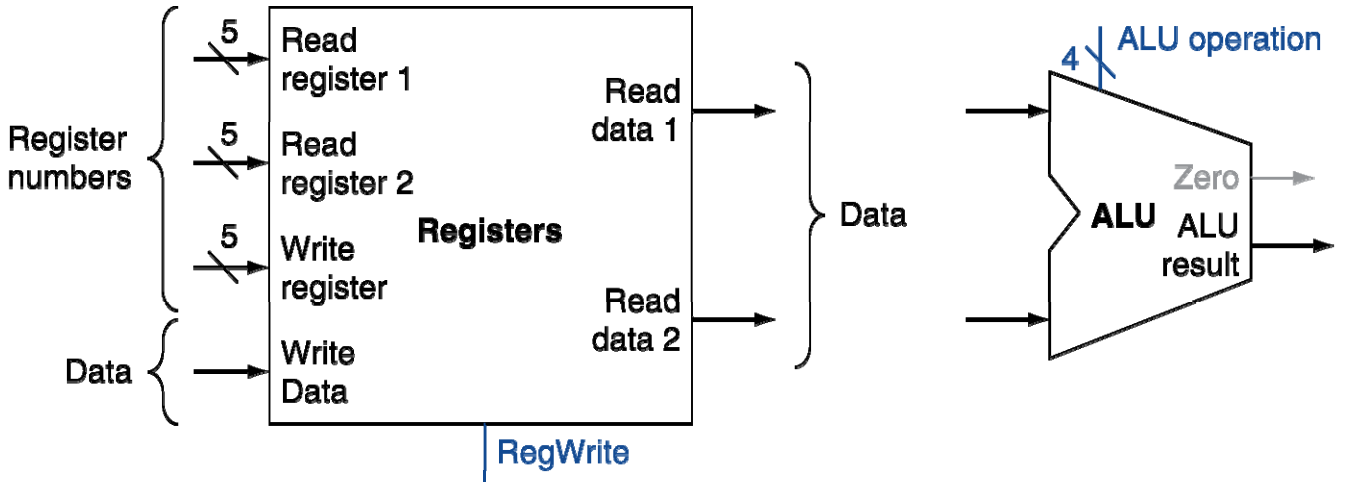
R-Format ALU Instructions

- E.g., ***add $t1, $t2, $t3***
- Perform arithmetic/logical operation
- Read two register operands and write register result

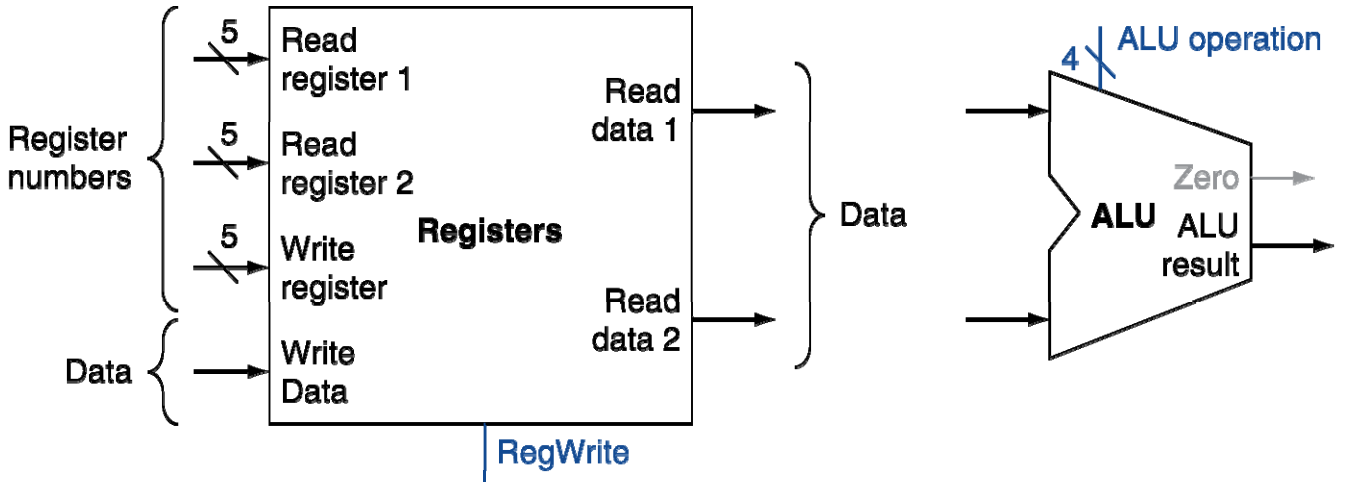
R-Format ALU Instructions



- Register file: A collection of the registers
  - Any register can be read or written by specifying the number of the register
  - Contains the register state of the computer

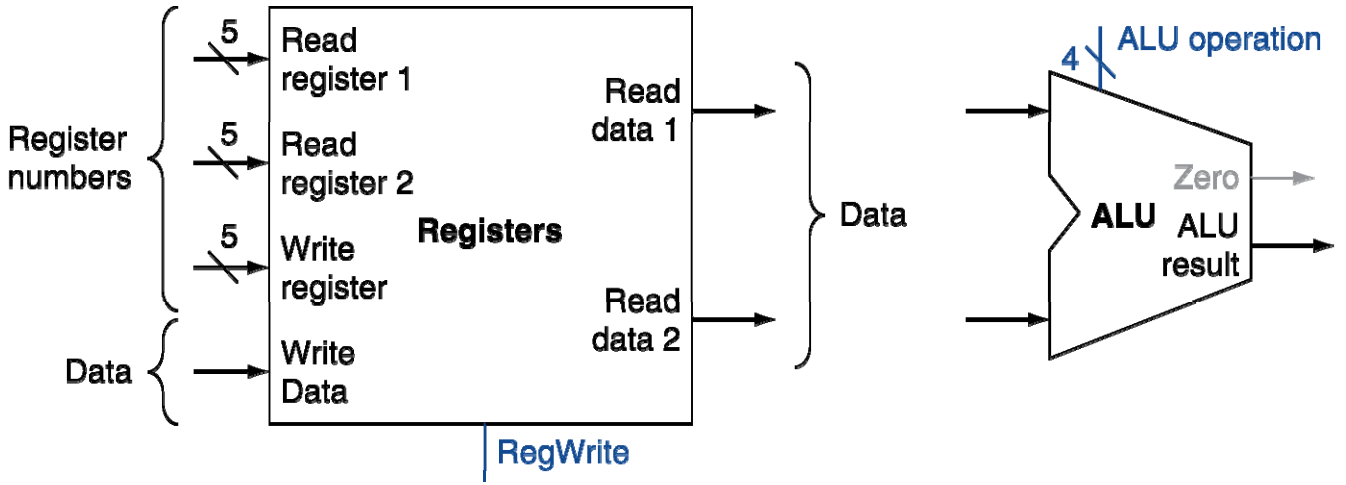


- Read from register file
  - 2 inputs to the register file specifying the numbers
    - 5 bit wide inputs for the 32 registers
  - 2 outputs from the register file with the read values
    - 32 bit wide
  - For all instructions. No control required.



a. Registers b. ALU

- Write to register file
  - 1 input to the register file specifying the number
    - 5 bit wide inputs for the 32 registers
  - 1 input to the register file with the value to be written
    - 32 bit wide
  - Only for some instructions. RegWrite control signal.



a. Registers

b. ALU

- ALU
  - Takes two 32 bit input and produces a 32 bit output
  - Also, sets one-bit signal if the results is 0
  - The operation done by ALU is controlled by a 4 bit control signal input. This is set according to the instruction

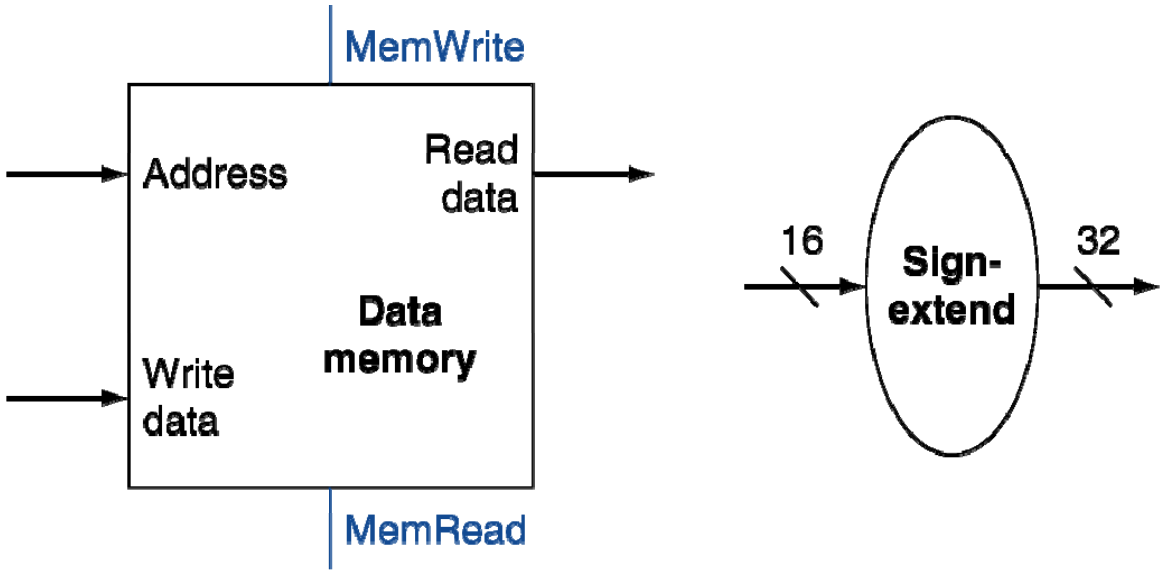
Data transfer instructions

- lw $t1, offset_value($t2)
- Load: Read memory and update register

- sw $t1, offset_value($t2)
- Store: Write register value to memory

Data transfer instructions

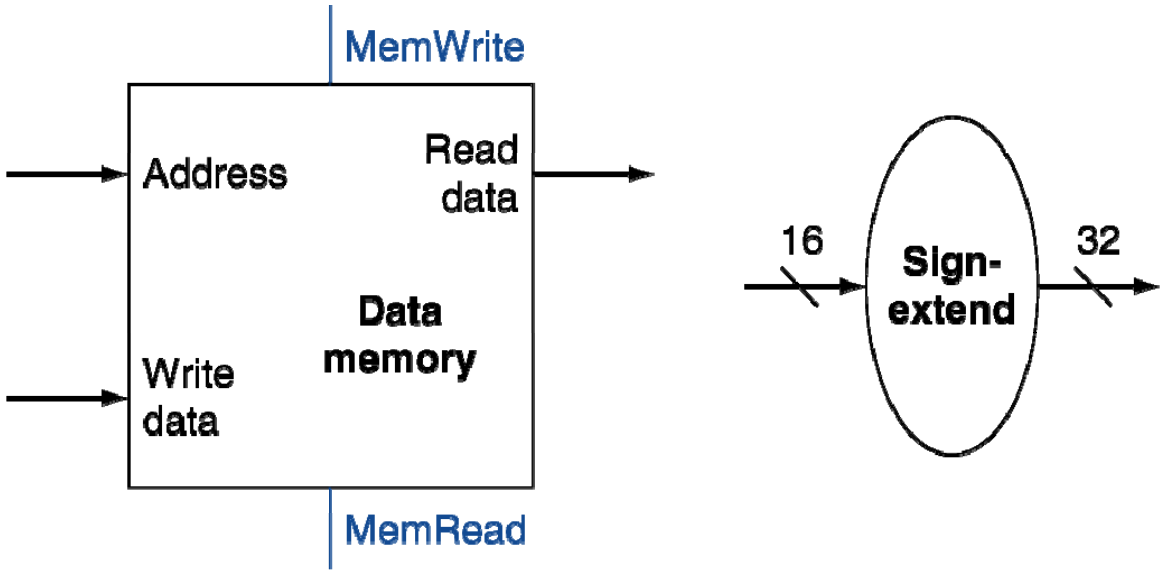
- Compute the memory address by adding the value in base register and the 16 bit offset
  - need the ALU
  - Calculate address using 16-bit offset
    - Use ALU, but sign-extend offset
- Write to or read from register
  - need the register file



a. Data memory unit

b. Sign extension unit

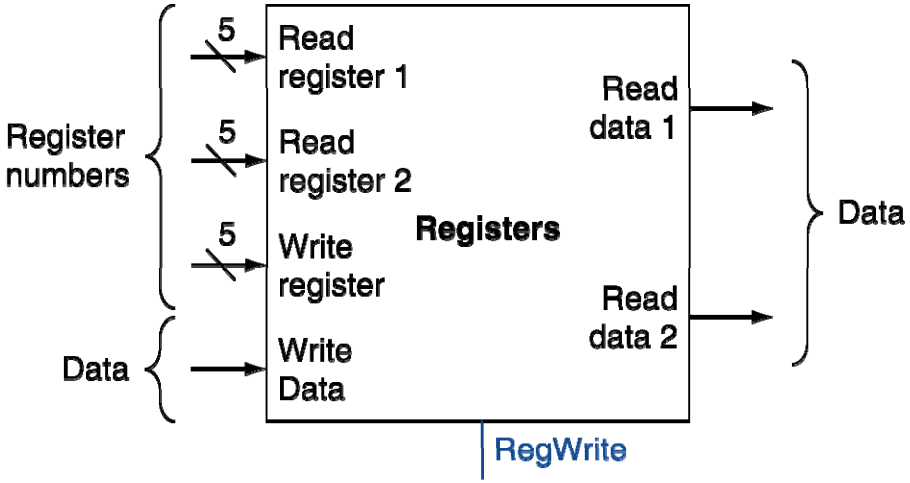
- Two additional units – data memory and sign unit extension
- Data memory
  - State element with
    - input for address and data to be written
    - output for read result



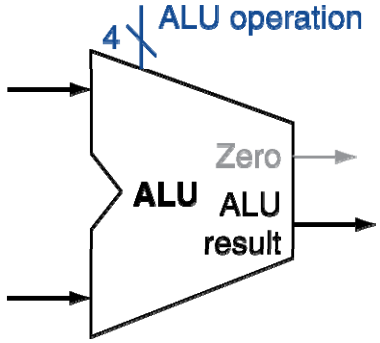
a. Data memory unit

b. Sign extension unit

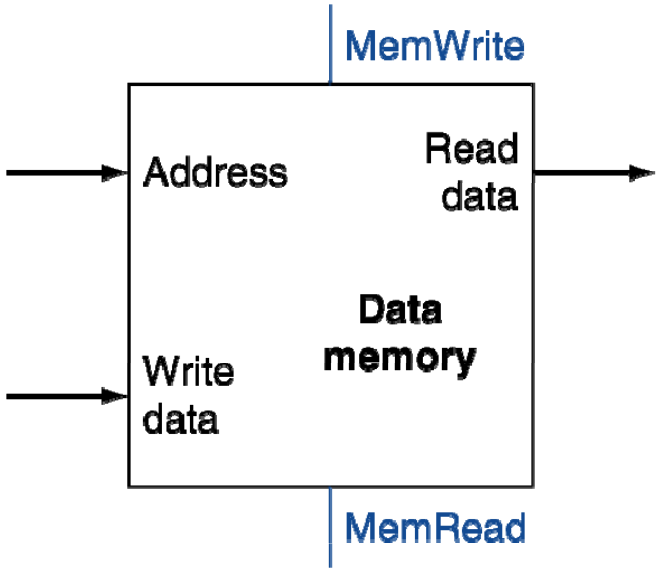
- Data memory
  - Separate control for read and write
  - Control for read is required because reading from invalid address can lead to problems
- Sign-extension unit takes a 16 bit input and extend it to a 32 bit output



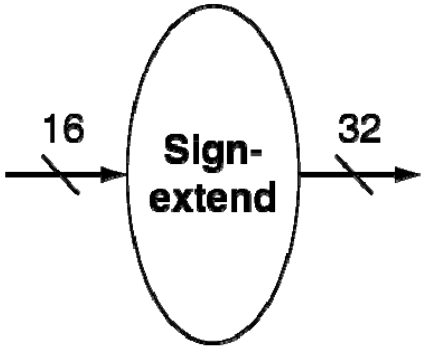
a. Registers



b. ALU



a. Data memory unit



b. Sign extension unit

Composing the Elements for R-type and data transfer instructions

- A simple data path that does an instruction in one clock cycle
  - Each datapath element can only do one function at a time
  - Hence, we need separate instruction and data memories
- Use multiplexers where alternate data sources are used for different instructions

Multiplexors

- An ALU might need input from
  - Two registers
  - Or one registers and one immediate field (or offset)

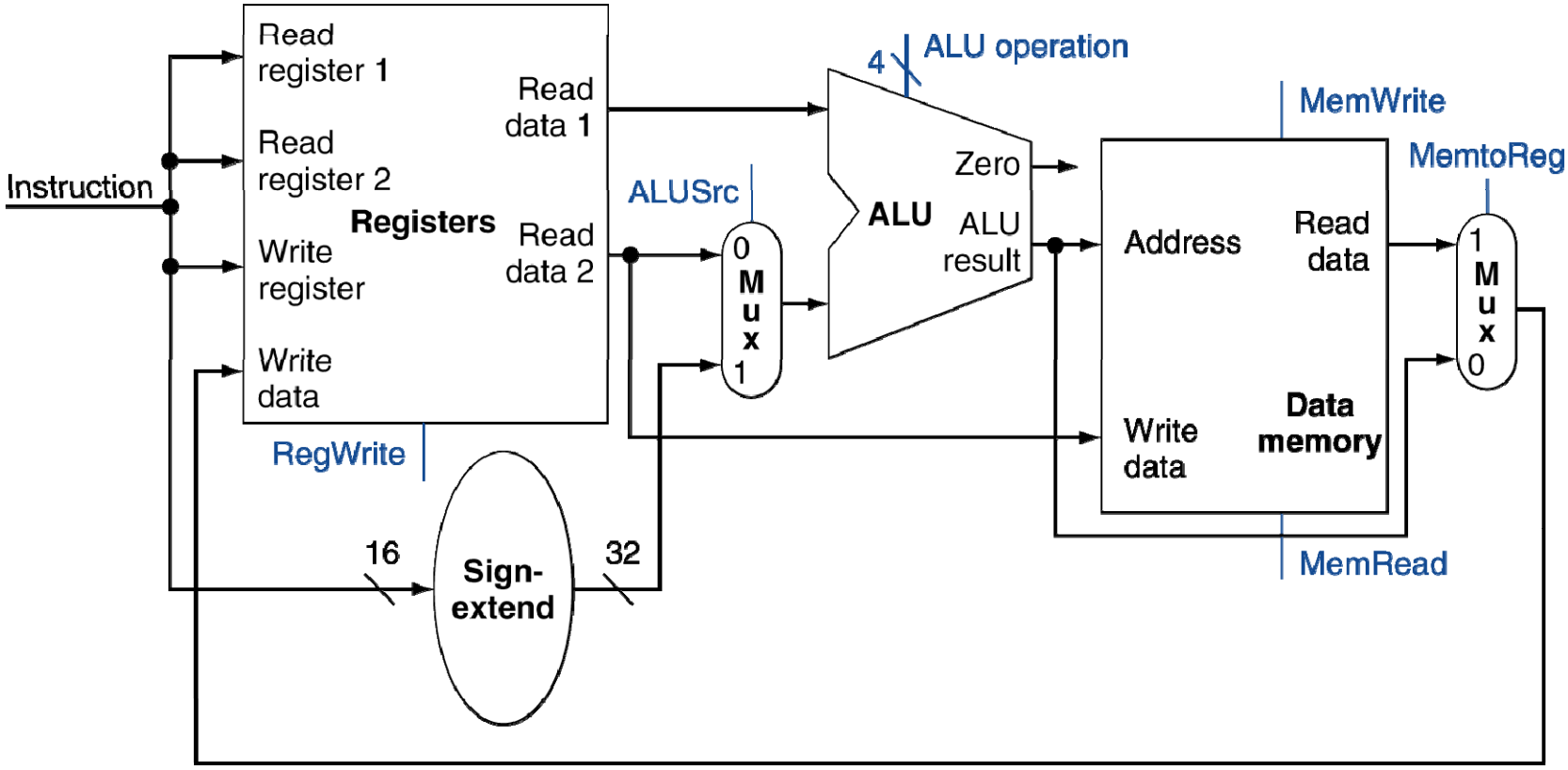
- To choose correctly from multiple sources, a hardware element called multiplexor is used with appropriate control signals

Multiplexors

- The data written to registers may come from
  - Data memory
  - Or ALU

- To choose correctly from multiple sources, a hardware element called multiplexor is used with appropriate control signals

R-Type/Load/Store Datapath



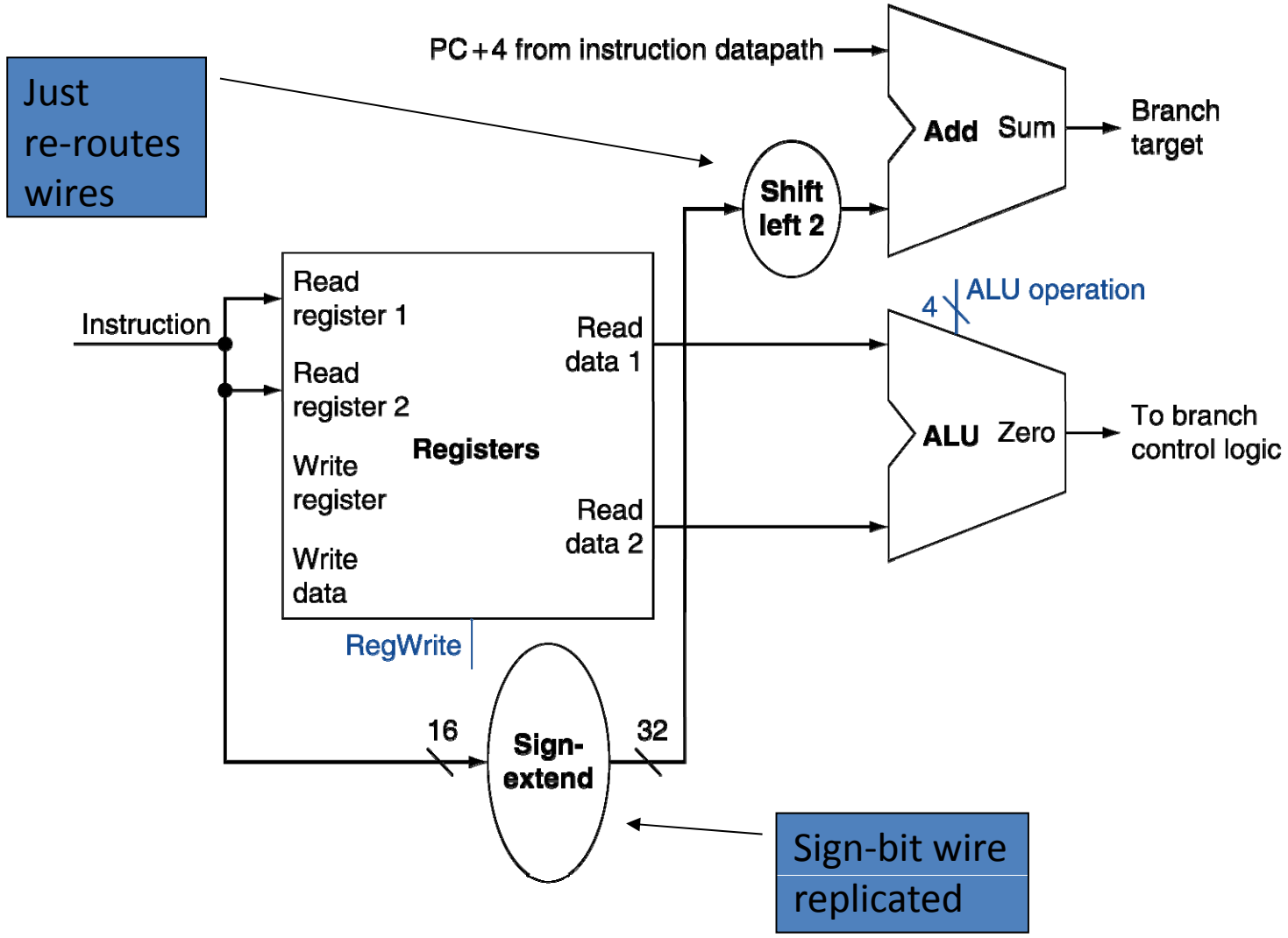
Branch Instructions

- beq $t1, $t2, offset
- Read two registers and compare them
- Take the 16 bit offset and add it to the address of next instruction following the branch instruction to obtain the branch target address

Branch Instructions

- Read register operands
- Compare operands
  - Use ALU, subtract and check Zero output
- Calculate target address
  - Sign-extend the offset
  - Shift left 2 places (word displacement)
  - Add to PC + 4
    - Already calculated by instruction fetch

Branch Instructions

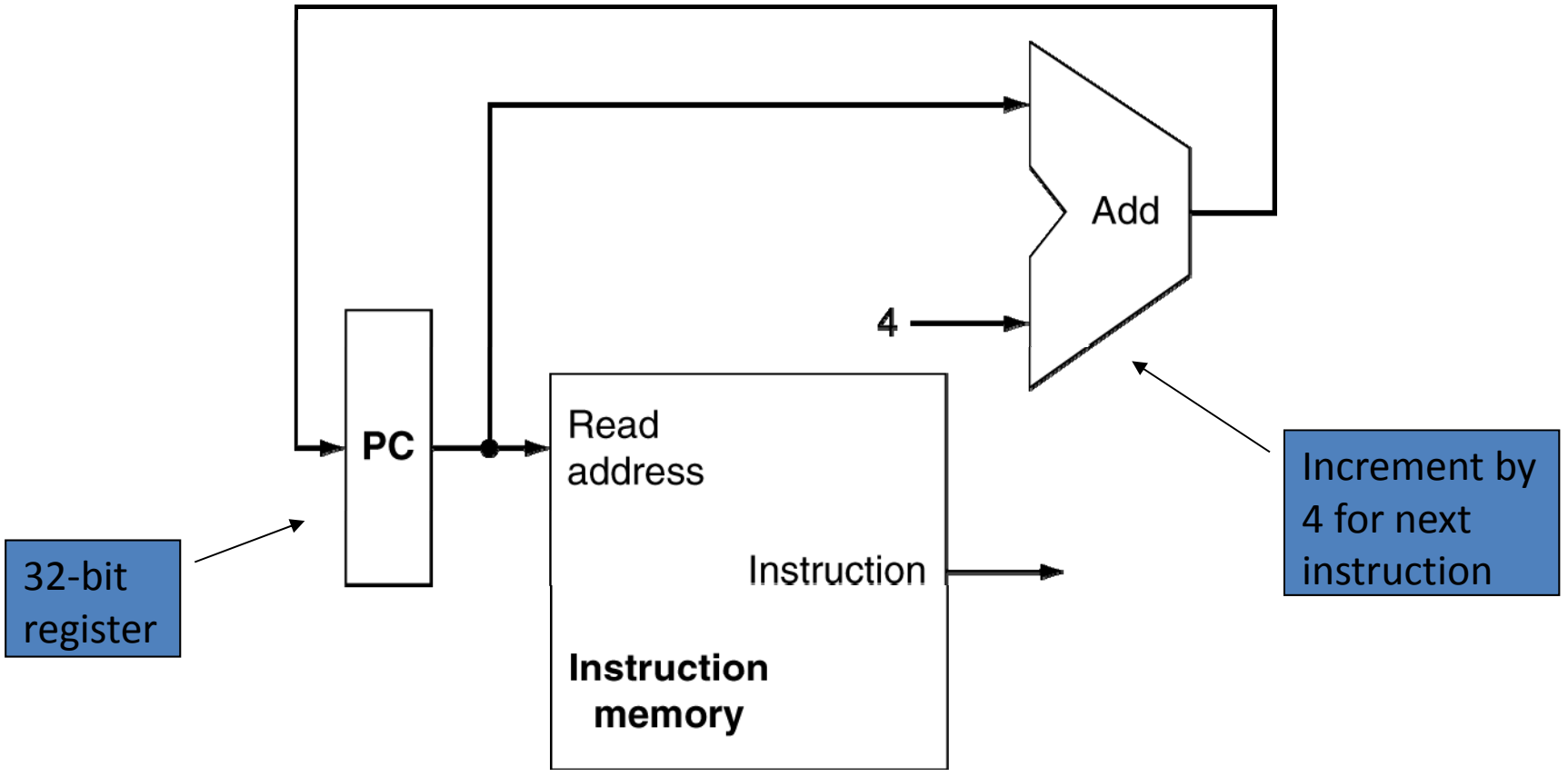


Composing all elements together

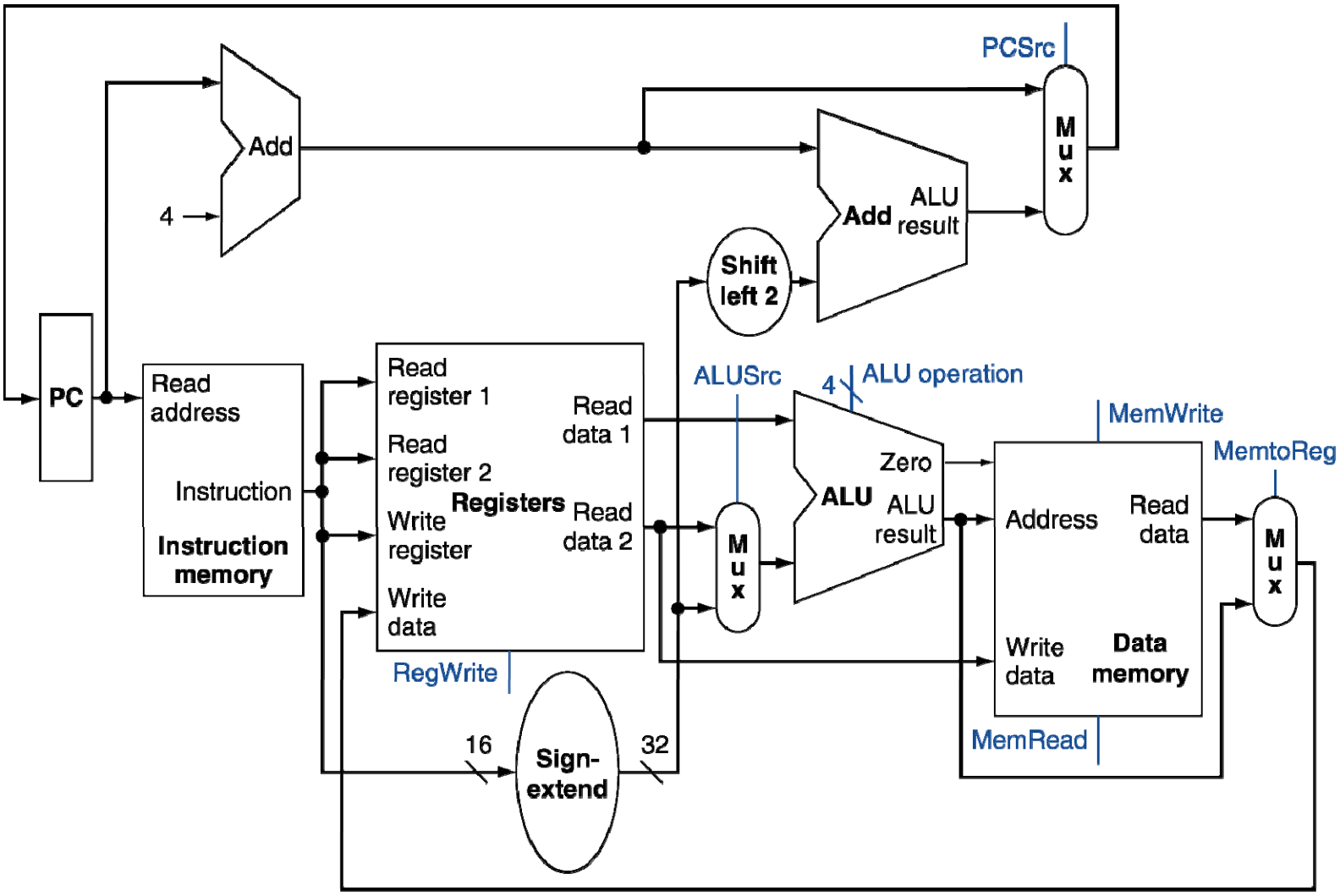
- Instruction fetch datapath
- Datapath for R-type and memory instructions
- Datapath for branches

- Need an additional multiplexor to select the sequential address after branch or the branch target address to be written to the PC

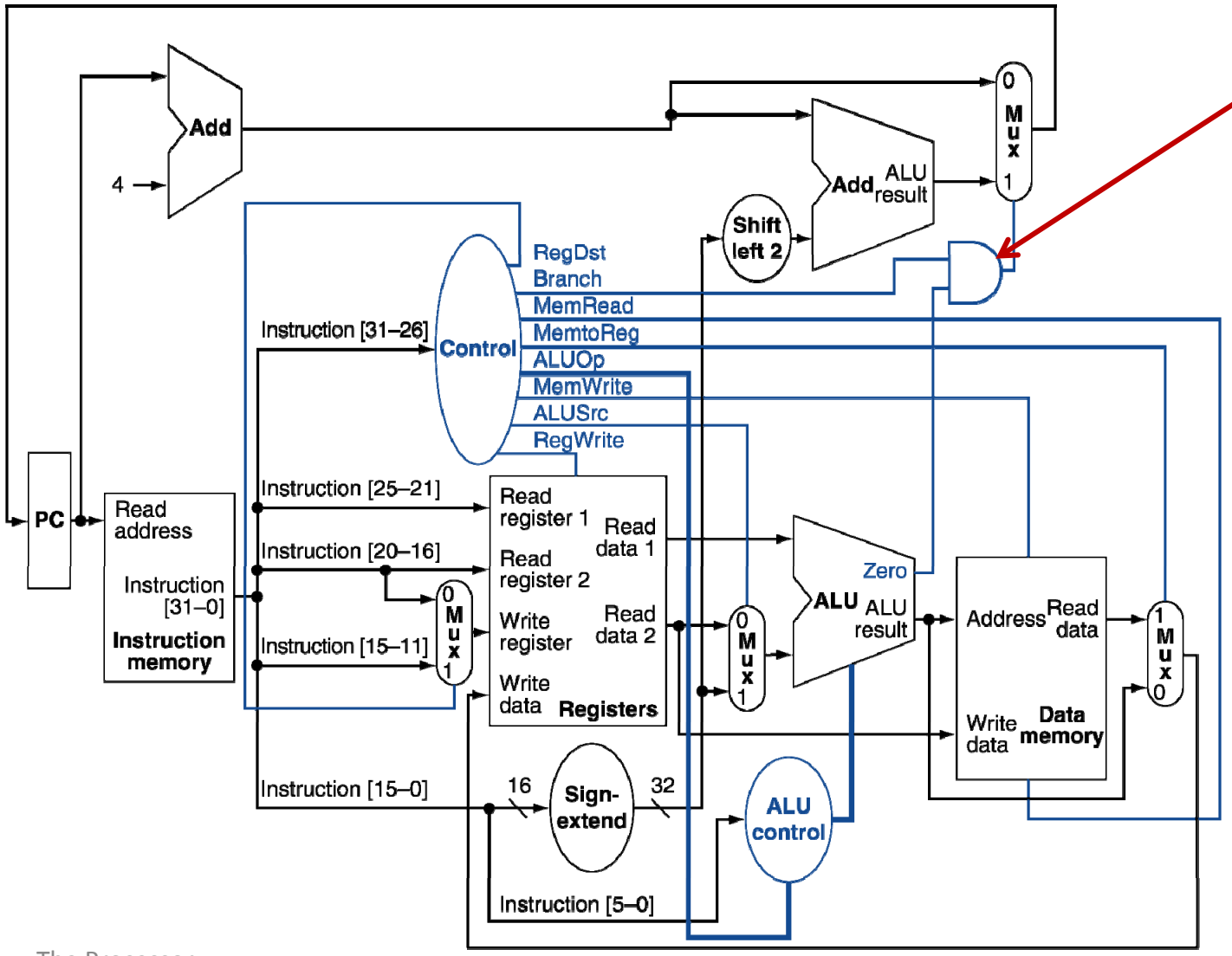
Datapath portion for Instruction Fetch



Full Datapath

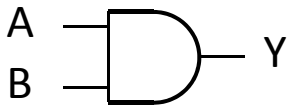


Datapath With Control

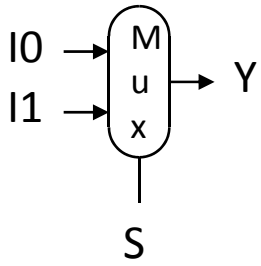


A Recap: Combinational Elements

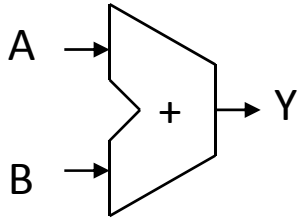
- AND-gate
  - Y = A & B



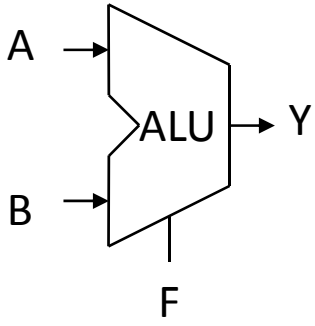
- Multiplexer
  - Y = S ? I1 : I0



- Adder
  - Y = A + B



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



A Recap: State Elements

- Registers
- Data Memory
- Instruction Memory

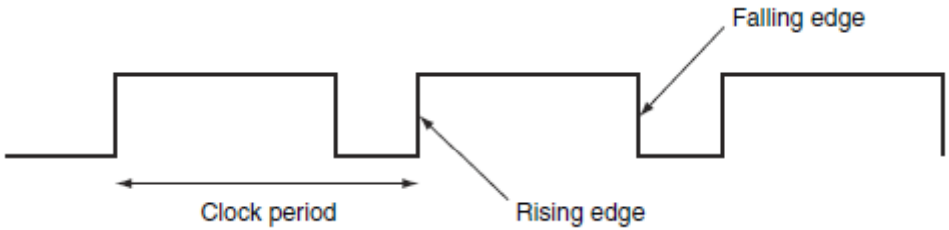
- Clocks are needed to decide when an element that contains state should be updated

Recap from Lecture 1: CPU Clocking

- Operation of digital hardware governed by a constant-rate clock

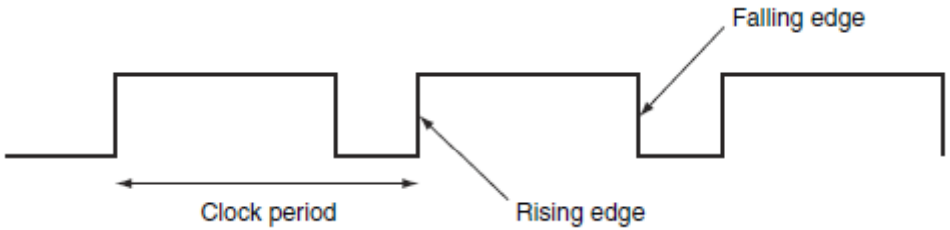
- Clock period: duration of a clock cycle
- Clock frequency (rate): cycles per second

Clocks



- A clock is a signal with a fixed cycle time (period)
- The clock frequency is the inverse of the cycle time

Clocks



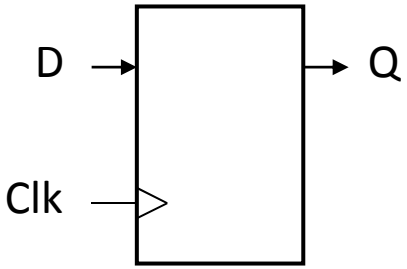
- The clock cycle time or clock period is divided into two portions:
  - when the clock is high
  - when the clock is low

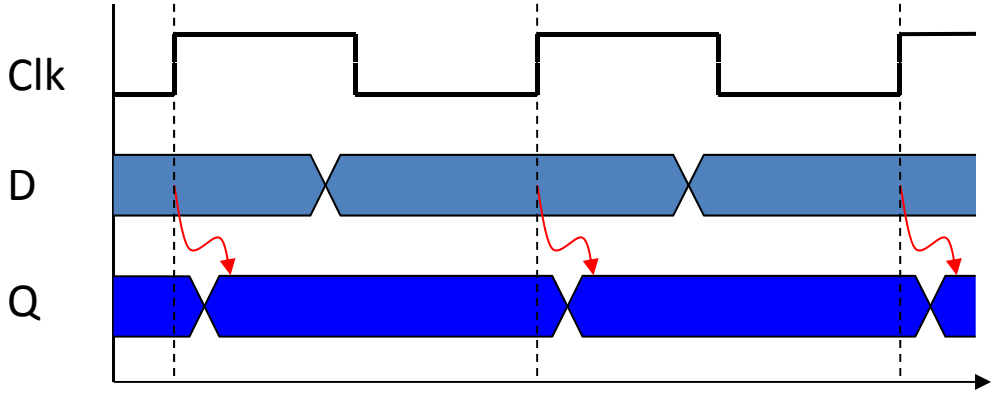
Clocking Methodology

- We study
  - Edge triggered methodology
    - Because it is simple
- Edge triggered methodology:
  - All state changes occur on a clock edge

Clocking Methodology : State Elements

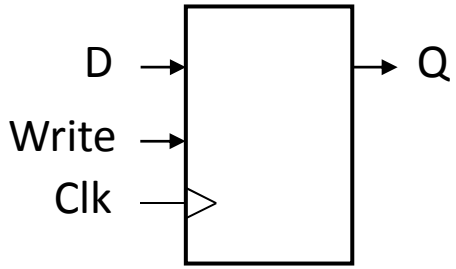
- Register: stores data in a circuit
  - Uses a clock signal to determine when to update the stored value
  - Edge-triggered: update when Clk changes from 0 to 1

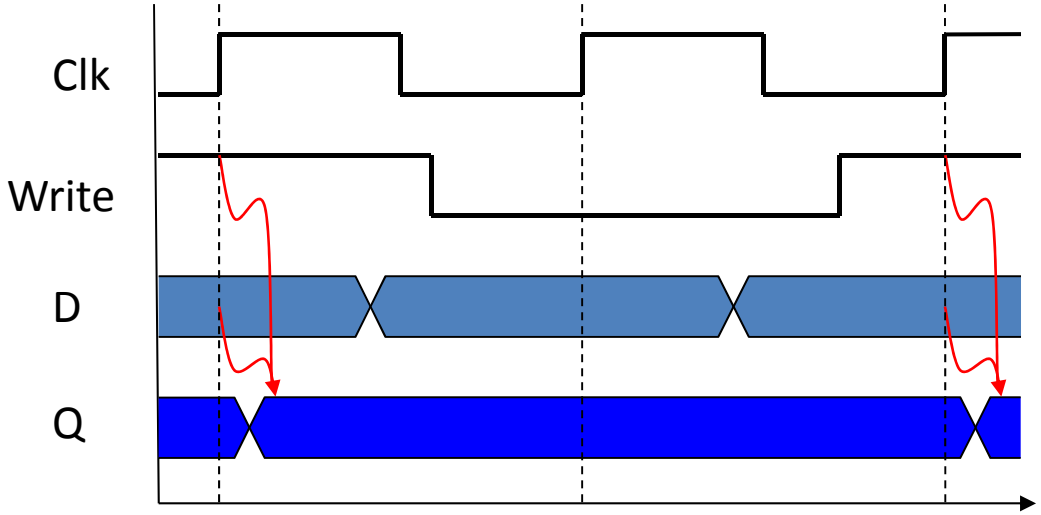




Clocking Methodology : State Elements

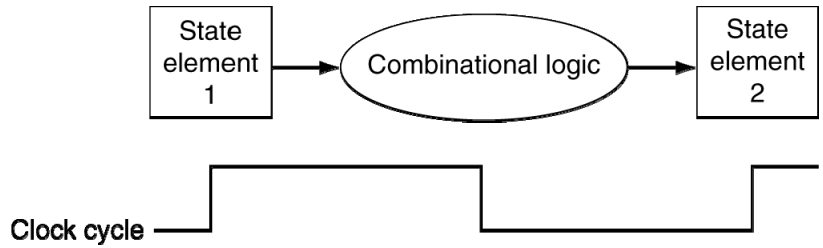
- Register with write control
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later





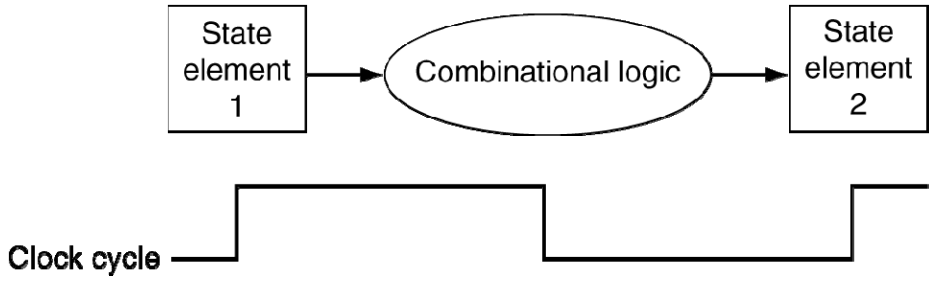
Clocking Methodology

- Combinational logic transforms data during clock cycles
  - Between clock edges
  - Input from state elements, output to state element
    - The state elements, whose outputs change only after the clock edge, provide valid inputs to the combinational logic block.



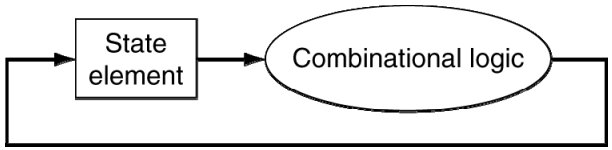
Clocking Methodology

- To ensure that the values written into the state elements on the active clock edge are valid, the clock must have a long enough period so that all the signals in the combinational logic block stabilize, then the clock edge samples those values for storage in the state elements.
- 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.
- Longest delay determines clock period



It is possible to have a state element that is used as both an input and output to the same combinational logic block

Ensure that the clock period is long enough



Single Clock Cycle

- We studied a simple implementation where a single clock cycle is required for every instruction. Every instruction begins on one clock edge and completes execution on the next

Performance Issues

- Longest delay determines clock period
  - Critical path: load instruction
  - Instruction memory → register file → ALU → data memory → register file
- Not feasible to vary period for different instructions
- The clock cycle must be extended to accommodate the longest instruction
- Improve performance by pipelining

Conclusion

- ISA influences the design of datapath and control for a processor

- We studied an implementation based on single cycle