Lecture 16: Basic CPU Design

- Today’s topics:
  - Single-cycle CPU
  - Multi-cycle CPU
- Reminder:
  - Assignment 6 will be posted today – due in a week

Basic MIPS Architecture

- Now that we understand clocks and storage of states, we'll design a simple CPU that executes:
  - basic math (add, sub, and, or, slt)
  - memory access (lw and sw)
  - branch and jump instructions (beq and j)

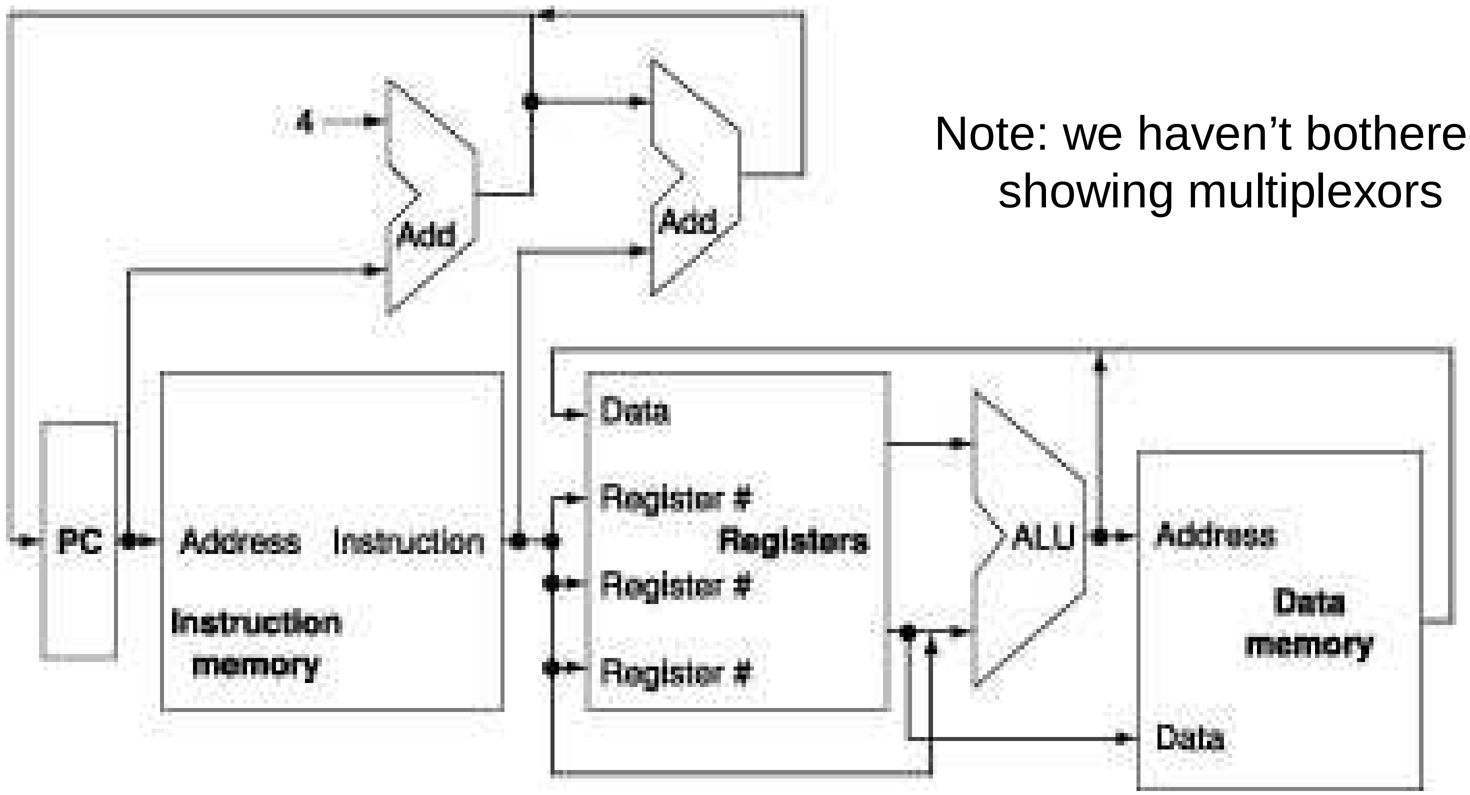
Implementation Overview

- We need memory
  - to store instructions
  - to store data
  - for now, let's make them separate units

- We need registers, ALU, and a whole lot of control logic

- CPU operations common to all instructions:
  - use the program counter (PC) to pull instruction out of instruction memory
  - read register values

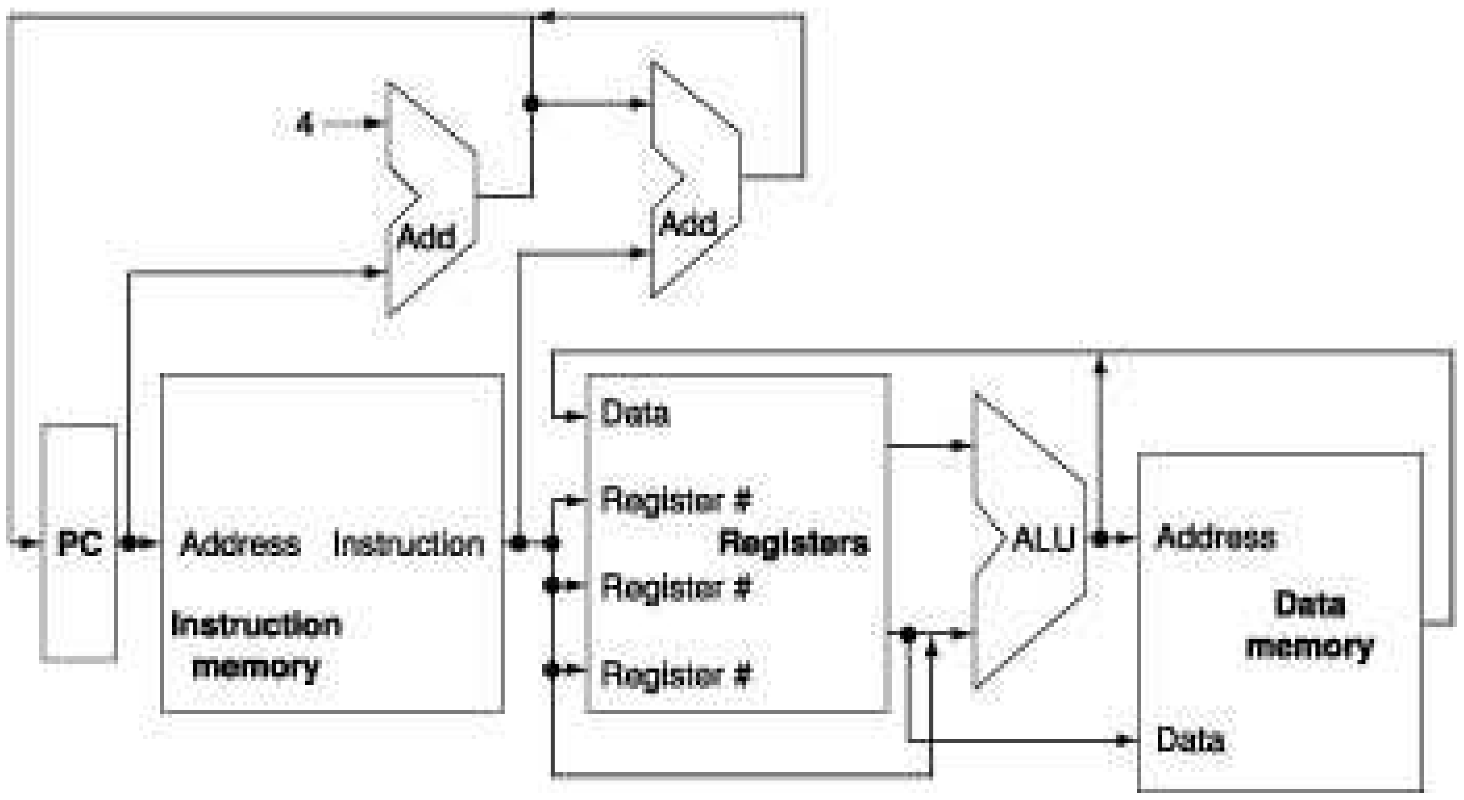
View from 30,000 Feet



Note: we haven’t bothered showing multiplexors

- What is the role of the Add units?
- Explain the inputs to the data memory unit
- Explain the inputs to the ALU
- Explain the inputs to the register unit

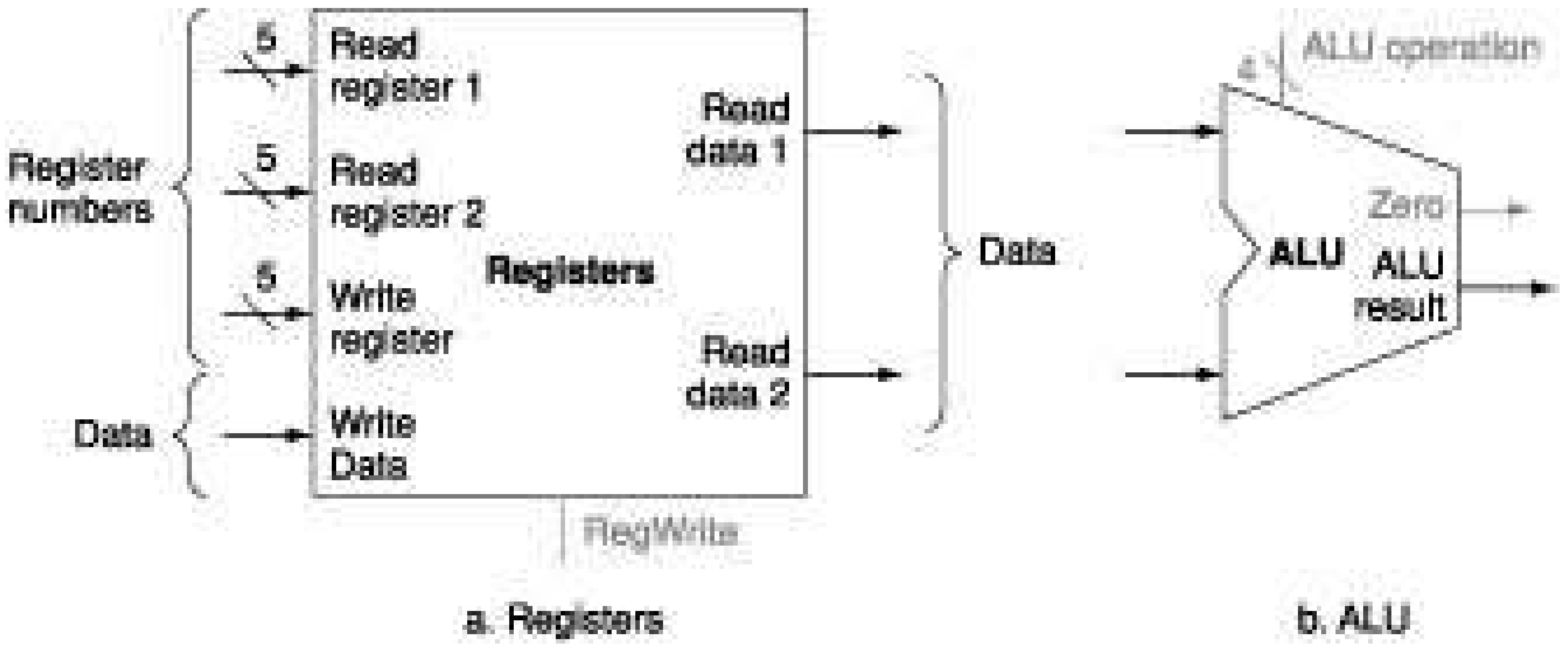
Clocking Methodology



- Which of the above units need a clock?
- What is being saved (latched) on the rising edge of the clock? Keep in mind that the latched value remains there for an entire cycle

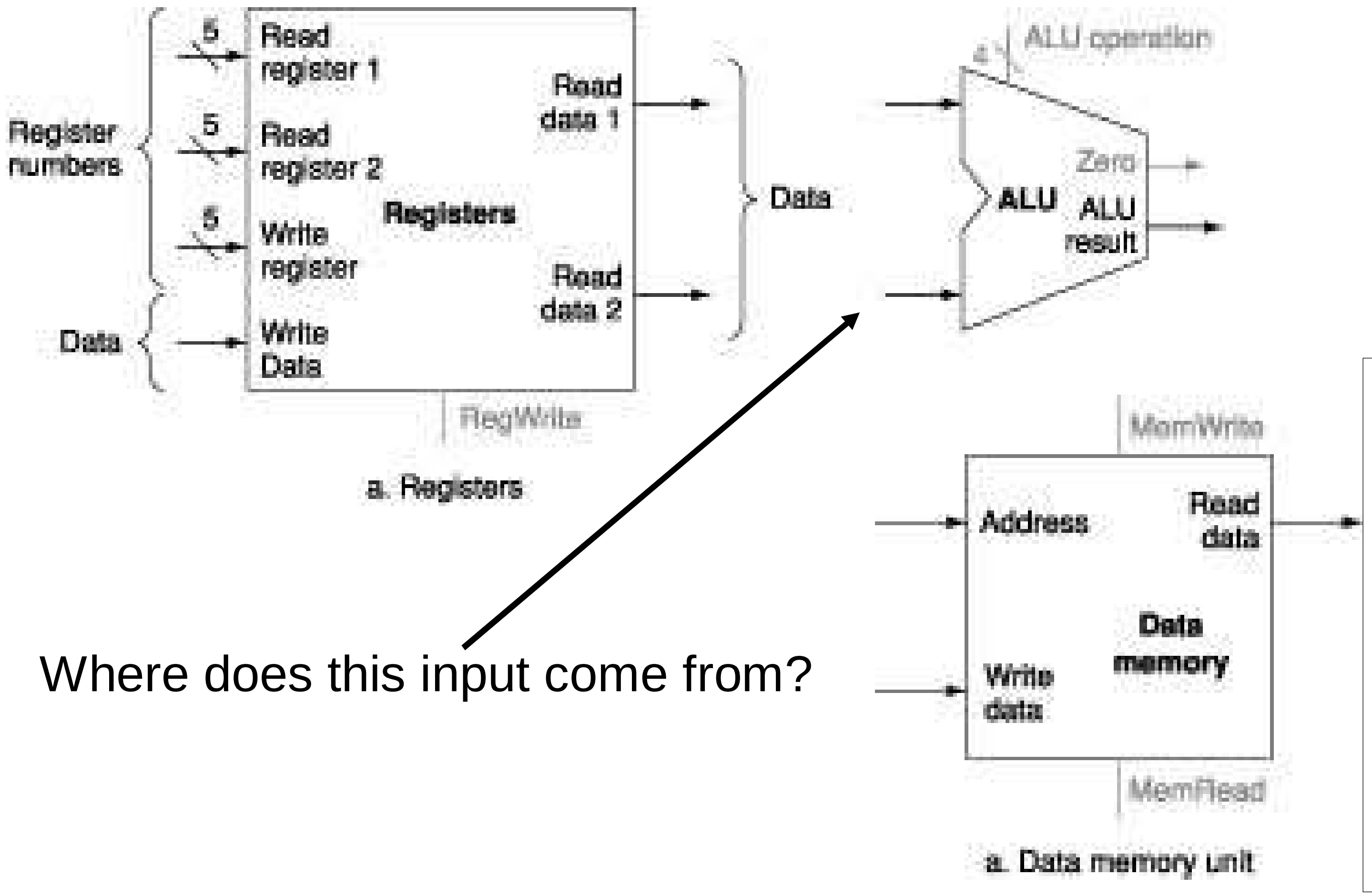
Implementing R-type Instructions

- Instructions of the form add $t1, $t2, $t3
- Explain the role of each signal



Implementing Loads/Stores

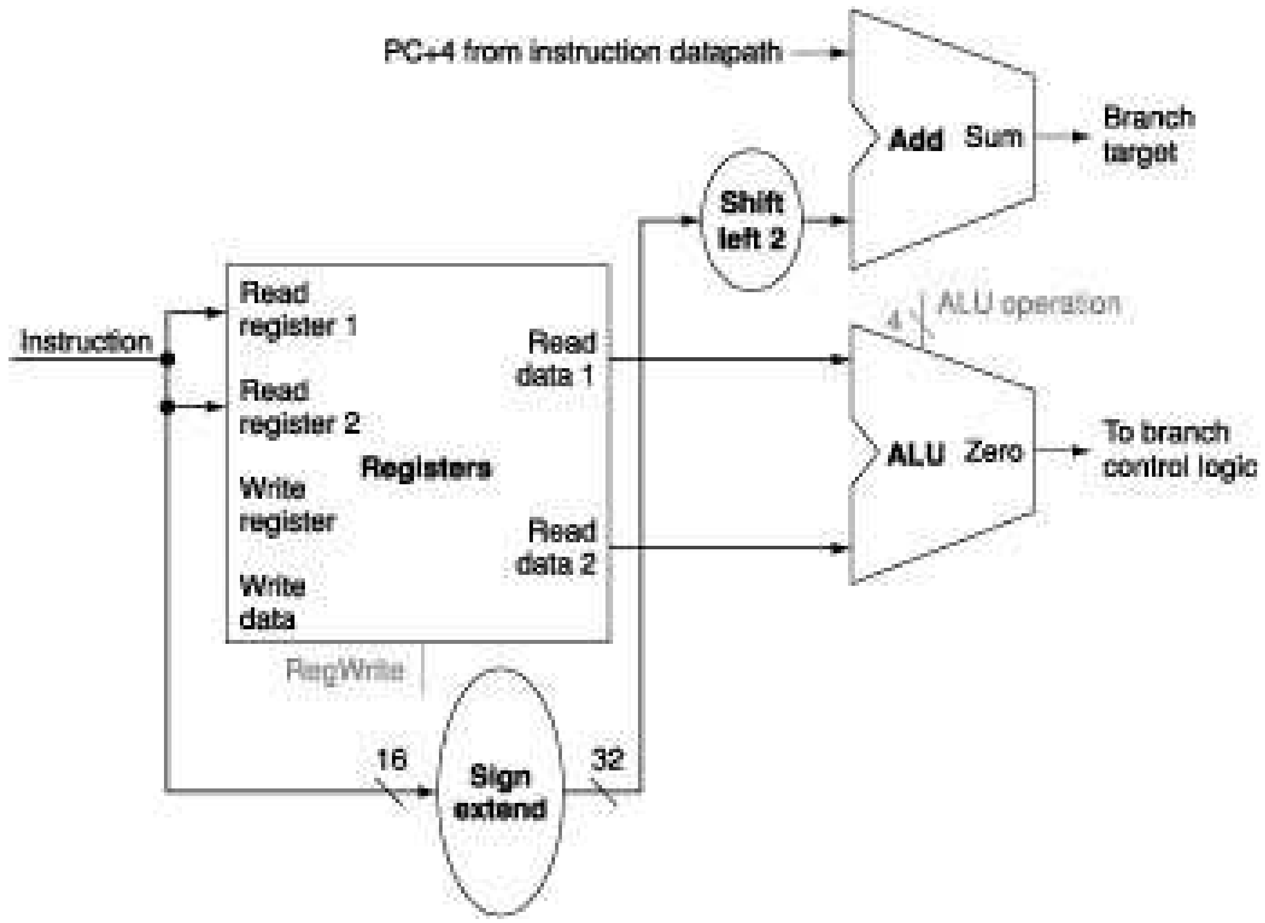
- Instructions of the form   lw $t1, 8($t2)   and   sw $t1, 8($t2)



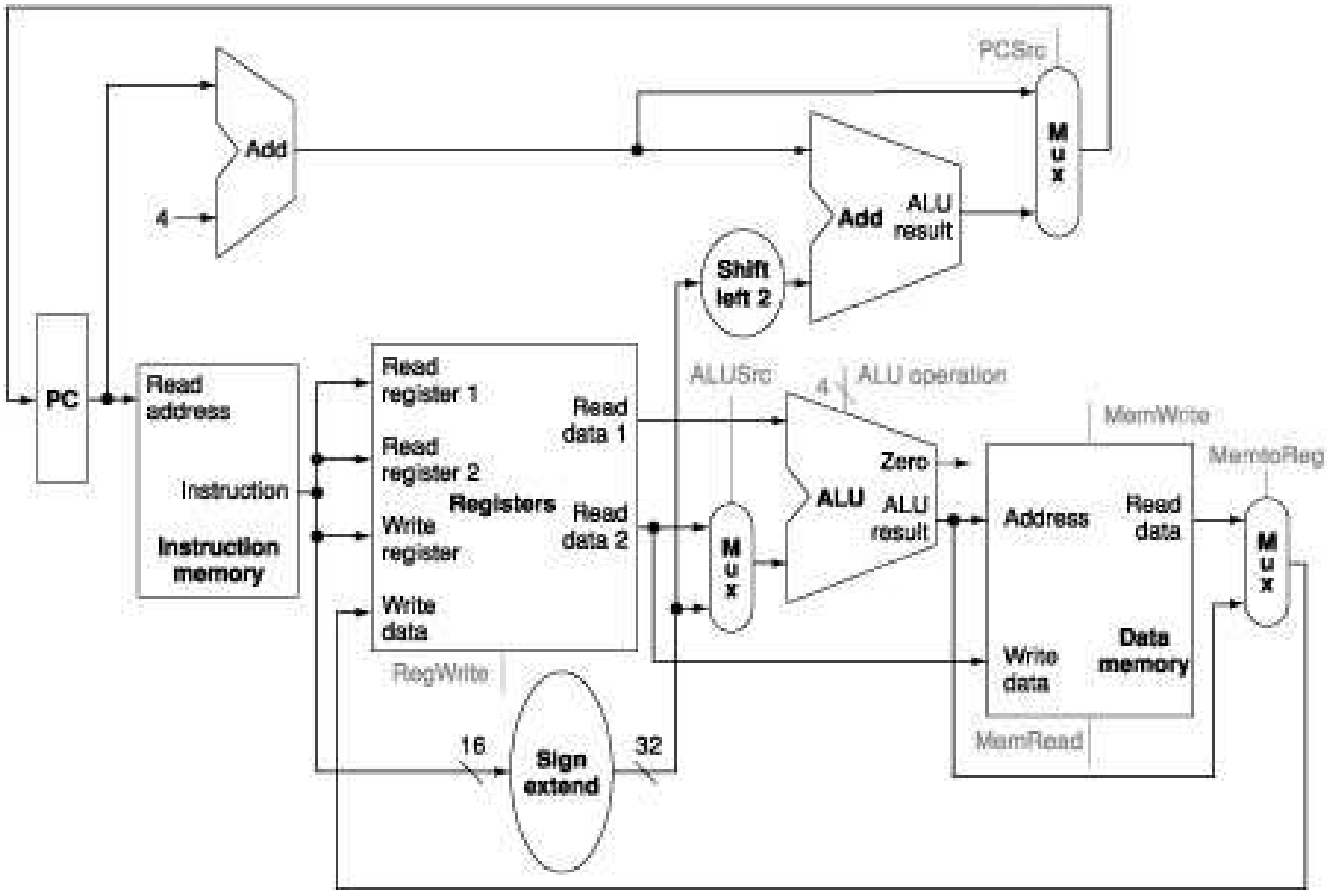
Where does this input come from?

Implementing J-type Instructions

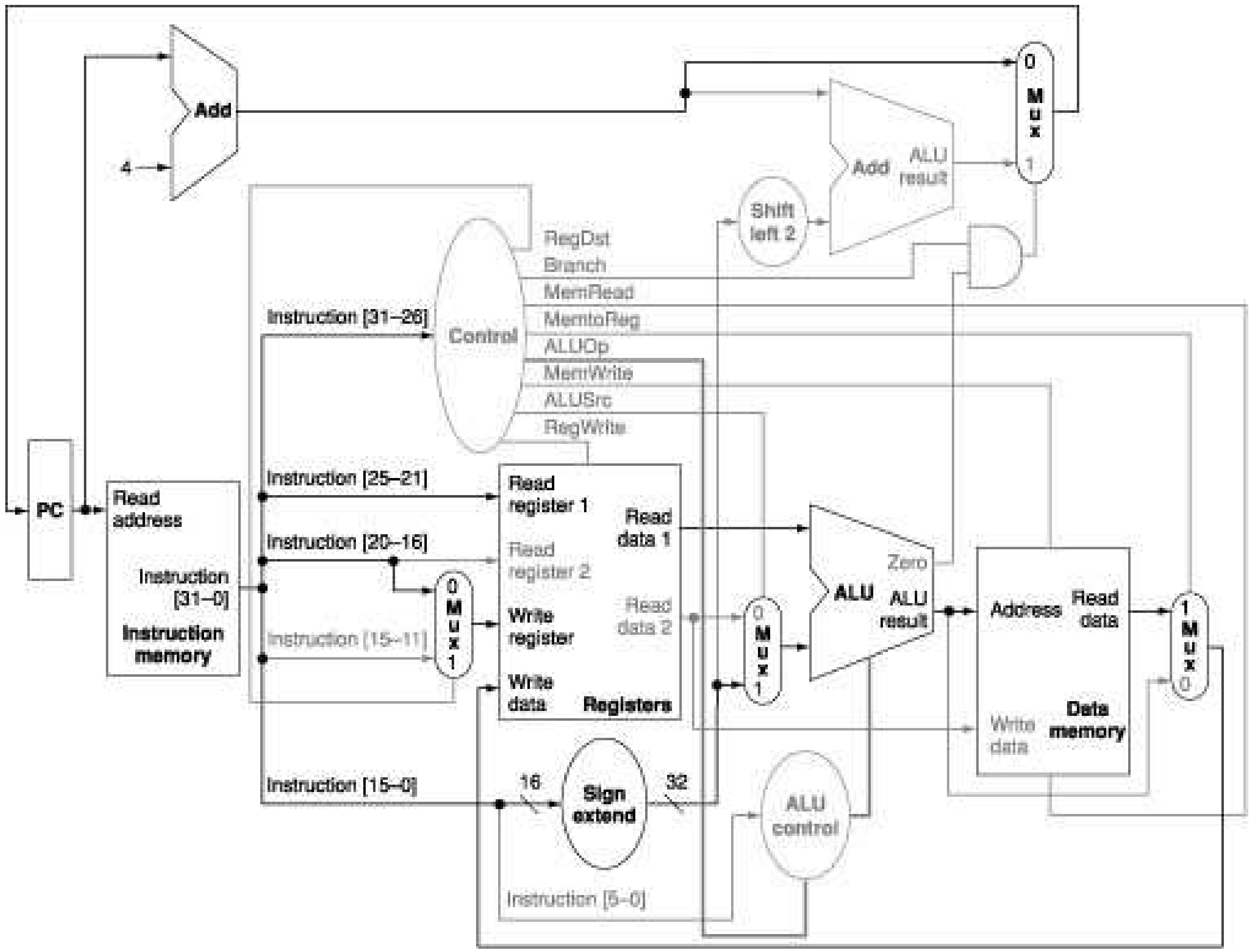
- Instructions of the form beq $t1, $t2, offset



View from 10,000 Feet



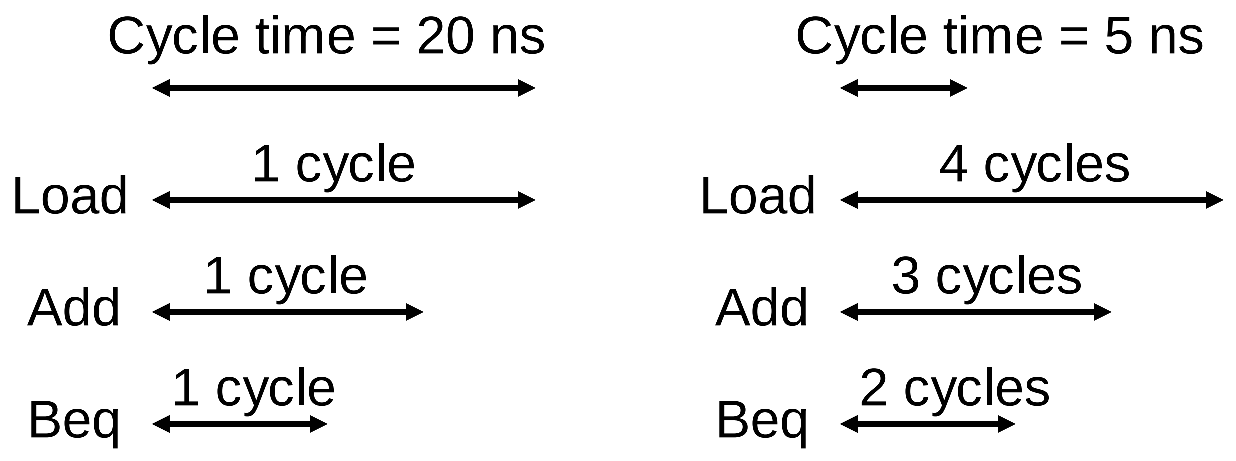
View from 5,000 Feet



Single Vs. Multi-Cycle Machine

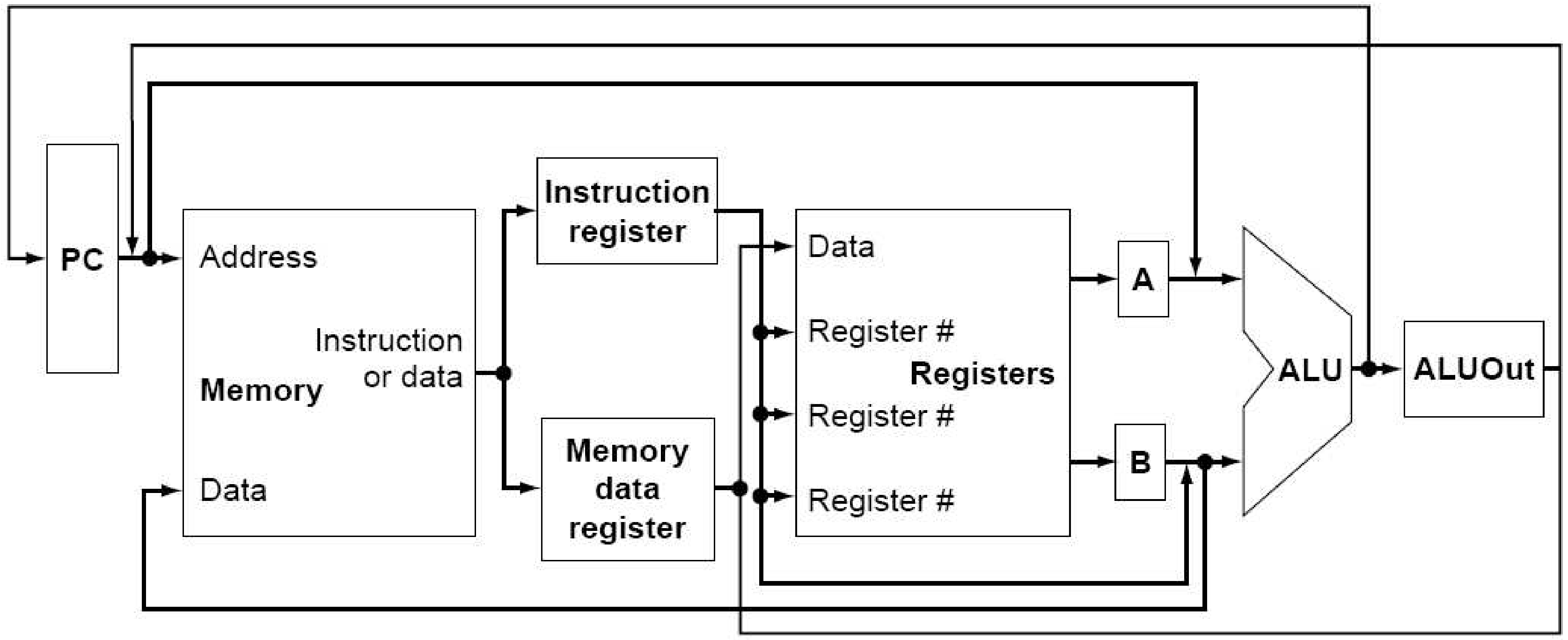
- In this implementation, every instruction requires one cycle to complete ➔ cycle time = time taken for the slowest instruction

- If the execution was broken into multiple (faster) cycles, the shorter instructions can finish sooner



Time for a load, add, and beq = 60 ns 45 ns

Multi-Cycle Processor



- Single memory unit shared by instructions and memory
- Single ALU also used for PC updates
- Registers (latches) to store the result of every block

Cycle 1

- The PC is used to select the appropriate instruction out of the memory unit
- The instruction is latched into the instruction register at the end of the clock cycle
- The ALU performs PC+4 and stores it in the PC at the end of the clock cycle (note that ALU is free this cycle)
- The control circuits must now be “cycle-aware” – the new PC need not look up the instr-memory until we’re done executing the current instruction

Cycle 2

- The instruction specifies the required register values – these are read from the register file and stored in latches A and B (this happens even if the operands are not required)

- The last 16 bits are also used to compute PC+4+offset (in case this instruction turns out to be a branch) – this is latched into ALUOut

- Note that we haven’t yet figured out the instruction type, so the above operations are “speculative”

Cycle 3

The operations depend on the instruction type

- Memory access: the address is computed by adding the offset to the value read from the register file, result is latched into ALUOut

- ALU: ALU operations are performed on the values read from the register file and the result is latched into ALUOut

- Branch: the ALU performs the operations for “beq” and if the branch happens, the branch target (currently in ALUOut) is latched into the PC at the end of the cycle

- Note that the branch operation has completed by the end of cycle 3, the other two are still

Cycle 4

- Memory access: the address in ALUOut is used to pick out a word from memory – this is latched into the memory data register
- ALU: the result latched into ALUOut is fed as input to the register file, the instruction stored in the instruction-latch specifies where the result is written into
- At the end of this cycle, the ALU operation and memory writes are complete

Cycle 5

- Memory read: the value read from memory (and latched in the memory data register) is now written into the register file

- Summary:
  - Branches and jumps: 3 cycles
  - ALU, stores: 4 cycles
  - Memory access: 5 cycles

  ALU is slower since it requires a register file write
  Store is slower since it requires a data memory write
  Load is slower since it requires a data memory read and a register file write

Average CPI

- Now we can compute average CPI for a program: if the given program is composed of loads (25%), stores (10%), branches (13%), and ALU ops (52%), the average CPI is

  0.25 x 5 + 0.1 x 4 + 0.13 x 3 + 0.52 x 4 = 4.12

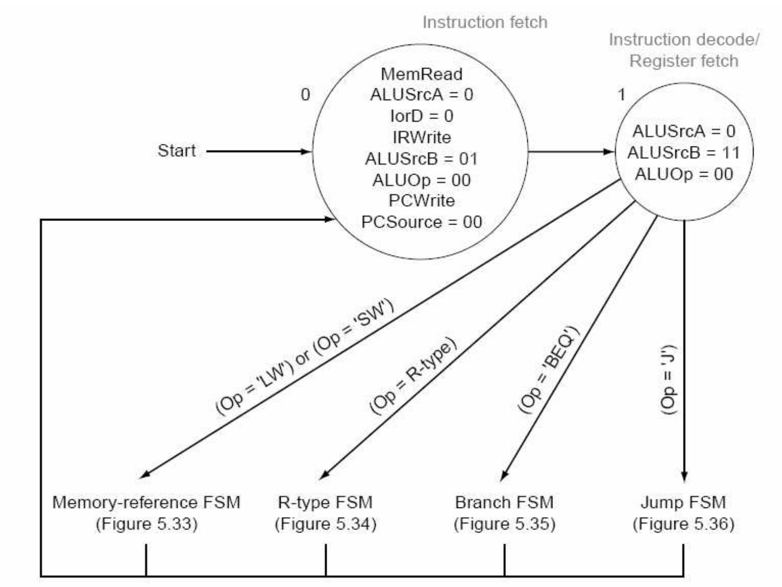
- You can break this CPU design into shorter cycles, for example, a load would then take 10 cycles, stores 8, ALU 8, branch 6 ➔ average CPI would double, but so would the clock speed, the net performance would remain roughly the same

  Later, we’ll see that this strategy does help in most other cases.

Control Logic

- Note that the control signals for every unit are determined by two factors:
    - the instruction type
    - the cycle number for this instruction

The control is therefore implemented as a finite state machine – every cycle, the FSM transitions to a new state with a certain set of outputs (the control signals) and this is a function of the inputs (the instr type)



Title

- Bullet