

# Computer Architecture

- **Having studied numbers, combinational and sequential logic, and assembly language programming, we begin the study of the overall computer system.**
- **The term “computer architecture” is just a fancy word for “the way the computer is put together.” It includes:**
  - **How logic circuits (gates, ff’s, and more complex devices like adders, registers, counters, etc.) are arranged in a modern central processor unit (CPU).**
  - **The relation of the CPU to memory, and how CPU/memory interaction occurs.**
  - **How the computer communicates to the user via peripheral devices such as the keyboard, CRT, printer, scanner, etc.**

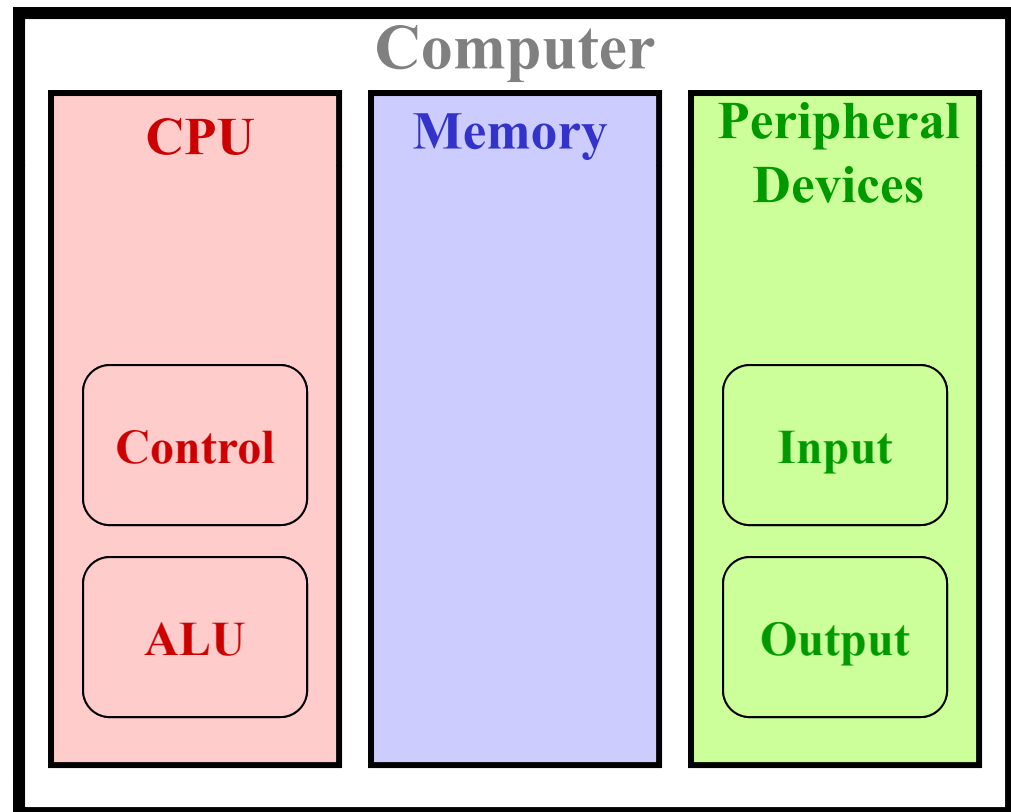


## Course of Study

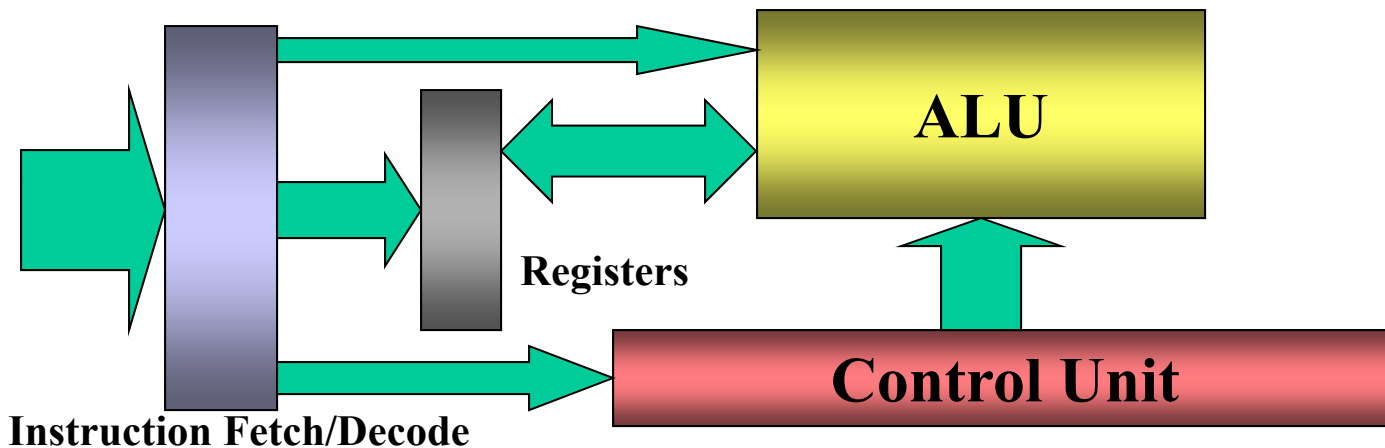
- **We will spend most of the remaining lectures discussing the internals of the CPU.**
- **We will devote some attention to the architecture of the modern computer (with emphasis on the MIPS-style architecture, the predominant approach today).**
- **Major topics of study will be:**
  - **Design of the simple arithmetic-logic unit or datapath.**
  - **Design of the CPU control unit.**
  - **Multicycle implementation, a step toward pipeline architecture.**
  - **Overview of pipelining**
  - **Memory management and design in a modern computer.**

## The ALU or Datapath

- The major components of a computer include:
  - The CPU
  - Memory
  - Peripherals that make the computer useful, such as the disk, printer, and mouse.
- The CPU consists of a control unit and the arithmetic/logic unit (ALU).
- We first study the ALU.



# The Central Processor Unit (CPU)

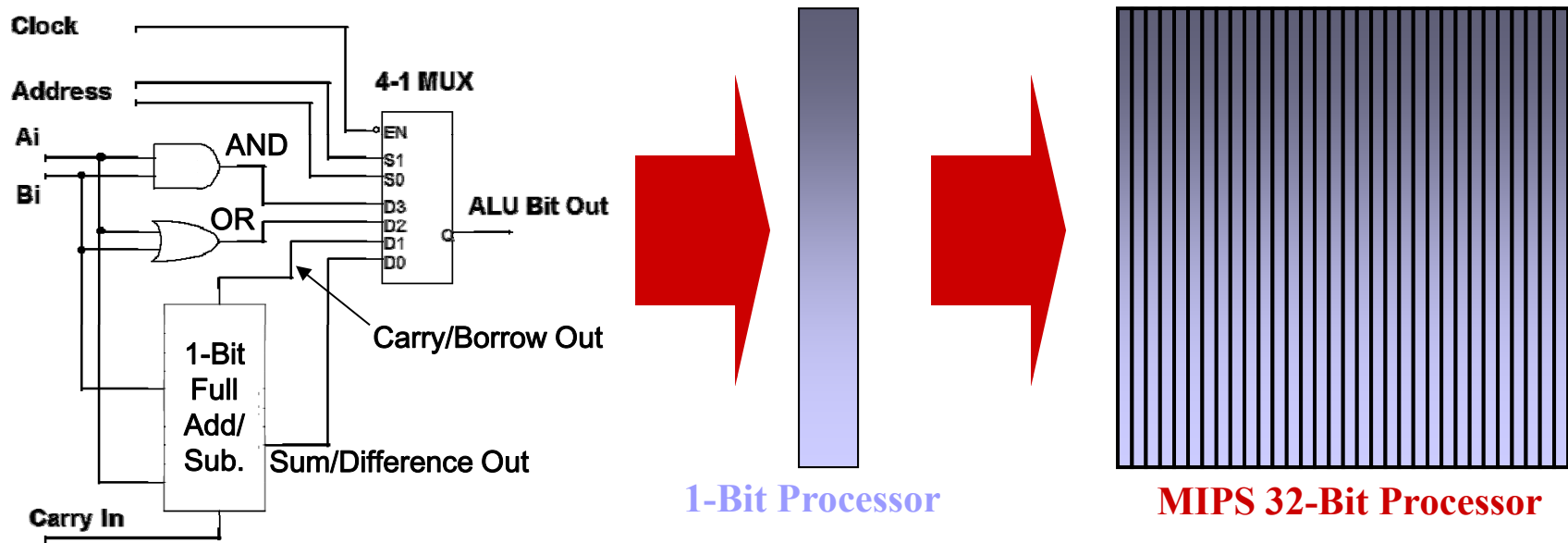


- **The major components of the CPU are:**
  - **A unit to fetch instructions from memory and decode them.**
  - **The arithmetic-logic unit (ALU) or datapath, to process data.**
  - **Registers to hold arguments and results of ALU processing.**
  - **A control unit to decode instructions and initiate ALU action.**

## The ALU

- The **ALU** (**A**rithmetic-**L**ogic **U**nit, or Patterson and Hennessy's **datapath**) is the “figuring” unit of the computer, providing the calculation capability.
- The **ALU** is connected to storage elements (**registers** for holding data to be processed and the results of the processing) by data buses.
- The **ALU** processor is normally composed of single-element (one-bit) processors (primarily adders), which are assembled together to form a 32-bit processor, in the case of the MIPS R2000. It also includes some other processing elements (**such as a shifter**).

# The MIPS Computer: An Example of “Bit-Slicing”



- Just as, in Lecture 4, we saw how we could use 1-bit adders to compose a 32-bit adder, the MIPS 32-bit ALU processing unit is simply an amalgam of 32 1-bit processors.

## ALU Components

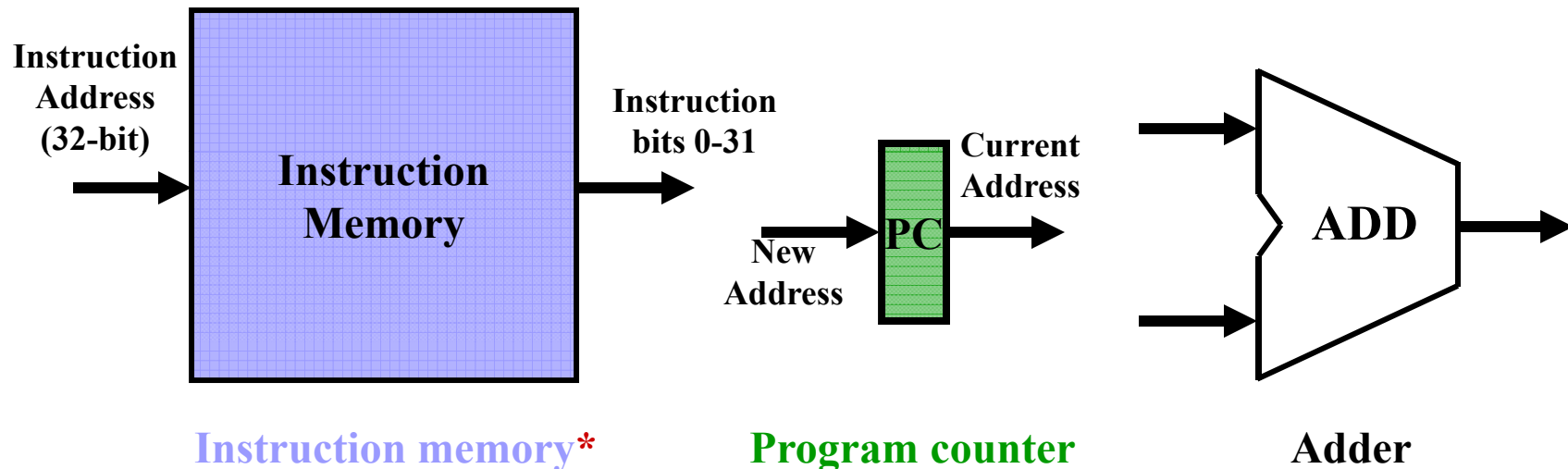
- **As noted above, the ALU needs other components as well:**
  - **Registers to store arguments and results.**
  - **Buses to carry data from registers to the ALU and results back to the register unit.**
  - **Two memory access units, with associated buses:**
    - **An instruction fetch unit to get instructions from computer memory as needed. This includes a program counter, which always points to the address of the next instruction to be accessed.**
    - **A second path to memory to obtain data to be used in the program and to store data back into memory as required.**
  - **A control unit that tells the ALU what to do (covered later).**

## ALU Components

- We will use new symbols in the design of the MIPS computer over the next three lectures.
- **Keep in mind that these are symbols for items you already understand. For instance, the MIPS register block is simply a group of 32-bit registers, and the registers are, in turn, collections of D Flip-Flops.**
- **Thus while the symbols are new, they represent, in general, digital logic with which we are by now quite familiar.**
- In most cases, the MIPS illustrations shown are from the Patterson and Hennessy book.\* Such illustrations are credited as they occur.

\* David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition

# ALU Components (3)



Instruction memory\*

Program counter

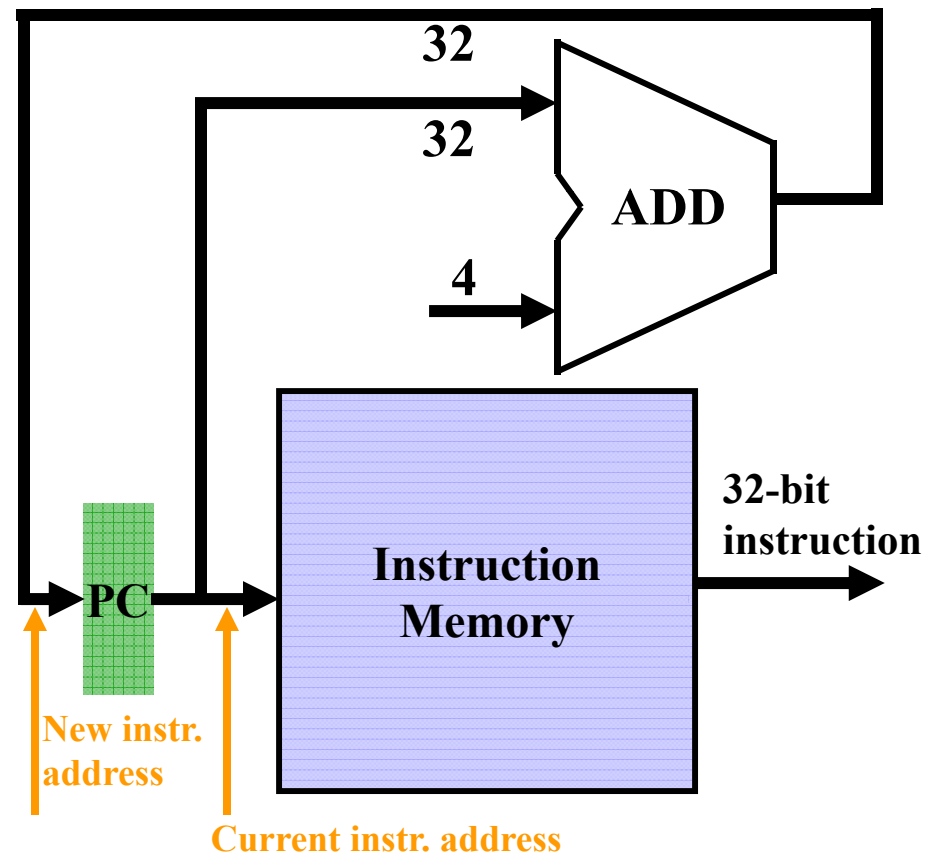
Adder

After David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition

- Shown above are a number of the symbols for components of the computer that we will be “designing.”
- \* We note that there is really only a single memory in modern computers, and the terms “data memory” or “instruction memory” refer only to the channel or pathway into the common memory unit.

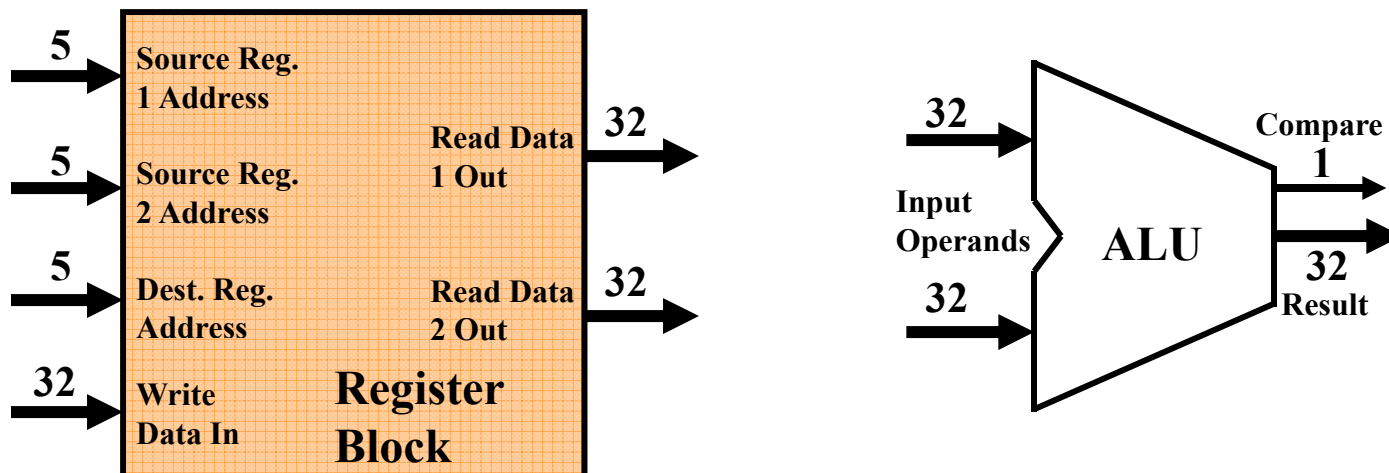
# Program Counter Architecture

- The program counter (PC) is a register that addresses the next instruction in memory.
- Since the PC always points to the “next instruction,” this address must be readily updated.
- The usual method is to add 4 to the current address (since the instruction is a 32-bit word and each 32-bit word has an address 4 bytes higher than the last word).



After David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition

## More ALU Components

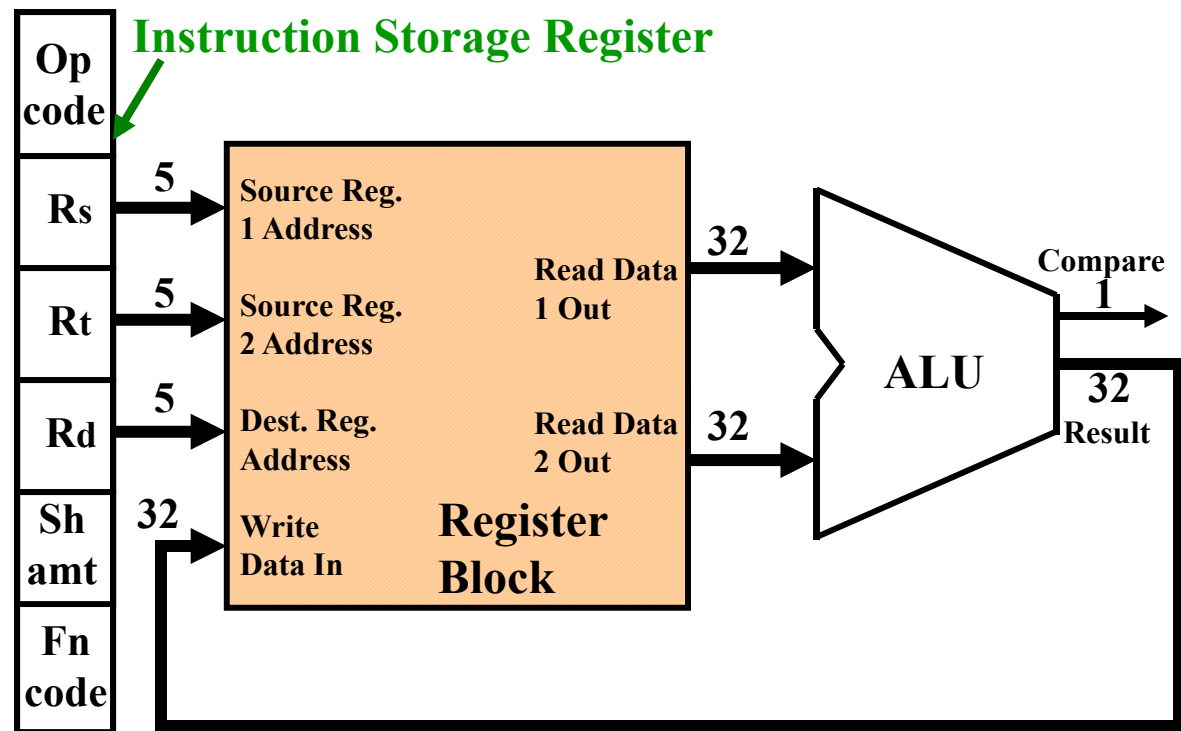


After David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition

- For the ALU, we need some more components: **(1) a register block (group of registers) in which to store operands and results, and (2) the ALU itself, which is primarily combinational logic, as mentioned earlier.** Since the ALU is largely an adder plus other logic, we use the adder symbol.

# ALU Architecture for Processing

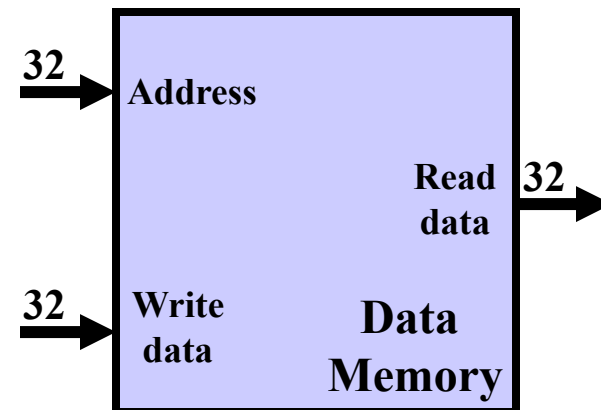
- In the MIPS CPU, **all arguments used in processing are stored in registers.**
- This requires **two buses** to route data to the ALU.
- **All registers are tied to both output buses.**
- A single input bus carries results back to all 32 registers.



After David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition

## More ALU Components – Data Memory

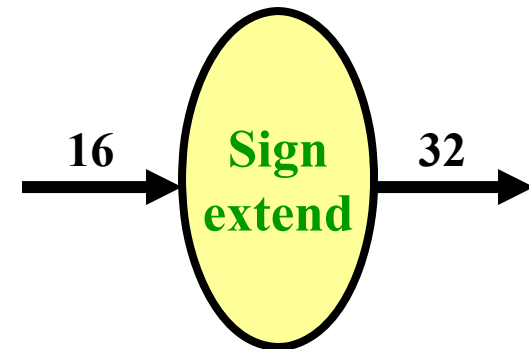
- In addition to instruction memory, we need data memory.
- Now in reality, memory is memory, so when we say “Data memory,” we simply mean we need a path to the single computer working memory to load/store data in load/store instructions.
- We color code data memory blue, like instruction memory, to remind us that it is the same memory, only accessed along a different path or bus.



After David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition

## The Sign Extender

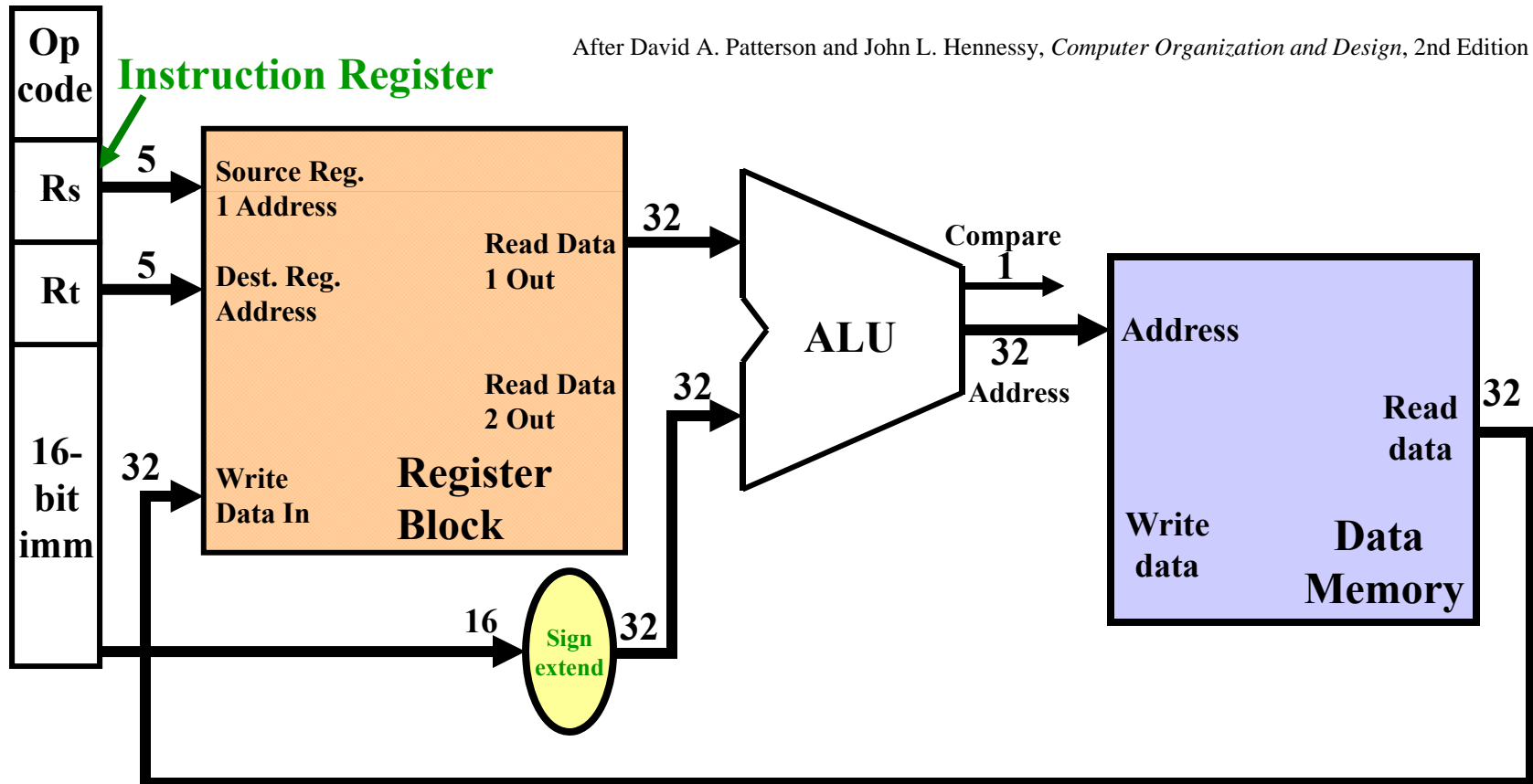
- We remember from studying MIPS instructions and SPIM that many instructions have immediate parts that are 16 bits in length.
- **The immediate is added to a 32-bit operand (register or PC contents) to create an operand or perhaps a new address, for instance in a branch instruction.**
- Since the immediate may be + or −, then to add it to a 32-bit argument, we must sign extend it to 32-bits (as we discussed in the MIPS/SPIM lectures).
- **The sign extension unit performs this function.**



After David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition

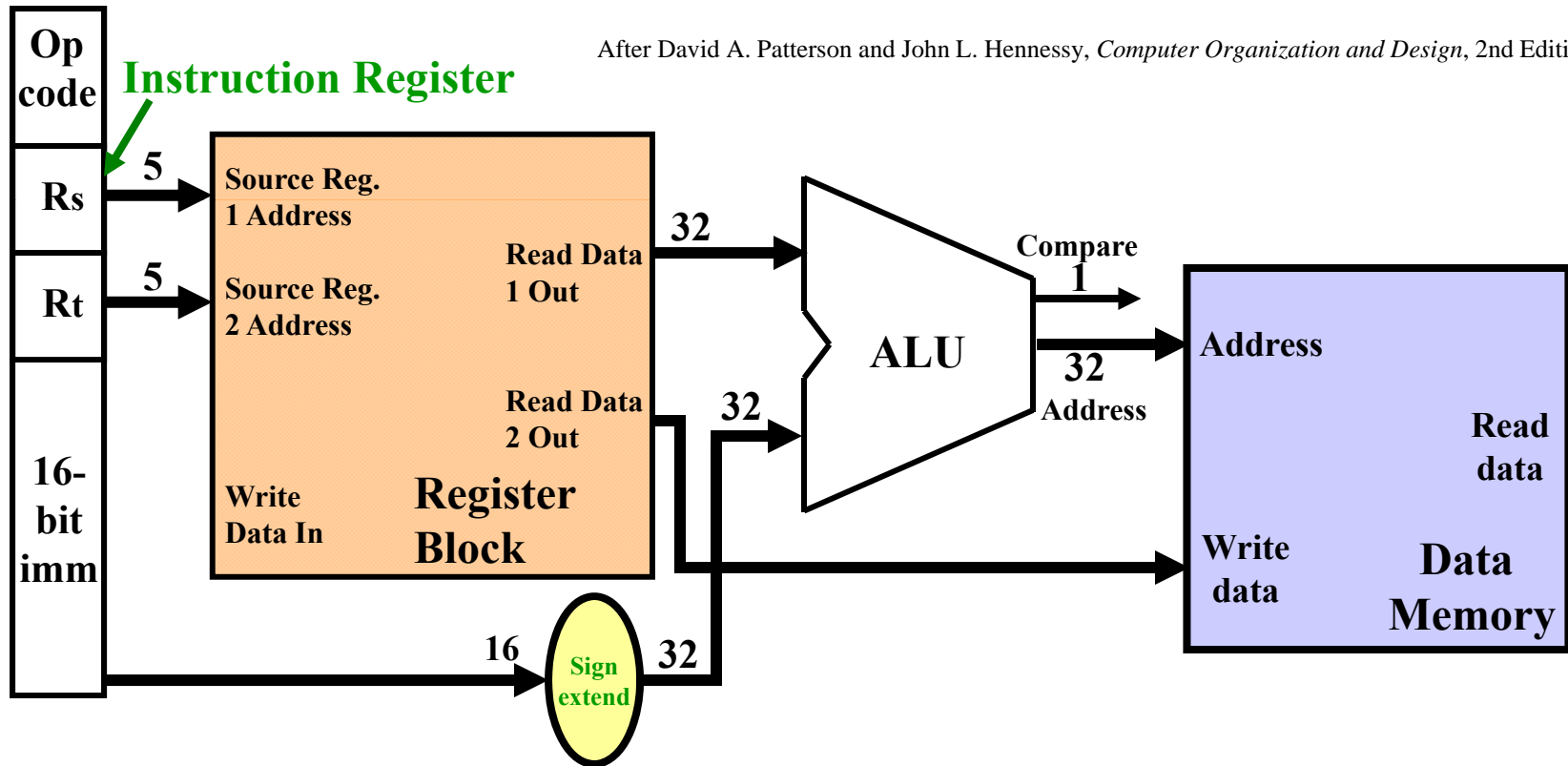
# Data Bus Connection in a Load Instruction

After David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition



**Note that in a load instruction, the ALU performs the address calculation.**

# Data Bus Connection in a Store Instruction

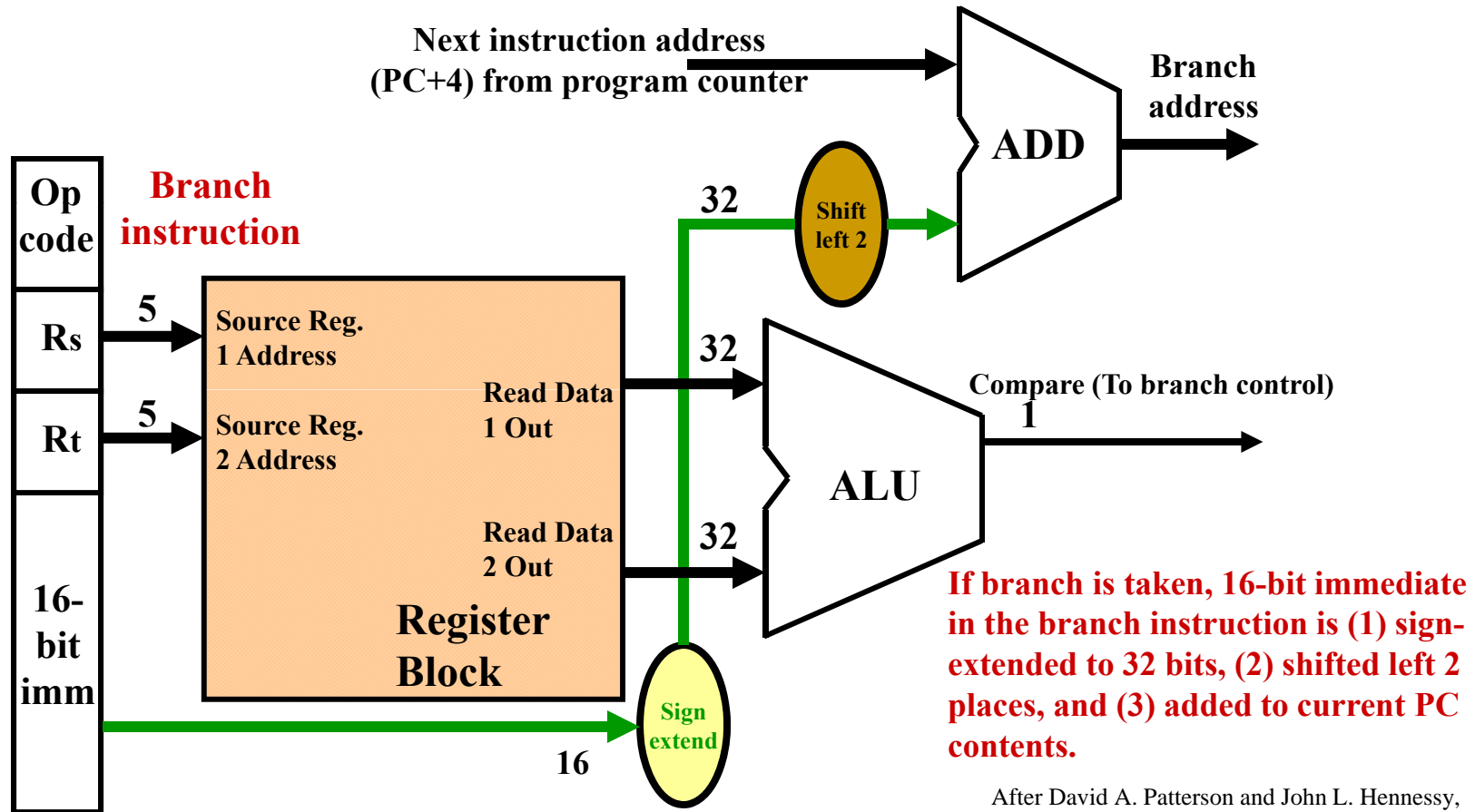


**In store instructions, the ALU also calculates the memory address.**

## Branch Instructions

- A branch instruction will have one of two results: (1) **the branch is not taken, in which case the next instruction is executed (i.e., instruction at address  $[PC]+4$ )**, or (2) **the branch is taken, in which case the program counter gets an entirely new address (new address =  $[PC] + 16\text{-bit immediate from instruction } [\pm 32,768]$ )**.
- In the first case, **we simply use the PC update scheme shown on slide 10 of this lecture**. In the second case, **the current PC contents (next instruction address, that is,  $[PC]+4$ ), is added to the 16-bit immediate contents of the instruction to form the new address**.
- Not only do we need an add function, but also a left shift 2 places (since memory is addressed as bytes, but the 16-bit immediate in the branch instruction is a **word address**).
- This scheme is shown on the next slide.

# Conditional Branch Circuit

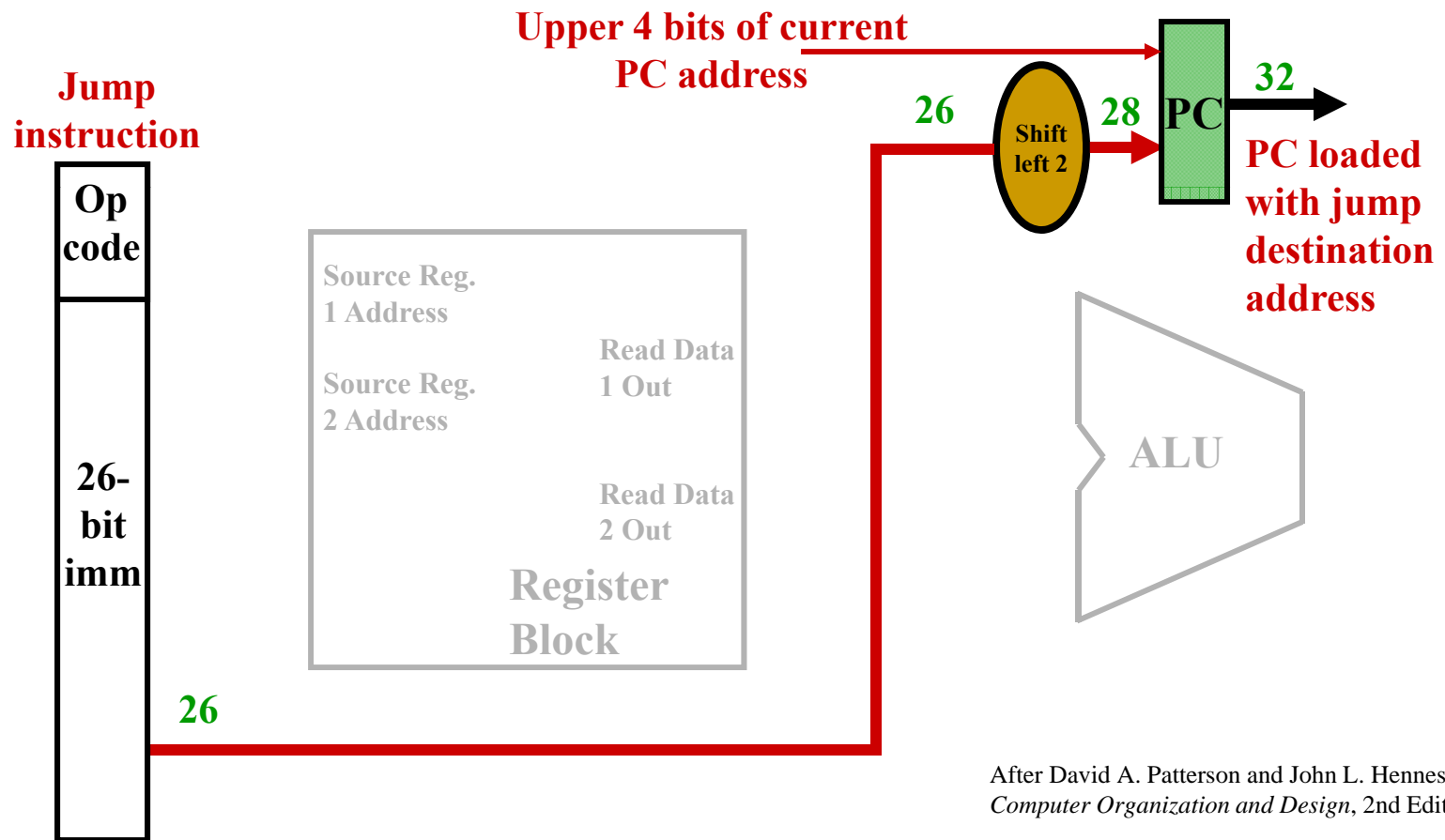


After David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition

## Jump Instruction Modifications

- A jump instruction **unconditionally transfers program control** to a new location in the program.
- The ALU must therefore use the 26-bit immediate address **in the instruction** to create a new PC address.
- This is done by **(1) shifting left 2 (to create a proper 28-bit byte address)**, and **(2) adding the upper four PC bits to complete the new 32-bit address**.
- This new address is then loaded into the program counter as the address of the next instruction to be executed.

# Jump ALU Path



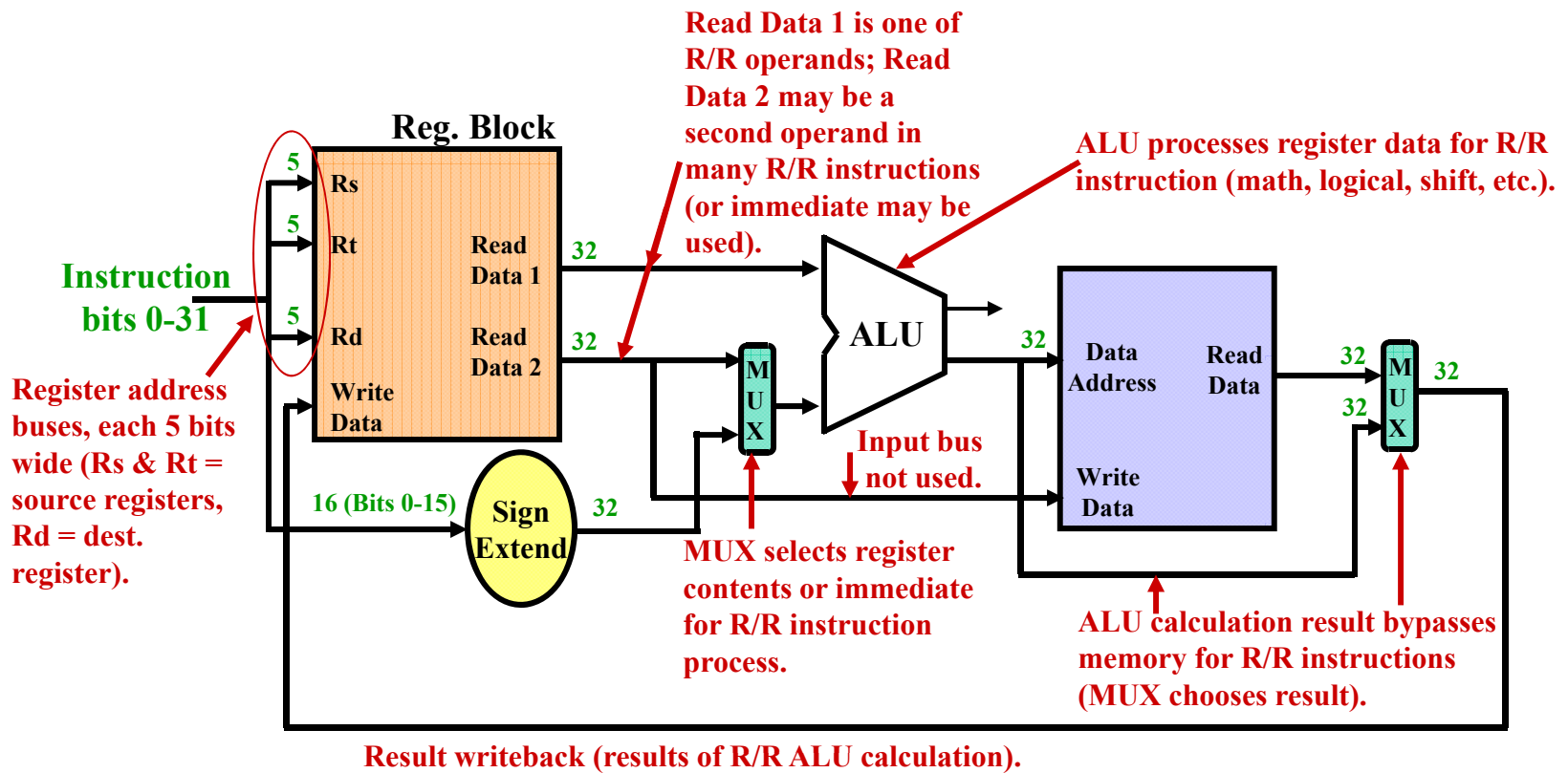
After David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition



## Combining the Elements to Make a Complete ALU

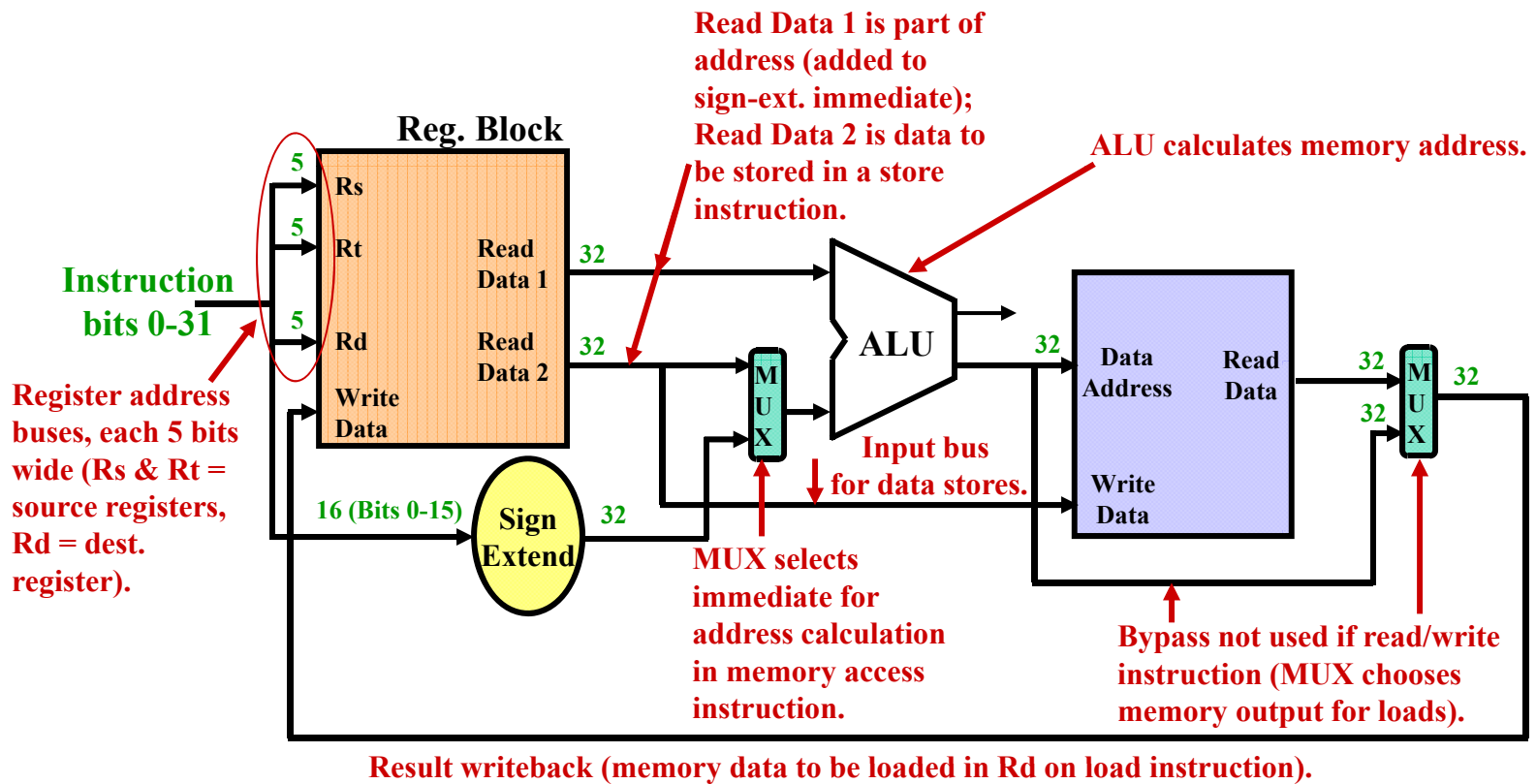
- **Over the last ten or so slides, we have “designed” many of the elements that, when assembled, will make up the ALU of our MIPS computer.**
- **We now connect the various sections of the ALU together to produce a final “single cycle” ALU; an ALU that executes one instruction each time the “clock” ticks.**
- **Note that up to now, we have NOT considered any of the Control elements of the ALU; that is, the circuits that interpret the instructions and direct the ALU as to (1) which operation to execute, and (2) what operands or variables to process.**
- **We will cover the so-called Control elements and a multi-cycle implementation in the next lecture.**

# Data Buses and ALU Register/Register Functions



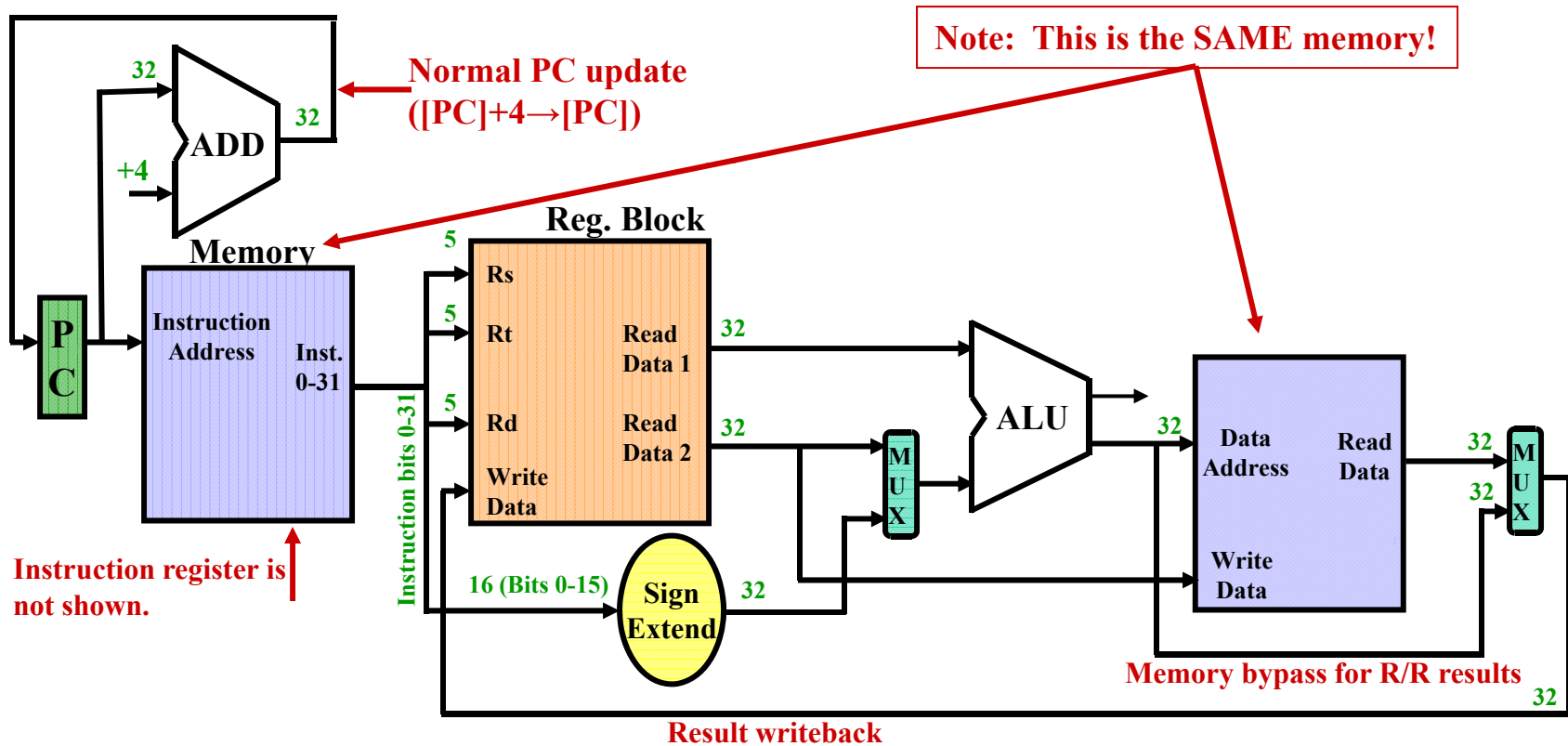
After David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition

# Load/Store Functions (Read or Write)



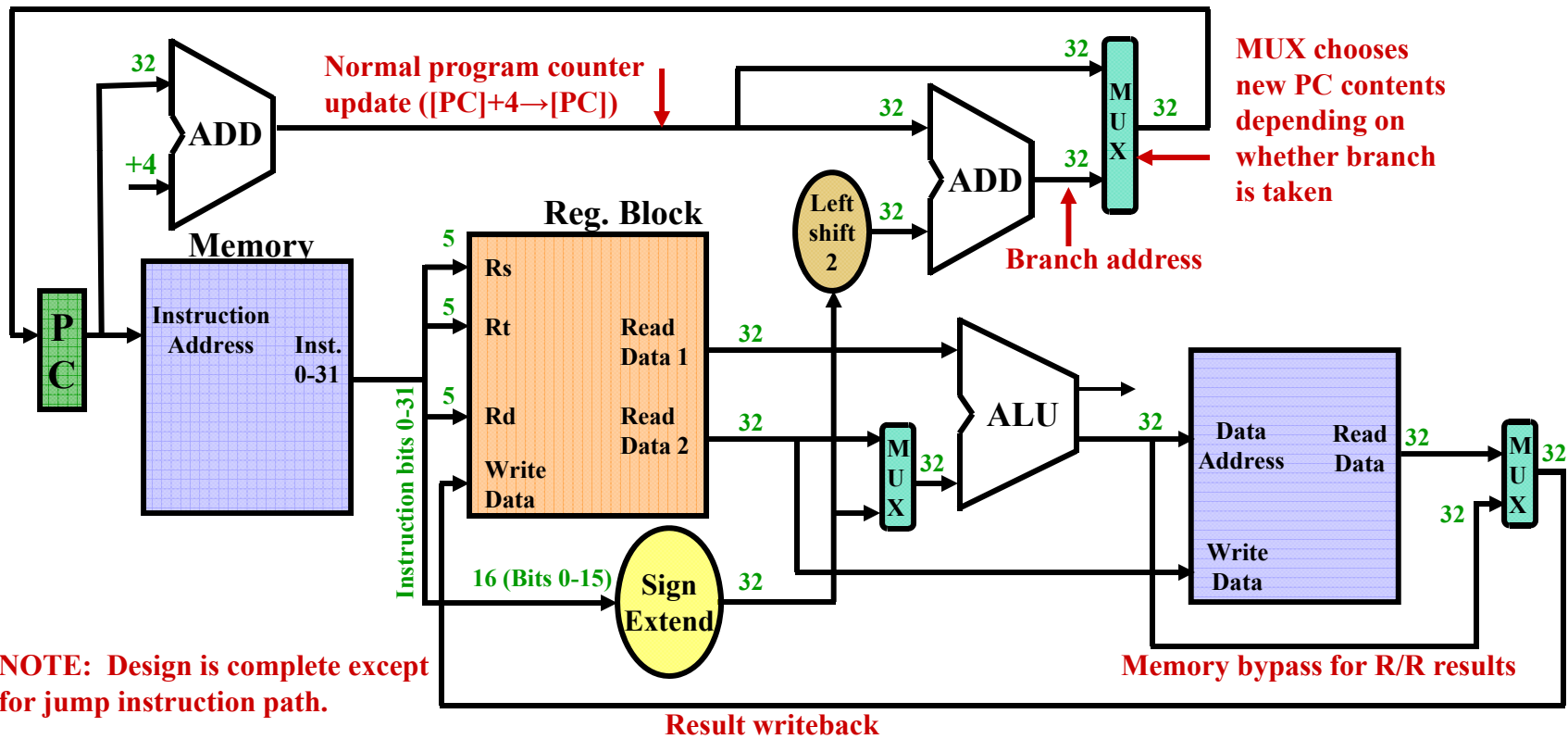
After David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition

# Adding the Instruction Fetch Circuit



After David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition

# Completing ALU Design



**NOTE:** Design is complete except for jump instruction path.

After David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition

## The “Single Cycle” ALU

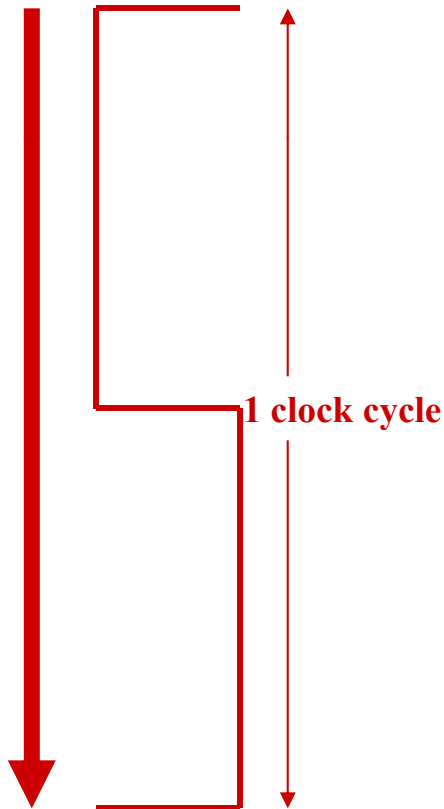
- **What we have accomplished in this “ALU design” is to come up with all the processing hardware necessary to implement the MIPS instruction set. Note that we have NOT considered the control circuits (that tell the ALU what to do), and will cover those next lecture.**
- **This basic MIPS processor design is referred to as the “single cycle ALU” (or sometimes the “single cycle CPU”). Why is this?**
- **The reason is that this MIPS CPU (more or less the original implementation of the MIPS instructions) is designed so that ANY instruction can be processed in one cycle of the CPU clock.**
- **Lets consider how this “single cycle” CPU works.**

## The “Single Cycle” ALU (2)

- Since the ALU is basically combinational logic, the “tick” of the clock governs ALU register behavior, which times the process.
- A single clock cycle is from rising to rising or falling to falling edge of the clock. Let us use falling to falling as the reference (remember: a master-slave ff’s output changes on the falling edge of the clock).
- Consider what happens when the clock “ticks.”
  - The PC is already updated.
  - The instruction at memory location [PC] is retrieved. [PC] → [PC+4]
  - Instruction is decoded/registers are identified (operands).
  - Register output buses send data into ALU; ALU function is identified.
  - Register data “flows through” the ALU and is processed.
  - (In loads/stores, data memory is accessed for load or store.)
  - The memory or ALU results are stored back in a register, if necessary.

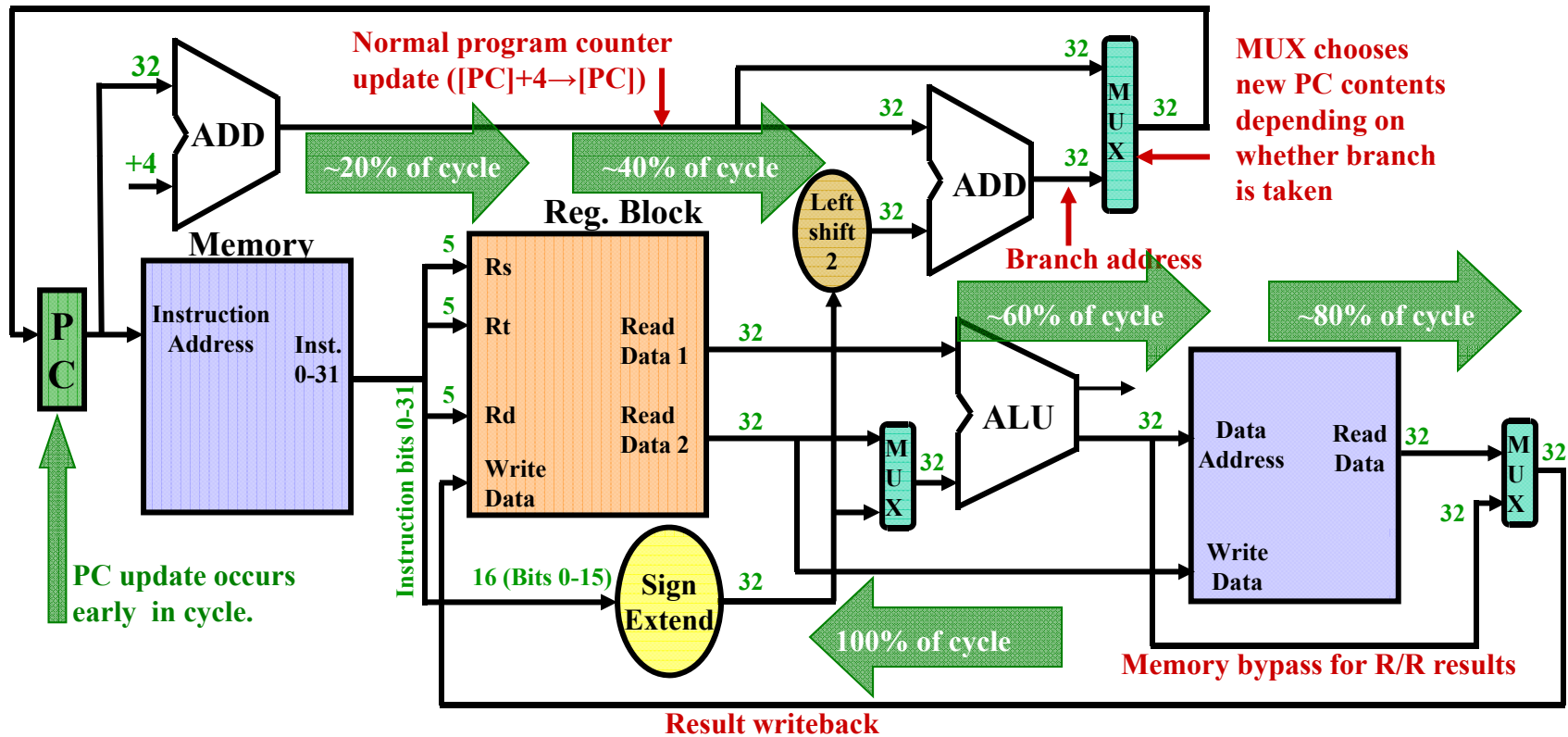
## Single Cycle Timing

Time



1. (Start of cycle) Instruction retrieved into instruction register.
2. (PC contents updated.)
3. Instruction decoded and registers accessed.
4. Register contents gated onto ALU input bus (immediate operands sign-extended and input to ALU).
5. ALU function identified and ALU result obtained.
6. ALU results used to access memory location if required.
7. Memory data written (store) or retrieved (load) if memory access instruction.
8. Data gated onto register input bus.
9. (End of cycle) Instruction result stored in designated register if required.

# Single-Cycle Timing



After David A. Patterson and John L. Hennessy, *Computer Organization and Design*, 2nd Edition



## ALU Design Summary

- We have now seen how the common elements we studied earlier are combined to make the MIPS ALU.
- This design is a solid, “single-cycle” implementation that supports the MIPS instruction set we studied in our lectures on assembly language programming.
- We will study the control logic for this in the next lecture.
- The “single-cycle” implementation has a few drawbacks, however, and we will also discuss in the next lecture how to improve on this design to speed it up considerably.

## **“Single-Cycle” ALU Quiz**

- **Why is this architecture called the “single-cycle” architecture?**
- **What is the difference in instruction and data memory?**
- **Why does the CPU need a sign extender?**
- **How is the branch address calculated?**
- **How is the ALU used in load and store instructions?**
- **Why is a MUX needed on one of the ALU inputs?**
- **Why is the jump address field shifted left two bits?**