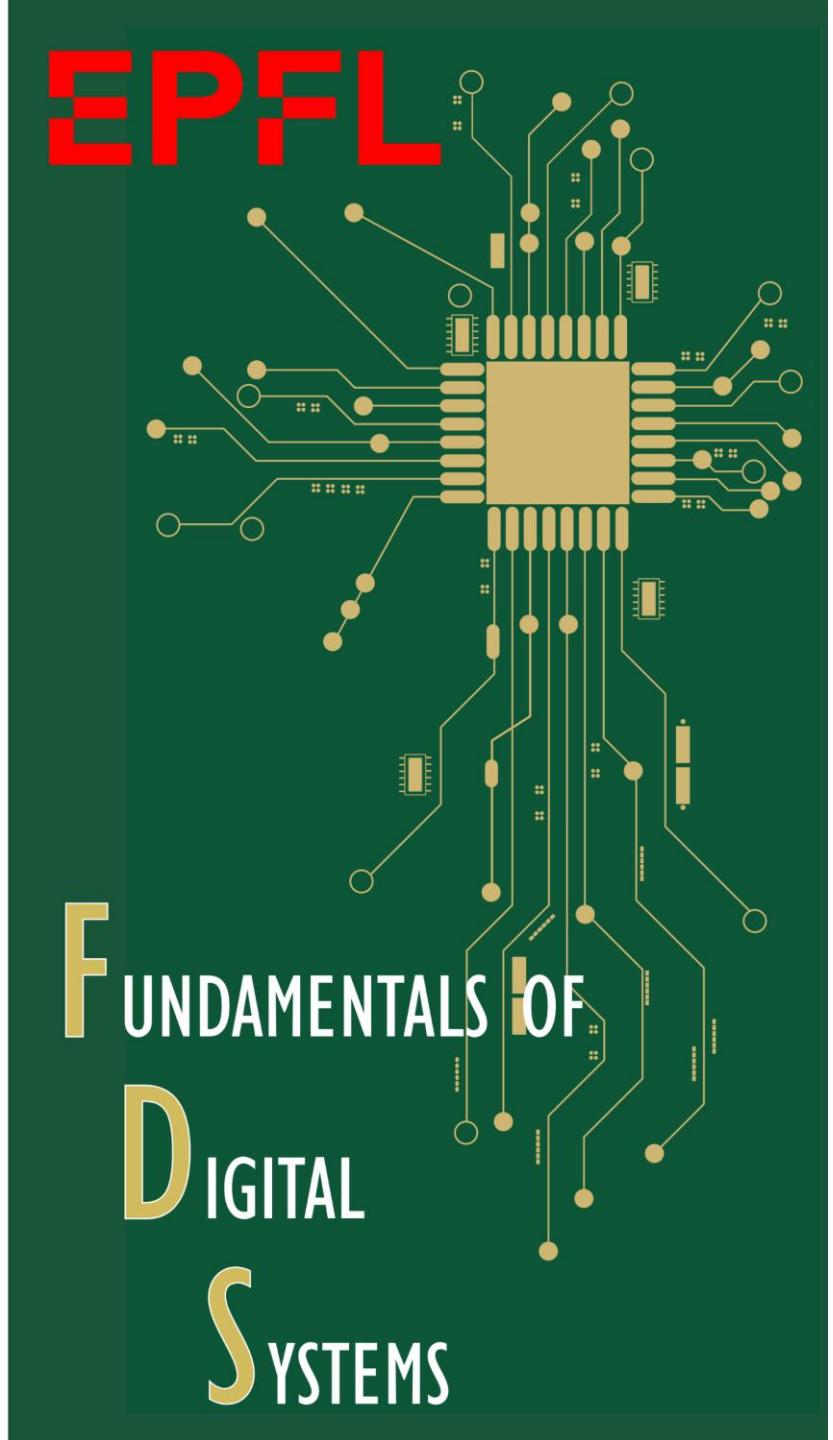


Computer Architecture

Performance and a Single-Cycle CPU

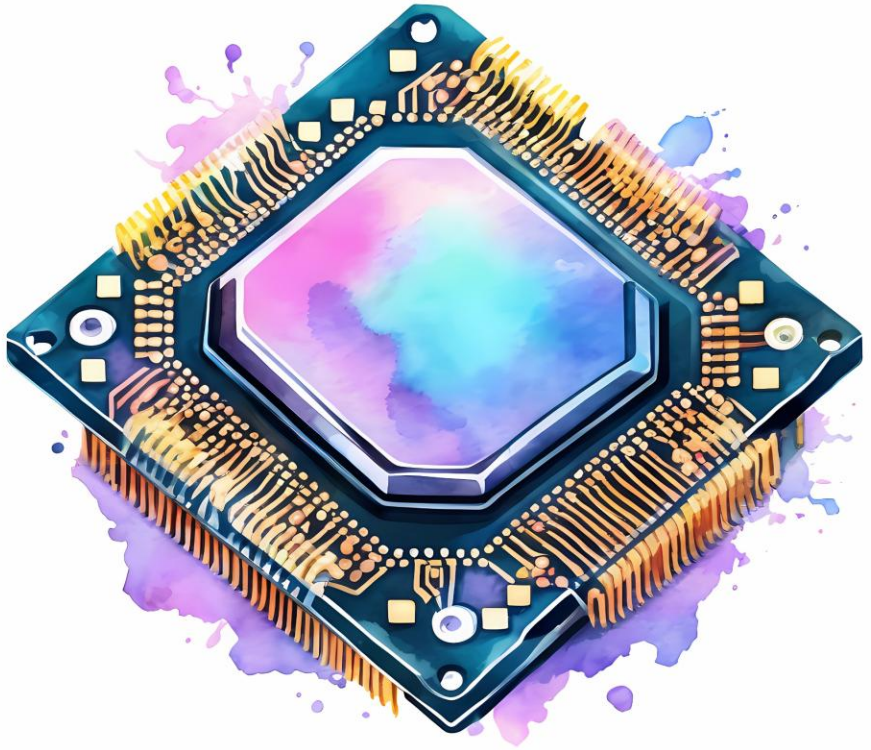
CS-173 Fundamentals of Digital Systems

Mirjana Stojilović

Spring 2025

Previously on FDS

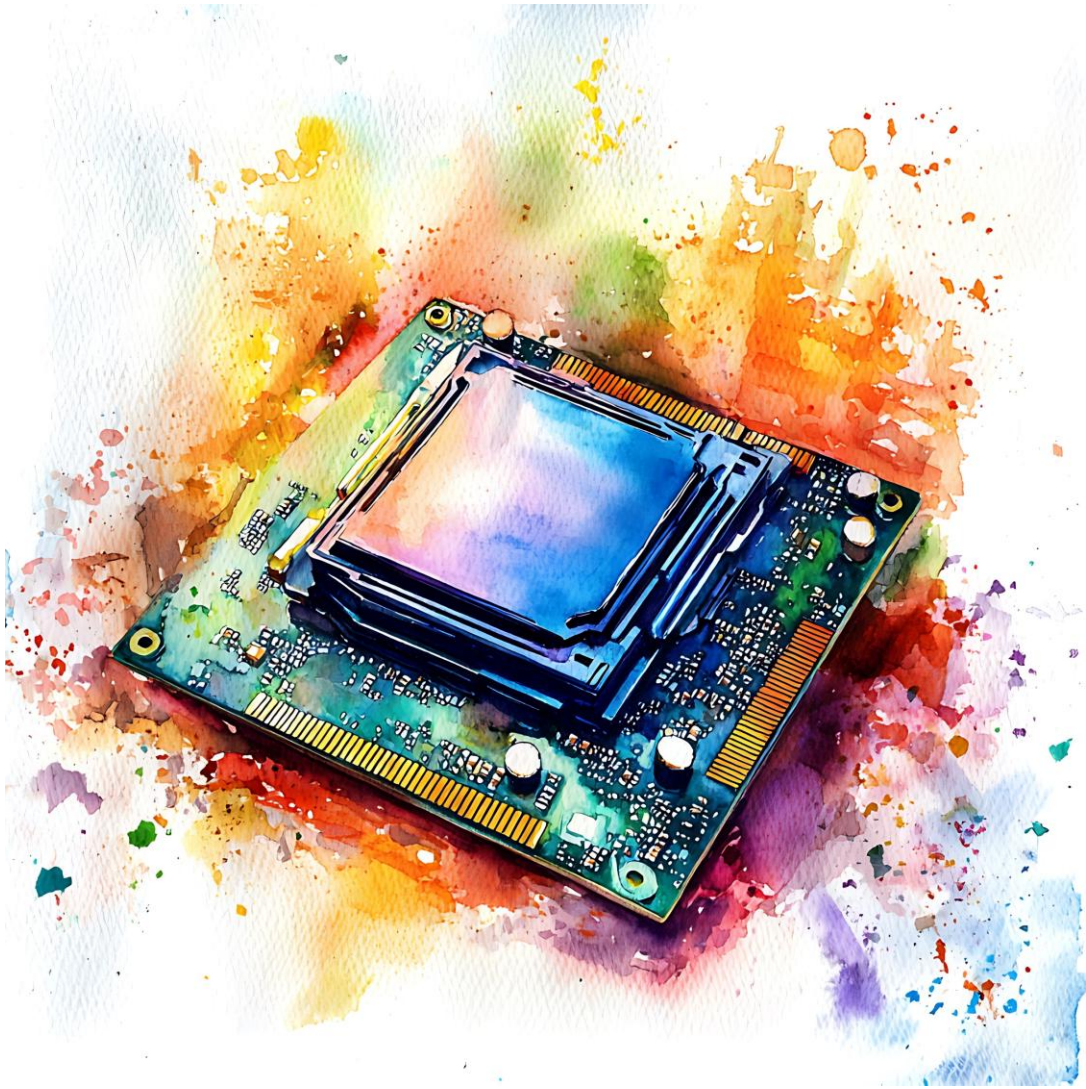
- Assembler directives
- `li` and `la` pseudoinstructions
- Do-while and if-then-else



© Anastasi17 / Adobe Stock

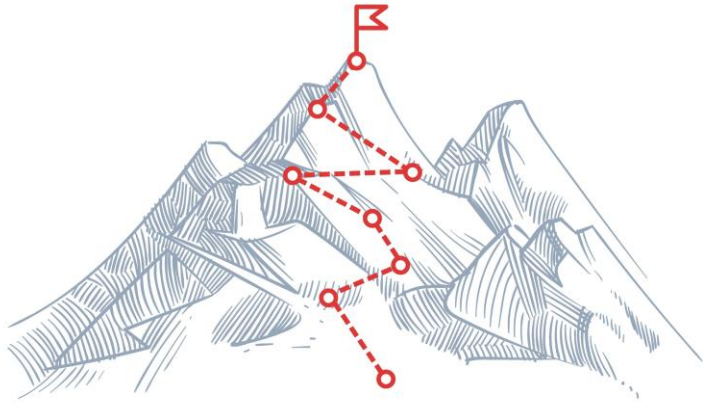
Let's Talk About

- CPU Performance
- Processor Implementations
  - Single-cycle CPU



© Woranuch / Adobe Stock

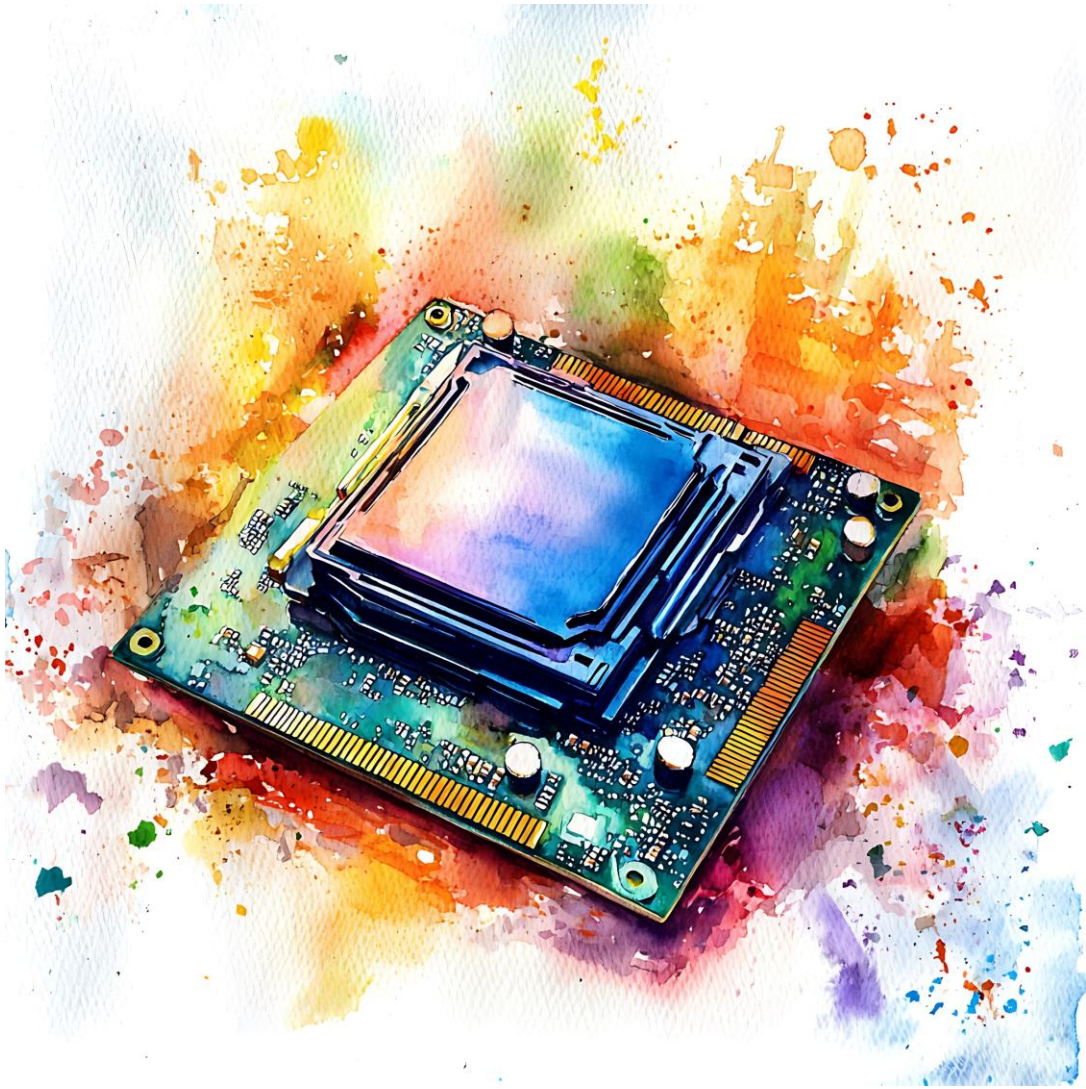
Learning Outcomes



- Reason about CPU performance and the factors affecting it
- Discover single-cycle CPU implementation
  - Pros and cons of single-cycle and multicycle implementations
- Draw a CPU block diagram
  - Datapath + control
- List and explain instruction execution steps
  - The hardware components and control signals involved

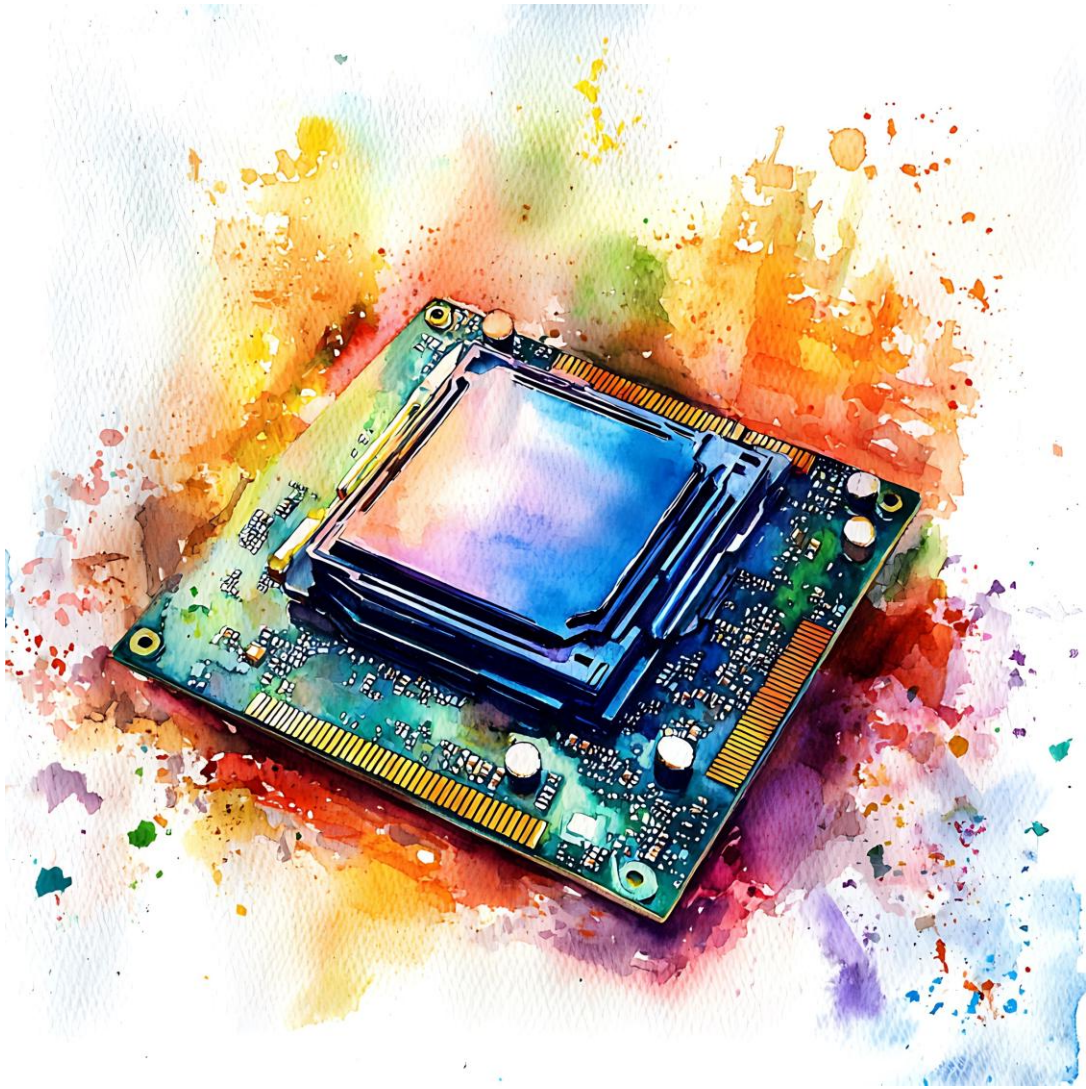
Quick Outline

- CPU performance
- CPU time: Example
- Instruction performance: Example
- CPI: Example
- Classic CPU performance equation
- Performance and power: Example
- Single-cycle CPU



© Woranuch / Adobe Stock

CPU Performance



© Woranuch / Adobe Stock

Time

- **Time** is the measure of computer **performance**
  - A computer that performs the same work in the least amount of time is the fastest and thus, most performant
- CPU execution time is measured in *seconds per program*
  - Program-specific
- **Definition of time**
  - Wall clock time ⇔ response time ⇔ elapsed time
  - Includes overheads (for tasks other than running our program)

CPU Performance

- **CPU execution time** or, simply, **CPU time**
  - The time the CPU spends on our program
  - Ignores overheads, such as the time to read/write to input/output devices (e.g., keyboard, screen, printer), and performing unrelated system tasks

- We measure time in discrete time intervals: **clock cycles**
  - **Synonyms: CPU cycles**, ticks, clock ticks, clock periods, clocks, **cycles**

CPU Performance and Its Factors

- A formula that relates basic metrics to CPU execution time

$$\begin{array}{c}\textbf{CPU time}\\ \text{for a program}\\ \textit{(in seconds)}\end{array} = \begin{array}{c}\textbf{CPU cycles}\\ \text{for a program}\end{array} \times \begin{array}{c}\textbf{Clock cycle}\\ \textit{(in seconds)}\end{array}$$

- Alternatively, as clock rate (i.e., frequency) and cycle are inverses

$$\begin{array}{c}\textbf{CPU time}\\ \text{for a program}\\ \textit{(in seconds)}\end{array} = \frac{\textbf{CPU cycles} \text{ for a program}}{\textbf{Clock rate } \textit{(in Hz)}}$$

CPU Time

Q: Consider CPU **A**, running on a 2 GHz clock. The CPU time to run our program is 10 s. How many CPU cycles does this CPU take to run the program?

A: *Recall:*

$$\textbf{CPU time}_A = \frac{\textbf{CPU cycles}_A}{\textbf{Clock rate}_A \textit{ (in Hz)}}$$

Therefore,

$$10\text{ s} = \frac{\text{CPU cycles}_A}{2 \times 10^9 \frac{\text{cycles}}{\text{second}}}$$

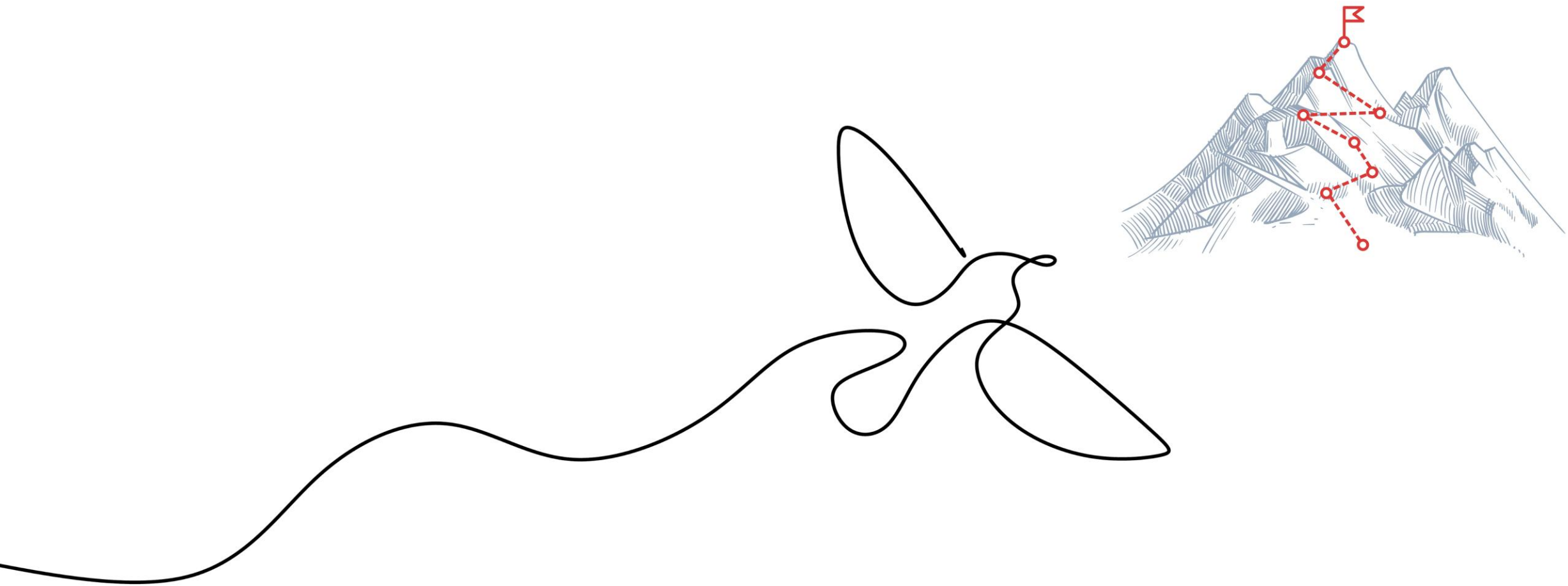
Finally, CPU A takes 20×10^9 cycles to run our program.

Improving Performance

- Let us help a designer build CPU B, which will run the same program in 6 s.
- The computer designer realized that a substantial increase in clock rate is possible. Still, this increase will affect the rest of the CPU, causing CPU B to require 1.2 × the cycles that CPU A takes to run this program.
- **Q:** What clock rate should the designer of CPU B target?
- **A:**

$$\text{CPU time}_B = \frac{1.2 \times \text{CPU cycles}_A}{\text{Clock rate}_B} = \frac{1.2 \times 20 \times 10^9}{\text{Clock rate}_B} = 6\ \text{s}$$

Therefore, the clock rate of CPU B is 4 GHz (double the clock rate of CPU A).



Instruction Performance

Clock Cycles Per Instruction (CPI)

- So far, we have not included any reference to the number of instructions needed for the program. However, CPU time depends on the number of instructions in a program

$$\text{CPU cycles} = \text{Instructions } \textit{for a program} \times \text{Average clock cycles } \textit{per instruction}$$

- **Average clock cycles per instruction** (**CPI**) is the **average** of all the instructions executed by the program
  - Depending on what they do, instructions may take different times

Comparing Code Segments

- Consider three instruction classes, K_1, K_2, and K_3, with CPIs of **1**, **2**, and **3** cycles per instruction, respectively.
- For a particular high-level code segment, the compiler writer is considering two code sequences requiring the following instruction counts:
  - **Seq_A**: **2** instructions from K_1, **1** from K_2, **2** from K_3
  - **Seq_B**: **4** instructions from K_1, **1** from K_2, **1** from K_3
- **Q:** Which of the two code sequences, Seq_A and Seq_B, is faster?
- **Q:** Compare the CPI of Seq_A and Seq_B.

Comparing Code Segments

Solution

- CPU clock cycles for any sequence of *n* instructions
  - CPU clock cycles = $\sum_{i=1}^{n} (\text{CPI}_i \times \text{Count}_i)$
  - Count_i = the number of occurrences of the corresponding instruction

- CPU cycles Seq_A = (**2** × 1) + (**1** × 2) + (**2** × 3) = 2+2+6 = 10 cycles
- CPU cycles Seq_B = (**4** × 1) + (**1** × 2) + (**1** × 3) = 4+2+3 = 9 cycles

- Therefore, code sequence Seq_B is faster; it takes one cycle less

Comparing Code Segments

Solution, Contd.

- CPI of a sequence is the average CPI of all of the corresponding instructions
- $\text{CPI} = \dfrac{\text{CPU cycles}}{\text{Instruction count}}$
- $\text{CPI Seq}_A = \dfrac{\text{CPU cycles Seq}_A}{\text{Instruction count Seq}_A} = 10 / 5 = 2.0 \text{ cycles-per-instr}$
- $\text{CPI Seq}_B = \dfrac{\text{CPU cycles Seq}_B}{\text{Instruction count Seq}_B} = 9 / 6 = 1.5 \text{ cycles-per-instr}$

Recall: Instruction Performance

Clock Cycles Per Instruction (CPI)

$$\text{CPU cycles} = \text{Instructions } \textit{for a program} \times \text{Average clock cycles } \textit{per instruction}$$

EXAMPLES

Applying Performance Equation

- Suppose we have two CPU implementations of the same instruction set architecture (ISA).
- CPU A has a clock cycle time of 250 ps and a CPI of 2.0 for some program, and CPU B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program.
- **Q:** Which CPU is faster for this program and by how much?

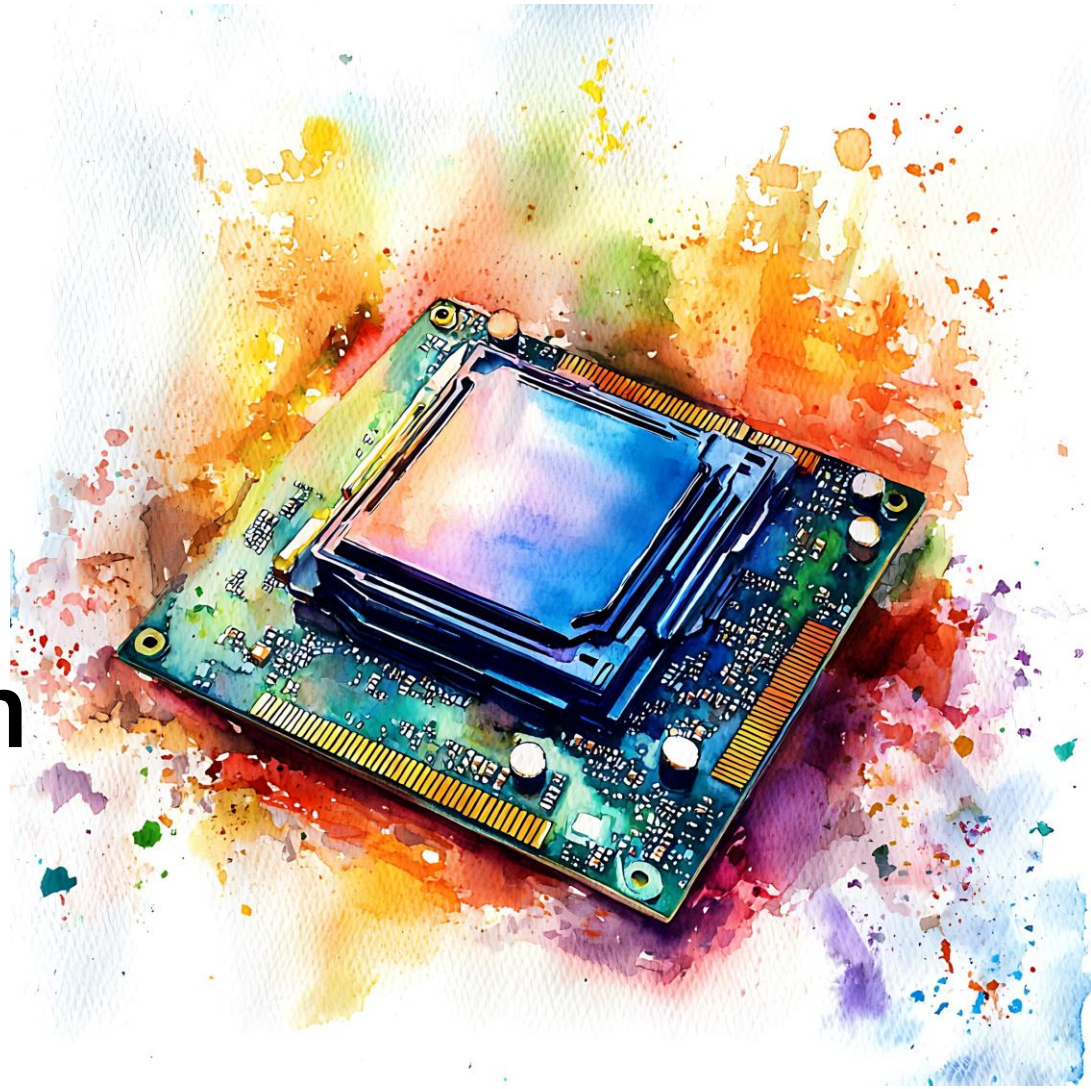
Note: 1 ps = 1 picosecond = 10^{-12} s

Applying Performance Equation
Solution

EXAMPLES

- Each CPU executes the same number of instr. for the program, ***I***
- Number of CPU cycles:
  - CPU clock cycles$_A$ = $I_A \times 2.0 = I \times 2.0$
  - CPU clock cycles$_B$ = $I_B \times 1.2 = I \times 1.2$
- Given the cycles, compute CPU time for both
  - CPU time$_A$ = CPU cycles$_A$ × Clock cycle$_A$ = $I \times 2.0 \times 250$ ps = $500 \times I$ ps
  - CPU time$_B$ = CPU cycles$_B$ × Clock cycle$_B$ = $I \times 1.2 \times 500$ ps = $600 \times I$ ps
- Finally, CPU A is 1.2× faster—takes less time—than CPU B
  - CPU time$_B$ / CPU time$_A$ = 600 / 500 = 1.2

CPU Performance Equation



© Woranuch / Adobe Stock

The Classic CPU Performance Equation

- We can express CPU performance in terms of
  - **Instruction count** (number of instructions executed by the program),
  - Average clock cycles per instruction (**CPI**), and
  - Clock cycle time

$$\underbrace{\text{CPU time for a program } \textit{(in seconds)}}_{\text{CPU time}} = \underbrace{\text{Instruction count for a program}}_{\text{Instruction count}} \times \underbrace{\text{CPI}}_{\text{CPI}} \times \underbrace{\textit{Clock cycle (in seconds)}}_{\text{Clock cycle}}$$

Factors Impacting Program Performance

Algorithm

$$\underbrace{\text{CPU time for a program (in seconds)}}_{\text{CPU time}} = \underbrace{\text{Instruction count for a program}}_{\text{Instruction count}} \times \underbrace{\text{CPI}}_{\text{CPI}} \times \underbrace{\text{Clock cycle (in seconds)}}_{\text{Clock cycle}}$$

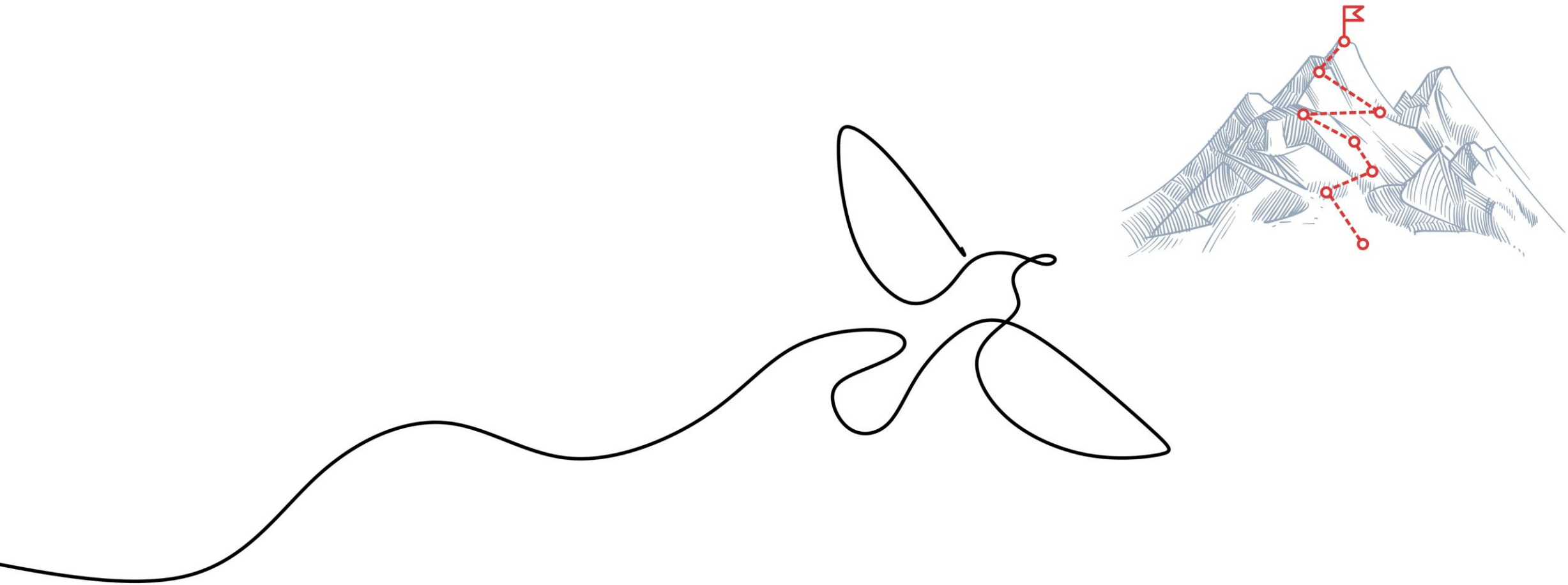
- **Q:** How does an algorithm impact program performance?
- **A:** It determines the number of instructions executed. It may affect CPI as well, favoring slower or faster instructions.

Factors Impacting Program Performance

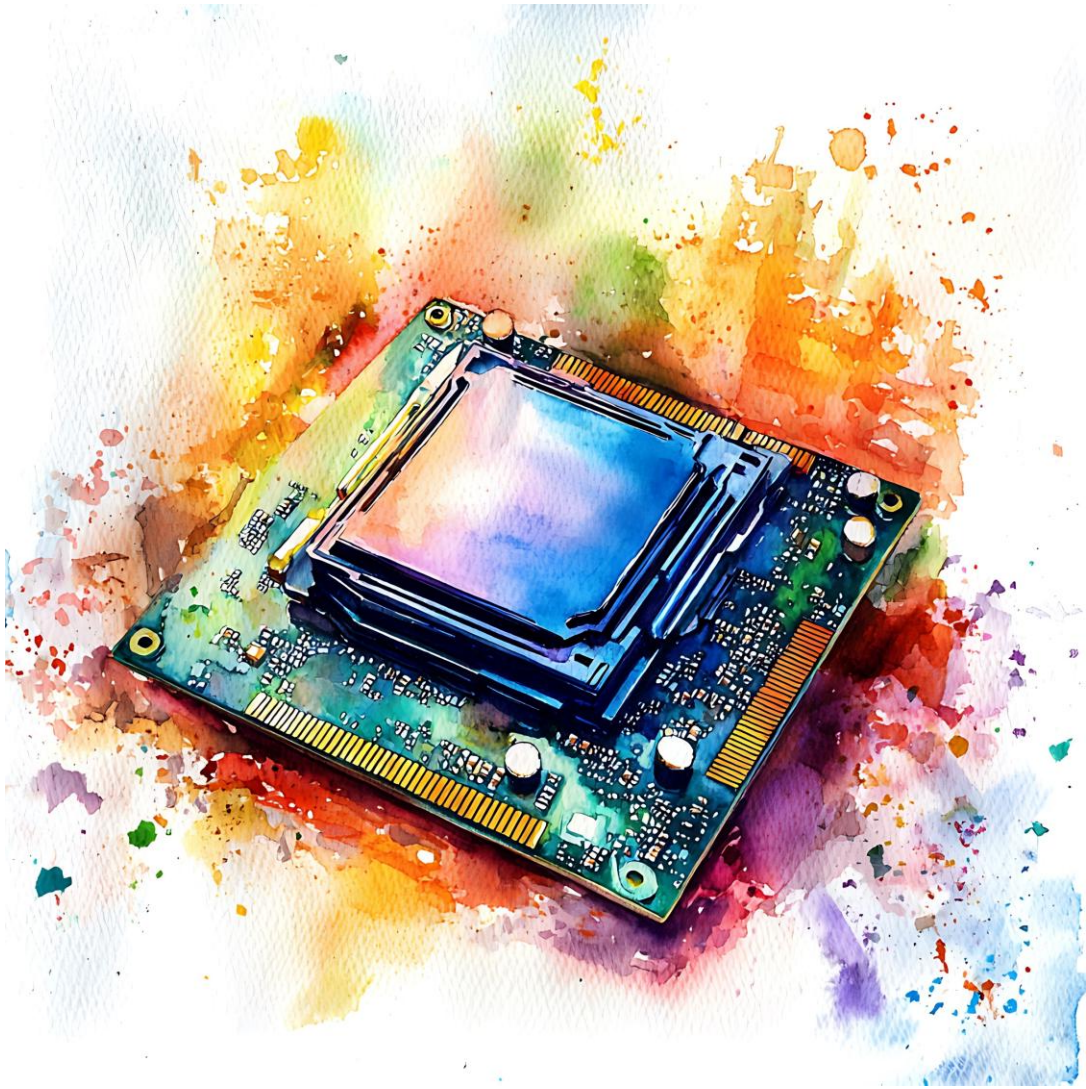
ISA

$$\underbrace{\text{CPU time for a program (in seconds)}}_{\text{CPU time}} = \underbrace{\text{Instruction count for a program}}_{\text{Instruction count}} \times \underbrace{\text{CPI}}_{\text{CPI}} \times \underbrace{\text{Clock cycle (in seconds)}}_{\text{Clock cycle}}$$

- **Q:** How does ISA impact program performance?
- **A:** It affects all three aspects of CPU performance:
  the instructions needed to perform the required function,
  the cost in cycles of each instruction, and the CPU clock rate.



Performance and Power



© Woranuch / Adobe Stock

Performance and Power

- An increase in clock rate brings improvement in performance, but it also increases power dissipation

- *Recall:* Power dissipation in CMOS logic gates, as a function of switching frequency, capacitive load, and power supply

$$P_D = fCV^2$$

Relative Dynamic Power

- Suppose we developed a new, simpler CPU_{NEW} that has **85%** of the capacitive load of the more complex older CPU_{OLD}.
- Further, assume that it can adjust the supply voltage so that it can reduce it by **25%** compared to CPU_{OLD}, which results in a **10**% shrink in clock frequency.
- **Q:** What is the impact on dynamic power?

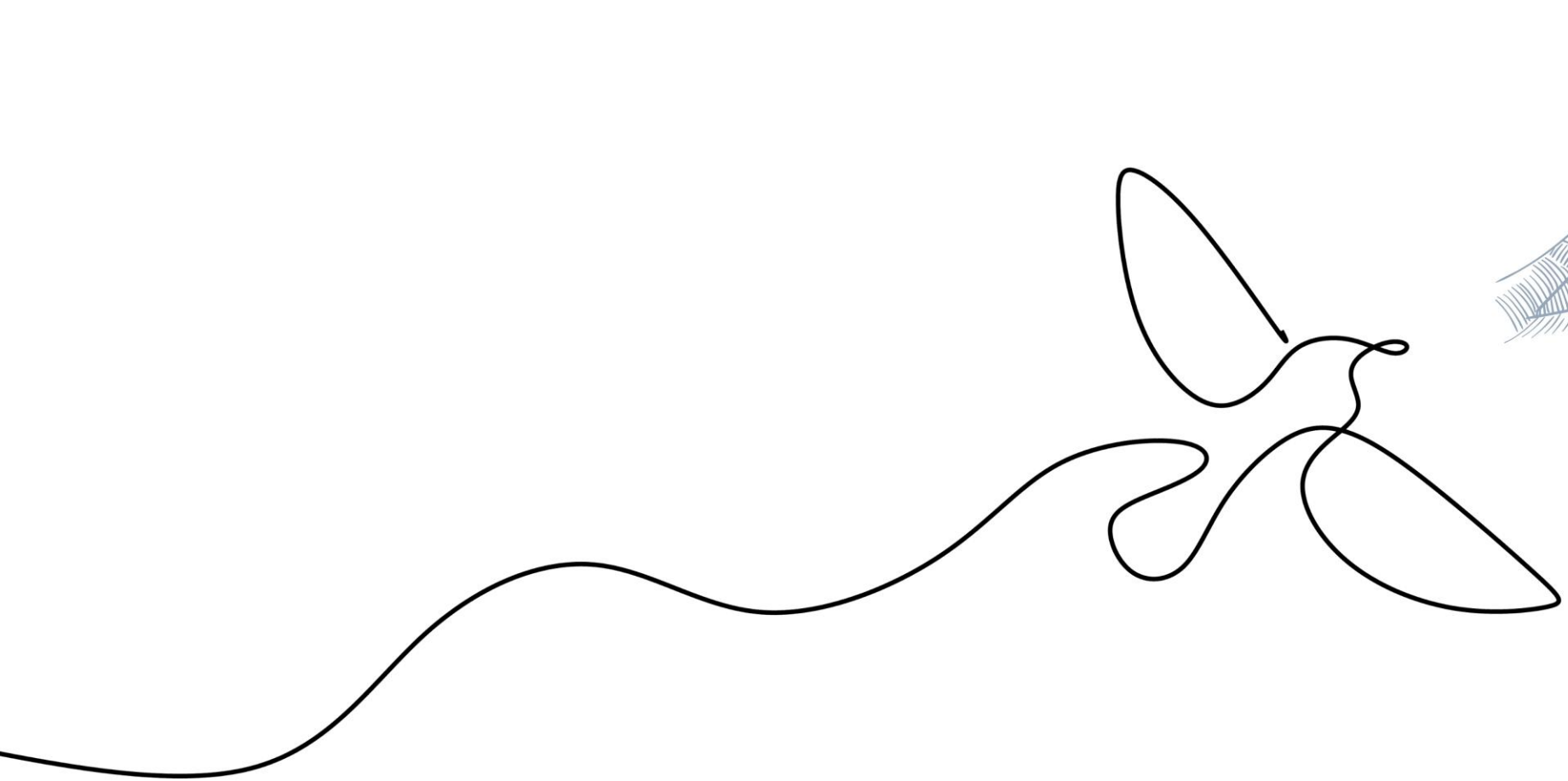
Relative Dynamic Power

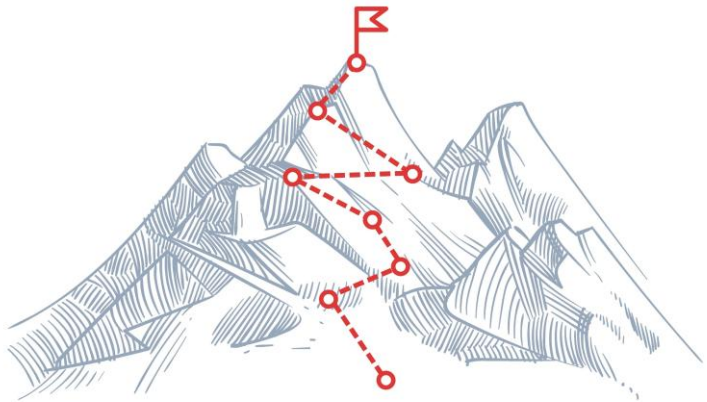
Solution

$$P_D = fCV^2$$

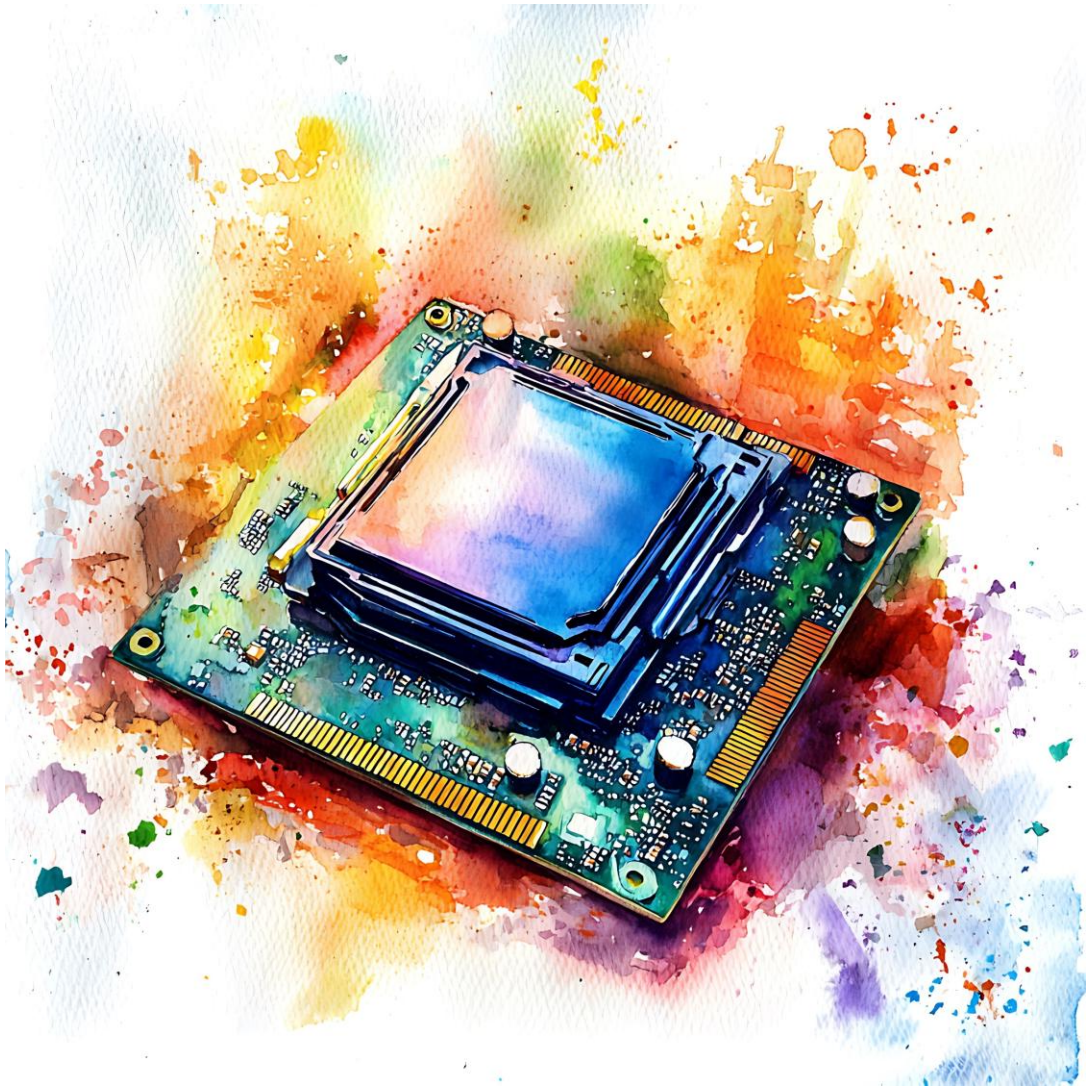
- $\text{Power}_{\text{OLD}} = f \times C \times V^2$
- $\text{Power}_{\text{NEW}} = (f \times 0.9) \times (C \times 0.85) \times (V \times 0.75)^2$

- Power ratio becomes
  - $\text{Power}_{\text{NEW}} / \text{Power}_{\text{OLD}} = 0.9 \times 0.85 \times 0.75^2 = 0.43$
- Therefore, the new CPU consumes less than half the power





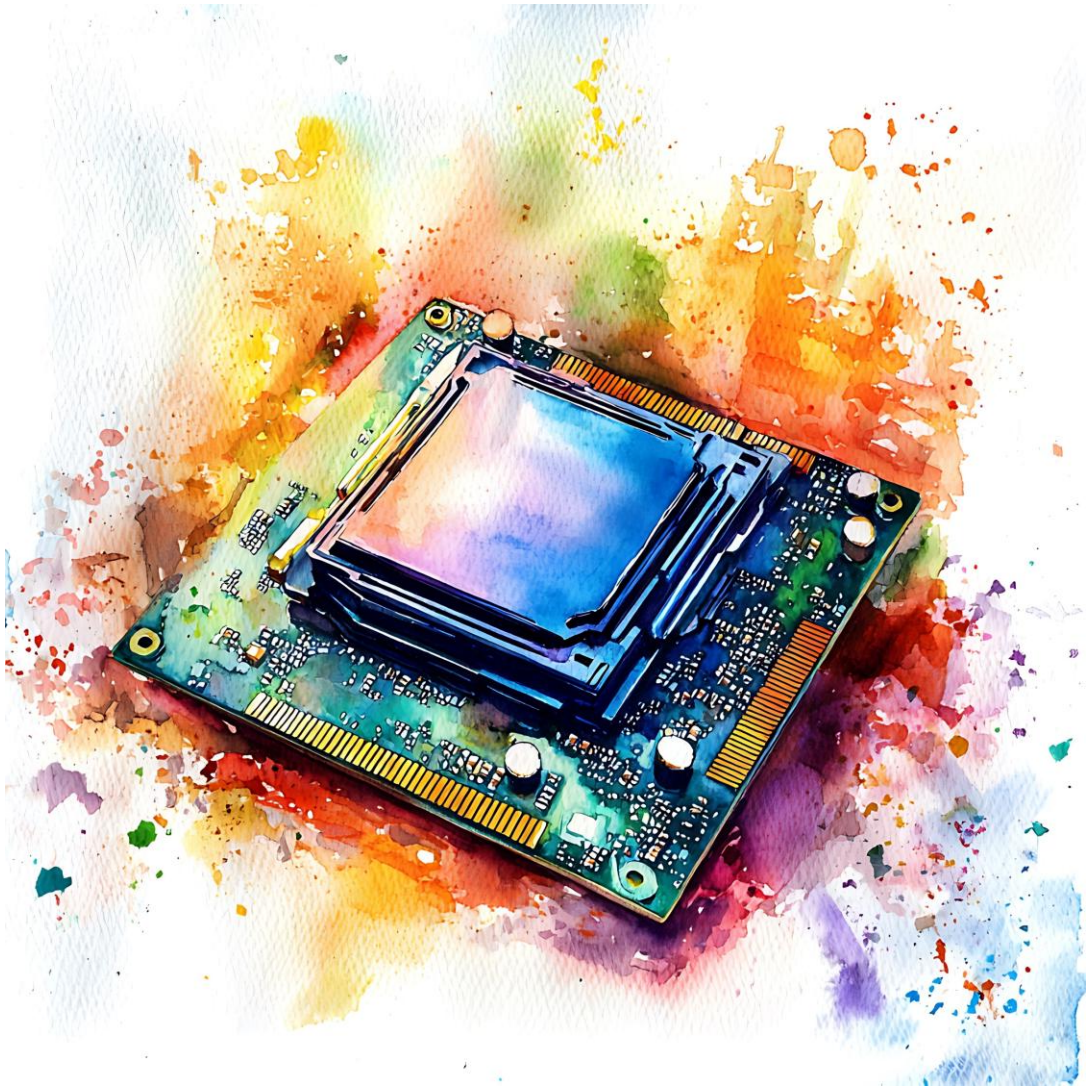
Single-Cycle CPU



© Woranuch / Adobe Stock

Outline

- Single-cycle CPU
  - Instruction memory
  - Register file
  - ALU
  - Load and store
  - Branch support
  - Completing the datapath
  - Control unit
- Single-cycle vs. multicycle CPU



Single-Cycle CPU

- In a single-cycle CPU, all operations required by an instruction are performed within one clock cycle (CPI = 1.0)
- In contrast, a multi-cycle CPU has a CPI greater than 1, because instruction execution is broken into multiple clock cycles

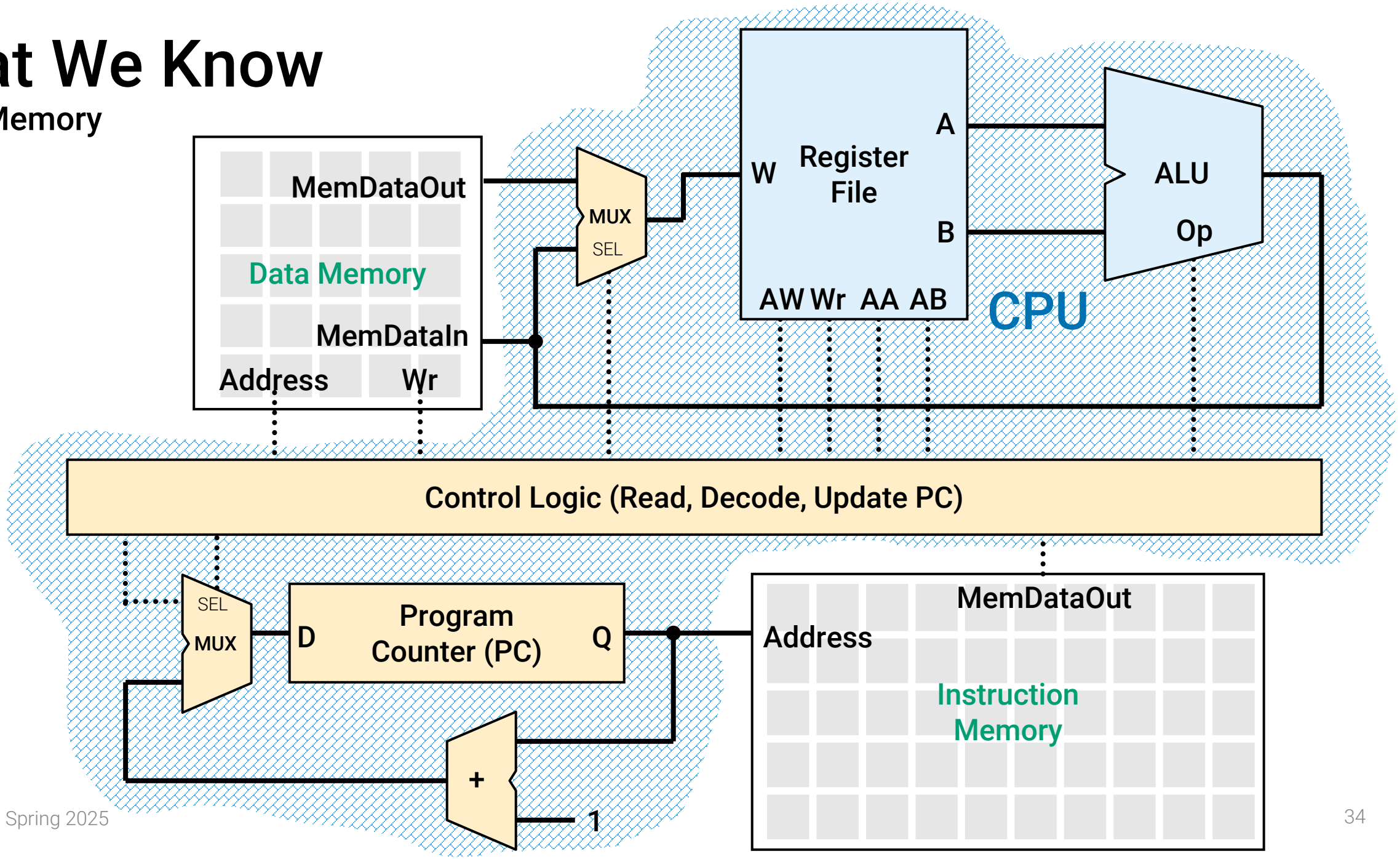
- Let us build a simple single-cycle CPU…

A Simple Single-Cycle CPU Implementation

- Let us build a simple CPU supporting the following **subset** of RISC-V instructions for simplicity
  - **R-type arithmetic-logical instructions**
    - add, sub, and, or
  - **Memory instructions**
    - load and store word
  - **Control flow**
    - branch if equal

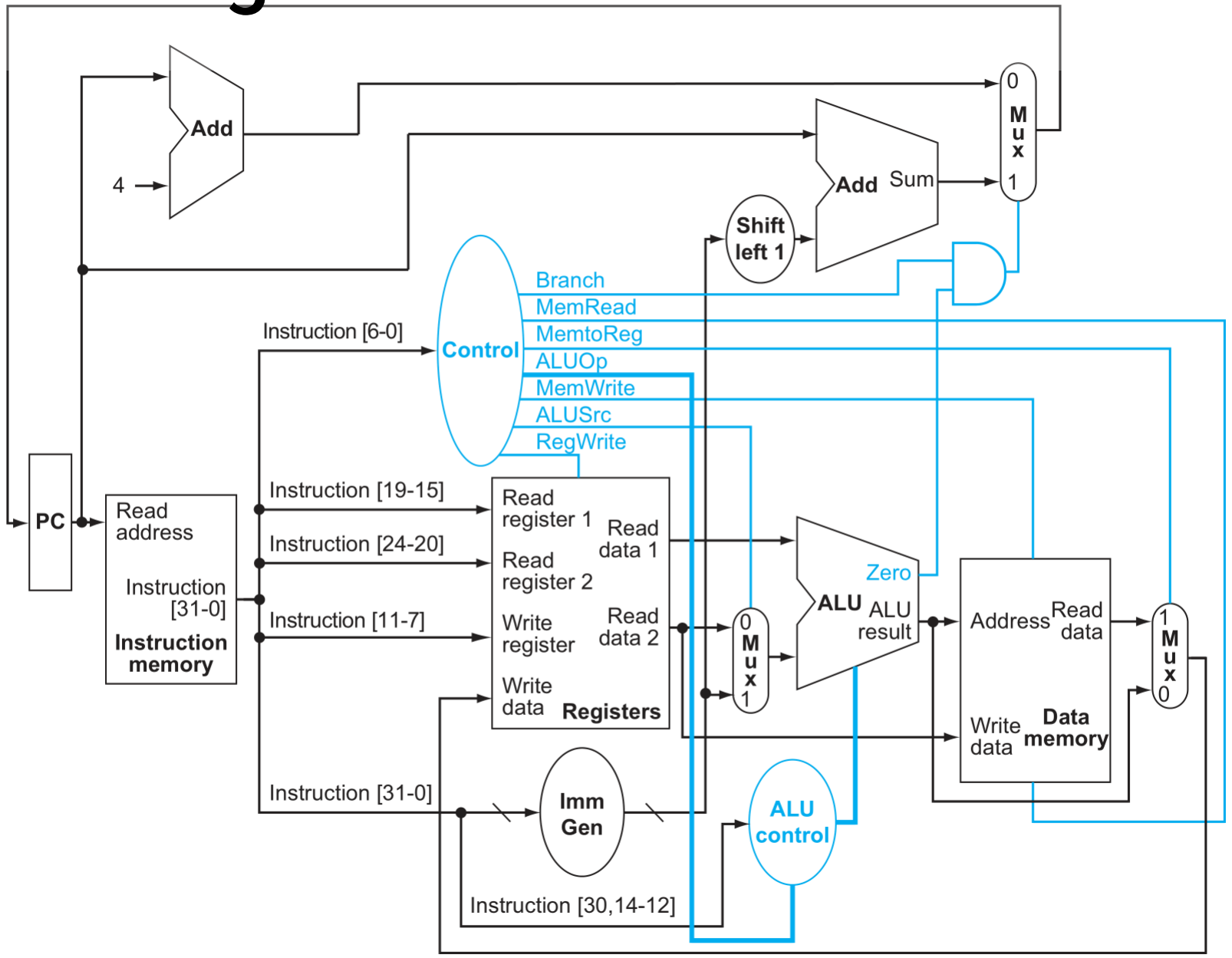
What We Know

CPU + Memory



Where We're Heading

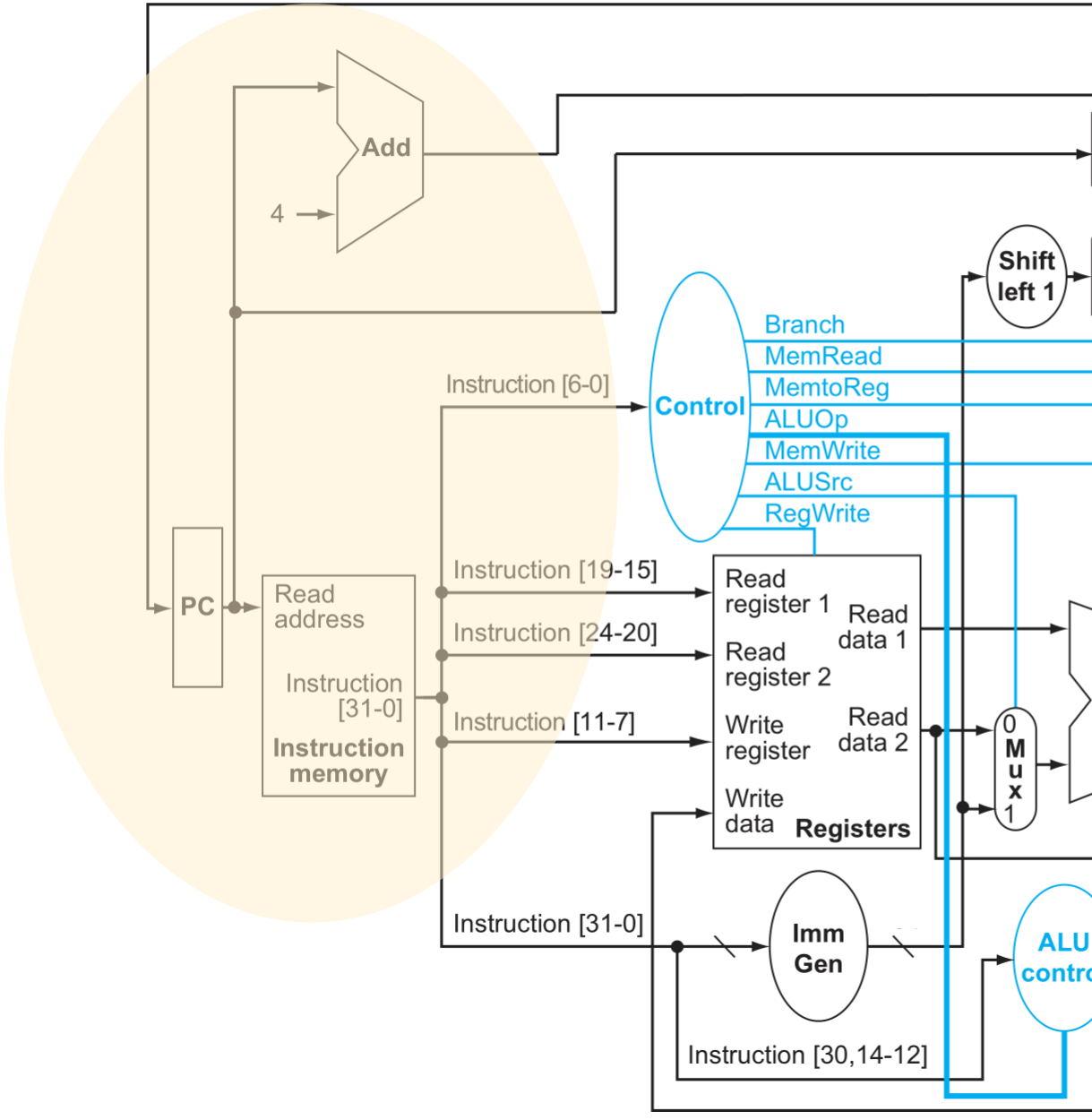
CPU + Memory



Single-Cycle CPU

Outline

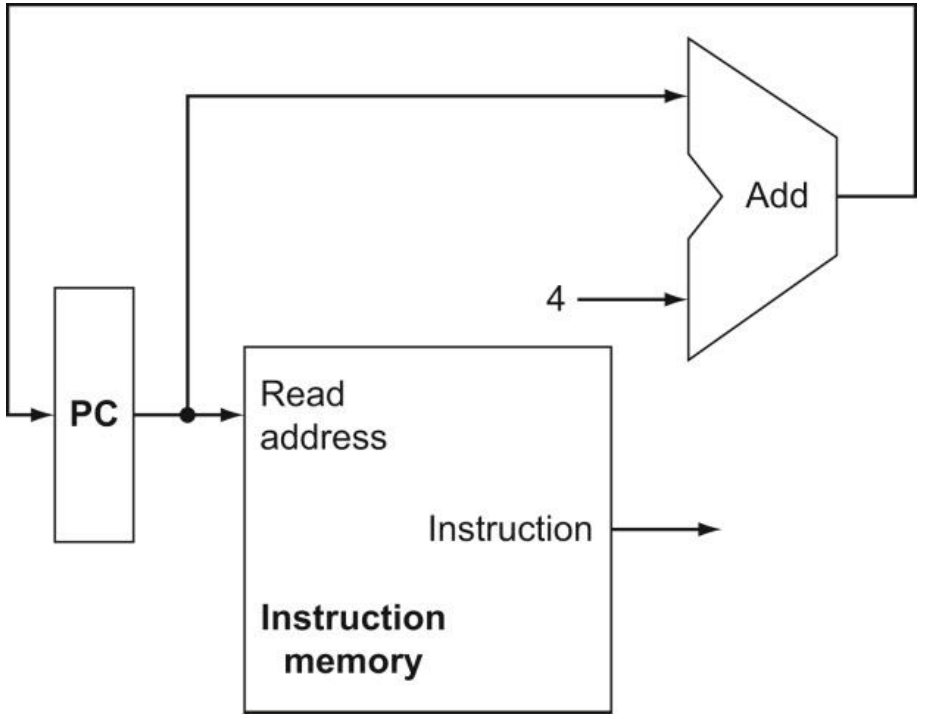
- **Instruction memory**
- General-purpose registers
- ALU
- Data memory load and store
- Branch support
- Completing the datapath
- Control signals



CPU's Elements For Instructions

Store and Access Instructions of the Program

- **Instruction memory:** An external memory unit to store the program (i.e., the instructions) and supply instructions when given an address
  - Much higher capacity than the register file
  - For simplicity, we shall treat it as read-only
  - The memory output at any time reflects the contents at the address specified; no read-control signal is needed

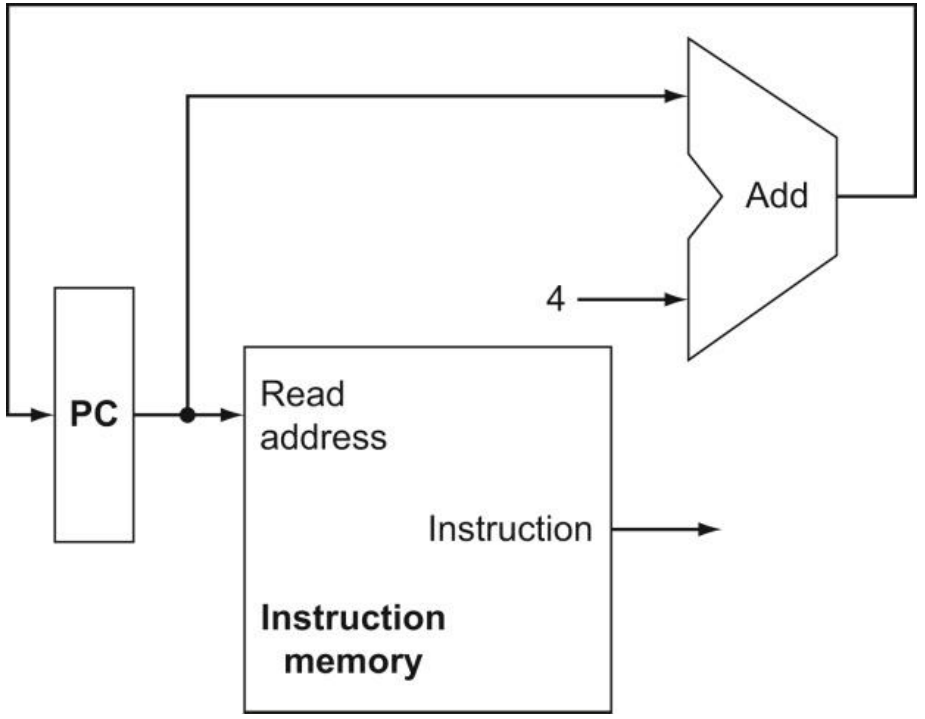


Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

CPU's Elements For Instructions

Store and Access Instructions of the Program, Contd.

- **Instruction memory:** An external memory unit to store the program
  - Byte addressable
  - **[32 b]** Address input
  - **[32 b]** Data (instruction) output
- **Program counter (PC):** a 32-bit register that holds the address of the current instruction
- An **adder** to **increment** the PC by four (to the address of the **next** instruction)

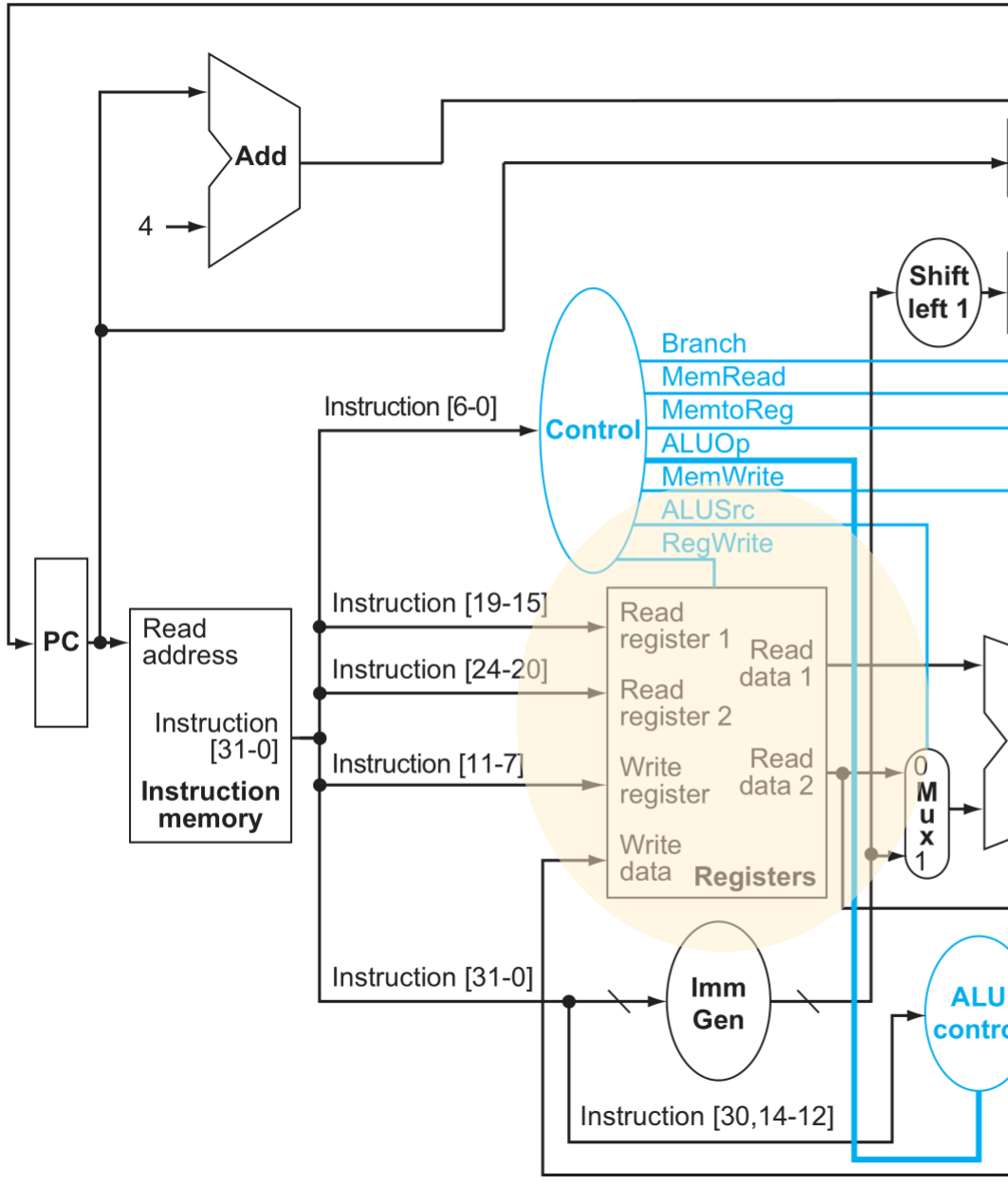


Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

Single-Cycle CPU

Outline

- Instruction memory
- **General-purpose registers**
- ALU
- Data memory
  load and store
- Branch support
- Completing the datapath
- Control signals



CPU's General Purpose Registers

Register File

- **32** general-purpose registers are grouped into a **register file**
- Registers are **32-bit** wide
- Registers can be read or written by specifying the **index** (number, address) of the register

Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

CPU's General Purpose Registers

Register File, Contd.

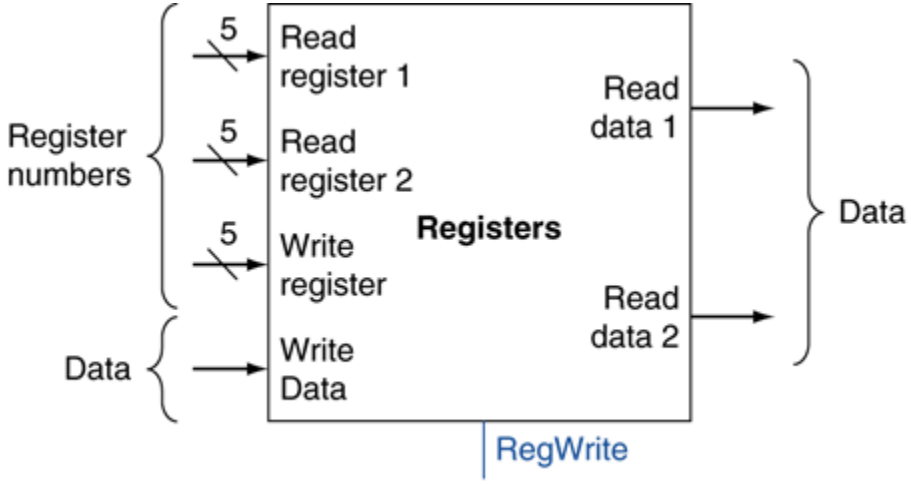
- Registers hold operands for the arithmetic and logic instructions and are the destination for their result
- *Recall*: R-format instructions take two register operands and write the result in the third, destination register
  - Therefore, the register file must permit reading two registers and writing to one in the same clock cycle
  - Asynchronous read

Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

CPU's General Purpose Registers

Register File, Contd.

- Registers hold operands for the arithmetic and logic instructions and are the destination for their result
- *Recall*: R-format instructions take two register operands and write the result in the third, destination register
  - Therefore, the register file must permit reading two registers and writing to one in the same clock cycle
  - Asynchronous read



Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

CPU's General Purpose Registers

Register File, Contd.

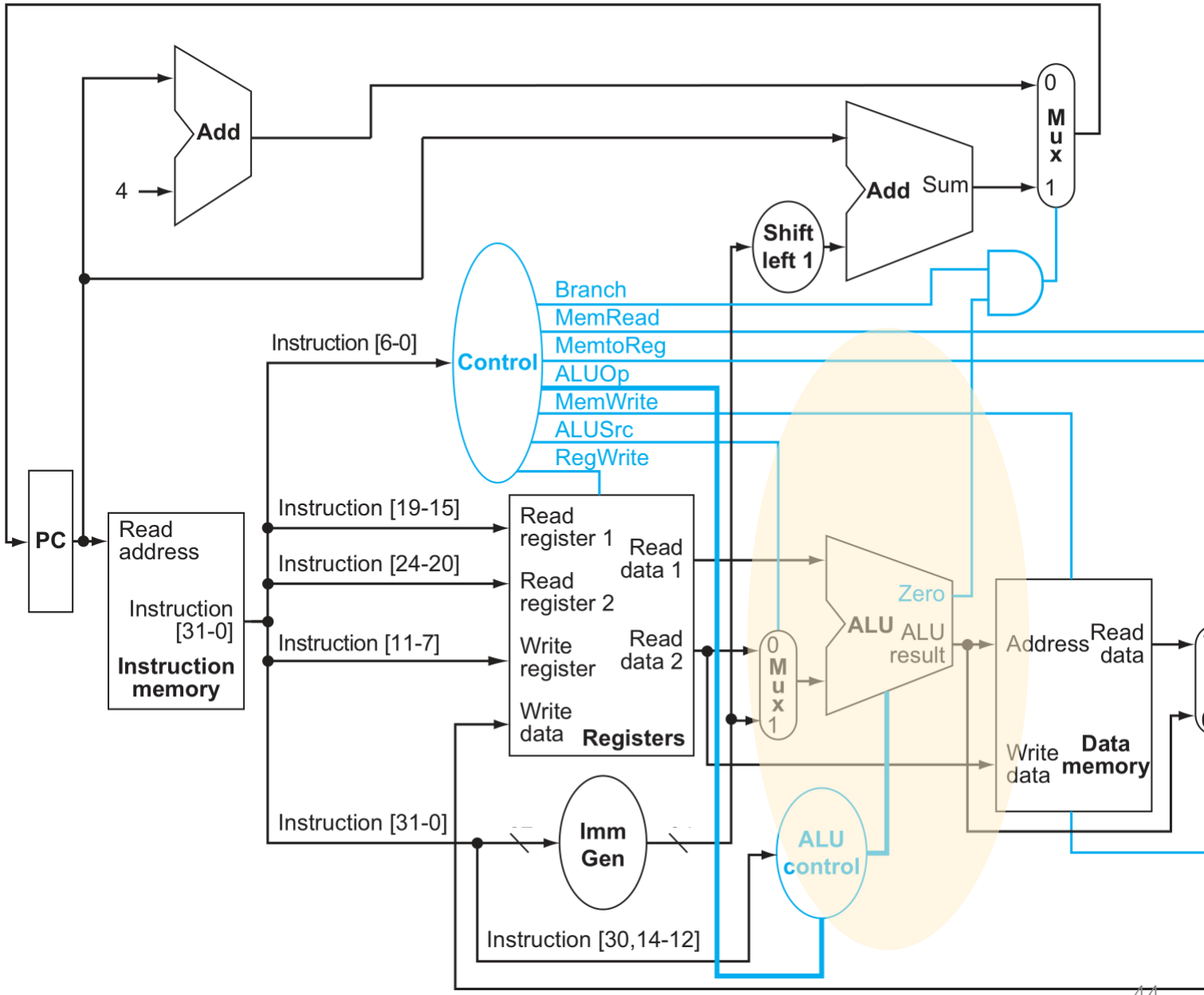
- For each data word to be read, the register file needs
  - **[5 b] Read register:** An input specifying the index of the register to be read
  - **[32 b] Read data:** An output carrying the value that has been read
- To write a data word, the RF needs
  - **[5 b] Write register:** An input specifying the index of the register to be written to
  - **[32 b] Write data:** Data to be written
  - **[1 b] RegWrite:** A write control signal, which must be asserted for a write to occur at the clock edge

Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

Single-Cycle CPU

Outline

- Instruction memory
- General-purpose registers
- **ALU**
- Data memory load and store
- Branch support
- Completing the datapath



CPU's Arithmetic Logic Unit

ALU

- Arithmetic logic unit (**ALU**)
- Takes **two** 32-bit inputs (operands)
- Produces a 32-bit result, along with some 1-bit signals (status "flags")
  - For example, the **Zero** flag is asserted if the result of the ALU operation is zero
- **[4 b] ALU operation:** Control signals determine (select) the operation performed by the ALU

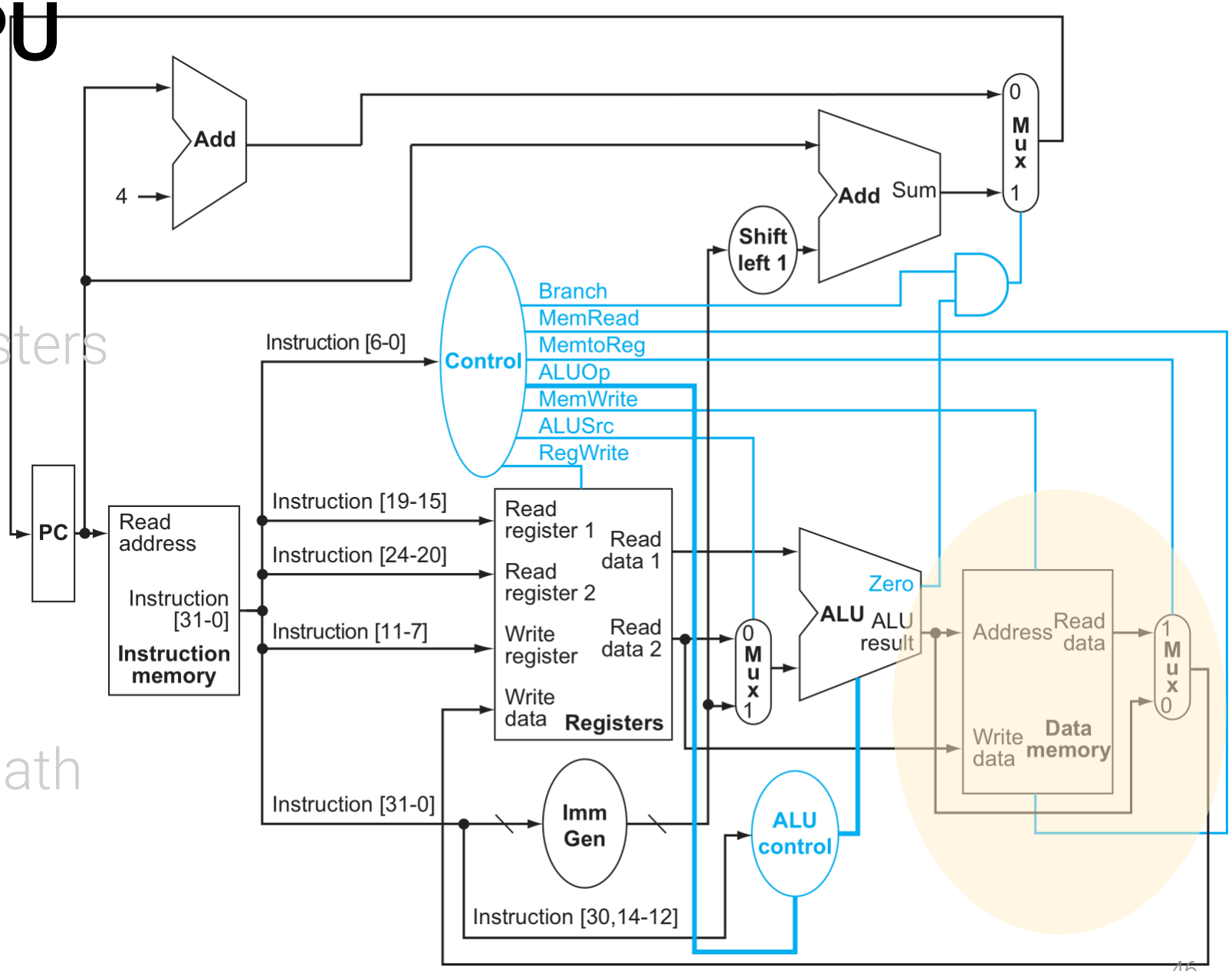
In our simplified CPU implementation, as only a few operations are to be supported, four ALU operation bits are sufficient; for detailed implementation, refer to the literature

Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

Single-Cycle CPU

Outline

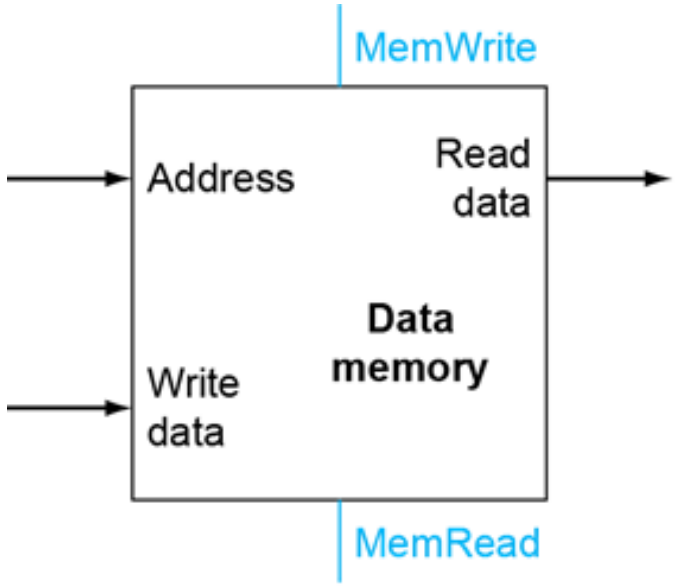
- Instruction memory
- General-purpose registers
- ALU
- **Data memory load and store**
- Branch support
- Completing the datapath
- Control signals



CPU's Units for Data Memory Load and Store

Data Memory

- Data memory is the external memory of much higher capacity than the register file
  - *Recall*: Register file capacity = = 32 × 4 B = 128 B
  - *Recall*: Memory capacity = = 2^{32} B = 4 GiB
- When needed, data from memory is transferred to the register file, and vice versa
- Reading and writing to memories is slower than accessing registers in the register file *(longer wires, more capacitance, more delay…)*

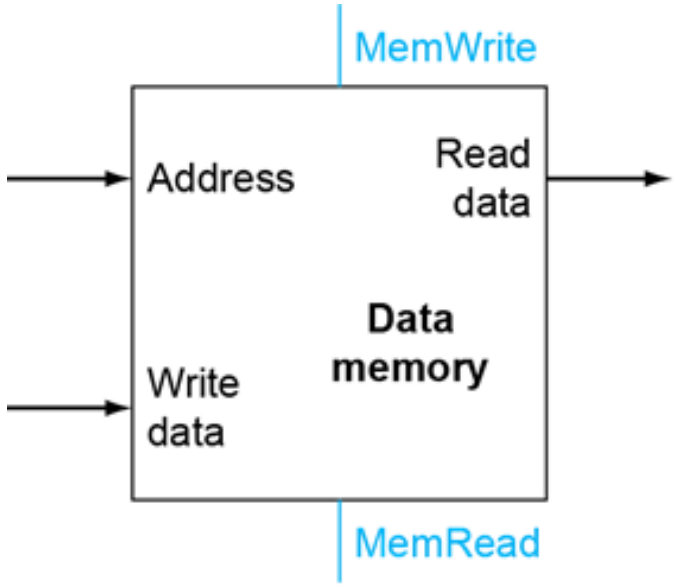


Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

CPU's Units for Data Memory Load and Store

Data Memory Interfaces

- Data memory interfaces
  - **[32 b] Address** input
  - **[32 b] Write data** input
  - **[32 b] Read data** output
  - [1 b] **MemRead:** Control signal for reading
  - [1 b] **MemWrite:** Control signal for writing

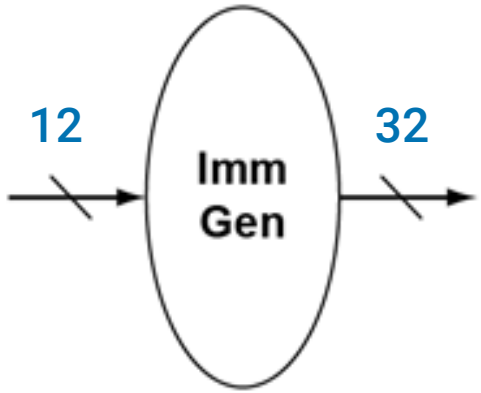


Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

CPU's Units for Data Memory Load and Store

Data Memory Addressing

- *Recall* the general format of the load and store instructions:
  - `lw rd,  offset(rs1)`
  - `sw rs2, offset(rs1)`
- Memory address is computed by adding the base register (**rs1**) with the **sign-extended 12-bit offset** (immediate field in the instruction word)
  - An **imm**ediate **gen**eration unit performs a sign extension



Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

CPU's Units for Data Memory Load and Store

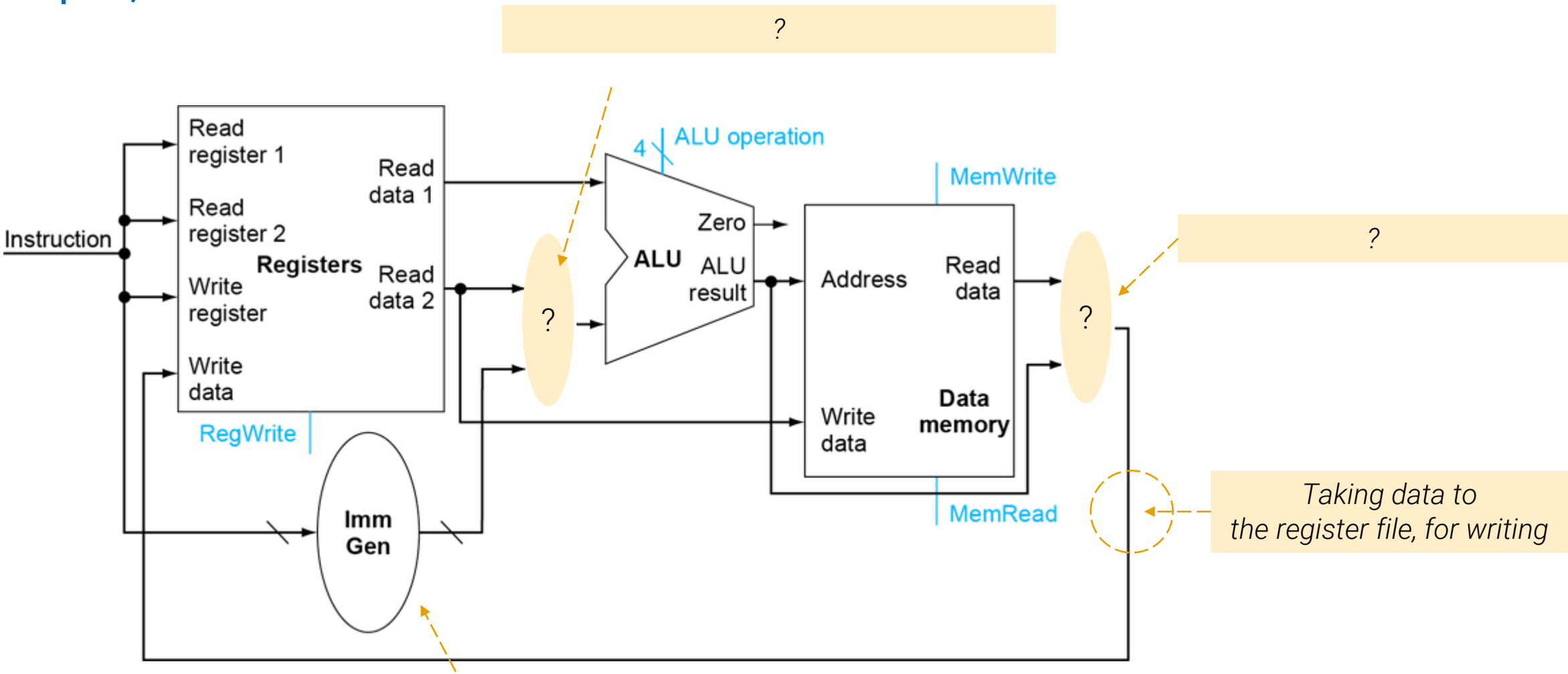
Link with the Register File

- **Load from the memory**: a value must be read from memory before it is **transferred to a register** in the register file
- **Store to the memory**: value must be **read from a register** in the register file before it is transferred to the memory
- Therefore, the register file is part of the CPU datapath for the memory access instructions

Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

Memory Load/Store and R-type Instructions

Datapath, Annotated

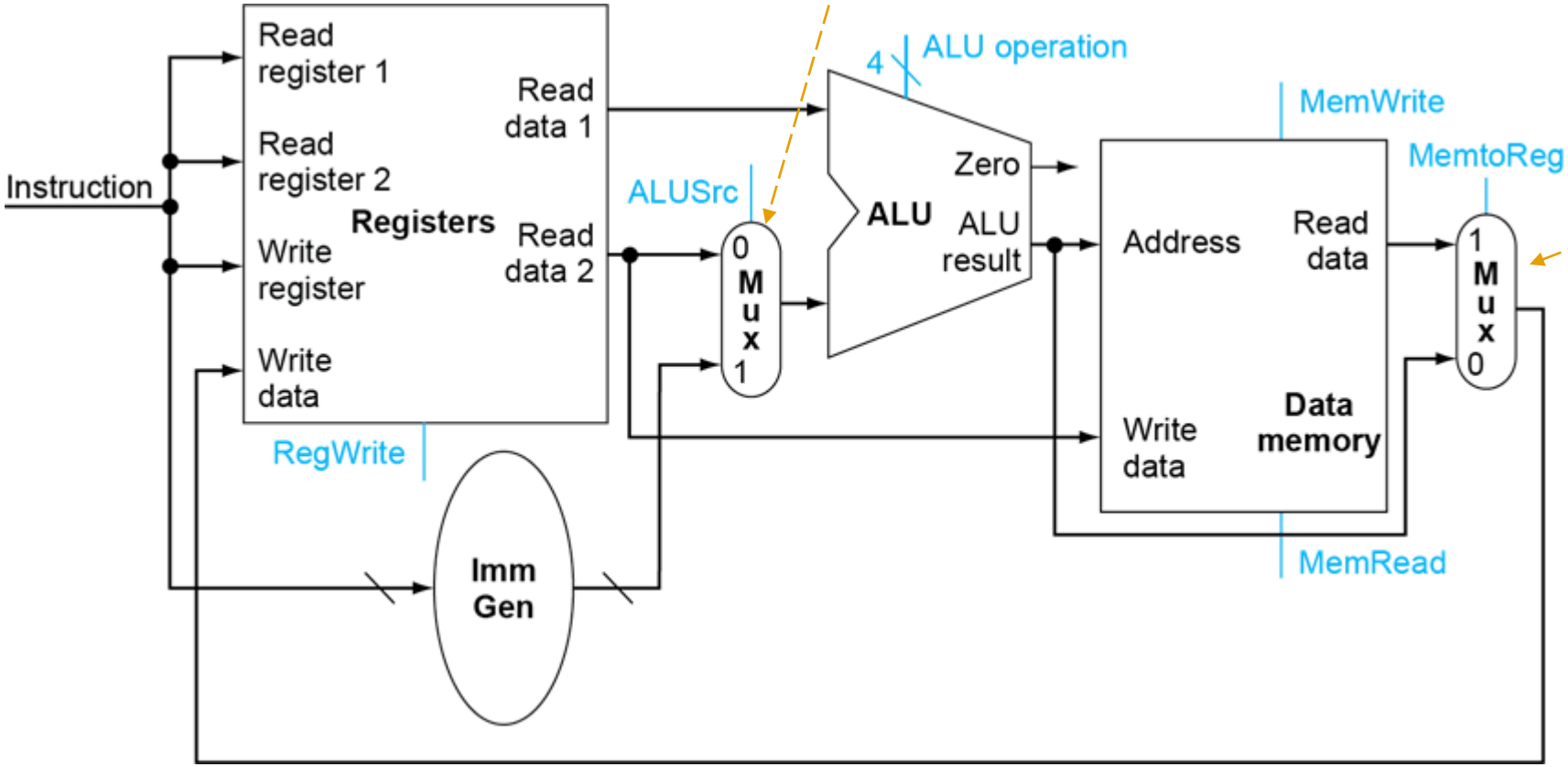


Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

Memory Load/Store and R-type Instructions

Datapath, Annotated

ALU operand: Selecting between a register and a sign-extended immediate (for address comp.)



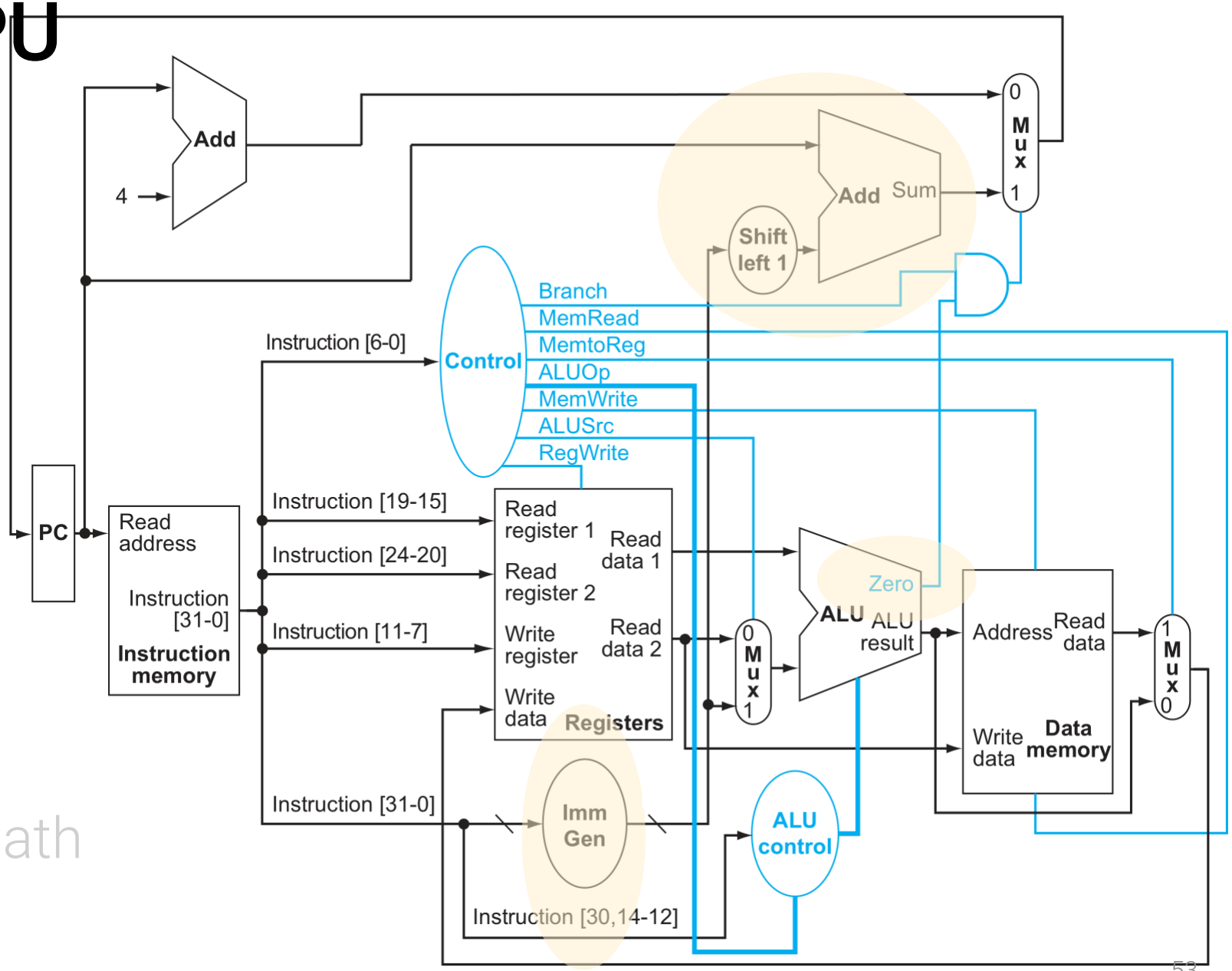
Selecting between the data from the memory or the ALU output

Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

Single-Cycle CPU

Outline

- Instruction memory
- General-purpose registers
- ALU
- Data memory load and store
- **Branch support**
- Completing the datapath
- Control signals



Datapath, Contd.

Branch Instruction Support

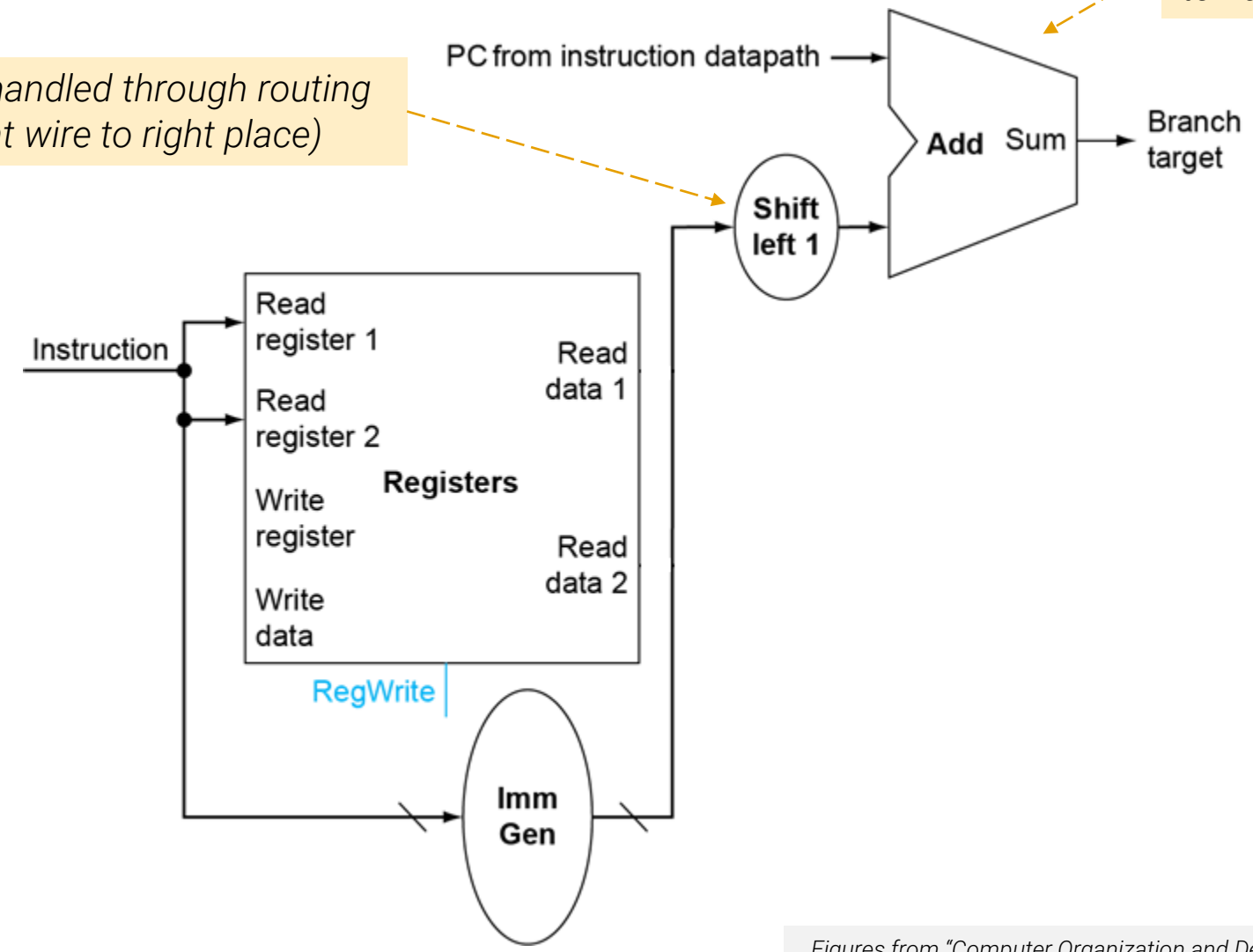
- *Recall:* branch if equal instruction
  - `beq  rs1, rs2, imm`
  - The immediate field is multiplied by 2 (i.e., shift left by 1 bit) to compute the offset
  - if `rs1` and `rs2` are equal, PC = **branch target address** = PC + sign-extended offset
- The base register for the memory address computation is the PC
- One of the following two cases determines the next instruction address:
  - **Branch taken**: when the two operand registers are equal
    - **PC = PC + sext(imm × 2)**
  - **Branch not taken**: when the two operand registers are **not** equal
    - **PC = PC + 4**

Branch if Equal

Datapath, Annotated

Compute PC + sext(immediate) × 2 to have it ready if the branch is taken

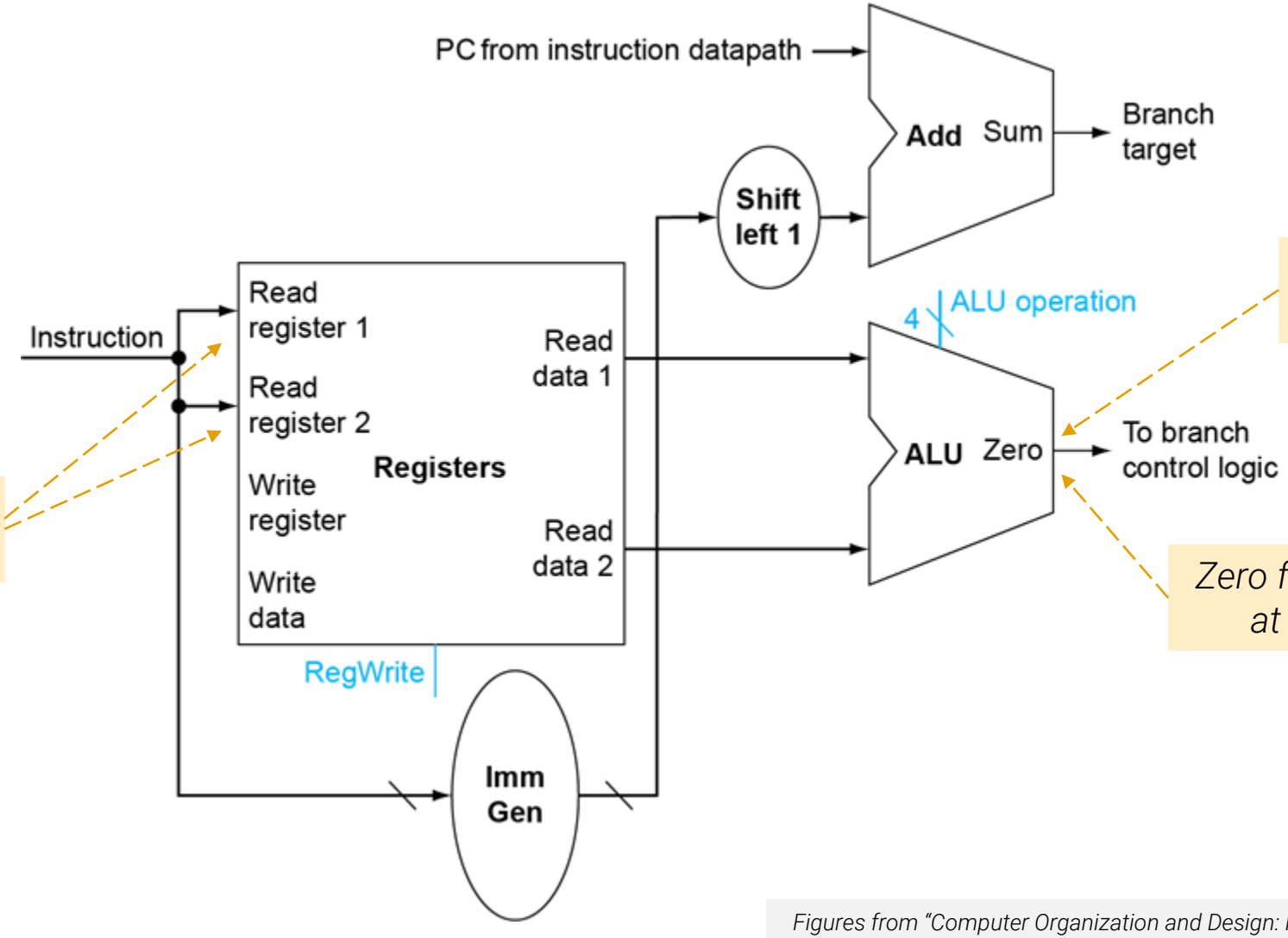
No gates required, handled through routing (bringing the right wire to right place)



Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

Branch if Equal

Datapath, Annotated



Indices of the registers to be read and compared

Compare two registers for equality

Zero flag set if two registers at the input are equal

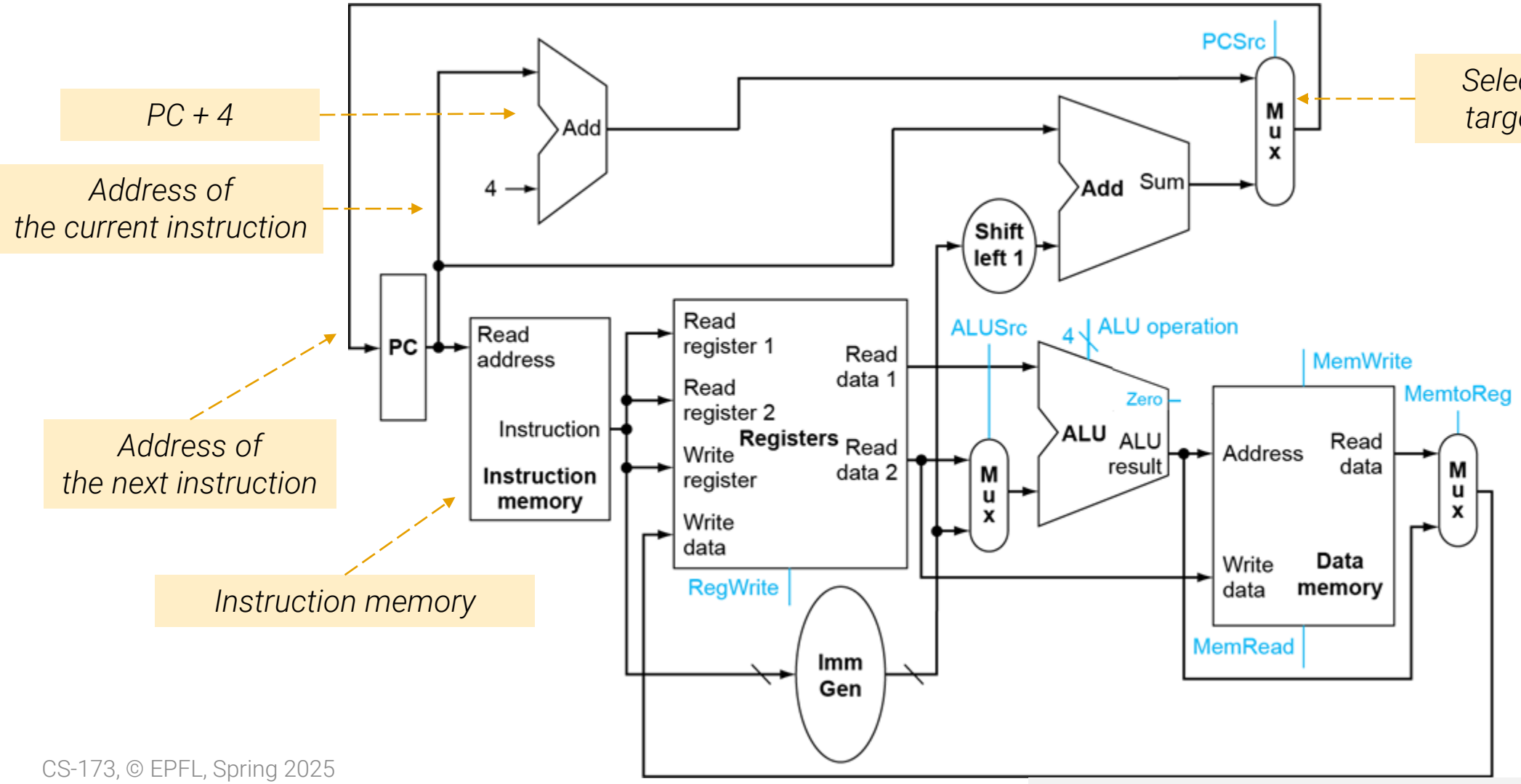
Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

Single-Cycle CPU
Outline

- Instruction memory
- General-purpose registers
- ALU
- Data memory load and store
- Branch support
- **Completing the datapath**
- Control signals

Completing the Datapath

Memory Load/Store, R-type ALU operations, Branch If Equal

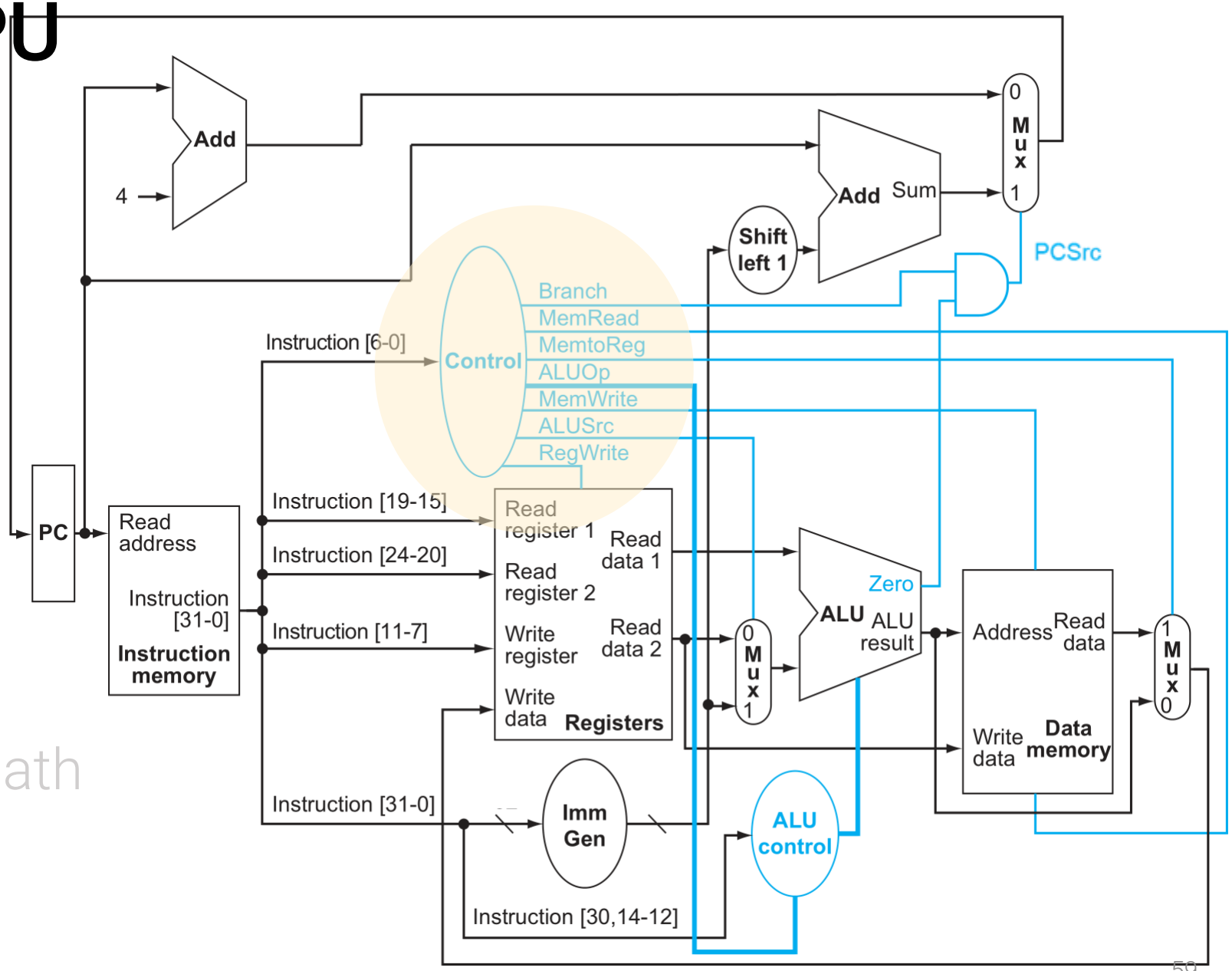


Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

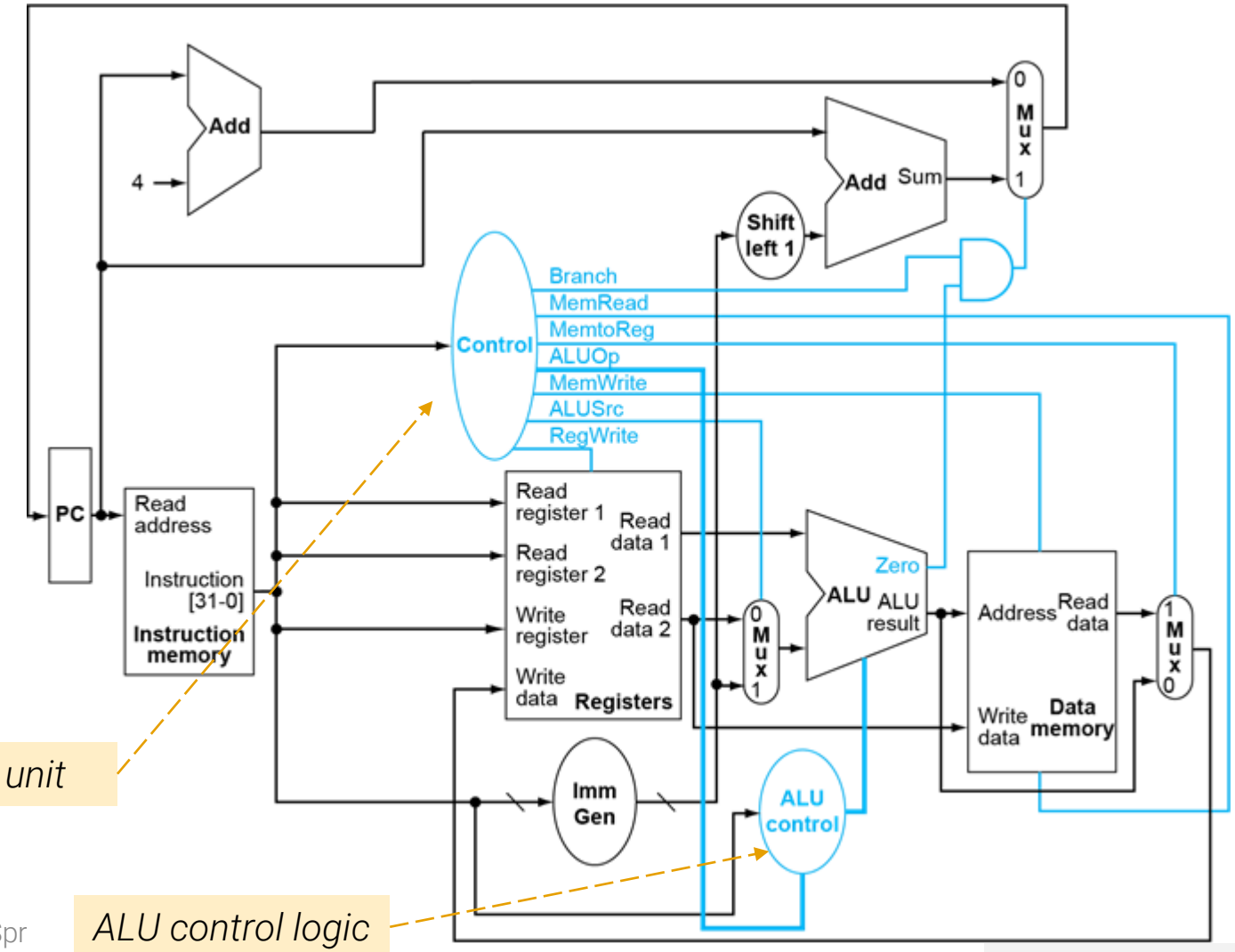
Single-Cycle CPU

Outline

- Instruction memory
- General-purpose registers
- ALU
- Data memory
- Branch support
- Completing the datapath
- **Control signals**



The End: Adding the Control Unit

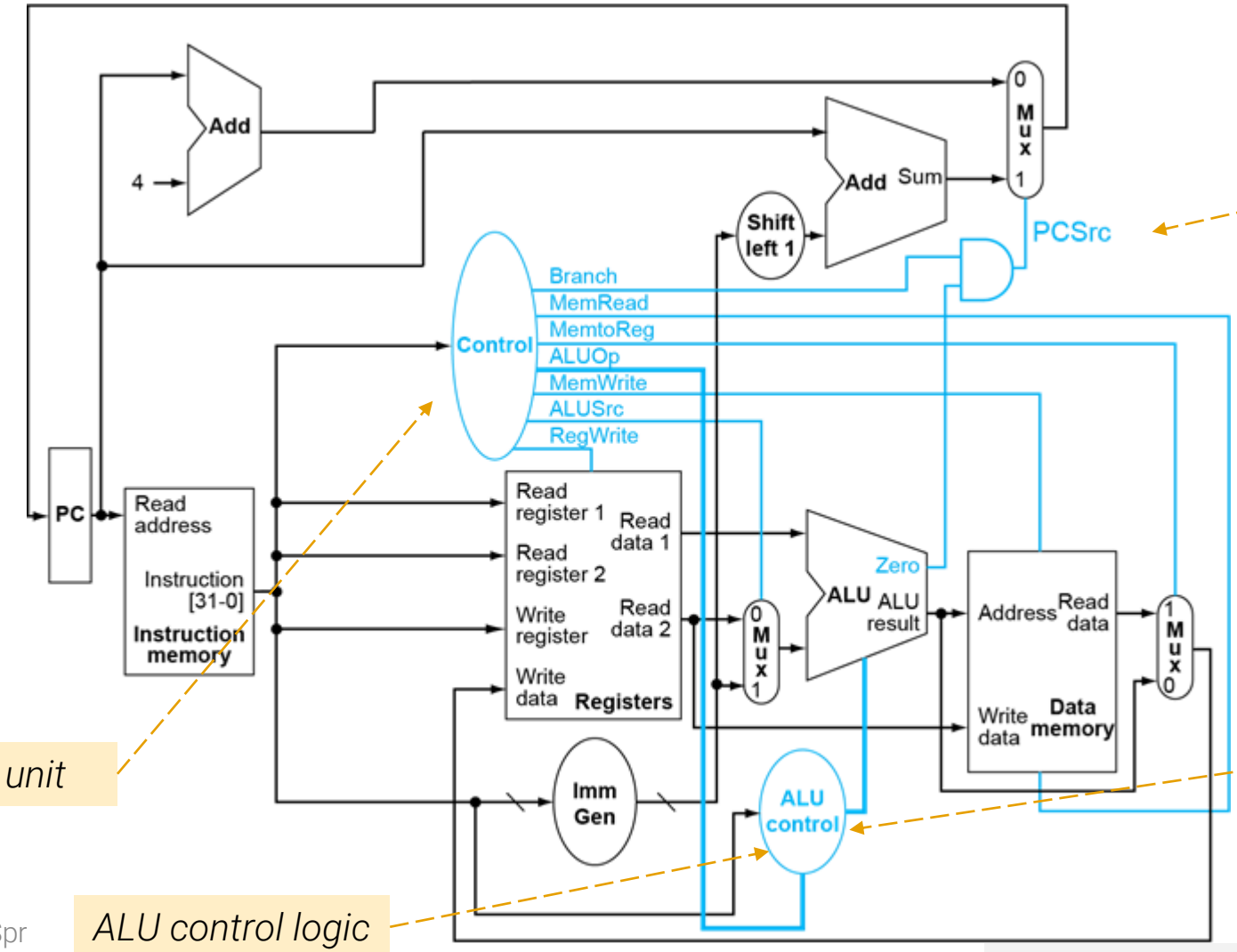


The control unit

ALU control logic

Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

The End: Adding the Control Unit



The control unit

ALU control logic

*Note: The control unit can set all but the **PCSrc** control signal based solely on the **opcode** and **funct** fields of the instruction.*

***PCSrc** should be asserted if the instruction is **beq** and if the **Zero** output of the ALU is asserted; hence the AND gate*

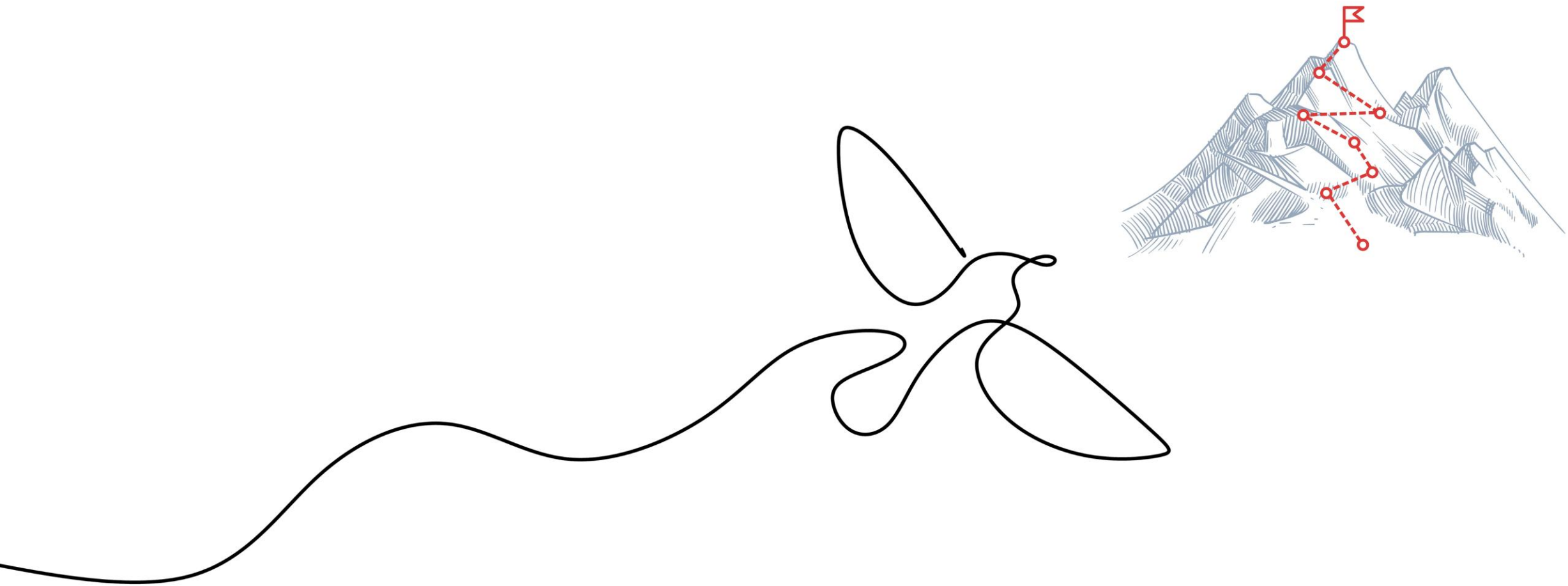
In our simplified CPU implementation, as only a few operations are to be supported, four ALU operation bits are sufficient; for detailed implementation, refer to the literature

Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

Control Signals

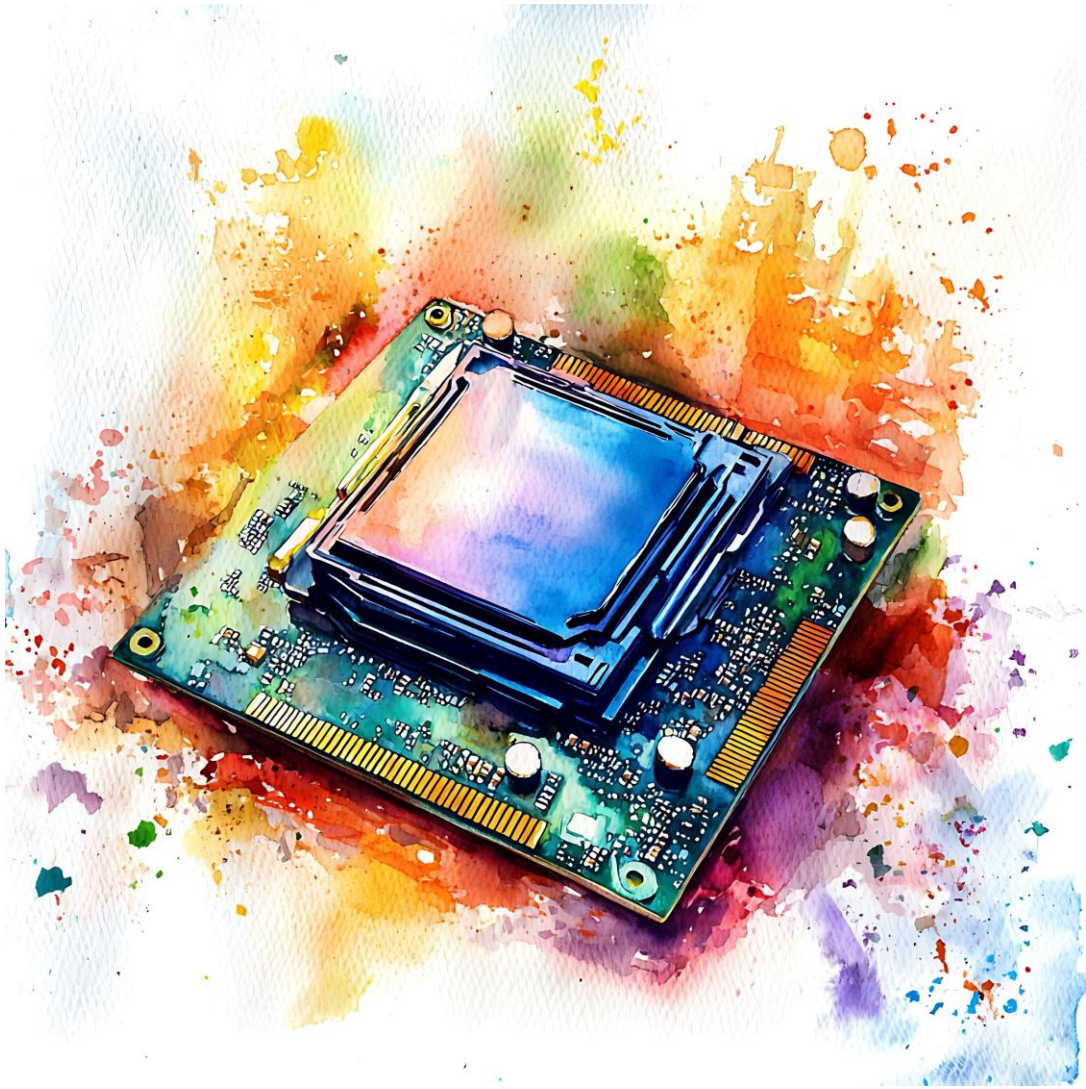
Summary

Signal name	Effect
RegWrite	If asserted, the register on the Write reg. input is written with the value on the Write data input
ALUSrc	Determines whether the second ALU operand is a register or the sign-extended immediate
PCSrc	Determines if the PC is replaced by PC + 4 or by the output of the adder that computes the branch target
MemRead	If asserted, data memory contents designated by the address input are put on the Read data output
MemWrite	If asserted, data memory contents designated by the address input are replaced by the value on the Write data input
MemtoReg	Determines if the value fed to the register Write data input comes from the ALU or from the data memory



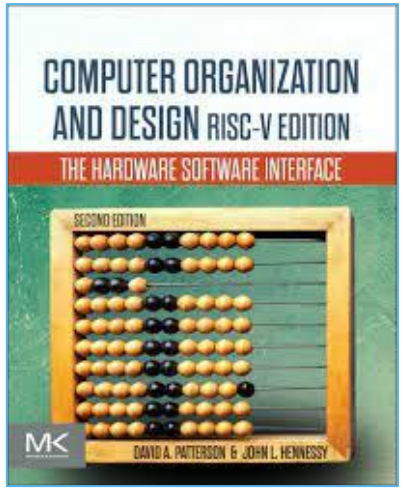
Single-Cycle vs Multicycle

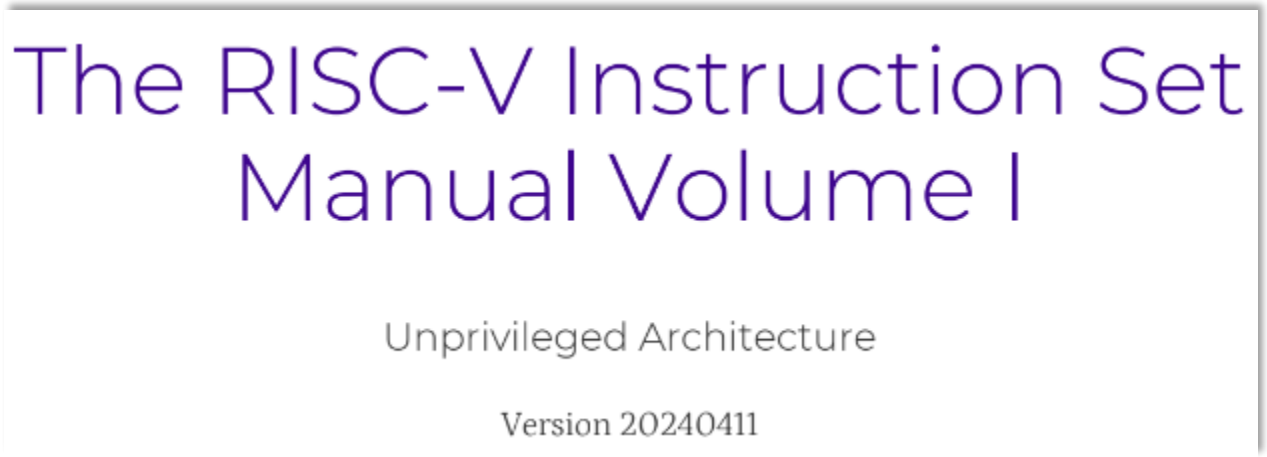
Next Lecture



© Woranuch / Adobe Stock

Literature





Visit online: Link

- Chapter 4: The Processor
  - 4.5