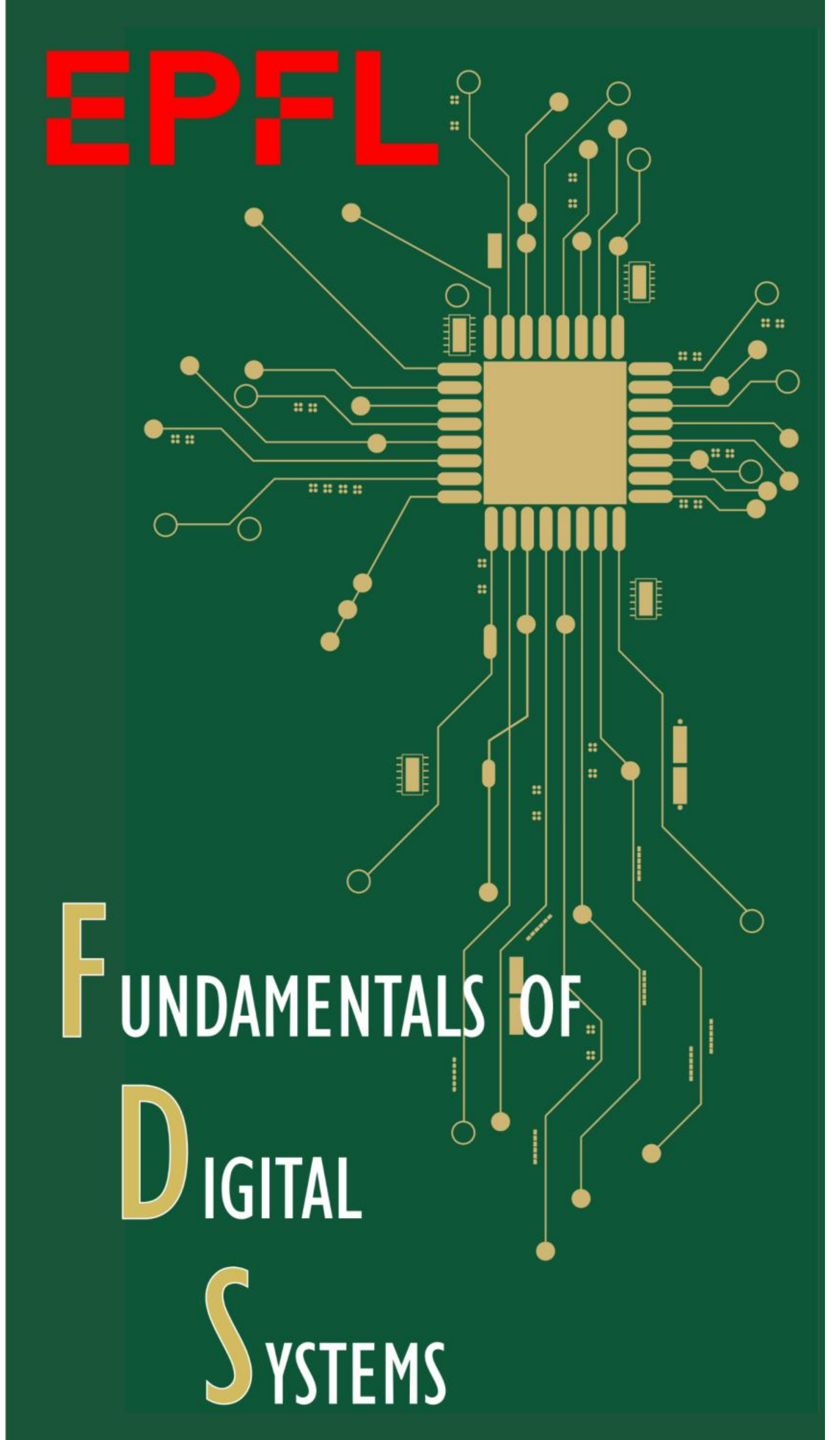


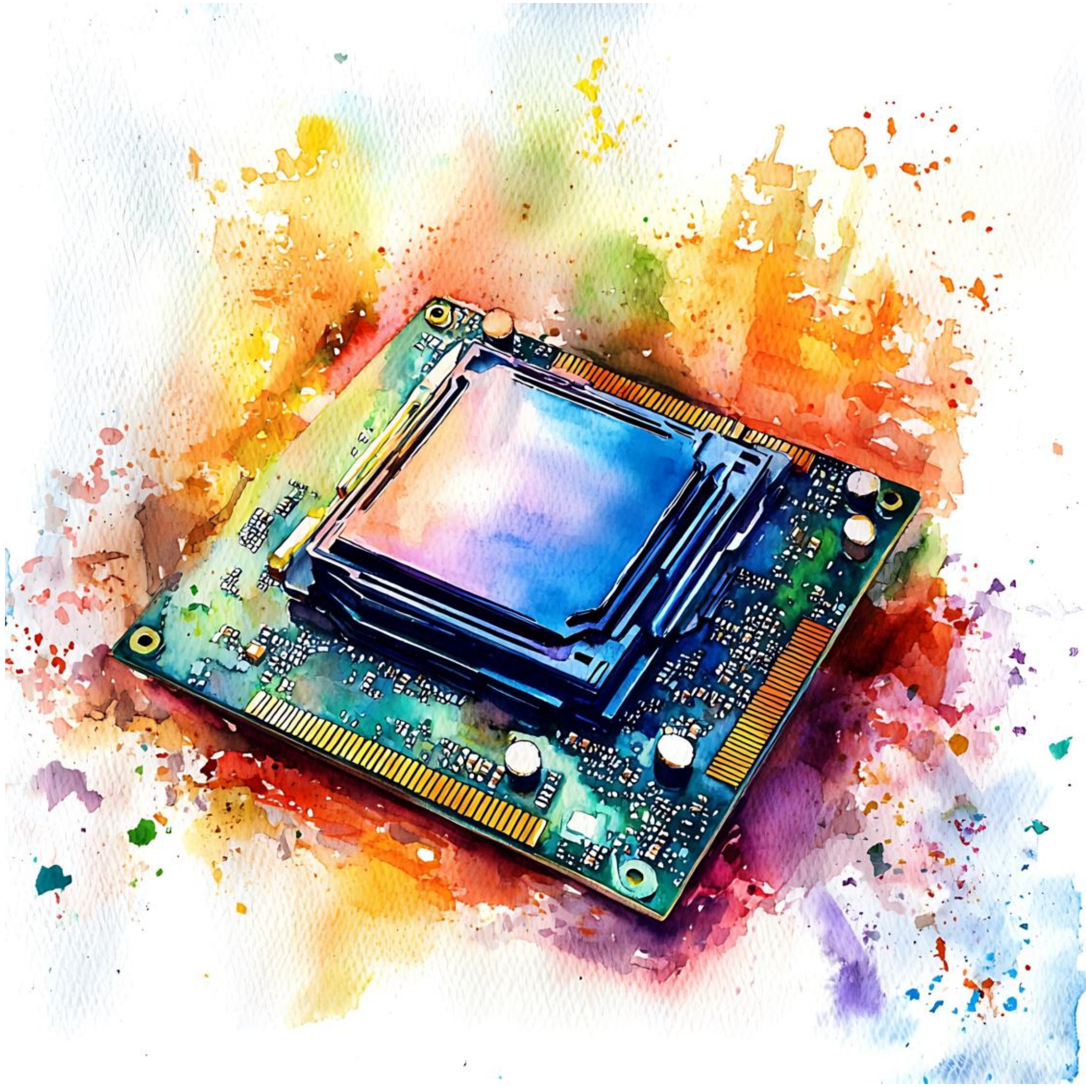
Computer Architecture

Multicycle CPU Implementation

CS-173 Fundamentals of Digital Systems
Mirjana Stojilović
Spring 2025

Previously on FDS

- CPU Performance
- Processor Implementations
  - Single-cycle CPU



Recall: 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

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

Recall: Single-Cycle CPU

- In a single-cycle CPU, all operations required by an instruction are performed within one clock cycle (CPI = 1.0)

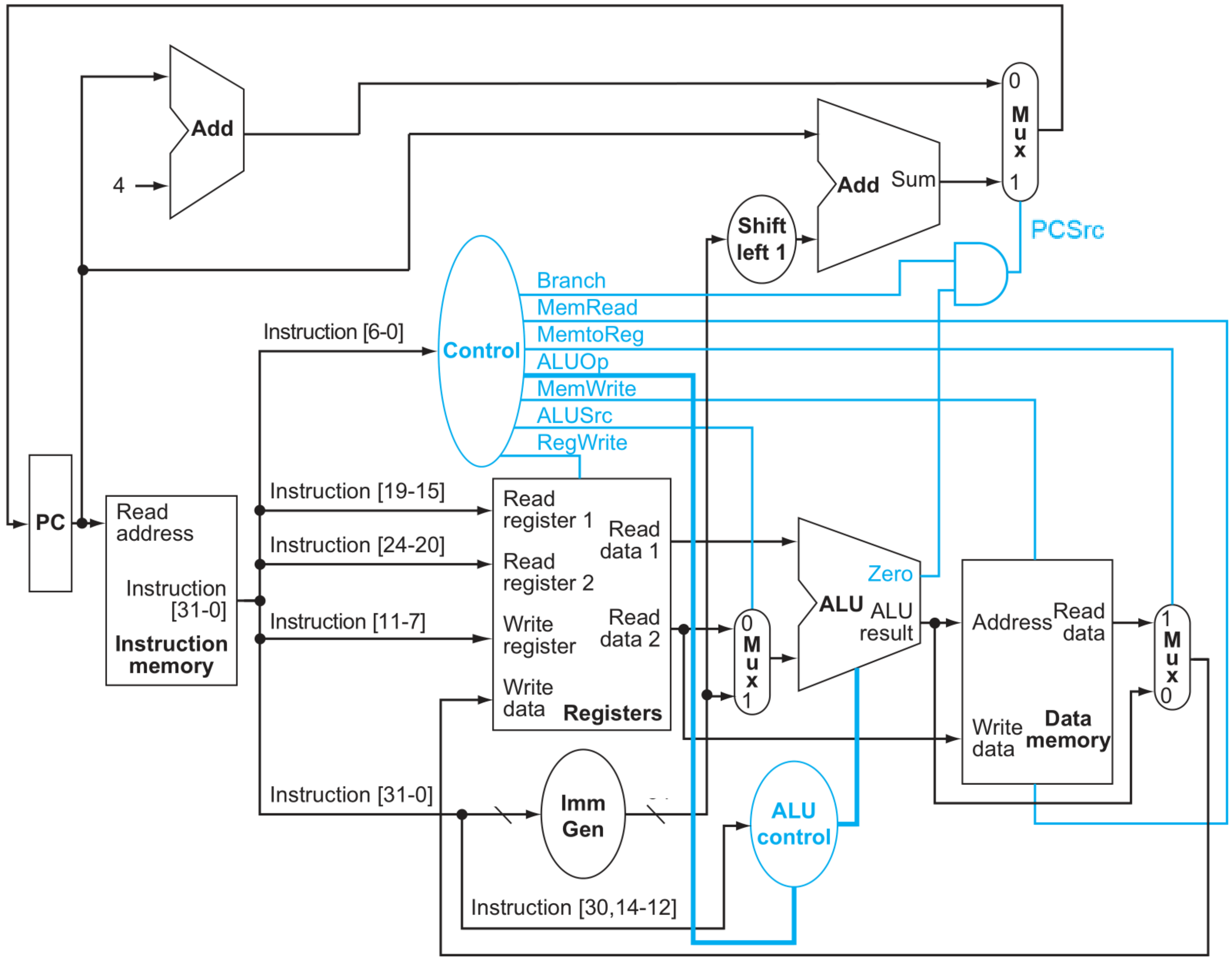
$$\underbrace{\textbf{CPU time}}_{\text{for a program } \textit{(in seconds)}} = \underbrace{\textbf{Instruction count}}_{\text{for a program}} \times \textbf{CPI} \times \underbrace{\textbf{Clock cycle}}_{\textit{(in seconds)}}$$

Recall: A Simple Single-Cycle CPU

- 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

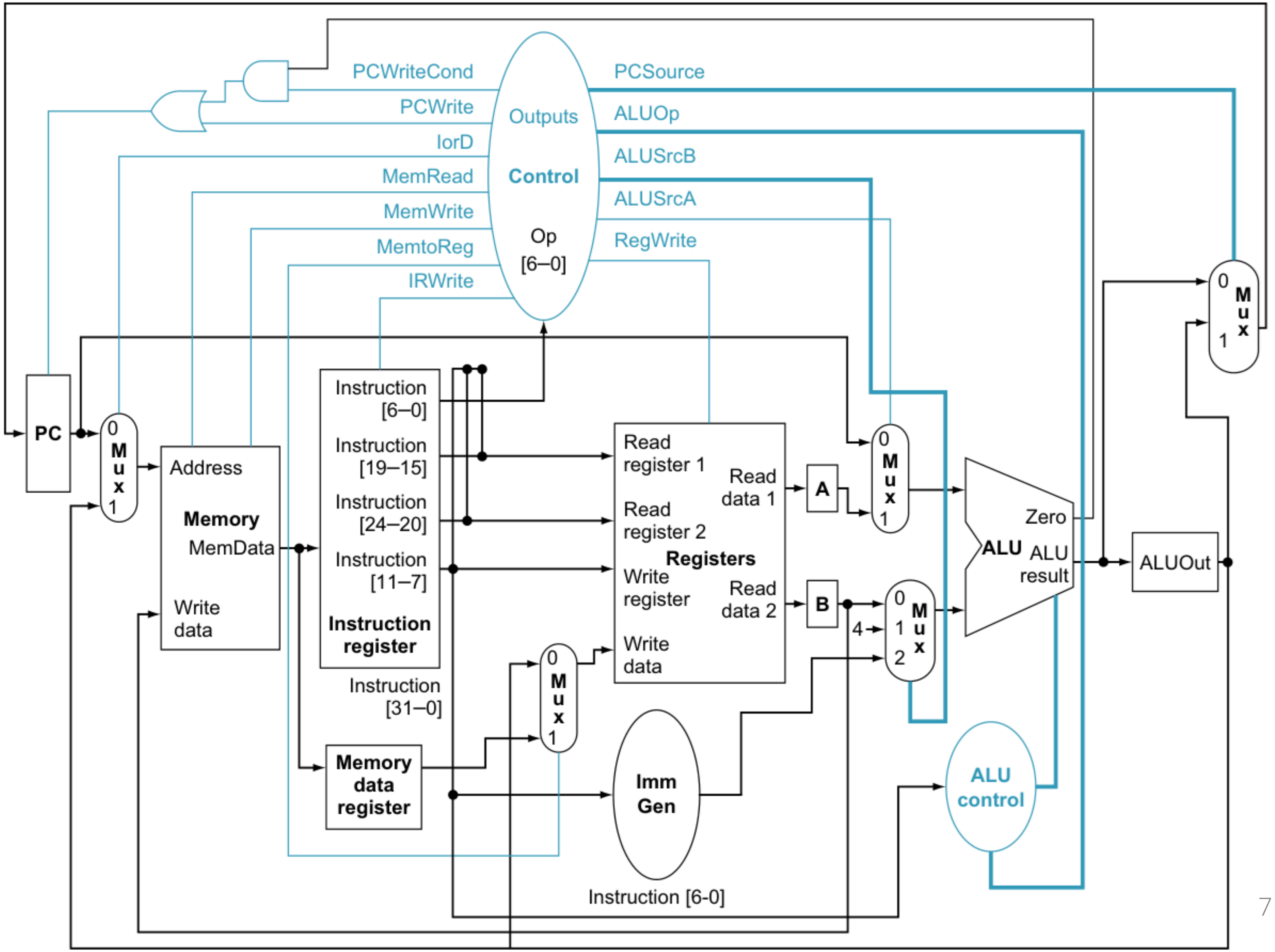
What We Know

A Simple Single-Cycle CPU



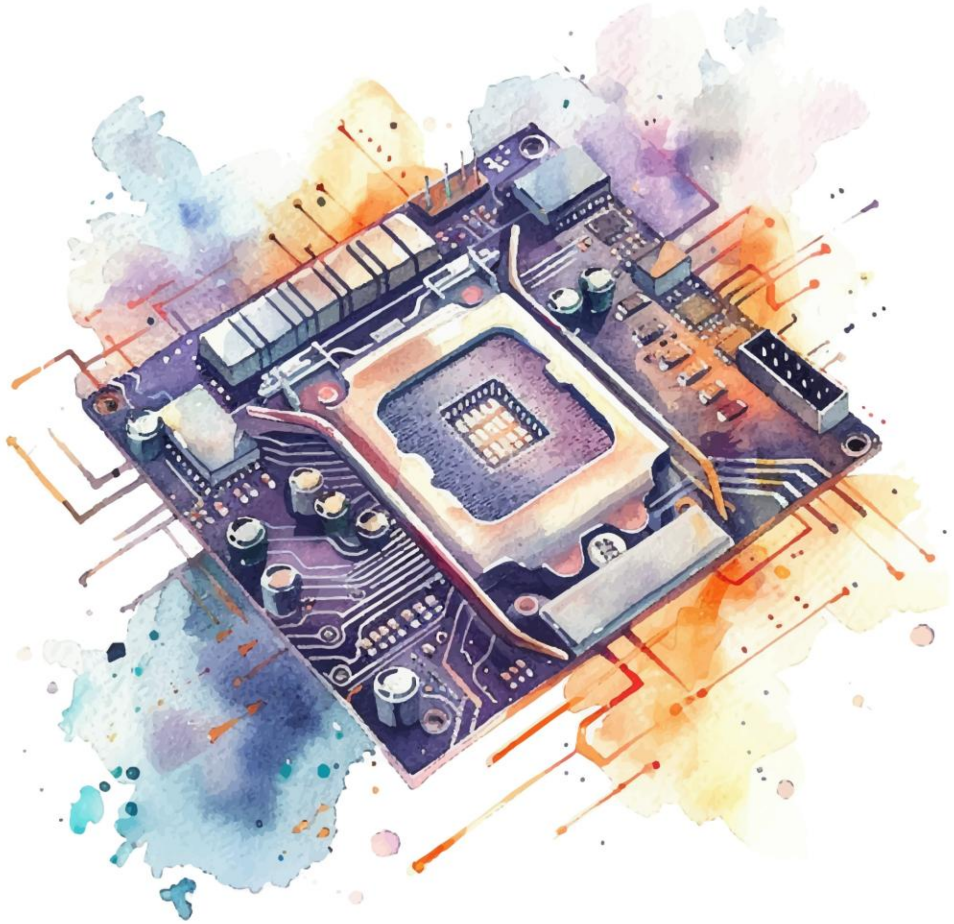
Where We're Heading

A Simple Multicycle CPU

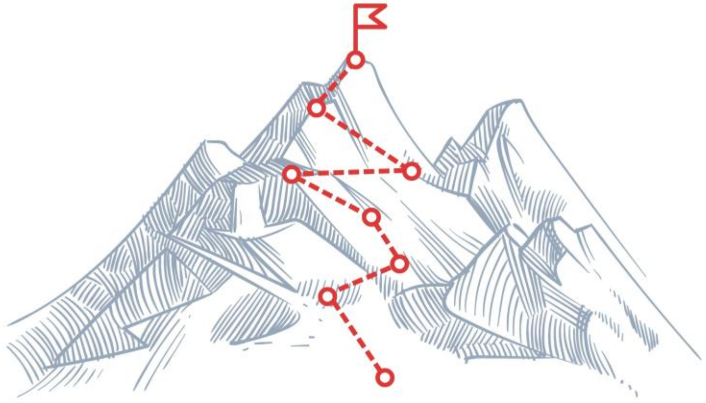


Let's Talk About

- Processor Implementations
  - Single-cycle vs. multicycle CPU
  - Multicycle CPU



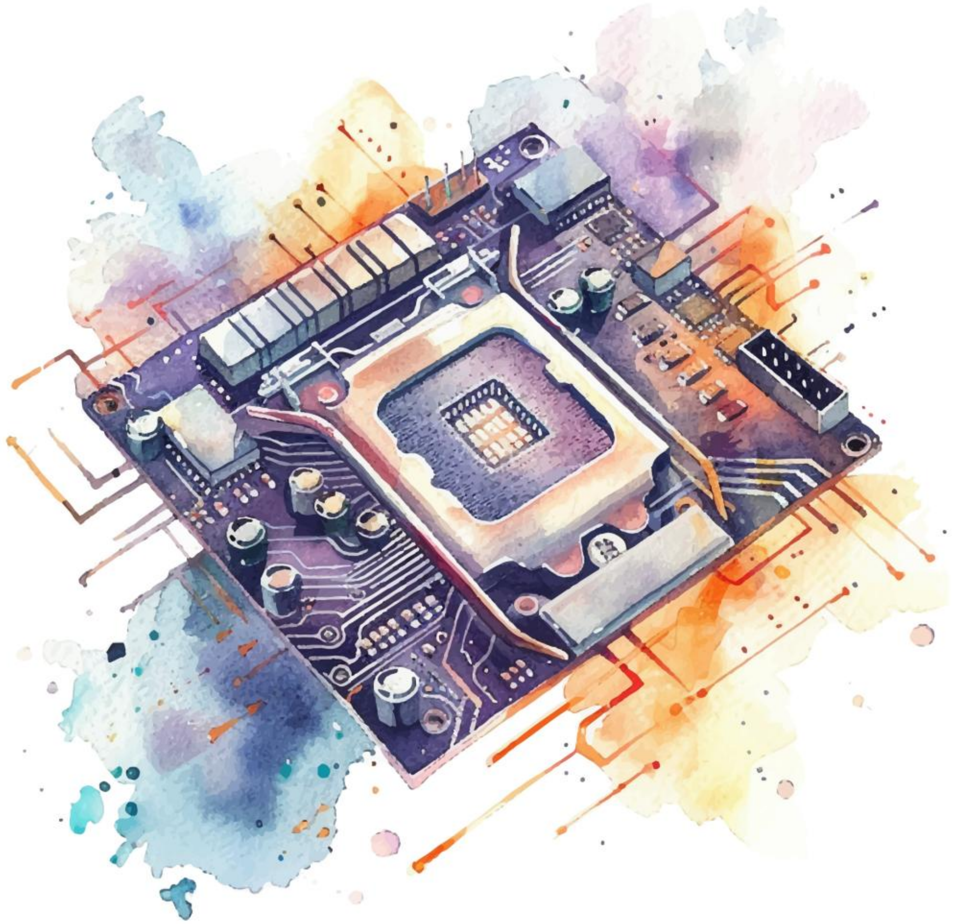
Learning Outcomes



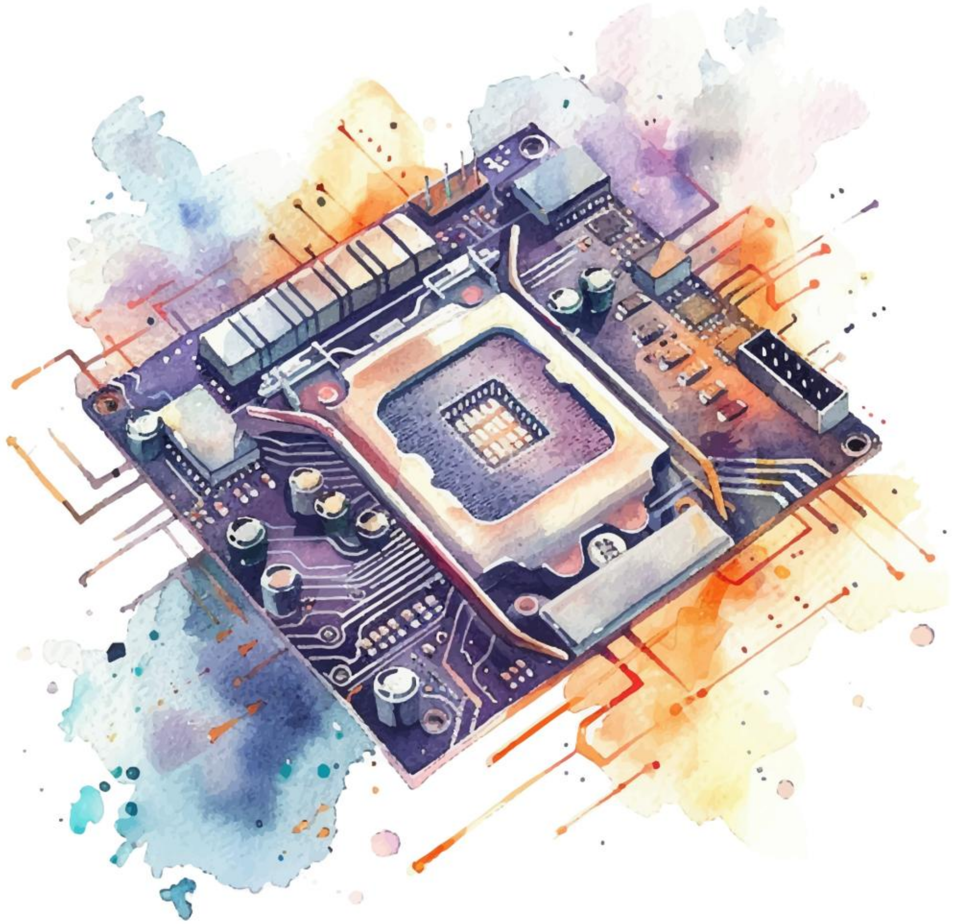
- Discover multi-cycle CPU implementation
- Advantages of multicycle vs. single-cycle
- Understand the schematic
  - Functional units
  - Control signals
- Be able to list and explain instruction steps

Quick Outline

- Single-cycle vs. multicycle CPU
- Multicycle CPU
  - Additional registers
  - Additional multiplexers
  - Control
  - Datapath + control



Single-Cycle vs Multicycle



Operation of the Datapath

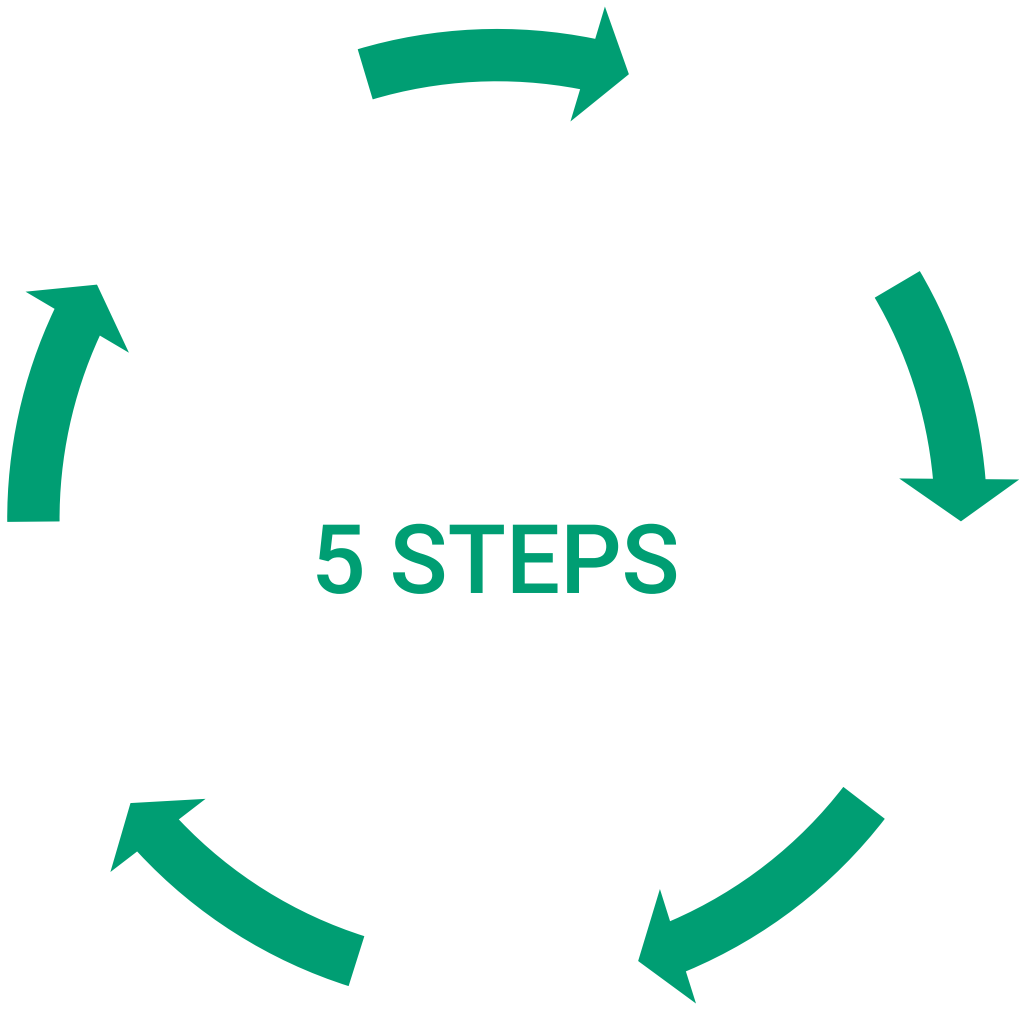
R-type Instructions

- Performing an **R-type instruction** takes **four** steps
  1. Instruction fetched from the instruction memory, and the PC incremented
  2. Two registers read from the register file; the control unit computes and sets the control signals correspondingly
  3. ALU operates on the data read from the register file, using bits of the instruction opcode to generate the desired ALU function
  4. The result from the ALU is written to the register file

Operation of the Datapath

Load Instruction

- Performing a **load instruction** takes **five** steps
  1. Instruction fetched from the instruction memory, and the PC incremented
  2. Base address read from the register file
  3. ALU computes the sum of the value read and the sign-extended 12 bits of the instruction (immediate)
  4. The sum from the ALU is used as the address for the data memory
  5. The data read from the data memory is written to the register file



Operation of the Datapath

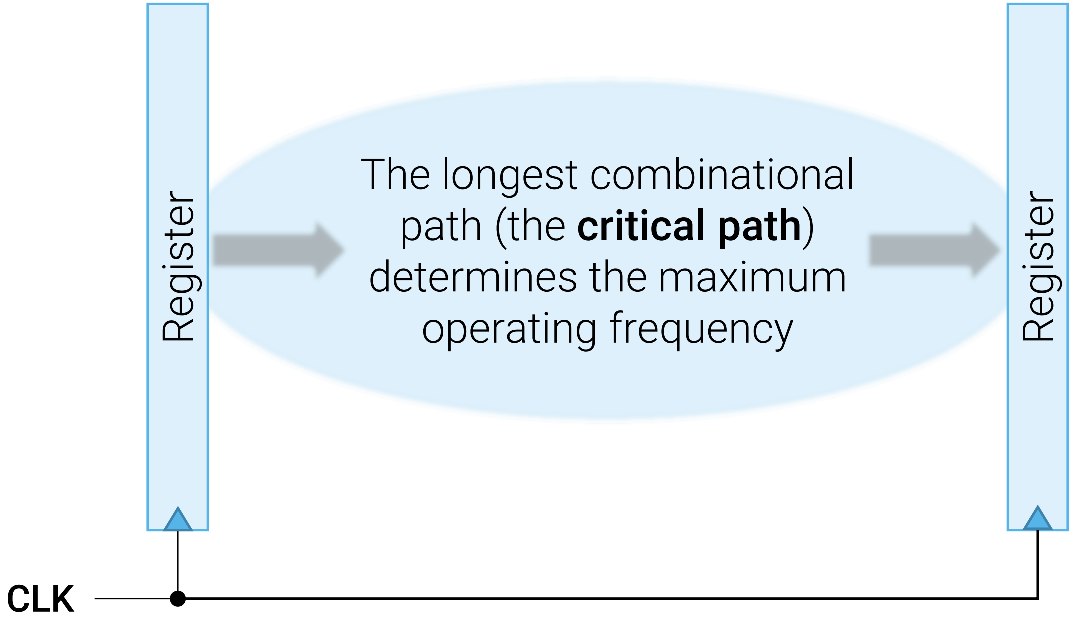
Branch if Equal Instruction

- Performing a **branch if equal** instruction takes **three** steps
    1. Instruction fetched from the instruction memory, and the PC incremented
    2. Two registers read from the register file
    3. ALU subtracts one value from the other. PC is added with the sign-extended 12 bits of the instruction (immediate) << 1, to prepare the branch target address; The Zero status bit from the ALU is used to select the new PC value (the branch target address *or* PC+4)

3 STEPS

Single-Cycle vs. Multicycle Implementation

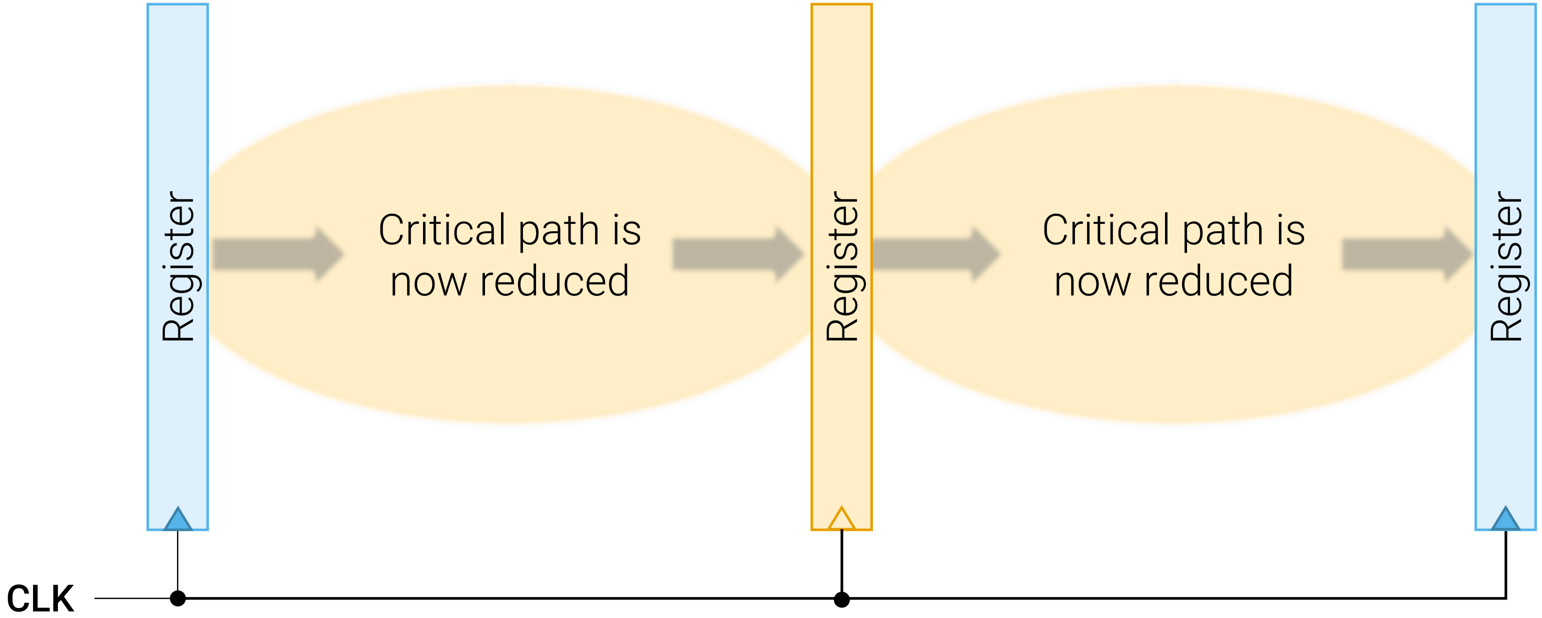
- If all instruction steps are performed in a single clock cycle, we have a **single-cycle** CPU implementation



Note: These two registers could be the same one (e.g., PC)

Single-Cycle vs. Multicycle Implementation

- Alternative implementation is a **multicycle** CPU
  - In a multi-cycle implementation, one or more instruction **steps** take one clock cycle, and consequently, some instructions take multiple clock cycles



Note: These two registers could be the same one (e.g., PC)

Note: This is an example; Other multi-cycle implementations are also possible

Is Single-Cycle CPU More Efficient?

- **A: No.** The clock cycle must be as long as necessary to accommodate all steps of all instructions.
  - Regardless of the number and complexity of instruction steps, every instruction takes the same time (one cycle)
- The max frequency is limited by the longest path any instruction takes

Why is Multi-Cycle CPU More Efficient?

- **A:** In a multi-cycle implementation, fewer instruction steps take one cycle. The **maximum frequency increases** compared to the single-cycle because now the critical path is shorter. Instructions that require fewer steps will likely be executed faster. The overall **execution time** of the program is reduced.

Other Advantages of Multi-Cycle CPUs

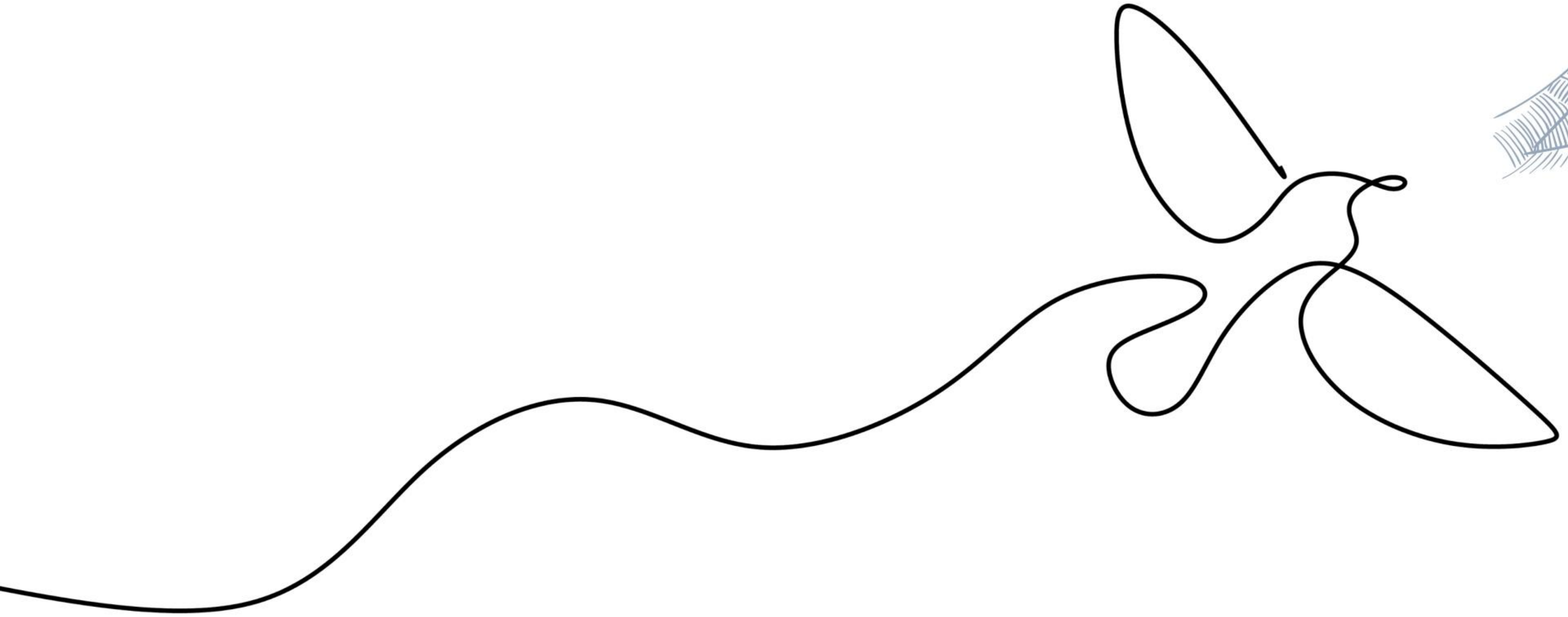
- Another important advantage of the multi-cycle implementation is the ability to **reuse** a functional unit more than once per instruction, as long as it is used in different clock cycles
- Resource reuse substantially **reduces** the overall hardware required, a key consideration in computer architecture

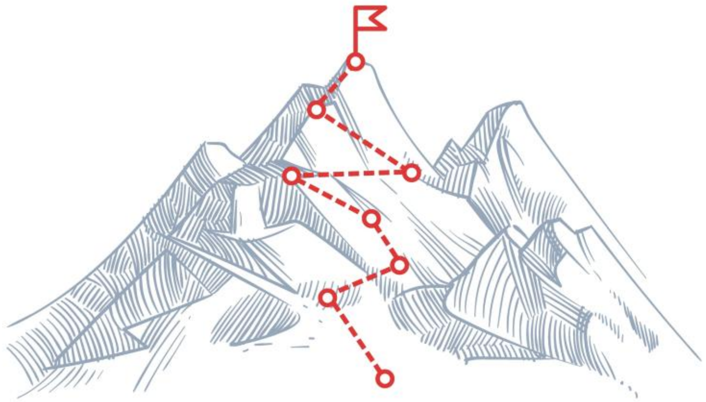
What Else is There?
Pipeline

- In practice, there exists another implementation technique called **pipelining**, in which multiple instructions are overlapped in execution, and hardware reuse is pushed to the limits

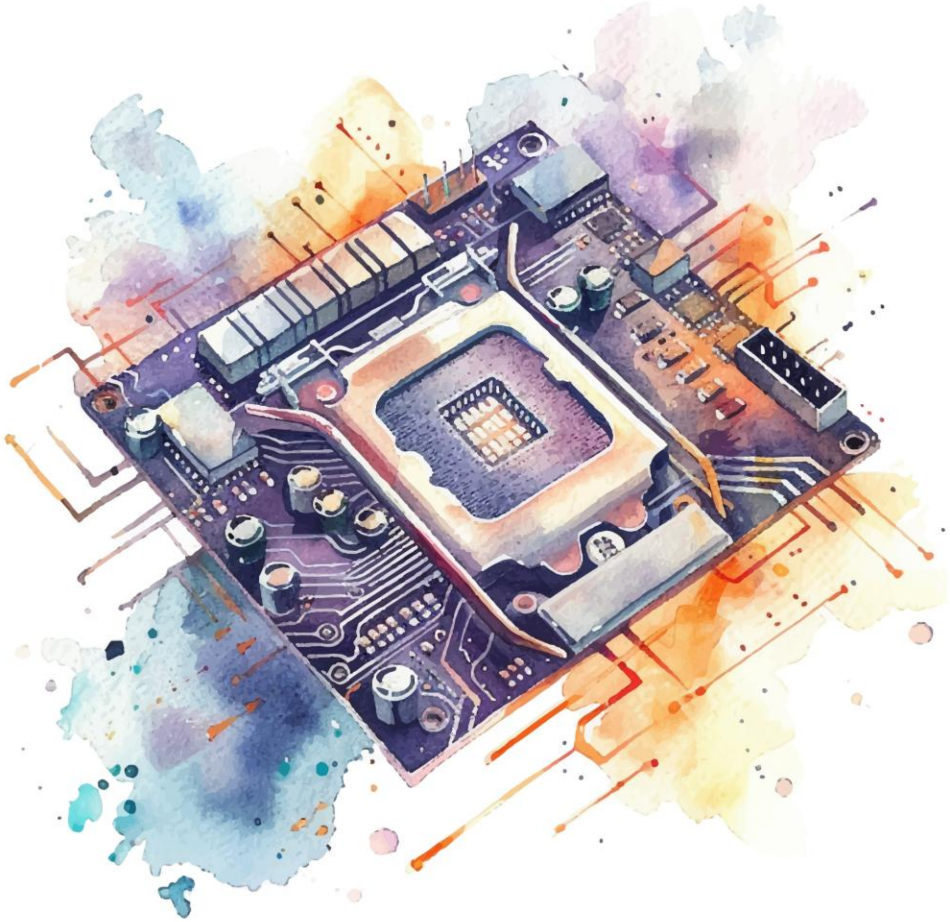
  Most modern processors are implemented using this technique, which comes with a set of challenges of its own

  Note: Out of scope for CS-173; taught in CS-200 (Computer Architecture)





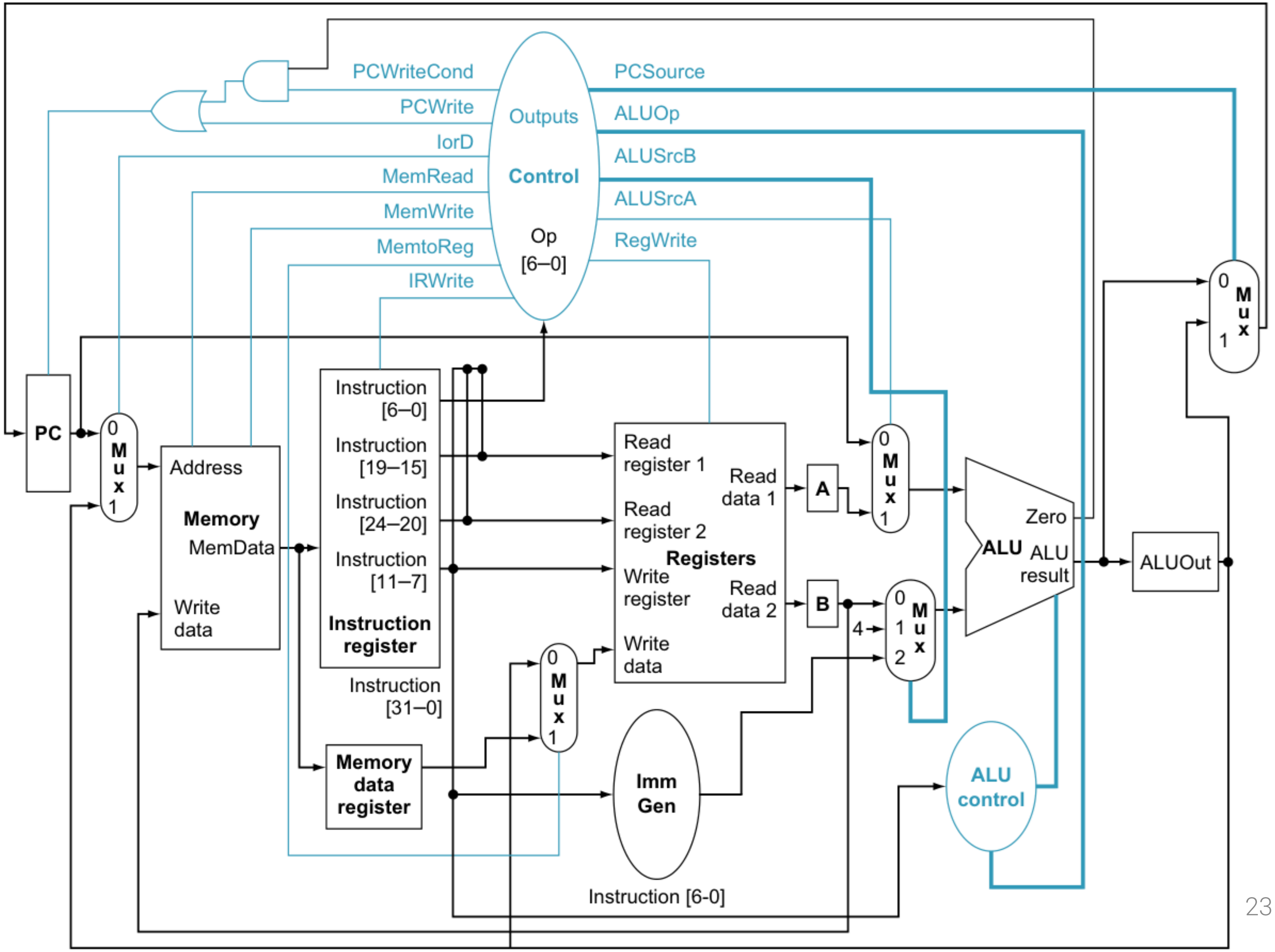
A Multicycle CPU



© EnelEva / Adobe Stock

Where We're Heading

A Simple Multicycle CPU

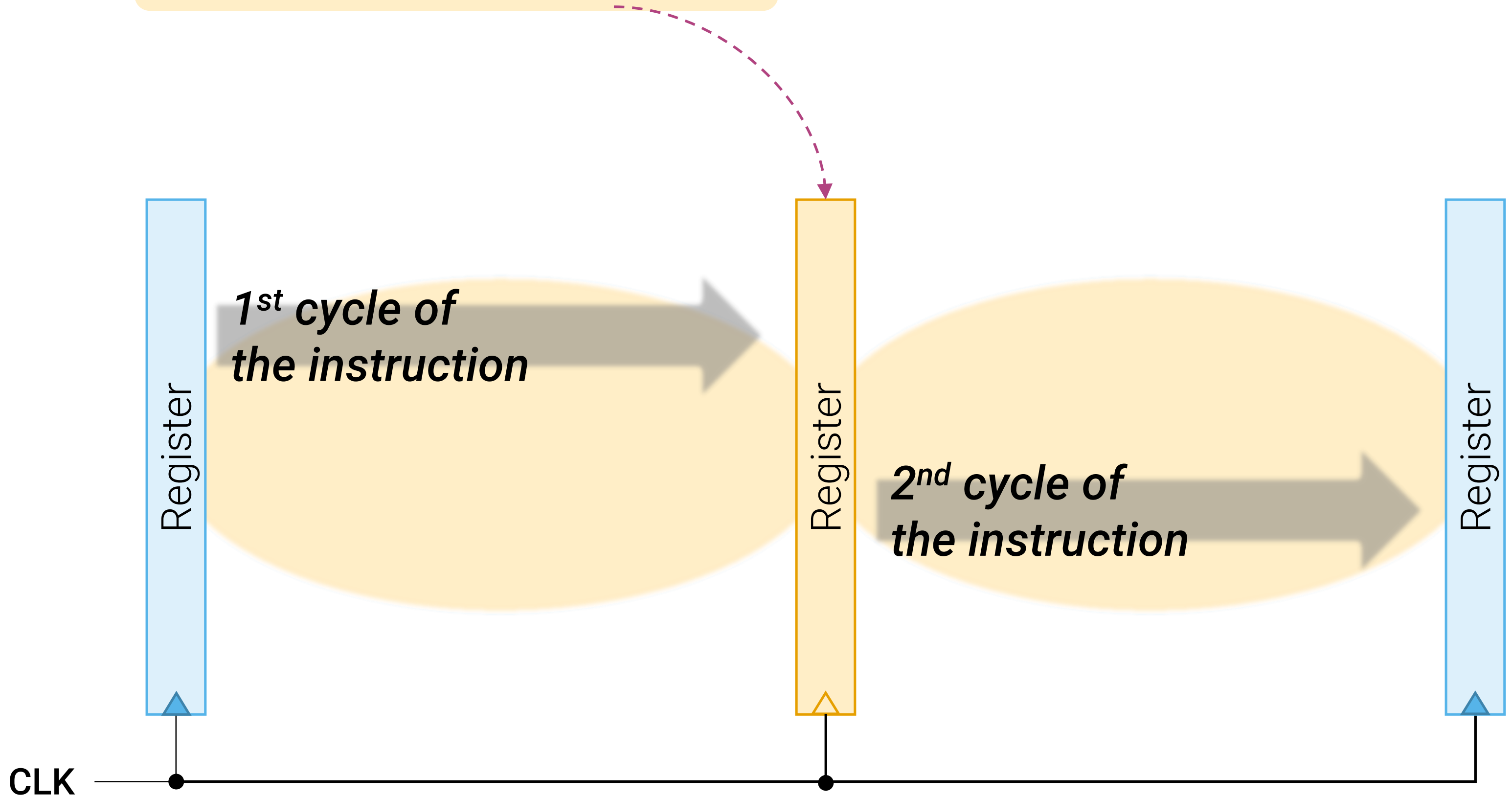


Multicycle CPU
vs. Single-Cycle CPU

- A **single memory** unit for **both** instructions and data
  - **Why?** Having more than one cycle available (more time to read instructions, read/write data) allows memory sharing
- A **single ALU** instead of an ALU and two adders
  - **Why?** The same ALU can be used in different clock cycles
- **Additional registers** to hold the outputs of the functional units until the value is used (consumed) in a subsequent clock cycle
  - **Why?** Ensure that the value to be used is “stable” for the entire cycle

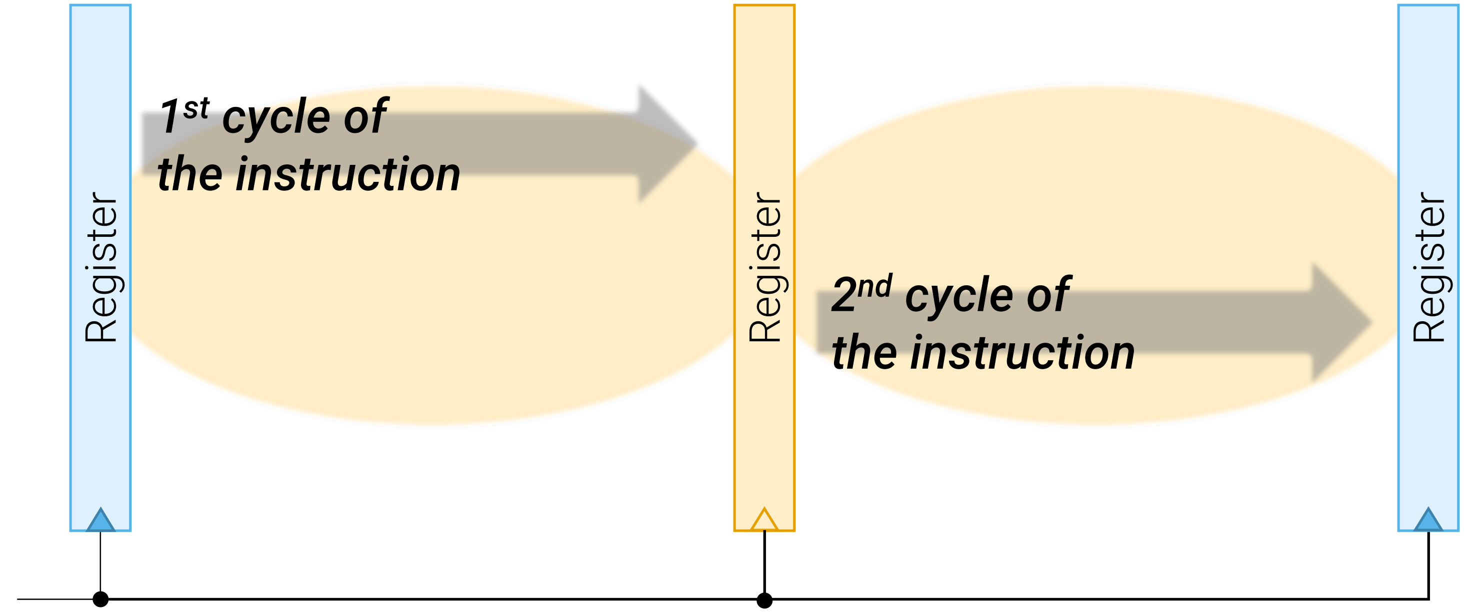
Multicycle CPU

- Data used by the **same instruction** in a later clock cycle must be stored in the **additional registers** appended to the functional units



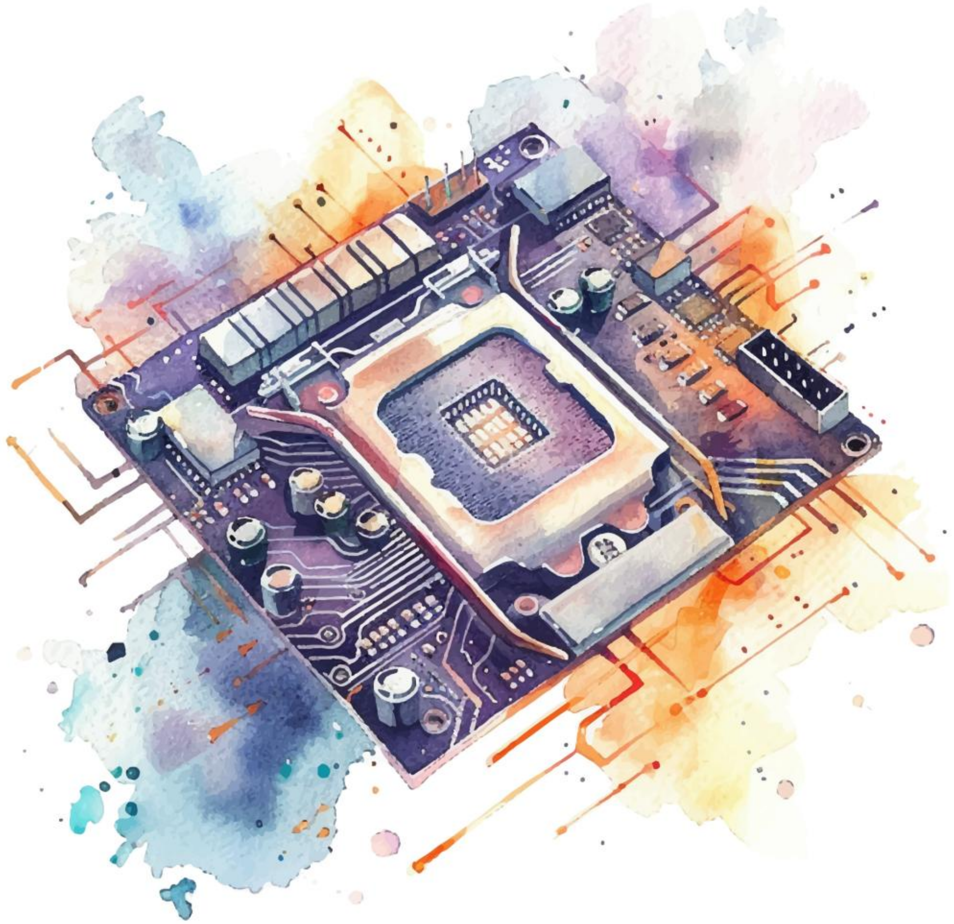
Multicycle CPU

- The location of the additional (temporary values) registers is determined by what functional units will “fit” in a clock cycle and what data is needed in later cycles



A Multicycle CPU

- Datapath



A Simple Multicycle CPU

- *Recall:* 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

A Simple Multicycle CPU

- Let us assume the CPU clock cycle can accommodate **at most**
  - one memory access,
  - one register file access (two reads or one write),
  - or an ALU operation.

- **Q:** How many additional temporary registers should we insert, and where?

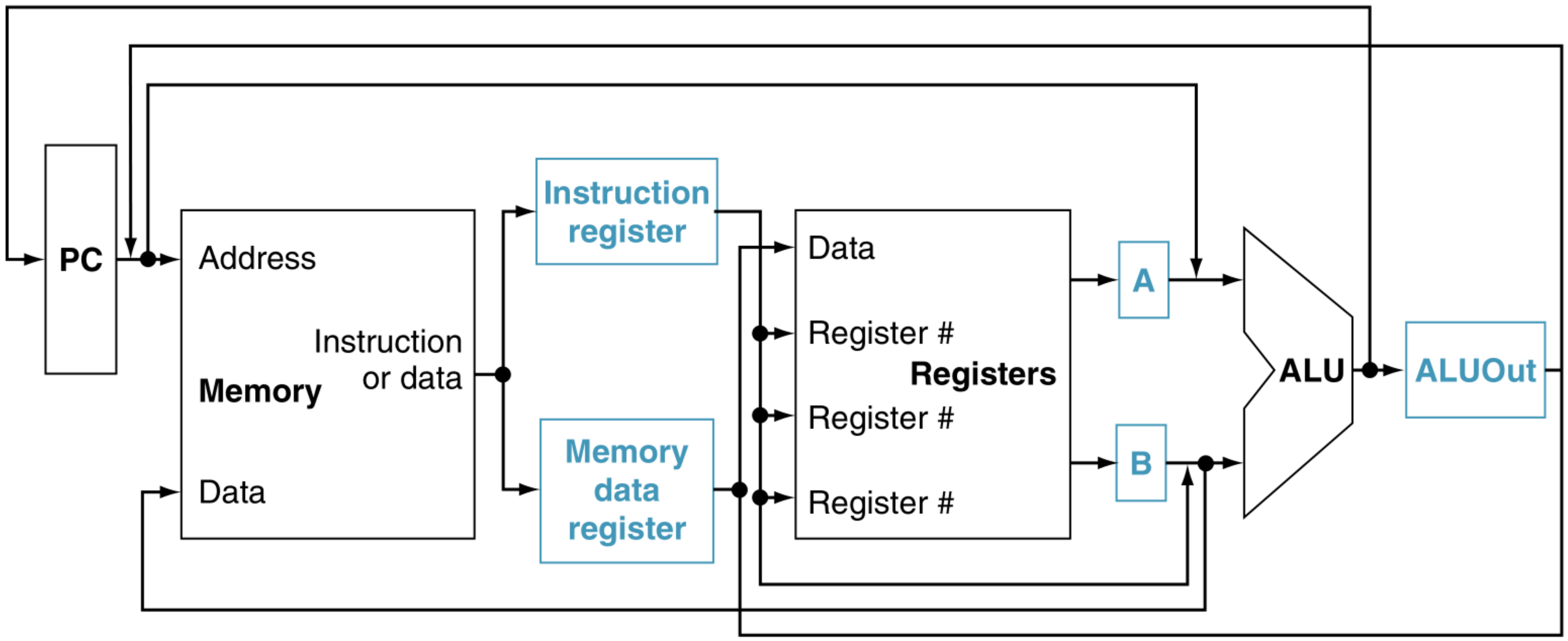
A Simple Multicycle CPU

Solution

- Any data produced by one of these three functional units (memory, register file, ALU) must be saved into a temporary register for use in a later cycle.
- **Five registers** should be added
  - The **instruction register (IR)** and the **memory data register (MDR)**
    - Save the output of the shared memory for an instruction or data read
    - Two registers because both instruction and data are needed in the same cycle
  - Two registers at the register file output, to hold the R-instruction operands read from the register file
  - One register at the output of the ALU, to hold its result

Additional Registers

Multicycle CPU Datapath, High-level View



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

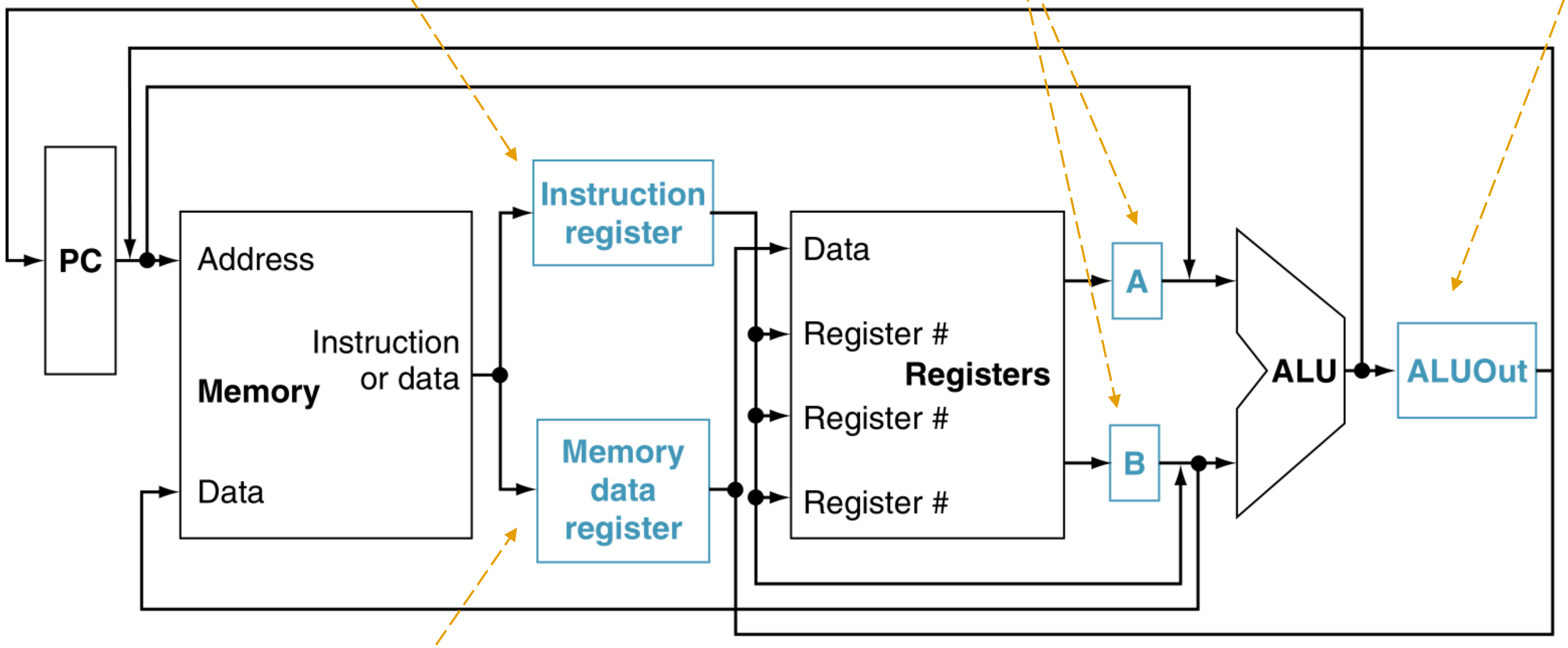
Additional Registers

Multicycle CPU Datapath, High-level View

*Instruction register (**IR**) – for the 32-bit instruction word read from memory*

One register per output of the register file, to hold the operands for the ALU

A register for ALU output



*Memory data register (**MDR**)– for the 32-bit data word read from memory*

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

How Many is Multi?

- *Recall:* Assume the CPU clock cycle can accommodate **at most** one memory access, one register file access (two reads or one write), or an ALU operation

- **Q:** Now that we have inserted additional registers in our CPU, how many cycles do the instructions take?
- **A:** It depends on the instructions and the functional units they use. CPI is instruction-dependent, unlike in a single-cycle CPU.

Functional Units Sharing
Adding MUXes

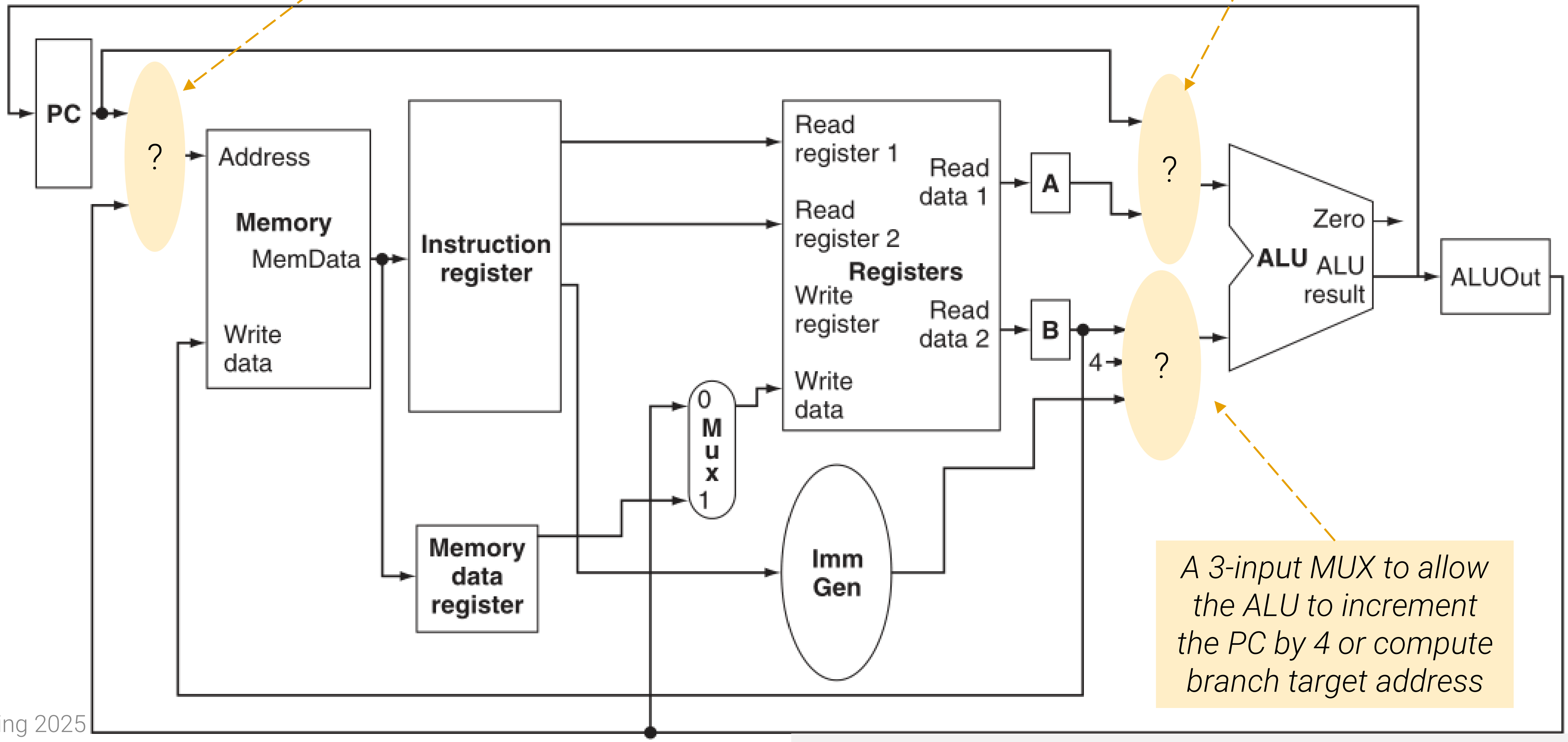
- In a multicycle CPU, several functional units are shared for various purposes.
- Compared to a single-cycle CPU implementation, we need to add new multiplexers and expand the already existing ones, to implement the support for the functional units sharing

Additional Multiplexers

Sharing Functional Units

A MUX to select between PC and ALU output for the next memory address

A MUX to select between the PC and a register from the register file



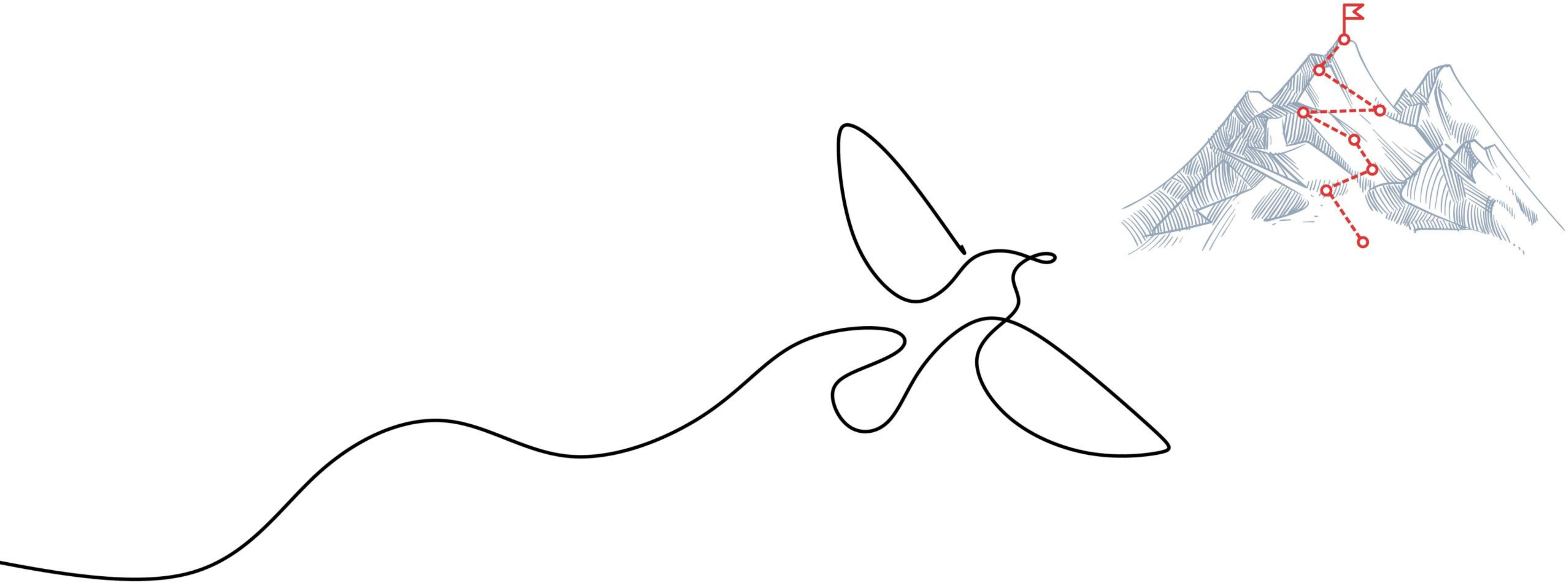
A 3-input MUX to allow the ALU to increment the PC by 4 or compute branch target address

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

Functional Units Sharing

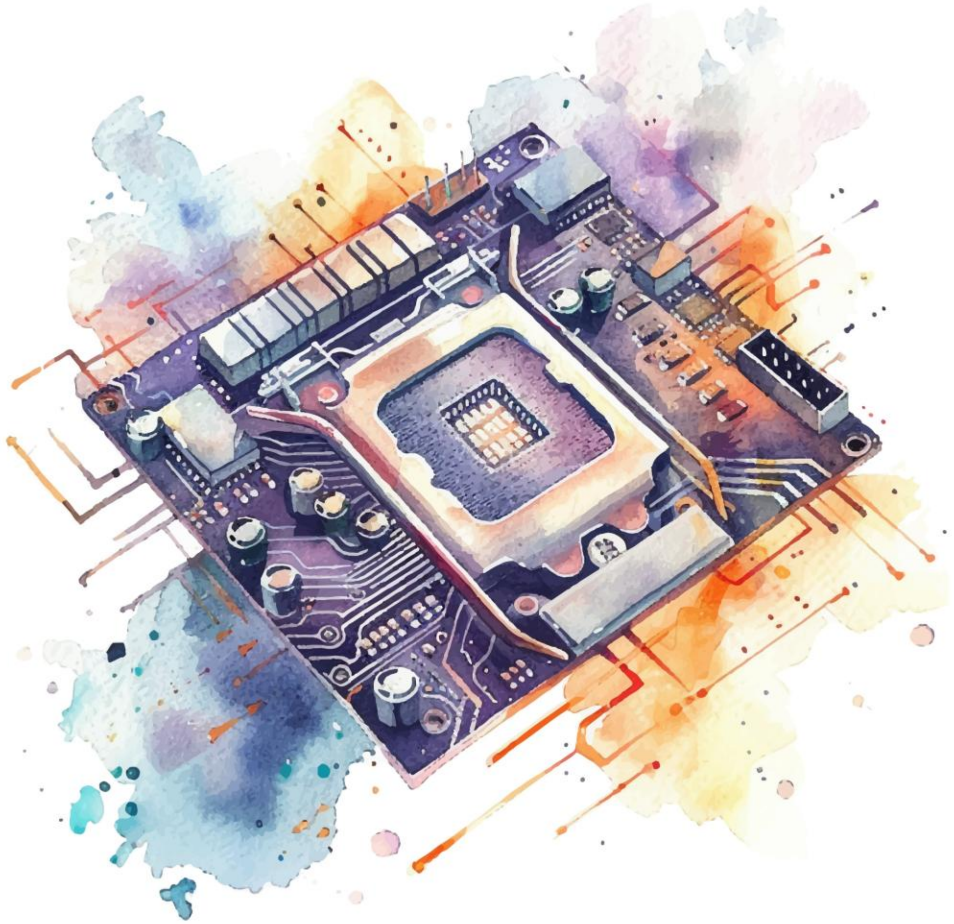
Summary

- *Recall:* In a multicycle CPU, several functional units are shared for various purposes. Therefore, we need to add new multiplexers and expand the already existing ones.
  - New mux to select the output of the ALU (store to data memory) or the PC (load from instruction memory) as the **memory address**
  - New mux for the first ALU input to select between the register file and the PC (to allow this ALU also to compute PC = PC+4 or the branch target address)
  - Widening the mux on the second ALU input (to allow this ALU also to compute PC = PC+4 or the branch target address)



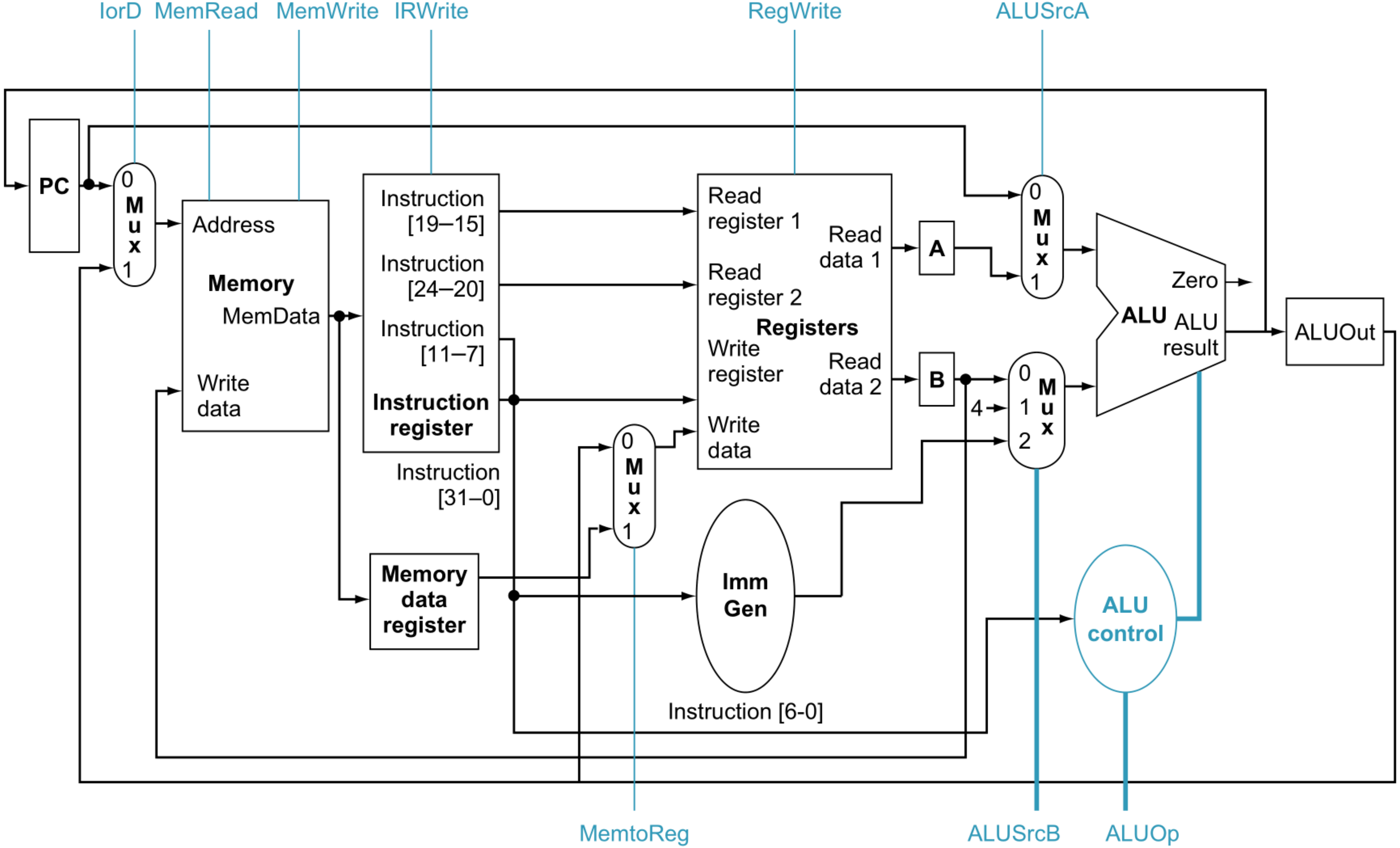
Multicycle CPU

- Control



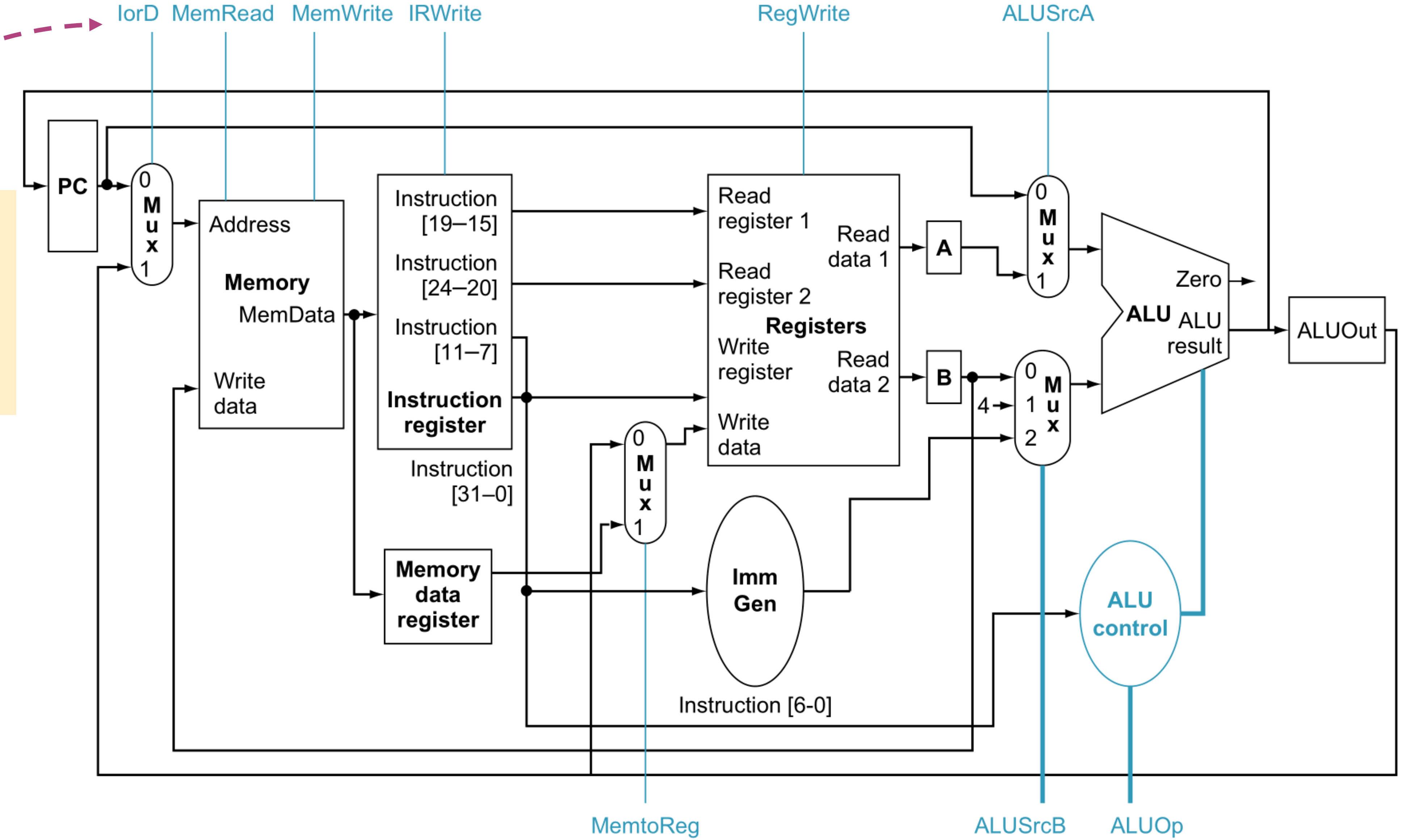
A Multicycle CPU

With Some Control Lines Shown



A Multicycle CPU

With Some Control Lines Shown



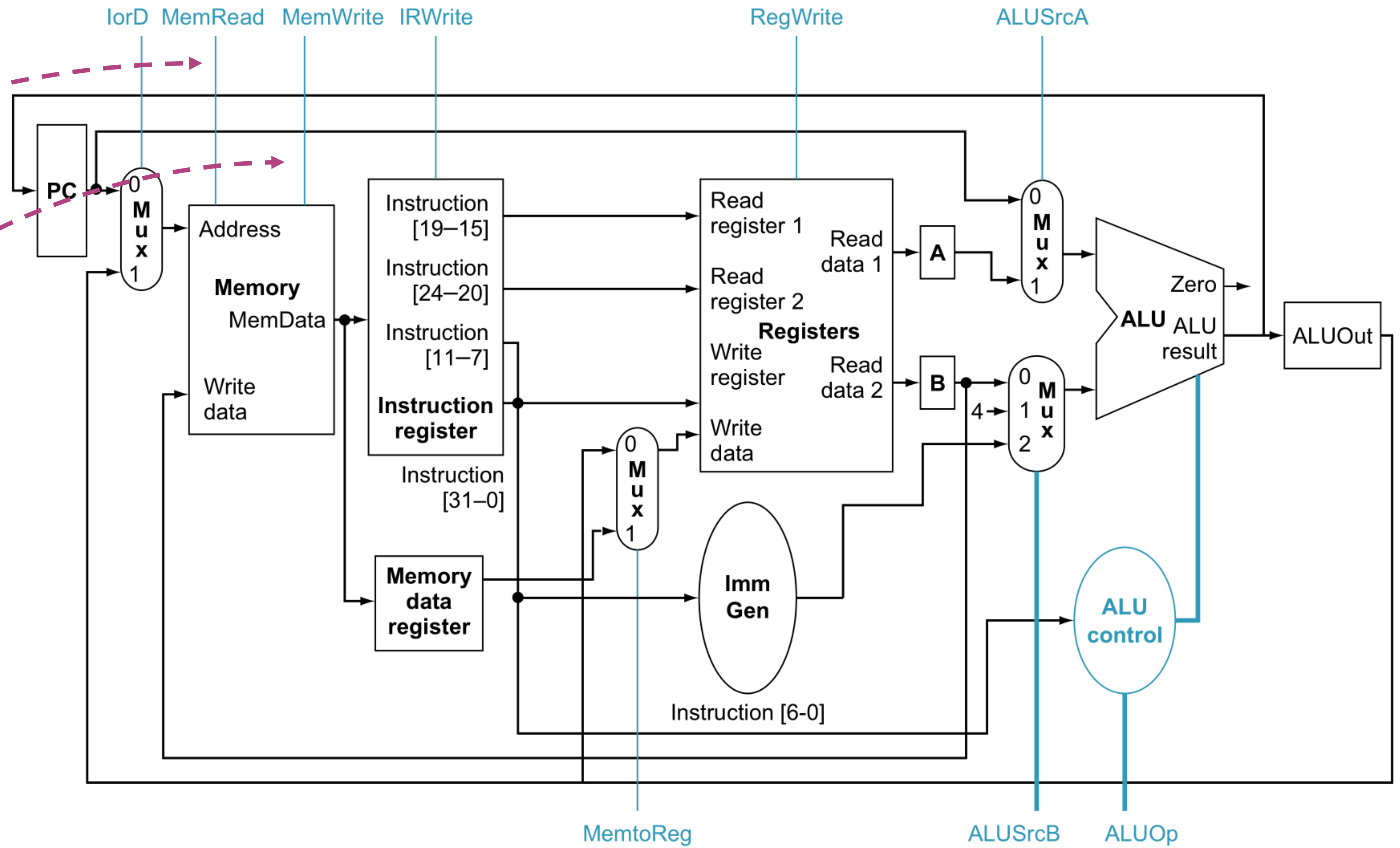
Determines if the address to the memory is supplied from ALUOut register or the PC

A Multicycle CPU

With Some Control Lines Shown

If asserted, memory contents designated by the address input are put on the output

If asserted, memory contents designated by the address input are replaced by the value on the Write data input



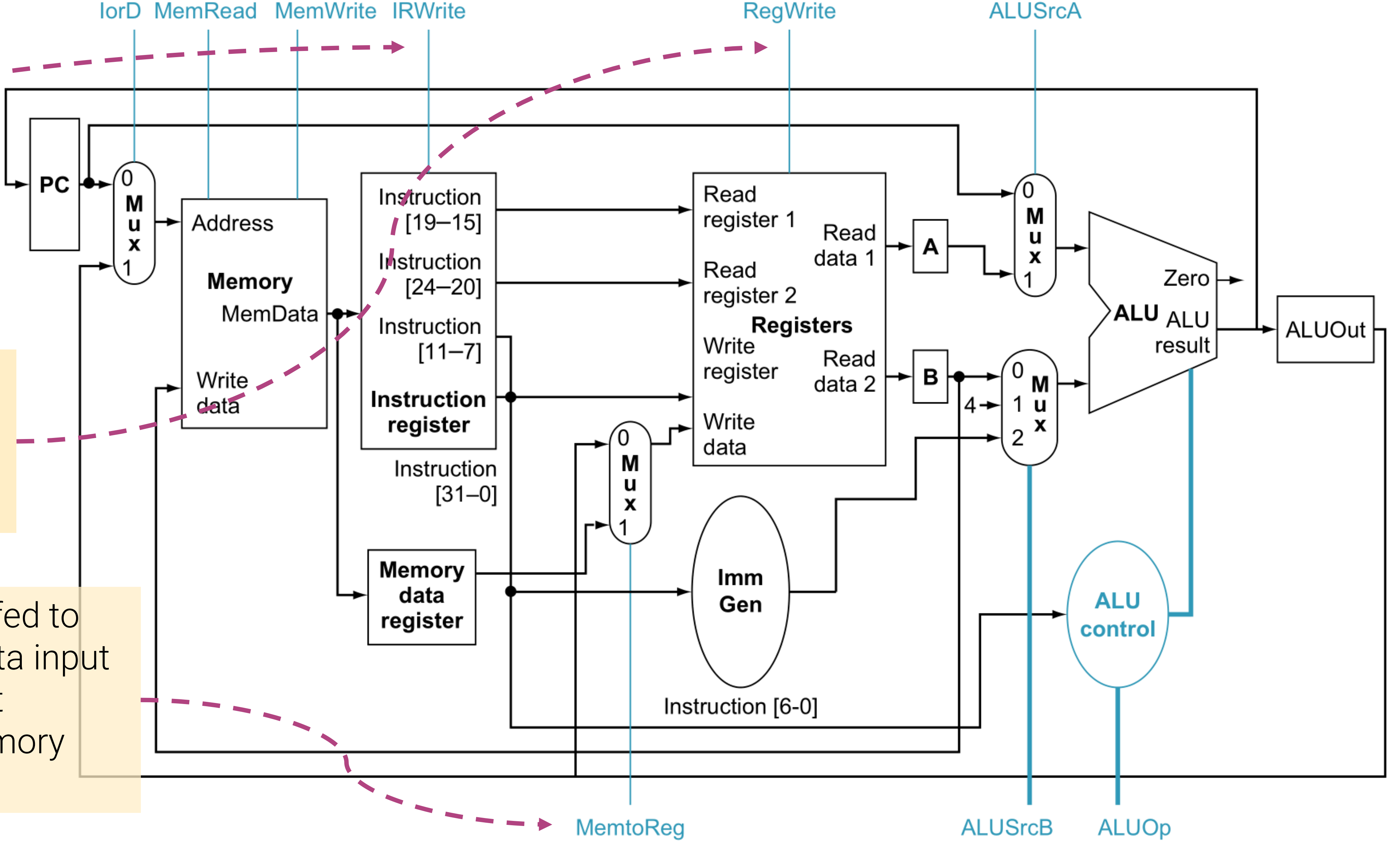
A Multicycle CPU

With Some Control Lines Shown

The output of the memory is written into the instruction register (IR)

If asserted, the register on the Write reg. input is written with the value on the Write data input

Determines if the value fed to the register file Write data input comes from the ALUOut register or from the memory data register (MDR)

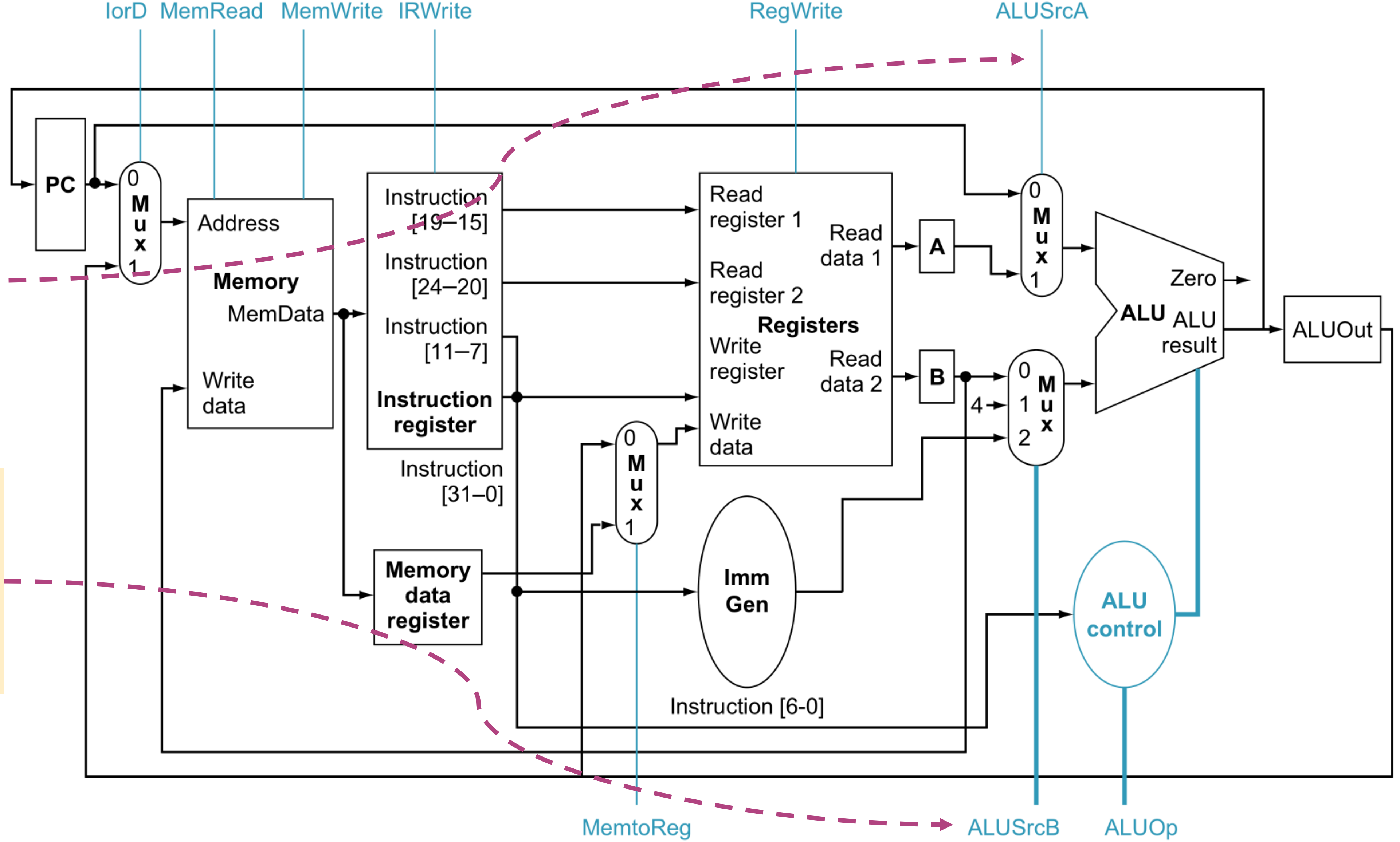


A Multicycle CPU

With Some Control Lines Shown

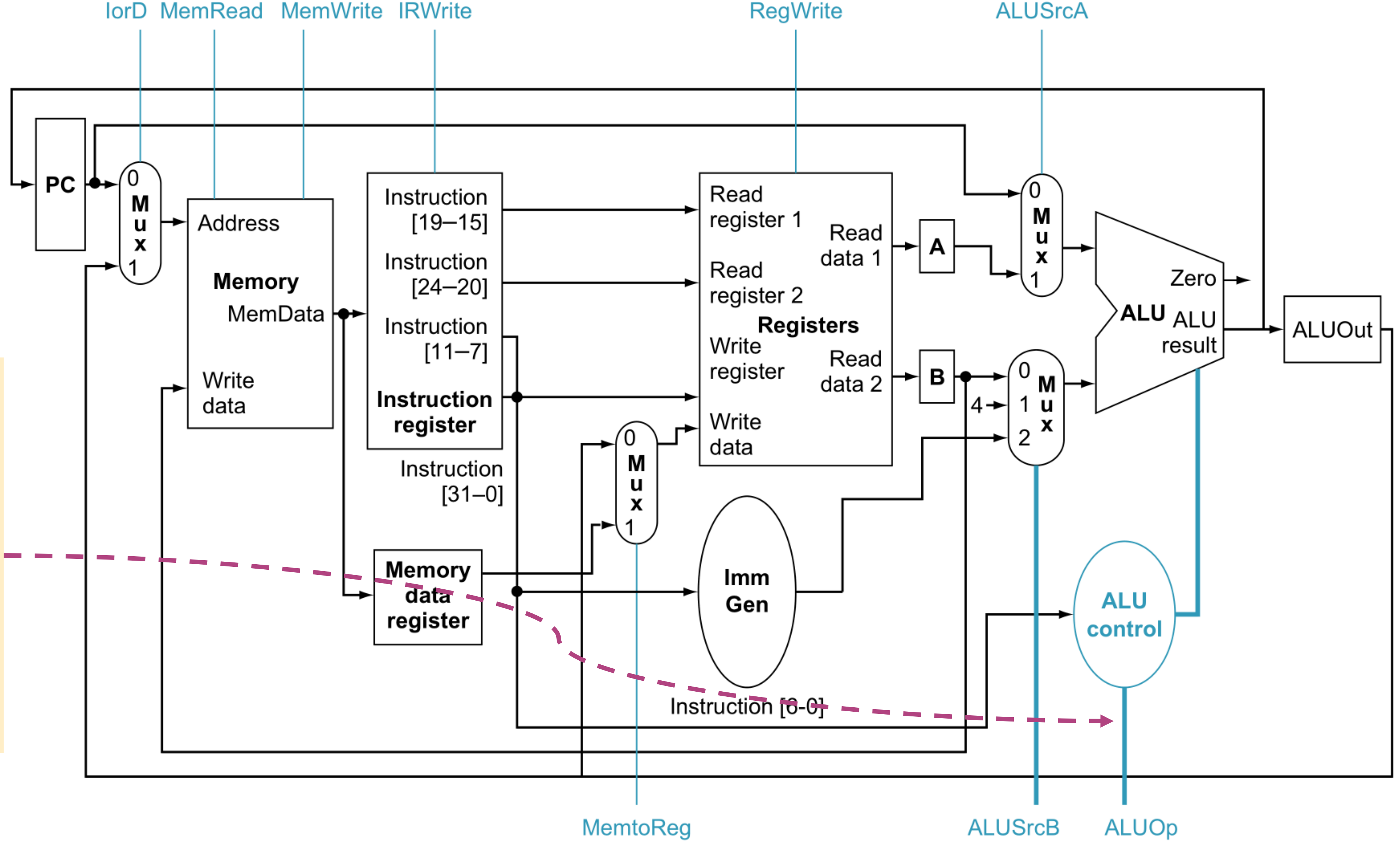
Determines whether the first ALU operand is register A or the PC

Determines whether the second ALU operand is register B, constant 4, or the sign-extended immediate

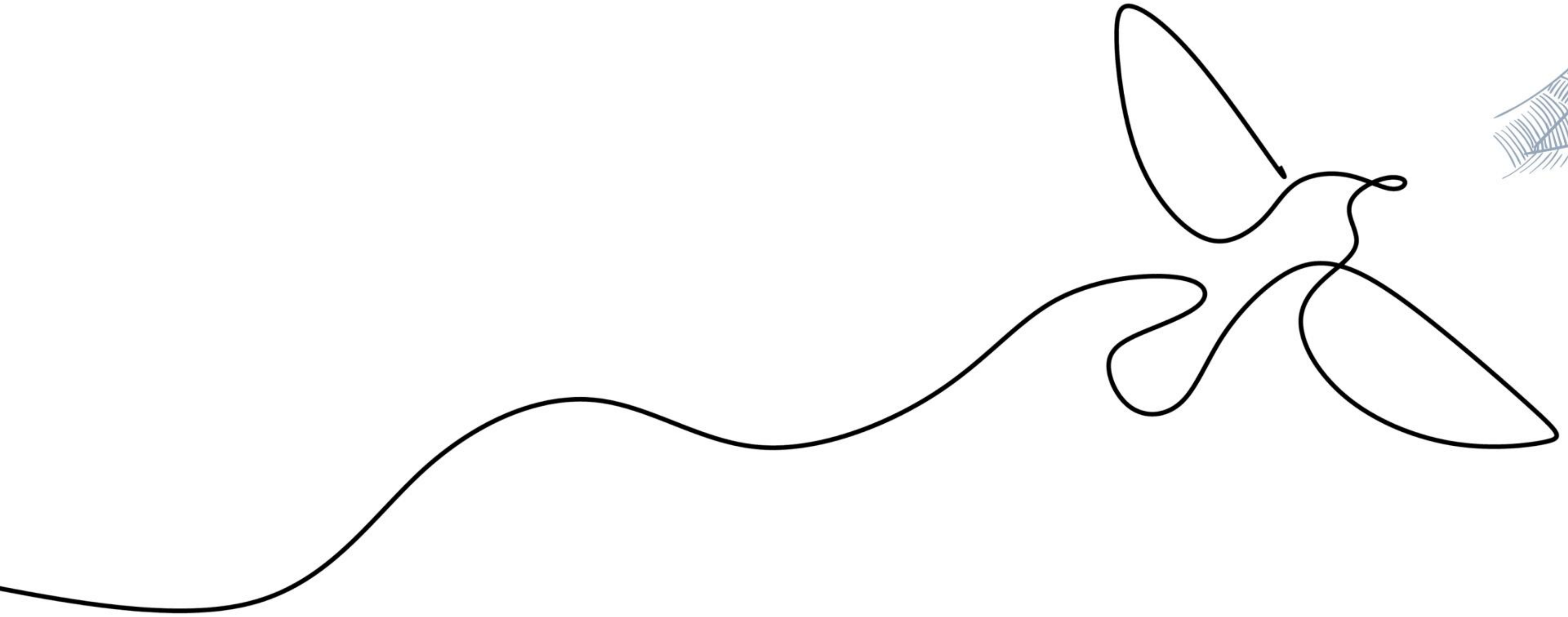


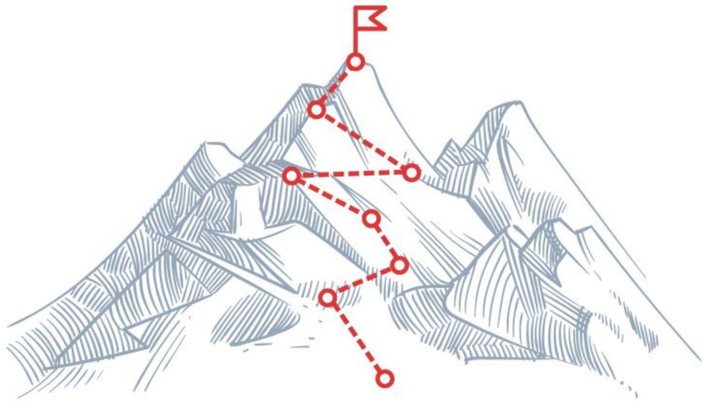
A Multicycle CPU

With Some Control Lines Shown



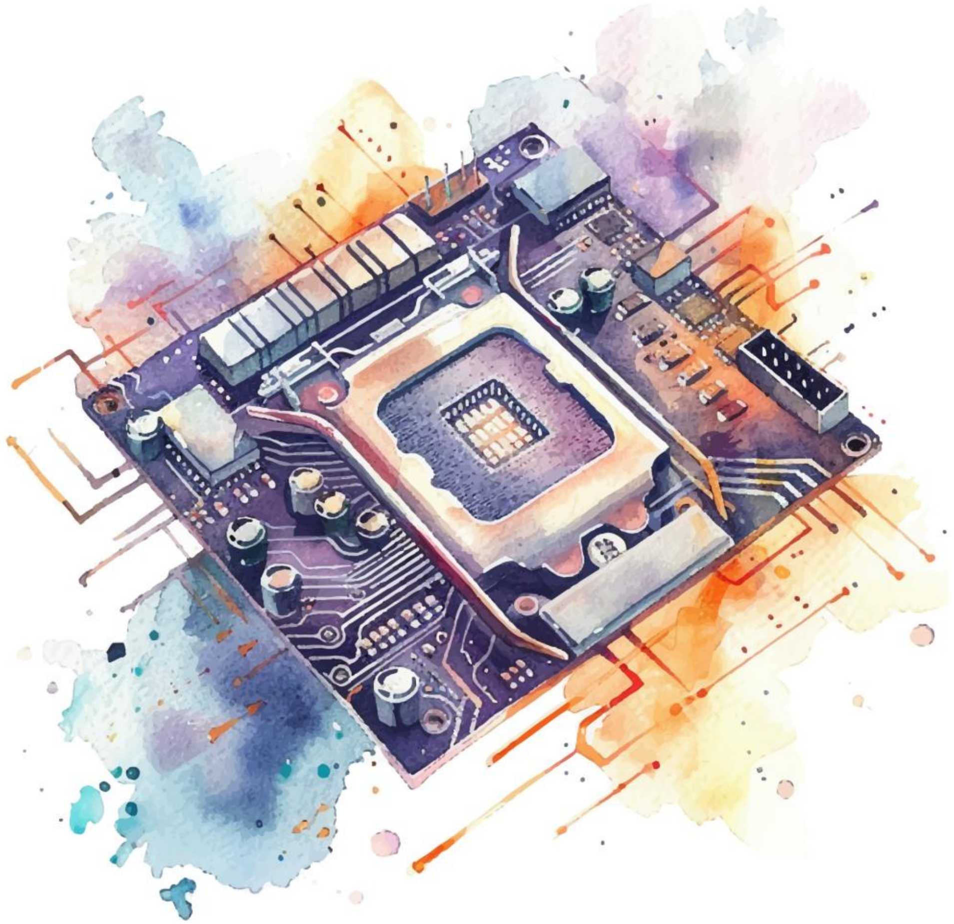
Determines if ALU performs addition (PC = PC+4, or branch target address), subtraction (comparing two registers for a branch if equal instruction), or another operation





Multicycle CPU

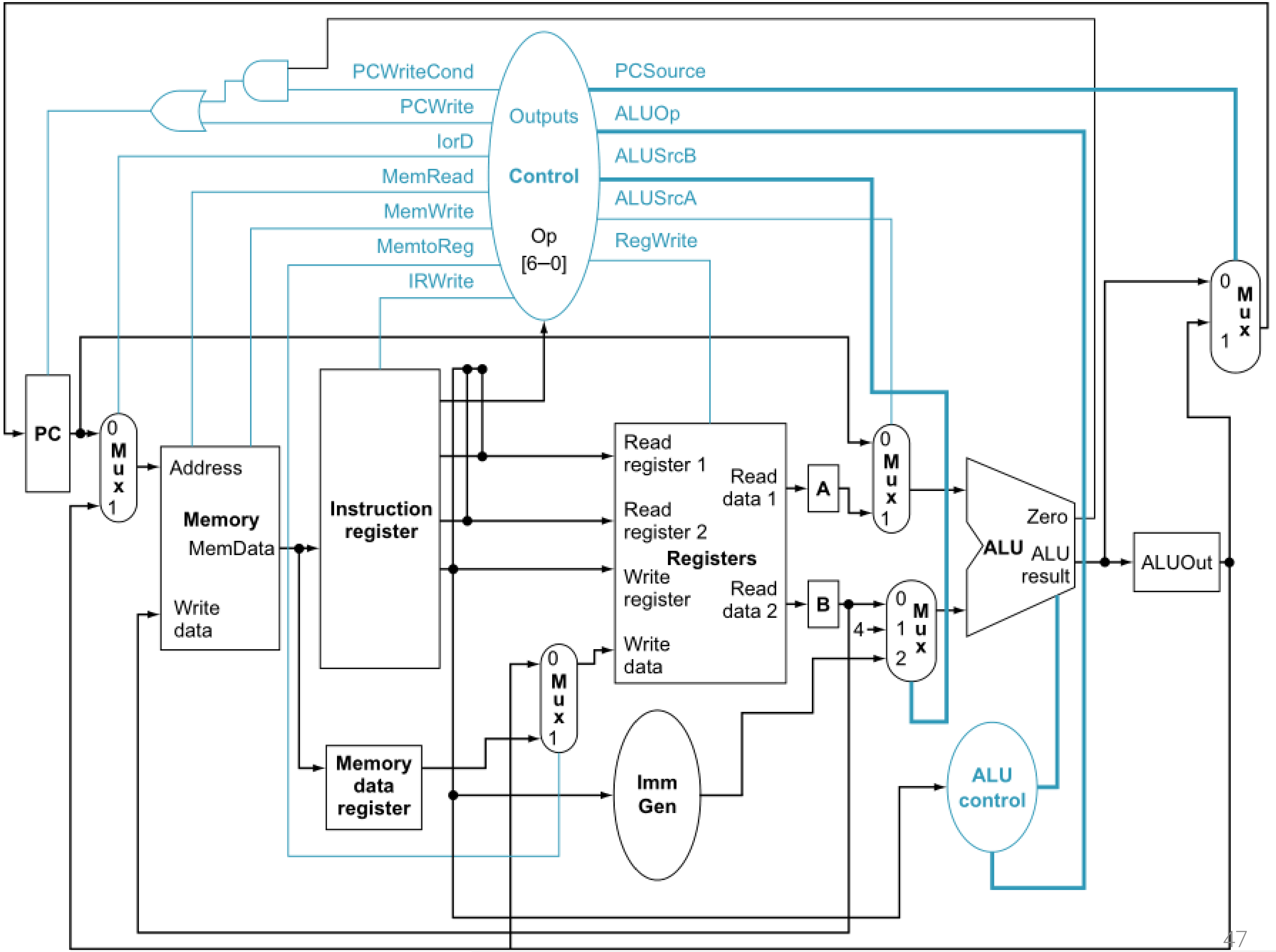
- Datapath + Control



© EnelEva / Adobe Stock

Multicycle CPU

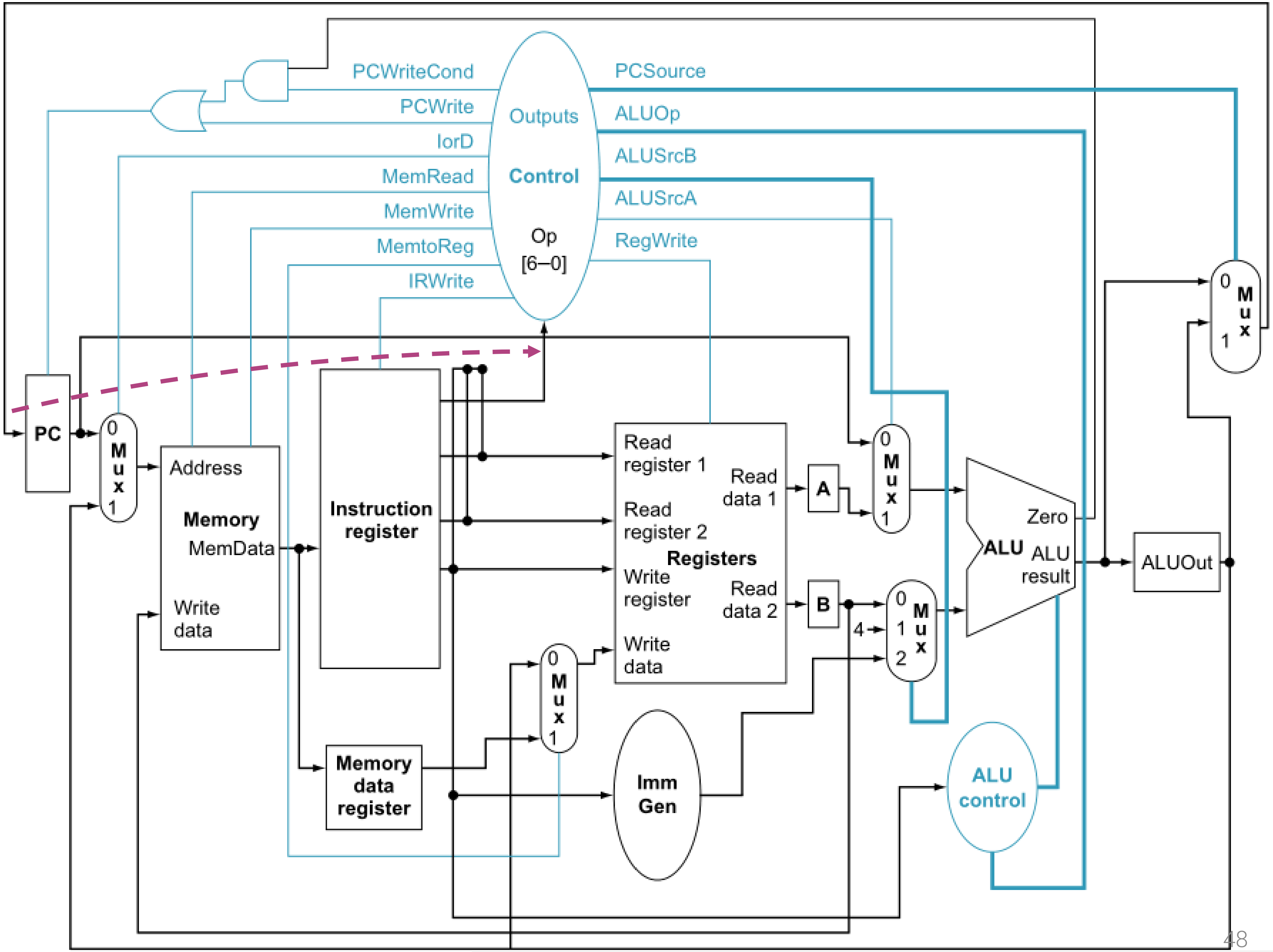
Datapath + Control



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

Multicycle CPU

Datapath + Control



The opcode field of the instruction (register IR) determines the operation of the ALU via ALUOp

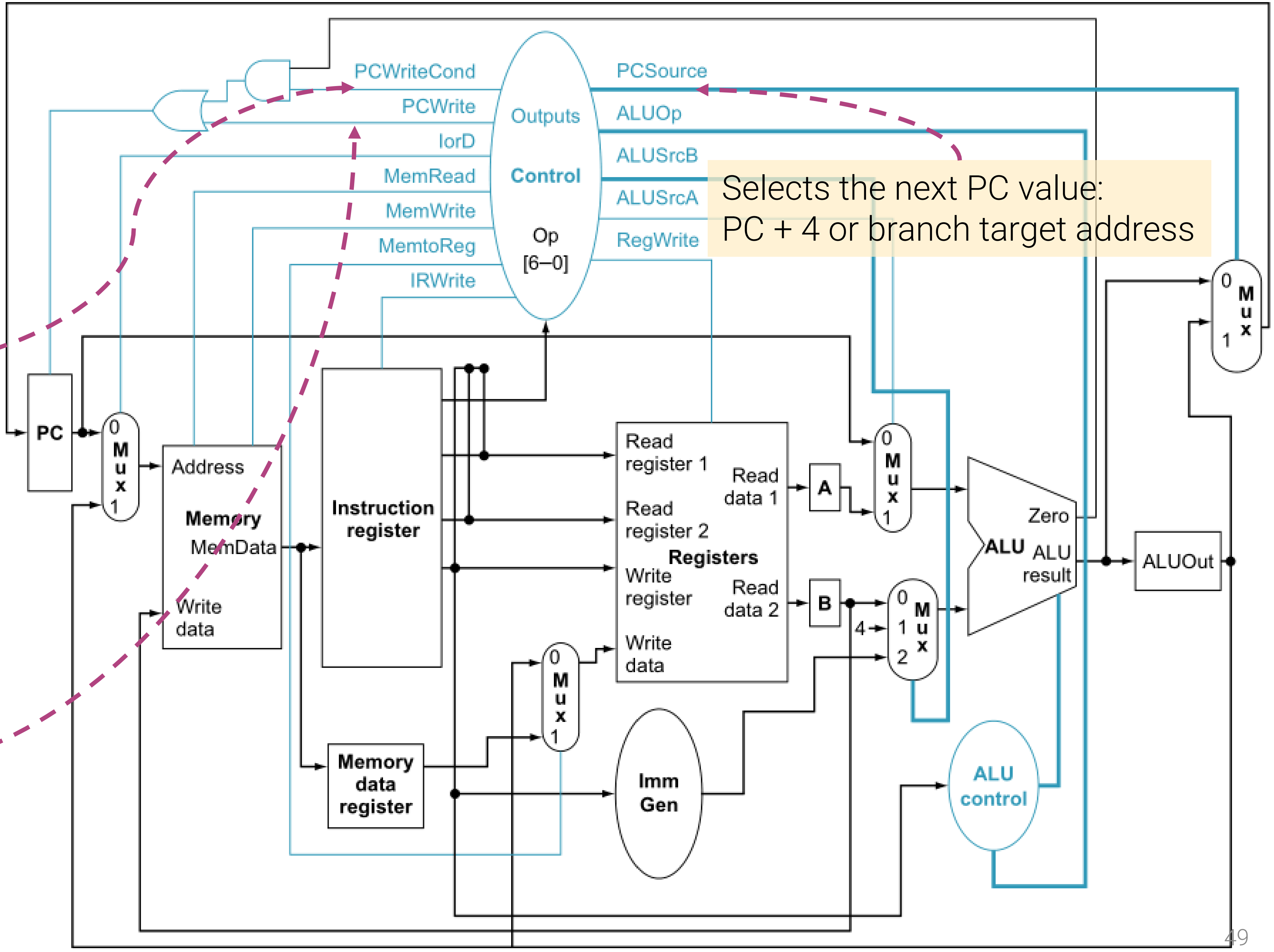
Figures from "Computer Organization and Design: RISC-V Edition," Patterson & Hennessy, 2nd Ed.

Multicycle CPU

Datapath + Control

Conditional PC write:
PCWriteCond causes a write of the PC if the branch condition is also true (if the Zero output from the ALU is also active).

Unconditional PC write:
PCwrite causes an unconditional write of the PC, during normal increment (PC = PC + 4)



Selects the next PC value:
PC + 4 or branch target address

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

Actions of the 1-bit Control Signals

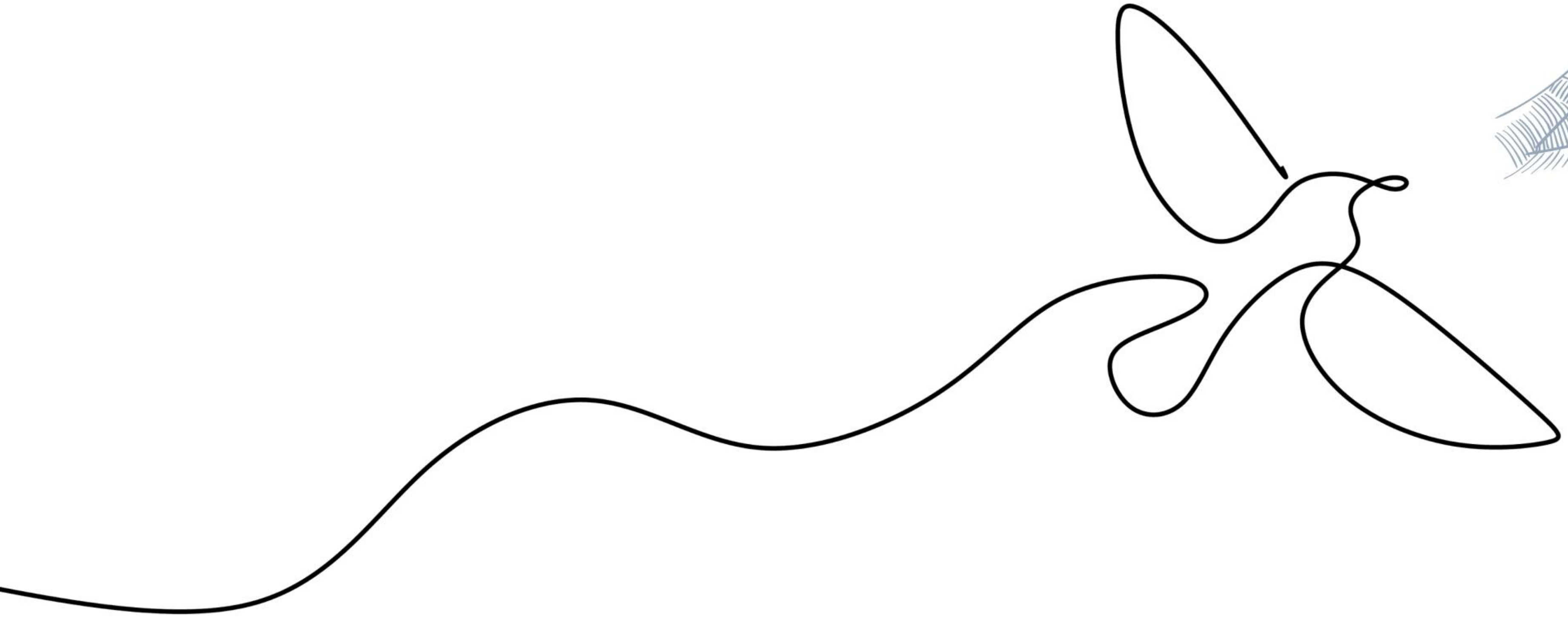
Summary

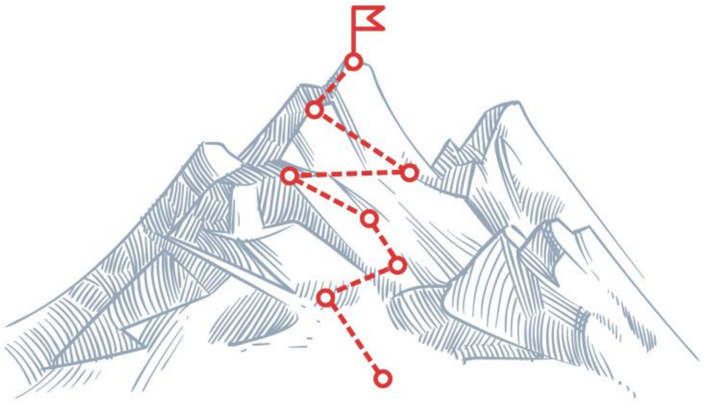
Signal name	Effect
RegWrite	If asserted, the register on the Write reg. input is written with the value on the Write data input
ALUSrcA	Determines whether the first ALU operand is register A or the PC
MemRead	If asserted, memory contents designated by the address input are put on the output
MemWrite	If asserted, 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 file Write data input comes from the ALUOut register or from the memory data register (MDR)
IorD	Determines if the address to the memory is supplied from ALUOut register or the PC
IRWrite	The output of the memory is written into the instruction register (IR)
PCWrite	The PC is written; the source is controlled by PCSource
PCWriteCond	The PC is written if the Zero output from the ALU is also active

Actions of the 2-bit Control Signals

Summary

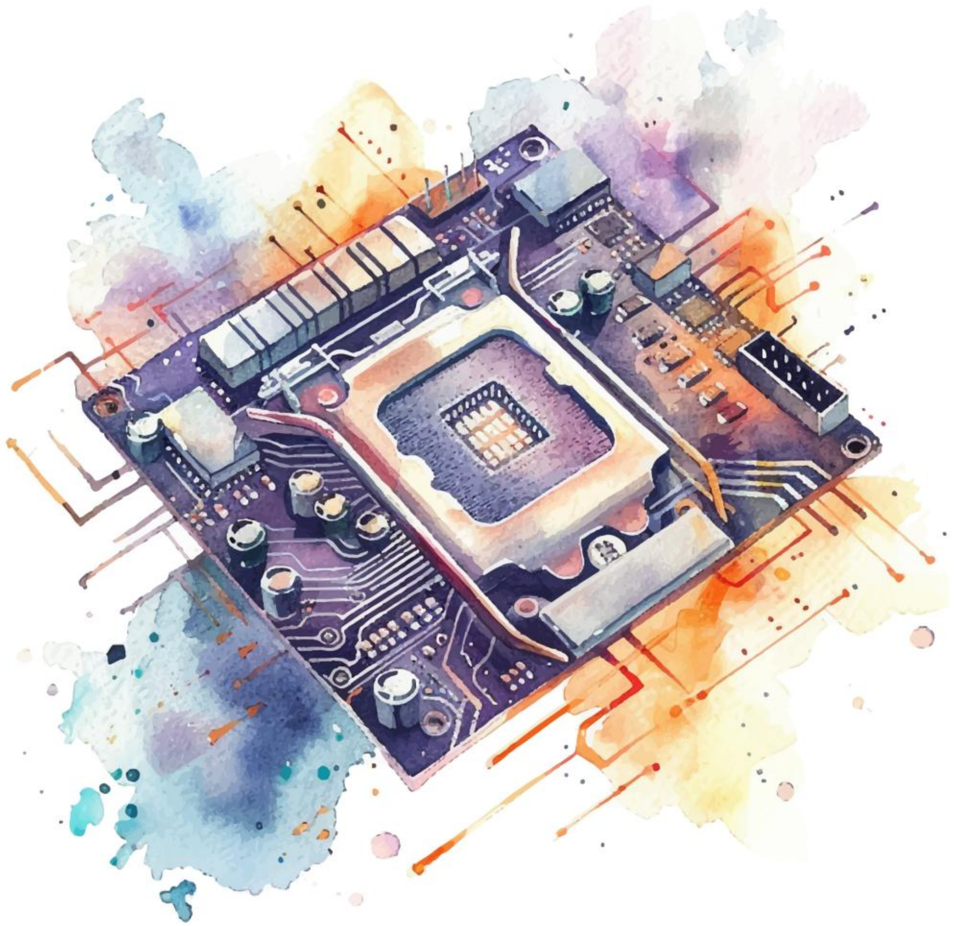
Signal name	Value	Effect
ALUOp	00	addition
	01	subtraction
	10	The funct field of the instruction determines the operation of the ALU
ALUSrcB	00	The second input to the ALU comes from the register B
	01	The second input to the ALU is the constant 4
	10	The second input to the ALU is the immediate generated from the instruction register (IR)
PCSource	00	Output of the ALU (PC+4) is sent to the PC for writing
	01	The contents of the ALUOut register (the branch target address) are sent to the PC for writing
	10	Additional functionality *(not covered in this example, ignore)*



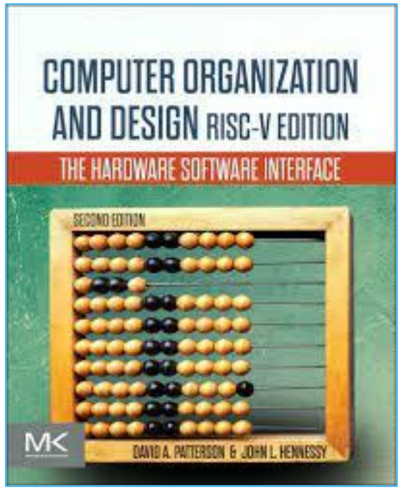


Breaking the Instruction Execution into Clock Cycles

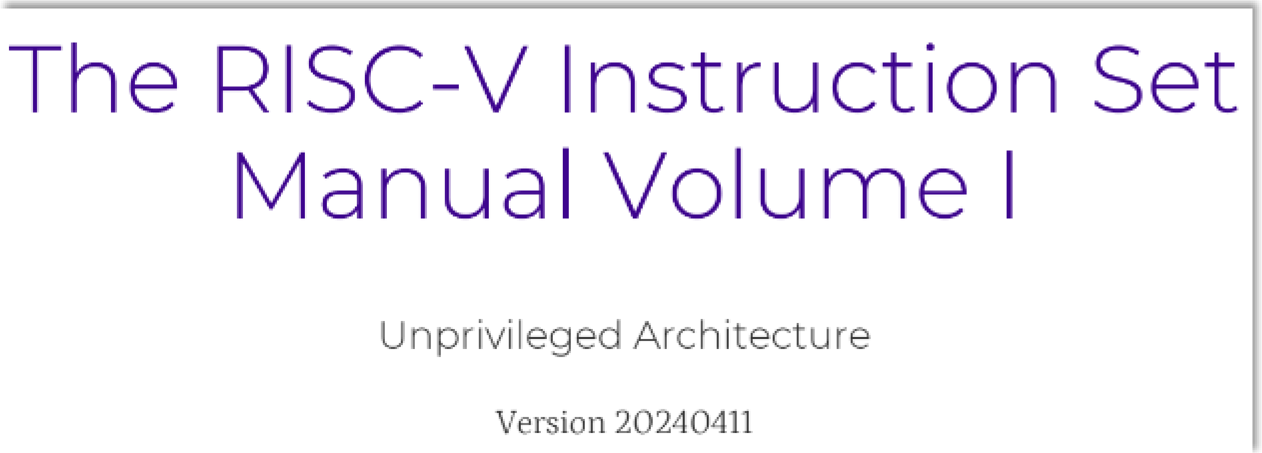
Next Lecture



Literature



- Chapter 4: The Processor
  - 4.5


The RISC-V Instruction Set Manual Volume I

Unprivileged Architecture

Version 20240411

Visit online: Link