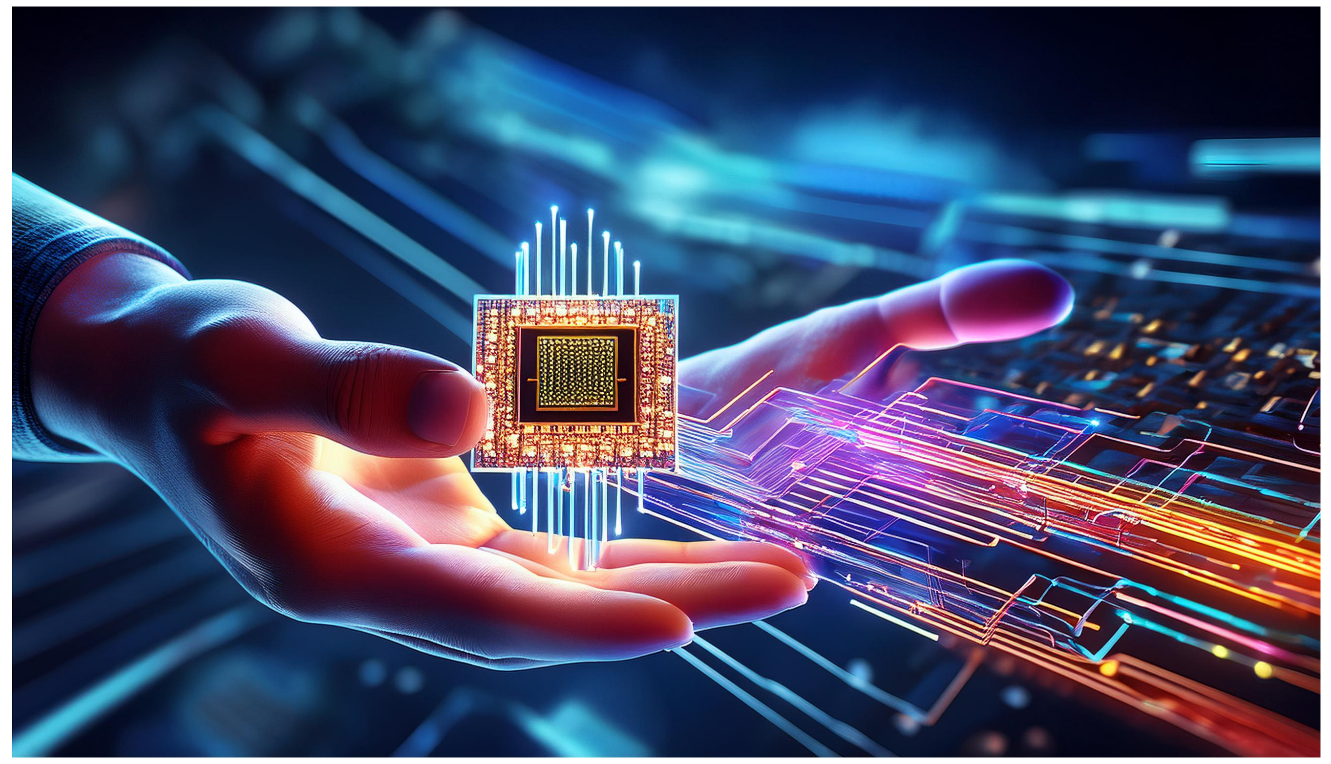
12.11.24 14:52

CSCI 250
Introduction to Computer Organisation
Lecture 4: Control Unit and Pipelines I



Jetic Gū
2024 Fall Semester (S3)

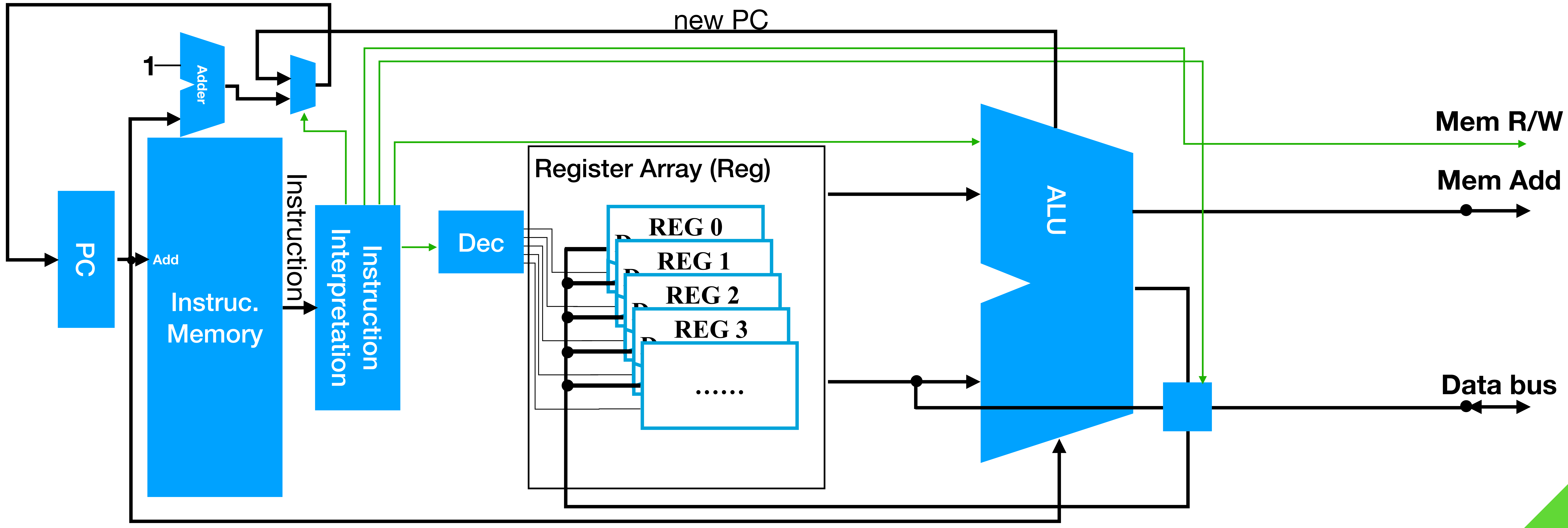
Overview

- Architecture: von Neumann
- Textbook: LCD: v4 9.8, 9.9; v5 8.8, 8.9; CO: 4.4
- Core Ideas:
  1. Single-Cycle Computers
  2. Multiple-Cycle Computers

Single-Cycle Computers

P1
Single-Cycle

MIPS Example CPU



- Programme Counter increases its value for every CLK, moving to the next instruction to execute

Review

MIPS Example CPU

- This particular MIPS implementation:
  - assumes that memory operations can be completed within 1 CLK cycle (e.g. 1 positive pulse);
  - assumes every arithmetic operations can be completed within 1 CLK cycle (e.g. 1 positive pulse);
  - assumes all Main databus operations (w/r) can be completed within 1 CLK cycle.
- This is called a **Single-Cycle Computer**

Concept

P1
Single-Cycle

Single Cycle Computer Issues

- Some Instructions will require more time
    - Can you think of examples?
- Some devices on the main data bus will need more time
    - Can you think of examples?
- How do we determine how fast this CPU can run?
    - Depends on the **worse-case delay path**

Technical

P1
Single-Cycle

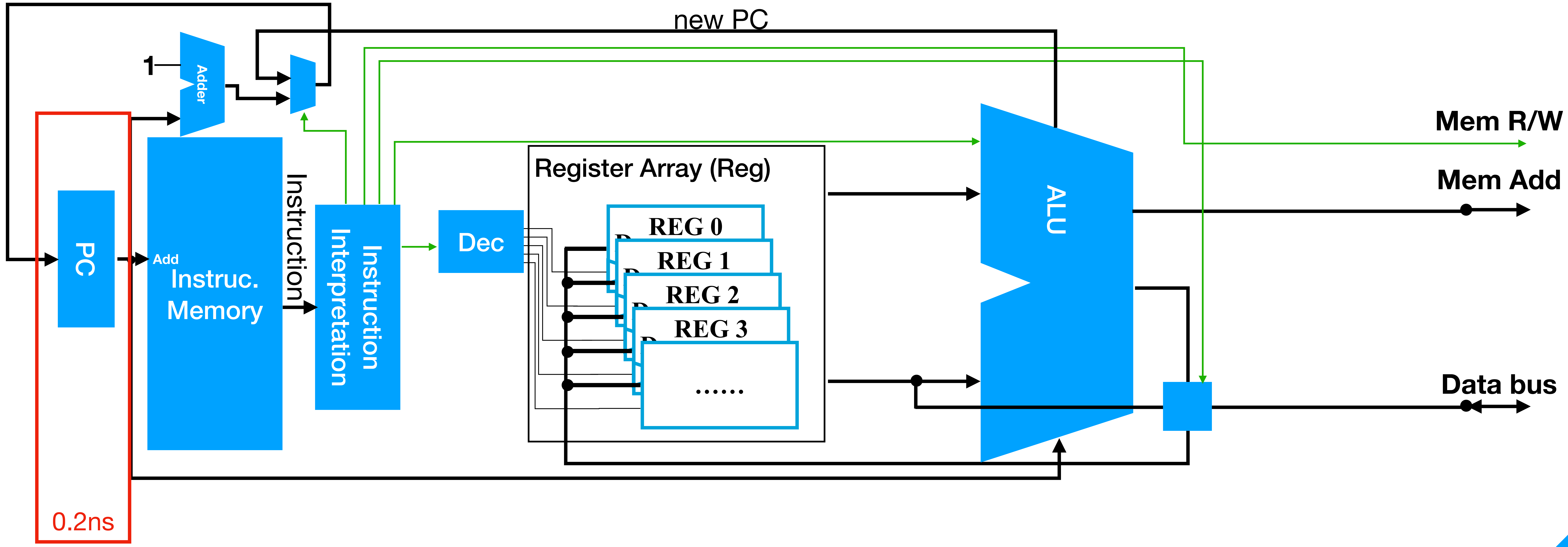
Worse-Case Delay Path

- Some Instructions will require more time
  - Can you think of examples?
- Some devices on the main data bus will need more time
  - Can you think of examples?
- How do we determine how fast this CPU can run?
  - Depends on the **worse-case delay path**

Technical

P1
Single-Cycle

Worst Case Delay Path

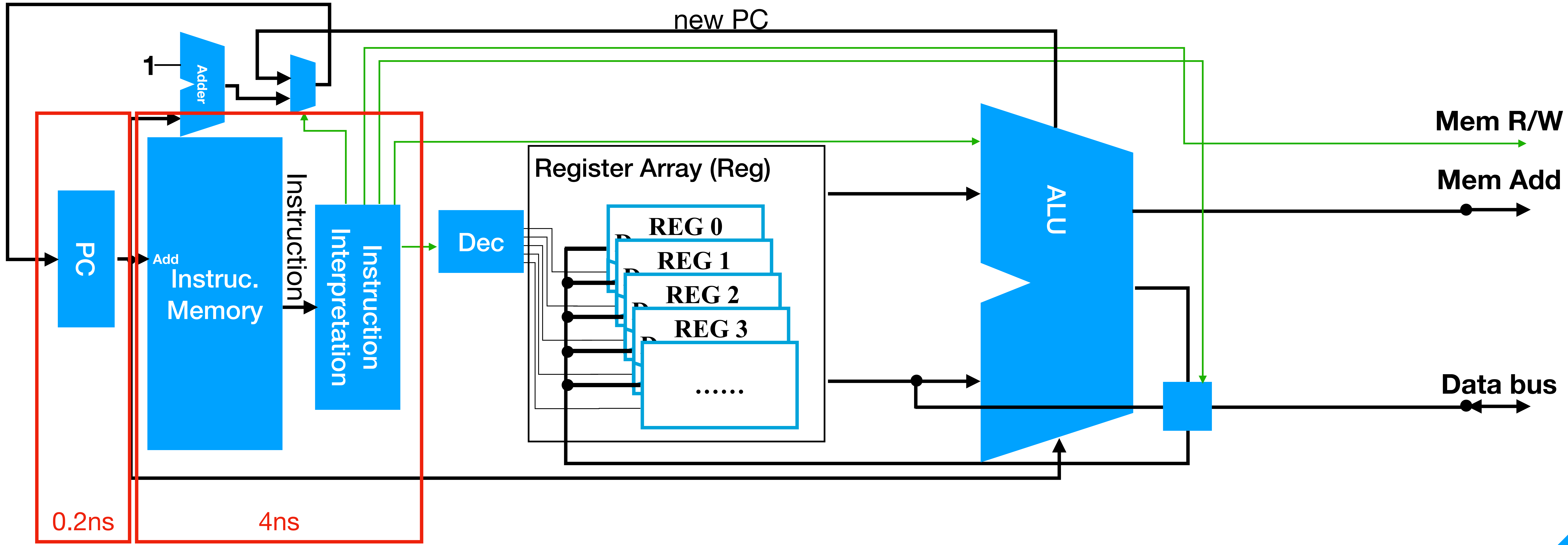


- The delay path starts, where the PC gets a new address through

Technical

P1
Single-Cycle

Worst Case Delay Path

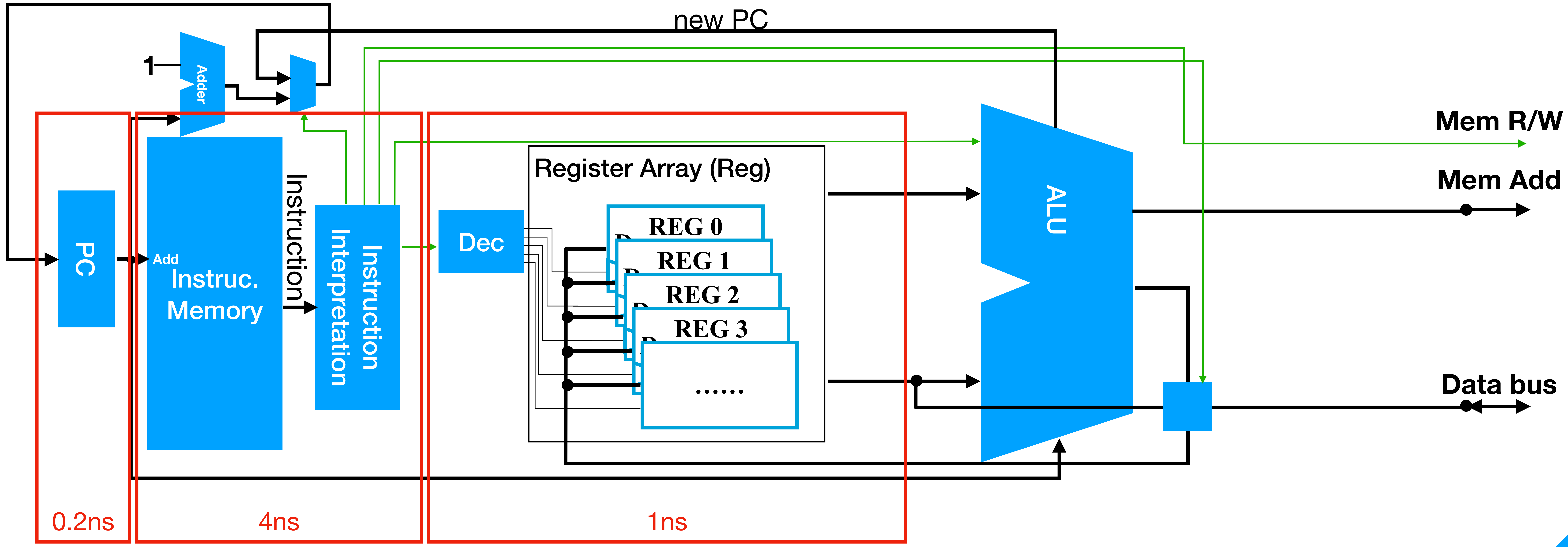


- Then, an instruction is retrieved from the **Instruction Memory**, and that gets decoded through the Instruction Interpreter (a.k.a. **Instruction Decoder**) (4ns)

Technical

P1
Single-Cycle

Worst Case Delay Path

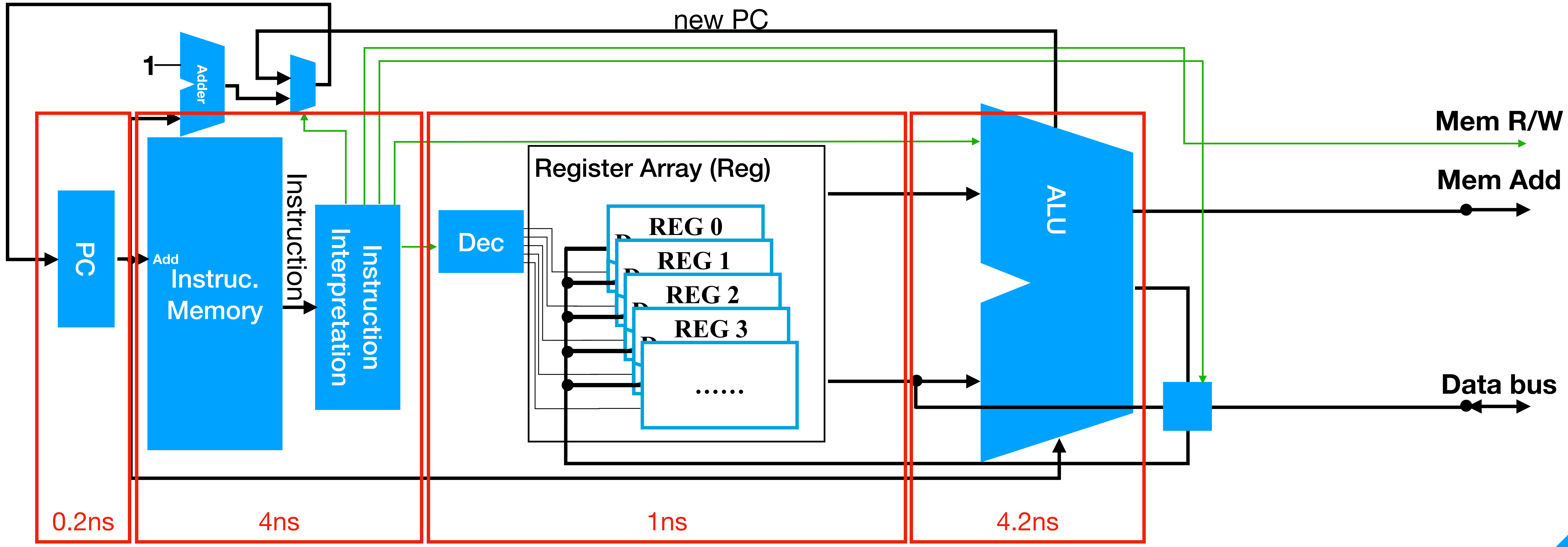


- After interpretation, the **registers are Read** (0.8ns), and fed through a few MUXs (0.2ns), that's 0.2+0.8 = 1ns

Technical

P1
Single-Cycle

Worst Case Delay Path

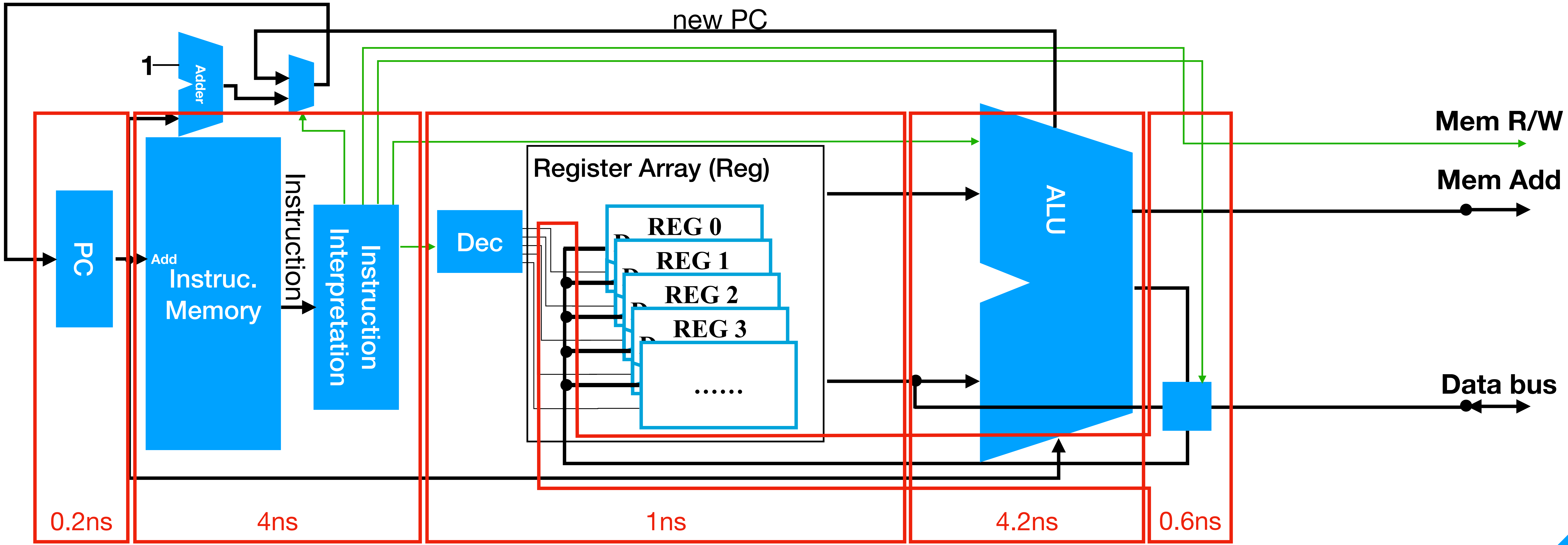


- The ALU performs the appropriate arithmetics, then uses another set of MUXs to select the output; or, the data bus handles some memory read/write. Let's be generous and say this costs 4.2ns

Technical

P1
Single-Cycle

Worst Case Delay Path



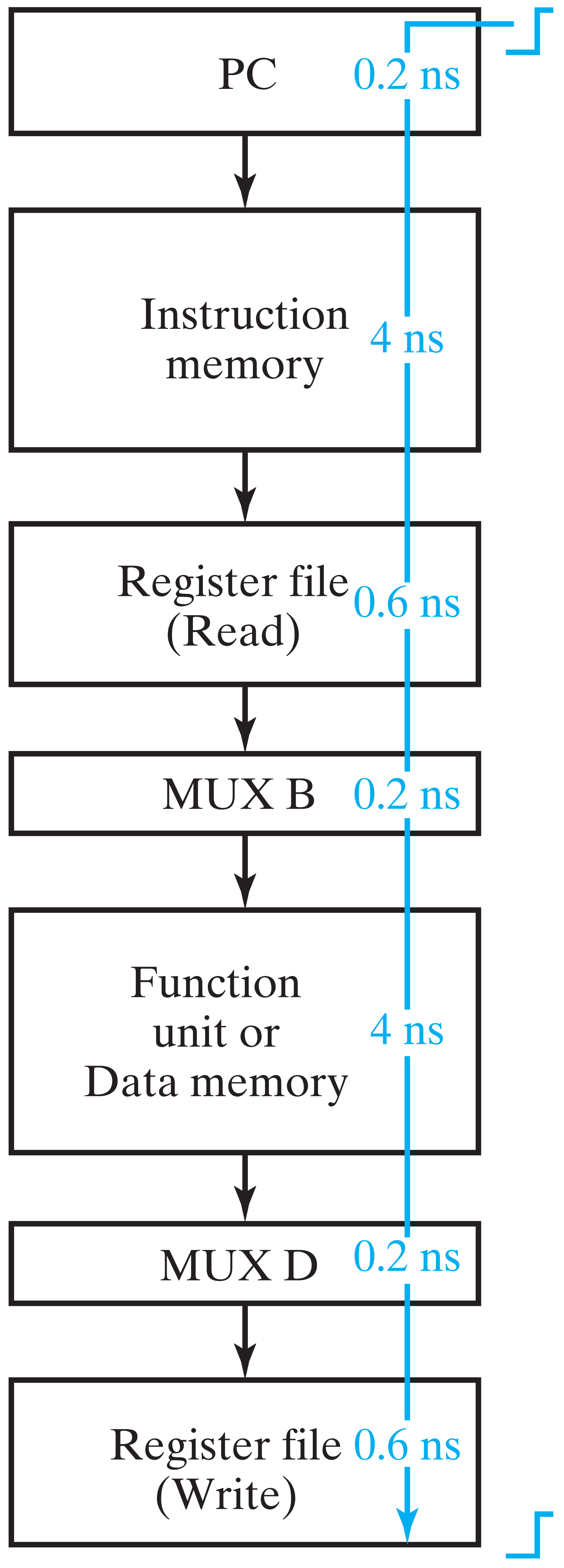
- Finally, the results are written back into the register array. Let's say this takes 0.6ns.

Technical

P1
Single-Cycle

Worst Case Delay Path

- This is called the Worst-Case Delay Path
- For every step of the way, we calculate the Worse-Case Delay for that step
- Add up all the delays across this entire path, we get the final WCDP time: 9.8ns
- 9.8ns = 9.8 x 10^{-9} s; This limits the max CPU speed at $1 \div (9.8 \times 10^{-9}) \approx$ 102MHz

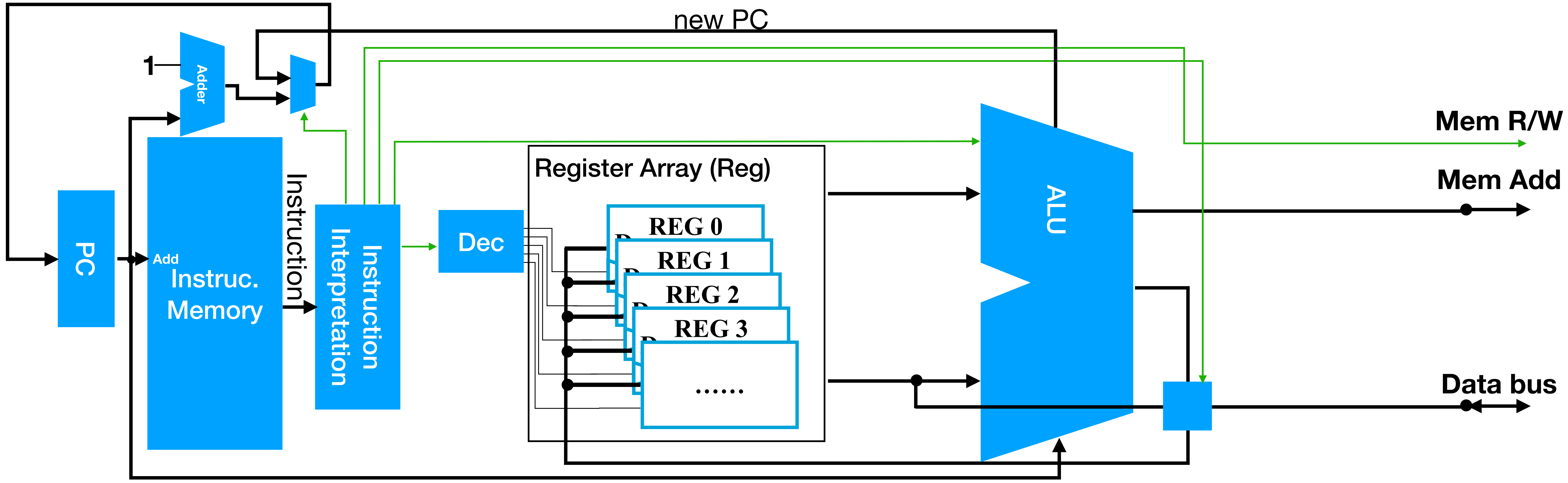


Concept

1. Figure 8.17

P1
Single-Cycle

Exercise 1: Worst Case Delay



- 1 Register: 0.2ns; Memory Read: 3ns; Memory Write: 4ns; ALU (Overall) 2ns; 1 MUX 0.2ns; Instruction Decoder 0.2ns

- With the above diagram and component-level WCD, provide a rough estimation of overall WCD

Exercise

How can we improve?

- Better Ingredients?
  Better(smaller) components will improve this

- Less?
  Less number of components will also improve this

- Is that all?
  Well, pipelining is the real solution. But we'll discuss that a small bit later.

Future

1. Figure 8.17

Multiple-Cycle Computers

Modern Computers are often Multiple-Cycle

- It means: it could take longer than 1 CLK cycle to execute one instruction
  Intel x86, AMD x86_64, PowerPC, ARM32, ARM64, the list goes on and on and on
- Why?
  - Memory is slower than Registers, SSDs and other devices are slower than memory
  - Some arithmetic function could be very complicated
    e.g. float multiplication and division
  - Also, we usually only get 1 main memory, not separate Instruction Memory and Data Memory
    - Why is this a big deal separating Single-Cycle CPUs from Multiple-Cycle CPUs?

P2
Multiple-Cycle

Multiple-Cycle Computer

- Occasions under which an instruction might take longer than 1 CLK to execute

1. ALU requires more time to execute some instructions

    - E.g. Float arithmetics, multipliers and dividers, usually take more than 1 cycle

    - Employ a counter to count the number of cycles needed for certain operations. When the counter reaches a certain value, we can move on

    - CPU halts all Write operations of registers (including PC and IR) until the ALU gives an OK

Concept

Multiple-Cycle Computer

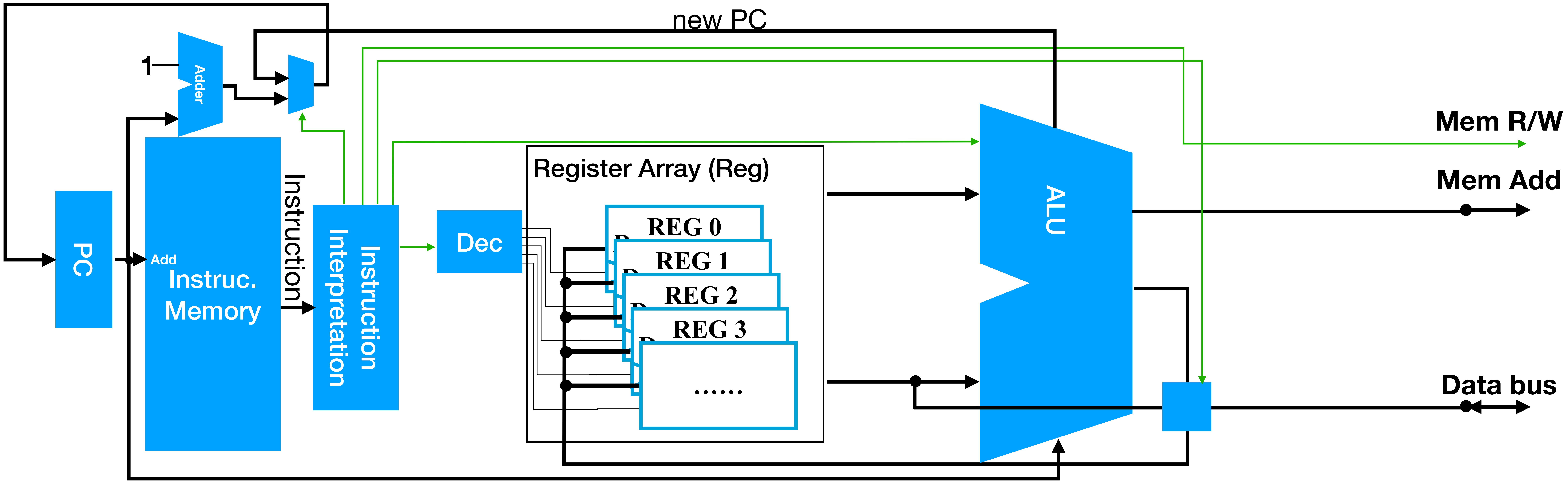
- Occasions under which an instruction might take longer than 1 CLK to execute

  2. Main data bus requires more time to transfer data

     - Devices on the main data bus could vary. E.g. Main memory, GPU, Hard drive, etc.
     - Each devices can have their own mechanisms for informing the CPU
     - CPU halts all Write operations of registers (including PC and IR) until the ALU gives an OK

P2
Multiple-Cycle

The Control Unit



- Previously, we've discussed the **Instruction Interpretation** module. In modern CPUs, this is usually also referred to as an **Instruction Decoder**

Review

The Control Unit

Instruction Interpretation $\neq$ Control unit

- In Single-Cycle CPUs, the **Instruction Decoder** is the **Control Unit**. It is **Combinational**.
- In Multiple-Cycle CPUs, the **Control Unit** is **Sequential**, and must also include:
  - a temporary place to hold the current Instruction
    e.g. the **Instruction Register**[1], or the **Instruction Queue**[2], etc.
  - a **Control Logic Unit**, which contains
    - A bunch of counters and status ports that connects to the ALU and Main data bus controller
    - A counter for certain operations whose required CLK cycles are known
    - Others, depending on the instruction set

1. Used commonly in MIPS. Modern CPUs with Pipelines often has more complex structures than just an IR.
2. Used in for example the 8086 CPU.
3. The CLU is something more unique to the LCD textbook, it is not discussed in CO

Concept

P2
Multiple-Cycle

The Control Unit Instruction Register

- IR is a register that holds the current instruction, as it is executed
  - Why isn't it required in Single-Cycle CPUs?
  - It has a **IL** signal for enabling "Instruction Load", this signal comes from the **Control Logic Unit**, and is part of the **Control Word**

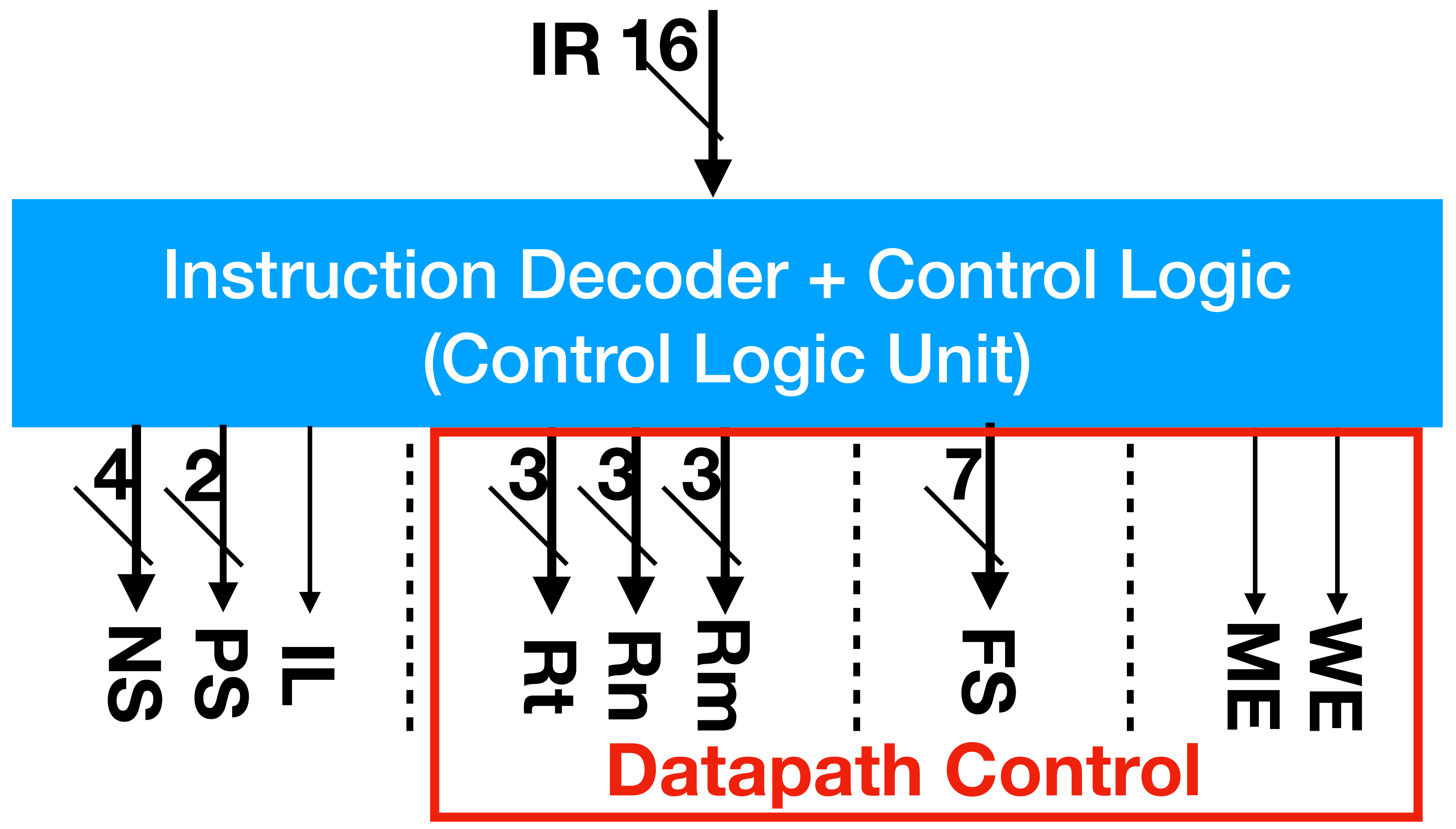
Technical

1. Control Word is something that's used in Single-Cycle CPUs, Multiple-Cycle CPUs, as well as Pipelined CPUs.

The Control Unit
Programme Counter

- During the execution of an instruction in a Multiple-Cycle Computer
  - The **Programme Counter**’s value must **NOT change with CLK** when we don’t want it to
  - When should it change?
    - When the control unit has determined that the current instruction has been completed, the control unit will **enable** the PC to have its value changed, either by performing increment or by going to another address using **Jump/ Branching**
  - Solution: having special control signals **PS** from the **Control Logic Unit**

An Example Control Logic Unit



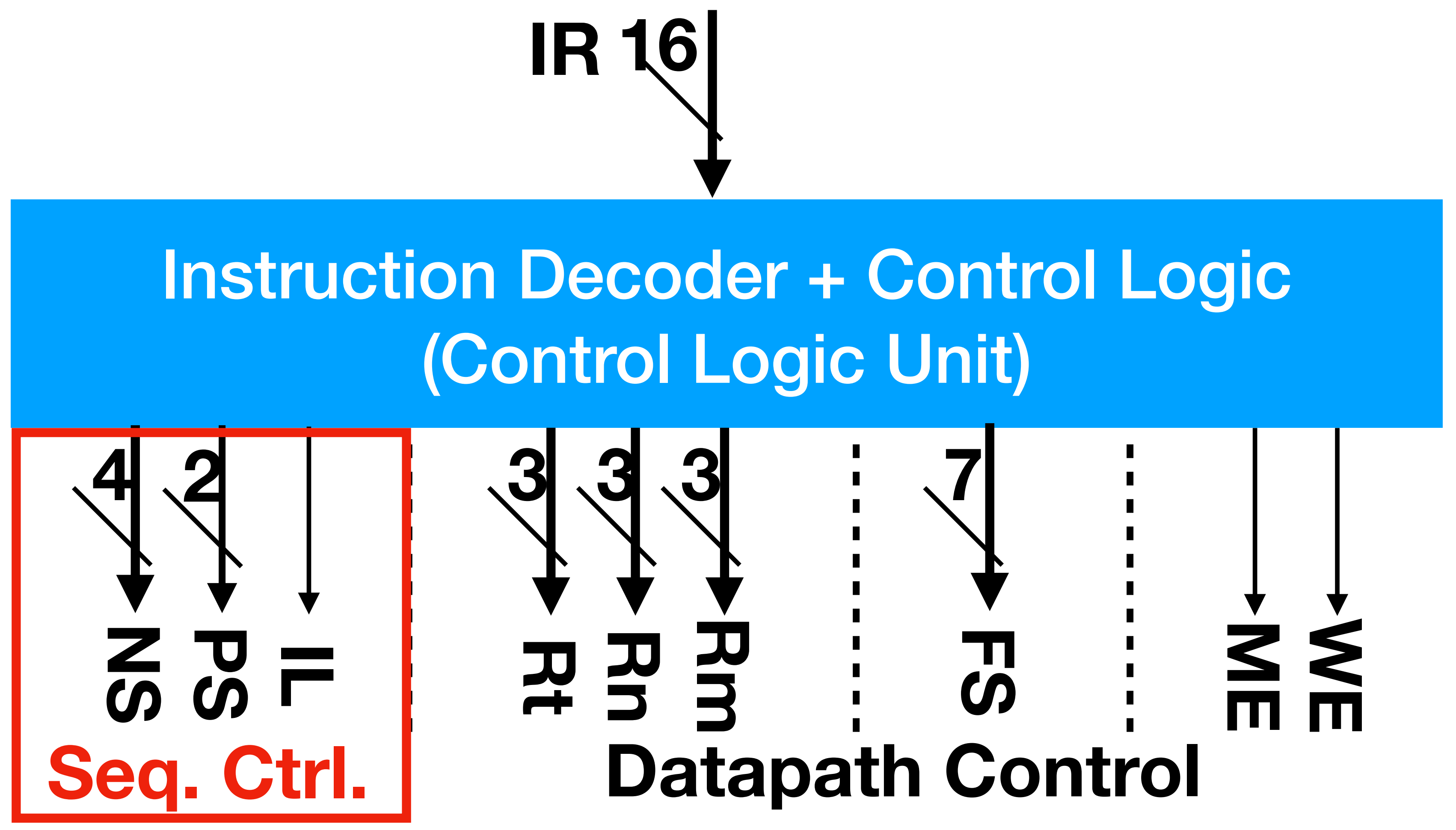
- The combined output of this Control Logic Unit is called the **Control Word**, it control the sequence of microoperations
  - **Datapath Control** signals are **combinational** interpretation of the Instruction, similar to the **Instruction Decoder** in Single-Cycle CPUs
    - **Rt, Rs, Rd**: connects to the register array; Ref. LS9
    - **FS**: function selection, connects to the ALU for function selection
    - **ME**: Memory Chip Enable; WE: Memory Write Enable; Ref. LS5

1. **Datapath Control** here NOT identical with the one in LCD textbook, so it's more in line with LogicWorks memory and ARM instructions

Example

P2
Multiple-Cycle

An Example Control Logic Unit



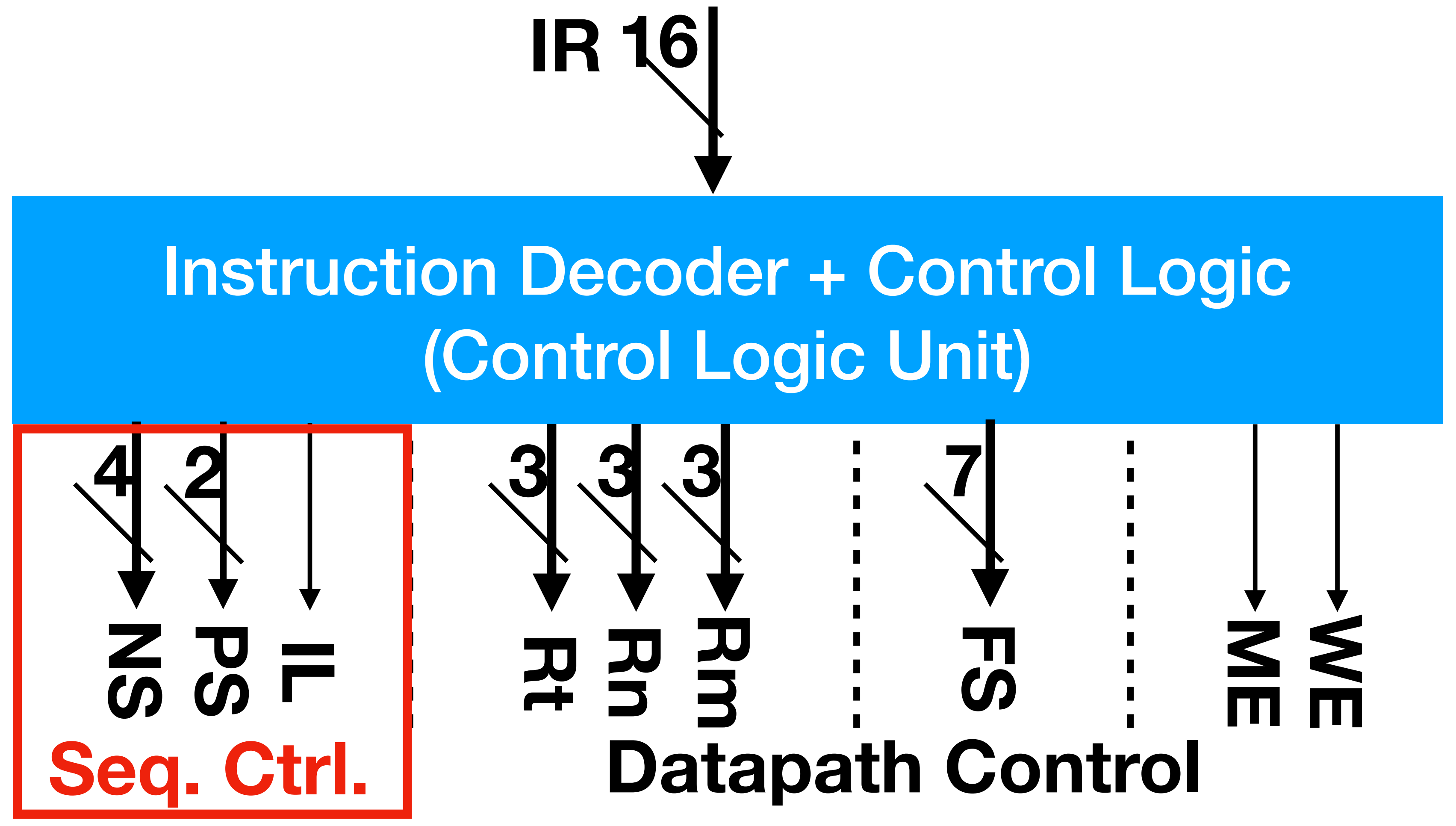
- Sequence Control: control the execution of multi-cycle instructions
  - **NS**: a 4bit counter's internal state; This is often implemented using flip-flops
  - **PS**: Programme Counter Control
  - **IL**: Instruction Load
    Controls whether to move on to the next instruction and reset the counter

1. Seq. Ctrl. here mostly consistent with LCD textbook

Example

P2
Multiple-Cycle

An Example Control Logic Unit



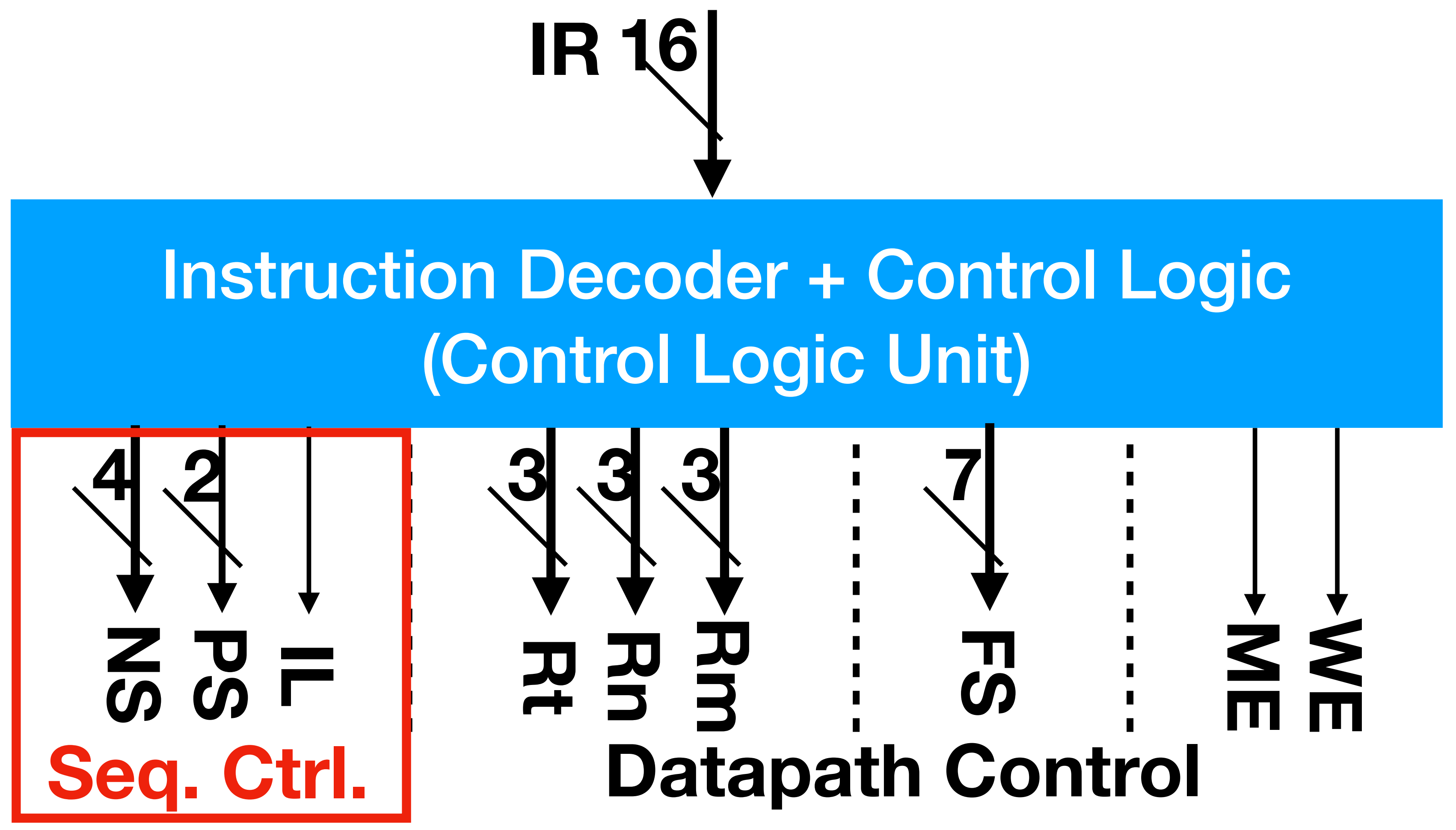
Seq. Ctrl.	Meaning
PS = 00	Hold PC
PS = 01	Increment PC, meaning PC += 1
PS = 10	Conditional Branch; PC receives new value from ALU when condition is met
PS = 11	Jump; PC receives new value from ALU
IL = 0	Hold IR
IL = 1	Load new instruction as specified by PC

1. Seq. Ctrl. here mostly consistent with LCD textbook

Technical

P2
Multiple-Cycle

An Example Control Logic Unit



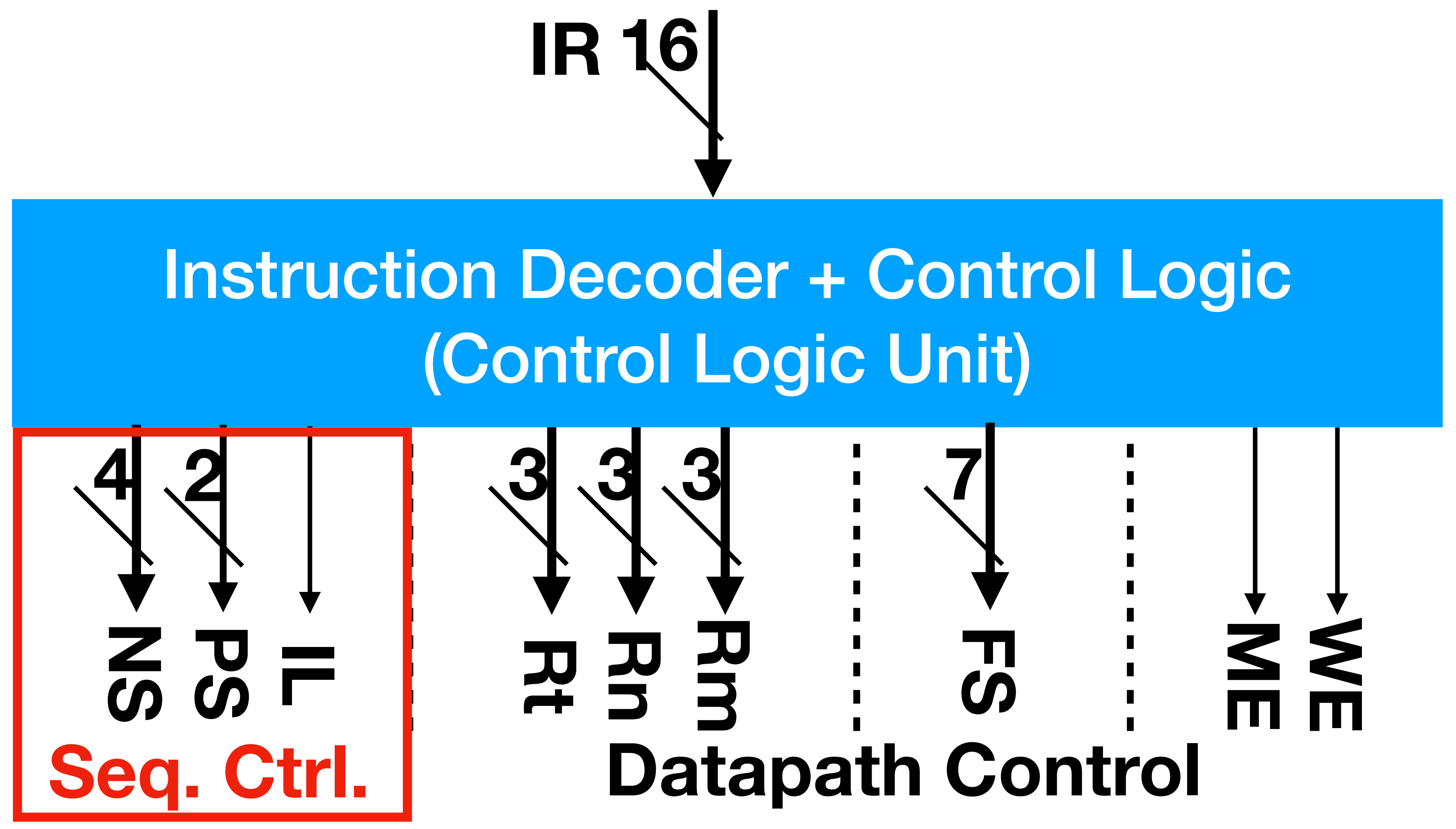
- **NS**: next state
  - Every CLK cycle, NS increases by 1
  - Different NS values for different instructions corresponds to different PS/IL values
  - After the execution of an instruction: **NS** is reset to 0; **PS** to 00 (Hold); **IL** to 1 (Load)
    This allows a new instruction to be loaded from the **Main Memory** to **Instruction Register**

1. Seq. Ctrl. here mostly consistent with LCD textbook

Technical

P2
Multiple-Cycle

An Example Control Logic Unit

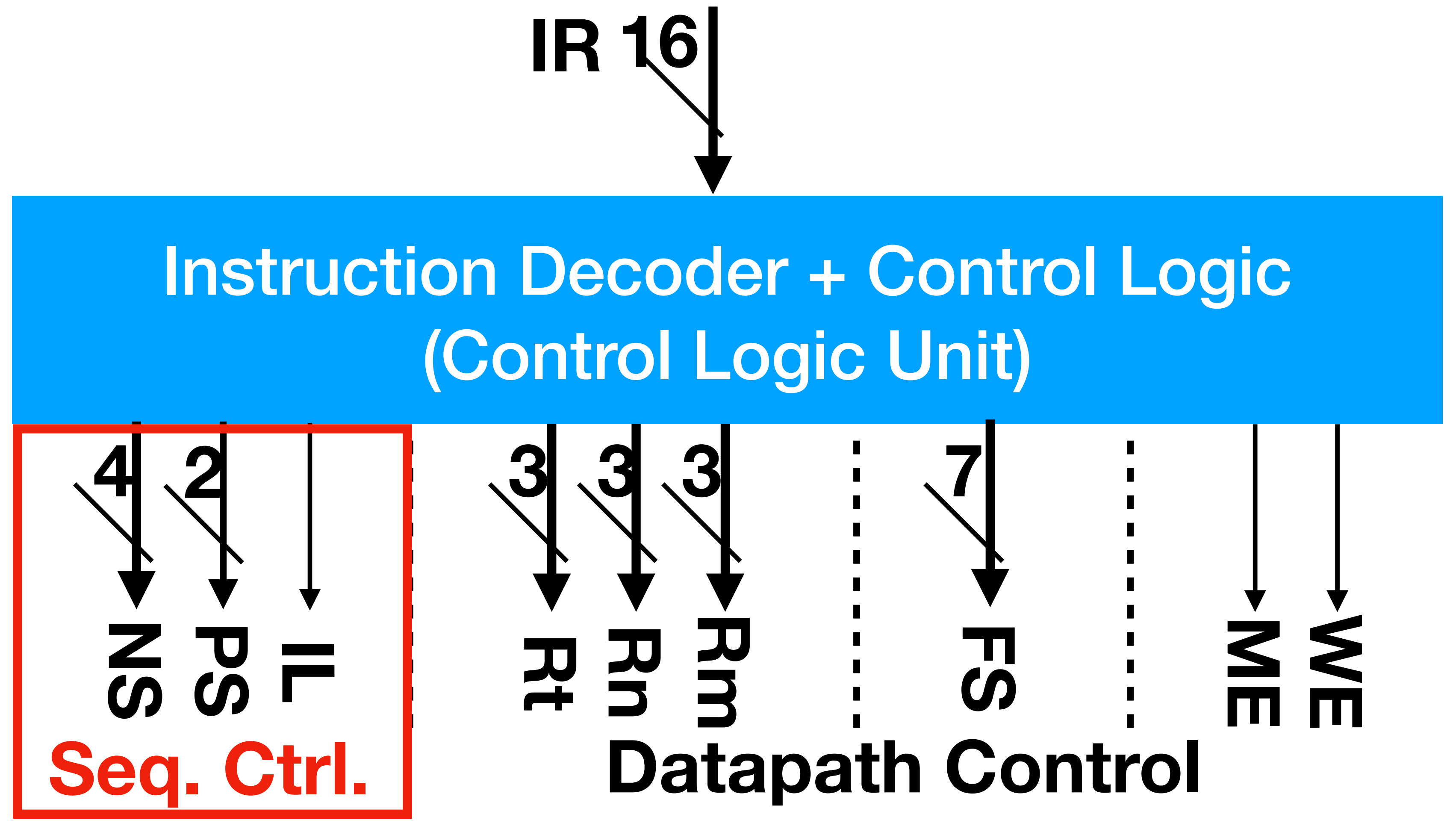


- **NS = 4**
  - Let’s say **Instruction Load** takes 4 cycles, **PS**/**IL**’s values are **Held** from NS=0 to NS=3
  - After **NS** reaches 4, the new instruction is **Fetched**
    - **IL** returns to 0 so **IR** holds the new instruction, until the execution of such is completed
    - **PS** can go to 01, so PC = PC + 1 as per ARM specification. This lasts 1 cycle.
    - We now perform **interpretation/decoding** for the newly acquired instruction. This can take 1 cycle

1. Seq. Ctrl. here mostly consistent with LCD textbook

Technical

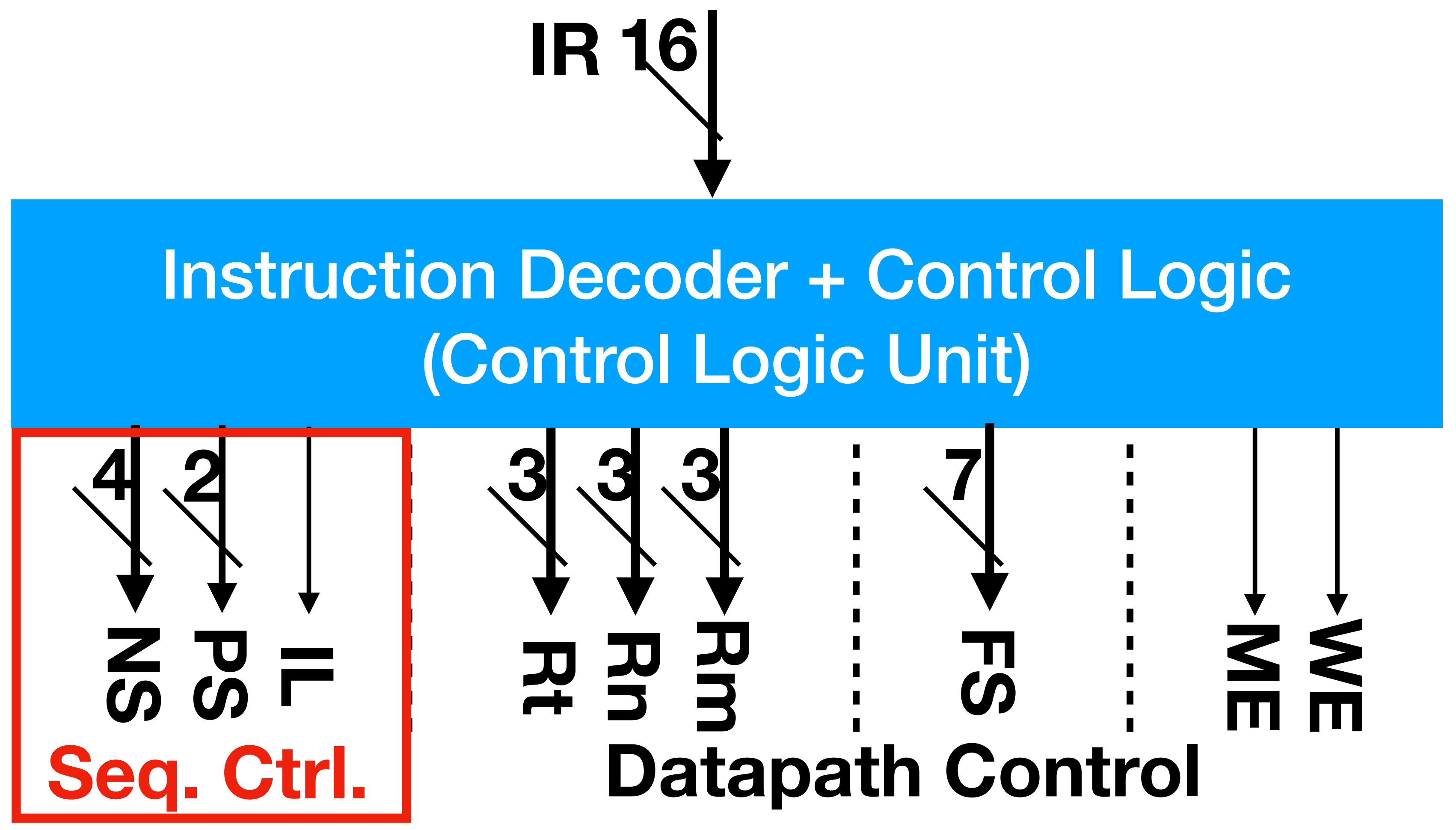
An Example Control Logic Unit



- **NS = 5**
  - **NS** reaches 4, the new instruction is **Fetched**
  - **NS** reaches 5, PC has increased by 1, the new instruction is decoded, ready for execution
    - **IL** remains at 0
    - **PS** can go to 00
    - Say we have a **register arithmetic operation**, this takes 1 cycle

1. Seq. Ctrl. here mostly consistent with LCD textbook

An Example Control Logic Unit



- **NS = 0**
  - **NS** reaches 4, the new instruction is **Fetched**
  - **NS** reaches 5, PC has increased by 1, the new instruction is **Decoded**, ready for execution
  - **NS** reaches 6, **t**he instruction is **Executed**, NS resets to 0, IL set to 1, PS set to 00, start loading the next instruction

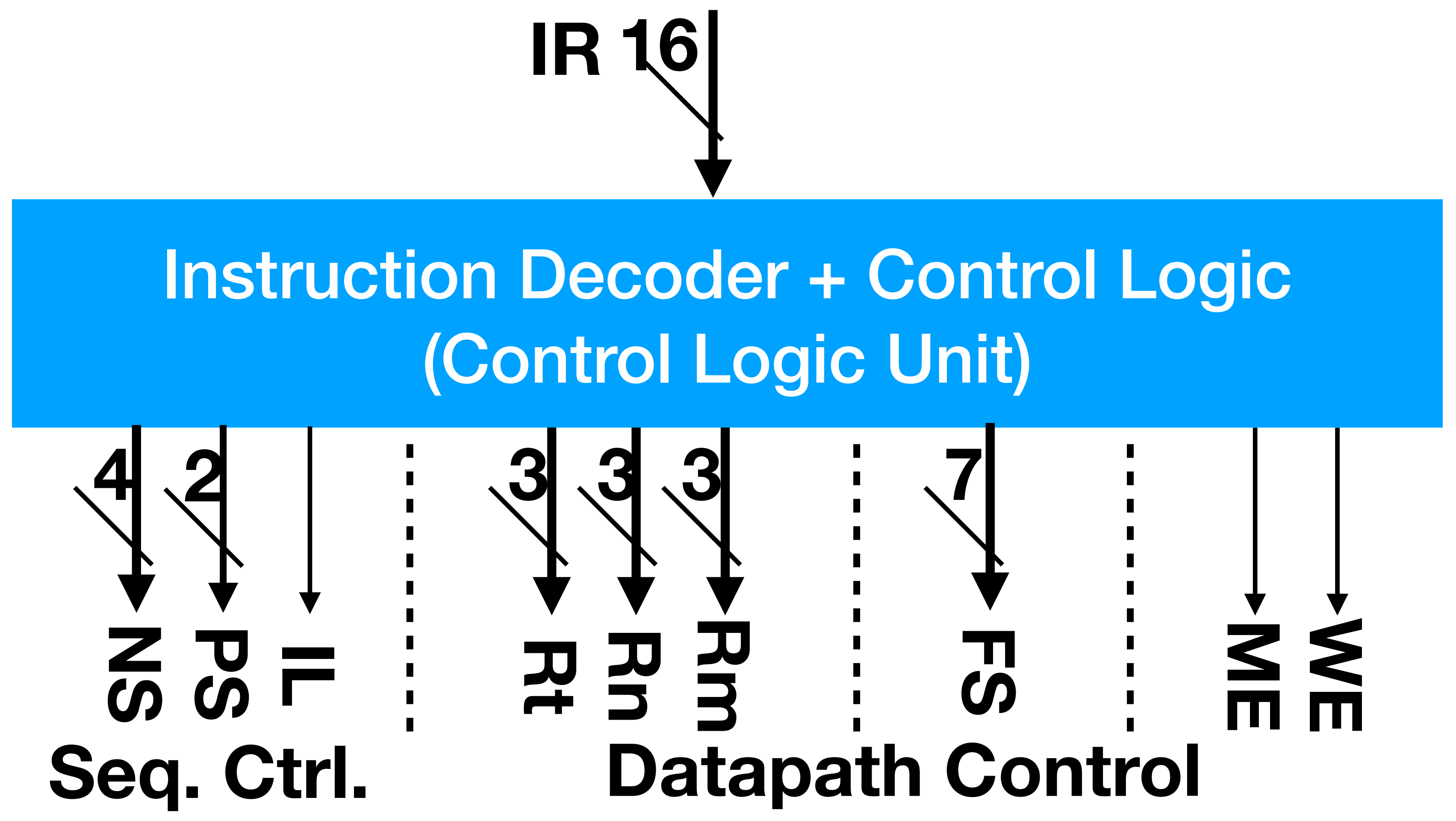
1. Seq. Ctrl. here mostly consistent with LCD textbook

3 Primary Stages of CPU Execution[1]

- **Fetch**
  - The fetching of information from either the Main Memory or Register
- **Decode**
  - The interpretation of the instruction, generating the appropriate **Datapath Control**
- **Execute**
  - Actual execution of the instruction
    After which the **NS** counter in the previous example should **reset**, so do the other Sequence Control signals

1. IBM Stretch Project, 1956-1961

Exercise 2: Control Word



- Are these **datapath control signals** really enough?
- At which stage will the **datapath control** signals be available? What values should they be?
- Assume a Multiple-Cycle CPU, with other I/O devices. Should the **Control Unit** require other input ports?

Exercise

Stages of ARM CPUs

- ARM largely follows the Fetch/Decode/Execution idea, with
  - **ARMv7**: exactly these 3 stages
  - **ARMv9**: 5 stages, but each stage is much more efficient than ARMv7, so the overall efficiency is actually higher
  - **ARMv10**: 6 stage, but each stage is much more efficient than ARMv7/v9, so the overall efficiency is almost double that of ARMv7
  - **ARMv11**: 8 stages, fetch for example is divided into two; even faster than v10
- Why does having more stages give faster execution?
  Because of **Pipelining**.

1. https://developer.arm.com/documentation/ddi0333/h/introduction/pipeline-stages