CS3350B Computer Organization
Chapter 3: CPU Control & Datapath
Part 2: Single Cycle Datapath

Iqra Batool

Department of Computer Science
University of Western Ontario, Canada

Wednesday February 14, 2024

Outline

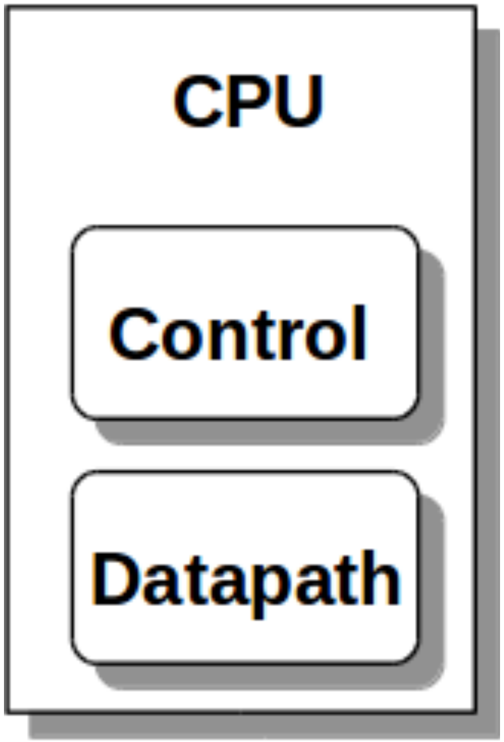
1 Overview

2 The Five Stages

3 Tracing the Datapath

4 Datapath In-Depth

Defining Parts of the Processor



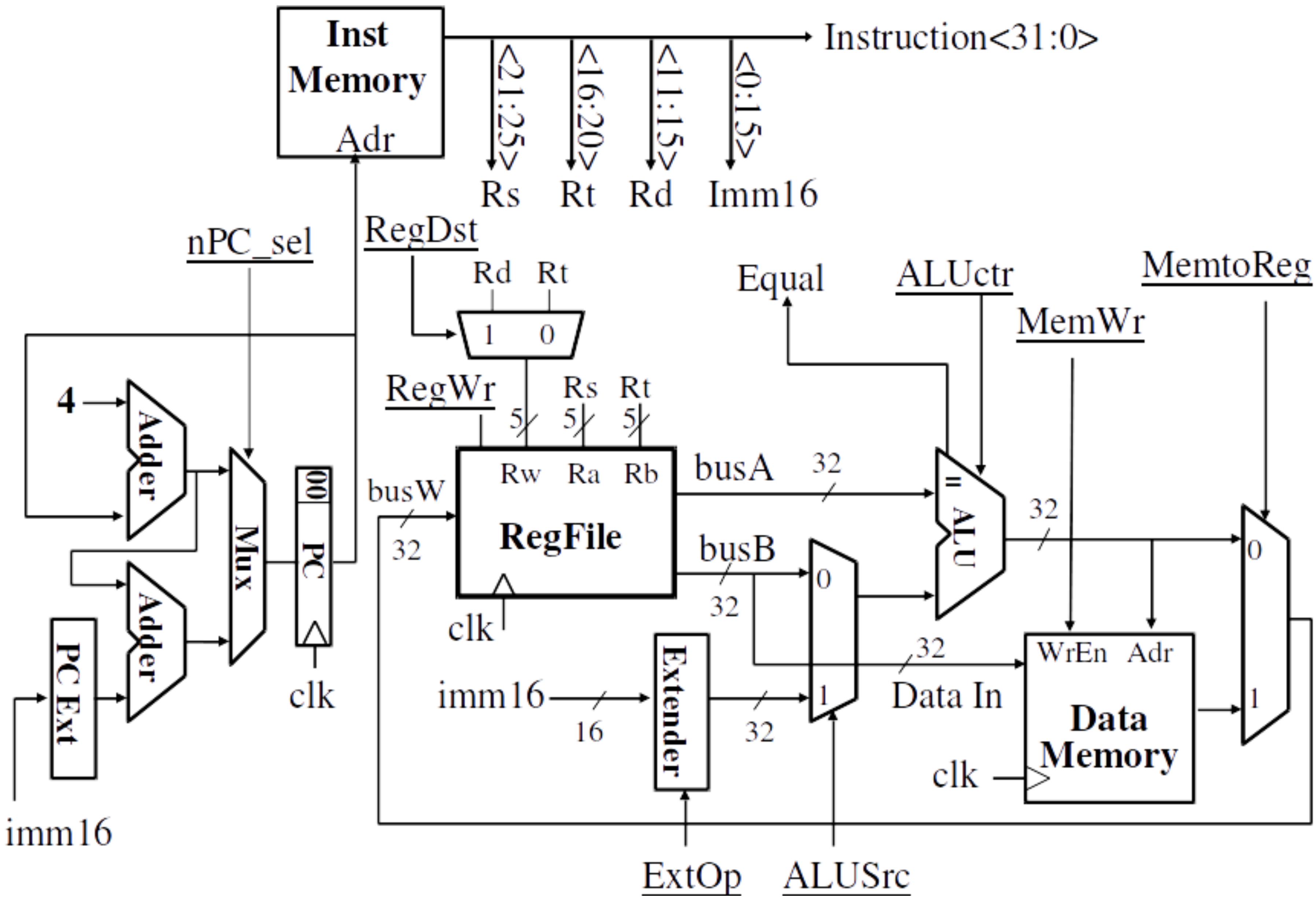
CPU/Processor: The encapsulation of the "working" part of the computer. A single integrated circuit ("chip") housing core(s).

Datapath: The flow of data through the processor. Contains circuits and logic, arithmetic, etc. What does the actual work.

Control: Controls the flow of data through the datapath. Controls the circuits' operations (e.g. what operation the ALU will perform).

Core: An independent "execution unit" on a CPU. Contains a datapath and a control unit.

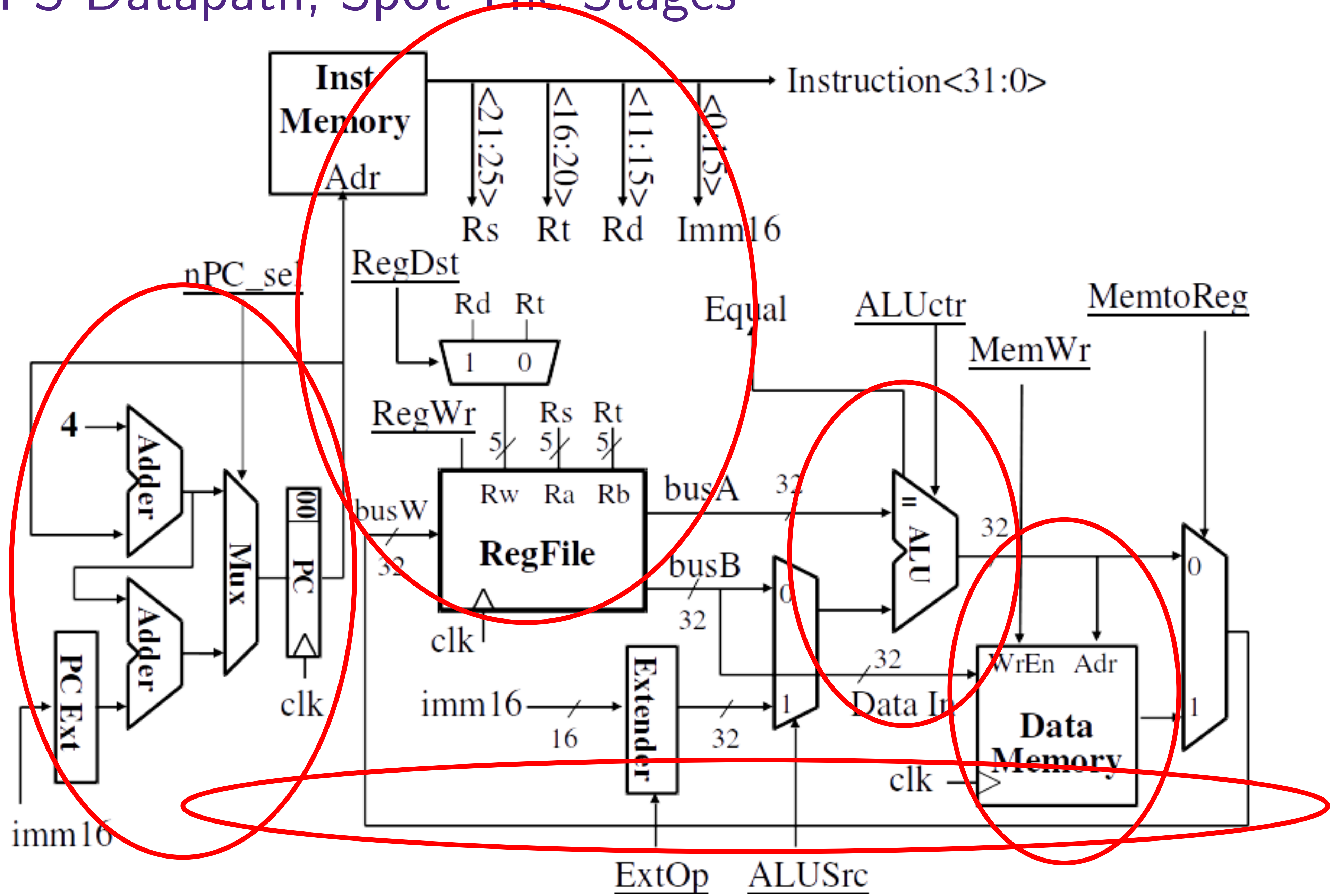
Preview: MIPS Datapath



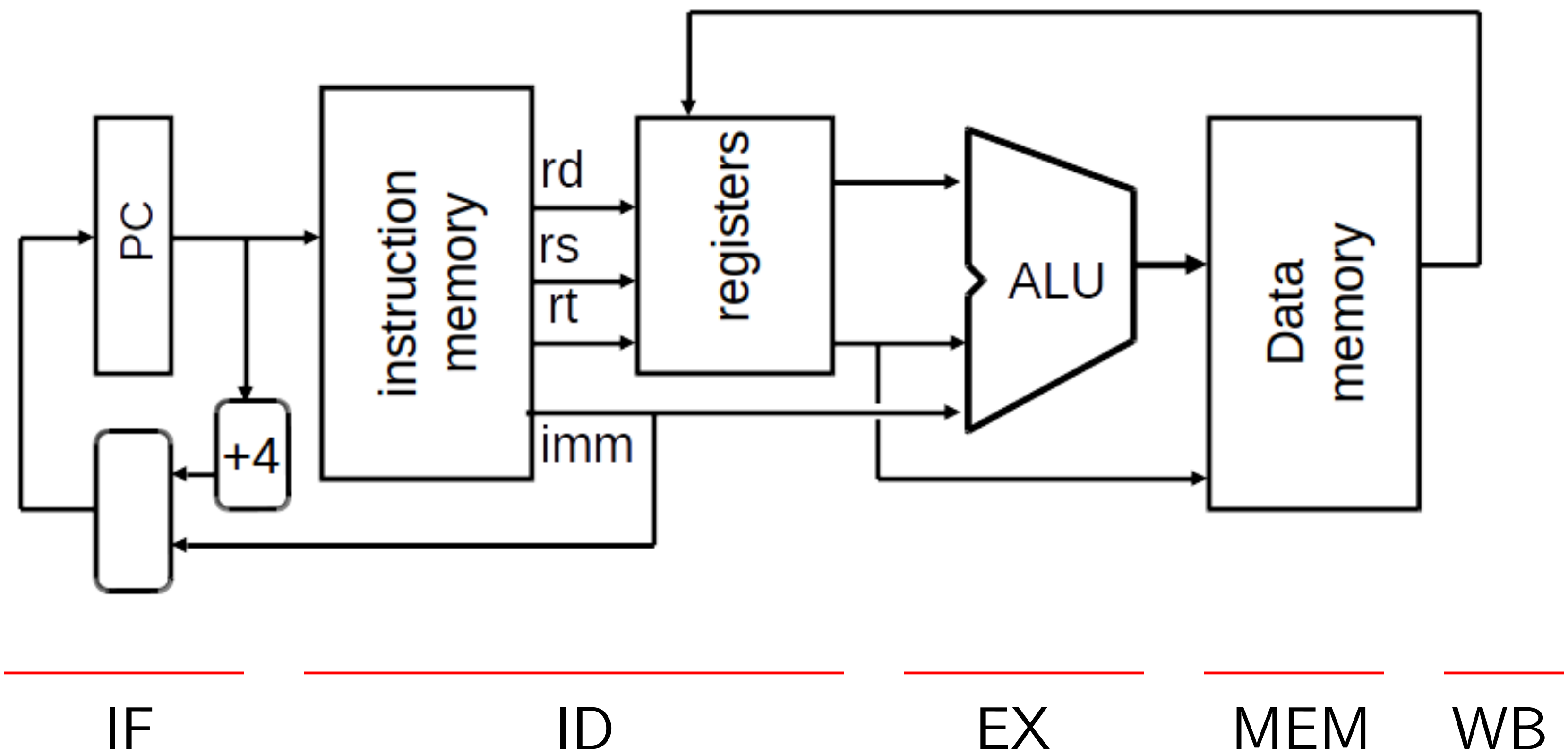
The 5-Stages of the Datapath

1. **IF**: Instruction Fetch
2. **ID**: Instruction Decode
3. **EX/ALU**: Execute/Arithmetic
4. **MEM**: Access Memory
5. **WB**: Write-back result

MIPS Datapath, Spot The Stages



A Simplified Datapath

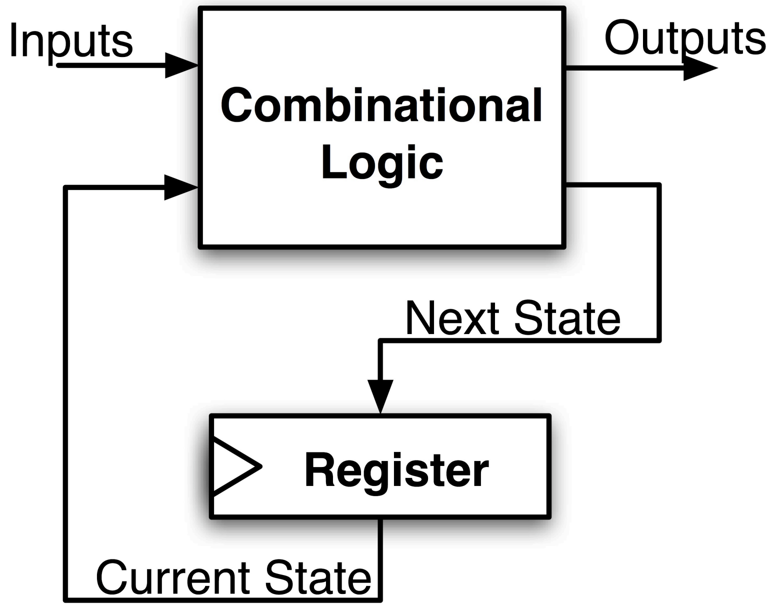


5 Stages in the Path

Why is there 5 stages?

- That's just what the designers of MIPS came up with.
  - ↳ Also, SPARC and Motorola.
  - ↳ Has been deemed the "Classic RISC Pipeline".
- Many other architectures use a different number of stages.
  - ↳ Intel has used 7, 10, 20, and 31 stages.
  - ↳ More stages $\Longrightarrow$ More complexity in circuits and control.
- Roughly speaking, each stage takes the same amount of time.
  - ↳ Prelude to Chapter 3: Part 4: The multi-cycle datapath

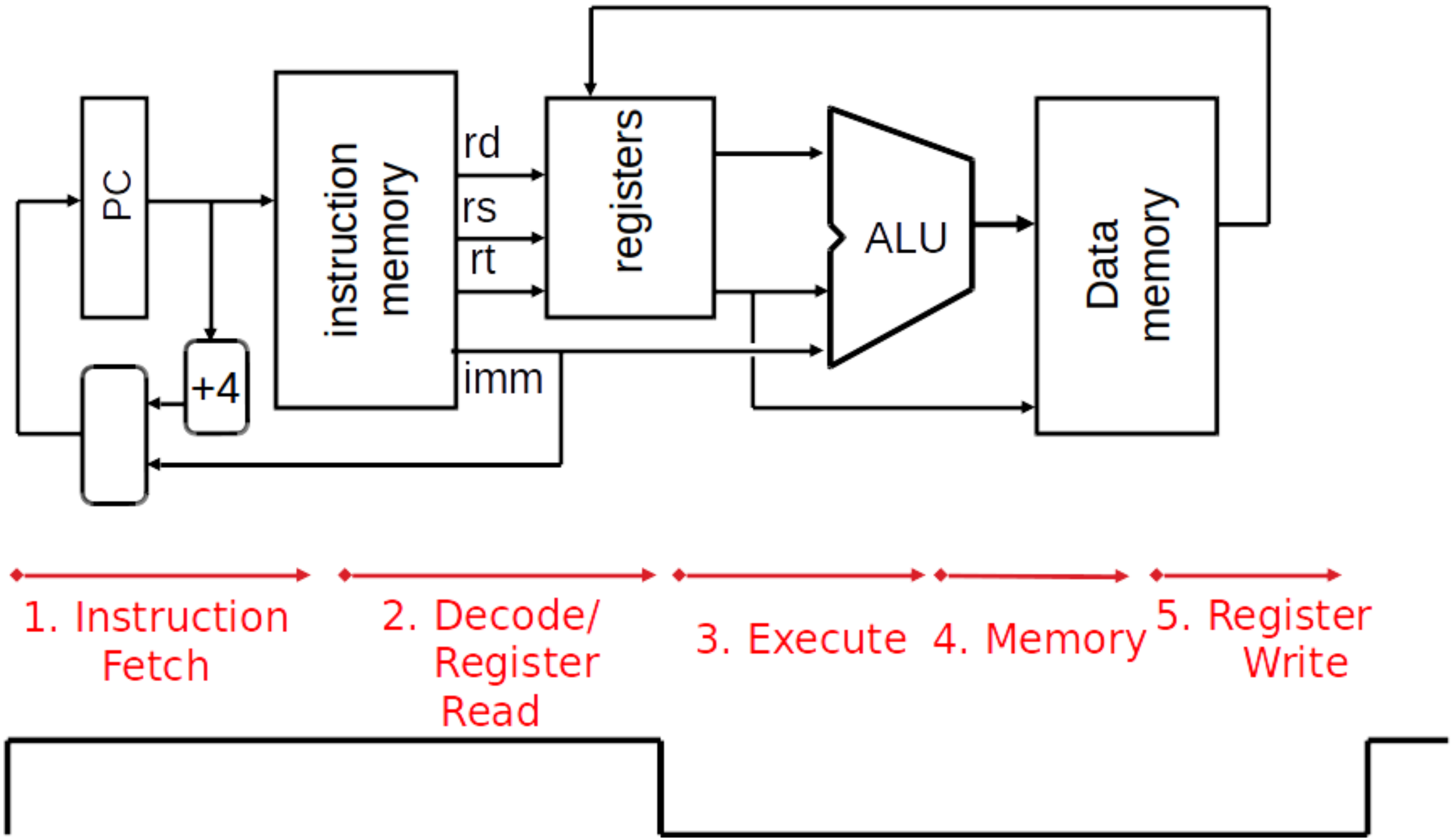
Single Cycle Datapath



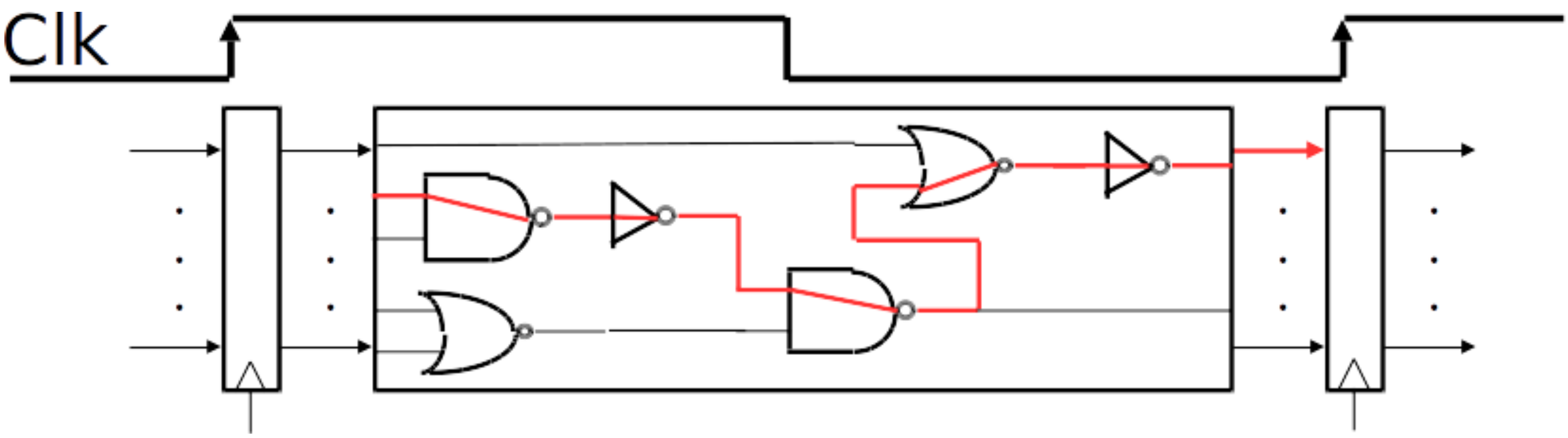
What makes a datapath **single cycle**?

- Flow of data through all stages of the datapath must occur within one clock cycle.
- The tic of the clock corresponds to the start of a new instruction starting to execute.
- One instruction is fetched, decoded, executed per clock cycle.
- Clock cycle must be long enough account for propagation delay of entire data path.

Clock Cycle for Single Cycle Datapath

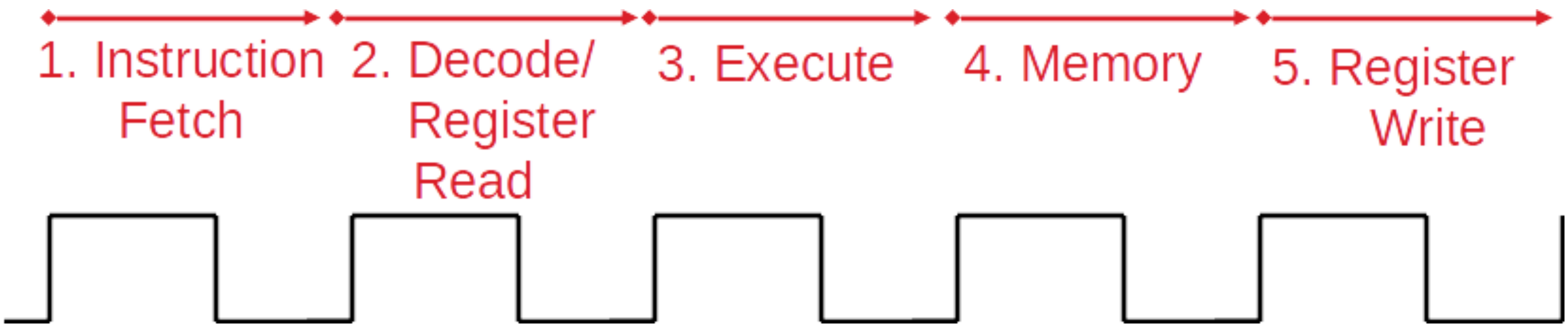


Critical Path Clocking



- The **critical path** determines length of clock cycle.
- Clock cycle must be long enough to accommodate the propagation delay of the longest path through the combination logic/datapath.
- *Recall:* all registers synchronized by the same rising edge of clock.

Multi-Cycle Datapath



- One clock cycle per *stage* within datapath.
- Clock cycle must be long enough to accommodate slowest stage.
- Allows for optimizations:
  - ↳ Skipping unused stages.
  - ↳ *Pipelining*.
  - ↳ We ignore these optimizations until the next chapter.

Outline

1 Overview

2 The Five Stages

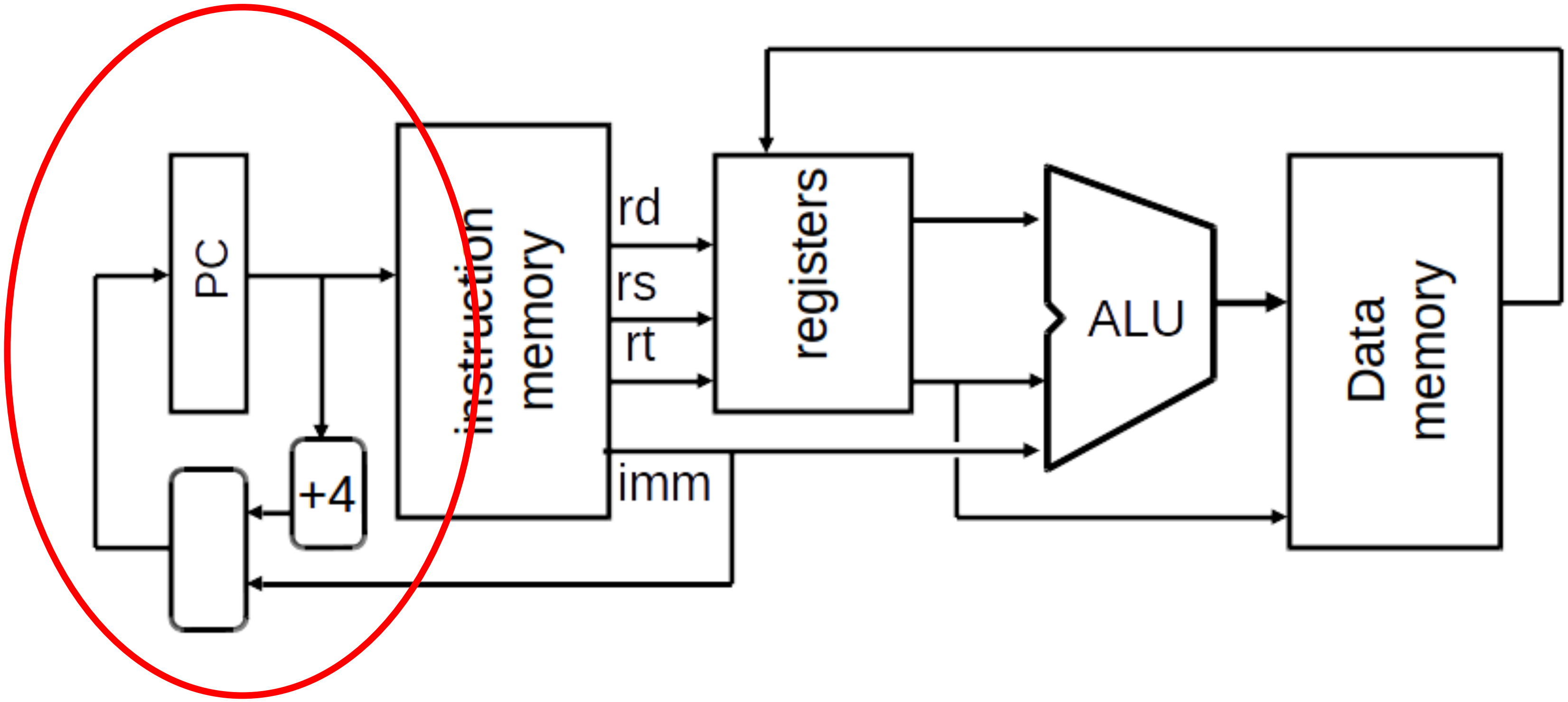
3 Tracing the Datapath

4 Datapath In-Depth

The Five Stages

- The components of the datapath represent the **union** of all circuitry needed by every instruction.
- Not every instruction will use every stage.
- Not every instruction will use every component within a stage.
- Nonetheless, all components are necessary to fulfill all instructions specified in the Instruction Set Architecture.

Instruction Fetch (1/2)

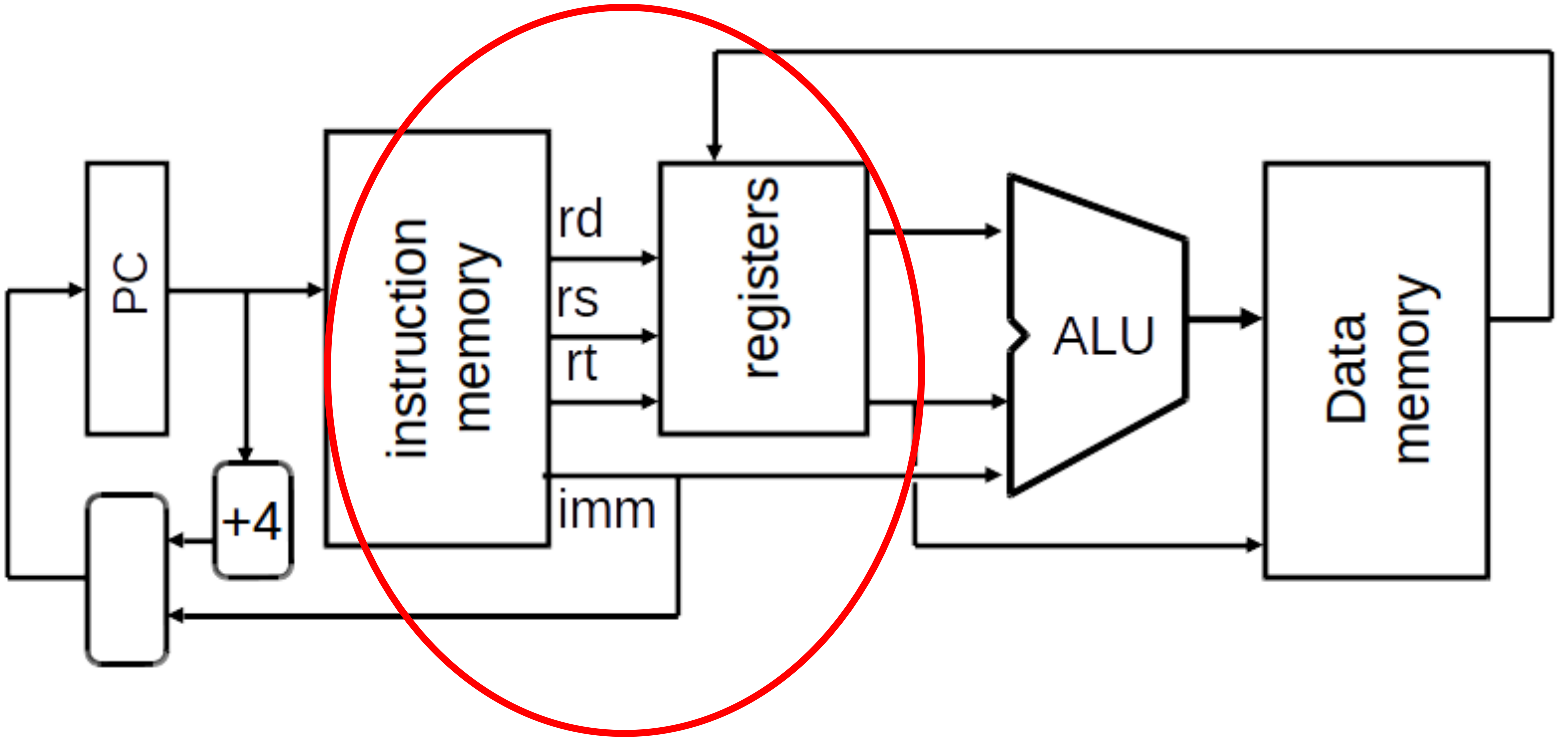


Instruction Fetch (2/2)

Instruction Fetch

- The instruction must be fetched from the instruction memory (banked L1 cache).
- Instructions are themselves encoded as a binary number.
- Instructions are stored in a memory word.
  - ↳ 32 bits in the case of MIPS.
- *Increment PC*: update the program counter for the next fetch.

Instruction Decode (1/2)

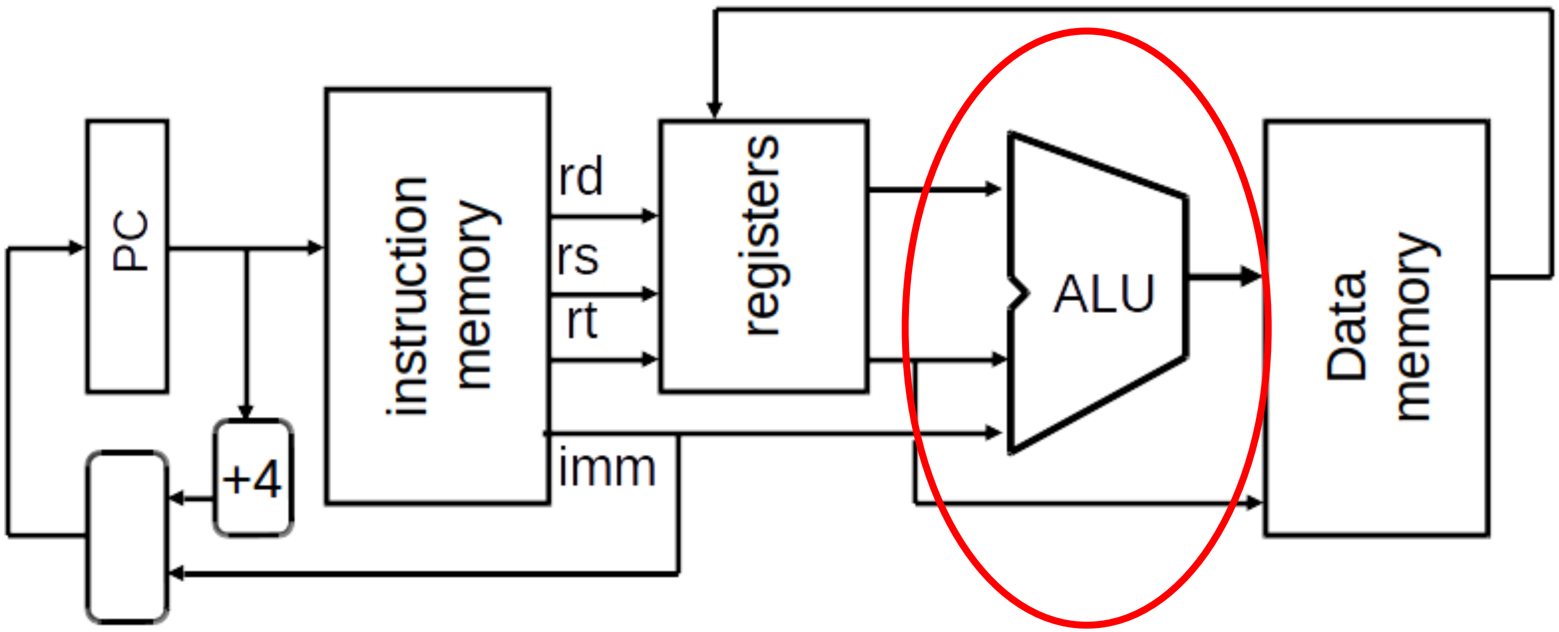


Instruction Decode (2/2)

Instruction Decode

- Determine the type of instruction to execute.
  - ↳ Read the *opcode*; it's always the first 6 bits in MIPS, regardless of the eventual type of instruction.
- Knowing the type of instruction, break up the instruction into the proper chunks; determine the instruction operands.
- Once operands are known, read the actual data (from registers) or extend the data to 32 bits (immediates).

Execute (a.k.a. ALU) (1/2)

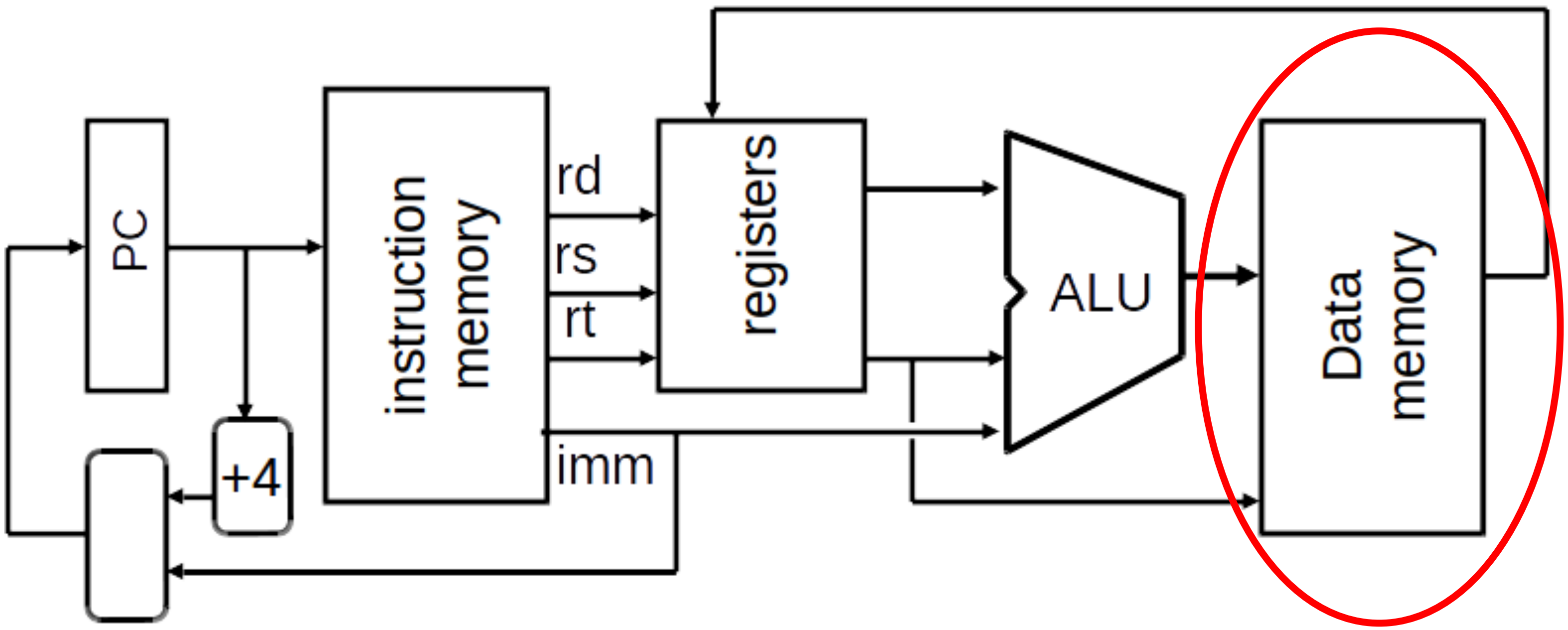


Execute (a.k.a. ALU) (2/2)

Execute

- Do the actual work of the instruction.
  - ↳ Add, subtract, multiply, shifts, logical operations, comparisons.
- For data transfer instructions, calculate the actual address to access.
  - ↳ Recall data transfer instructions have an offset and a base address.
  - ↳ `lw $t1, 12($t0)`
  - ↳ Calculates memory address `$t0 + 12`.

Memory Access (1/2)

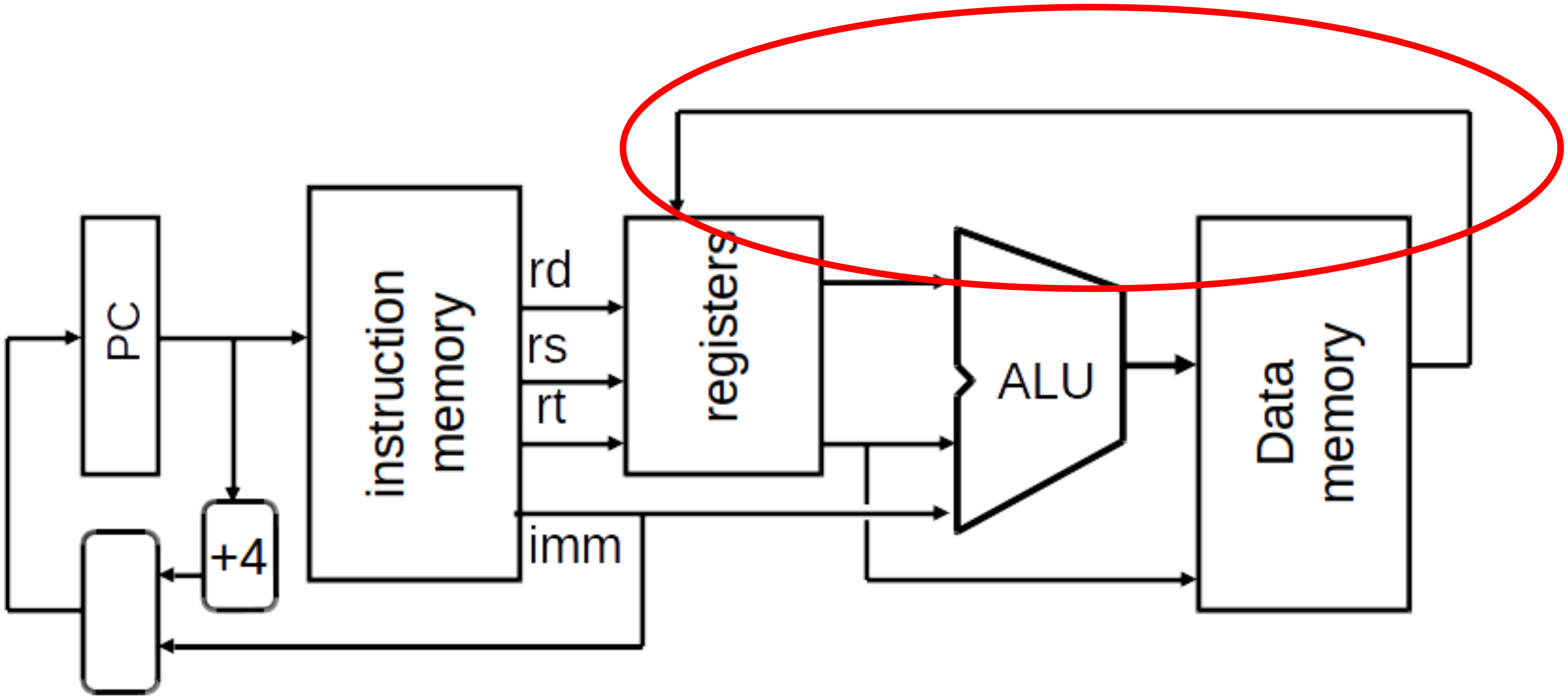


Memory Access (2/2)

Memory Access

- Access the memory using the address calculated in EX stage.
- Can be a read or a write.
- If the particular instruction is not a memory-accessing instruction, just *do nothing*.
- Since memory is relatively slow, just reading (or writing) data from it takes as much time as doing a full arithmetic operation.
  - ↳ **But** still quite fast due to caching and the memory hierarchy.
  - ↳ EX stage and MEM stage roughly same time. (Well really all stages are all roughly the same time.)

Write Back (1/2)



Write Back (2/2)

Write Back

- Write back the calculated value to the register.
- Could be the result of some arithmetic operation.
- Could be the result of some memory load.
- If nothing is being written back (e.g. on a memory store) just *do nothing*.
- Not to be confused with write back cache policy.

Outline

1 Overview

2 The Five Stages

3 Tracing the Datapath

4 Datapath In-Depth

Example 1: add (1/2)

two read, one write.

individual stages

add $3, $1, $2 ⇒ $3 = $1 + $2

W R

op	rs	rt	rd	shamt	funct
0	1	2	3	0	32
000000	00001	00010	00011	00000	100000

all times.

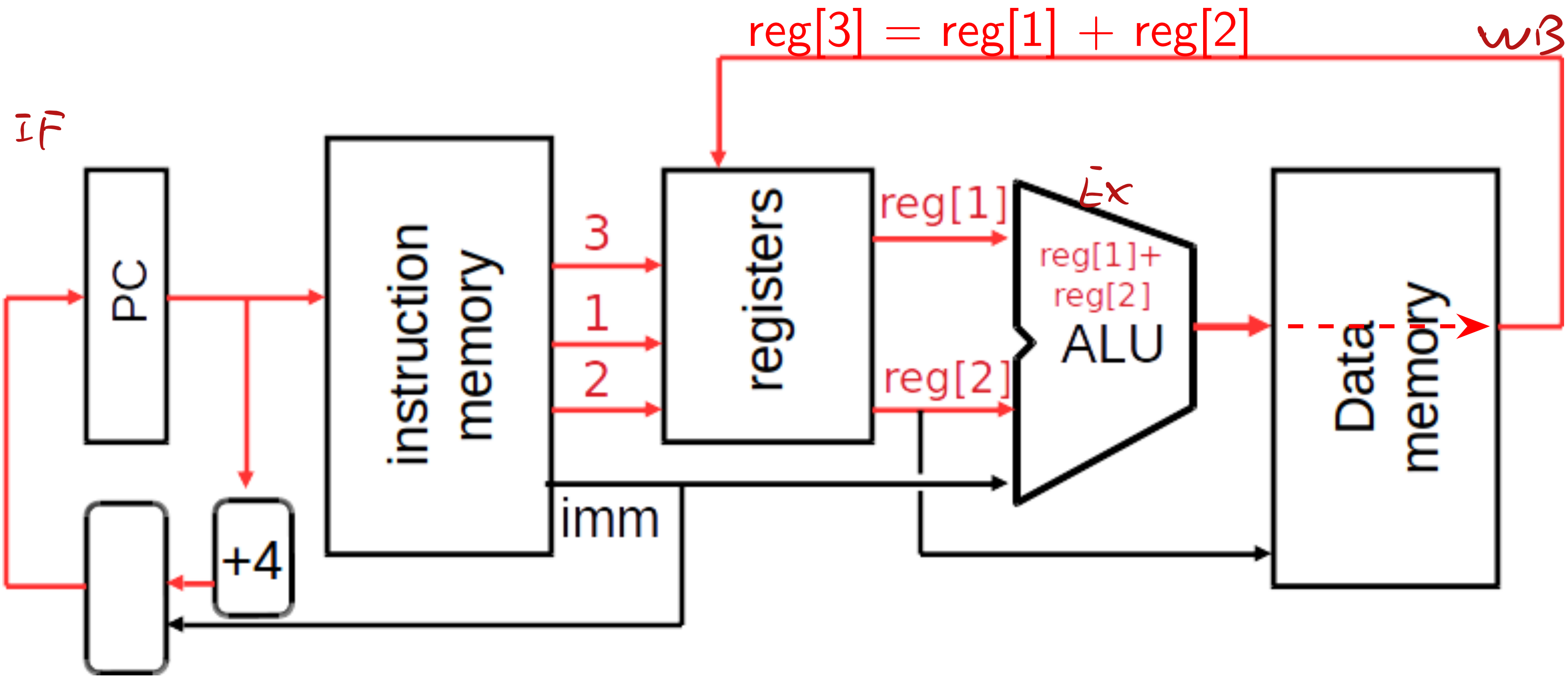
R-type: Read these.

prepare for next stages

- IF: Fetch instruction and increment PC.
- ID: Read opcode, determine R-type instruction, read values of $rs, $rt.
- EX: Perform addition operation on values stored in $1 and $2.

  calculation.
- MEM: Do nothing.
- WB: Write the sum back to $3.

Example 1: add (2/2)



`add $3, $1, $2`

Example 2: `slti` (1/2)

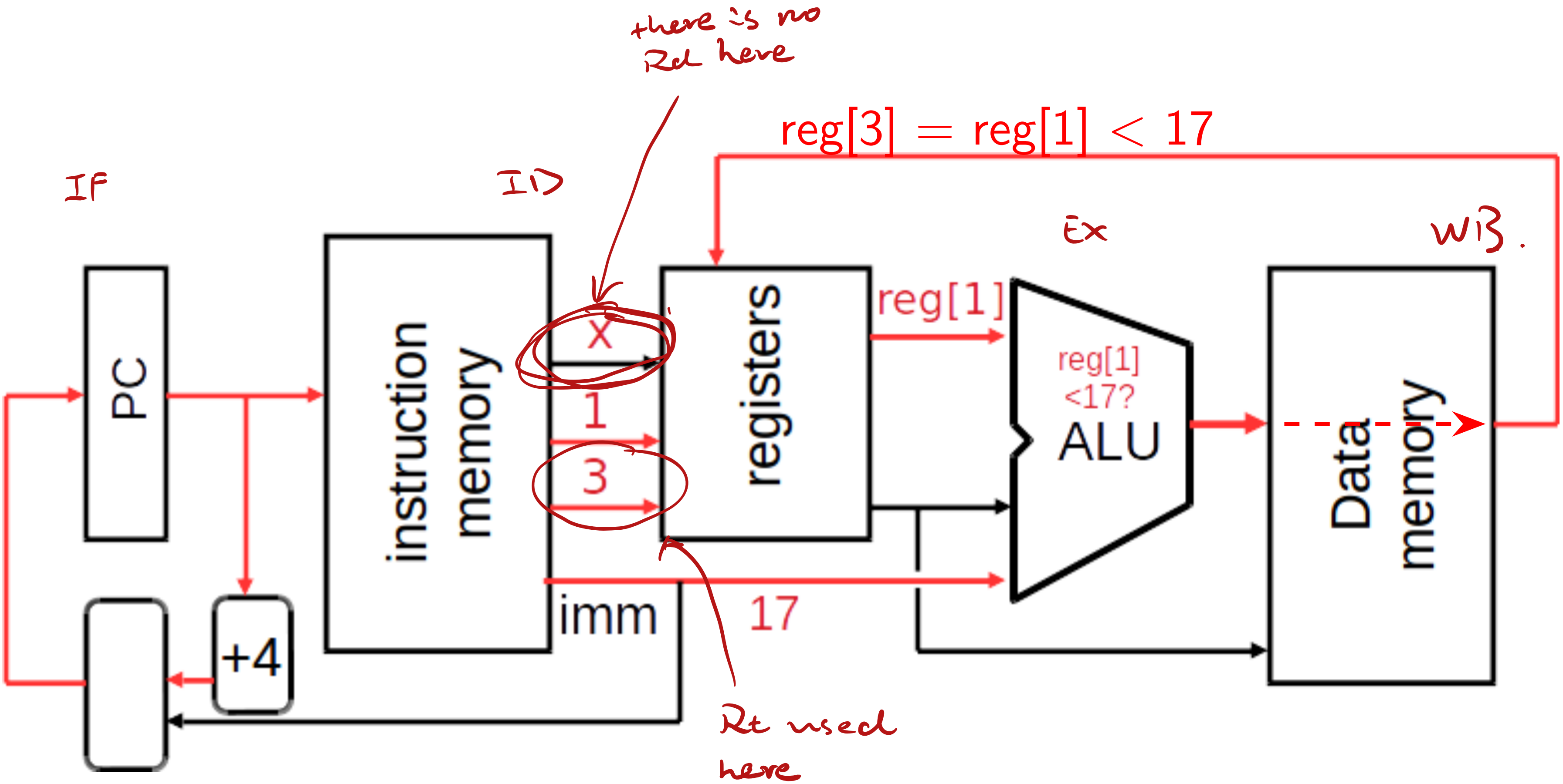
set less than immediate

$$\texttt{slti \$3, \$1, 17} \Rightarrow \$3 = (\$1 < 17)$$

op	rs	rt	immediate
001010	00001	00011	0000000000010001

- IF: Fetch instruction and increment PC.
- ID: Read opcode, determine I-type instruction, read values of $rs, immediate. extend imme
- EX: Perform comparison operation on value of $1 and immediate.
- MEM: Do nothing.
- WB: Write the comparison result back to $3.

Example 2: `slti` (2/2)



`slti $3, $1, 17`

Example 3: `sw` (1/2)

store word.

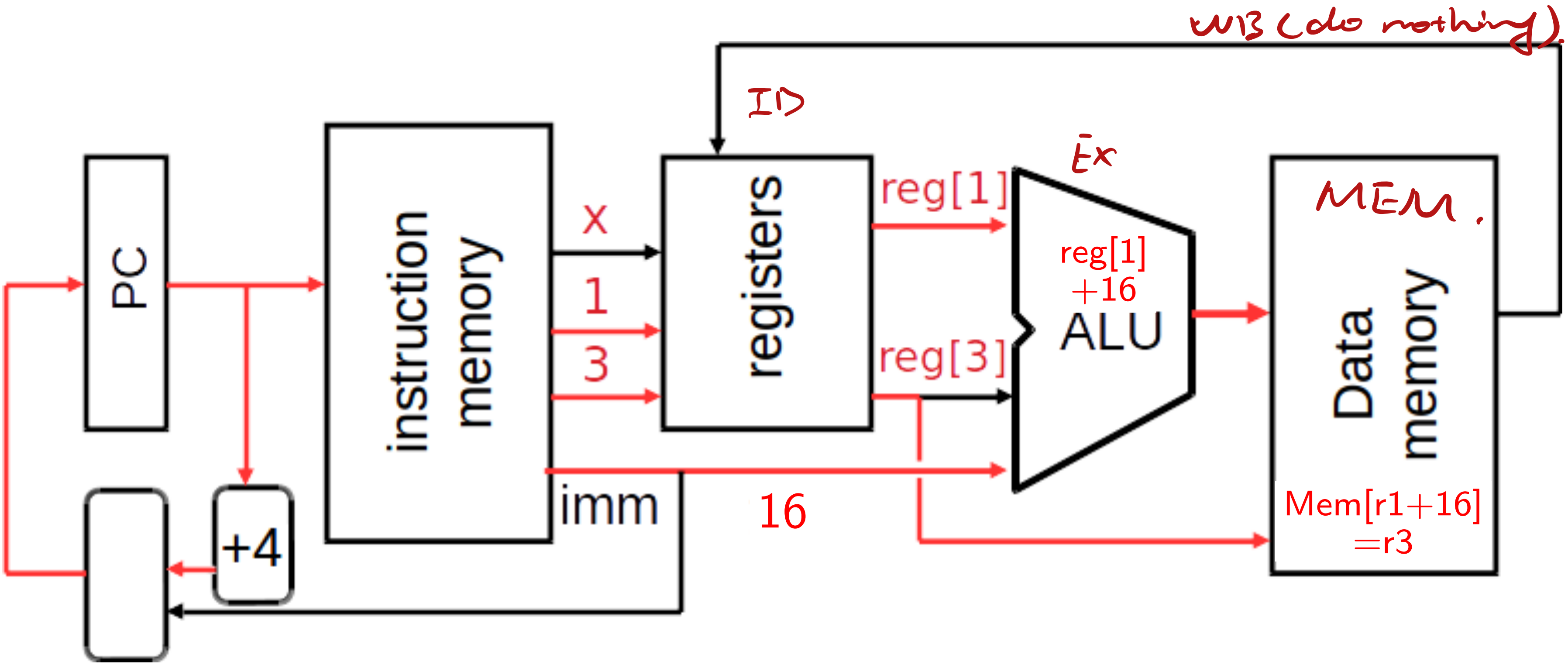
$$\texttt{sw \$3, 16(\$1)} \Rightarrow \mathrm{Mem}[\$1+16] = \$3$$

op	rs	rt	immediate
101011	00001	00011	0000000000010000

- IF: Fetch instruction and increment PC.
- ID: Read opcode, determine I-type instruction, read values of $rs, $rt, imm.
- EX: Calculate memory address from reg[1] and 16 (offset).
- MEM: Write value of $3 into Mem[reg[1] + 16].
- WB: Do nothing.

WB for register only!!!

Example 3: sw (2/2)



sw $3, 16($1)

Example 4: lw (1/2)

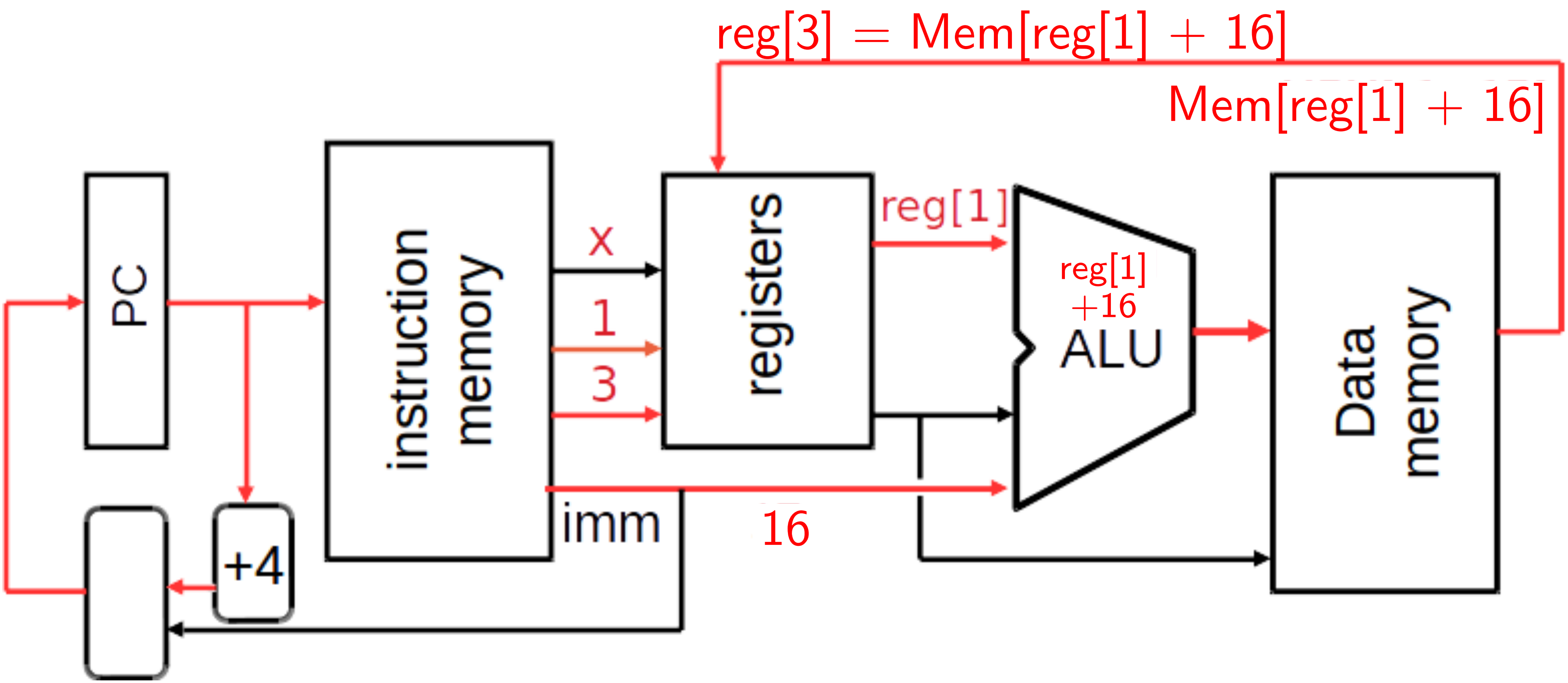
load word.

$$\texttt{lw \$3, 16(\$1)} \Rightarrow \$3 = \text{Mem}[\$1 + 16]$$

op	rs	rt	immediate
101011	00001	00011	0000000000010000

- IF: Fetch instruction and increment PC.
- ID: Read opcode, determine I-type instruction, read values of $rs, imm. Rt here used as destination.
- EX: Calculate memory address from reg[1] and 16 (offset).
- MEM: Read value of Mem[reg[1] + 16].
- WB: Write value of Mem[reg[1] + 16] to $3.

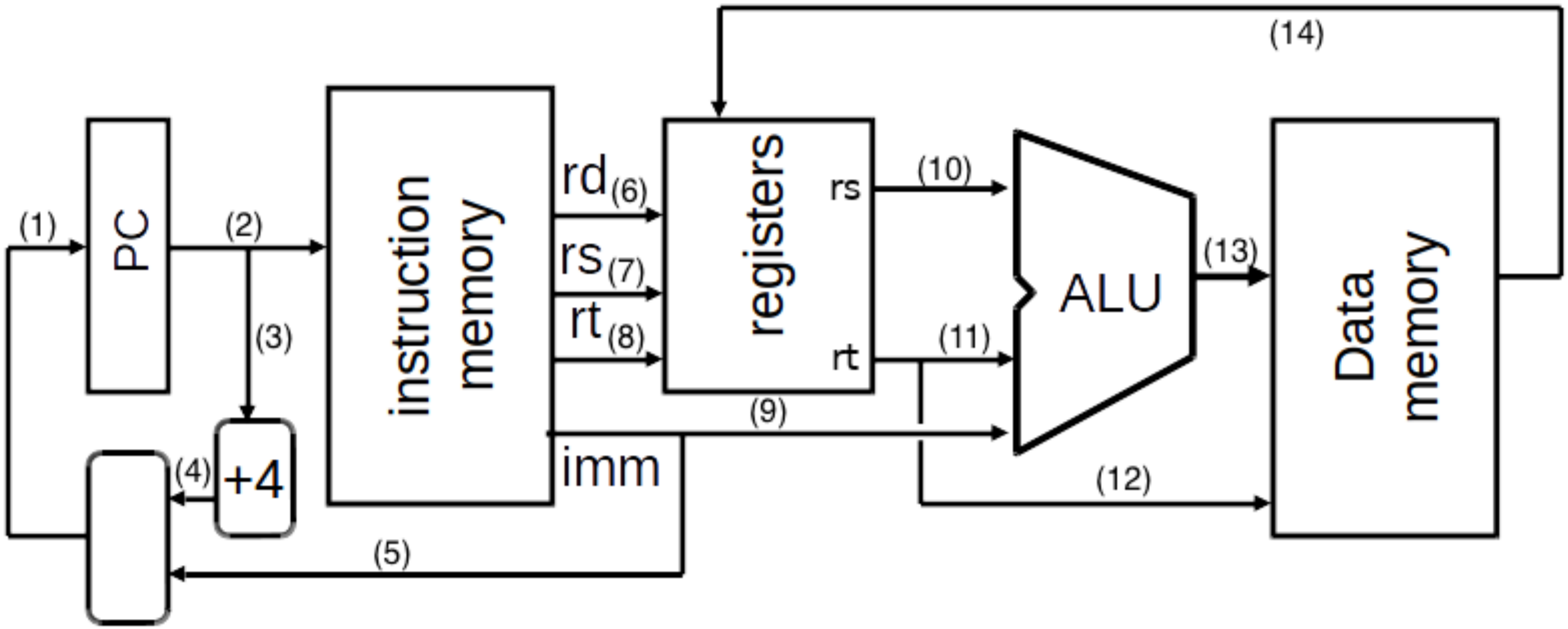
Example 4: `lw` (2/2)




```
lw $3, 16($1)
```

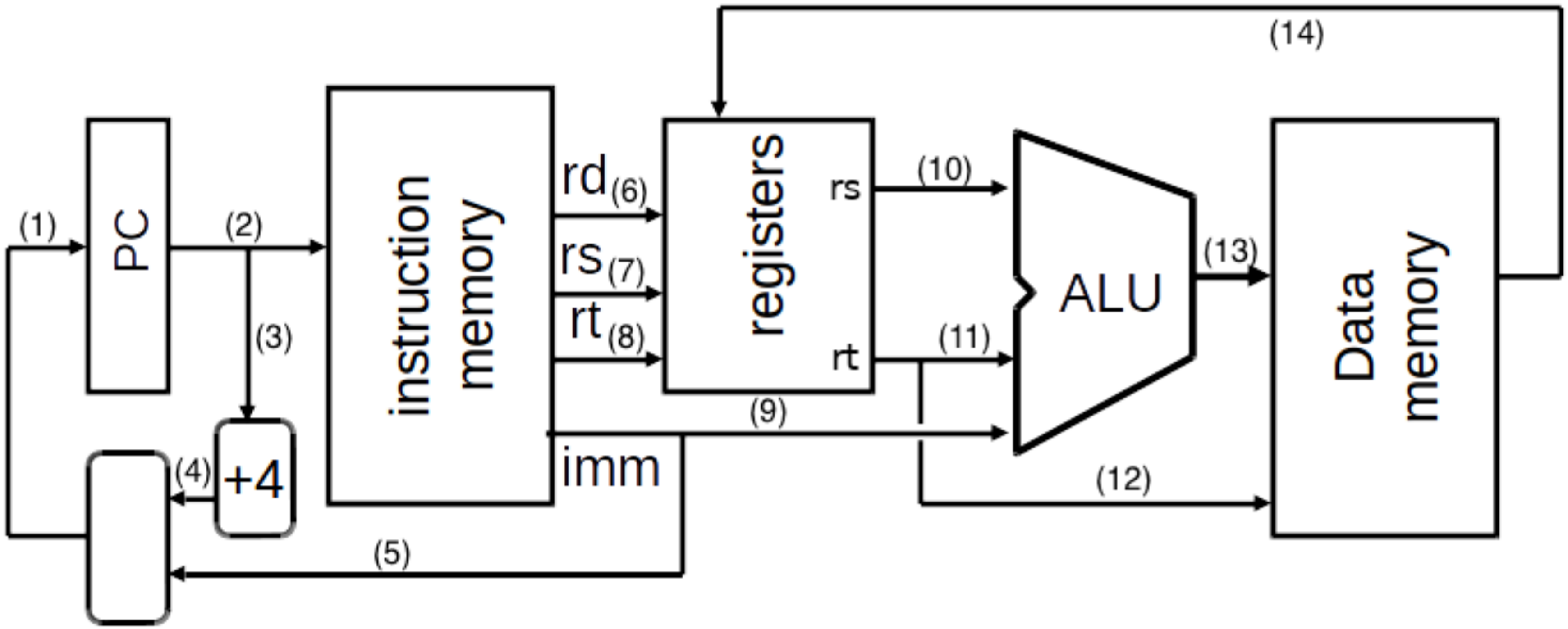
lw use all five stages.

Exercise: `slt`



`slt $10, $5, $6`

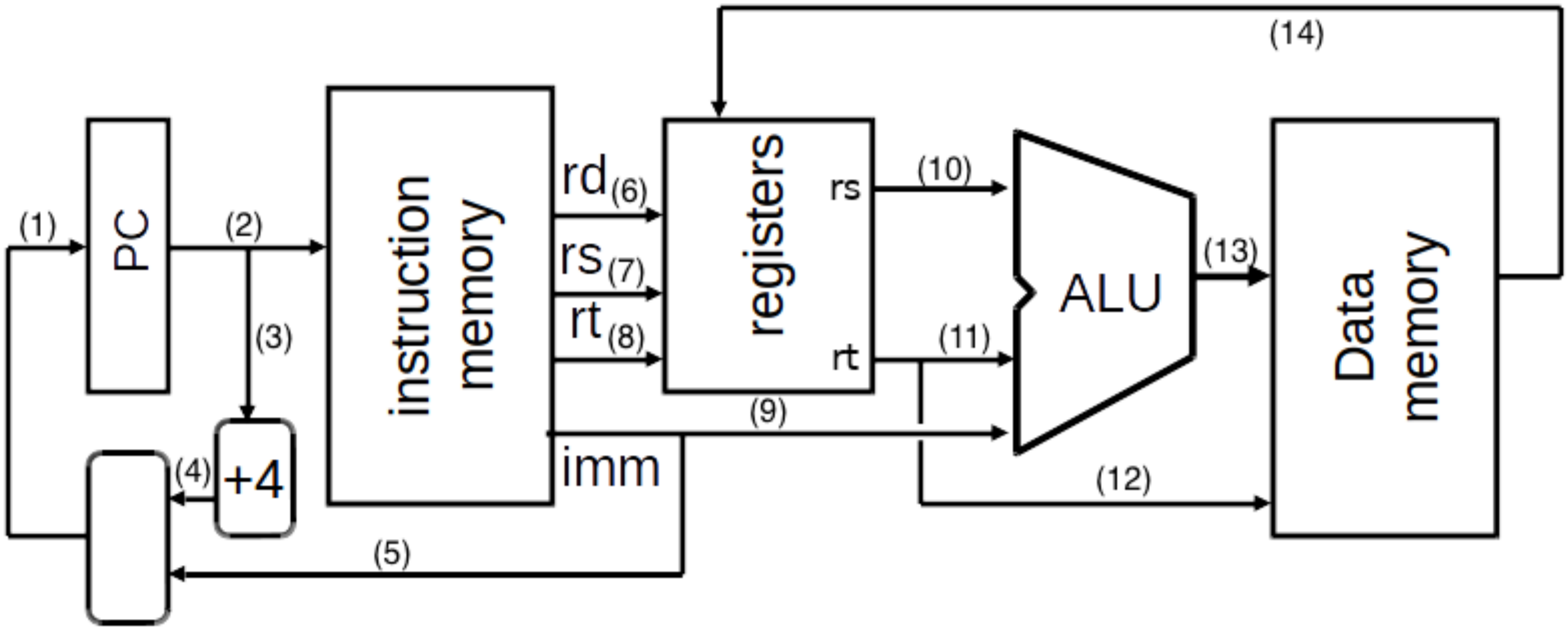
Exercise: `slt`



`slt $10, $5, $6`

$$reg[10] = reg[5] < reg[6]$$

Exercise: `slt`

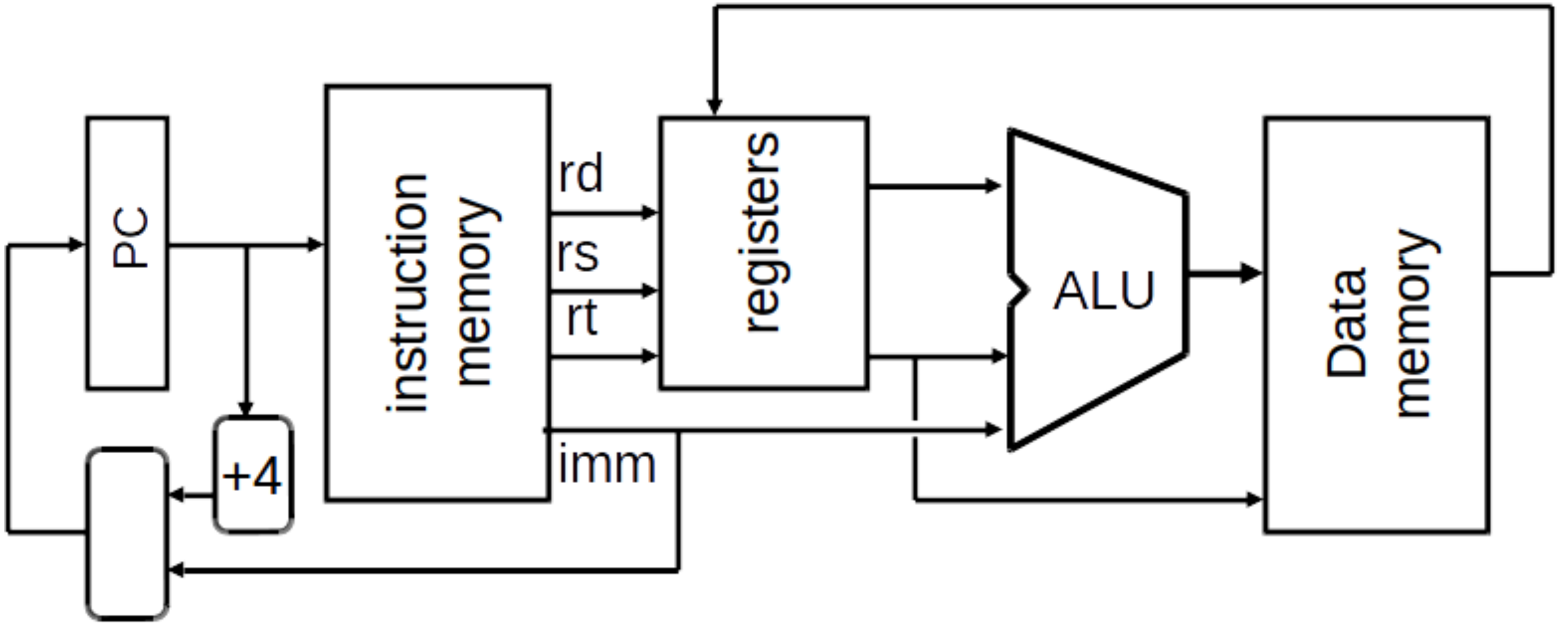


`slt $10, $5, $6`

$$reg[10] = reg[5] < reg[6]$$

1, 2, 3, 4, 6, 7, 8, 10, 11, 13, 14

Exercise: beq



`beq $8, $9, 128`

Outline

1 Overview

2 The Five Stages

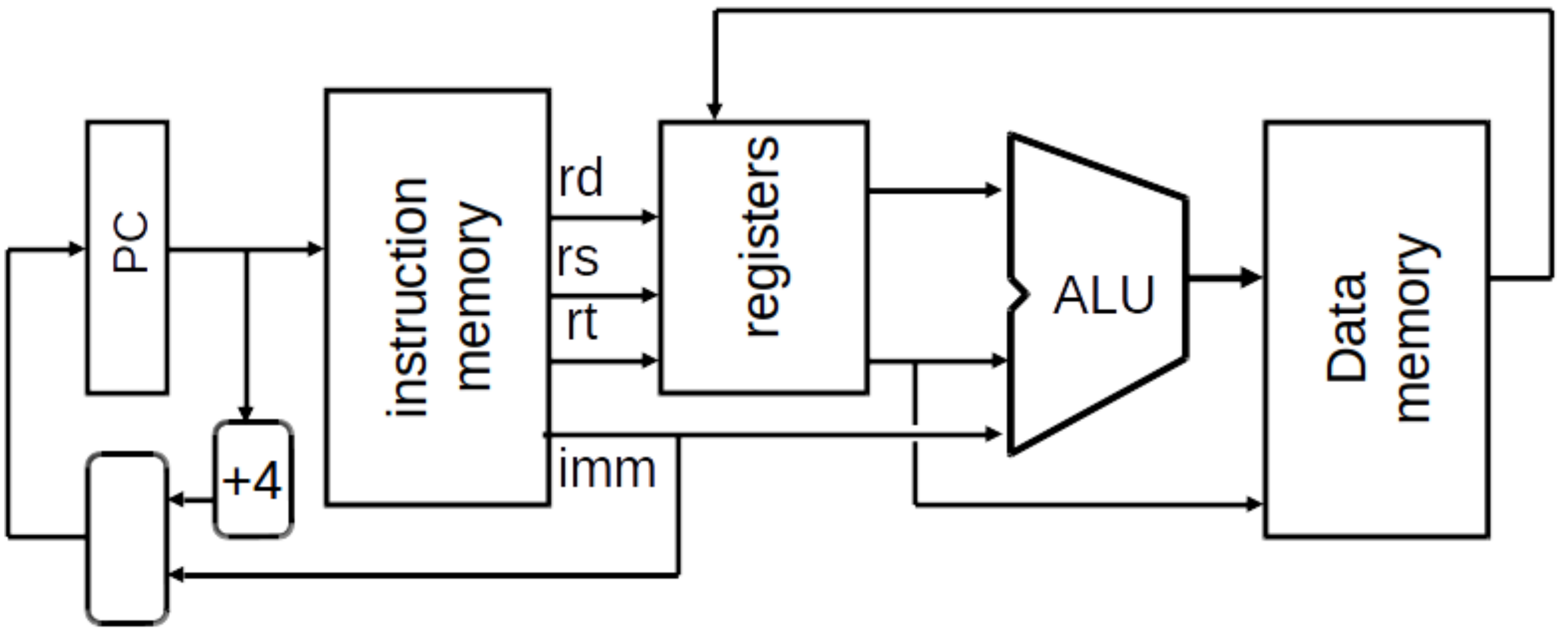
3 Tracing the Datapath

4 Datapath In-Depth

Satisfying the ISA

- *Recall:* The specification of the ISA and the datapath are highly coupled.
  - ↳ We need enough circuity to accommodate every possible instruction in the ISA.
- Instructions belong to a few general categories. We need circuitry for to satisfy each and every one.
  - ↳ All instructions use PC and instruction memory.
  - ↳ Arithmetic: ALU, Registers.
  - ↳ Data transfer: Register, Memory.
  - ↳ Conditional jumping: PC, Registers, Comparator (ALU).
  - ↳ Unconditional jumping: PC, Registers.
- `lw` is one instruction which makes use of *every* stage.

Missing Datapath Details

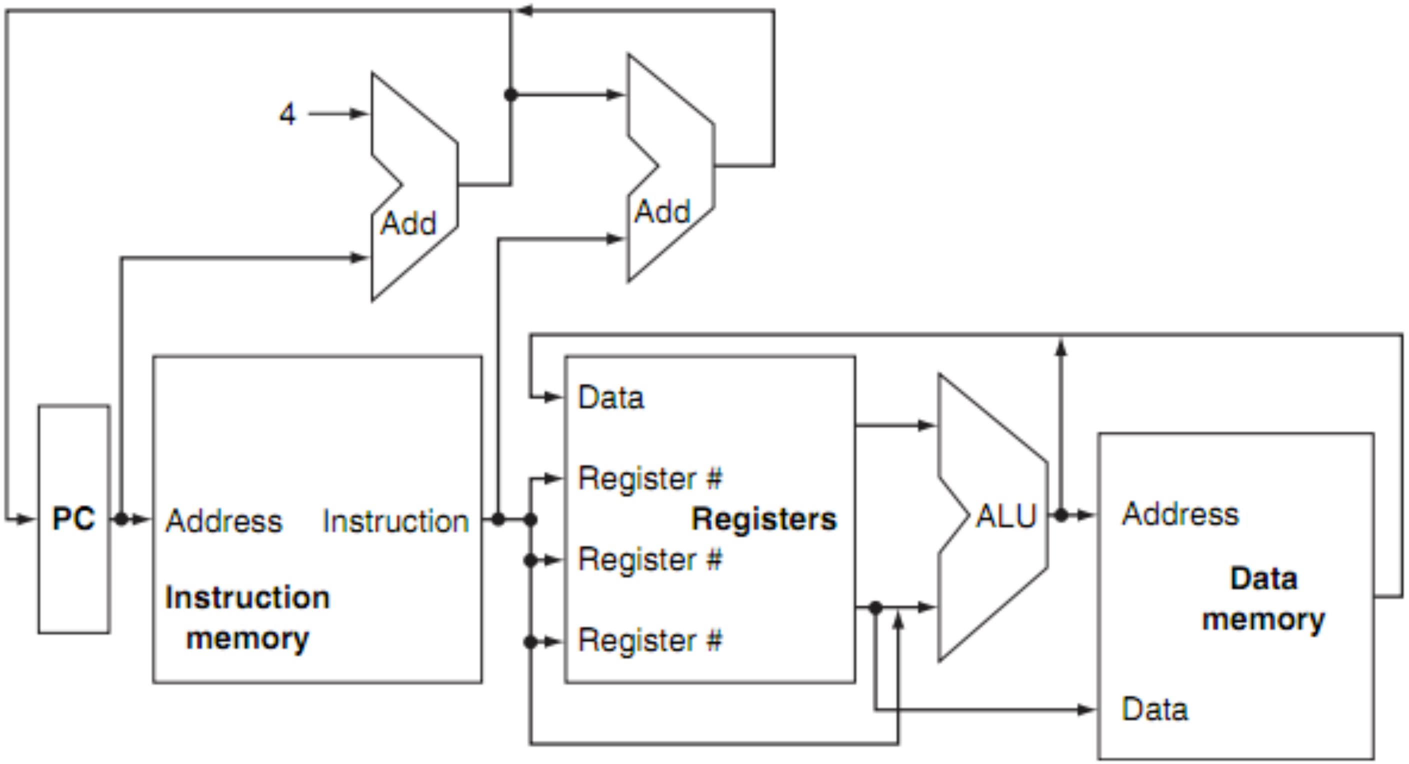


Many subtle details are missing from this simplified datapath. i.e. some pass through MEM

- Multiplexers needed to control flow to/from registers, ALU, memory.
- On Write Back is data coming from ALU ("pass-through memory") or beign loaded from memory?
- Control which operation ALU performs.
- Control whether reading or writing write to memory, registers.

Multiplexers in the Datapath

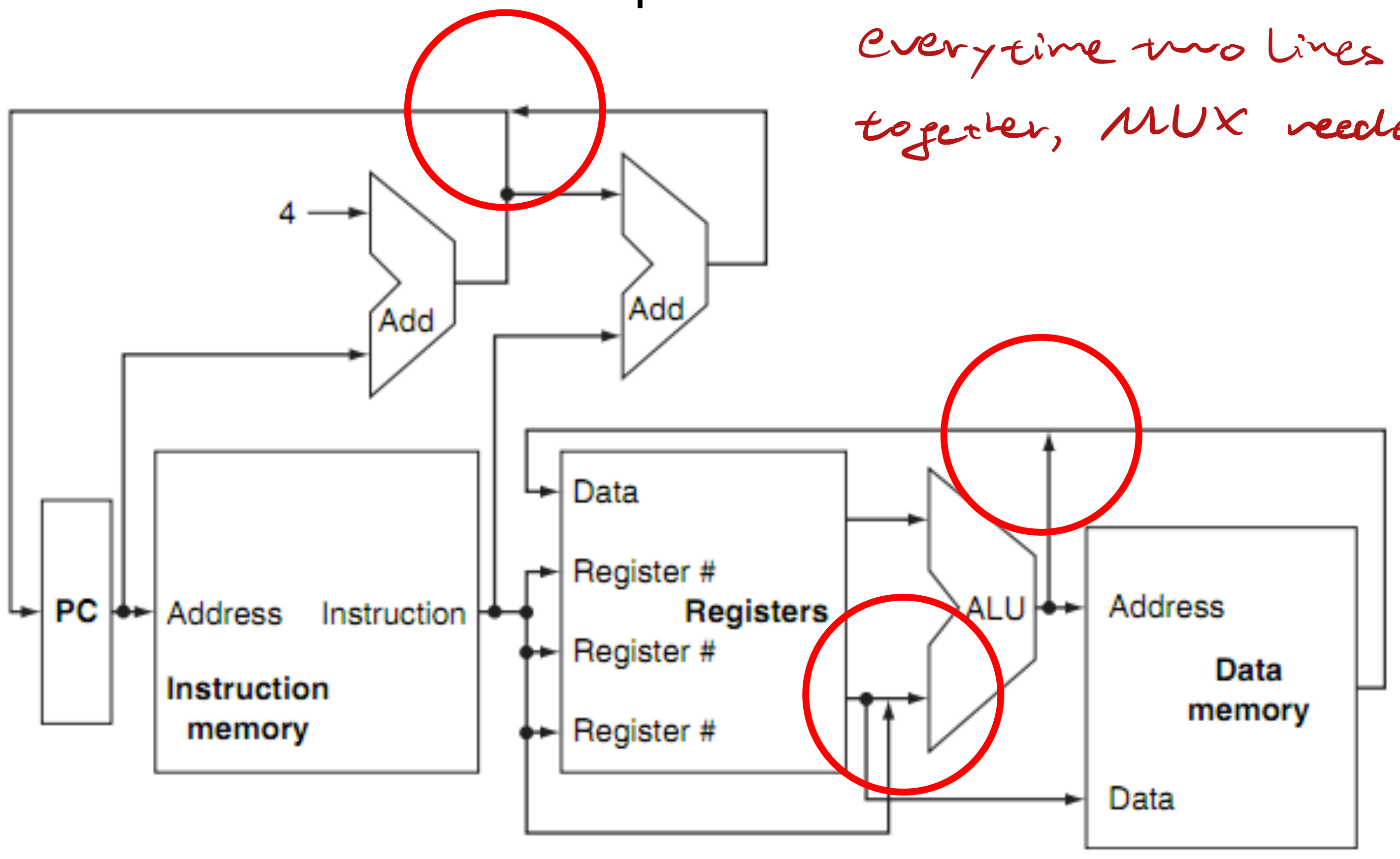
Where do we need multiplexers to control data flow?



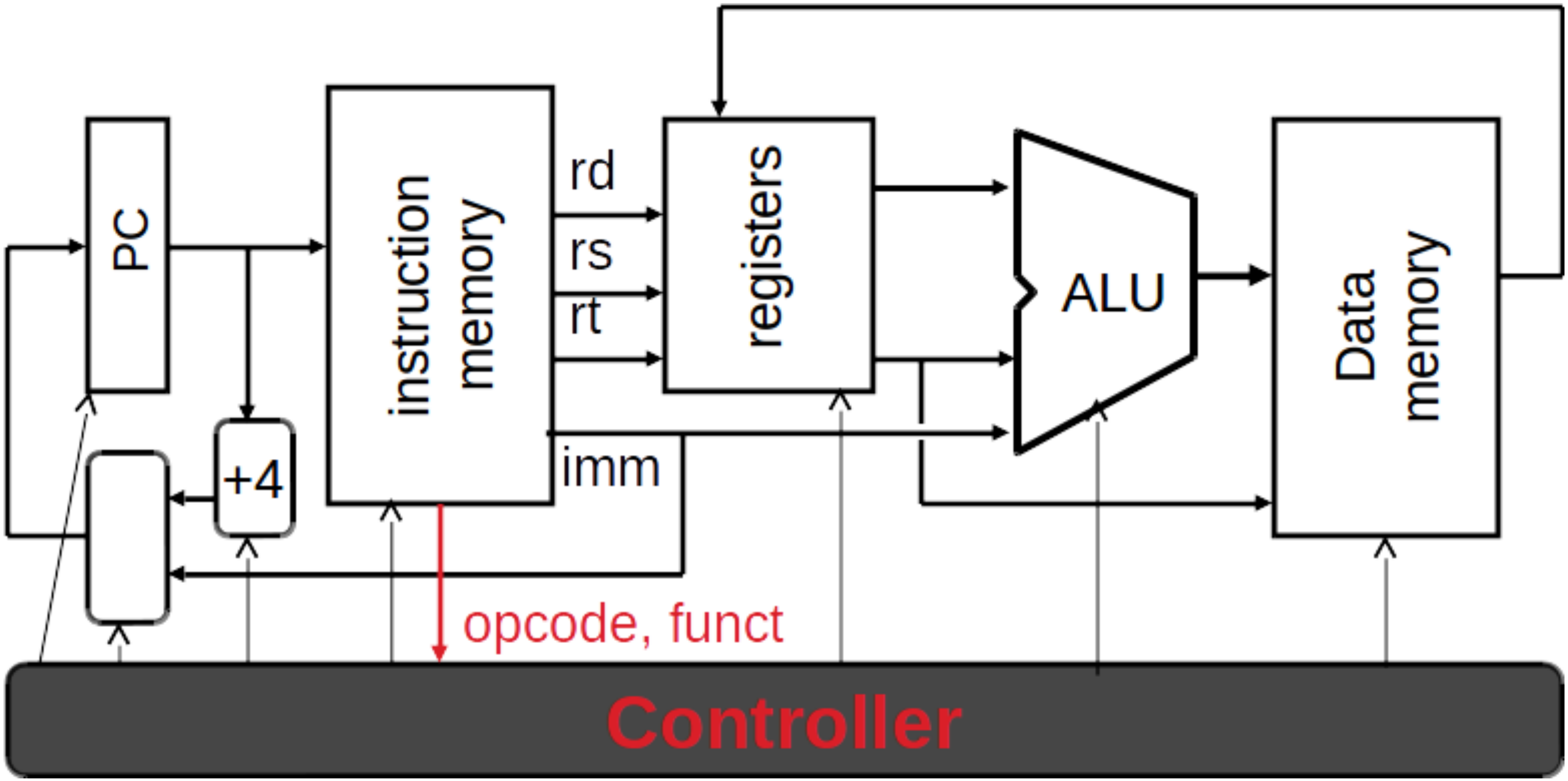
Multiplexers in the Datapath

Where do we need multiplexers to control data flow?

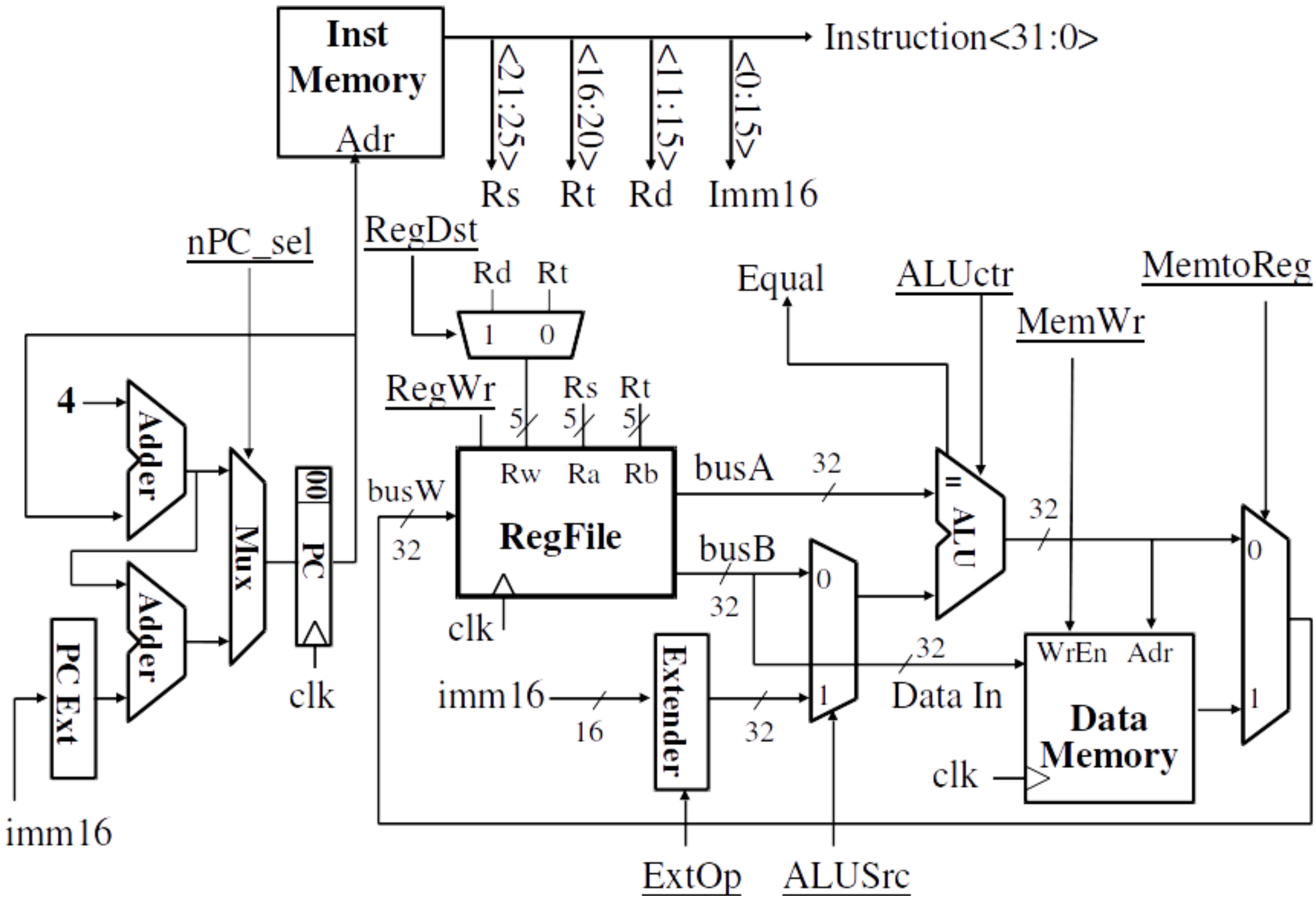
everytime two lines joint together, MUX needed.



Controlling the Multiplexers, ALU, Circuitry



MIPS Datapath with Control Signals



Datapath Summary

- ISA and circuitry highly coupled.
- 5 Stages: IF, ID, EX, MEM, WB.
- Some stages go unused for some instructions.
- Single cycle: clock cycle determined by propagation delay of entire datapath.
- Multi-cycle: clock cycle determined by propagation delay of *slowest* stage.
- Additional control (multiplexers, ALU, read/write) needed for the datapath.