CSL7070: Computer Architecture
Lecture 6, 2nd February 2022

Dip Sankar Banerjee

Indian Institute of Technology, Jodhpur
January-April 2022

Five Components of a Computer

Computer

Keyboard, Mouse Disk

Processor

Control

Datapath

Memory (passive)

(where programs, data live when running)

Devices

Input

Output

(where programs, data live when not running)

Display, **Printer**
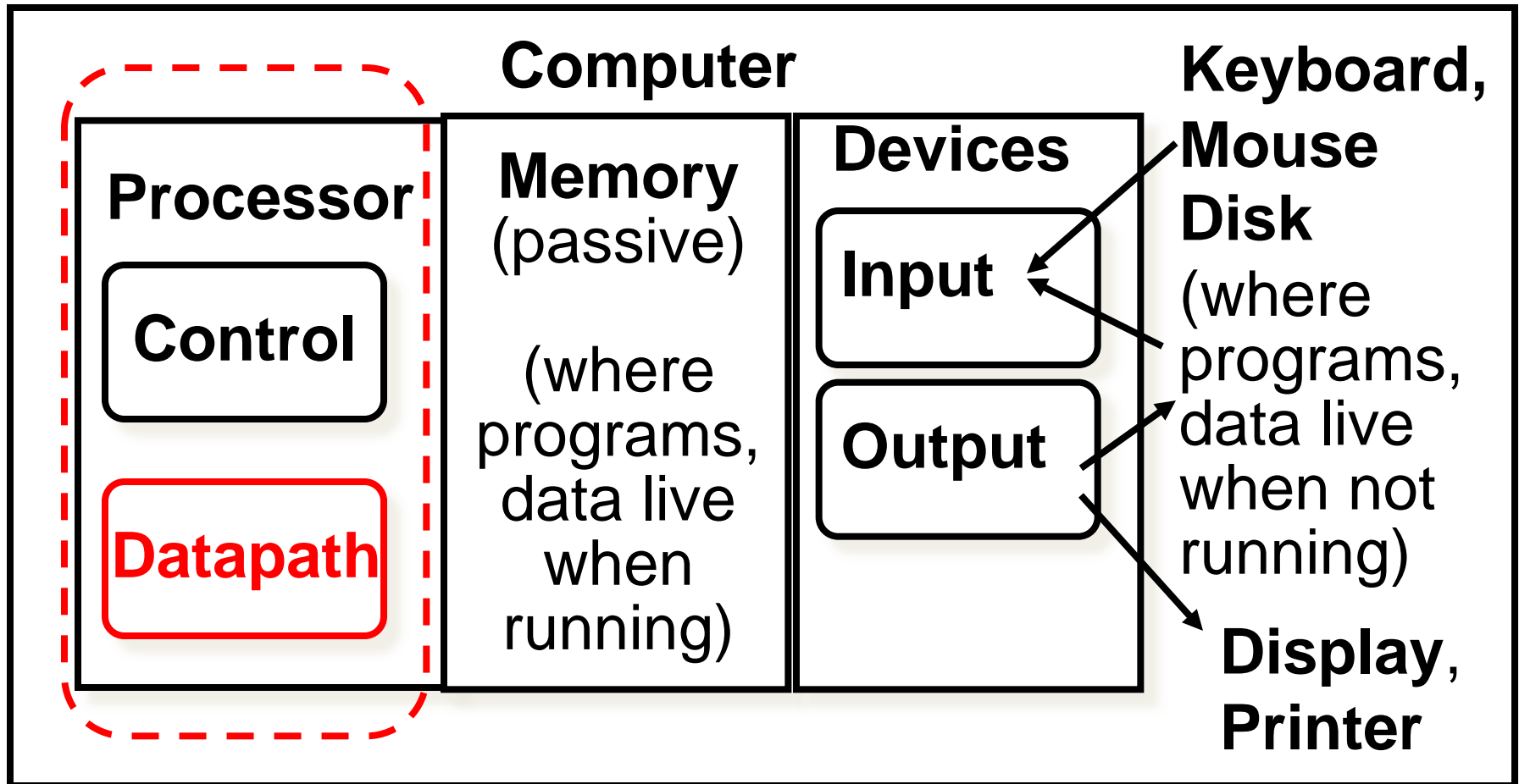
The CPU

- Processor (CPU): the active part of the computer that does all the work (data manipulation and decision-making)

- Datapath: portion of the processor that contains hardware necessary to perform operations required by the processor (the brawn)

- Control: portion of the processor (also in hardware) that tells the datapath what needs to be done (the brain)

Stages of the Datapath : Overview

- Problem: a single, atomic block that "executes an instruction" (performs all necessary operations beginning with fetching the instruction) would be too bulky and inefficient

- Solution: break up the process of "executing an instruction" into stages, and then connect the stages to create the whole datapath

  – smaller stages are easier to design

  – easy to optimize (change) one stage without touching the others

Five Stages of the Datapath

- Stage 1: *Instruction Fetch*
- Stage 2: *Instruction Decode*
- Stage 3: *ALU* (Arithmetic-Logic Unit)
- Stage 4: *Memory Access*
- Stage 5: *Register Write*

Stages of the Datapath (1/5)

- There is a wide variety of MIPS instructions: so what general steps do they have in common?

- Stage 1: Instruction Fetch
  - no matter what the instruction, the 32-bit instruction word must first be fetched from memory (the cache-memory hierarchy)
  - also, this is where we Increment PC (that is, PC = PC + 4, to point to the next instruction: byte addressing so + 4)

Stages of the Datapath (2/5)

- Stage 2: Instruction Decode
  - upon fetching the instruction, we next gather data from the fields (decode all necessary instruction data)
  - first, read the `opcode` to determine instruction type and field lengths
  - second, read in data from all necessary registers
    - for `add`, read two registers
    - for `addi`, read one register
    - for `jal`, no reads necessary

Stages of the Datapath (3/5)

- Stage 3: ALU (Arithmetic-Logic Unit)
  - the real work of most instructions is done here: arithmetic (+, -, *, /), shifting, logic (&, |), comparisons (`slt`)

  - what about loads and stores?
    - `lw   $t0, 40($t1)`
    - the address we are accessing in memory = the value in `$t1` PLUS the value 40
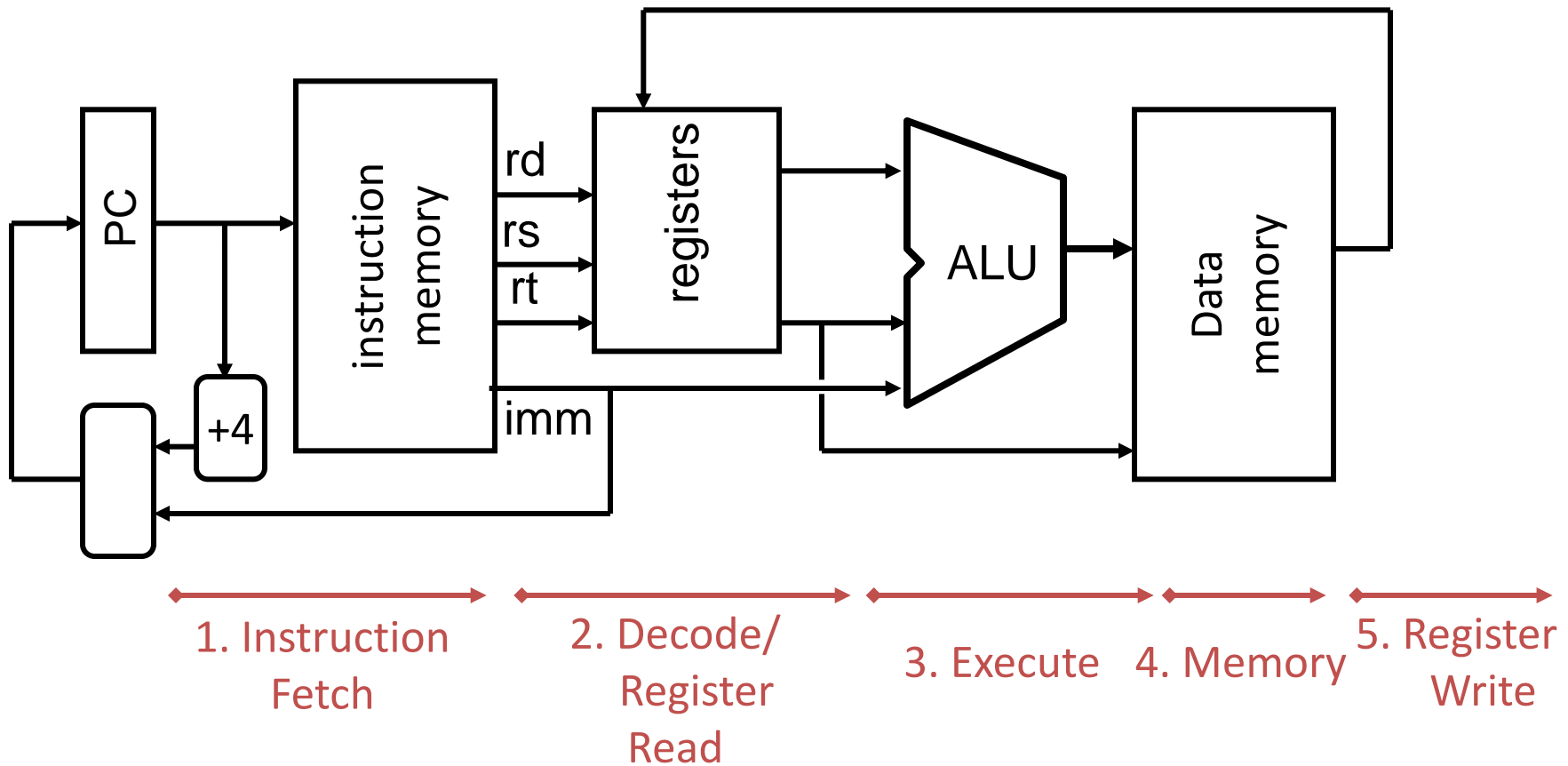   - so we do this addition in this stage

Stages of the Datapath (4/5)

- Stage 4: Memory Access
  - actually only the load and store instructions do anything during this stage; the others remain idle during this stage or skip it all together
  - since these instructions have a unique step, we need this extra stage to account for them
  - as a result of the cache system, this stage is expected to be fast

Stages of the Datapath (5/5)

- Stage 5: Register Write

  - most instructions write the result of some computation into a register

  - examples: arithmetic, logical, shifts, loads, slt

  - what about stores, branches, jumps?

    - don't write anything into a register at the end

   - these remain idle during this fifth stage or skip it all together

Generic Steps of Datapath

PC

instruction memory

+4

rd
rs
rt

imm

registers

ALU

Data memory

1. Instruction Fetch
2. Decode/ Register Read
3. Execute
4. Memory
5. Register Write

Datapath Walkthroughs (1/3)

- `add $r3,$r1,$r2  # r3 = r1+r2`
  - Stage 1: fetch this instruction, increment PC
  - Stage 2: decode to determine it is an `add`, then read registers `$r1` and `$r2`
  - Stage 3: add the two values retrieved in Stage 2
  - Stage 4: idle (nothing to write to memory)
  - Stage 5: write result of Stage 3 into register `$r3`

Example: add Instruction
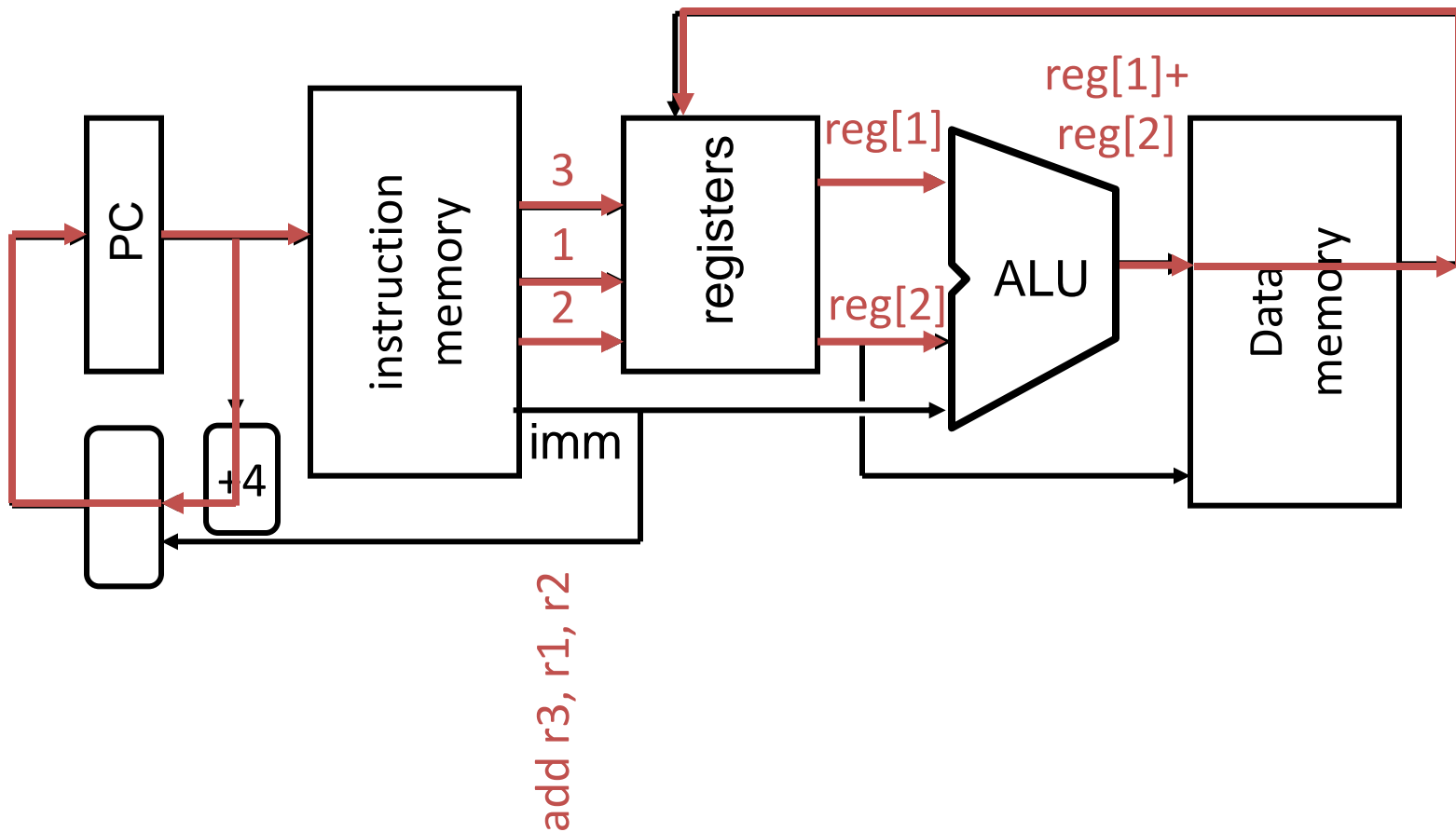
Datapath Walkthroughs (2/3)

- `slti $r3,$r1,17`
  `# if (r1 <17 )r3 = 1 else r3 = 0`
  - Stage 1: fetch this instruction, increment PC
  - Stage 2: decode to determine it is an `slti`, then read register `$r1`
  - Stage 3: compare value retrieved in Stage 2 with the integer 17
  - Stage 4: idle
  - Stage 5: write the result of Stage 3 (1 if reg source was less than signed immediate, 0 otherwise) into register `$r3`

Example: `slti` Instruction
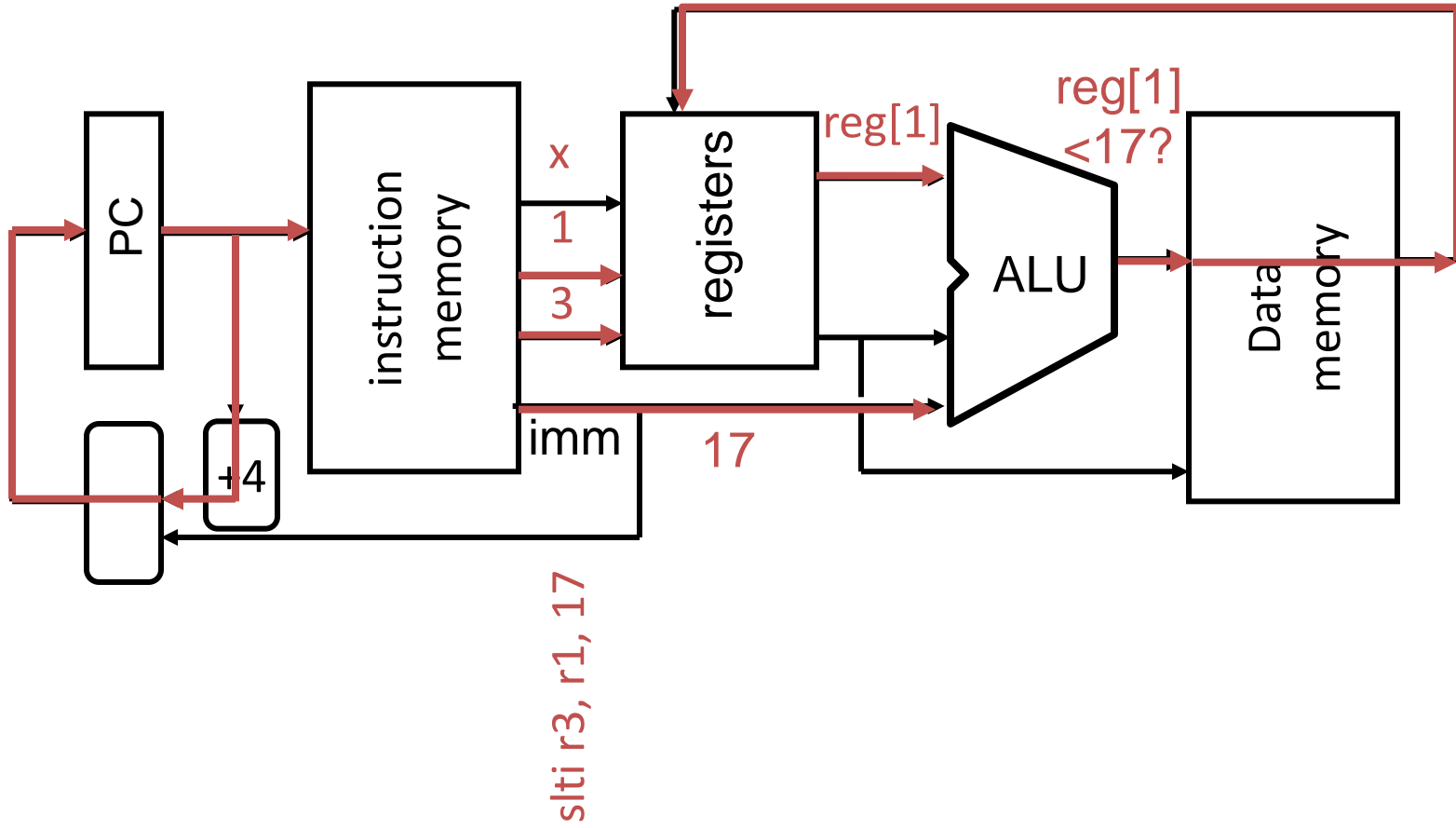
Datapath Walkthroughs (3/3)

- `sw $r3,17($r1) #`
`Mem[r1+17]=r3`

  – Stage 1: fetch this instruction, increment PC

  – Stage 2: decode to determine it is a `sw`, then read registers `$r1` and `$r3`

  – Stage 3: add `17` to value in register `$r1` (retrieved in Stage 2) to compute address

  – Stage 4: write value in register `$r3` (retrieved in Stage 2) into memory address computed in Stage 3

  – Stage 5: idle (nothing to write into a register)

Example: sw Instruction
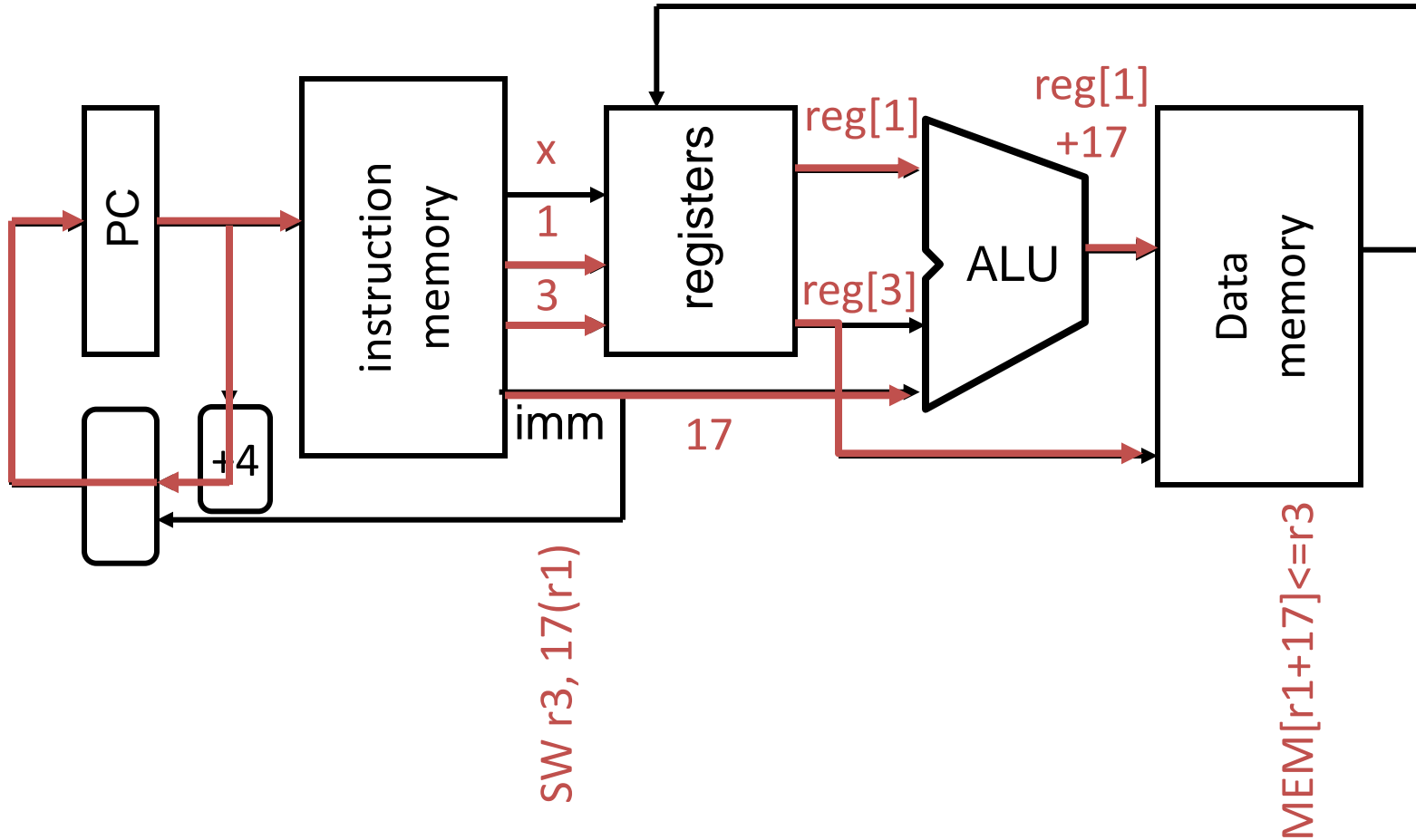
Why Five Stages? (1/2)

- Could we have a different number of stages?
  - Yes, and other architectures do
- So why does MIPS have five if instructions tend to idle for at least one stage?
  - Five stages are the union of all the operations needed by all the instructions.
 - One instruction uses all five stages: the load

Why Five Stages? (2/2)

- `lw $r3,17($r1)  #`
`r3=Mem[r1+17]`
  - Stage 1: fetch this instruction, increment PC
  - Stage 2: decode to determine it is a `lw`, then read register `$r1`
  - Stage 3: add `17` to value in register `$r1` (retrieved in Stage 2)
  - Stage 4: read value from memory address computed in Stage 3
  - Stage 5: write value read in Stage 4 into register `$r3`

Example: `lw` Instruction
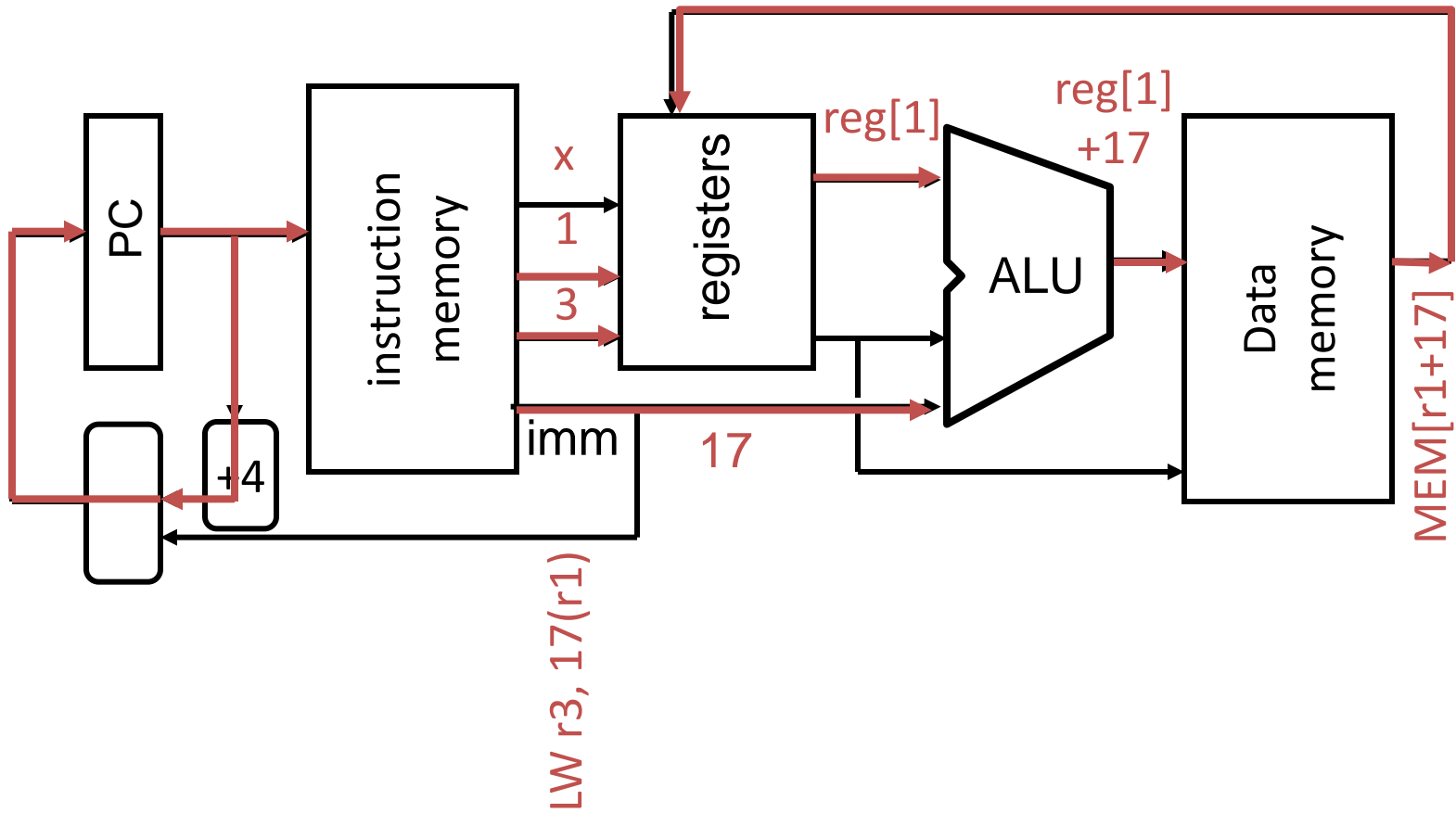
Datapath and Control

- Datapath based on data transfers required to perform instructions
- Controller causes the right transfers to happen
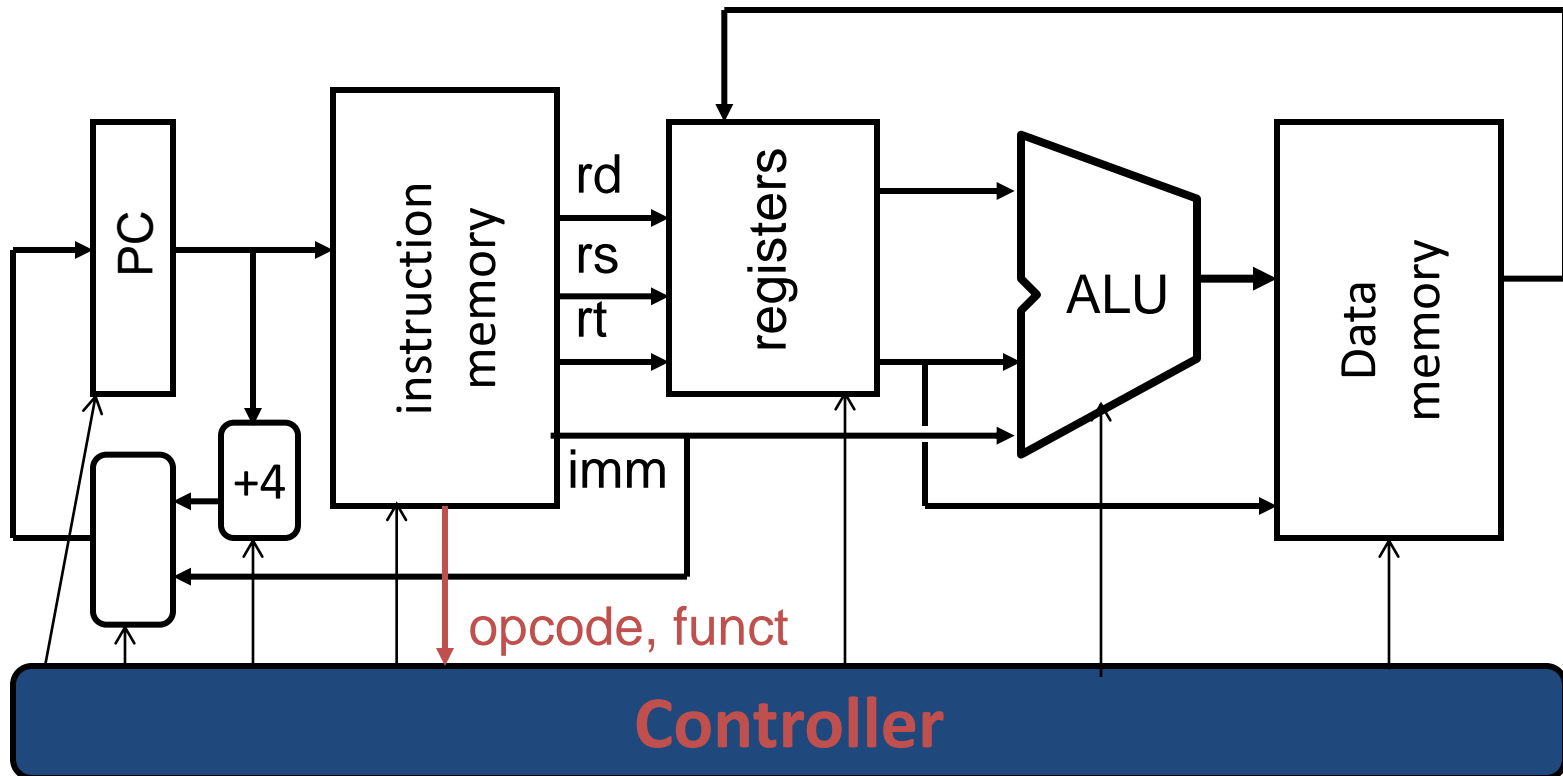
What Hardware Is Needed? (1/2)

- PC: a register that keeps track of address of the *next* instruction to be fetched

- General Purpose Registers
  - Used in Stages 2 (Read) and 5 (Write)
  - MIPS has 32 of these

- Memory
  - Used in Stages 1 (Fetch) and 4 (R/W)
  - Caches makes these stages as fast as the others (on average, otherwise multicycle stall)

What Hardware Is Needed? (2/2)

- ALU
  - Used in Stage 3
  - Performs all necessary functions: arithmetic, logicals, etc.

- Miscellaneous Registers
  - One stage per clock cycle: Registers inserted between stages to hold intermediate data and control signals as they travel from stage to stage
  - Note: Register is a general purpose term meaning something that stores bits. Realize that not all registers are in the "register file"

CPU Clocking (1/2)

- For each instruction, how do we control the flow of information though the datapath?

- Single Cycle CPU: All stages of an instruction completed within one long clock cycle

  – Clock cycle sufficiently long to allow each instruction to complete all stages without interruption within one cycle

1. Instruction Fetch   2. Decode/ Register Read   3. Execute   4. Memory   5. Reg. Write
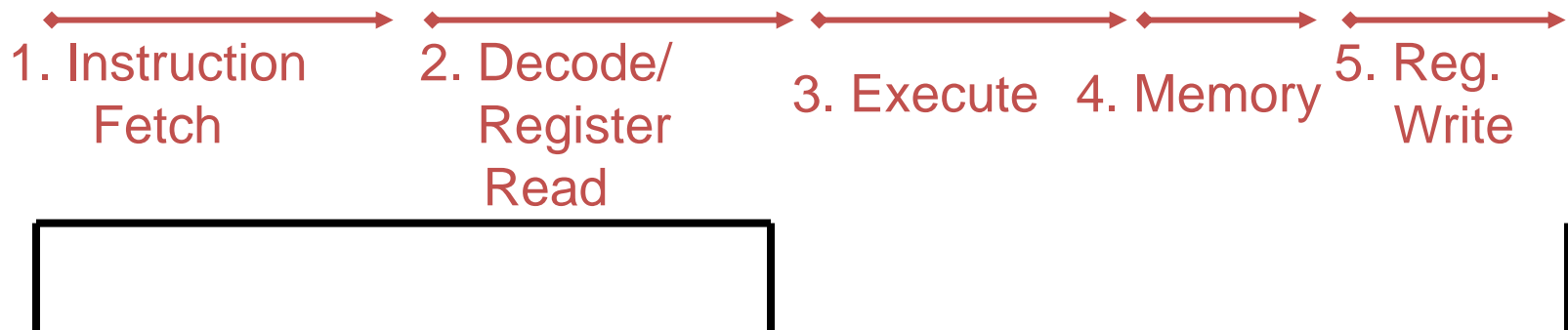
CPU Clocking (2/2)

- Alternative multiple-cycle CPU: only one stage of instruction per clock cycle
  - Clock is made as long as the slowest stage

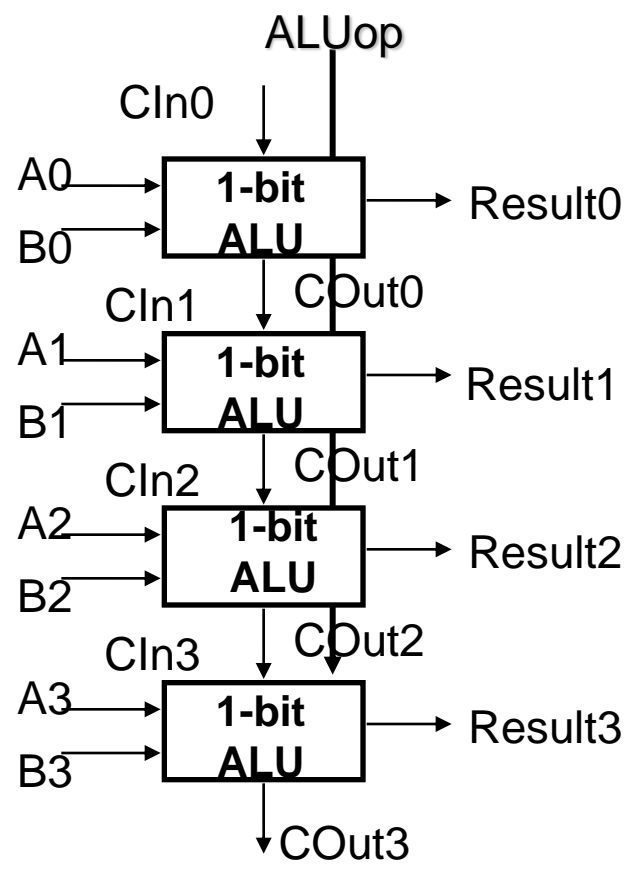| 1. Instruction Fetch | 2. Decode/ Register Read | 3. Execute | 4. Memory | 5. Register Write |
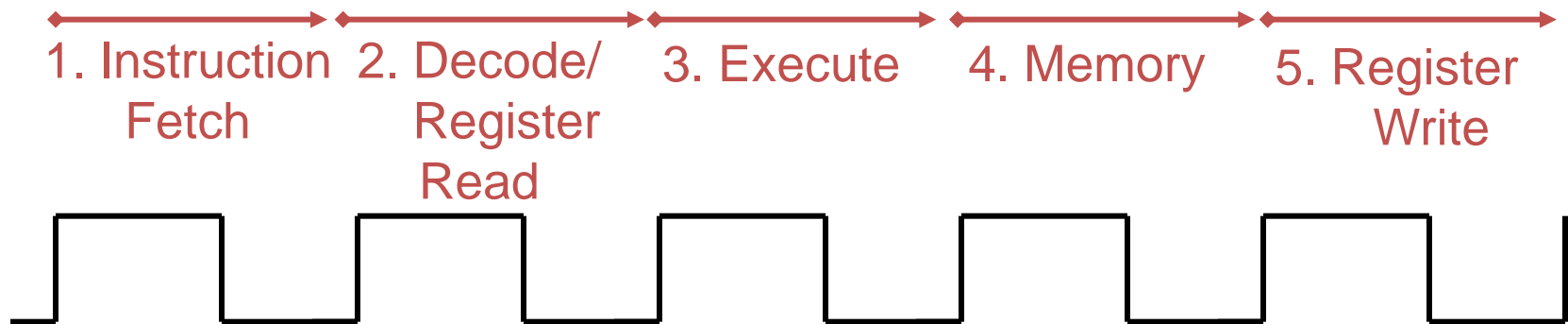
  - Several significant advantages over single cycle execution: Unused stages in a particular instruction can be skipped OR instructions can be pipelined (overlapped)
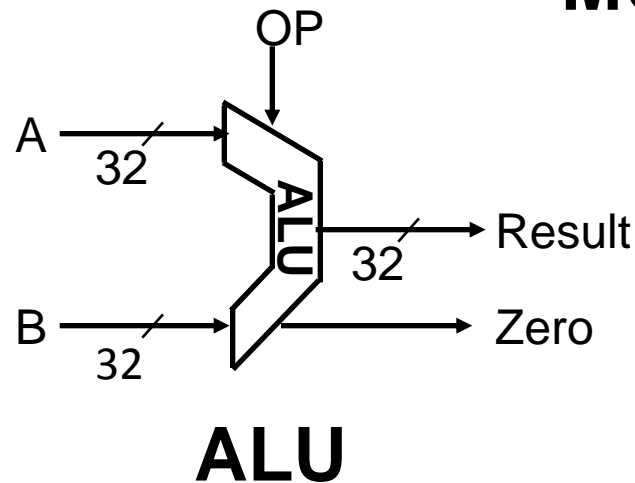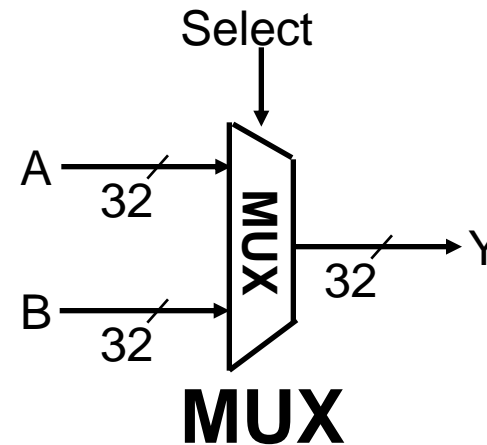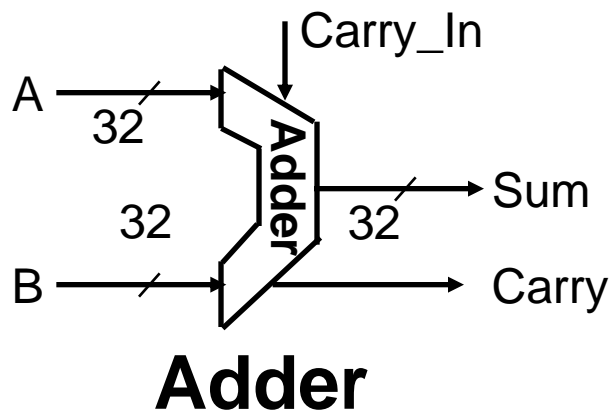
1 bit ALU

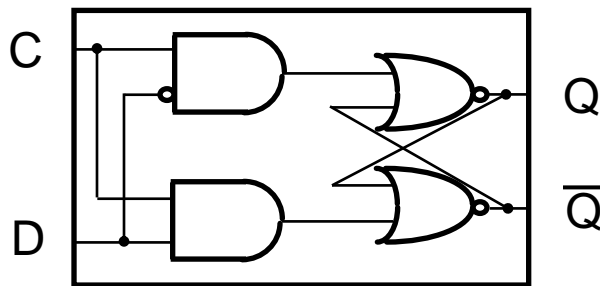- Using a MUX we can add the AND, OR, and adder operations into a single ALU
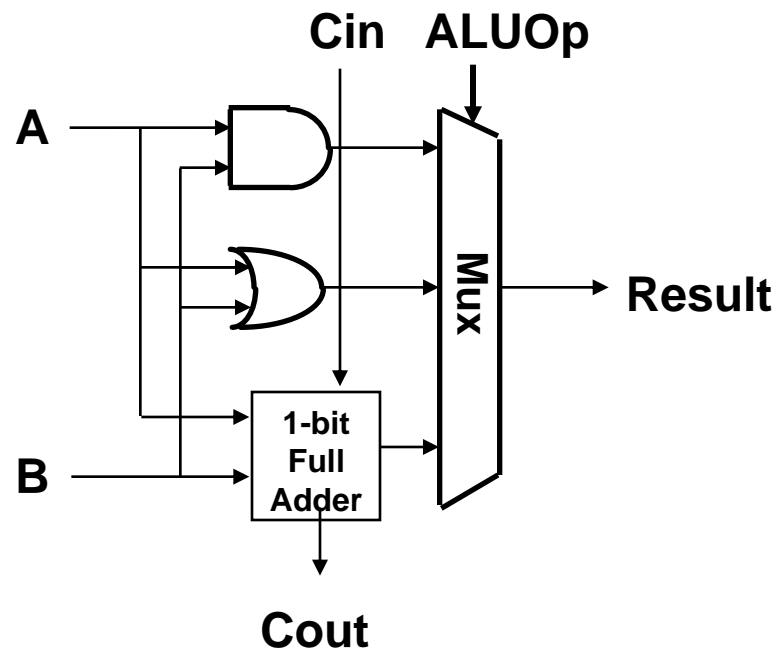
4 bit ALU
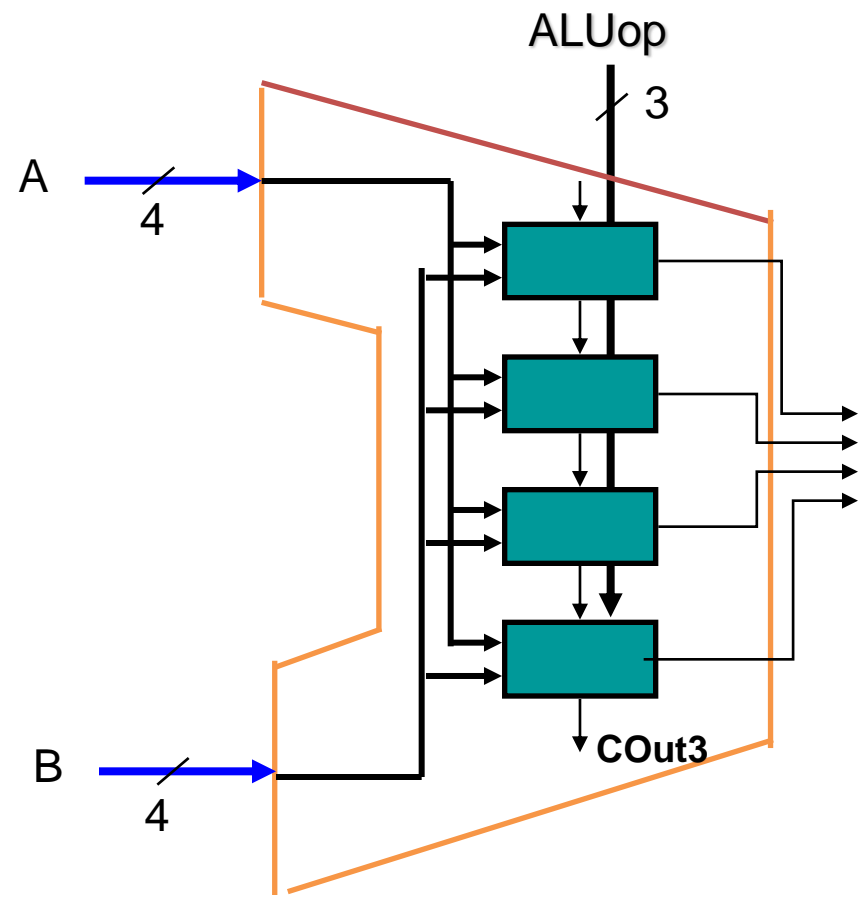
Combinational Elements
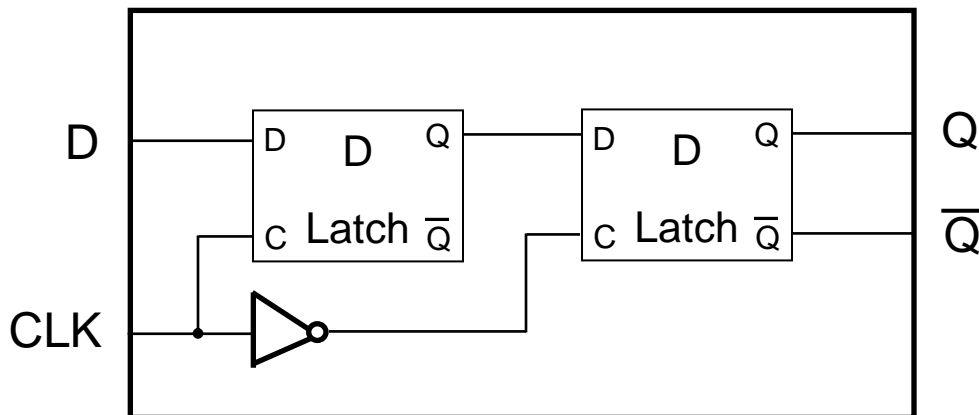

Adder

MUX

ALU

D Latches

- Modified SR Latch

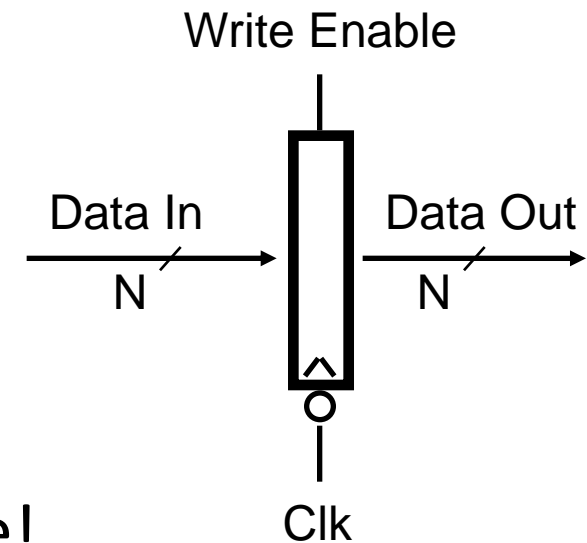- Latches value when C is asserted

D Flip Flop

- Uses Master/Slave D Latches

Storage Element: Register

- Register
  - Similar to D Flip Flop
    - N bit input and output
    - Write Enable input
  - Write Enable
    - 0: Data Out will not change
    - 1: Data Out will become Data In
  - Data changes only on falling edge!

Storage Element: Reg File

- Register File consists of 32 registers
  - Two 32 bit output busses
    - busA and busB
  - One 32 bit input bus
    - busW
  - Register 0 hard wired to value 0
  - Register selected by
    - RA selects register to put on busA
    - RB selects register to put on busB
    - RW selects register to be written via busW when Write Enable is 1
  - Clock input (CLK)
    - CLK input is a factor only for write operation
    - During read, behaves as combinational logic block
      - RA or RB stable $\Rightarrow$ busA or busB valid after "access time"
      - Minor simplification of reality

RW RA RB

Write Enable
(0/1)

5   5   5

busW

32

**32 32-bit
Registers**

Clk

busA

32

busB

32
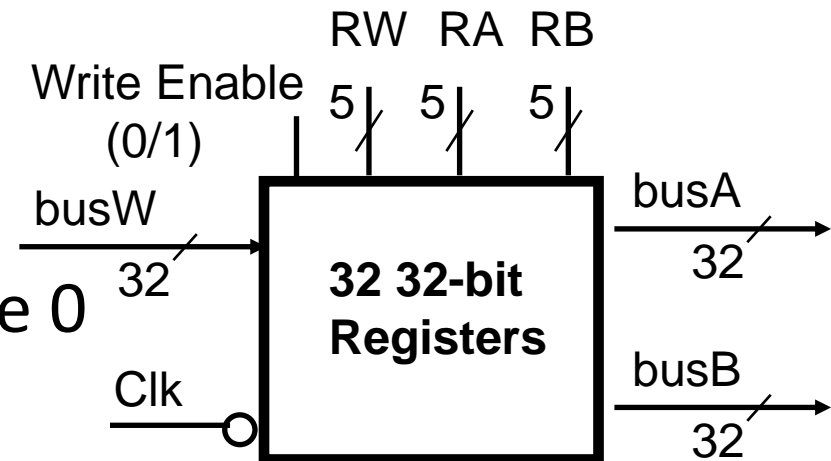
Storage Element: Memory

- Memory
  - One input bus: Data In
  - One output bus: Data Out
  - Address selection
    - Address selects the word to put on Data Out
    - To write to address, set Write Enable to 1
  - Clock input (CLK)
    - CLK input is a factor only for write operation
    - During read, behaves as combinational logic block
      - Valid Address $\Rightarrow$ Data Out valid after "access time"
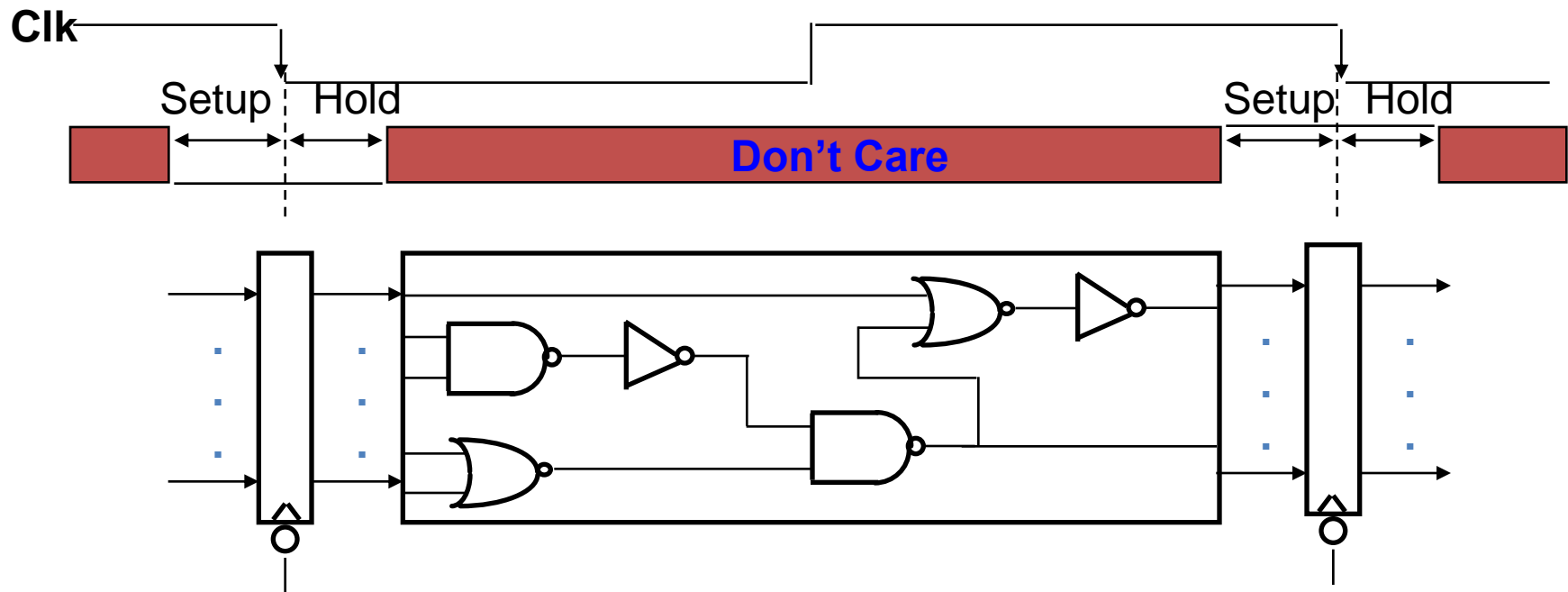   - Minor simplification of reality

Address

Write Enable

Data In

32

Clk

Data Out

32
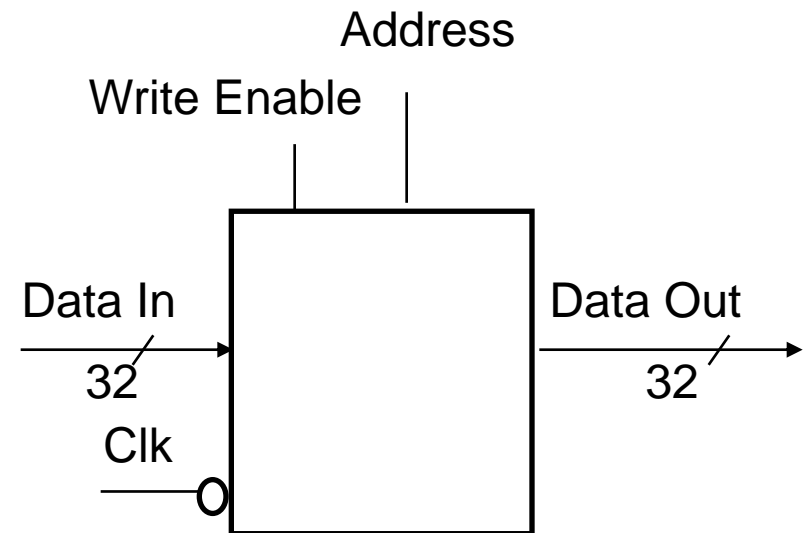
Some Logic Design…

- All storage elements have same clock
  - Edge-triggered clocking
  - "Instantaneous" state change (simplification!)
  - Timing always work if the clock is slow enough

Cycle Time = Clk-to-Q + Longest Delay + Setup + Clock Skew

Summary

- CPU design involves Datapath, Control
  - 5 Stages for MIPS Instructions
    1. Instruction Fetch
    2. Instruction Decode & Register Read
    3. ALU (Execute)
    4. Memory
    5. Register Write

- Datapath timing: single long clock cycle or one short clock cycle per stage