

CSE 31

Computer Organization

Lecture 22 – CPU Design (1)

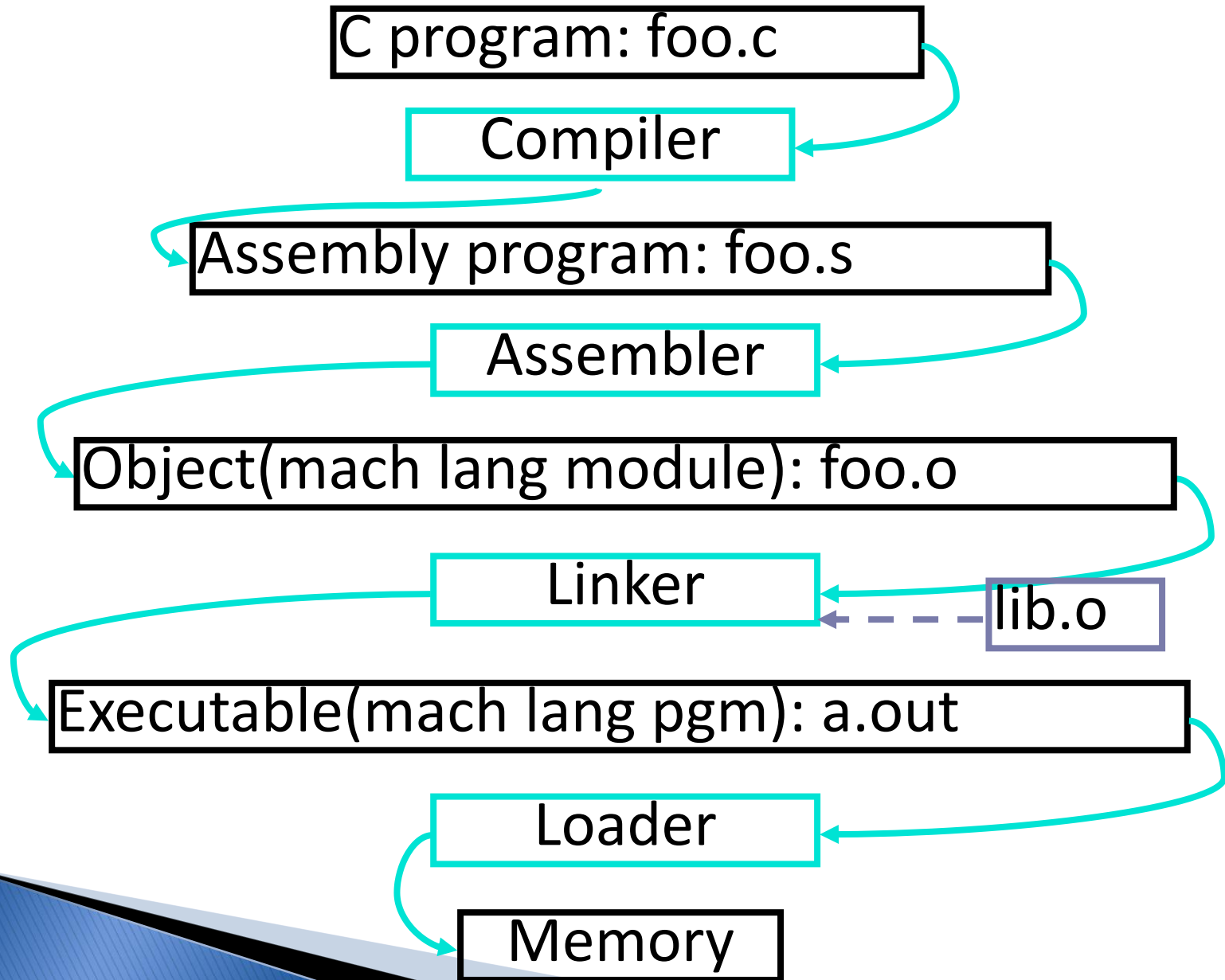
Announcement

- ▶ No lab this week
 - Project #1 grading during lab
- ▶ HW #6 out at CatCourses (not zyBooks)
 - Due Friday (11/30, extended) at 11:59pm
- ▶ Project #2
 - Due Monday (12/3)
 - Don't start late, you won't have time!
- ▶ Course evaluation online
 - Fill out by 12/6 (Thursday)
- ▶ Reading assignment
 - Chapter 5.1 – 5.6 of zyBooks (Reading Assignment #6)
 - Make sure to do the Participation Activities
 - Due Monday (11/26)

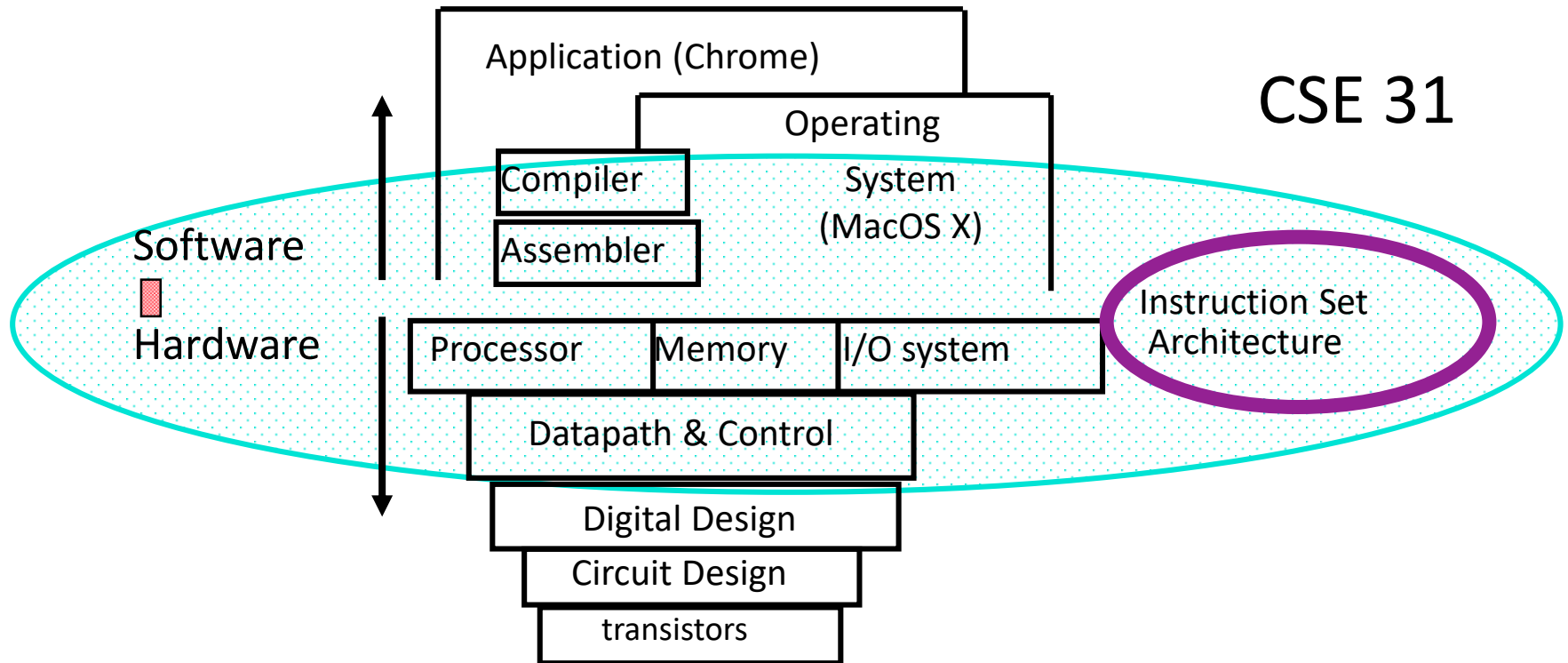
Announcement

- ▶ Midterm Exam 2
 - 11/28 (Wednesday, in lecture) Not 11/7 as scheduled
 - Lectures #8 - #18
 - HW #3 - #5
 - Practice exam in CatCourses
 - Closed book
 - 1 sheet of note (8.5" x 11")
 - MIPS reference sheet will be provided

Review



What are “Machine Structures”?



Coordination of many **levels of abstraction**

ISA is an important abstraction level:
contract between HW & SW

Below the Program

- High-level language program (in C)

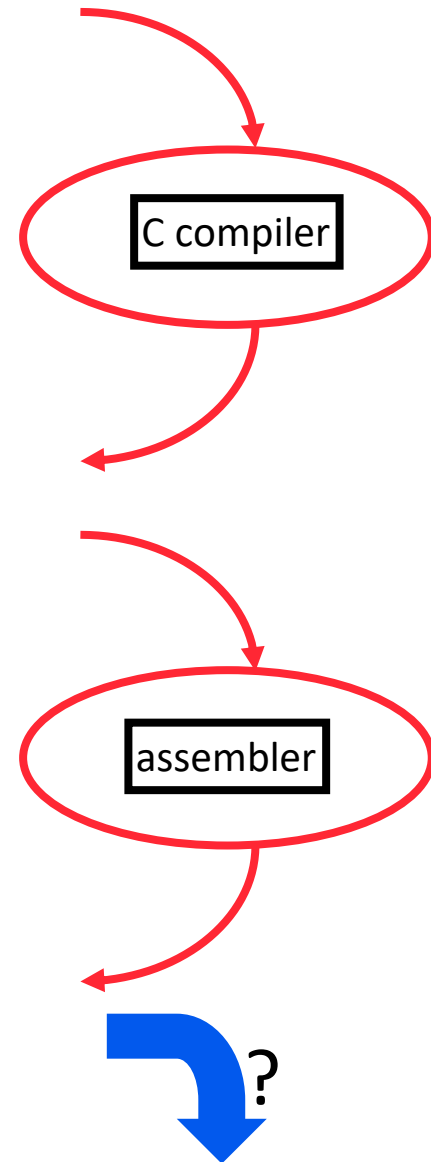
```
swap  int v[], int k){  
    int temp;  
    temp = v[k];  
    v[k] = v[k+1];  
    v[k+1] = temp;  
}
```

- Assembly language program (for MIPS)

```
swap: sll    $2, $5, 2  
      add    $2, $4, $2  
      lw     $15, 0($2)  
      lw     $16, 4($2)  
      sw     $16, 0($2)  
      sw     $15, 4($2)  
      jr     $31
```

- Machine (object) code (for MIPS)

```
000000 00000 00101 0001000010000000  
000000 00100 00010 0001000000100000 . . .
```



Synchronous Digital Systems

The hardware of a processor, such as the MIPS, is an example of a Synchronous Digital System

Synchronous:

- Means all operations are coordinated by a central clock.
 - It keeps the “heartbeat” of the system!

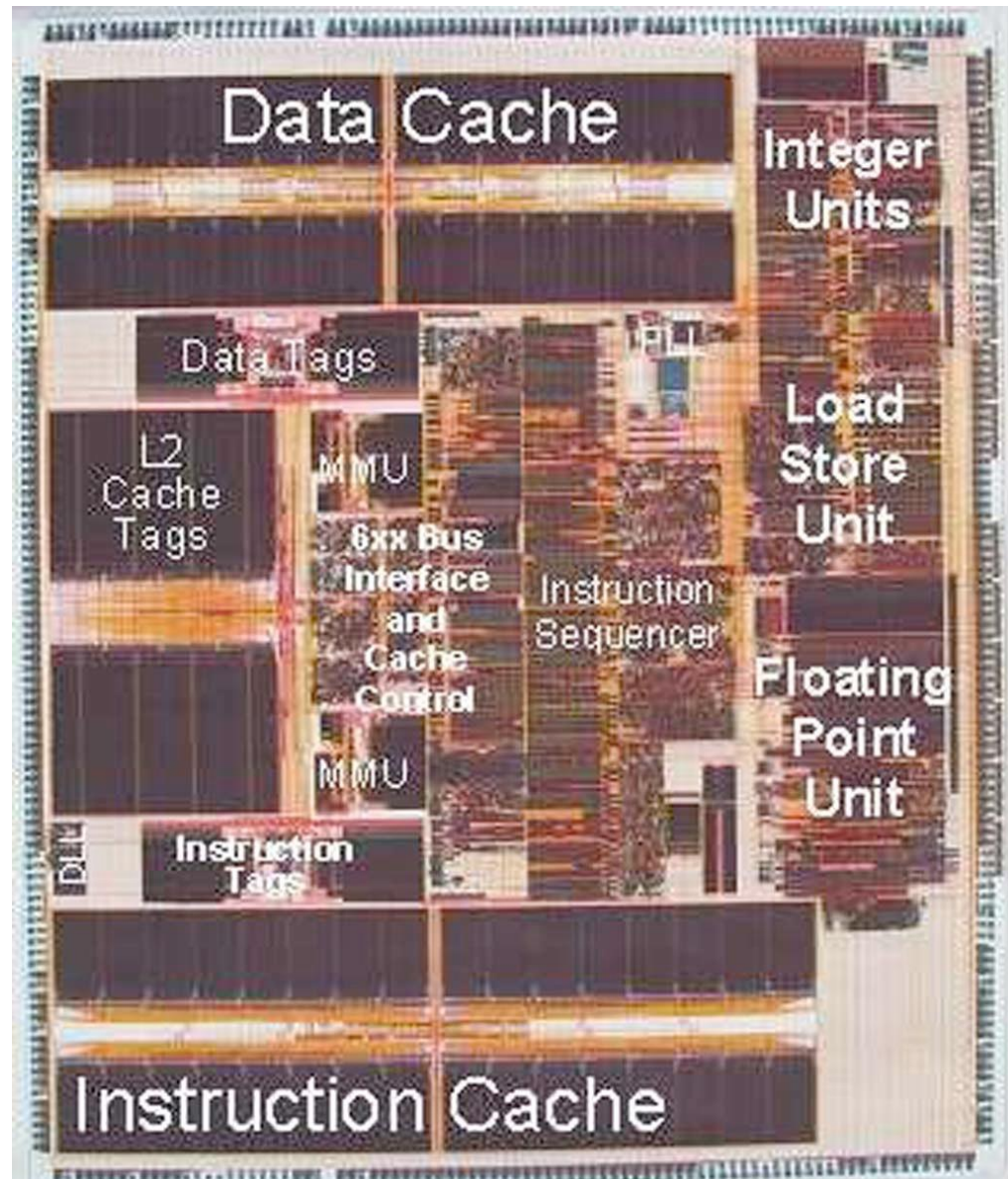
Digital:

- Mean all values are represented by discrete values
- Electrical signals are treated as 1's and 0's and grouped together to form words.

PowerPC Die Photograph

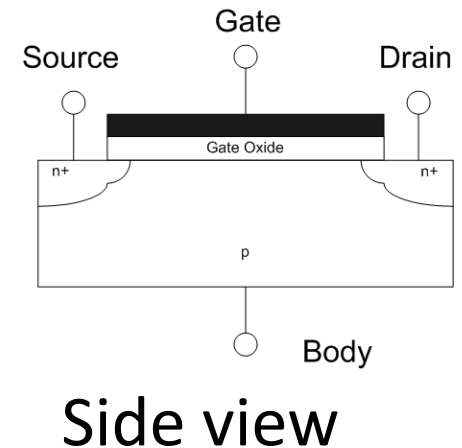
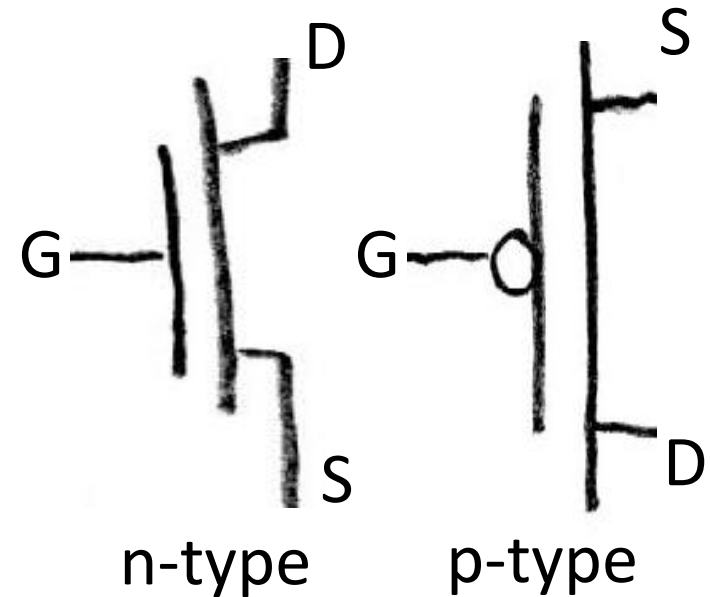


Let's look
closer...



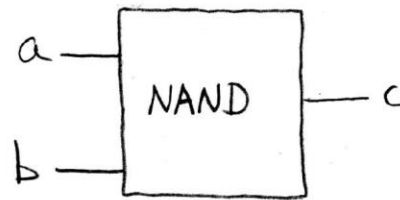
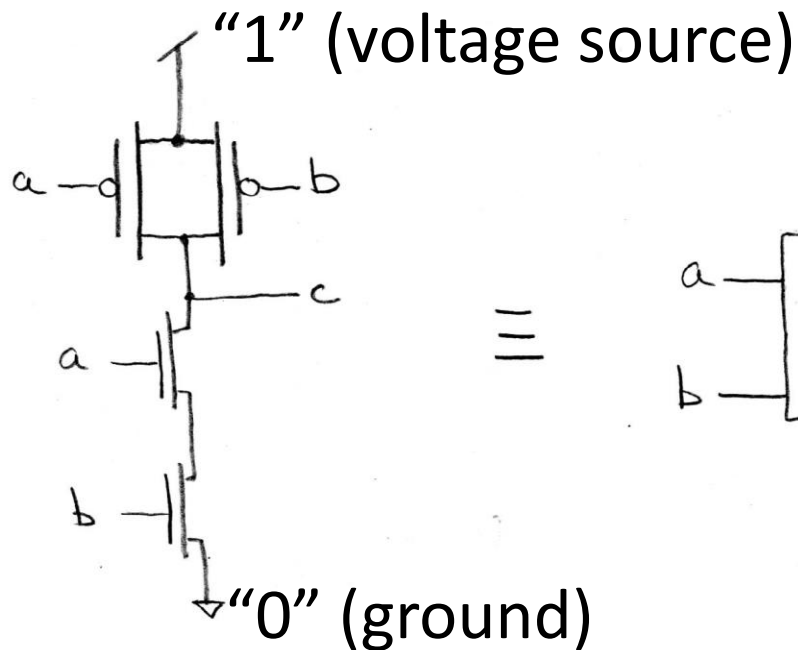
Transistors 101

- MOSFET
 - Metal-Oxide-Semiconductor Field-Effect Transistor
 - Come in two types:
 - n-type NMOSFET
 - p-type PMOSFET
- For n-type (p-type opposite)
 - If voltage not enough between G & S, transistor turns “off” (cut-off) and Drain-Source NOT connected
 - If the G & S voltage is high enough, transistor turns “on” (saturation) and Drain-Source ARE connected



Transistor Circuit Rep. vs. Block diagram

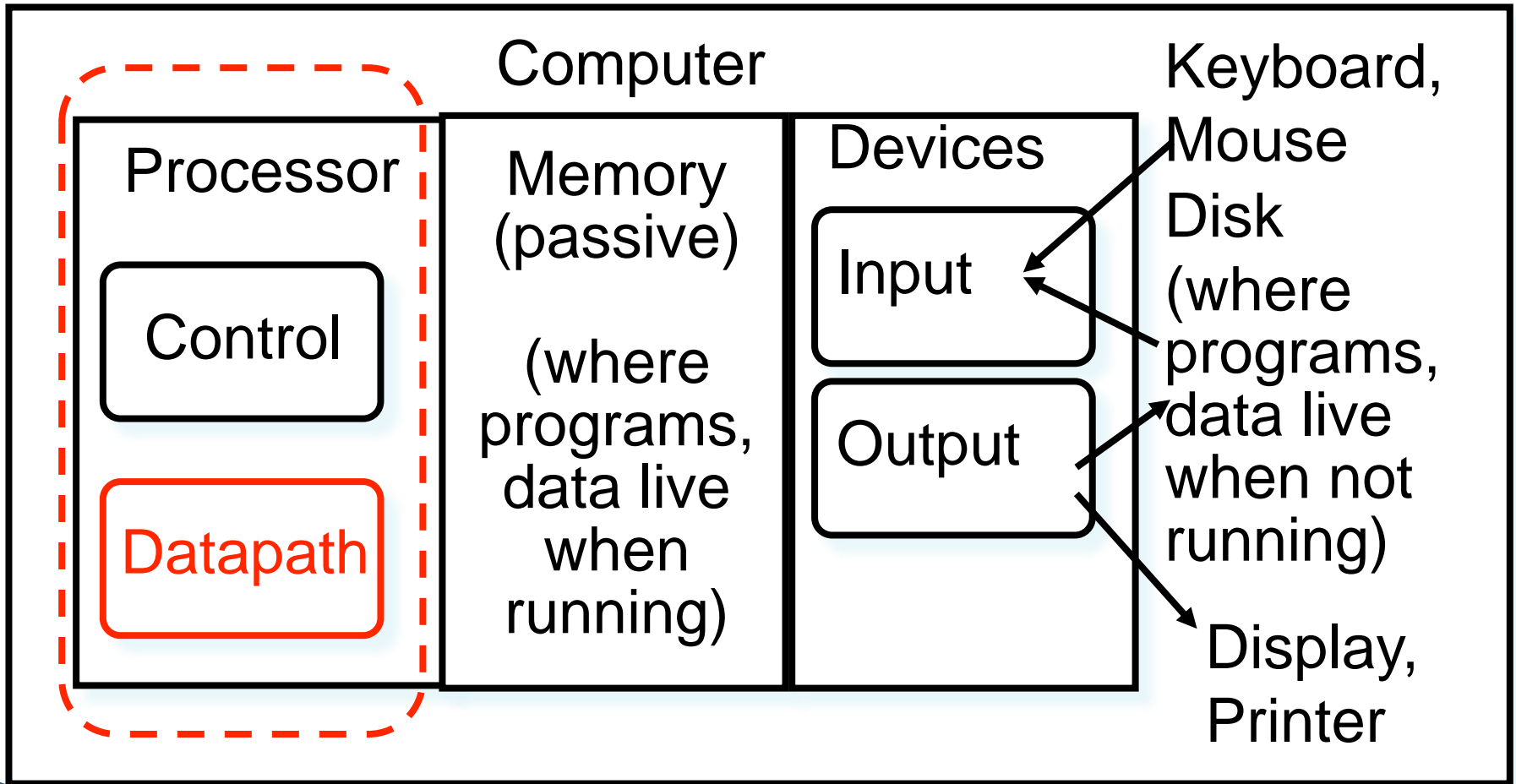
- Chips is composed of nothing but transistors and wires.
- Small groups of transistors form useful building blocks.



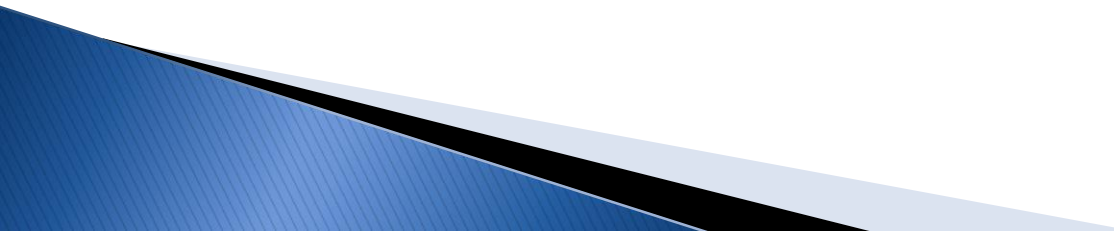
a	b	c
0	0	1
0	1	1
1	0	1
1	1	0

- Block are organized in a hierarchy to build higher-level blocks: ex: adders.

Five Components of a Computer



The CPU

- ▶ **Processor (CPU)**: the active part of the computer, which does all the work (data manipulation and decision-making)
 - ▶ **Datapath**: portion of the processor which contains hardware necessary to perform operations required by the processor (the brawn)
 - ▶ **Control**: portion of the processor (also in hardware) which tells the datapath what needs to be done (the brain)
- 

Stages of the Datapath

- ▶ Problem: a single, atomic block which “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

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 type is, 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 (R-format)
 - for `addi`, read one register (I-format)
 - for `jal`, no reads necessary (J-format)

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, do they need this stage?
 - 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

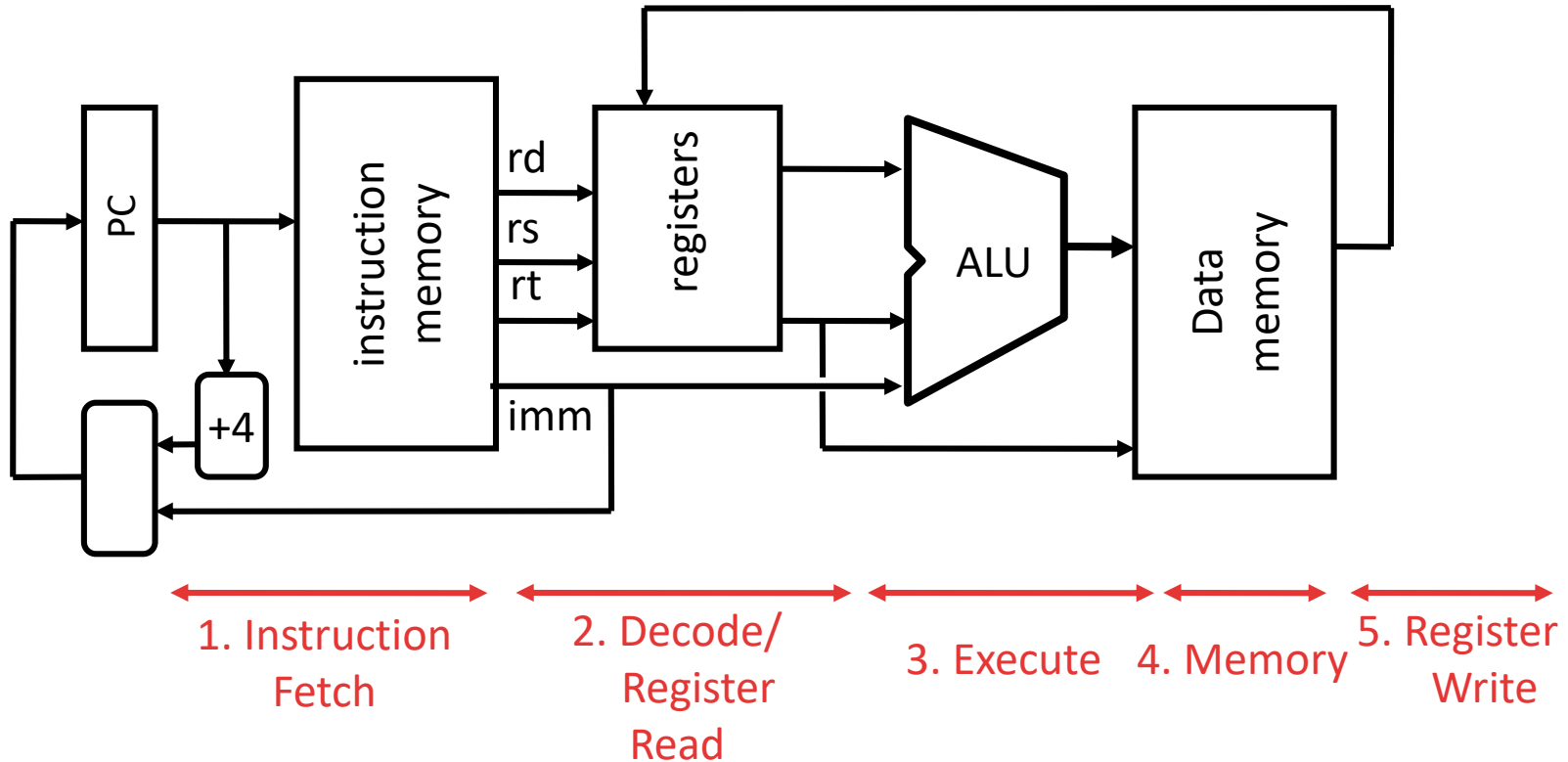
- actually only the load and store instructions do anything during this stage; the others remain idle during this stage or skip it all together
- since these instructions have a unique step, we need this extra stage to account for them
- as a result of the cache system, this stage is expected to be fast

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

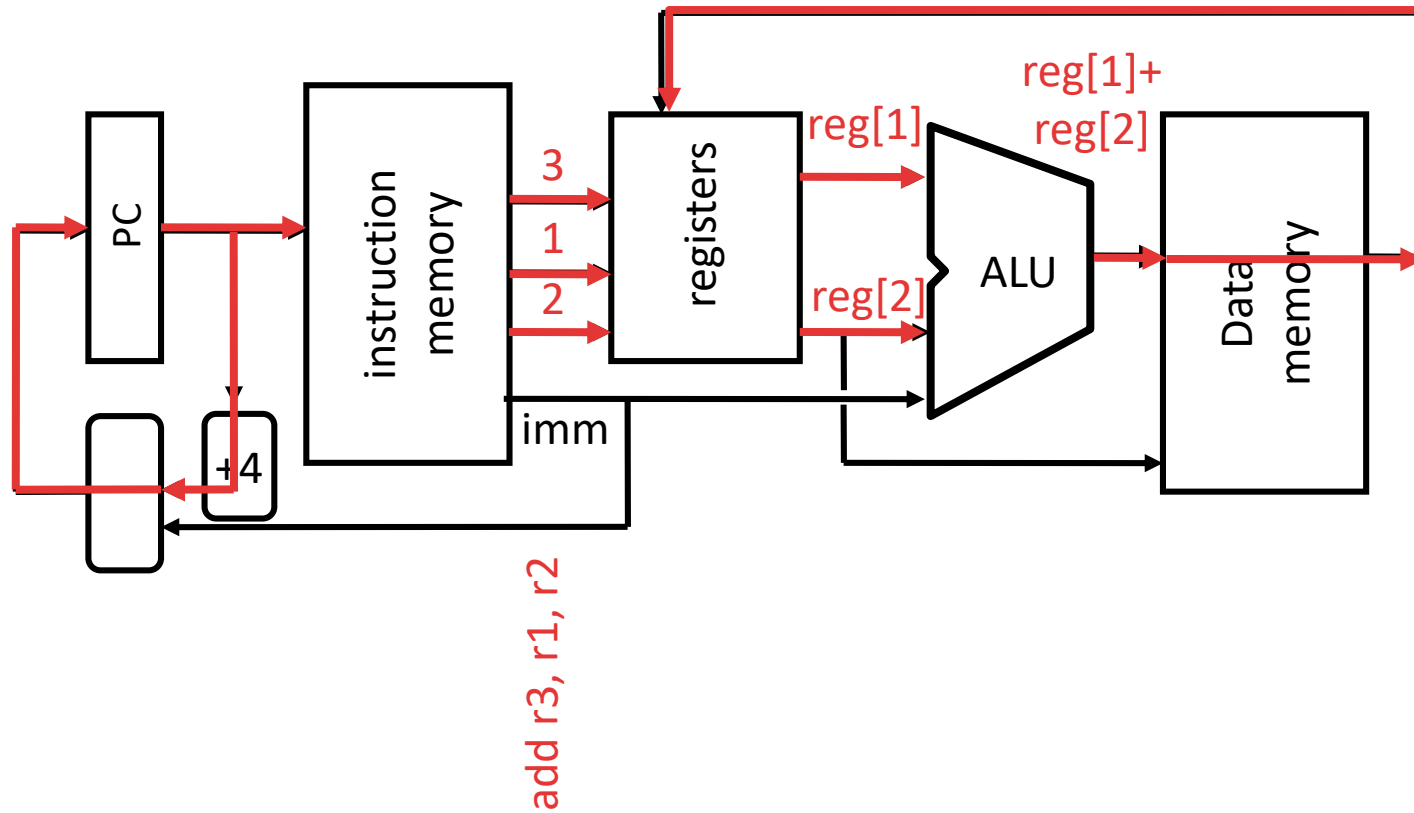
Generic Diagram of Datapath



Datapath Walkthroughs (1/3)

- ▶ `add $r3,$r1,$r2 # r3 = r1+r2`
 - Stage 1: fetch this instruction, inc. PC
 - Stage 2: decode to find it's an `add`, then read registers `$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

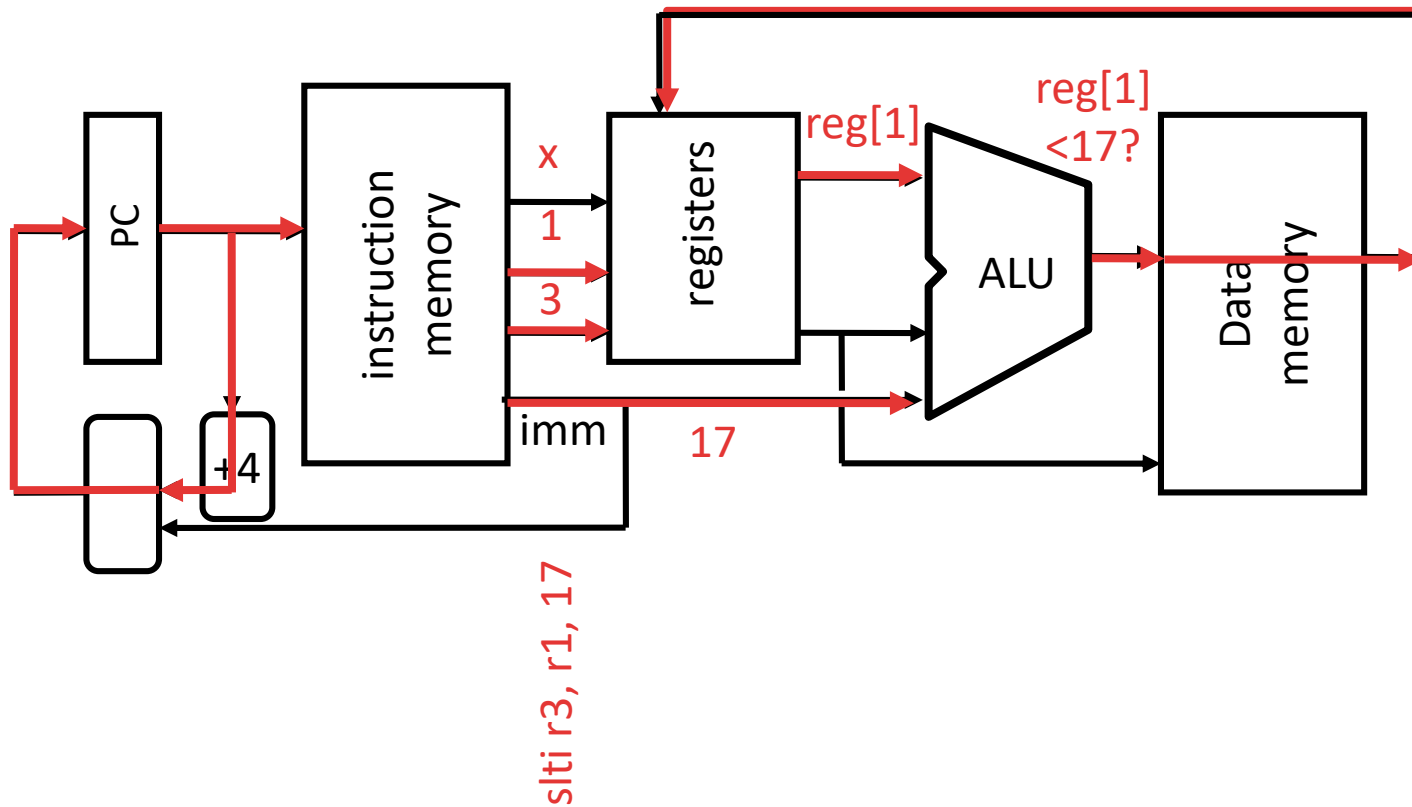


Datapath Walkthroughs (2/3)

▶ `slti $r3, $r1, 17`

- Stage 1: fetch this instruction, inc. PC
- Stage 2: decode to find it's an `slti`, then read register `$r1`
- Stage 3: compare value retrieved in Stage 2 with the integer 17
- Stage 4: idle
- Stage 5: write the result of Stage 3 in register `$r3`

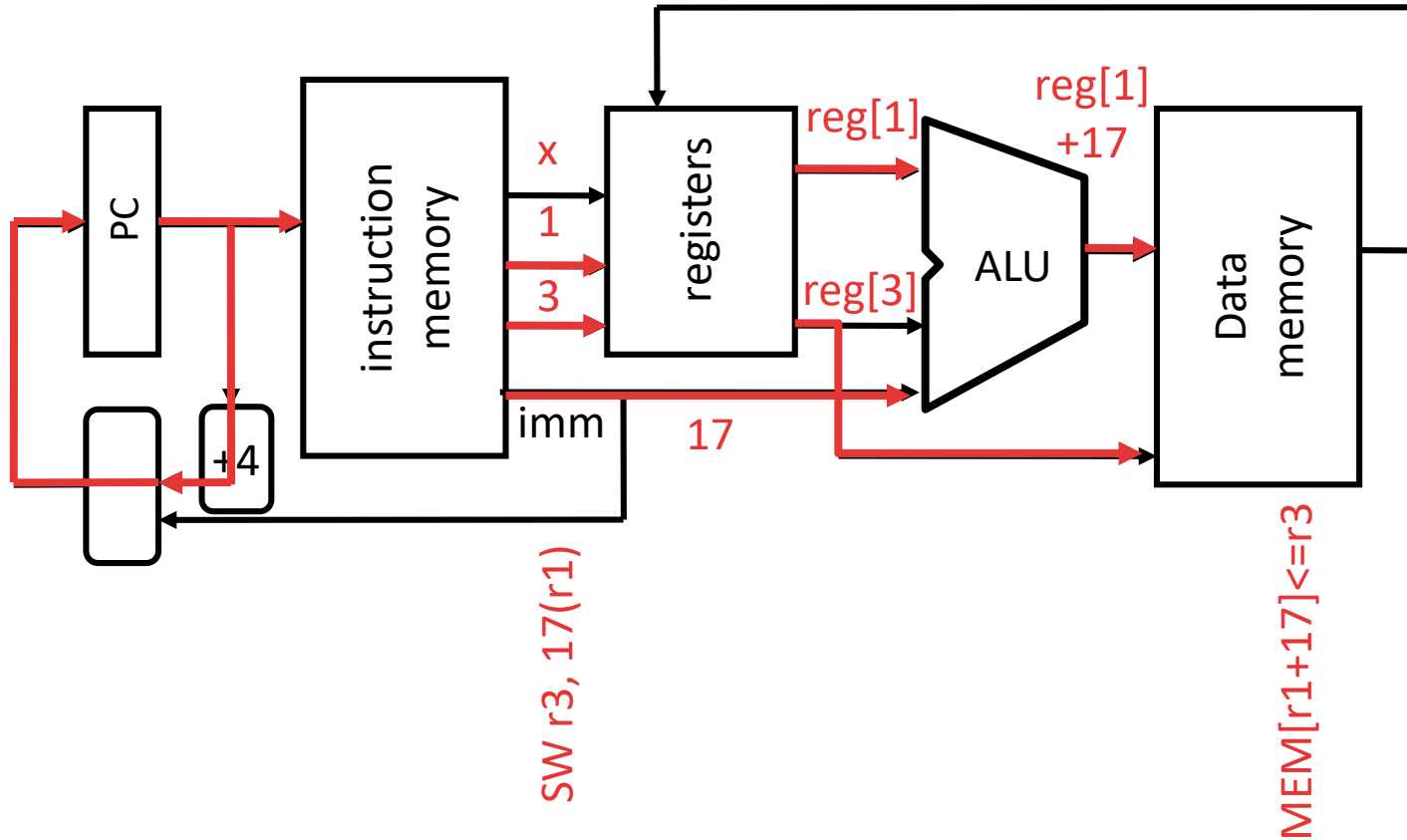
Example: `slti` Instruction



Datapath Walkthroughs (3/3)

- ▶ `sw $r3, 17($r1)`
 - Stage 1: fetch this instruction, inc. PC
 - Stage 2: decode to find it's a `sw`, then read registers `$r1` and `$r3`
 - Stage 3: add 17 to value in register `$r1` (retrieved in Stage 2)
 - Stage 4: write value in register `$r3` (retrieved in Stage 2) into memory address computed in Stage 3
 - Stage 5: idle (nothing to write into a register)

Example: sw Instruction



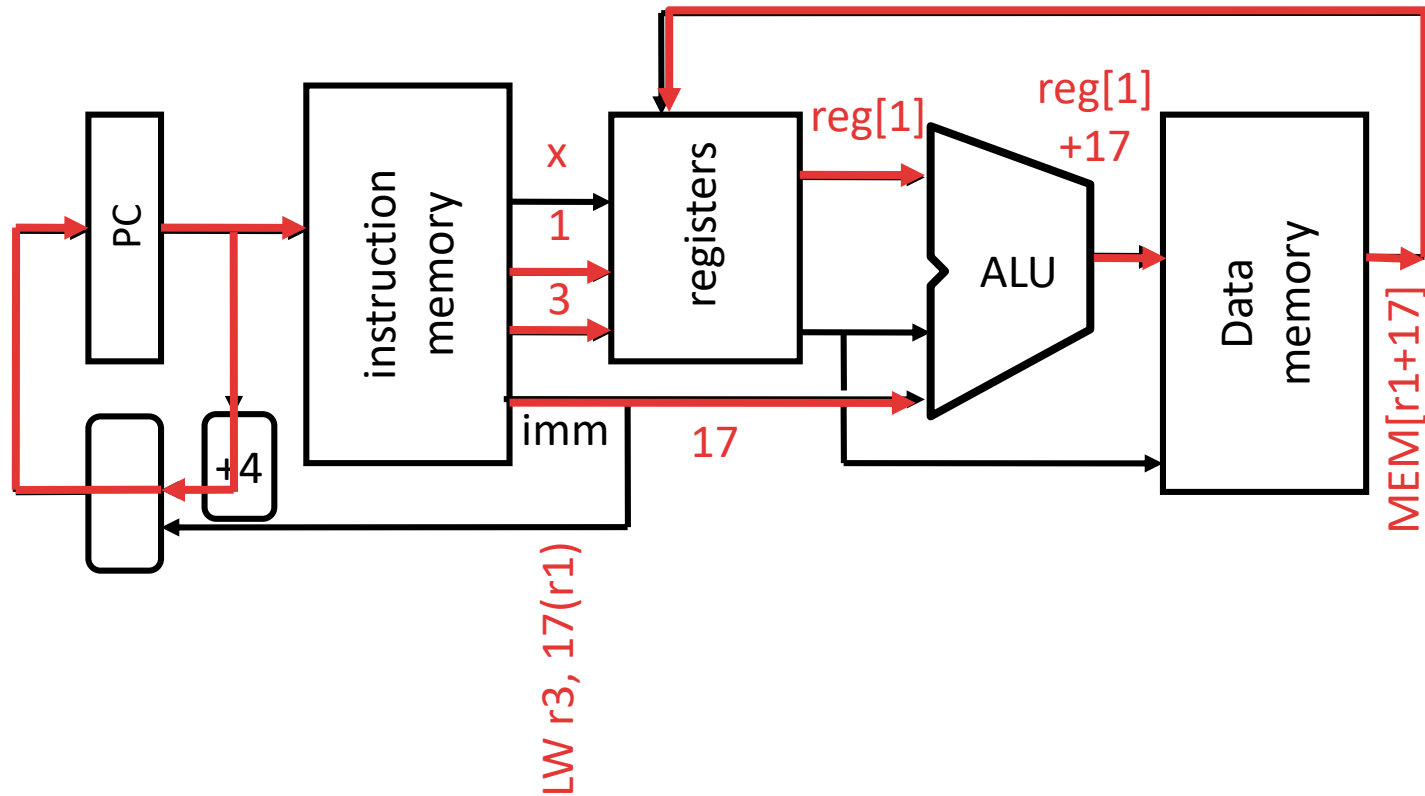
Why Five Stages? (1/2)

- ▶ 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?
 - The five stages are the union of all the operations needed by all the instructions.
 - There is one instruction that uses all five stages: the **load**

Why Five Stages? (2/2)

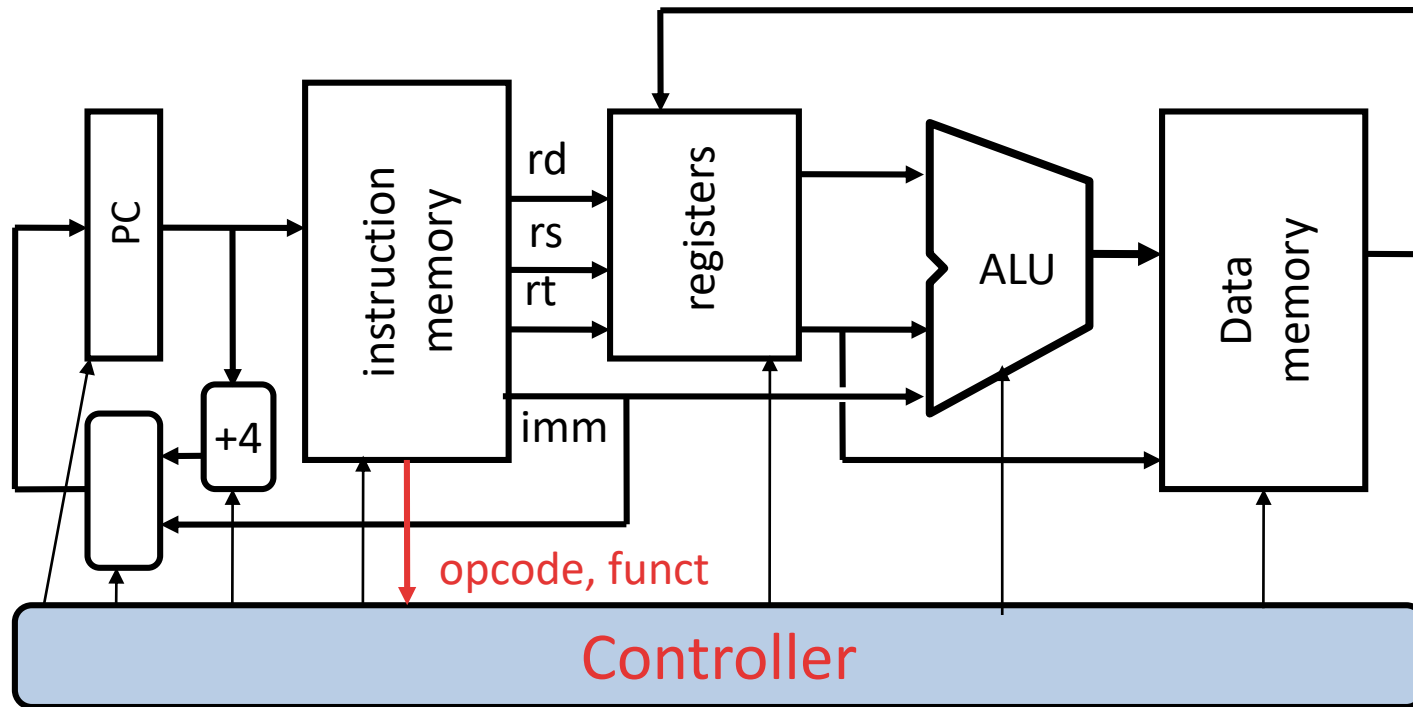
- ▶ `lw $r3, 17($r1)`
 - Stage 1: fetch this instruction, inc. PC
 - Stage 2: decode to find it's a `lw`, then read register `$r1`
 - Stage 3: add `17` to value in register `$r1` (retrieved in Stage 2)
 - Stage 4: read value from memory address compute in Stage 3
 - Stage 5: write value found in Stage 4 into register `$r3`

Example: 1w Instruction



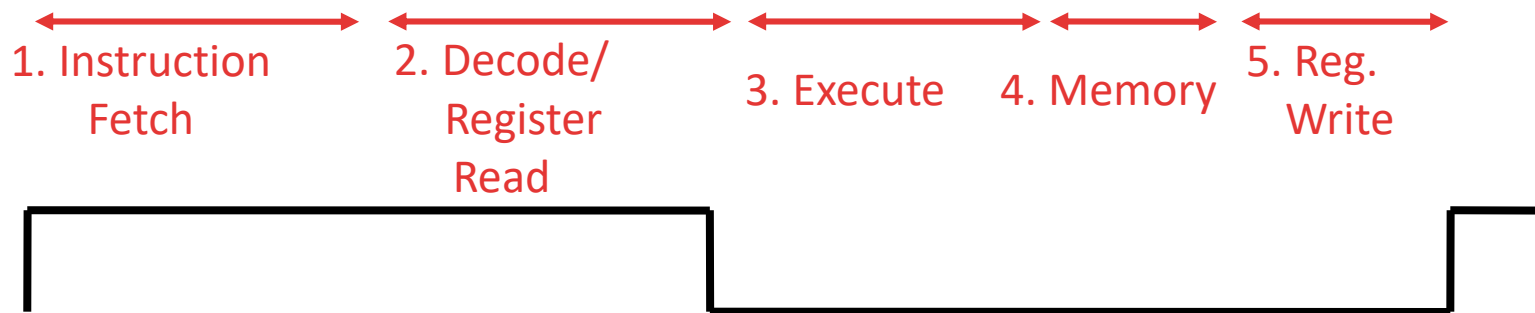
Datapath Summary

- Datapath based on data transfers required to perform instructions
- Controller causes the right transfers to happen



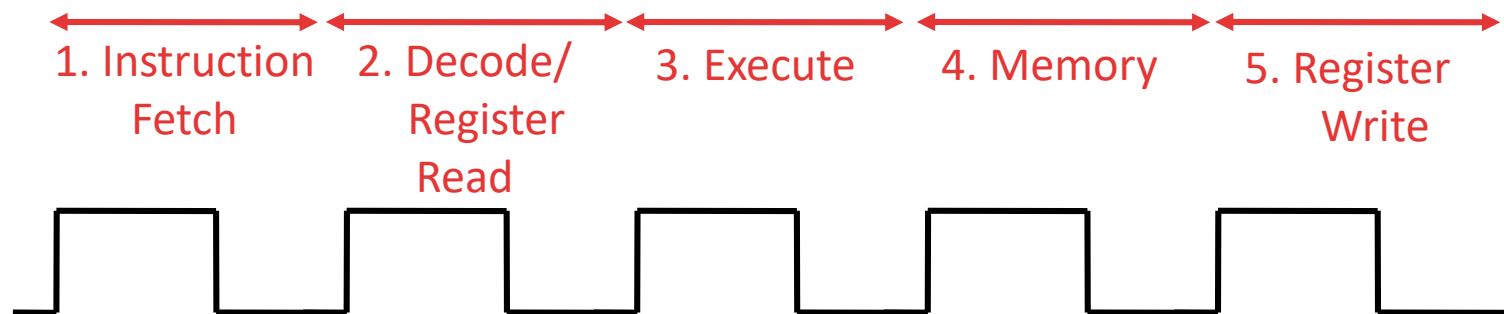
CPU clocking (1/2)

- For each instruction, how do we control the flow of information through 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



CPU clocking (2/2)

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



- Several significant advantages over single cycle execution:
 - Unused stages in a particular instruction can be skipped OR
 - instructions can be pipelined (overlapped)
- Will talk about this in CSE 140!

What Hardware Is Needed? (1/2)

- ▶ PC: a register which keeps track of memory addr of the next instruction
- ▶ 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)
 - cache system makes these two stages as fast as the others, on average

What Hardware Is Needed? (2/2)

▶ ALU

- used in Stage 3
- something that performs all necessary functions: arithmetic, logicals, etc.
- we'll design details later

▶ Miscellaneous Registers

- In implementations with only one stage per clock cycle, registers are 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. Not all registers are in the "register file".

Quiz

- A. If the destination reg is the same as the source reg, we **could compute the incorrect value!**
- B. We're going to be able to read 2 registers and write a 3rd in **1 cycle**
- C. Datapath is hard, **Control is easy**

Summary

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