Slide 1

Administrivia

- Homework 5 scores not bad, but few people got full or near-full credit (six out of 34) for that last problem. "Hm!"
- Homework 6 due date moved to Wednesday.

  Probably there will be one more homework, but likely not a long one.
- Quiz 6 scheduled for next Monday. Topics TBA next time.
- Keep in mind that the second exam is scheduled for in class April 24; it will focus on material we've covered since the first exam.

Slide 2

Minute Essay From Last Lecture

- (Later — I haven't looked carefully at responses yet but have the sense that it wasn't a good question to ask this group.)

Slide 3

Homework 6 Help — Tracing Operation of the Processor Circuit

- For the first problem, my intent was that you would trace through what the circuit in Figure 4.17 is actually doing rather than what you think it's supposed to be doing.
- So, you start with what you know — current saved value of the PC and what's at that address (in instruction memory) and contents of selected registers and data memory locations — and work from there. Taking the first few steps . . .
- Right away you can write down output of PC and input/output of instruction memory. For output of instruction memory, it's probably more helpful to write down the various fields of the instruction rather than the hex value of the whole instruction. You can do this the same way you did earlier (translating MIPS assembler language to machine language).
- (Continued on next slide.)

Slide 4

Tracing Operation of the Processor Circuit, Continued

- Now you can write down all the control signals, the inputs and output of the top left adder, and the register-number inputs to the register file. You can get the control signals from the table in Figure 4.18.
- Once you have those, you can write down outputs of the register file and start figuring out what the main ALU is doing. You can also determine whether the top right adder and the data memory will be used (based on control signals).
- Figuring out what the ALU does . . . You need to determine what operation it's doing (based on the `ALUop` control signal and the instruction function field, as shown in Figure 4.13). You also need to determine what the second operand is (contents of a register? sign-extended value from instruction?), again using control signals.
- "And so forth" . . .

Memory Hierarchy — Overview

Slide 5

- Significant overlap between Chapter 5 and material covered in operating-systems course (as I teach it anyway). In previous years pretty much all students went on to that course. Now possibly not. Either way, not a bad idea to discuss briefly now.
- A key idea (borrowed from one writer of O/S textbooks): In a perfect world, we could have as much memory as we wanted, and it would be very fast and very cheap. In the real world, there are tradeoffs (e.g., fast versus cheap, fast versus large).

"Principle of Locality"

Slide 6

- Basic underlying idea — most applications exhibit locality with regard to memory.
- "Temporal locality" — memory locations referenced in the near past are likely to be referenced again in the near future. (At any given time, a program isn't going to be working with *all* of its data.)
- "Spatial locality" — memory locations close together in space likely to be referenced close together in time. (Examples include processing arrays sequentially, accessing local variables, which are apt to be located together in memory).

Slide 7

Memory Hierachy and Caching

- To exploit temporal locality, can use “caching” — keep copies of frequently-used data in faster but smaller memory. Can do this on multiple levels.
- To exploit spatial locality, can move data between levels in blocks.
- *Notice* that while impact of caching on performance can be significant, it should not affect results (which is why it makes some sense to just ignore it initially).

Slide 8

Caches (Between Processor and RAM) — Executive-Level Summary

- Idea here is to interpose a “cache” (small but fast) between the processor and the memory, and use it to hold frequently-referenced data, and have this managed mostly by the hardware.
- Read “from memory” tries cache first, and then if not found there goes to RAM and updates cache.
- Write “to memory” is maybe more interesting — writes to cache but then must at some point write to RAM also — maybe right away (easier to get right but can be slow) or later.

Virtual Memory — Executive-Level Summary

Slide 9

- Basic idea here is to fake having more RAM than you really have, by keeping some data that would be in RAM on disk. In a sense, RAM is a cache for the "real" memory, on disk(!).
- It turns out that a common way to do this also facilitates protecting one process's data from other processes.

Cache Coherence — Executive-Level Summary

Slide 10

- Clearly(?) possible for cached data to be out of synch with data in memory. Probably of most concern if multiple processing elements, each with its own cache, share memory.
- Various schemes exist for ensuring that programs don't have to be aware of this complication. Details in the textbook.

  Something

Slide 11

Caches and Applications Programming

- Mostly the memory hierarchy (including virtual memory) is managed transparently by a combination of hardware and (operating-system) software, so the first approximation presented in introductory courses (memory is essentially a really big array of bytes, with addresses as indices) is okay, especially if you just want right answers.
- However, effects on performance can be significant, so if you want right answers fast . . .

  For single-threaded programs, key idea is to maximize locality (temporal and spatial). Rearranging order in which data is accessed can have a big effect. (Matrix-multiplication example.)

  For multi-threaded programs, also need to consider whether multiple threads need to share access to the same data (problem for correctness too!) or even nearby data ("false sharing" — no effect on correctness but can be slow).

Slide 12

Minute Essay

- None — quiz.