Slide 1

Administrivia

- Reminder: Homework 7 due today.

  A request: Please be sure your name appears on every page of the assignment.
- Homework 8 posted; due next week. (Due date is Wednesday, but I will accept without penalty through Friday.)
- Quiz 6 Monday. Likely topic is pipelining, high-level conceptual questions only. (So it may be enough to review lecture notes, though at some point you should at least skim textbook!)
- About Exam 1, I'm making progress on grading, but it has just been slow! As soon as I finish I plan to do another round of grade summaries. I'm aware that some of you are likely worrying . . .

Slide 2

More Administrivia

- About Exam 2:

  Not a final in the sense of being comprehensive, and same weight as Exam 1.

  As with Exam 1, my plan is for a two-hour timed exam, but this time with more flexibility about timing. I'm thinking we pick two two-hour time slots (with another Doodle poll?) during which I'll be available for questions, or you choose your own time if that's better.

  Short review sheet like the one for Exam 1 to be posted shortly; my plan is to also use part of next Wednesday's class for review.
- I will also likely post a set of extra-credit problems you can use to possibly help your grade. They'd be due near the end of the finals period.
- Just for fun(?): Today is Earth Day! 50th anniversary of the first one . . .

Slide 3

Memory Hierarchy — Recap/Review/Revisited

- In a perfect world, would be a way to store bits that's very fast and can be had in almost arbitrarily large amounts for a reasonable cost. In this world: "Good, fast, cheap: Pick any two."
- Textbook talks about four basic technologies for storing (lots of) bits:
  - SRAM: Pretty fast, but costly, so not feasible on a large scale.
  - DRAM: Significantly less expensive but also significantly slower.
  - "Flash memory": Slower but cheaper still, but does have the problem of "wearing out".
  - Magnetic disks: Cheap enough to be about as big as is needed for most general-purpose computing, but far, far slower.

Slide 4

Memory Hierarchy — Recap/Review/Revisited

- So where does "hierarchy" come in? Well . . .
- Programs' use of memory mainly exhibits "locality" (in both time and space).
- So, common to design systems in terms of hierarchy, with each level larger but slower than one above it. Idea is then to store (a copy of) most-frequently-used data in upper levels, hierarchy, where it's fast to get at, and access lower levels less frequently.

Slide 5

Memory Hierarchy — Recap/Review/Revisited, Continued

- Idea is that data moves up and down in this hierarchy as needed, all in a way that's invisible to application programs, *except* for effects on performance.
- In order for this to work, each level (hardware or virtual memory) must have space for some data from the next level down, plus some way of (correctly!) reading from / writing to next level down, which means having some way to map from lower-level addresses to elements.

Slide 6

Memory Hierarchy — Levels, Fastest to Slowest

- Registers — use is explicit at machine-language level, but managed by compiler for HLLs. (Compilers topic — "register spill".)
- Processor cache(s) — managed by hardware, possibly with some help from operating system.
- Main memory (RAM) — use is like that of registers.
- Virtual memory (on disk) — typically managed by operating system, with some help from hardware. ("Typically" because in earlier times sometimes programmers made use of the idea in their own code.)

Slide 7

Processor Caches — A Bit More Detail

- Processor caches can be multi-level. Sometimes in multicore systems each core has its own, and then there's one shared among cores.

  (Aside: On our machines we have "hardware locality" package installed. `module load hwloc-latest` and then `lstopo` will show you some details about hardware.)

- In order for this to work, each cache must have space for some data from the next level down, plus some way of (correctly!) reading from / writing to next level down, which means having some way to map from lower-level addresses to elements.

Slide 8

Processor Caches — A Bit More Detail, Continued

- Idea is that for reads, processor just reads using address as we've discussed, and either:
  - Data is found in the cache — "cache hit" — and given back to processor.
  - Data is not found — "cache miss" — and hardware/software does whatever is necessary to get it there and then continues as for hit.

  Obviously(?) the fewer caches misses the better.

- (Same idea applies to virtual memory, except that "not found" is called a page fault and handled by the O/S. Topic for a course in operating systems!)

Slide 9

Caching — A Bit More Detail, Continued

- But wait: If cache is smaller than what it's caching, how can this work? Each cache element could potentially contain one of many pieces of data? So include in cache element a "tag" that says which one it contains, plus a "valid" bit.
- For writes, things a bit more complicated: Similar idea applies, but must decide whether to write to lower levels immediately or wait. Writing immediately easier but slower, probably enough so that it's worth the trouble to do something more complicated. More details in textbook.
- Overall, textbook (section 5.8) presents four questions that pretty much sum it up; adding one more . . .

Slide 10

Caching — Size of Elements

- Processor caches *can* store single words, but often store larger units (2 words, or 4, or . . .) — "cache lines". Idea is to exploit spatial locality.
- Virtual memory typically uses much bigger units (often "pages" of 2K or 4K). Again, spatial locality.

Slide 11

Caching — Mapping Addresses to Cache Elements

- “Direct map” cache is simple: Each memory address maps to exactly one cache element.
- “Fully associative” cache is opposite extreme: Any memory address can map to any cache element.
- “Set associative” cache is in between: Each memory element maps to a set of entries. Reasonable compromise between extremes?

Slide 12

Caching — Looking Up Data

- For “direct map” cache, simple: Only one cache element to check, so just compare tags. So, fast but not very flexible.
- For “fully associative” cache, more complicated: Potentially have to search whole cache for matching address. Very flexible but costly to implement with good performance.
- For “set associative” cache, in between: Still have to check multiple elements, but fewer of them. Reasonable compromise between extremes?
- Which is used? for virtual memory, likely fully-associative; for processor caches, one of the others.

Slide 13

Caching — Replacing Cache Elements

- On “cache miss”, if appropriate cache elements are all in use, must pick one to replace. For direct mapping, trivial (only one choice); for the other two not so trivial.
- How to choose? Goal should be to replace something that won’t be needed again soon. Often approaches based on temporal locality (if not used recently maybe won’t be used again soon).
- For processor caches, hardware problem; various solutions exist.
- For virtual memory, software (O/S) problem; again various solutions exist (“page replacement algorithms”).

Slide 14

Caching — How to Manage Writes

- One complication: If cache elements are more than one word, need to read old element, then change word being written.
- And then: Write back immediately (“write-through”), or wait (write buffer or “write-back”)? Former is easier but could be quite slow; latter is more complicated but probably needed for acceptable performance.

Slide 15

Minute Essay

- Any preliminary thoughts about when we should do Exam 2? early date, late date, time of day?
- I'm hearing that some students feel like they're not learning current material because of the remote-learning setup. *If this is you*, any thoughts on how I could help? I feel like office-hours help may be indicated, but I don't often seem to get "customers" during my admittedly few scheduled times. ?
- Quiz 5.

  I want from you (by e-mail or in your "TurnIn" folder) a PDF if possible. Just a document with your answers — and the pledge and times — is fine. *Important:* Please be sure your name appears on every page. I will print to grade, and otherwise . . .