Slide 1

Administrivia

- Reminder: Homework 7 due Monday.

Slide 2

A Little More About Pipelining

- Recall preliminary sketch of pipelined MIPS implementation (figure 6.12).
- Many additions to make things actually work:
  - As mentioned last time — “write register” number needs special handling.
  - Control is more complicated, and needs to be passed down the pipeline.
  - Need to deal with data hazards: Resolve some by adding “forwarding” hardware to (when appropriate) get register values from values previously computed but not yet stored. Resolve others by “stalling” (inserting no-op instructions).
  - Need to deal with branch hazards: Assume no branch and then “flush” pipeline if branch is taken.
  - Need to “flush” pipeline on exceptions.
- Final version of pipelined implementation in figure 6.65.

Slide 3

Memory Hierarchies — Overview

- Ideally, a computer would have lots and lots of memory that can be accessed as fast as the processor works. Alas, gigabytes of “fast” memory would be too expensive.
- Workaround is to define “memory hierarchy”:
  - Processor / registers.
  - “Cache” — small but fast.
  - Main memory — large but relatively slow.
  - Disk — larger but even slower.
- Idea is that data moves between adjacent levels, and upper levels hold frequently-used data. Works because most programs’ data access exhibits “locality of reference” — with regard to time (“temporal locality”) and to memory location (“spatial locality”).

Slide 4

Cache Basics — Reading From Memory

- Idea is that when we want to read data from memory (e.g., `lw`), we first look in cache. How do we know what’s there? One way is “direct-mapped” cache, as in figure 7.7:
  - Last part of memory address tells where in cache to look.
  - Each “entry” in cache contains valid/invalid bit, “tag” with rest of memory address, data.
- If data is in cache, “hit” (good). If not, “miss”, and must read from memory, stalling processor meanwhile.

Slide 5

Cache Basics — Writing to Memory

- What happens when we want to write data to memory (e.g., `sw`)? faster to just write to cache, but at some point must also write to memory, or data will be lost.
- One approach — "write-through cache" (always write to memory too). Pluses/minuses?
- Another approach — "write buffer" (write to memory too, but buffer the data so we don't have to wait).
- And one more — "write back" (write only when data item is replaced in cache).

Slide 6

Cache Improvements

- Simple direct-mapped cache will help, but maybe we can do (even) better?
- One idea: Rather than caching individual words, cache larger blocks (several words). Spatial locality of reference means this will likely be a good idea — fetch several words at a time from memory.
- Another idea: Replace simple direct-mapped scheme (each memory element has only one possible cache location) with something more flexible. In "fully associative" cache, a memory element can be in any element of the cache. Must keep track, for each spot in the cache, what it contains. Must also decide where to put new data — usually replace least recently used data. "Set associative" combines elements of direct-mapped and fully associative schemes.

Slide 7

Virtual Memory

- Can think of virtual memory as applying caching idea to lower levels of the memory hierarchy (main memory and disk).
- Basic idea is to keep data we need right now in memory, rest on disk (figure 7.20). Allows us to pretend we have more memory than we really have. Rather than “caching” individual memory elements, work with bigger chunks (“pages”, usually) and do more of the management in software (operating system).
- Distinguish between virtual addresses and real addresses. Mapping from one to the other is via hardware (“memory management unit” (MMU)) and page tables (figure 7.23). To make this practical, keep some of page table in — another cache! “translation lookaside buffer” (TLB).

Slide 8

Virtual Memory, Continued

- Idea can be extended to give each process a separate “address space” and provide hardware support for protecting one process’s address space from other processes.

Slide 9

Caching — Summary

- Key idea — keep data we need soon/often someplace where we can get to it quickly, have a larger storage area for all data.
- Many uses:
  - Cache between processor and main memory.
  - Virtual memory (and TLB).
  - “Buffer cache” of data read from disk.
  - Caching for Web pages.

Slide 10

Minute Essay

- Any questions about pipelining you’d like me to try to answer next week?
- Any questions about memory hierarchies/caching you’d like me to try to answer next week?