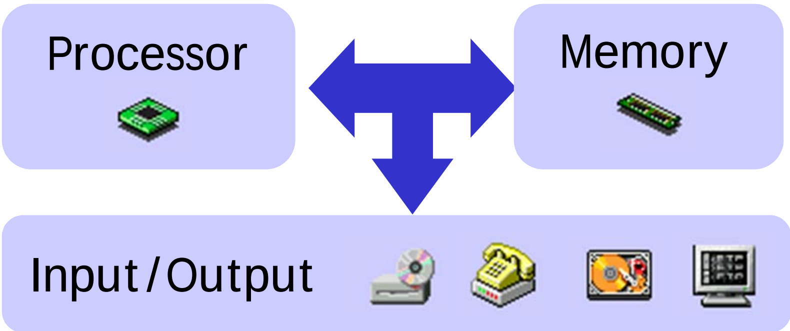
Final Lecture ☹

- A few minutes to wrap up and add some perspective

Instant replay

- The quarter was split into roughly three parts and a coda.
  - The 1st part covered instruction set architectures—the connection between software and hardware.
  - In the 2nd part of the course we discussed processor design. We focused on pipelining, which is one of the most important ways of improving processor performance.
  - The 3rd part focused on large and fast memory systems (via caching), virtual memory, and I/O.
  - Finally, we briefly discussed performance tuning, including profiling and exploiting data parallelism via SIMD and Multi-Core processors.
- We also introduced many performance metrics to estimate the actual benefits of all of these fancy designs.



Some recurring themes

- There were several recurring themes throughout the quarter.
  - Instruction set and processor designs are intimately related.
  - Parallel processing can often make systems faster.
  - Performance and Amdahl's Law quantifies performance limitations.
  - Hierarchical designs combine different parts of a system.
  - Hardware and software depend on each other.

Instruction sets and processor designs

- The MIPS instruction set was designed for pipelining.
  - All instructions are the same length, to make instruction fetch and jump and branch address calculations simpler.
  - Opcode and operand fields appear in the same place in each of the three instruction formats, making instruction decoding easier.
  - Only relatively simple arithmetic and data transfer instructions are supported.
- These decisions have multiple advantages.
  - They lead to shorter pipeline stages and higher clock rates.
  - They result in simpler hardware, leaving room for other performance enhancements like forwarding, branch prediction, and on-die caches.

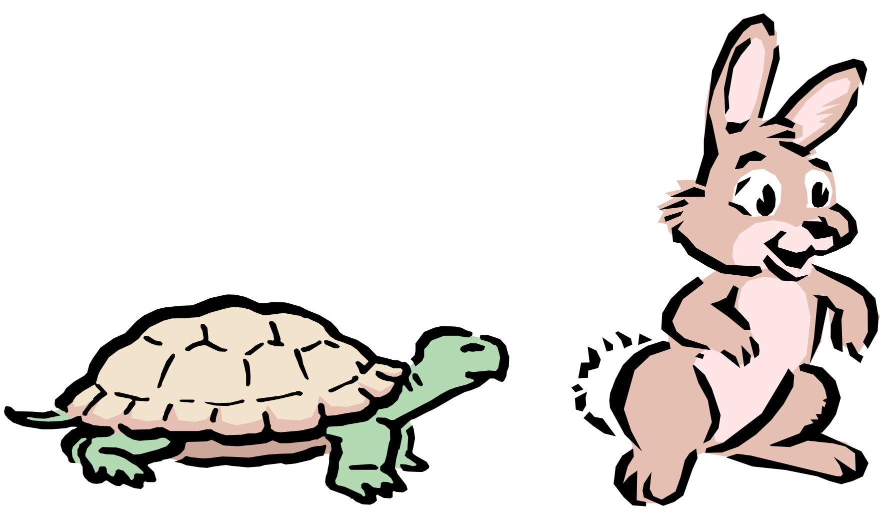
Parallel processing

- One way to improve performance is to do more processing at once.
- There were several examples of this in our CPU designs.
  - Multiple functional units can be included in a datapath to let single instructions execute faster. For example, we can calculate a branch target while reading the register file.
  - Pipelining allows us to overlap the executions of several instructions.
  - SIMD performance operations on multiple data items simultaneously.
  - Multi-core processors enable thread-level parallel processing.
- Memory and I/O systems also provide many good examples.
  - A wider bus can transfer more data per clock cycle.
  - Memory can be split into banks that are accessed simultaneously. Similar ideas may be applied to hard disks, as with RAID systems.
  - A direct memory access (DMA) controller performs I/O operations while the CPU does compute-intensive tasks instead.

Performance and Amdahl's Law

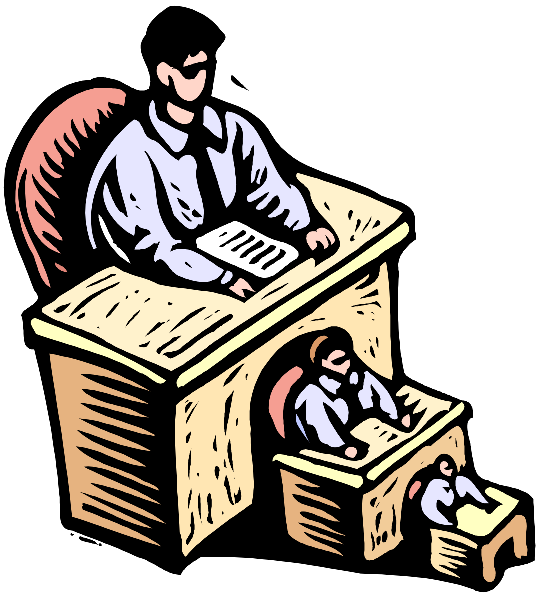
- First Law of Performance: **Make the common case fast!**

- But, performance is limited by the slowest component of the system.
- We've seen this in regard to cycle times in our CPU implementations.
  - Single-cycle clock times are limited by the slowest instruction.
  - Pipelined cycle times depend on the slowest individual stage.
- Amdahl's Law also holds true outside the processor itself.
  - Slow memory or bad cache designs can hamper overall performance.
  - I/O bound workloads depend on the I/O system's performance.



Hierarchical designs

- Hierarchies separate fast and slow parts of a system, and minimize the interference between them.
  - Caches are fast memories which speed up access to frequently-used data and reduce traffic to slower main memory. (Registers are even faster…)
  - Buses can also be split into several levels, allowing higher-bandwidth devices like the CPU, memory and video card to communicate without affecting or being affected by slower peripherals.



Architecture and Software

- Computer architecture plays a vital role in many areas of software.
- Compilers are critical to achieving good performance.
  - They must take full advantage of a CPU's instruction set.
  - Optimizations can reduce stalls and flushes, or arrange code and data accesses for optimal use of system caches.
- Operating systems interact closely with hardware.
  - They should take advantage of CPU features like support for virtual memory and I/O capabilities for device drivers.
  - The OS handles exceptions and interrupts together with the CPU.

Five things that I hope you will remember

- Abstraction: the separation of interface from implementation.
  - ISA's specify what the processor does, not how it does it.
- Locality:
  - Temporal Locality: "if you used it, you'll use it again"
  - Spatial Locality: "if you used it, you'll use something near it"
- Caching: buffering a subset of something nearby, for quicker access
  - Typically used to exploit locality.
- Indirection: adding a flexible mapping from names to things
  - Virtual memory's page table maps virtual to physical address.
- Throughput vs. Latency: (# things/time) vs. (time to do one thing)
  - Improving one does not necessitate improving the other.