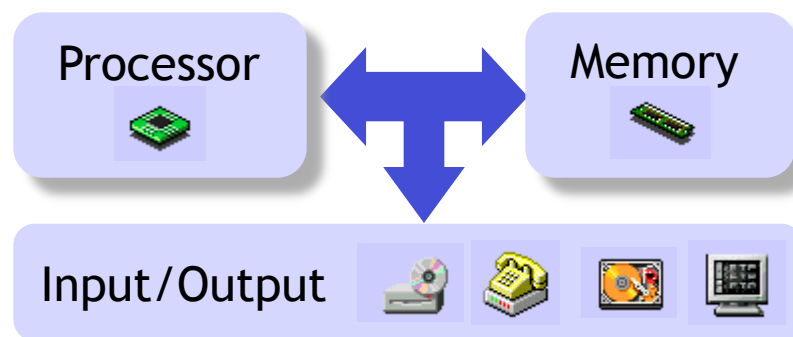


Instant replay

- The semester was split into roughly four parts.
 - The 1st quarter covered **how a (very simple) computer works**—from gates to the fetch-execute model.
 - The 2nd quarter covered **instruction set architectures**—the interface between software and hardware and how high-level language code actually runs on processors.
 - In the 3rd quarter of the course we discussed processor performance. We focused on **pipelining**, for improving computation speed, and **caching** for providing quick access to data.
 - Finally, we discussed **hardware support for parallel execution** in the form of SIMD, cache coherence, and atomic operations.



Some recurring themes



- There were several recurring themes throughout the semester.
 - 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

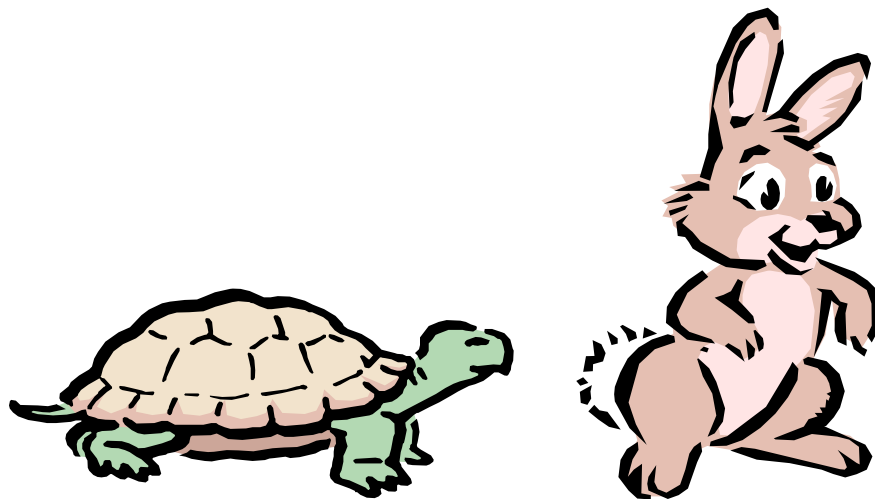
- The MIPS instruction set was designed for pipelining.
 - All instructions are the same length, to make instruction fetch and jump and branch address calculations simpler.
 - Opcode and operand fields appear in the same place in each of the three instruction formats, making instruction decoding easier.
 - 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 performs 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.



Hardware/Software Co-design

- Hardware and software are good at different things
- Many problems are **best** solved by cooperation between HW and SW
- Good **compilers** are critical for performance.
 - Optimizations can reduce stalls and flushes, or arrange code and data accesses for good use of system caches.
- Hardware provides primitives for use by **operating systems**.
 - TLB and user/supervisor mode for implementing virtual memory
 - Timer interrupts to enable O/S scheduling
 - Detection of exceptions, which are handled by the OS.
- Multiprocessor synchronization/communication
 - Primitives for atomic operations, enable building software locks
 - Multiprocessor cache coherence enable inter-thread communication

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.

Where to go from here?

- **CS433: Advanced Comp. Arch:** All of the techniques used in modern processors that I didn't talk about (out-of-order execution, superscalar, advanced branch prediction, prefetching...). Homework-oriented + project.
- **CS431: Embedded Systems:** How hardware/software gets used in things we don't think of as computers (e.g., anti-lock breaks, pacemakers, GPS). Lab-oriented.
- **CS484: Parallel Programming:** How to write parallel programs.
- **CS498-MG: Program Optimization:** How to make a program run really fast (like 4th quarter of 233, but more so). Project-oriented.
- **ECE 498 AL : Programming Massively Parallel Processors:** How to write general purpose programs to run on GPU using CUDA.
- **ECE411: Computer Organization and Design:** Some content overlap with CS232 and CS433, but you actually build the hardware. Lab-oriented.
- **CS426: Compiler Construction:** How does a compiler translate a programming language down to assembly and optimization. Project-oriented.

Good luck on your exams and have a great break!

