Overview

• Computer architecture is an exciting field
  - Computer architects are always on the cutting edge
  - Designing several future generations of processors now

• Exciting time to be in computer architecture!
  - Paradigm shift from single-core to multi-core
  - But this class focuses on single-core
  - Multi-core architecture is just a collection of single cores, so must know single-core architecture first.

• Computer architects have a different design philosophy as compared to software designers

What is this class about?

• Computer Architecture:
  • Instruction sets: how are microprocessors programmed?
  • Organization: how does data flow in the microprocessor?
  • Hardware design: how are logic components implemented?
What is this class about?

° Computer Architecture:
  • Instruction sets: how are microprocessors programmed?
  • Hardware/software interface: How are instruction sets designed? How does it impact the design of microprocessors and the software running on them?
  • Example: Apple’s move from PowerPC to "x86" (Intel)
    - Enabled greater choice in terms of processor configurations
    - Software migration was a major issue; addressed with “binary translation” software (Rosetta)
What should you expect to achieve in this class?

- In-depth understanding of the inner-workings of modern computers, their evolution, and trade-offs present at the hardware/software boundary.
  - Insight into fast/slow operations that are easy/hard to implement in hardware
  - Tradeoffs between these designs
- Computer architecture design process
- Hands-on experience with the design process in the context of a large, complex hardware system
  - From functional specification to control and datapath implementation and simulation
  - Using modern CAD tools and methodologies (VHDL)

Course structure

- Class syllabus:
  - Also refer to policies document for information on academic honesty and late assignments
- Book to be used as supplement for lectures
  - When a topic is covered in class, not all details will be presented.
  - I expect you to read on your own to learn those details
- Additional reading materials
- Key ingredient to success:
  - Read material *before* lecture
- Grading:
  - Lab assignments – 55%
  - Homework questions from book – 10%
  - Exams (two midterms, second one is not cumulative) – 35%
    - Midterm 1 date tentative, Midterm 2 date fixed

Lab Assignments/Home Homework Questions

- No late assignments/homework will be accepted, no matter what
- Homeworks and labs will essentially alternate
- Demo assignments in lab, turn in report via Sakai
  - Two sections:
    - Setup section: Get started with tools used
    - Lab section: Hands-on design experience
- Homework questions
  - Helps you keep up with material for exams, reinforces concepts
  - You must use the revised 4th edition, the green one
- Dos and Don'ts
  - While studying together in groups is encouraged to foster discussion and learning, all work submitted must be your own
    - Not your neighbors, partners, past years’ students, from the web, etc. not even with citation
  - Plagiarism will result in an F in the course!
Lab Assignments

- Lab assignments are a major component of this class
  - Goal: expose you to the process of designing a microprocessor
  - Labs will upon each other
  - Challenging but rewarding

- Throughout this class you will design a MIPS microprocessor:
  - To the extent that it can be simulated within a VHDL-based hardware development framework
  - Starting with the major components of a MIPS datapath
  - Integrate the components and control logic into a processor implementing a subset of MIPS

- Your tools:
  - VHDL and Altera Quartus II
  - Proficiency with these is key to success

Internet companions

- EEL-4713 Web site - Sakai:
  - Lecture slides
  - Assignments
  - Announcements
  - Software documentation, tutorials
  - Discussion forum
  - Course schedule
  - All course files are linked off of my webpage, Sakai may simply refer you to that directory at times

Next lectures

- Homework #1 is posted, due next week
- First lab assignment will be up soon
- Reading for the next few lectures: chapters 1 and 2
- Computer Abstractions and Technology
  - Textbook, chapter 1
- Instruction set architectures
  - Textbook, Chapter 2
  - Sections 2.1-2.8, 2.10, 2.12-2.13, 2.18-2.20

What is “Computer Architecture”

Computer Architecture =

Instruction Set Architecture (ISA) + Machine Organization

Classic computer organization:

- John von Neumann
- Stored program computer
- Read instruction and data from memory; decode and execute; write results back to memory

Five key components:
- Input, Output, Memory, Datapath and Control
Abstraction layers

- High-level language (e.g. C++, Java)
- Low-level language (Assembly)
- Register-level transfer (Datapath)
- Basic logic gates (AND, OR)
- Devices (CMOS transistors)

Hardware organization

- Tradeoff: support an efficient implementation, while providing a standard interface to software

The big picture

- Registers

Software interface

- High-level language (e.g. C++, Java)
- Low-level language (Assembly)

Instruction set architecture defines the interface between the microprocessor hardware and software

The Pentium™ 4 (~40M transistors)
The big picture (2)

```
addiu  $s2,$s2,1
bne    $s2,$t1,L3
s.d    $f4, 0($t2)
```

Course Overview

**Computer Architecture**
- **Instruction Set**
  - Machine Language
  - Compiler View
  - Software interface
  - e.g. IA-32 vs. IA-64

**Hardware Design**
- Machine Implementation
- Logic Design
  - e.g. 90nm vs. 65nm; low-power vs. fast clock

**Organization**
- Datapath and control
  - e.g. Core Duo vs. Athlon

Level of abstraction

**Execution cycle (control)**

- **Instruction Fetch**
- **Instruction Decode**
- **Operand Fetch**
- **Execute**
- **Result Store**
- **Next Instruction**

- Obtain instruction from program storage
- Determine required actions and instruction size
- Locate and obtain operand data
- Compute result value or status
- Deposit results in storage for later use
- Determine successor instruction

Topics addressed in this course

° How are programs written in a high-level language translated into the hardware language?
° What is the interface between the software and the hardware? What are the design criteria used in defining it?
° What determines the performance of a program? How can a programmer improve performance?
° What is the design process starting from the definition of a microprocessor’s behavior and finishing with a functional implementation?
° What are techniques that a microprocessor designer can employ to improve performance while maintaining software compatibility?
° Focus on the architecture and organization aspects
Five “classic” components of a computer organization

• Fetch, decode, execute, store

Understanding program performance

• Algorithms and data structures
  - Time/space complexity – e.g. naïve/bubble sort \(O(n^2)\) vs. quick sort \(O(n\log n)\) determines number of source-level statements executed
  - Not covered in this class

• Programming language, compiler, architecture
  - Determines number of machine-level instructions for each source-level statement

• Processor and memory system
  - Determines how fast instructions go through a fetch/execute/store cycle

• I/O subsystem (hardware and software)
  - How fast instructions which read from/write to I/O devices are executed

Before and during a program execution

• Before - Applications written in high-level language (e.g. C++) need to be translated to the machine language microprocessors recognize before they execute
  - Compilers

• During - At runtime, applications use services from an operating system to facilitate interaction with the hardware and sharing by multiple entities
  - E.g. Linux, Mac OS, Windows
  - Basic I/O operations on files, network sockets, ...
  - Memory allocation
  - Scheduling of CPU cycles across multiple processes

Application classes and characteristics

<table>
<thead>
<tr>
<th></th>
<th>Price of system</th>
<th>Price of microprocessor module</th>
<th>Critical system design issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop</td>
<td>$500-$5,000</td>
<td>$50-$500</td>
<td>• Tradeoff price/performance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• High graphics performance</td>
</tr>
<tr>
<td>Server</td>
<td>$5,000-$5,000,000</td>
<td>$200-$10,000</td>
<td>• High throughput</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• High availability/dependability</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• High scalability</td>
</tr>
<tr>
<td>Embedded</td>
<td>Free-$100,000</td>
<td>$0.01-$100</td>
<td>• Low price</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Low power consumption</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Application-specific performance</td>
</tr>
</tbody>
</table>
Course Overview

**Computer Architecture**
- Instruction Set
  - Machine Language
  - Compiler View
  - Software interface
- Organization
  - Datapath and control
- Core Duo vs. Athlon

**Hardware Design**
- Machine Implementation
- Logic Design
- 90nm vs. 65nm; low-power vs. fast clock

Instruction Set Architecture

... the attributes of a [computing] system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls of the logic design, and the physical implementation.

Amdahl, Blaaw, and Brooks, 1964

-- Organization of programmable storage
-- Data types & data structures: encodings & representations
-- Instruction formats
-- Instruction (or operation code) set
-- Modes of addressing and accessing data items and instructions
-- Exceptional conditions
### Levels of Representation

- **High Level Language Program**
  - Compiler
  - Assembly Language Program
    - Assembler
    - Machine Language Program
    - Machine Interpretation

#### ISA
- temp = v[k];
- v[k] = v[k+1];
- v[k+1] = temp;

#### Control Signal Spec
- assert address 0($2) on bus
- assert memory read signal
- select register $15; latch

---

### Example Desktop/server Instruction Set Architectures

#### Same ISA
- Digital Alpha (v1, v3)
- HP PA-RISC (v1.1, v2.0)
- Sun Sparc (v8, v9)
- SGI MIPS (MIPS I, II, III, IV, V)
- "x86" (IA-32) (Intel 8086, 80286, 80386, 80486, Pentium, MMX, AMD Athlon, ...)
- HP/Intel EPIC/IA-64 (Itanium)

#### Different Hardware Implementations

---

### Microprocessor sales by ISA

32- and 64-bit
- ARM: 80% sales for cell phones
- Other: application-specific or customized architectures

---

### Example Instruction Set Architecture (ISA): MIPS R3000

#### Instruction Categories
- Load/Store
- Integer computation
- Jump and Branch
- Floating Point
- Memory Management
- System

#### Instruction Format
- OP | rs | rt | rd | shamt | funct
- OP | rs | rt | immediate
- OP | target

### Special range designations

---

EEL-4713C – Ann Gordon-Ross
Course Overview

Computer Architecture

Instruction Set
- Machine Language
- Compiler View
- Software interface

IA-32 vs. IA-64

Organization
- Datapath and control

Core Duo vs. Athlon

Hardware Design
- Machine Implementation
- Logic Design

90nm vs. 65nm; low-power vs. fast clock

Organization

Logic Designer's View
-- capabilities & performance characteristics of principal functional units
  (e.g., registers, ALU, shifters, etc.)
-- ways in which these components are interconnected
-- nature of information flows between components
-- logic and means by which such information flow is controlled.

Choreography of units to realize the ISA

Register Transfer Level description

Example: Pentium III die

Example: Pentium III pipeline overview
**Course Overview**

- Computer Architecture
  - Instruction Set
    - Machine Language
    - Compiler View
    - Software interface
  - Organization
    - Datapath and control
    - Core Duo vs. Athlon
  - Hardware Design
    - Machine Implementation
    - Logic Design
    - 90nm vs. 65nm; low-power vs. fast clock

**Hardware design and implementation**

- Impact performance, cost, and power consumption of architectures

- So far we have enjoyed exponential improvements over time in:
  - Microprocessor performance
  - Main memory capacity
  - Secondary storage capacity

- “Moore’s Law”
  - Not an actual physical law; observation of a technology trend
  - Microprocessor capacity doubles roughly every 18-24 months

**Technology => dramatic change**

- Processor
  - Logic capacity: about 30% per year
  - Clock rate: about 20% per year

- Memory
  - DRAM capacity: about 60% per year (4x every 3 years)
  - Memory speed: about 10% per year
  - Cost per bit: reduced by about 25% per year

- Disk
  - Capacity: about 60% per year

**DRAM capacity**

![DRAM Capacity Graph](image)
**Microprocessor performance**

- Improvements also exponential
- Key technology driver: device scaling
- As transistors get smaller (e.g. 180nm to 90nm to 65nm feature sizes)
  - They tend to also get faster and consume less power
    - Faster clock rates
  - More transistors can be packed in the same area
    - Superscalar pipelines; multiple cores; larger caches
- Problems faced by scaling at current (nanoscale) technologies:
  - Fast transistors, but slow interconnect
  - Transient errors
  - Low power per device, but billions of them packed together

---

**The power wall**

- Dynamic power = capacitive load × Voltage² × Frequency
  - Load: function of transistor, wire technologies, fan-in/out
  - As frequency increases, voltage had to be dropped to maintain power at check => 5V down to 1V
  - At very low voltages, leakage and static power consumption become problems, approximately 40%
  - A “wall” blocking frequency scaling

---

**Uniprocessor Performance**

Constrained by power, instruction-level parallelism, memory latency

---

**From unprocessors to multiprocessors**

- Clock frequency scaling limited
- Can get better performance by exploiting parallelism – multiple operations per cycle
- Instruction-level (superscalars): diminishing returns circa 2004
- Process/thread-level parallelism: multi-core processors
Multiprocessors

- **Multicore microprocessors**
  - More than one processor per chip

- **Requires explicitly parallel programming**
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

Next lectures

- Sign up for the Google group, check for assignment #1
- Reading for the next few lectures: chapters 1 and 2
- **Computer Abstractions and Technology**
  - Textbook, chapter 1
- **Instruction set architectures**
  - Textbook, Chapter 2
  - Sections 2.1-2.8, 2.10, 2.12-2.13, 2.18-2.20