Language of the Machine

I speak Spanish to God,
Italian to women,
French to men,
and German to my Horse.

Charles V, King of France
1337-1380

Instructor: Francis G. Wolff
wolff@eecs.cwru.edu
Case Western Reserve University
This presentation uses powerpoint animation: please viewshow
Computer Architecture Trends: Post PC era

- 100 million processors were sold for desktop computers
- 3 BILLION processors were sold for embedded systems
Consumer Markets: Processor trends

**Embedded Systems Market Worth** $66 Billion by 2004

It’s expected that the average car will be Internet ready and have over $2000 worth of embedded computers.

**Internet Appliances** will grow by more than 1500% between now and the end of 2004. That translates into some 37 million devices shipping in 2004.

This market includes TV-based Internet access devices, web phones, and other terminals that use wires to connect to the web.
Bionics:

Sensors in latex fingers instantly register hot and cold, and an electronic interface in his artificial limb stimulates the nerve endings in his upper arm, which then pass the information to his brain.

The $3,000 system allows his hand to feel pressure and weight, so for the first time since losing his arms in a 1986 accident, he can pick up a can of soda without crushing it or having it slip through his fingers. *One Digital Day*
Future PC Design: System-on-a-Chip

12 million logic gates can now be placed on a single chip.

Computer designers must be experienced:
- in both hardware and software co-design,
- as well as in embedded applications,
- be familiar with optimization techniques to perform the specific program using the least size, power, and time.

How do we design such large systems....
Design Abstractions

- Coordination of many *levels of abstraction*
Design Abstractions

An abstraction omits unneeded detail, helps us cope with complexity.
Instruction Set Architecture

A very important abstraction: Instruction Set Architecture

- interface between hardware and low-level software
- standardizes instructions, machine language bit patterns, ...
- advantage: different implementations of the same architecture
- disadvantage: sometimes prevents using new innovations

Modern instruction set architectures:
80x86/Pentium/K6, PowerPC, DEC Alpha, MIPS, SPARC, HP

True or False: Binary compatibility is important?

Yes (Microsoft/Intel alliance)   No - (Unix, Linux, C++, Java)
Yes - Sales, Marketing   No - Speed, Engineers, Programmers
Example: Digital Pager Architecture

Why not just use an Intel Pentium instead?

Two completely differently optimized Instruction Set Architectures

Cost, size, power, speed, weight, ...

CWRU EECS 322
Course Overview

Hundreds of years later, and IEEE/ABET accreditation, this course has evolved as follows:

I speak
C++ to my Compiler,
Machine Instructions to Assemblers,
Datapath Design to Digital Logic Gates,
and German to my Horse.
Smart cards differ from credit cards in using onboard memory chips and microprocessors or micro-controllers instead of magnetic strips.

- Each chip can hold 100 times the information contained on a standard magnetic-stripe card.

- Smart cards make personal and business data available only to the appropriate users.

There are currently 2.8 billion smart cards in use:

- 575 million phone,
- 36 million financial,
- 30 million ID cards,
- 17 million pay TV, …
Smart cards have embedded within them a processor and often a cryptographically enhanced co-processor.

Today's smart card hardware controller typically includes an 8-bit CPU (such as the Motorola 68HC05), 780 bytes of RAM, 20 KB of ROM, 16 KB of EEPROM on a single die, and (optionally) an on-chip hardware encryption module.
An example of the software handshaking protocol is shown below.
Textbook:

Computer Organization and Design
“*The Hardware/Software Interface*”
2nd edition
John L. Hennessy & Patterson
Morgan Kaufmann Publishers
ISBN = 1-55860-428-6
http://www.mkp.com

Homeworks, exams, lecture material are heavily based on the textbook! Avoiding it will be hard.
Instructor: Frank Wolff
Office: Olin Building Room 514
Phone: 1-216-368-5038

Preferred form of communication
email: wolff@eecs.cwru.edu

Office hours: generally before & after class

Course Website:
http://bear.ces.cwru.edu/eecs_322
http://129.22.150.65/eecs_322
Course Graders / Teaching Assistants

Priority: Graders/TAs then Instructor

Primary Grader: Ramakrishnan Vijayakumar
Office: Olin 413, Embedded Systems Lab
Phone: TBA

Preferred form of communication
email: rxv20@po.cwru.edu

Office hours: TBA
Course Grading

Exams = Projects = 25% each
Total: 4 exams and 1 programming project
Homeworks assigned for next class day

Tentative Exam dates:
(disclaimer: subject to change in time/topics)
1 week advanced confirmation notice

February 12: Chapters 3, 2, 1
March 7: Chapter 4
April 4: Chapter 5-6
April 30: Chapter 6-7-8
Course Schedule

Class: Monday & Wednesday 4:30-5:45pm

First Class Day: January 17

Spring Break: March 12 - 16

Last Class Day: April 30 (Last Exam)

Get Unix & NT accounts
Course Outline

1. Introduction
2. Instruction Set Design
3. Computer System Design
4. Data Path Design
5. Instruction Sequencing and Control
6. Pipeline Design
7. Memory Systems
8. Input - Output and Communications
Course Outline Concepts (1-3)


Course Outline Concepts (4-6)

4. **Data Path Design**: Basic processor datapath design. Design of Arithmetic Logic Unit (ALU). Design of Fast ALUs. Multipliers and Dividers. Floating Point Units.


Course Outline Concepts (7-8)


C Operators/Operands

- **Arithmetic operators**: +, -, *, /, % (mod)
- **Assignment statements**: Variable = expression;
  celsius = 5 * (fahr - 32) / 9;
- **Operands**:
  - Variables: lower, upper, fahr, celsius
  - Constants: e.g., 0, 1000, -17, 15
- In C (and most High Level Languages) variables declared 1st and given a data type
  - Example:
    ```c
    int celsius; /* declare celsius as an integer */
    int a, b, c, d, e;
    ```

Note: we begin at chapter 3 of the text book
Assembly Operators

- **Syntax of Assembly Operator**
  1) operation by name
  2) operand getting result
  3) 1st operand for operation
  4) 2nd operand for operation

- **Example**
  add b to c and put the result in a: \texttt{add a, b, c}
  — Called an (assembly language) **Instruction**

- **Equivalent assignment statement in C:**
  \texttt{a = b + c;}

Assembly Operators/Instructions

• **MIPS Assembly Syntax is rigid:**
  1 operation, 3 variables
  Why? Keep Hardware simple via regularity
  Note: Unlike C each line of assembly contains at most 1 instruction

• **How do following C statement?**
  \[
a = b + c + d - e;
  \]

• **Break into multiple instructions**
  
  \[
  \begin{align*}
  \text{add a, b, c} & \quad \# a = \text{sum of } b \& c \\
  \text{add a, a, d} & \quad \# a = \text{sum of } b,c,d \\
  \text{sub a, a, e} & \quad \# a = b+c+d-e
  \end{align*}
  \]

• # is a comment terminated by end of the line
• /* comment */ is a C comments & can span many lines
Compilation

• Example: compile by hand this C code:

\[ f = (g + h) - (i + j); \]

• First sum of \( g \) and \( h \). Where put result?

\[ \text{add } f,g,h \quad \# \ f \text{ contains } g+h \]

• Now sum of \( i \) and \( j \). Where put result?

– Cannot use \( f \)!

– Compiler creates temporary variable to hold sum: \( t1 \)

\[ \text{add } t1,i,j \quad \# \ t1 \text{ contains } i+j \]

• Finally produce difference

\[ \text{sub } f,f,t1 \quad \# \ f=(g+h)-(i+j) \]
Compilation Summary

- C statement (5 operands, 3 operators):
  
  \[ f = (g + h) - (i + j); \]

- Becomes 3 assembly instructions
  (6 unique operands, 3 operators):

  - add f, g, h  \# f contains g+h
  - add t1, i, j  \# t1 contains i+j
  - sub f, f, t1  \# f=(g+h)-(i+j)

- Big Idea: compiler translates notation from 1 level of abstraction to lower level

- In general, each line of C produces many assembly instructions

  – One reason why people program in C vs. Assembly; fewer lines of code

  – Other reasons? Portability, Optimization
Registers

- Unlike C++, assembly instructions cannot use variables
  - Why not? Keep Hardware Simple

- Instruction operands are **registers**: limited number of special locations; 32 registers in MIPS (r0 - r31)
  - Why 32? Smaller is faster

- Each MIPS register is 32 bits wide
  - Groups of 32 bits called a **word** in MIPS
Assembly Operands: Registers

- Naming of 32 registers:
  instead of $r0, $r1, ..., $r31, use

  $s0, $s1, ... for registers corresponding to
  C variables

  $t0, $t1, ... for registers corresponding to
  temporary variables

  Will explain mapping convention later of $s0, $s1,
  ..., $t0, $t1, ..., to $r0, $r1, ...

- Note: whereas C declares its operands, Assembly
  operands (registers) are fixed and not declared
Compilation using Registers

- Compile by hand using registers:

\[
f = (g + h) - (i + j);
\]

# assign registers
#int f: $s0, int g: $s1, int h: $s2,
#int i: $s3, int j: $s4

- MIPS Instructions:

\[
\begin{align*}
\text{add } &\quad \$s0,\$s1,\$s2 \quad \text{# } \$s0 = g+h \\
\text{add } &\quad \$t1,\$s3,\$s4 \quad \text{# } \$t1 = i+j \\
\text{sub } &\quad \$s0,\$s0,\$t1 \quad \text{# } f=(g+h)-(i+j)
\end{align*}
\]