The Hardware/Software Interface
CSE351 Winter 2013

Instructor:
Peter Hornyack

Teaching Assistants:
Elliott Brossard, Matthew Dorsett, Dustin Lundquist, Mark Wyse

Who is your instructor?

- **History:**
  - Undergrad at University of Pennsylvania
  - Software engineer at Cisco
  - Grad student at UW

- **My research:**
  - Operating systems
  - Computer architecture
  - Security / privacy

- **First-time instructor**
  - I am not a doctor, nor a professor

Who are your TAs?

Elliott (AC)
Matt (AB)
Dustin
Mark (AA)

Notice

- **This lecture is being recorded!**
  - The microphone and projector are captured and will be combined into a screencast
  - Recordings are only accessible to students enrolled in the class after logging in with UW ID
  - Audio recordings can be downloaded
  - If any of this concerns you, see these links and come speak to me:
    - [http://www.css.washington.edu/wiki/Frequently_Asked_Questions_about_On-Demand_Audio_Service](http://www.css.washington.edu/wiki/Frequently_Asked_Questions_about_On-Demand_Audio_Service)

- **Information about how to view the recordings will be posted to website**
Who are you?

- 85 students
  
  We’ll do our best to get to know each of you!

Who has written a program in:

- Java?
- C?
- Assembly language?

The Hardware/Software Interface

- What is hardware? Software?
- What is an interface?
- Why do we need a hardware/software interface?
- Why do we need to understand both sides of this interface?

C/Java, assembly, and machine code

```c
if (x != 0) y = (y+z)/x;
```

```assembly
cmpq $0, -4(%ebp)  
je .L2
movl -12(%ebp), %eax  
movl -8(%ebp), %edx
lea (%edx, %eax), %eax
sarl $31, %edx
idivl -4(%ebp)
movl %eax, -8(%ebp)
.L2:
```

- The three program fragments are equivalent
- You’d rather write C! - a more human-friendly language
- The hardware likes bit strings! - everything is voltages
- The machine instructions are actually much shorter than the number of bits we would need to represent the characters in the assembly language
**HW/SW Interface: The Historical Perspective**

- **Hardware started out quite primitive**
  - Hardware designs were expensive => instructions had to be very simple
    - e.g., a single instruction for adding two integers
- **Software was also very basic**
  - Software primitives reflected the hardware pretty closely

---

**HW/SW Interface: Assemblers**

- **Life was made a lot better by assemblers**
  - 1 assembly instruction = 1 machine instruction, but...
  - different syntax: assembly instructions are character strings, not bit strings, a lot easier to read/write by humans
  - can use symbolic names

---

**HW/SW Interface: Higher-Level Languages**

- **Higher level of abstraction:**
  - 1 line of a high-level language is compiled into many (sometimes very many) lines of assembly language

---

**HW/SW Interface: Code / Compile / Run Times**

- **Code Time**
- **Compile Time**
- **Run Time**

*Note: The compiler and assembler are just programs, developed using this same process.*
Outline for today

- Course themes: big and little
- Roadmap of course topics
- Three important realities
- How the course fits into the CSE curriculum
- Logistics

Roadmap

The Big Theme: Interfaces and Abstractions

- Computing is about abstractions
  - (but we can’t forget reality)
- What are the abstractions that we use?
- What do YOU need to know about them?
  - When do they break down and you have to peek under the hood?
  - What bugs can they cause and how do you find them?
- How does the hardware (0s and 1s, processor executing instructions) relate to the software (C/Java programs)?
  - Become a better programmer and begin to understand the important concepts that have evolved in building ever more complex computer systems

Little Theme 1: Representation

- All digital systems represent everything as 0s and 1s
  - The 0 and 1 are really two different voltage ranges in the wires
- “Everything” includes:
  - Numbers – integers and floating point
  - Characters – the building blocks of strings
  - Instructions – the directives to the CPU that make up a program
  - Pointers – addresses of data objects stored away in memory
- These encodings are stored throughout a computer system
  - In registers, caches, memories, disks, etc.
- They all need addresses
  - A way to find them
  - Find a new place to put a new item
  - Reclaim the place in memory when data no longer needed
Little Theme 2: Translation

- There is a big gap between how we think about programs and data and the 0s and 1s of computers
- Need languages to describe what we mean
- Languages need to be translated one step at a time
  - Words, phrases and grammars
- We know Java as a programming language
  - Have to work our way down to the 0s and 1s of computers
  - Try not to lose anything in translation!
  - We’ll encounter Java byte-codes, C language, assembly language, and machine code (for the X86 family of CPU architectures)

Little Theme 3: Control Flow

- How do computers orchestrate the many things they are doing – seemingly in parallel
- What do we have to keep track of when we call a method, and then another, and then another, and so on
- How do we know what to do upon “return”
- User programs and operating systems
  - Multiple user programs
  - Operating system has to orchestrate them all
    - Each gets a share of computing cycles
    - They may need to share system resources (memory, I/O, disks)
  - Yielding and taking control of the processor
    - Voluntary or “by force”?

Course Outcomes

- Foundation: basics of high-level programming (Java)
- Understanding of some of the abstractions that exist between programs and the hardware they run on, why they exist, and how they build upon each other
- Knowledge of some of the details of underlying implementations
- Become more effective programmers
  - More efficient at finding and eliminating bugs
  - Understand some of the many factors that influence program performance
  - Facility with a couple more of the many languages that we use to describe programs and data
- Prepare for later classes in CSE

Reality #1: ints ≠ integers & floats ≠ reals

- Representations are finite
- Example 1: Is \( x^2 \geq 0 ? \)
  - Floats: Yes!
  - Ints:
    - \( 40000 \times 40000 \rightarrow 1600000000 \)
    - \( 50000 \times 50000 \rightarrow ?? \)
- Example 2: Is \( (x + y) + z = x + (y + z) ? \)
  - Unsigned & Signed Ints: Yes!
  - Floats:
    - \( (1e20 + -1e20) + 3.14 \rightarrow 3.14 \)
    - \( 1e20 + (-1e20 + 3.14) \rightarrow ?? \)
Reality #2: Assembly still matters

- Why? Because we want you to suffer?

Assembly Code Example

- Time Stamp Counter
  - Special 64-bit register in Intel-compatible machines
  - Incremented every clock cycle
  - Read with rdtsc instruction

- Application
  - Measure time (in clock cycles) required by procedure

```c
double t;
start_counter();
P();
t = get_counter();
printf("%f clock cycles\n", t);
```

Code to Read Counter

- Write small amount of assembly code using GCC’s asm facility
- Inserts assembly code into machine code generated by compiler

```c
/* Set *hi and *lo (two 32-bit values) to the high and low order bits of the cycle counter. */

void access_counter(unsigned *hi, unsigned *lo) {
    asm("rdtsc; movl %%edx, %0; movl %%eax, %1"
         : "=r" (*hi), "=r" (*lo) /* output */
         : "%edx", "%eax" /* input */
         : "clobbered" /* clobbered */
    );
}
```
Reality #3: Memory Matters

- So, what is memory?

Memory is not unbounded

- It must be allocated and managed
- Many applications are memory-dominated

Memory referencing bugs are especially pernicious

- Effects are distant in both time and space

Memory performance is not uniform

- Cache and virtual memory effects can greatly affect program performance
- Adapting program to characteristics of memory system can lead to major speed improvements

Memory Referencing Bug Example

```c
double fun(int i)
{
    volatile double d[1] = {3.14};
    volatile long int a[2];
    a[i] = 1073741824; /* Possibly out of bounds */
    return d[0];
}
```

fun(0)  ->  3.14
fun(1)  ->  3.14
fun(2)  ->  3.139999664856
fun(3)  ->  2.00000061035156
fun(4)  ->  3.14, then segmentation fault

Explanation:

<table>
<thead>
<tr>
<th>Saved State</th>
<th>Location accessed by fun(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>d7 ... d4</td>
<td>3</td>
</tr>
<tr>
<td>d3 ... d0</td>
<td>2</td>
</tr>
<tr>
<td>a[1]</td>
<td>1</td>
</tr>
<tr>
<td>a[0]</td>
<td>0</td>
</tr>
</tbody>
</table>

Saved State:

- d4
- d3
- d0
- a0
- a1
- a2
Memory Referencing Errors

- C (and C++) do not provide any memory protection
  - Out of bounds array references
  - Invalid pointer values
  - Abuses of malloc/free

- Can lead to nasty bugs
  - Whether or not bug has any effect depends on system and compiler
  - Action at a distance
    - Corrupted object logically unrelated to one being accessed
  - Effect of bug may be first observed long after it is generated

- How can I deal with this?
  - Program in Java (or C#, or ML, or …)
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors

Memory System Performance Example

- Hierarchical memory organization
- Performance depends on access patterns
  - Including how program steps through multi-dimensional array

How can I deal with this?

- Program in Java (or C#, or ML, or …)
- Understand what possible interactions may occur
- Use or develop tools to detect referencing errors

CSE351’s role in CSE Curriculum

- Pre-requisites
  - 142 and 143: Intro Programming I and II
  - Also recommended: 390A: System and Software Tools

- One of 6 core courses
  - 311: Foundations of Computing I
  - 312: Foundations of Computing II
  - 331: SW Design and Implementation
  - 332: Data Abstractions
  - 351: HW/SW Interface
  - 352: HW Design and Implementation

- 351 provides the context for many follow-on courses

CSE351’s place in CSE Curriculum

- Pre-requisites
  - 143: Intro Programming II

- Core Courses
  - CSE351: HW/SW Interface
  - CSE332: HW Design
  - CSE333: Systems Prog.
  - CSE451: Compilers
  - CSE452: Networks
  - CSE454: Security

- Electives
  - CSE466: Emb Systems
  - CSE461: Distributed Systems
  - CSE468: Machine Code
  - CSE477: Exec Model

The HW/SW Interface: underlying principles linking hardware and software

21 times slower
(Pentium 4)
Course Perspective

- **This course will make you a better programmer**
  - Purpose is to show how software really works
  - By understanding the underlying system, one can be more effective as a programmer
    - Better debugging
    - Better basis for evaluating performance
    - How multiple activities work in concert (e.g., OS and user programs)
  - Not just a course for dedicated hackers
    - What every CSE major needs to know
    - Job interviewers love to ask questions from 351!
  - Provide a context in which to place the other CSE courses you’ll take

Textbooks

- **Computer Systems: A Programmer’s Perspective, 2nd Edition**
  - Randal E. Bryant and David R. O’Hallaron
  - Prentice-Hall, 2010
  - [http://csapp.cs.cmu.edu](http://csapp.cs.cmu.edu)
  - This book really matters for the course:
    - How to solve labs
    - Practice problems typical of exam problems

- **A good C book – any will do**
  - The C Programming Language (Kernighan and Ritchie)
  - C: A Reference Manual (Harbison and Steele)

Course Components

- **Lectures (27)**
  - Introduce the concepts; supplemented by textbook

- **Sections (10)**
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam review and preparation

- **Written homework assignments (4)**
  - Mostly problems from text to solidify understanding

- **Labs (5, plus “lab 0”)**
  - Provide in-depth understanding (via practice) of an aspect of system

- **Exams (midterm + final)**
  - Test your understanding of concepts and principles
  - Midterm currently scheduled for Friday, February 15

Resources

- **Course web page**
  - [http://www.cse.washington.edu/351](http://www.cse.washington.edu/351)
  - Schedule, policies, labs, homeworks, and everything else

- **Course discussion board**
  - Keep in touch outside of class – help each other
  - Staff will monitor and contribute

- **Course mailing list**
  - Low traffic – mostly announcements; you are already subscribed

- **Office hours**
  - Will be posted this week

- **Staff e-mail**
  - Things that are not appropriate for discussion board or better offline

- **Anonymous feedback**
  - Any comments about anything related to the course where you would feel better not attaching your name
Policies: Grading

- Exams (40%): 15% midterm, 25% final
- Written assignments (20%): weighted according to effort
  - We’ll try to make these about the same
- Lab assignments (40%): weighted according to effort
  - These will likely increase in weight as the quarter progresses
- Late days:
  - 3 late days to use as you wish throughout the quarter – see website
- Collaboration:
  - http://www.cs.washington.edu/students/policies/misconduct

Welcome to CSE351!

- Let’s have fun
- Let’s learn – together
- Let’s communicate
- Let’s make this a useful class for all of us

- Many thanks to the many instructors who have shared their lecture notes – I will be borrowing liberally through the qtr – they deserve all the credit, the errors are all mine
  - CMU: Randy Bryant, David O’Halloran, Gregory Kesden, Markus Püschel
  - Harvard: Matt Welsh (now at Google-Seattle)
  - UW: Gaetano Borriello, Luis Ceze, Hal Perkins, John Zahorjan