Slide 1

Administrivia

- One purpose of the syllabus is to spell out policies (next slides).
- Most information will be on the Web, either on page my home page (office hours) or the course Web site (next slide).

  A request: If you spot something wrong with course material on the Web, please let me know!

Slide 2

Course Web Site

- "Course Web site" is meant to point you to pretty much all information for the course — readings, assignments, etc.
- You can find it via the link from my home page (findable from Trinity's Web site, or almost surely by doing a Web search on my name). (I will also put a link in TLearn.)

Slide 3

Course FAQ

- "What will my grade be based on?" (See syllabus.)
- "When are the exams?" (See syllabus.)
- "What happens if I can't turn in work on time, or I miss a class?" (See syllabus.)
- "What's your policy on collaboration?" (See syllabus.)

Slide 4

Course FAQ, Continued

- "When is the next homework due?" (See "Lecture topics and assignments" page.)
- "When are your office hours?" (See my home page.)

  Note that part of my job is to answer your questions outside class, so if you need help, please ask! in person or by e-mail. (*Note:* E-mail is often a very good way to reach me.)

Slide 5

Course FAQ, Continued

- "What computer(s) can I use to do programming homework?"

  Easiest option may be department's Linux machines. You should have physical access via your TigerCard to at least one classroom (the one where this course meets) the classrooms and labs (not today but soon) any time the building is open. You should also be able to log in remotely to any that are booted into Linux, or to a cluster of Linux-only machines in ITS's server room. To log in from off-campus, we are currently recommending that you use ITS's VDI. More about these computers, and remote access, can be found in my Web site about department computers (see syllabus or my home page).

Slide 6

Course FAQ, Continued

- You'll likely get an e-mail at some point about `echo360.org`. This is an ITS-sponsored platform that can be used to make available videos; I may use it later in the semester if I have to miss class for some reason, or to present some material in a way you can replay.

Slide 7

A Little About Me

- Up until recently I hadn't thought about telling students a little about myself on the first day. A colleague mentioned that she does, so — okay!
- Short version of biography: Undergrad degrees from UT Austin, math and Plan II. More than ten years in what we now call IT. Back to school for master's and PhD in computer science. Two years as a postdoc, then at Trinity since Fall 1999.
- I teach a variety of courses, but currently focusing more on courses "close to the machine". My research area (sadly neglected for some years) is parallel computing.
- (What do I do for fun? well . . .)

Slide 8

"Why Do I Have To Take This Course?"

- We could view computer systems (hardware/software) in terms of layers of abstraction:
  - User interface.
  - Operating system / application programs / tools (compilers, e.g.).
  - High-level programming language / ADTs.
  - Machine language / data representations ("it's all 1s and 0s").
  - Hardware (could break this down, maybe, into logical design and EE).
- One goal of a CS degree program is to "demystify" as many of these as we can.

Slide 9

"Why Do I Have To Take This Course?", Continued

- Relating courses to layers of abstraction:
  - Programming courses — bridge gap between user interface and high-level languages.
  - Operating systems course — bridge gap between user interface / applications programs and hardware.
  - Course on compilers — bridge gap between application programs and machine language.
  - This course — bridge gaps between application programs and machine language (a bit) and between machine language and hardware.

Slide 10

Course Topics

- An overview of how hardware is structured logically and how hardware and software are related.
- A little about defining and measuring performance.
- Assembly language (MIPS because it's simple and representative).
- Machine language (also MIPS).
- Hardware (at level of AND/OR gates).

Slide 11

Why Study Assembly / Machine Language?

- Understand the general principles of how things work at this level, which helps you:
  - Write more efficient programs.
  - Understand operating systems (which also helps you write more efficient programs).
  - Generally understand better what's really happening in the machine.
- It might be fun?

Slide 12

Introduction

- "Computers are everywhere" — you know about servers and desktops and smaller computing devices, all of which are more and more central to our lives, but also consider "embedded processors", largely invisible but even more prevalent.
- It seems to be a truism that however fast computers can process information, they can't keep up with humans' ability to imagine things for them to do. So performance matters.
- We'll start with an overview of hardware and software and how they interact (cf. textbook subtitle) and also talk a little about measuring performance.

Slide 13

"Below Your Program" — Review?

- Most programming these days is done in a high-level language (HLL), often using a lot of library code. Processors, however, can't execute it directly. How do we get from HLL to something the processor can do? Traditional view:
- First step is to *compile* — conceptually, to *assembly language* (symbolic representation of instructions the processor can execute).
- Next step is to translate assembly language into *machine language* (actual instructions for processors, in 1s and 0s), a.k.a. *object code*. Might be combined with compiling.

Slide 14

"Below Your Program", Continued

- Final step is to combine object code for your program with library object code. Can be done as part of compiling process to create an *executable file* or at runtime, or some combination of the two.
- Actual execution of program typically involves operating system (something that manages physical resources / provides abstraction for applications). Contents/format of executable files depends on operating system as well as hardware.
- Worth noting that some languages/implementations don't exactly follow this scheme: Some languages (e.g., shell scripts) are translated/interpreted at runtime, and others (e.g., Scala and Java) are compiled to machine language for a virtual processor (the JVM), which may then be translated into "native code" at runtime.

Slide 15

Minute Essay

- (Most lectures will end with a "minute essay" — as a quick check on your understanding, a way for me to get some information, etc., and also to track attendance. Just put your answer in the body of the message; no Word documents please, and *please* put "minute essay" and the course in the Subject line.)
- Tell me about your background: What programming classes have you taken (at Trinity or elsewhere)? What programming languages are you reasonably comfortable with?
- What are your goals for this course? Anything else you want to tell me? (Maybe something interesting you did over the break?)