Preface
(Revision number 00)

Why a New Book on Computer Systems?

There is an excitement when you talk to high school students about computers. There is a sense of mystery as to what is "inside the box" that makes the computer do such things as play video games with cool graphics, play music be it rap or symphony, send instant messages to friends, and so on. The purpose behind this textbook is to take the journey together to unravel the mystery of what is "inside the box." As a glimpse of what is to come, let us say at the outset that what makes the box interesting is not just the hardware but also how the hardware and the system software work in tandem to make it all happen. Therefore, the path we take in this book is to look at hardware and software together to see how one helps the other and together make the box interesting and useful. We call this approach, "unraveling the box": basically look inside the box and understand how to design the key hardware elements (processor, memory, and peripheral controllers) and the OS abstractions needed to manage all the hardware resources inside a box including processor, memory, I/O and disk, multiple processors, and network. Hence, this is a textbook for a first course in **computer systems**.

The book is intended for giving the breadth of knowledge in these topics at an early stage in a student's UG career (in CS or ECE). This textbook serves the need for teaching a course in an integrated fashion so that students can see the connection between the architecture and the system software. The material in this book may be taught as a 4-credit semester course, as a 5-credit quarter course, or as a two-quarter 3-credit course sequence. A course based on this textbook would serve well to prepare a student for more in-depth senior-level or graduate level courses that go deeper into computer architecture, operating systems, and networking, respectively, to cater to specialization in those areas. Further, such a course kindles interest in systems early on that can be capitalized to involve students in UG research.

Key features of the book (in addition to processor and memory systems):

1. A detailed treatment of storage systems.
2. A chapter devoted to networking issues.
3. A chapter devoted to multiprocessing and multithreaded programming.

Incidentally, this textbook grew out of teaching such an integrated course every semester from the Fall of 1999 in the College of Computing at Georgia Institute of Technology. In the beginning, we developed a comprehensive set of notes and slides for the course and used two standard textbooks (one for architecture and one for operating systems) as background reference material for the students to supplement the course material. In the Spring of 2005, we turned our courseware into a manuscript for a textbook since the students continually communicated to us a need for a textbook that matched the style and

contents of our course. An online version of this textbook has been in use at Georgia Tech since the Spring of 2005 for over 10 consecutive semesters (including summers).

The structure of the book

The intellectual content of the book is broken up into five units.

1. **Processor**
   The first unit deals with the processor and software issues associated with the processor. We start by understanding how to design the brain inside the box, the processor. What are the software issues? Since computers are programmed in high-level language for the most part, we consider the influence of high-level language (HLL) constructs on the instruction set of the processor (Chapter 2). Once we understand the design of instruction-set, next we focus on the hardware issue of implementing the processor. We start with a simple implementation of the processor (Chapter 3) and then go on to consider a performance-conscious implementation using pipelining techniques (Chapter 5). The processor is a precious resource that has to be multiplexed among several competing program that may need to run on it as illustrated by the video game example in Section 1.3. It is the OS's responsibility that this resource be used well. This unit concludes with OS algorithms for processor scheduling (Chapter 6).

2. **Memory System**
   The second unit deals with memory systems and memory hierarchies. Programs comprised of code and data need space to reside. The memory system of a computer is perhaps the most crucial determinant in determining its performance. The processor speed (measured in Gigahertz these days) may mean nothing if the memory system does not match that speed by providing the code and data for executing a program. While the size of memory systems is growing by leaps and bounds thanks to advances in technology, applications' appetite for using memory is growing equally if not faster. Thus, memory is a precious resource as well and it is the responsibility of the OS to manage this resource well. The first part of this unit concerns the OS algorithms for efficient management of memory and the architectural assists for supporting it (Chapters 7 and 8); the second part of this unit deals with the memory hierarchies that help to reduce the latency seen by the processor for accessing code and data (Chapter 9).

3. **Storage System**
   The third unit deals with I/O (particularly stable storage) and file system. What makes the computer useful and interesting is being able to interact with it. First, we deal with hardware mechanisms for grabbing the attention of the processor away from the currently executing program (Chapter 4). This is both for dealing with external events as well as internal exceptions encountered by the processor during program execution. Associated with the hardware mechanisms are software issues for dealing with these "discontinuities" in the normal program execution that include remembering where we are in the original program and the current state of the

program execution. Next, we delve into the mechanisms for interfacing the processor to I/O devices and the corresponding low-level software issues such as device drivers (Chapter 10), with a special emphasis on disk subsystem. This is followed with a comprehensive treatment of file system (Chapters 11) built on stable storage such as the disk.

4. **Parallel System**
   Computer architecture is a fast changing field. Chip density, processor speed, memory capacity, etc., have all been showing exponential growth over the last two decades and are expected to continue that trend for the foreseeable future. Parallel processing is no longer an esoteric concept reserved for supercomputers. With the advent of multicore technology that contains multiple CPUs inside a single chip, parallelism is becoming a commodity. Therefore, understanding the hardware and software issues surrounding parallelism is necessary to answer the question "what is inside a box?" This unit deals with operating systems issues and the corresponding architectural features in multiprocessors for supporting parallel programming (Chapter 12).

5. **Networking**
   In the world we live in, a box is almost useless unless it is connected to the outside world. The multiplayer video game with your friends as fellow warriors on the network is a nice motivating example, but even in our everyday mundane activities, we need the network for e-mail, web browsing, etc. What distinguishes the network from other input/output devices is the fact that your box is now exposed to the world! You need a language to talk to the outside world from your box and deal with the vagaries of the network such as temporary or permanent disconnections. This unit deals with the evolution of networking hardware, and the features of the network protocol stack (which is part of the operating system) for dealing with the vagaries of the network (Chapter 13).

The hardware and software issues for each of the above five units will be treated together in this textbook.

Where Does This Textbook Fit into the Continuum of CS Curriculum?

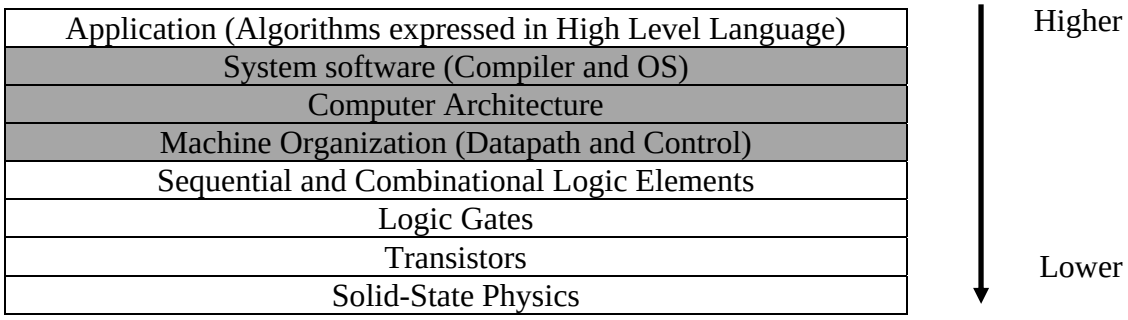

Figure 1.1: Levels of Abstraction in a Computer System

Figure 1.1 shows the levels of abstraction in a computer system. We can try to relate the levels of abstraction in Figure 1.1 to courses in a typical CS curriculum. Courses such as basic programming, object oriented design and programming, graphics, HCI generally deal with higher layers of abstraction. Typically, CS and ECE curricula offer courses dealing with the fundamentals of digital electronics and logic design, followed by a course on computer organization that deals purely with the hardware design of a computer. Beyond the computer organization course (moving up the levels of abstraction shown in Figure 1.1), most curricula take a stovepipe approach: distinct courses dealing with advanced concepts in computer architecture, operating systems, and computer networks, respectively.

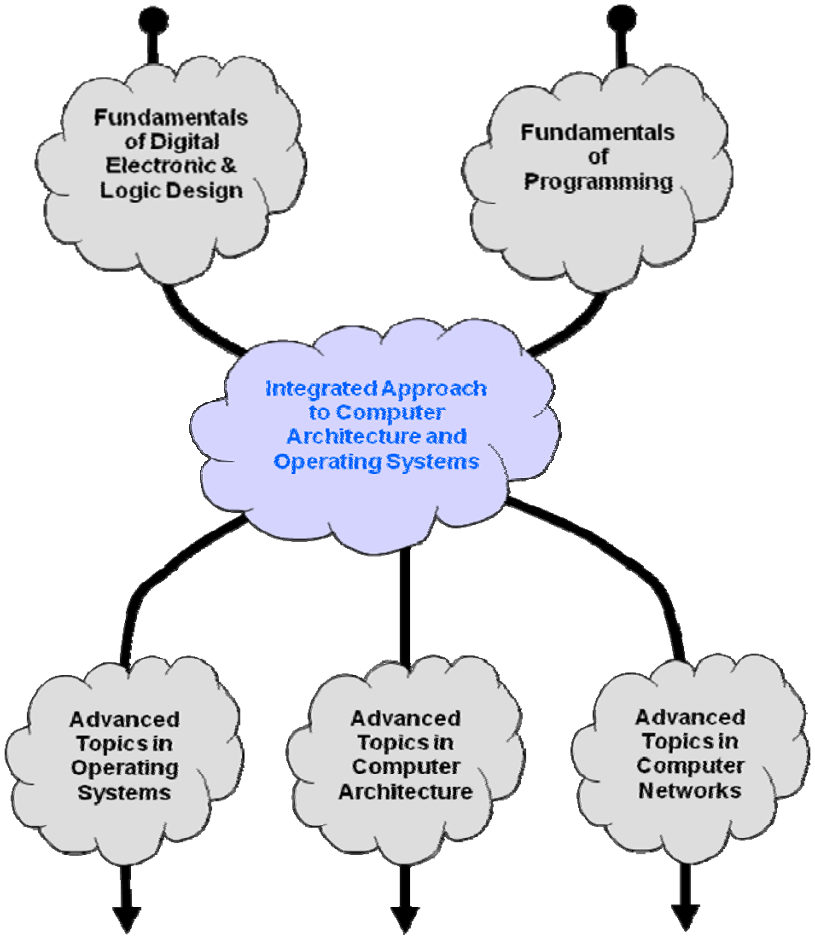

Figure 1.2: Systems Course Sequence

Design of computer systems is such an integrated process today that one has to seriously question this stovepipe approach, especially in the early stages of the development of a student in an undergraduate curriculum in Computer Science.

A course structured around the topics covered in this book is a unique attempt to present concepts in the middle (covering topics in the shaded area - systems software and their relationship to computer architecture) in a unified manner in an introductory systems

course. Such a course would serve as a solid preparation for students aspiring to learn advanced topics in computer architecture, operating systems, and networking (Figure 1.2).

There are excellent textbooks that cater to the fundamentals of digital electronics and logic design, as well as fundamentals of programming. Similarly, there are excellent textbooks that deal with advanced topics in computer architecture, operating systems, and computer networking. ***What is missing is a simplified and integrated introduction to computer systems that serves as the bridge between the fundamentals and the advanced topics. The aim of this textbook is to fill this void.***

The boundary of computer science as a discipline has expanded. Correspondingly, students coming into this discipline have varied interests. There is a need for CS curricula to offer choices for students to pursue in their undergraduate careers. At the same time, there is a responsibility to ensure that students acquire "core" knowledge in systems (broadly defined) regardless of these choices. We believe a course structured around this textbook would fulfill such a core systems requirement. If done right, it should give ample opportunity for students to pursue deeper knowledge in systems, if they so choose, through further coursework. For example, our recommendation would be to have a course based on this textbook in the sophomore year. In the junior year, the students may be able to take courses that are more design oriented in architecture, operating systems, and/or networking, building on the basic concepts learned in the sophomore year via this textbook. Finally, in the senior year the students may be able to take more conceptual courses on advanced topics in these areas.

Supplementary Material for Teaching an Integrated Course in Systems

The authors fully understand the challenge an instructor faces in teaching an integrated course in computer systems that touches on architecture, operating systems, and networking.

To this end, we make available a set of online resources. Since we have been teaching this course for the last nine years as a required one for all CS majors (3 offerings in each calendar year), there is a significant collection of online resources:

1. We have PowerPoint slides for all the topics covered in the course making preparation and transition (from the stovepipe model) easy.
2. As enumerated above, a significant project component dovetails each of the five modules. We have detailed project descriptions of several iterations of these projects along with software modules (such as simulators) for specific aspects of the projects.
3. In addition to the problems at the end of each chapter, we have additional problem sets for the different modules of the course, as well as homework problems and midterm and final exams used thus far in the course.

Example project ideas included in the supplementary material

Processor Design: Students are supplied a data path design that is 90% complete. Students complete the data path to help them become familiar with the design. Then they design the microcode-based control logic (using LogicWorks) for implementing a simple instruction-set using the data path. This allows the students to get a good understanding of how a data path functions and to appreciate some of the design tradeoffs. The students get actual circuit design experience and functionally test their design using the built-in functional simulator of LogicWorks.

Interrupts & Input/Output: Students take the design from the first project and add circuitry to implement an interrupt system. Then they write (in assembly language) an interrupt handler. The circuit design part of the project is once again implemented and functionally simulated using LogicWorks. In addition, the students are supplied with a processor simulator that they enhance with the interrupt support and use it in concert with the interrupt handler that they write in assembly language. This project not only makes operation of the interrupt system clear but also illustrates fundamental concepts of low-level device input/output.

Virtual Memory Subsystem: Students implement a virtual memory subsystem that operates with a supplied processor simulator. The students get the feel for developing the memory management part of an operating system through this project. The project is implemented in the C programming language.

Multi-Threaded Operating System: Students implement the basic modules of a multi-threaded operating system including CPU and I/O queues on top of a simulator that we supply. They experiment with different processor scheduling policies. The modules are implemented in C using pthreads. The students get experience with parallel programming as well as exposure to different CPU scheduling algorithms.

Reliable Transport Layer: Students implement a simple reliable transport layer on top of a simulated network layer provided to them. Issues that must be dealt with in the transport layer include corrupt packets, missing packets, and out-of-order delivery. This project is also implemented in C using pthreads.