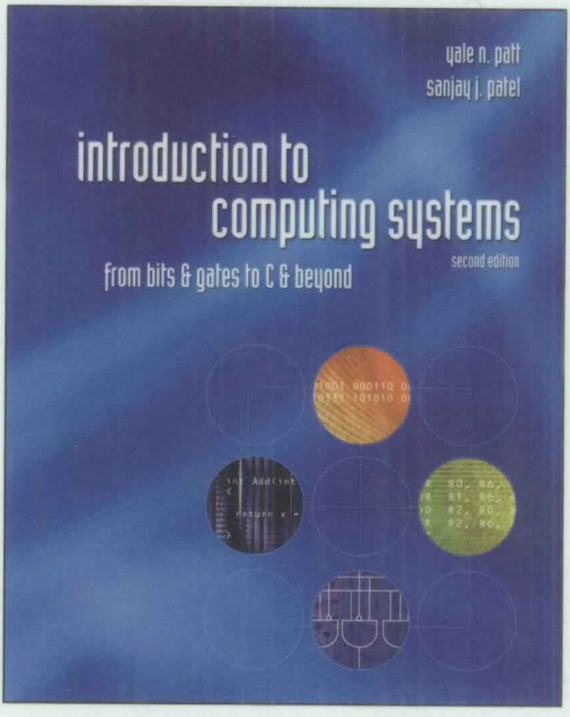
HZ BOOKS

经 典 原 版 书 库

计算机系统概论

（英文版 · 第2版）

INTERNATIONAL EDITION



McGRAW-HILL

Yale N. Patt
得克萨斯大学奥斯汀分校
（美）
Sanjay J. Patel
伊利诺伊大学厄巴纳-尚佩恩分校

机械工业出版社
China Machine Press

Mc Graw Hill Education

经 典 原 版 书 库

计算机系统概论

（英文版 · 第2版）

Introduction to Computing Systems
From Bits and Gates to C and Beyond
(Second Edition)

(美) Yale N. Patt
得克萨斯大学奥斯汀分校
Sanjay J. Patel
伊利诺伊大学厄巴纳-尚佩恩分校
著

机械工业出版社
China Machine Press

Yale N. Patt and Sanjay J. Patel: Introduction to Computing Systems: From Bits and Gates to C and Beyond, Second Edition (ISBN 0-07-246750-9).

本书版权登记号：图字：01-2006-3885

图书在版编目（CIP）数据

计算机系统概论（英文版·第2版）/（美）帕特（Patt, Y. N.）等著. －北京：机械工业出版社，2006.9

（经典原版书库）

书名原文：Introduction to Computing Systems: From Bits and Gates to C and Beyond, Second Edition

ISBN 7-111-19766-6

Ⅰ. 计…　Ⅱ. 帕…　Ⅲ. 计算机系统－英文　Ⅳ. TP30

中国版本图书馆CIP数据核字（2006）第096647号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：迟振春

北京京北制版印刷厂印刷·新华书店北京发行所发行

2006年9月第1版第1次印刷

170mm × 242mm · 41.25印张

定价：66.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔

滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzjsj@hzbook.com
联系电话：(010) 68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

To the memory of my parents,
Abraham Walter Patt A"H and Sarah Clara Patt A"H,
who taught me to value "learning"
even before they taught me to ride a bicycle.

To Mira and her grandparents,
Sharda Patel and Jeram Patel.

preface

It is a pleasure to be writing a preface to the second edition of this book. Three years have passed since the first edition came out. We have received an enormous number of comments from students who have studied the material in the book and from instructors who have taught from it. Almost all have been very positive. It is gratifying to know that a lot of people agree with our approach, and that this agreement is based on real firsthand experience learning from it (in the case of students) or watching students learn from it (in the case of instructors). The excitement displayed in their e-mail continues to be a high for us.

However, as we said in the preface to the first edition, this book will always be a "work in progress." Along with the accolades, we have received some good advice on how to make it better. We thank you for that. We have also each taught the course two more times since the first edition came out, and that, too, has improved our insights into what we think we did right and what needed improvement. The result has been a lot of changes in the second edition, while hopefully maintaining the essence of what we had before. How well we have succeeded we hope to soon learn from you.

Major Changes to the First Edition

The LC-3

One of the more obvious changes in the second edition is the replacement of the LC-2 with the LC-3. We insisted on keeping the basic concept of the LC-2: a rich ISA that can be described in a few pages, and hopefully mastered in a short time. We kept the 16-bit instruction and 4-bit opcode. One of our students pointed out that the subroutine return instruction (RET) was just a special case of LC-2's JMPR instruction, so we eliminated RET as a separate opcode. The LC-3 specifies only 15 opcodes—and leaves one for future use (perhaps, the third edition!).

We received a lot of push-back on the PC-concatenate addressing mode, particularly for branches. The addressing mode had its roots in the old PDP-8 of the mid-1960s. A major problem with it comes up when an instruction on one page wants to dereference the next (or previous) page. This has been a major hassle, particularly for forward branches close to a page boundary. A lot of people have asked us to use the more modern PC+offset, and we agreed. We have replaced all uses of PC'offset with PC+SEXT(offset).

We incorporated other changes in the LC-3. Stacks now grow toward 0, in keeping with current conventional practice. The offset in LDR/STR is now

a signed value, so addresses can be computed plus or minus a base address. The opcode 1101 is not specified. The JSR/JMP opcodes have been reorganized slightly. Finally, we expanded the condition codes to a 16-bit processor status register (PSR) that includes a privilege mode and a priority level. As in the first edition, Appendix A specifies the LC-3 completely.

Additional Material

Although no chapter in the book has remained untouched, some chapters have been changed more than others. We added discussions to Chapter 1 on the nature and importance of abstraction and the interplay of hardware and software because it became clear that these points needed to be made explicit. We added a full section to Chapter 3 on finite state control and its implementation as a sequential switching circuit because we believe the concept of state and finite state control are among the most important concepts a computer science or engineering student encounters. We feel it is also useful to the understanding of the von Neumann model of execution discussed in Chapter 4. We added a section to Chapter 4 giving a glimpse of the underlying microarchitecture of the LC-3, which is spelled out in all its detail in the overhauled Appendix C. We were told by more than one reader that Chapter 5 was too terse. We added little new material, but lots of figures and explanations that hopefully make the concepts clearer. We also added major new sections on interrupt-driven I/O to Chapters 8 and 10.

Just as in the first edition, Chapters 11 through 14 introduce the C programming language. Unlike the first edition, these chapters are more focused on the essential aspects of the language useful to a beginning programmer. Specialized features, for example the C switch construct, are relegated to the ends of the chapters (or to Appendix D), out of the main line of the text. All of these chapters include more examples than the first edition. The second edition also places a heavier emphasis on "how to program" via problem-solving examples that demonstrate how newly introduced C constructs can be used in C programming. In Chapter 14, students are exposed to a new LC-3 calling convention that more closely reflects the calling convention used by real systems. Chapter 15 contains a deeper treatment of testing and debugging. Based on our experiences teaching the introductory course, we have decided to swap the order of the chapter on recursion with the chapter on pointers and arrays. Moving recursion later (now Chapter 17) in the order of treatment allows students to gain more experience with basic programming concepts before they start programming recursive functions.

The Simulator

Brian Hartman has updated the simulator that runs on Windows to incorporate the changes to the LC-3. Ashley Wise has written an LC-3 simulator that runs on UNIX. Both have incorporated interrupt-driven I/O into the simulator's functionality. We believe strongly that there is no substitute for hands-on practice testing one's knowledge. With the addition of interrupt-driven I/O to the simulator, the student can now interrupt an executing program by typing a key on the keyboard and invoke an interrupt service routine.

Alternate Uses of the Book

We wrote the book as a textbook for a freshman introduction to computing. We strongly believe, as stated more completely in the preface to our first edition, that our motivated bottom-up approach is the best way for students to learn the fundamentals of computing. We have seen lots of evidence that suggests that in general, students who understand the fundamentals of how the computer works are better able to grasp the stuff that they encounter later, including the high-level programming languages that they must work in, and that they can learn the rules of these programming languages with far less memorizing because everything makes sense. For us, the best use of the book is a one-semester freshman course for particularly motivated students, or a two-semester sequence where the pace is tempered. If you choose to go the route of a one-semester course heavy on high-level language programming, you probably want to leave out the material on sequential machines and interrupt-driven I/O. If you choose to go the one-semester route heavy on the first half of the book, you probably want to leave out much of Chapters 15, 17, 18, and 19.

We have also seen the book used effectively in each of the following environments:

Two Quarters, Freshman Course

In some sense this is the best use of the book. In the first quarter, Chapters 1 through 10 are covered; in the second quarter, Chapters 11 through 19. The pace is brisk, but the entire book can be covered in two academic quarters.

One-Semester Second Course

The book has been used successfully as a second course in computing, after the student has spent the first course with a high-level programming language. The rationale is that after exposure to high-level language programming in the first course, the second course should treat at an introductory level digital logic, basic computer organization, and assembly language programming. Most of the semester is spent on Chapters 1 through 10, with the last few weeks spent on a few topics from Chapters 11 through 19, showing how some of the magic from the students' first course can actually be implemented. Functions, activation records, recursion, pointer variables, and some elementary data structures are typically the topics that get covered.

A Sophomore-Level Computer Organization Course

The book has been used to delve deeply into computer implementation in the sophomore year. The semester is spent in Chapters 1 through 10, sometimes culminating in a thorough study of Appendix C, which provides the complete microarchitecture of a microprogrammed LC-3. We note, however, that some very important ideas in computer architecture are not covered here, most notably cache memory, pipelining, and virtual memory. We agree that these topics are very important to the education of a computer scientist or computer engineer, but we feel these topics are better suited to a senior course in computer architecture and design. This book is not intended for that purpose.

Acknowledgments

Our book continues to benefit greatly from important contributions of many, many people. We particularly want to acknowledge Brian Hartman and Matt Starolis.

Brian Hartman continues to be a very important part of this work, both for the great positive energy he brings to the table and for his technical expertise. He is now out of school more than three years and remains committed to the concept. He took the course the first year it was offered at Michigan (Winter term, 1996), TAed it several times as an undergraduate student, and wrote the first LC-2 simulator for Windows while he was working on his master's degree. He recently upgraded the Windows simulator to incorporate the new LC-3.

Matt Starolis took the freshman course at UT two years ago and TAed it as a junior last fall. He, too, has been very important to us getting out this second edition. He has been both critic of our writing and helpful designer of many of the figures. He also updated the tutorials for the simulators, which was necessary in order to incorporate the new characteristics of the LC-3. When something needed to be done, Matt volunteered to do it. His enthusiasm for the course and the book has been a pleasure.

With more than 100 adopters now, we regularly get enthusiastic e-mail with suggestions from professors from all over the world. Although we realize we have undoubtedly forgotten some, we would at least like to thank Professors Vijay Pai, Rice; Richard Johnson, Western New Mexico; Tore Larsen, Tromso; Greg Byrd, NC State; Walid Najjar, UC Riverside; Sean Joyce, Heidelberg College; James Boettler, South Carolina State; Steven Zeltmann, Arkansas; Mike McGregor, Alberta; David Lilja, Minnesota; Eric Thompson, Colorado, Denver; and Brad Hutchings, Brigham Young.

Between the two of us, we have taught the course four more times since the first edition came out, and that has produced a new enthusiastic group of believers, both TAs and students. Kathy Buckheit, Mustafa Erwa, Joseph Grzywacz, Chandresh Jain, Kevin Major, Onur Mutlu, Moinuddin Qureshi, Kapil Sachdeva, Russell Schreiber, Paroma Sen, Santhosh Srinath, Kameswar Subramaniam, David Thompson, Francis Tseng, Brian Ward, and Kevin Woley have all served as TAs and have demonstrated a commitment to helping students learn that can only be described as wonderful. Linda Bigelow, Matt Starolis, and Lester Guillory all took the course as freshmen, and two years later they were among the most enthusiastic TAs the course has known.

Ashley Wise developed the Linux version of the LC-3 simulator. Ajay Ladsaria ported the LCC compiler to generate LC-3 code. Gregory Muthler and Francesco Spadini enthusiastically provided critical feedback on drafts of the chapters in the second half.

Kathy Buckheit wrote introductory tutorials to help students use the LC-2 simulator because she felt it was necessary.

Several other faculty members at The University of Texas have used the book and shared their insights with us: Tony Ambler, Craig Chase, Mario Gonzalez, and Earl Swartzlander in ECE, and Doug Burger, Chris Edmundson, and Steve Keckler in CS. We thank them.

We continue to celebrate the commitment displayed by our editors, Betsy Jones and Michelle Flomenhoft.

As was the case with the first edition, our book has benefited from extensive reviews provided by faculty members from many universities. We thank Robert Crisp, Arkansas; Allen Tannenbaum, Georgia Tech; Nickolas Jovanovic, Arkansas–Little Rock; Dean Brock, North Carolina–Asheville; Amar Raheja, Cal State–Pomona; Dayton Clark, Brooklyn College; William Yurcik, Illinois State; Jose Delgado-Frias, Washington State; Peter Drexel, Plymouth State; Mahmoud Manzoul, Jackson State; Dan Connors, Colorado; Massoud Ghyam, Southern Cal; John Gray, UMass–Dartmouth; John Hamilton, Auburn; Alan Rosenthal, Toronto; and Ron Taylor, Wright State.

Finally, there are those who have contributed in many different and often unique ways. Without listing their individual contributions, we simply list them and say thank you. Amanda, Bryan, and Carissa Hwu, Mateo Valero, Rich Belgard, Janak Patel, Matthew Frank, Milena Milenkovic, Lila Rhoades, Bruce Shriver, Steve Lumetta, and Brian Evans. Sanjay would like to thank Ann Yeung for all her love and support.

A Final Word

It is worth repeating our final words from the preface to the first edition: We are mindful that the current version of this book will always be a work in progress, and we welcome your comments on any aspect of it. You can reach us by e-mail at patt@ece.utexas.edu and sjp@crhc.uiuc.edu. We hope you will.

Yale N. Patt
Sanjay J. Patel
May, 2003

preface to the first edition

This textbook has evolved from EECS 100, the first computing course for computer science, computer engineering, and electrical engineering majors at the University of Michigan, that Kevin Compton and the first author introduced for the first time in the fall term, 1995.

EECS 100 happened because Computer Science and Engineering faculty had been dissatisfied for many years with the lack of student comprehension of some very basic concepts. For example, students had a lot of trouble with pointer variables. Recursion seemed to be "magic," beyond understanding.

We decided in 1993 that the conventional wisdom of starting with a high-level programming language, which was the way we (and most universities) were doing it, had its shortcomings. We decided that the reason students were not getting it was that they were forced to memorize technical details when they did not understand the basic underpinnings.

The result is the bottom-up approach taken in this book. We treat (in order) MOS transistors (very briefly, long enough for students to grasp their global switch-level behavior), logic gates, latches, logic structures (MUX, Decoder, Adder, gated latches), finally culminating in an implementation of memory. From there, we move on to the Von Neumann model of execution, then a simple computer (the LC-2), machine language programming of the LC-2, assembly language programming of the LC-2, the high level language C, recursion, pointers, arrays, and finally some elementary data structures.

We do not endorse today's popular information hiding approach when it comes to learning. Information hiding is a useful productivity enhancement technique after one understands what is going on. But until one gets to that point, we insist that information hiding gets in the way of understanding. Thus, we continually build on what has gone before, so that nothing is magic, and everything can be tied to the foundation that has already been laid.

We should point out that we do not disagree with the notion of top-down *design*. On the contrary, we believe strongly that top-down design is correct design. But there is a clear difference between how one approaches a design problem (after one understands the underlying building blocks), and what it takes to get to the point where one does understand the building blocks. In short, we believe in top-down design, but bottom-up learning for understanding.

What Is in the Book

The book breaks down into two major segments, a) the underlying structure of a computer, as manifested in the LC-2; and b) programming in a high level language, in our case C.

The LC-2

We start with the underpinnings that are needed to understand the workings of a real computer. Chapter 2 introduces the bit and arithmetic and logical operations on bits, Then we begin to build the structure needed to understand the LC-2. Chapter 3 takes the student from a MOS transistor, step by step, to a real memory. Our real memory consists of 4 words of 3 bits each, rather than 64 megabytes. The picture fits on a single page (Figure 3.20), making it easy for a student to grasp. By the time the students get there, they have been exposed to all the elements that make memory work. Chapter 4 introduces the Von Neumann execution model, as a lead-in to Chapter 5, the LC-2.

The LC-2 is a 16-bit architecture that includes physical I/O via keyboard and monitor; TRAPs to the operating system for handling service calls; conditional branches on N, Z, and P condition codes; a subroutine call/return mechanism; a minimal set of operate instructions (ADD, AND, and NOT); and various addressing modes for loads and stores (direct, indirect, Base+offset, and an immediate mode for loading effective addresses).

Chapter 6 is devoted to programming methodology (stepwise refinement) and debugging, and Chapter 7 is an introduction to assembly language programming. We have developed a simulator and an assembler for the LC-2. Actually, we have developed two simulators, one that runs on Windows platforms and one that runs on UNIX. The Windows simulator is available on the website and on the CD-ROM. Students who would rather use the UNIX version can download and install the software from the web at no charge.

Students use the simulator to test and debug programs written in LC-2 machine language and in LC-2 assembly language. The simulator allows online debugging (deposit, examine, single-step, set breakpoint, and so on). The simulator can be used for simple LC-2 machine language and assembly language programming assignments, which are essential for students to master the concepts presented throughout the first 10 chapters.

Assembly language is taught, but not to train expert assembly language programmers. Indeed, if the purpose was to train assembly language programmers, the material would be presented in an upper-level course, not in an introductory course for freshmen. Rather, the material is presented in Chapter 7 because it is consistent with the paradigm of the book. In our bottom-up approach, by the time the student reaches Chapter 7, he/she can handle the process of transforming assembly language programs to sequences of 0s and 1s. We go through the process of assembly step-by-step for a very simple LC-2 Assembler. By hand assembling, the student (at a very small additional cost in time) reinforces the important fundamental concept of translation.

It is also the case that assembly language provides a user-friendly notation to describe machine instructions, something that is particularly useful for the

second half of the book. Starting in Chapter 11, when we teach the semantics of C statements, it is far easier for the reader to deal with ADD R1, R2, R3 than with 0001001010000011.

Chapter 8 deals with physical input (from a keyboard) and output (to a monitor). Chapter 9 deals with TRAPs to the operating system, and subroutine calls and returns. Students study the operating system routines (written in LC-2 code) for carrying out physical I/O invoked by the TRAP instruction.

The first half of the book concludes with Chapter 10, a treatment of stacks and data conversion at the LC-2 level, and a comprehensive example that makes use of both. The example is the simulation of a calculator, which is implemented by a main program and 11 subroutines.

The Language C

From there, we move on to C. The C programming language occupies the second half of the book. By the time the student gets to C, he/she has an understanding of the layers below.

The C programming language fits very nicely with our bottom-up approach. Its low-level nature allows students to see clearly the connection between software and the underlying hardware. In this book we focus on basic concepts such as control structures, functions, and arrays. Once basic programming concepts are mastered, it is a short step for students to learn more advanced concepts such as objects and abstraction.

Each time a new construct in C is introduced, the student is shown the LC-2 code that a compiler would produce. We cover the basic constructs of C (variables, operators, control, and functions), pointers, recursion, arrays, structures, I/O, complex data structures, and dynamic allocation.

Chapter 11 is a gentle introduction to high-level programming languages. At this point, students have dealt heavily with assembly language and can understand the motivation behind what high-level programming languages provide. Chapter 11 also contains a simple C program, which we use to kick-start the process of learning C.

Chapter 12 deals with values, variables, constants, and operators. Chapter 13 introduces C control structures. We provide many complete program examples to give students a sample of how each of these concepts is used in practice. LC-2 code is used to demonstrate how each C construct affects the machine at the lower levels.

In Chapter 14, students are exposed to techniques for debugging high-level source code. Chapter 15 introduces functions in C. Students are not merely exposed to the syntax of functions. Rather they learn how functions are actually executed using a run-time stack. A number of examples are provided.

Chapter 16 teaches recursion, using the student's newly gained knowledge of functions, activation records, and the run-time stack. Chapter 17 teaches pointers and arrays, relying heavily on the student's understanding of how memory is organized. Chapter 18 introduces the details of I/O functions in C, in particular,

streams, variable length argument lists, and how C I/O is affected by the various format specifications. This chapter relies on the student's earlier exposure to physical I/O in Chapter 8. Chapter 19 concludes the coverage of C with structures, dynamic memory allocation, and linked lists.

Along the way, we have tried to emphasize good programming style and coding methodology by means of examples. Novice programmers probably learn at least as much from the programming examples they read as from the rules they are forced to study. Insights that accompany these examples are highlighted by means of lightbulb icons that are included in the margins.

We have found that the concept of pointer variables (Chapter 17) is not at all a problem. By the time students encounter it, they have a good understanding of what memory is all about, since they have analyzed the logic design of a small memory (Chapter 3). They know the difference, for example, between a memory location's address and the data stored there.

Recursion ceases to be magic since, by the time a student gets to that point (Chapter 16), he/she has already encountered all the underpinnings. Students understand how stacks work at the machine level (Chapter 10), and they understand the call/return mechanism from their LC-2 machine language programming experience, and the need for linkages between a called program and the return to the caller (Chapter 9). From this foundation, it is not a large step to explain functions by introducing run-time activation records (Chapter 15), with a lot of the mystery about argument passing, dynamic declarations, and so on, going away. Since a function can call a function, it is one additional small step (certainly no magic involved) for a function to call itself.

How to Use This Book

We have discovered over the past two years that there are many ways the material in this book can be presented in class effectively. We suggest six presentations below:

1. The Michigan model. First course, no formal prerequisites. Very intensive, this course covers the entire book. We have found that with talented, very highly motivated students, this works best.
2. Normal usage. First course, no prerequisites. This course is also intensive, although less so. It covers most of the book, leaving out Sections 10.3 and 10.4 of Chapter 10, Chapters 16 (recursion), 18 (the details of C I/O), and 19 (data structures).
3. Second course. Several schools have successfully used the book in their second course, after the students have been exposed to programming with an object-oriented programming language in a milder first course. In this second course, the entire book is covered, spending the first two-thirds of the semester on the first 10 chapters, and the last one-third of the semester on the second half of the book. The second half of the book can move more quickly, given that it follows both Chapters 1–10 and the

introductory programming course, which the student has already taken. Since students have experience with programming, lengthier programming projects can be assigned. This model allows students who were introduced to programming via an object-oriented language to pick up C, which they will certainly need if they plan to go on to advanced software courses such as operating systems.

4. Two quarters. An excellent use of the book. No prerequisites, the entire book can be covered easily in two quarters, the first quarter for Chapters 1–10, the second quarter for Chapters 11–19.
5. Two semesters. Perhaps the optimal use of the book. A two-semester sequence for freshmen. No formal prerequisites. First semester, Chapters 1–10, with supplemental material from Appendix C, the Microarchitecture of the LC-2. Second semester, Chapters 11–19 with additional substantial programming projects so that the students can solidify the concepts they learn in lectures.
6. A sophomore course in computer hardware. Some universities have found the book useful for a sophomore level breadth-first survey of computer hardware. They wish to introduce students in one semester to number systems, digital logic, computer organization, machine language and assembly language programming, finishing up with the material on stacks, activation records, recursion, and linked lists. The idea is to tie the hardware knowledge the students have acquired in the first part of the course to some of the harder to understand concepts that they struggled with in their freshman programming course. We strongly believe the better paradigm is to study the material in this book before tackling an object-oriented language. Nonetheless, we have seen this approach used successfully, where the sophomore student gets to understand the concepts in this course, after struggling with them during the freshman year.

Some Observations

Understanding, Not Memorizing

Since the course builds from the bottom up, we have found that less memorization of seemingly arbitary rules is required than in traditional programming courses. Students understand that the rules make sense since by the time a topic is taught, they have an awareness of how that topic is implemented at the levels below it. This approach is good preparation for later courses in design, where understanding of and insights gained from fundamental underpinnings are essential to making the required design tradeoffs.

The Student Debugs the Student's Program

We hear complaints from industry all the time about CS graduates not being able to program. Part of the problem is the helpful teaching assistant, who contributes far too much of the intellectual component of the student's program, so the student