HZ BOOKS

经　典　原　版　书　库

ELSEVIER

计算机硬件及组成原理

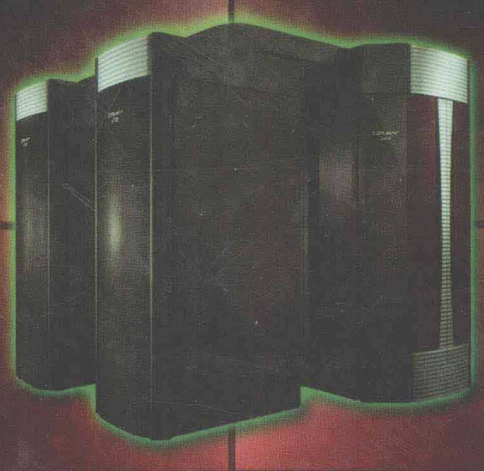
(英文版)

EMBEDDED TECHNOLOGY™
SERIES

ELSEVIER

Hardware and Computer Organization

The Software Perspective



Arnold S. Berger

DVD-ROM Included
Contains Instruction Set Simulators, PowerPoint® slides and video lectures from industry experts

附赠
CD-ROM

(美) Arnold S. Berger 著

机械工业出版社
China Machine Press

经 典 原 版 书 库

计算机硬件及组成原理

(英文版)

Hardware and Computer Organization
The Software Perspective

(美) Arnold S. Berger 著

机 械 工 业 出 版 社
China Machine Press

本书版权登记号：图字：01-2005-4838

图书在版编目（CIP）数据

计算机硬件及组成原理（英文版）/（美）伯杰（Berger, A. S.）著；　－北京：机械工业出版社，2006.1

（经典原版书库）

书名原文：Hardware and Computer Organization: The Software Perspective

ISBN 7-111-17472-0

Ⅰ. 计…　Ⅱ. 伯…　Ⅲ. ①硬件-英文　②计算机体系结构-英文　Ⅳ. TP303

中国版本图书馆CIP数据核字（2005）第112827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

北京京北制版厂印刷 · 新华书店北京发行所发行

2006年 1 月第 1 版第 1 次印刷

718mm × 1020mm 1/16 · 32印张

印数：0 001- 3 000册

定价：59.00元（附光盘）

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）

Preface

Thank you for buying my book. I know that may ring hollow if you are a poor student and your instructor made it the required text for your course, but I thank you nevertheless. I hope that you find it informative and easy to read. At least that was one of my goals when I set out to write this book.

This text is an outgrowth of a course that I've been teaching in the Computing and Software Systems Department of the University of Washington-Bothell. The course, CSS 422, *Hardware and Computer Organization,* is one of the required core courses for our undergraduate students. Also, it is the only required architecture course in our curriculum. While our students learn about algorithms and data structures, comparative languages, numeric methods and operating systems, this is their only exposure to "what's under the hood." Since the University of Washington is on the quarter system, I'm faced with the uphill battle to teach as much about the architecture of computers as I can in about 10 weeks.

The material that forms the core of this book was developed over a period of 5 years in the form of about 500 Microsoft PowerPoint® slides. Later, I converted the material in the slides to HTML so that I could also teach the course via a distance learning (DL) format. Since first teaching this course in the fall of 1999, I've taught it 3 or 4 times each academic year. I've also taught it 3 times via DL, with excellent results. In fact, the DL students as a whole have done equally well as the students attending lectures in class. So, if you think that attending class is a highly overrated part of the college experience, then this book is for you.

The text is appropriate for a first course ın computer architecture at the sophomore through senior level. It is reasonably self-contained so that it should be able to serve as the only hardware course that CS students need to take in order to understand the implications of the code that they are writing. At the University of Washington-Bothell (UWB), this course is predominantly taught to seniors. As a faculty, we've found that the level of sophistication achieved through learning programming concepts in other classes makes for an easier transition to low-level programming. If the book is to be used with lower division students, then additional time should be allotted for gaining fluency with assembly language programming concepts. For example, in introducing certain assembly language branching and loop constructs, an advanced student will easily grasp the similarity to WHILE, DO-WHILE, FOR and IF-THEN-ELSE constructs. A less sophisticated student may need more concrete examples in order to see the similarities.

Why write a book on Computer Architecture? I'm glad you asked. In the 5+ years that I taught the course, I changed the textbook 4 times. At the end of the quarter, when I held an informal course debriefing with the students, they universally panned every book that I used. The "gold standard" textbooks, the texts that almost every Computer Science student uses in their architecture class, were just not relevant to their needs. For the majority of these students, they were not going to go on and study architecture in graduate schools, or design computers for Intel or AMD. They needed to understand the architecture of a computer and its supporting hardware in order to write efficient and defect-free code to run on the machines. Recently, I did find a text that at least approached the subject matter in the same way that I thought it should be done, but I also found that text lacking in several key areas. On the plus side, switching to the new text eliminated the complaints from my students and it also reinforced my opinion that I wasn't alone in seeing a need for a text with a different perspective. Unfortunately, this text, even though it was a great improvement, still did not cover several areas that I considered to be very important, so I resolved to write one that did, without losing the essence of what I think the new perspective correctly accomplished.

It's not surprising that, given the UWB campus is less than 10 miles from Microsoft's main campus in Redmond, WA, we are strongly influenced by the Microsoft culture. The vast majority of my students have only written software for Windows and the Intel architecture. The designers of this architecture would have you believe that these computers are infinitely fast machines with unlimited resources. How do you counter this view of the world?

Often, my students will cry out in frustration, "Why are you making me learn this (deleted)?" (Actually it is more of a whimper.) This usually happens right around the mid-term examination. Since our campus is also approximately equidistant from where Boeing builds the 737 and 757 aircraft in Renton, WA and the wide body 767 and 777 aircraft in Everett, WA, analogies to the aircraft industry are usually very effective. I simply answer their question this way, "Would you fly on an airplane that was designed by someone who is clueless about what keeps an airplane in the air?" Sometimes it works.

The book is divided into four major topic areas:

1. Introduction to hardware and asynchronous logic.
2. Synchronous logic, state machines and memory organization.
3. Modern computer architectures and assembly language programming.
4. I/O, computer performance, the hierarchy of memory and future directions of computer organization.

There is no sharp line of demarcation between the subject areas, and the subject matter builds upon the knowledge base from prior chapters. However, I've tried to limit the interdependencies so later chapters may be skipped, depending upon the available time and desired syllabus.

Each chapter ends with some exercises. The solutions to the odd-numbered problems are located in Appendix A, and the solutions to the even-numbered problems are available through the instructor's resource website at http://www.elsevier.com/0750678860.

The text approach that we'll take is to describe the hardware from the ground up. Just as a Geneticist can describe the most complex of organic beings in terms of a DNA molecule that contains only four nucleotides, adenine, cytosine, guanine, and thymine, abbreviated, A, C, G and T, we can describe the most complex computer or memory system in terms of four logical building blocks, AND, OR, NOT and TRI-STATE. Strictly speaking, TRI-STATE isn't a logical building block like AND, it is more like the "glue" that enables us to interconnect the elements of a computer in such a way that the complexity doesn't overwhelm us. Also, I really like the DNA analogy, so we'll need to have 4 electronic building blocks to keep up with the A, C, G and T idea.

I once gave a talk to a group of middle school teachers who were trying to earn some in-service credits during their summer break. I was a volunteer with the Air Academy School District in Colorado Springs, Colorado while I worked for the Logic Systems Division of Hewlett-Packard. None of the teachers were computer literate and I had two hours to give them some appreciation of the technology. I decided to start with Aristotle and concept of the logical operators as a branch of philosophy and then proceeded with the DNA analogy up through the concept of registers. I seemed to be getting through to them, but they may have been stroking my ego so I would sign-off on their attendance sheets. Anyway, I think there is value in demonstrating that even the most complex computer functionality can be described in terms of the logic primitives that we study in the first part of the text.

We will take the DNA or building-block approach through most of the first half of the text. We will start with the simplest of gates and build compound gates. From these compound gates we'll progress to the posing and solution of asynchronous logical equations. We'll learn the methods of truth table design and simplification using Boolean algebra and then Karnaugh Map (K-map) methodology. The exercises and examples will stress the statement of the problem as a set of specifications which are then translated into a truth table, and from there to K-maps and finally to the gate design. At this point the student is encouraged to actually "build" the circuit in simulation using the Digital Works® software simulator (see the following) included on the DVD-ROM that accompanies the text. I have found this combination of the abstract design and the actual simulation to be an extremely powerful teaching and learning combination.

One of the benefits of taking this approach is that the students become accustomed to dealing with variables at the bit level. While most students are familiar with the C/C++ Boolean constructs, the concept of a single wire carrying the state of a variable seems to be quite new.

Once the idea of creating arbitrarily complex, asynchronous algebraic functions is under control, we add the dimension of the clock and of synchronous logic. Synchronous logic takes us to flip-flops, counters, shifters, registers and state machines. We actually spend a lot of effort in this area, and the concepts are reintroduced several times as we look at micro-code and instruction decomposition later on.

The middle part of the book focuses on the architecture of a computer system. In particular, the memory to CPU interface will get a great deal of attention. We'll design simple memory systems and decoding circuits using our knowledge gained in the preceding chapters. We'll also take a brief look at memory timing in order to better understand some of the more global issues of system design.

We'll then make the transition to looking at the architecture of the 68K, ARM and X86 processor families. This will be our introduction to assembly language programming.

Each of the processor architectures will be handled separately so that it may be skipped without creating too much discontinuity in the text.

This text does emphasize assembly language programming in the three architectures. The reason for this is twofold: First, assembly language may or may not be taught as a part of a CS student's curriculum, and it may be their only exposure to programming at the machine level. Even though you as a CS student may never have to write an assembly language program, there's a high probability that you'll have to debug some parts of your C++ program at the assembly language level, so this is as good a time to learn it as any. Also, by looking at three very different instruction sets we will actually reinforce the concept that once you understand the architecture of a processor, you can program it in assembly language. This leads to the second reason to study assembly language. Assembly language is a metaphor for studying computer architecture from the software developer's point of view.

I'm a big fan of "Dr. Science." He's often on National Public Radio and does tours of college campuses. His famous tag line is, "I have a Master's degree...in science." Anyway, I was at a Dr. Science lecture when he said, "I like to scan columns of random numbers, looking for patterns." I remembered that line and I often use it in my lectures to describe how you can begin to see the architecture of the computer emerging through the seeming randomness of the machine language instruction set. I could just see in my mind's eye a bunch of Motorola CPU architects and engineers sitting around a table in a restaurant, pizza trays scattered hither and yon, trying to figure out the correct bit patterns for the last few instructions so that they don't have a bloated and inefficient microcode ROM table. If you are a student reading this and it doesn't make any sense to you now, don't worry...yet.

The last parts of the text steps back and looks at general issues of computer architecture. We'll look at CISC versus RISC, modern techniques, such as pipelines and caches, virtual memory and memory management. However, the overriding theme will be computer performance. We will keep returning to the issues associated with the software-to-hardware interface and the implications of coding methods on the hardware and of the hardware on the coding methods.

One unique aspect of the text is the material included on the accompanying DVD-ROM. I've included the following programs to use with the material in the text:

- Digital Works (freeware): A hardware design and simulation tool
- Easy68K: A freeware assembler/simulator/debugger package for the Motorola (Now Freescale) 68,000 architecture.
- X86emul: A shareware assembler/simulator/debugger package for the X86 architecture.
- GNU ARM Tools: The ARM developers toolset with Instruction Set Simulator from the Free Software Foundation.

The ARM company has an excellent tool suite that you can obtain directly from ARM. It comes with a free 45-day evaluation license. This should be long enough to use in your course. Unfortunately, I was unable to negotiate a license agreement with ARM that would enable me to include the ARM tools on the DVD-ROM that accompanies this text. This tool suite is excellent and easy to use. If you want to spend some additional time examining the world's most popular RISC architecture, then contact ARM directly and ask them nicely for a copy of the ARM tools suite. Tell them Arnie sent you.

I have also used the Easy68K assembler/simulator extensively in my CSS 422 class. It works well and has extensive debugging capabilities associated with it. Also, since it is freeware, the logistical problems of licenses and evaluation periods need not be dealt with. However, we will be making some references to the other tools in the text, so it is probably a good idea to install them just before you intend to use them, rather than at the beginning of your class.

Also included on the DVD-ROM are 11 short lectures on various topics of interest in this text by experts in the field of computer architecture. These videos were made under a grant by the Worthington Technology Endowment for 2004 at UWB. Each lecture is an informal 15 to 30 minute "chalk talk." I hope you take the time to view them and integrate them into your learning experience for this subject matter.

Even though the editors, my students and I have read through this material several times, Murphy's Law predicts that there is still a high probability of errors in the text. After all, it's software. So if you come across any "bugs" in the text, please let me know about it. Send your corrections to aberger@u.washington.edu. I'll see to it that the corrections will gets posted on my website at UW (http://faculty.uwb.edu/aberger).

The last point I wanted to make is that textbooks can just go so far. Whether you are a student or an instructor reading this, please try to seek out experts and original sources if you can. Professor James Patterson, writing in the July, 2004 issue of *Physics Today*, writes,

> *When we want to know something, there is a tendency to seek a quick answer in a textbook. This often works, but we need to get in the habit of looking at original papers. Textbooks are often abbreviated second-or third-hand distortions of the facts...*[1]

Let's get started.

Arnold S. Berger
Sammamish, Washington

[1] James D. Patterson, *An Open Letter to the Next Generation, Physics Today*, Vol. 57, No. 7, July, 2004, p. 56.

Acknowledgments

First, and foremost, I would like to acknowledge the sponsorship and support of Professor William Erdly, former Director of the Department of Computing and Software Systems at the University of Washington-Bothell. Professor Erdly first hired me as an Adjunct Professor in the fall of 1999 and asked me to teach a course called Hardware and Computer Organization, even though the course I wanted to teach was on Embedded System Design.

Professor Erdly then provided me with financial support to convert my Hardware and Computer Organization course from a series of lectures on PowerPoint slides to a series of online lessons in HTML. These lessons became the core of the material that led to this book.

Professors Charles Jackels and Frank Cioch, in their capacities as Acting Directors, both supported my work in perfecting the on-line material and bringing multimedia into the distance learning experience. Their support helped me to see the educational value of technology in the classroom.

I would also like to thank the Richard P. and Lois M. Worthington Foundation for their 2004 Technology Grant Award which enabled me to travel around the United States and videotape short talks in Computer Architecture. Also, I would like to thank the 11 speakers who gave of their time to participate in this project.

Carol Lewis, my Acquisitions Editor at Newnes Books saw the value of my approach and I thank her for it. I'm sorry that she had to leave Newnes before she could see this book finally completed. Tiffany Gasbarrini of Elsevier Press and Kelly Johnson of Borrego Publishing brought the book to life. Thank you both.

In large measure, this book was designed by the students who have taken my CSS 422 course. Their end-of-Quarter evaluations and feedback were invaluable in helping me to see how this book should be structured.

Finally, and most important, I want to thank my wife Vivian for her support and understanding. I could not have written 500+ pages without it.

What's on the DVD-ROM?

One unique aspect of the text is the material included on the accompanying DVD-ROM. I've included the following programs to use with the material in the text:

- Digital Works (freeware): A hardware design and simulation tool.
- Easy68K: A freeware assembler/simulator/debugger package for the Motorola (now Freescale) 68,000 architecture.
- X86emul: A shareware assembler/simulator/debugger package for the X86 architecture.
- GNU ARM Tools: The ARM developers toolset with Instruction Set Simulator from the Free Software Foundation.
- Eleven industry expert video lectures on significant hardware design and development topics.

Contents