计 算 机 科 学 丛 书

Computer Architecture
A Quantitative Approach

计算机体系结构量化研究方法

(英文版 · 第2版)

(美) David A. Patterson
John L. Hennessy 著

机械工业出版社 China Machine Press
MORGAN KAUFMANN

计算机科学丛书

计算机体系结构量化研究方法

（英文版 第2版）

Computer Architecture A Quantitative Approach

(Second Edition)

(美) David A. Patterson
John L. Hennessy 著

机械工业出版社
China Machine Press

本书版权登记号：图字：01-1999-2022

图书在版编目（CIP）数据

计算机体系结构：量化研究方法：第2版：英文/(美)帕蒂森(Patterson, D. A.)，(美)亨尼丝(Hennessy, J. L.)著.-北京：机械工业出版社，1999.9

(计算机科学丛书)

ISBN 7-111-07439-4

Ⅰ.计… Ⅱ.①帕… ②亨… Ⅲ.电子计算机－系统结构－英文 Ⅳ.TP303

中国版本图书馆CIP数据核字(1999)第33785号

出 版 者：马九荣（北京市西城区百万庄大街22号 邮政编码 100037）

北京忠信诚胶印厂印刷·新华书店北京发行所发行

1999年9月第1版第1次印刷

787mm × 1092mm1/16 · 64.5印张

印数：0 001- 4 000册

定价：88.00元

Computer Architecture Definitions, Formulas, and Rules of Thumb

Definitions

Antidependence: Dependence resulting from reuse of a name; corresponds to WAR hazard (page 232).

ATM (asynchronous transfer mode): A switch-based network standard (page 613).

Big Endian: The byte with the binary address "x...x00" is in the most significant position ("big end") of a 32-bit word (page 73).

Clock rate: Inverse of clock cycle time, usually measured in MHz (page 32).

CPI: Clock cycles per instruction (page 32).

Data dependence: True dependence resulting from use of a data value produced by an earlier statement; corresponds to a RAW hazard (page 229).

DSM: Distributed shared memory (page 640).

Hit rate: Fraction of memory references found in the cache, equal to 1 – Miss rate (page 400).

Hit time: Memory-access time for a cache hit, including time to determine if hit or miss (page 384).

Instruction count: Number of instructions executed while running a program (page 32).

Little Endian: The byte with the binary address "x...x00" is in the least significant position ("little end") of a 32-bit word (page 73).

MIMD (multiple instruction stream, multiple data stream): A multiprocessor or multicomputer (page 637).

Miss penalty: Time to replace a block in the top level of a cache system with the corresponding block from the lower level (page 43).

Miss rate: Fraction of memory references not found in the cache, equal to 1 – Hit rate (page 43).

$N_{1/2}$: The vector length needed to reach one-half R_∞ (page B-30).

N_v: The vector length needed so that vector mode is faster than scalar mode (page B-30).

Output dependence: Dependence resulting from two writes to the same name; corresponds to a WAW hazard (page 232).

R_∞: The megaFLOPS rate of an infinite-length vector (page B-30).

RAW data hazard (read after write): An instruction tries to read a source before a prior instruction writes it, so it incorrectly gets the old value (page 151).

SIMD (single instruction stream, multiple data stream): An array processor (page 637).

SISD (single instruction stream, single data stream): A uniprocessor (page 637).

Spatial locality (locality in space): If an item is referenced, nearby items will tend to be referenced soon (page 38).

Superscalar: A machine that executes multiple instructions per clock cycle with instruction issue controlled dynamically by hardware (page 278).

Temporal locality (locality in time): If an item is referenced, it tends to be referenced again soon (page 38).

TCP/IP (Transmission Control Protocol/Internet Protocol): The standard network protocols used on the Internet (page 609).

VLIW: A machine that executes multiple instructions per clock cycle with instruction issue controlled statically by software (page 278).

WAR data hazard (write after read): An instruction tries to write a destination before it is read by a prior instruction, so the prior instruction incorrectly gets the new value (page 151).

WAW data hazard (write after write): An instruction tries to write an operand before it is written by a prior instruction. The writes are performed in the wrong order, incorrectly leaving the value of the prior instruction in the destination (page 151).

Formulas

1. *CPU time* = Instruction count × Clock cycles per instruction × Clock cycle time (page 32)

2. *Amdahl's Law:* $\text{Speedup}_{\text{overall}} = \dfrac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \dfrac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \dfrac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$ (page 30)

3. *Average memory-access time* = Hit time + Miss rate × Miss penalty (page 384)

4. *Means*—arithmetic (AM), weighted arithmetic (WAM), harmonic (HM), and weighted harmonic (WHM):

$$\text{AM} = \frac{1}{n}\sum_{i=1}^{n} \text{Time}_i \qquad \text{WAM} = \sum_{i=1}^{n} \text{Weight}_i \times \text{Time}_i \qquad \text{HM} = \frac{n}{\sum_{i=1}^{n} \frac{1}{\text{Rate}_i}} \qquad \text{WHM} = \frac{1}{\sum_{i=1}^{n} \frac{\text{Weight}_i}{\text{Rate}_i}}$$

   where Time_i is the execution time for the ith program of a total of n in the workload, Weight_i is the weighting of the ith program in the workload, and Rate_i is a function of $1/\text{Time}_i$ (pages 25–26)

5. *Cost of integrated circuit* = $\dfrac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$ (page 10)

6. *Die yield* = $\text{Wafer yield} \times \left(1 + \dfrac{\text{Defects per unit area} \times \text{Die area}}{\alpha}\right)^{-\alpha}$

   where Wafer yield accounts for wafers that are so bad they need not be tested and α corresponds to the number of masking levels critical to die yield (usually $\alpha = 3.0$, page 12)

7. *Speedup from pipelining* = $\dfrac{1}{1 + \text{Pipeline stall cycles per instruction}} \times \dfrac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}$

   $= \dfrac{1}{1 + \text{Pipeline stall cycles per instruction}} \times \text{Pipeline depth}$

   where Pipeline stall cycles account for clock cycles lost due to pipeline hazards (page 141)

8. *Little's Law:* Mean number of tasks in system = Arrival rate × Mean response time assuming the system is in equilibrium (page 508)

Rules of Thumb

1. *Amdahl/Case Rule:* A balanced computer system needs about 1 MB of main memory capacity and 1 megabit per second of I/O bandwidth per MIPS of CPU performance.
2. *90/10 Locality Rule:* A program executes about 90% of its instructions in 10% of its code (page 38).
3. *DRAM-Growth Rule:* Density increases by about 60% per year, quadrupling in three years (page 7).
4. *Disk-Growth Rule:* Density increases by about 50% per year, quadrupling in just over three years (page 7).
5. *Address-Consumption Rule:* The memory needed by the average program grows by about a factor of 1.5 to 2 per year; thus, it consumes between 1/2 and 1 address bit per year (page 6).
6. *85/60 Branch-Taken Rule:* About 85% of backward-going branches are taken while about 60% of forward-going branches are taken (page 166).
7. *2:1 Cache Rule:* The miss rate of a direct-mapped cache of size N is about the same as a two-way set-associative cache of size $N/2$ (page 396).

DLX Instruction Set, Description Notation, and DLX Pipeline Structure

DLX Standard Instruction Set

Instruction type/opcode	Instruction meaning
Data transfers	**Move data between registers and memory, or between the integer and FP or special registers; only memory address mode is 16-bit displacement + contents of a GPR**
LB, LBU, SB	Load byte, load byte unsigned, store byte
LH, LHU, SH	Load half word, load half word unsigned, store half word
LW, SW	Load word, store word (to/from integer registers)
LF, LD, SF, SD	Load SP float, load DP float, store SP float, store DP float
MOVI2S, MOVS2I	Move from/to GPR to/from a special register
MOVF, MOVD	Copy one FP register or a DP pair to another register or pair
MOVFP2I, MOVI2FP	Move 32 bits from/to FP registers to/from integer registers
Arithmetic/logical	**Operations on integer or logical data in GPRs; signed arithmetic trap on overflow**
ADD, ADDI, ADDU, ADDUI	Add, add immediate (all immediates are 16 bits); signed and unsigned
SUB, SUBI, SUBU, SUBUI	Subtract, subtract immediate; signed and unsigned
MULT, MULTU, DIV, DIVU	Multiply and divide, signed and unsigned; operands must be FP registers; all operations take and yield 32-bit values
AND, ANDI	And, and immediate
OR, ORI, XOR, XORI	Or, or immediate, exclusive or, exclusive or immediate
LHI	Load high immediate—loads upper half of register with immediate
SLL, SRL, SRA, SLLI, SRLI, SRAI	Shifts: both immediate (S__I) and variable form (S__); shifts are shift left logical, right logical, right arithmetic
S__, S__I	Set conditional: "__" may be LT, GT, LE, GE, EQ, NE
Control	**Conditional branches and jumps; PC-relative or through register**
BEQZ, BNEZ	Branch GPR equal/not equal to zero; 16-bit offset from PC+4
BFPT, BFPF	Test comparison bit in the FP status register and branch; 16-bit offset from PC+4
J, JR	Jumps: 26-bit offset from PC+4 (J) or target in register (JR)
JAL, JALR	Jump and link: save PC+4 in R31, target is PC-relative (JAL) or a register (JALR)
TRAP	Transfer to operating system at a vectored address
RFE	Return to user code from an exception; restore user mode
Floating point	**FP operations on DP and SP formats**
ADDD, ADDF	Add DP, SP numbers
SUBD, SUBF	Subtract DP, SP numbers
MULTD, MULTF	Multiply DP, SP floating point
DIVD, DIVF	Divide DP, SP floating point
CVTF2D, CVTF2I, CVTD2F, CVTD2I, CVTI2F, CVTI2D	Convert instructions: CVTx2y converts from type x to type y, where x and y are I (integer), D (double precision), or F (single precision). Both operands are FPRs.
__D, __F	DP and SP compares: "__" = LT, GT, LE, GE, EQ, NE; sets bit in FP status register

Hardware Description Notation (and some standard C operators)

Notation	Meaning	Example	Meaning
←	Data transfer. Length of transfer is given by the destination's length; the length is specified when not clear. The left-hand side may have multiple destinations separated by commas.	`Regs[R1]← Regs[R2];`	Transfer contents of R2 to R1. Registers have a fixed length, so transfers shorter than the register size must indicate which bits are used.
M	Array of memory accessed in bytes. The starting address for a transfer is indicated as the index to the memory array.	`Regs[R1]←M[x];`	Place contents of memory location x into R1. If a transfer starts at M[i] and requires 4 bytes, the transferred bytes are M[i], M[i+1], M[i+2], and M[i=3].
←n	Transfer an *n*-bit field, used whenever length of transfer is not clear.	$M[y] \leftarrow_{16} M[x]$;	Transfer 16 bits starting at memory location x to memory location y. The length of the two sides should match.
X_n	Subscript selects a bit.	$Regs[R1]_0 \leftarrow 0$;	Change sign bit of R1 to 0. (Bits are numbered from MSB starting at 0.)
$X_{m..n}$	Subscript selects a field.	$Regs[R3]_{24..31} \leftarrow M[x]$;	Moves contents of memory location x into low-order byte of R3.
X^{n}	Superscript replicates a bit field.	$Regs[R3]_{0..23} \leftarrow 0^{24}$;	Sets high-order three bytes of R3 to 0.
##	Concatenates two fields.	$Regs[R3] \leftarrow 0^{24}$ ## M[x]; $F2\#\#F3 \leftarrow_{64} M[x]$;	Moves contents of location x into low byte of R3; clears upper three bytes. Moves 64 bits from memory starting at location x; first 32 bits go into F2, second 32 into F3.
*, &	Dereference a pointer; get the address of a variable.	p*←&x;	Assign to object pointed to by p, the address of the variable x.
<<, >>	C logical shifts (left, right).	Regs[R1] << 5	Shift R1 left 5 bits.
==, !=, >, <, >=, <=	C relational operators: equal, not equal, greater, less, greater or equal, less or equal.	(Regs[R1]== Regs[R2]) & (Regs[R3]!= Regs[R4])	True if the contents of R1 equal the contents of R2 and the contents of R3 do not equal the contents of R4.
&, |, ^, !	C bitwise logical operations: and, or, exclusive or, and complement.	(Regs[R1] & (Regs[R2]| Regs[R3]))	Bitwise and of R1 and the bitwise or of R2 and R3.

DLX Pipeline Structure

Stage	Any instruction		
IF	IF/ID.IR ← Mem[PC]; IF/ID.NPC,PC ← (if EX/MEM.cond {EX/MEM.ALUOutput} else {PC+4});		
ID	ID/EX.A ← Regs[IF/ID.IR$_{6..10}$]; ID/EX.B ← Regs[IF/ID.IR$_{11..15}$]; ID/EX.NPC ← IF/ID.NPC; ID/EX.IR ← IF/ID.IR; ID/EX.Imm ← (IF/ID.IR$_{16}$)^{16}##IF/ID.IR$_{16..31}$;		
	ALU instruction	**Load or store instruction**	**Branch instruction**
EX	EX/MEM.IR ← ID/EX.IR; EX/MEM.ALUOutput← ID/EX.A *func* ID/EX.B; or EX/MEM.ALUOutput ← ID/EX.A *op* ID/EX.Imm; EX/MEM.cond ← 0;	EX/MEM.IR← ID/EX.IR EX/MEM.ALUOutput ← ID/EX.A + ID/EX.Imm; EX/MEM.cond ← 0; EX/MEM.B← ID/EX.B;	 EX/MEM.ALUOutput ← ID/EX.NPC+ID/EX.Imm; EX/MEM.cond ← (ID/EX.A *op* 0);
MEM	MEM/WB.IR ← EX/MEM.IR; MEM/WB.ALUOutput ← EX/MEM.ALUOutput;	MEM/WB.IR ← EX/MEM.IR; MEM/WB.LMD ← Mem[EX/MEM.ALUOutput]; or Mem[EX/MEM.ALUOutput] ← EX/MEM.B;	
WB	Regs[MEM/WB.IR$_{16..20}$] ← MEM/WB.ALUOutput; or Regs[MEM/WB.IR$_{11..15}$] ← MEM/WB.ALUOutput;	Regs[MEM/WB.IR$_{11..15}$] ← MEM/WB.LMD;	

To Andrea, Linda, and our four sons

About the Authors

David A.Patterson(University of California at Berkeley)has taught computer architecture since joining the faculty in 1977 and is holder of the E. H. and M. E. Pardee Chair of Computer Science. His teaching has been honored by the ACM with the Outstanding Educator Award and by the University of California with the Distinguished Teaching Award. He also received the inaugural Outstanding Alumnus Award of the UCIA Computer Science Department. He is a member of the National Academy of Engineering and is a Fellow of both the IEEE and the Association for Computing Machinery (ACM).

Past chair of the CS Division in the EECS Department at Berkeley and the ACM Special Interest Group in Computer Architecture, Patterson is currently chair of the Computing Research Association. He has consulted for many companies, including Digital, HP, Intel, and Sun, and is also co-author of five books.

At Berkeley, he led the design and implementation of RISC I, likely the first VLSI Reduced Instruction Set Computer, This research became the foundation of the SPARC architecture, currently used by Fujitsu, ICL, Sun, TI, and Xerox. He was also a leader of the Redundant Arrays of Inexpensive Disks (RAID) project, which led to high-performance storage systems from many companies. These projects led to three distinguished dissertation awards from the ACM. His current research interests are in large-scale computing using networks of workstations (NOW).

JOHN L. HENNESSY (Stanford University) has been a member of the Stanford faculty since 1997, where he teaches computer architecture and supervises a group of energetic Ph. D. students. He is currently Chairman of the Computer Science Department and holds the Willard R. and Inez Kerr Bell Professorship in the School of Engineering. Hennessy is a Fellow of the IEEE, a member of the National Academy of Engineering , and a Fellow of the American Academy of Arts and Sciences. He received the 1994 IEEE Piore Award for his contributions to the development of RISC technology.

Hennessy's original research area was optimizing compilers. His research group at Stanford developed many of the techniques now in commercial use. In 1981, he started the MIPS project at Stanford with a handful of graduate students. After completing the project in 1984, he took a one-year leave form the university to co-found MIPS Computer Systems, which has since merged with Silicon Graphics. Hennessy's recent research at Stanford focuses on the area of designing and exploiting multiprocessors. Most recently, he has been involved in the development of the DASH multiprocessor architecture, one of the first distributed shared-memory multiprocessors.

Foreword

by C. Gordon Bell

Once again, I am honored and delighted to write the foreword to what is now the second edition of this landmark book.

With this edition, which is actually a whole new book, my 1990 prediction that *Computer Architecture: A Quantitative Approach* would be the standard reference for computer systems analysis and design is reaffirmed. In the interim between editions, the authors wrote and published *Computer Organization and Design: The Hardware/Software Interface* to serve as the introductory foundation for this book.

In the two three-year generations of Moore's Law since the first edition of *Computer Architecture*, the number of transistors per chip has increased by a factor of 16 and circuit speed has more than quadrupled. Disk densities have increased at similar rates of 60% per year. However, main memory and disk access times have not improved at comparable rates. These order-of-magnitude changes in technology have initiated numerous changes in instruction set and system architecture, which have driven the revision of this text around the theme of increased parallelism.

Of course, many of the technological changes we are speaking of were spurred on by advances in hardware. Since the first edition, we have seen the need for a two- or three-level cache and memory hierarchy to match processing speeds; the adoption of hardwired, pipelined instruction level parallelism in architectures; the universal adoption of the shared bus multiprocessor, or "multi," to replace the use of many uniprocessors; a significant decrease in the use of vector processors; the use of parallel disks or redundant arrays of inexpensive disks (RAID)—previously covered in only two pages; and the prevalence of interconnected computers for performance and reliability.

In addition to these hardware-induced changes, the new text reflects a conceptual change: The notion of a unique computer architecture for size and class has been replaced by a comparison and description of various Alpha, MIPS, SPARC, PowerPC, and PA-RISC microprocessors that can be combined to scale a range of applications. Gone from this edition are the VAX and IBM 360 architectures, known as a minicomputer and mainframe, respectively. Instead, the Intel 80x86, or PC architecture, is described in its own appendix, because after all it is the reference architecture that all others compete with in PCs, workstations, various multiprocessors, and networks.

Thus, the material herein is required *understanding* for anyone working with architecture or hardware, including architects, chip and computer system engineers, and compiler and operating system engineers. It would prove beneficial to

software engineers writing parallel programs for multiprocessors and computer clusters, as well.

The second edition continues the authors' particular style of analysis in sections such as *Fallacies and Pitfalls* and *Putting It All Together* and in new sections on *Crosscutting Issues.* In addition, they continue to provide a sense of context in sections on *Historical Perspective.*

The reason this book remains a standard of comparison is the understanding, experience, taste, and *uniqueness* of the authors. Originally, Hennessy and Patterson stimulated a major change in instruction set architecture through their work on RISC (Patterson coined the word). Their university research led to the MIPS and SPARC instruction set architectures for the '90s. Most recently, their work has moved to higher architecture levels, stimulating the understanding and development of single bus and distributed shared-memory multiprocessors (Hennessy) and of clusters, such as networks of workstations and RAID (Patterson).

I'm looking forward to the changes that will necessitate the 2002 version. This book will help make all the intervening computers better and more useful, and therefore I hope you enjoy, learn from, and use this fine text.

Preface

I started in 1962 to write a single book with this sequence of chapters, but soon found that it was more important to treat the subjects in depth rather than to skim over them lightly. The resulting length has meant that each chapter by itself contains enough to publish the series in separate volumes...

Donald Knuth, *The Art of Computer Programming,*
Preface to Volume 1 (of 7) (1968)

Why We Wrote This Book

This quote from Donald Knuth has never been truer, at least, not for us! Although this is officially the second edition of *Computer Architecture: A Quantitative Approach,* it is really our third book in a series that began with the first edition, continued with *Computer Organization and Design: The Hardware/Software Interface,* and brings us now to this book, one that we have largely written from scratch. Since a major goal of our efforts has been to present concepts in the context of the most recent machines, there was remarkably little from the first edition that could be preserved intact.

This said, we once again offer an enthusiastic welcome to anyone who came along with us the first time, as well as to those who are joining us now. Either way, we can promise the same quantitative approach to, and analysis of, real machines.

As with the first edition, our goal has been to describe the basic principles underlying what will be tomorrow's technological developments. For readers who are new to our series with this book, we hope to demonstrate what we stated about computer architecture in our preface to the first edition: It is not a dreary science of paper machines that will never work. No! It's a discipline of keen intellectual interest, requiring the balance of marketplace forces to cost/performance, leading to glorious failures and some notable successes.

Our primary objective in writing our first book was to change the way people learn and think about computer architecture. We feel this goal is still valid and important. The field is changing daily and must be studied with real examples and measurements on real machines, rather than simply as a collection of definitions and designs that will never need to be realized.

We have strived to produce a new edition that will continue to be as relevant for professional engineers and architects as it is for those involved in advanced computer architecture and design courses. As much as its predecessor, this edition aims to demystify computer architecture through an emphasis on cost/performance tradeoffs and good engineering design. We believe that the field has continued to mature and move toward the rigorous quantitative foundation of

long-established scientific and engineering disciplines. Our greatest satisfaction derives from the fact that the principles described in our first edition in 1990 could be applied successfully to help predict the landscape of computing technology that exists today. We hope that our new book will allow readers to apply the fundamentals for similar results as we look forward to the coming decades.

This Edition

The second edition of *Computer Architecture: A Quantitative Approach* should have been easy to write. After all, our approach hasn't changed and we sought to continue our focus on the basic principles of computer design. Unfortunately for our task as authors, though the principles haven't changed, the examples that are relevant today and the conclusions that may be drawn have changed a great deal.

Consequently, we started with a careful evaluation of the first edition content from some very helpful readers who are gratefully acknowledged in the section that follows. We also thought carefully about what we would need to do to fulfill the book's original objectives, given the current state of computing. Being pragmatists, we started with a selective view of what would need to be revised, but there was no way to avoid the fact that everything needed to be reexamined. In the end we can offer you something that is virtually new.

We created fresh drafts, we condensed, we rewrote, we revised, and we eliminated less essential discussions. We enlarged the discussion of the topics we feel are the most exciting and valuable new ideas of computer engineering and science, and we reduced the coverage of other topics. In addition, we benefited from the help of another beta class-testing.

Despite these efforts or perhaps because of them, the second edition was actually harder to write than the first. We blame our workload on society's hunger for faster computers. The microprocessors being shipped today are the most sophisticated computers ever built, and feeding an expectation of doubled performance every 18 months has meant introducing new and extremely sophisticated implementation techniques.

Perhaps most important, we did not want to produce a new edition unless we were certain that it could meet the standard of contribution that we set for ourselves in our first book. In the final cut, the basic concepts and approach remain, but they are now viewed with the added perspective of several years of progress in the field.

Topic Selection and Organization

As before, we have taken a conservative approach to topic selection, for there are many more interesting ideas in the field than can reasonably be covered in a treatment of basic principles. We have steered away from a comprehensive survey of every architecture a reader might encounter. Instead, our presentation focuses on core concepts likely to be found in any new machine. The key criterion remains

that of selecting ideas that have been examined and utilized successfully enough to permit their discussion in quantitative terms.

Again, our hope is that the design principles and quantitative data in this book will delimit discussions of architecture styles to terms like *faster* and *cheaper*, in contrast to previous debates.

Our first dilemma in determining the new topic selections was that topics requiring only a few pages in the first edition have since exploded in their importance. Secondly, topics that we excluded previously have matured to a point where they can be discussed based on our quantitative criteria and their success in the marketplace.

As a final filter, we evaluated the extent of introductory material necessary for this presentation independent of our decision making for the first edition. Again, our reviewers and readers were immensely helpful. To the extent that readers felt certain discussions were less necessary, we eliminated them in favor of new material. For example, many of our readers learned the basics of our approach in the first edition or from *Computer Organization and Design: The Hardware/Software Interface.*

Our intent has always been to focus on material that is not available in equivalent form from other sources, so we continue to emphasize advanced content wherever possible. With the availability at this writing of a complete introduction in *Computer Organization and Design,* the focus on advanced topics was somewhat easier to maintain than in our first book.

An Overview of the Content

In keeping with our focus on advanced topics, we consolidated the first five chapters of our earlier edition into two. The basic content of the first edition Chapters 1 (Fundamentals of Computer Design) and 2 (Performance and Cost) are introduced in the opening chapter, Fundamentals of Computer Design. These ideas work better together than they did in isolation.

The two chapters on instruction sets from the previous edition are now presented in our second chapter, Instruction Set Principles and Examples, and its companion piece, Appendix D, An Alternative to RISC: The Intel 80x86. We rewrote this material to show how program measurements led to the design of our generic RISC machine, DLX. Because of the dominance of the 80x86 design, we no longer cover the VAX or IBM 360 instruction sets. The 80x86 appendix represents increased coverage and can form the basis of a careful study of this architecture and other computers of its class.

This de-emphasis on the history of instruction sets follows the continuing evolution of the content of computer architecture courses. In the 1950s and 1960s, course content consisted primarily of computer arithmetic. In the 1970s and early 1980s, it was largely instruction set design. Today, it is largely high-performance processor implementation, memory hierarchy, input/output, and multiprocessors.

Hence the organization of the remaining chapters and appendices.

Chapters 3 and 4 cover all aspects of instruction execution in high-performance processors: pipelining, superscalar execution, branch prediction, and dynamic scheduling. This encompasses so much material that, even with the introduction to pipelining available in *Computer Organization and Design,* we needed an intermediate and an advanced chapter.

Chapter 5 discusses the remarkable number of optimizations in cache design that have occurred over the last six years. A partial list of topics new to this edition includes victim caches, pseudo-associative caches, prefetching, cache-oriented compiler optimizations, nonblocking caches, and pipelined caches.

Chapters 6 and 7 represent the increasing importance of input/output in today's computers. Chapter 6, Storage Systems, concentrates on storage I/O, presenting trends in magnetic disks and tapes. It describes redundant arrays of inexpensive disks, or RAID, and gives a comparison of disk I/O performance for a wide range of UNIX systems. What was covered in three pages on interconnection networks in the first edition has expanded into Chapter 7. This new chapter takes the novel approach of covering in a common framework the interconnection networks found in local area networks, wide area networks, and massively parallel processors.

Chapter 8, Multiprocessors, and Appendix E, Implementing Coherence Protocols, expand the 15 pages from the first edition into an extensive discussion of the issues involved with shared-memory multiprocessors. Snooping bus protocols and directory-based protocols are fully covered, with Appendix E describing in detail why these protocols work and providing insight into some of the complexities of real coherency algorithms.

This brings us to Appendices A through C. Appendix A updates computer arithmetic, including fused multiply-add and the infamous Pentium floating-point error that was front-page news in 1994. Vector processors, covered previously as Chapter 7 of the first edition, has been revised as Appendix B in this presentation, due in part to the 100-plus pages on instruction-level parallelism in the main text and in part reflecting a decline in the popularity of vector architectures. Appendix C updates the first edition RISC appendix, replacing coverage of the Intel i860 and Motorola 88000 with that of the HP PA-RISC and IBM/Motorola PowerPC and giving the 64-bit instruction set architectures of MIPS, PowerPC, and SPARC.

There is no single best order in which to approach these chapters. We wrote the text so that it can be covered in several ways, the only real restriction being that some chapters should be read in sequence, namely, Chapters 2, 3, and 4 (pipelining) and Chapters 6 and 7 (storage systems, interconnection networks). Readers should start with Chapter 1 and read Chapter 5 (memory-hierarchy design) before Chapter 8 (multiprocessors), but the rest of the material can be covered in any order.

In summary, about 70% of the pages are new to this edition. As the second edition is also about 20% longer than the first, you can see why it was as much work as writing a new book.

Readers interested in a gentle introduction to computer architecture should read *Computer Organization and Design: The Hardware/Software Interface*. If you already have the first edition of *Computer Architecture* and are not interested in advanced processor design, advanced cache design, storage systems, interconnection networks, multiprocessors, 64-bit RISC architectures, or the 80x86, then you can save both money and space on your bookshelf. Obviously, we think owners of the first edition should be interested in many of these topics or we wouldn't have written about them!

Chapter Structure and Exercises

The material we have selected has been stretched upon a consistent framework that is followed in each chapter. We start by explaining the ideas of a chapter. These ideas are followed by a Crosscutting Issues section, a feature new to the second edition that shows how the ideas covered in one chapter interact with those given in other chapters. This is followed by a Putting It All Together section that ties these ideas together by showing how they are used in a real machine.

Next in the sequence is Fallacies and Pitfalls, which lets readers learn from the mistakes of others. We show examples of common misunderstandings and architectural traps that are difficult to avoid even when you know they are lying in wait for you. Each chapter ends with a Concluding Remarks section, followed by a Historical Perspective and References section that attempts to give proper credit for the ideas in the chapter and a sense of the history surrounding the inventions. We like to think of this as presenting the human drama of computer design. It also supplies references that the student of architecture may want to pursue. If you have time, we recommend reading some of the classic papers in the field that are mentioned in these sections. It is both enjoyable and educational to hear the ideas directly from the creators.

Each chapter ends with exercises, over 200 in total, which vary from one-minute reviews to term projects. Brackets for each exercise (<chapter.section>) indicate the text sections of primary relevance to answering the question. We hope this helps readers to avoid exercises for which they haven't read the corresponding section, in addition to providing the source for review. We also rate the exercises, estimating the amount of time a problem might take:

[10] 1 minute (read and understand)

[20] 15 to 20 minutes for full answer

[25] 1 hour for full written answer

[30] Short programming project: less than 1 full day of programming

[40] Significant programming project: 2 weeks of elapsed time

[50] Term project (2 to 4 weeks by two people)

[Discussion] Topic for discussion with others interested in computer architecture

Supplements

An Instructor's Manual with fully worked-out solutions to the exercises in the book is available from the publisher only to instructors teaching from this book.

If you are not an instructor and would like access to solutions, *Solutions to Selected Exercises in Computer Architecture: A Quantitative Approach, Second Edition* is available for sale from the publisher. It contains solutions to 81 of the exercises in this book. Software to accompany this book, including DLX compilers, assemblers, and simulators, cache simulation tools, and traces, is available to readers at the Morgan Kaufmann home page on the World Wide Web at http://www.mkp.com. This page also contains a list of errata, postscript versions of the numbered figures in the book, and pointers to related material that readers may enjoy. In response to your continued support, the publisher will add new materials and establish links to other sites on a regular basis.

Finally, it is possible to make money while reading this book. Talk about cost/performance! If you read the Acknowledgments that follow, you will see that we went to great lengths to correct mistakes. Since a book goes through many printings, we have the opportunity to make even more corrections. If you uncover any remaining resilient bugs, please contact the publisher by electronic mail (arc2bugs@mkp.com). The first reader to report an error with a fix that we incorporate in a future printing will be rewarded with a $1.00 bounty. Please check the errata sheet on the home page (http://www.mkp.com) to see if the bug has already been reported. We process the bugs and send the checks about once a year, so please be patient.

Concluding Remarks

Once again this book is a true co-authorship, with each of us writing half the chapters and half the appendices. We can't imagine how long it would have taken without someone else doing half the work, offering inspiration when the task seemed hopeless, providing the key insight to explain a difficult concept, supplying reviews over the weekend of 100-page chapters, and commiserating when the weight of our other obligations made it hard to pick up the pen. Thus, once again we share equally the blame for what you are about to read.

John Hennessy David Patterson