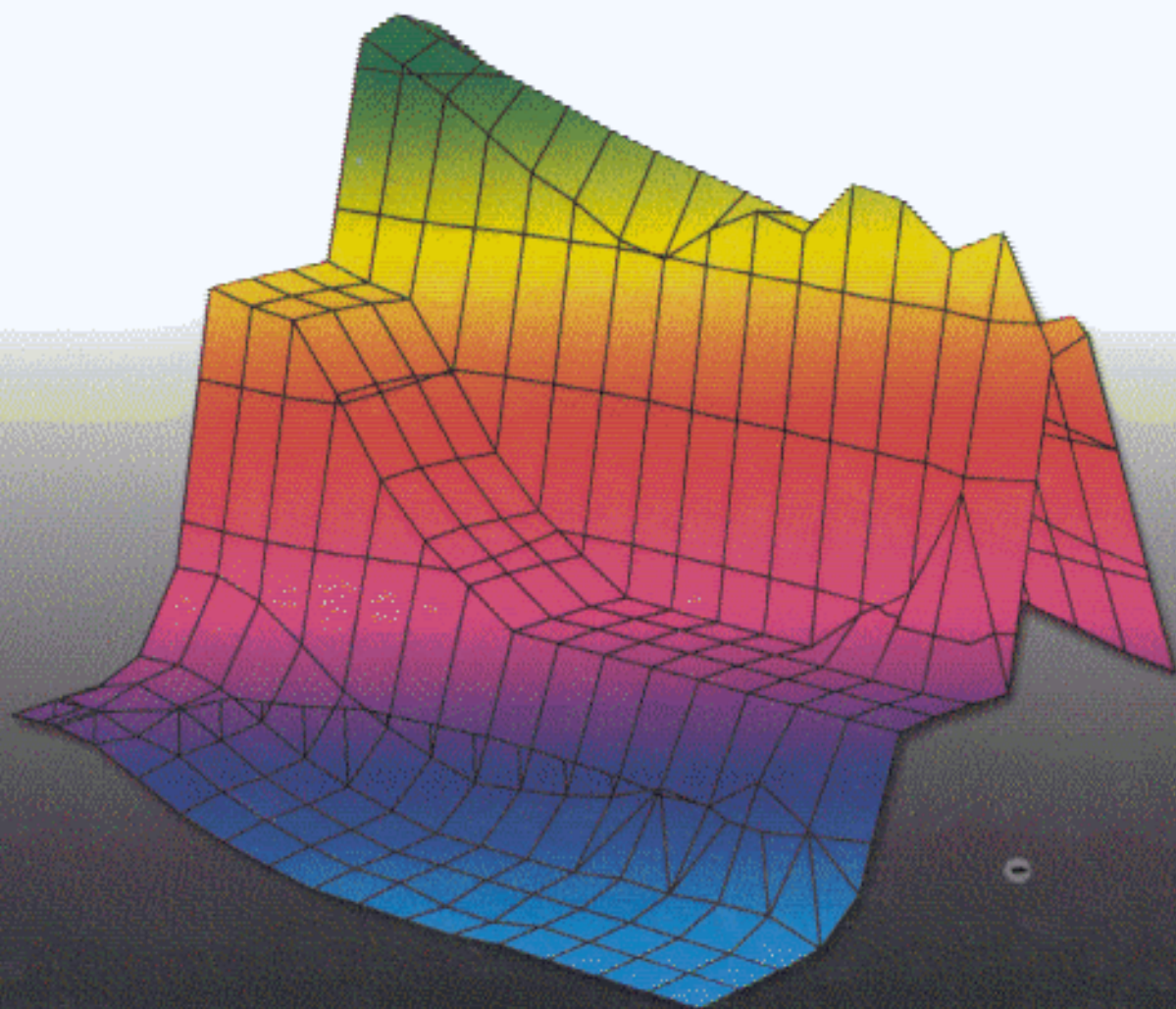


COMPUTER SYSTEMS

A PROGRAMMER'S
PERSPECTIVE



Randal E. Bryant and David O'Hallaron

Contents

Preface xvii

About the Authors xxviii

1

A Tour of Computer Systems 1

- 1.1 Information is Bits + Context 2
- 1.2 Programs Are Translated by Other Programs into Different Forms 4
- 1.3 It Pays to Understand How Compilation Systems Work 6
- 1.4 Processors Read and Interpret Instructions Stored in Memory 6
 - 1.4.1 Hardware Organization of a System 7
 - 1.4.2 Running the hello Program 9
- 1.5 Caches Matter 11
- 1.6 Storage Devices Form a Hierarchy 12
- 1.7 The Operating System Manages the Hardware 13
 - 1.7.1 Processes 15
 - 1.7.2 Threads 16
 - 1.7.3 Virtual Memory 16
 - 1.7.4 Files 18
- 1.8 Systems Communicate With Other Systems Using Networks 18
- 1.9 The Next Step 20
- 1.10 Summary 20
 - Bibliographics Notes 21

Part I Program Structure and Execution

2

Representing and Manipulating Information 24

- 2.1 Information Storage 28
 - 2.1.1 Hexadecimal Notation 28
 - 2.1.2 Words 32
 - 2.1.3 Data Sizes 32
 - 2.1.4 Addressing and Byte Ordering 34
 - 2.1.5 Representing Strings 40
 - 2.1.6 Representing Code 41
 - 2.1.7 Boolean Algebras and Rings 42

	2.1.8	Bit-Level Operations in C	46
	2.1.9	Logical Operations in C	49
	2.1.10	Shift Operations in C	50
2.2		Integer Representations	51
	2.2.1	Integral Data Types	51
	2.2.2	Unsigned and Two's-Complement Encodings	51
	2.2.3	Conversions Between Signed and Unsigned	56
	2.2.4	Signed vs. Unsigned in C	59
	2.2.5	Expanding the Bit Representation of a Number	61
	2.2.6	Truncating Numbers	63
	2.2.7	Advice on Signed vs. Unsigned	65
2.3		Integer Arithmetic	65
	2.3.1	Unsigned Addition	66
	2.3.2	Two's-Complement Addition	69
	2.3.3	Two's-Complement Negation	72
	2.3.4	Unsigned Multiplication	74
	2.3.5	Two's-Complement Multiplication	75
	2.3.6	Multiplying by Powers of Two	76
	2.3.7	Dividing by Powers of Two	77
2.4		Floating Point	80
	2.4.1	Fractional Binary Numbers	81
	2.4.2	IEEE Floating-Point Representation	83
	2.4.3	Example Numbers	85
	2.4.4	Rounding	89
	2.4.5	Floating-Point Operations	91
	2.4.6	Floating Point in C	92
2.5		Summary	98
		Bibliographic Notes	99
		Homework Problems	99
		Solution to Practice Problems	108

3

Machine-Level Representation of Programs 122

3.1	A Historical Perspective	125
3.2	Program Encodings	128
	3.2.1	Machine-Level Code 129
	3.2.2	Code Examples 130
	3.2.3	A Note on Formatting 133
3.3	Data Formats	135
3.4	Accessing Information	136
	3.4.1	Operand Specifiers 137
	3.4.2	Data Movement Instructions 138

- 3.4.3 Data Movement Example 141
- 3.5 Arithmetic and Logical Operations 143
 - 3.5.1 Load Effective Address 143
 - 3.5.2 Unary and Binary Operations 144
 - 3.5.3 Shift Operations 145
 - 3.5.4 Discussion 146
 - 3.5.5 Special Arithmetic Operations 147
- 3.6 Control 148
 - 3.6.1 Condition Codes 149
 - 3.6.2 Accessing the Condition Codes 150
 - 3.6.3 Jump Instructions and their Encodings 152
 - 3.6.4 Translating Conditional Branches 156
 - 3.6.5 Loops 158
 - 3.6.6 Switch Statements 166
- 3.7 Procedures 170
 - 3.7.1 Stack Frame Structure 170
 - 3.7.2 Transferring Control 172
 - 3.7.3 Register Usage Conventions 173
 - 3.7.4 Procedure Example 174
 - 3.7.5 Recursive Procedures 178
- 3.8 Array Allocation and Access 180
 - 3.8.1 Basic Principles 180
 - 3.8.2 Pointer Arithmetic 182
 - 3.8.3 Arrays and Loops 183
 - 3.8.4 Nested Arrays 183
 - 3.8.5 Fixed Size Arrays 186
 - 3.8.6 Dynamically Allocated Arrays 188
- 3.9 Heterogeneous Data Structures 191
 - 3.9.1 Structures 191
 - 3.9.2 Unions 194
- 3.10 Alignment 198
- 3.11 Putting it Together: Understanding Pointers 201
- 3.12 Life in the Real World: Using the GDB Debugger 204
- 3.13 Out-of-Bounds Memory References and Buffer Overflow 206
- 3.14 *Floating-Point Code 211
 - 3.14.1 Floating-Point Registers 211
 - 3.14.2 Stack Evaluation of Expressions 212
 - 3.14.3 Floating-Point Data Movement and Conversion Operations 215
 - 3.14.4 Floating-Point Arithmetic Instructions 217
 - 3.14.5 Using Floating Point in Procedures 220
 - 3.14.6 Testing and Comparing Floating-Point Values 221
- 3.15 *Embedding Assembly Code in C Programs 223

- 3.15.1 Basic Inline Assembly 224
- 3.15.2 Extended Form of asm 226
- 3.16 Summary 230
 - Bibliographic Notes 231
 - Homework Problems 231
 - Solutions to Practice Problems 238

4

Processor Architecture 254

- 4.1 The Y86 Instruction Set Architecture 258
- 4.2 Logic Design and the Hardware Control Language HCL 271
 - 4.2.1 Logic Gates 271
 - 4.2.2 Combinational Circuits and HCL Boolean Expressions 272
 - 4.2.3 Word-Level Combinational Circuits and HCL Integer Expressions 274
 - 4.2.4 Set Membership 278
 - 4.2.5 Memory and Clocking 279
- 4.3 Sequential Y86 Implementations 280
 - 4.3.1 Organizing Processing into Stages 281
 - 4.3.2 SEQ Hardware Structure 291
 - 4.3.3 SEQ Timing 295
 - 4.3.4 SEQ Stage Implementations 298
 - 4.3.5 SEQ+: Rearranging the Computation Stages 305
- 4.4 General Principles of Pipelining 309
 - 4.4.1 Computational Pipelines 309
 - 4.4.2 A Detailed Look at Pipeline Operation 311
 - 4.4.3 Limitations of Pipelining 313
 - 4.4.4 Pipelining a System with Feedback 315
- 4.5 Pipelined Y86 Implementations 317
 - 4.5.1 Inserting Pipeline Registers 317
 - 4.5.2 Rearranging and Relabeling Signals 321
 - 4.5.3 Next PC Prediction 322
 - 4.5.4 Pipeline Hazards 323
 - 4.5.5 Avoiding Data Hazards by Stalling 328
 - 4.5.6 Avoiding Data Hazards by Forwarding 330
 - 4.5.7 Load/Use Data Hazards 335
 - 4.5.8 PIPE Stage Implementations 337
 - 4.5.9 Pipeline Control Logic 343
 - 4.5.10 Performance Analysis 352
 - 4.5.11 Unfinished Business 354
- 4.6 Summary 359
 - 4.6.1 Y86 Simulators 360

Bibliographic Notes	360
Homework Problems	360
Solutions to Practice Problems	365

5

Optimizing Program Performance 376

5.1	Capabilities and Limitations of Optimizing Compilers	379
5.2	Expressing Program Performance	382
5.3	Program Example	384
5.4	Eliminating Loop Inefficiencies	387
5.5	Reducing Procedure Calls	391
5.6	Eliminating Unneeded Memory References	393
5.7	Understanding Modern Processors	395
5.7.1	Overall Operation	395
5.7.2	Functional Unit Performance	399
5.7.3	A Closer Look at Processor Operation	400
5.8	Reducing Loop Overhead	408
5.9	Converting to Pointer Code	412
5.10	Enhancing Parallelism	415
5.10.1	Loop Splitting	415
5.10.2	Register Spilling	420
5.10.3	Limits to Parallelism	421
5.11	Putting it Together: Summary of Results for Optimizing Combining Code	423
5.11.1	Floating-Point Performance Anomaly	423
5.11.2	Changing Platforms	425
5.12	Branch Prediction and Misprediction Penalties	425
5.13	Understanding Memory Performance	429
5.13.1	Load Latency	429
5.13.2	Store Latency	431
5.14	Life in the Real World: Performance Improvement Techniques	436
5.15	Identifying and Eliminating Performance Bottlenecks	437
5.15.1	Program Profiling	437
5.15.2	Using a Profiler to Guide Optimization	439
5.15.3	Amdahl's Law	443
5.16	Summary	444
	Bibliographic Notes	445
	Homework Problems	445
	Solutions to Practice Problems	450

6

The Memory Hierarchy 454

- 6.1 Storage Technologies 457
 - 6.1.1 Random-Access Memory 457
 - 6.1.2 Disk Storage 464
 - 6.1.3 Storage Technology Trends 476
- 6.2 Locality 478
 - 6.2.1 Locality of References to Program Data 478
 - 6.2.2 Locality of Instruction Fetches 480
 - 6.2.3 Summary of Locality 481
- 6.3 The Memory Hierarchy 482
 - 6.3.1 Caching in the Memory Hierarchy 484
 - 6.3.2 Summary of Memory Hierarchy Concepts 486
- 6.4 Cache Memories 487
 - 6.4.1 Generic Cache Memory Organization 488
 - 6.4.2 Direct-Mapped Caches 490
 - 6.4.3 Set Associative Caches 497
 - 6.4.4 Fully Associative Caches 499
 - 6.4.5 Issues with Writes 503
 - 6.4.6 Instruction Caches and Unified Caches 504
 - 6.4.7 Performance Impact of Cache Parameters 505
- 6.5 Writing Cache-Friendly Code 507
- 6.6 Putting it Together: The Impact of Caches on Program Performance 511
 - 6.6.1 The Memory Mountain 512
 - 6.6.2 Rearranging Loops to Increase Spatial Locality 517
 - 6.6.3 Using Blocking to Increase Temporal Locality 520
- 6.7 Putting It Together: Exploiting Locality in Your Programs 523
- 6.8 Summary 524
 - Bibliographic Notes 524
 - Homework Problems 525
 - Solutions to Practice Problems 531

Part II Running Programs on a System

7

Linking 538

- 7.1 Compiler Drivers 541
- 7.2 Static Linking 542
- 7.3 Object Files 543
- 7.4 Relocatable Object Files 544

- 7.5 Symbols and Symbol Tables 545
- 7.6 Symbol Resolution 548
 - 7.6.1 How Linkers Resolve Multiply Defined Global Symbols 549
 - 7.6.2 Linking with Static Libraries 553
 - 7.6.3 How Linkers Use Static Libraries to Resolve References 556
- 7.7 Relocation 557
 - 7.7.1 Relocation Entries 558
 - 7.7.2 Relocating Symbol References 558
- 7.8 Executable Object Files 561
- 7.9 Loading Executable Object Files 564
- 7.10 Dynamic Linking with Shared Libraries 566
- 7.11 Loading and Linking Shared Libraries from Applications 568
- 7.12 *Position-Independent Code (PIC) 570
 - 7.12.1 PIC Data References 572
 - 7.12.2 PIC Function Calls 572
- 7.13 Tools for Manipulating Object Files 574
- 7.14 Summary 575
 - Bibliographic Notes 575
 - Homework Problems 576
 - Solutions to Practice Problems 582

8

Exceptional Control Flow 584

- 8.1 Exceptions 587
 - 8.1.1 Exception Handling 588
 - 8.1.2 Classes of Exceptions 590
 - 8.1.3 Exceptions in Intel Processors 592
- 8.2 Processes 594
 - 8.2.1 Logical Control Flow 594
 - 8.2.2 Private Address Space 595
 - 8.2.3 User and Kernel Modes 596
 - 8.2.4 Context Switches 597
- 8.3 System Calls and Error Handling 599
- 8.4 Process Control 600
 - 8.4.1 Obtaining Process ID's 600
 - 8.4.2 Creating and Terminating Processes 600
 - 8.4.3 Reaping Child Processes 605
 - 8.4.4 Putting Processes to Sleep 610
 - 8.4.5 Loading and Running Programs 611
 - 8.4.6 Using `fork` and `execve` to Run Programs 614
- 8.5 Signals 617
 - 8.5.1 Signal Terminology 617

- 8.5.2 Sending Signals 619
- 8.5.3 Receiving Signals 623
- 8.5.4 Signal Handling Issues 625
- 8.5.5 Portable Signal Handling 631
- 8.5.6 Explicitly Blocking Signals 633
- 8.6 Nonlocal Jumps 635
- 8.7 Tools for Manipulating Processes 638
- 8.8 Summary 638
 - Bibliographic Notes 639
 - Homework Problems 639
 - Solutions to Practice Problems 645

9

Measuring Program Execution Time 650

- 9.1 The Flow of Time on a Computer System 653
 - 9.1.1 Process Scheduling and Timer Interrupts 654
 - 9.1.2 Time from an Application Program's Perspective 655
- 9.2 Measuring Time by Interval Counting 658
 - 9.2.1 Operation 658
 - 9.2.2 Reading the Process Timers 659
 - 9.2.3 Accuracy of Process Timers 660
- 9.3 Cycle Counters 663
 - 9.3.1 IA32 Cycle Counters 663
- 9.4 Measuring Program Execution Time with Cycle Counters 665
 - 9.4.1 The Effects of Context Switching 665
 - 9.4.2 Caching and Other Effects 667
 - 9.4.3 The K -Best Measurement Scheme 671
- 9.5 Time-of-Day Measurements 680
- 9.6 Putting it Together: An Experimental Protocol 683
- 9.7 Looking into the Future 684
- 9.8 Life in the Real World: An Implementation of the K -Best Measurement Scheme 684
- 9.9 Lessons Learned 685
- 9.10 Summary 686
 - Bibliographic Notes 686
 - Homework Problems 687
 - Solutions to Practice Problems 688

10

Virtual Memory 690

- 10.1 Physical and Virtual Addressing 693

- 10.2** Address Spaces 694
- 10.3** VM as a Tool for Caching 695
 - 10.3.1** DRAM Cache Organization 696
 - 10.3.2** Page Tables 696
 - 10.3.3** Page Hits 698
 - 10.3.4** Page Faults 698
 - 10.3.5** Allocating Pages 700
 - 10.3.6** Locality to the Rescue Again 700
- 10.4** VM as a Tool for Memory Management 701
 - 10.4.1** Simplifying Linking 701
 - 10.4.2** Simplifying Sharing 702
 - 10.4.3** Simplifying Memory Allocation 702
 - 10.4.4** Simplifying Loading 703
- 10.5** VM as a Tool for Memory Protection 703
- 10.6** Address Translation 704
 - 10.6.1** Integrating Caches and VM 707
 - 10.6.2** Speeding up Address Translation with a TLB 707
 - 10.6.3** Multi-Level Page Tables 709
 - 10.6.4** Putting it Together: End-to-End Address Translation 711
- 10.7** Case Study: The Pentium/Linux Memory System 715
 - 10.7.1** Pentium Address Translation 716
 - 10.7.2** Linux Virtual Memory System 721
- 10.8** Memory Mapping 724
 - 10.8.1** Shared Objects Revisited 725
 - 10.8.2** The `fork` Function Revisited 727
 - 10.8.3** The `execve` Function Revisited 727
 - 10.8.4** User-Level Memory Mapping with the `mmap` Function 728
- 10.9** Dynamic Memory Allocation 730
 - 10.9.1** The `malloc` and `free` Functions 731
 - 10.9.2** Why Dynamic Memory Allocation? 733
 - 10.9.3** Allocator Requirements and Goals 735
 - 10.9.4** Fragmentation 736
 - 10.9.5** Implementation Issues 737
 - 10.9.6** Implicit Free Lists 737
 - 10.9.7** Placing Allocated Blocks 739
 - 10.9.8** Splitting Free Blocks 740
 - 10.9.9** Getting Additional Heap Memory 740
 - 10.9.10** Coalescing Free Blocks 741
 - 10.9.11** Coalescing with Boundary Tags 741
 - 10.9.12** Putting it Together: Implementing a Simple Allocator 744
 - 10.9.13** Explicit Free Lists 751
 - 10.9.14** Segregated Free Lists 752

- 10.10** Garbage Collection 755
 - 10.10.1** Garbage Collector Basics 756
 - 10.10.2** Mark&Sweep Garbage Collectors 757
 - 10.10.3** Conservative Mark&Sweep for C Programs 758
- 10.11** Common Memory-Related Bugs in C Programs 759
 - 10.11.1** Dereferencing Bad Pointers 759
 - 10.11.2** Reading Uninitialized Memory 760
 - 10.11.3** Allowing Stack Buffer Overflows 760
 - 10.11.4** Assuming that Pointers and the Objects they Point to Are the Same Size 761
 - 10.11.5** Making Off-by-One Errors 761
 - 10.11.6** Referencing a Pointer Instead of the Object it Points to 762
 - 10.11.7** Misunderstanding Pointer Arithmetic 762
 - 10.11.8** Referencing Nonexistent Variables 763
 - 10.11.9** Referencing Data in Free Heap Blocks 763
 - 10.11.10** Introducing Memory Leaks 764
- 10.12** Recapping Some Key Ideas About Virtual Memory 764
- 10.13** Summary 764
 - Bibliographic Notes 765
 - Homework Problems 766
 - Solutions to Practice Problems 770

Part III Interaction and Communication Between Programs

11

System-Level I/O 776

- 11.1** Unix I/O 778
- 11.2** Opening and Closing Files 779
- 11.3** Reading and Writing Files 781
- 11.4** Robust Reading and Writing with the RIo Package 783
 - 11.4.1** RIo Unbuffered Input and Output Functions 783
 - 11.4.2** RIo Buffered Input Functions 784
- 11.5** Reading File Metadata 789
- 11.6** Sharing Files 791
- 11.7** I/O Redirection 793
- 11.8** Standard I/O 795
- 11.9** Putting It Together: Which I/O Functions Should I Use? 796
- 11.10** Summary 797
 - Bibliographic Notes 798
 - Homework Problems 798

12

Network Programming 800

- 12.1** The Client–Server Programming Model 802
- 12.2** Networks 803
- 12.3** The Global IP Internet 807
 - 12.3.1** IP Addresses 809
 - 12.3.2** Internet Domain Names 811
 - 12.3.3** Internet Connections 815
- 12.4** The Sockets Interface 816
 - 12.4.1** Socket Address Structures 817
 - 12.4.2** The socket Function 818
 - 12.4.3** The connect Function 818
 - 12.4.4** The open_clientfd Function 819
 - 12.4.5** The bind Function 819
 - 12.4.6** The listen Function 820
 - 12.4.7** The open_listenfd Function 821
 - 12.4.8** The accept Function 821
 - 12.4.9** Example Echo Client and Server 823
- 12.5** Web Servers 826
 - 12.5.1** Web Basics 826
 - 12.5.2** Web Content 827
 - 12.5.3** HTTP Transactions 828
 - 12.5.4** Serving Dynamic Content 831
- 12.6** Putting it Together: The TINY Web Server 834
- 12.7** Summary 841
 - Bibliographic Notes 842
 - Homework Problems 842
 - Solutions to Practice Problems 843

13

Concurrent Programming 846

- 13.1** Concurrent Programming With Processes 849
 - 13.1.1** A Concurrent Server Based on Processes 851
 - 13.1.2** Pros and Cons of Processes 851
- 13.2** Concurrent Programming With I/O Multiplexing 853
 - 13.2.1** A Concurrent Event-Driven Server Based on I/O Multiplexing 856
 - 13.2.2** Pros and Cons of I/O Multiplexing 860
- 13.3** Concurrent Programming With Threads 861
 - 13.3.1** Thread Execution Model 862

	13.3.2	Posix Threads	863
	13.3.3	Creating Threads	864
	13.3.4	Terminating Threads	864
	13.3.5	Reaping Terminated Threads	865
	13.3.6	Detaching Threads	865
	13.3.7	Initializing Threads	866
	13.3.8	A Concurrent Server Based on Threads	866
13.4		Shared Variables in Threaded Programs	868
	13.4.1	Threads Memory Model	869
	13.4.2	Mapping Variables to Memory	870
	13.4.3	Shared Variables	870
13.5		Synchronizing Threads with Semaphores	871
	13.5.1	Progress Graphs	874
	13.5.2	Using Semaphores to Access Shared Variables	877
	13.5.3	Posix Semaphores	878
	13.5.4	Using Semaphores to Schedule Shared Resources	879
13.6		Putting It Together: A Concurrent Server Based on Prethreading	882
13.7		Other Concurrency Issues	885
	13.7.1	Thread Safety	885
	13.7.2	Reentrancy	888
	13.7.3	Using Existing Library Functions in Threaded Programs	889
	13.7.4	Races	890
	13.7.5	Deadlocks	891
13.8		Summary	894
		Bibliographic Notes	895
		Homework Problems	895
		Solutions to Practice Problems	899

A

HCL Descriptions of Processor Control Logic	905
---	-----

B

Error Handling	925
----------------	-----

Bibliography	949
--------------	-----

Index	953
-------	-----