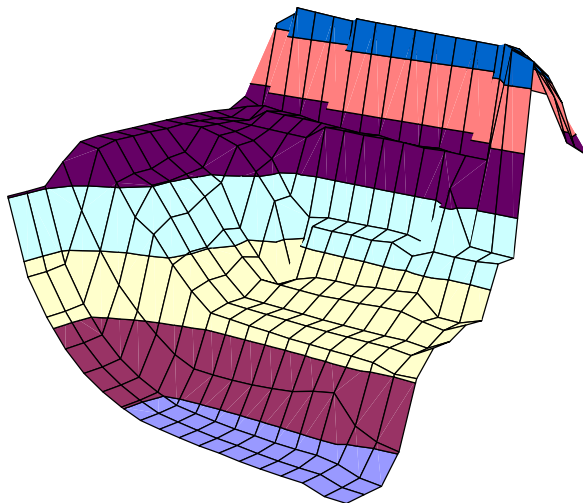


Computer Systems
*A Programmer's Perspective, Second Edition*¹



Randal E. Bryant
David R. O'Hallaron

January 13, 2010

¹Copyright © 2010, R. E. Bryant, D. R. O'Hallaron. All rights reserved.

Contents

Preface	xv
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	3
1.3 It Pays to Understand How Compilation Systems Work	5
1.4 Processors Read and Interpret Instructions Stored in Memory	6
1.4.1 Hardware Organization of a System	6
1.4.2 Running the <code>hello</code> Program	8
1.5 Caches Matter	9
1.6 Storage Devices Form a Hierarchy	11
1.7 The Operating System Manages the Hardware	12
1.7.1 Processes	13
1.7.2 Threads	15
1.7.3 Virtual Memory	15
1.7.4 Files	16
1.8 Systems Communicate with Other Systems Using Networks	17
1.9 Important Themes	19
1.9.1 Concurrency and Parallelism	19
1.9.2 The Importance of Abstractions in Computer Systems	22
1.10 Summary	23
I Program Structure and Execution	25
2 Representing and Manipulating Information	29

2.1	Information Storage	32
2.1.1	Hexadecimal Notation	32
2.1.2	Words	36
2.1.3	Data Sizes	36
2.1.4	Addressing and Byte Ordering	37
2.1.5	Representing Strings	43
2.1.6	Representing Code	44
2.1.7	Introduction to Boolean Algebra	45
2.1.8	Bit-Level Operations in C	47
2.1.9	Logical Operations in C	50
2.1.10	Shift Operations in C	51
2.2	Integer Representations	52
2.2.1	Integral Data Types	52
2.2.2	Unsigned Encodings	54
2.2.3	Two's-Complement Encodings	55
2.2.4	Conversions Between Signed and Unsigned	60
2.2.5	Signed vs. Unsigned in C	64
2.2.6	Expanding the Bit Representation of a Number	66
2.2.7	Truncating Numbers	70
2.2.8	Advice on Signed vs. Unsigned	71
2.3	Integer Arithmetic	74
2.3.1	Unsigned Addition	74
2.3.2	Two's-Complement Addition	78
2.3.3	Two's-Complement Negation	82
2.3.4	Unsigned Multiplication	83
2.3.5	Two's-Complement Multiplication	83
2.3.6	Multiplying by Constants	87
2.3.7	Dividing by Powers of Two	89
2.3.8	Final Thoughts on Integer Arithmetic	92
2.4	Floating Point	93
2.4.1	Fractional Binary Numbers	94
2.4.2	IEEE Floating-Point Representation	96
2.4.3	Example Numbers	99

2.4.4	Rounding	104
2.4.5	Floating-Point Operations	106
2.4.6	Floating Point in C	107
2.5	Summary	110
3	Machine-Level Representation of Programs	145
3.1	A Historical Perspective	147
3.2	Program Encodings	150
3.2.1	Machine-Level Code	151
3.2.2	Code Examples	152
3.2.3	Notes on Formatting	155
3.3	Data Formats	157
3.4	Accessing Information	158
3.4.1	Operand Specifiers	159
3.4.2	Data Movement Instructions	160
3.4.3	Data Movement Example	164
3.5	Arithmetic and Logical Operations	166
3.5.1	Load Effective Address	167
3.5.2	Unary and Binary Operations	168
3.5.3	Shift Operations	168
3.5.4	Discussion	169
3.5.5	Special Arithmetic Operations	171
3.6	Control	173
3.6.1	Condition Codes	174
3.6.2	Accessing the Condition Codes	175
3.6.3	Jump Instructions and Their Encodings	177
3.6.4	Translating Conditional Branches	181
3.6.5	Loops	184
3.6.6	Conditional Move Instructions	195
3.6.7	Switch Statements	201
3.7	Procedures	207
3.7.1	Stack Frame Structure	207
3.7.2	Transferring Control	209

3.7.3	Register Usage Conventions	211
3.7.4	Procedure Example	212
3.7.5	Recursive Procedures	217
3.8	Array Allocation and Access	219
3.8.1	Basic Principles	219
3.8.2	Pointer Arithmetic	221
3.8.3	Nested Arrays	222
3.8.4	Fixed-Size Arrays	223
3.8.5	Variable-Size Arrays	225
3.9	Heterogeneous Data Structures	227
3.9.1	Structures	227
3.9.2	Unions	231
3.9.3	Data Alignment	234
3.10	Putting It Together: Understanding Pointers	237
3.11	Life in the Real World: Using the GDB Debugger	239
3.12	Out-of-Bounds Memory References and Buffer Overflow	241
3.12.1	Thwarting Buffer Overflow Attacks	246
3.13	x86-64: Extending IA32 to 64 Bits	251
3.13.1	History and Motivation for x86-64	252
3.13.2	An Overview of x86-64	254
3.13.3	Accessing Information	257
3.13.4	Control	263
3.13.5	Data Structures	273
3.13.6	Concluding Observations about x86-64	274
3.14	Machine-Level Representations of Floating-Point Programs	274
3.15	Summary	275
4	Processor Architecture	317
4.1	The Y86 Instruction Set Architecture	319
4.1.1	Programmer-Visible State	320
4.1.2	Y86 Instructions	320
4.1.3	Instruction Encoding	322
4.1.4	Y86 Exceptions	327

4.1.5	Y86 Programs	328
4.1.6	Some Y86 Instruction Details	332
4.2	Logic Design and the Hardware Control Language HCL	334
4.2.1	Logic Gates	334
4.2.2	Combinational Circuits and HCL Boolean Expressions	335
4.2.3	Word-Level Combinational Circuits and HCL Integer Expressions	337
4.2.4	Set Membership	341
4.2.5	Memory and Clocking	342
4.3	Sequential Y86 Implementations	344
4.3.1	Organizing Processing into Stages	344
4.3.2	SEQ Hardware Structure	354
4.3.3	SEQ Timing	358
4.3.4	SEQ Stage Implementations	361
4.4	General Principles of Pipelining	369
4.4.1	Computational Pipelines	369
4.4.2	A Detailed Look at Pipeline Operation	371
4.4.3	Limitations of Pipelining	373
4.4.4	Pipelining a System with Feedback	375
4.5	Pipelined Y86 Implementations	376
4.5.1	SEQ+: Rearranging the Computation Stages	376
4.5.2	Inserting Pipeline Registers	381
4.5.3	Rearranging and Relabeling Signals	382
4.5.4	Next PC Prediction	383
4.5.5	Pipeline Hazards	385
4.5.6	Avoiding Data Hazards by Stalling	390
4.5.7	Avoiding Data Hazards by Forwarding	392
4.5.8	Load/Use Data Hazards	396
4.5.9	Exception Handling	400
4.5.10	PIPE Stage Implementations	402
4.5.11	Pipeline Control Logic	409
4.5.12	Performance Analysis	421
4.5.13	Unfinished Business	423
4.6	Summary	426

4.6.1	Y86 Simulators	427
5	Optimizing Program Performance	449
5.1	Capabilities and Limitations of Optimizing Compilers	451
5.2	Expressing Program Performance	454
5.3	Program Example	457
5.4	Eliminating Loop Inefficiencies	460
5.5	Reducing Procedure Calls	466
5.6	Eliminating Unneeded Memory References	466
5.7	Understanding Modern Processors	470
5.7.1	Overall Operation	471
5.7.2	Functional Unit Performance	474
5.7.3	An Abstract Model of Processor Operation	476
5.8	Loop Unrolling	481
5.9	Enhancing Parallelism	486
5.9.1	Multiple Accumulators	486
5.9.2	Reassociation Transformation	491
5.10	Summary of Results for Optimizing Combining Code	497
5.11	Some Limiting Factors	498
5.11.1	Register Spilling	499
5.11.2	Branch Prediction and Misprediction Penalties	500
5.12	Understanding Memory Performance	503
5.12.1	Load Performance	504
5.12.2	Store Performance	505
5.13	Life in the Real World: Performance Improvement Techniques	511
5.14	Identifying and Eliminating Performance Bottlenecks	512
5.14.1	Program Profiling	512
5.14.2	Using a Profiler to Guide Optimization	514
5.14.3	Amdahl's Law	518
5.15	Summary	519
6	The Memory Hierarchy	531
6.1	Storage Technologies	532

6.1.1	Random-Access Memory	532
6.1.2	Disk Storage	539
6.1.3	Solid State Disks	551
6.1.4	Storage Technology Trends	553
6.2	Locality	556
6.2.1	Locality of References to Program Data	556
6.2.2	Locality of Instruction Fetches	558
6.2.3	Summary of Locality	558
6.3	The Memory Hierarchy	559
6.3.1	Caching in the Memory Hierarchy	562
6.3.2	Summary of Memory Hierarchy Concepts	565
6.4	Cache Memories	565
6.4.1	Generic Cache Memory Organization	566
6.4.2	Direct-Mapped Caches	568
6.4.3	Set Associative Caches	575
6.4.4	Fully Associative Caches	578
6.4.5	Issues with Writes	581
6.4.6	Anatomy of a Real Cache Hierarchy	581
6.4.7	Performance Impact of Cache Parameters	582
6.5	Writing Cache-friendly Code	584
6.6	Putting it Together: The Impact of Caches on Program Performance	589
6.6.1	The Memory Mountain	589
6.6.2	Rearranging Loops to Increase Spatial Locality	593
6.6.3	Exploiting Locality in Your Programs	597
6.7	Summary	598
 II Running Programs on a System		 619
 7 Linking		 623
7.1	Compiler Drivers	624
7.2	Static Linking	625
7.3	Object Files	626
7.4	Relocatable Object Files	627

7.5	Symbols and Symbol Tables	628
7.6	Symbol Resolution	631
7.6.1	How Linkers Resolve Multiply Defined Global Symbols	632
7.6.2	Linking with Static Libraries	635
7.6.3	How Linkers Use Static Libraries to Resolve References	638
7.7	Relocation	640
7.7.1	Relocation Entries	640
7.7.2	Relocating Symbol References	641
7.8	Executable Object Files	645
7.9	Loading Executable Object Files	646
7.10	Dynamic Linking with Shared Libraries	648
7.11	Loading and Linking Shared Libraries from Applications	650
7.12	*Position-Independent Code (PIC)	653
7.13	Tools for Manipulating Object Files	656
7.14	Summary	656
8	Exceptional Control Flow	667
8.1	Exceptions	668
8.1.1	Exception Handling	670
8.1.2	Classes of Exceptions	671
8.1.3	Exceptions in Linux/IA32 Systems	673
8.2	Processes	677
8.2.1	Logical Control Flow	677
8.2.2	Concurrent Flows	678
8.2.3	Private Address Space	679
8.2.4	User and Kernel Modes	679
8.2.5	Context Switches	681
8.3	System Call Error Handling	682
8.4	Process Control	683
8.4.1	Obtaining Process IDs	683
8.4.2	Creating and Terminating Processes	684
8.4.3	Reaping Child Processes	688
8.4.4	Putting Processes to Sleep	694

8.4.5	Loading and Running Programs	695
8.4.6	Using <code>fork</code> and <code>execve</code> to Run Programs	697
8.5	Signals	701
8.5.1	Signal Terminology	701
8.5.2	Sending Signals	703
8.5.3	Receiving Signals	708
8.5.4	Signal Handling Issues	710
8.5.5	Portable Signal Handling	716
8.5.6	Explicitly Blocking and Unblocking Signals	717
8.5.7	Synchronizing Flows to Avoid Nasty Concurrency Bugs	719
8.6	Nonlocal Jumps	721
8.7	Tools for Manipulating Processes	727
8.8	Summary	727
9	Virtual Memory	741
9.1	Physical and Virtual Addressing	742
9.2	Address Spaces	743
9.3	VM as a Tool for Caching	744
9.3.1	DRAM Cache Organization	745
9.3.2	Page Tables	745
9.3.3	Page Hits	746
9.3.4	Page Faults	747
9.3.5	Allocating Pages	748
9.3.6	Locality to the Rescue Again	749
9.4	VM as a Tool for Memory Management	749
9.5	VM as a Tool for Memory Protection	751
9.6	Address Translation	752
9.6.1	Integrating Caches and VM	755
9.6.2	Speeding up Address Translation with a TLB	755
9.6.3	Multi Level Page Tables	756
9.6.4	Putting it Together: End-to-end Address Translation	759
9.7	Case Study: The Intel Core i7/Linux Memory System	763
9.7.1	Core i7 Address Translation	763

9.7.2	Linux Virtual Memory System	768
9.8	Memory Mapping	771
9.8.1	Shared Objects Revisited	771
9.8.2	The <code>fork</code> Function Revisited	773
9.8.3	The <code>execve</code> Function Revisited	774
9.8.4	User-level Memory Mapping with the <code>mmap</code> Function	775
9.9	Dynamic Memory Allocation	776
9.9.1	The <code>malloc</code> and <code>free</code> Functions	778
9.9.2	Why Dynamic Memory Allocation?	779
9.9.3	Allocator Requirements and Goals	781
9.9.4	Fragmentation	783
9.9.5	Implementation Issues	784
9.9.6	Implicit Free Lists	784
9.9.7	Placing Allocated Blocks	786
9.9.8	Splitting Free Blocks	786
9.9.9	Getting Additional Heap Memory	787
9.9.10	Coalescing Free Blocks	787
9.9.11	Coalescing with Boundary Tags	788
9.9.12	Putting it Together: Implementing a Simple Allocator	790
9.9.13	Explicit Free Lists	798
9.9.14	Segregated Free Lists	799
9.10	Garbage Collection	801
9.10.1	Garbage Collector Basics	802
9.10.2	Mark&Sweep Garbage Collectors	803
9.10.3	Conservative Mark&Sweep for C Programs	805
9.11	Common Memory-Related Bugs in C Programs	806
9.11.1	Dereferencing Bad Pointers	806
9.11.2	Reading Uninitialized Memory	806
9.11.3	Allowing Stack Buffer Overflows	807
9.11.4	Assuming that Pointers and the Objects they Point to Are the Same Size	807
9.11.5	Making Off-by-One Errors	808
9.11.6	Referencing a Pointer Instead of the Object it Points To	808
9.11.7	Misunderstanding Pointer Arithmetic	809

9.11.8 Referencing Nonexistent Variables	809
9.11.9 Referencing Data in Free Heap Blocks	809
9.11.10 Introducing Memory Leaks	810
9.12 Summary	810
III Interaction and Communication Between Programs	821
10 System-Level I/O	825
10.1 Unix I/O	826
10.2 Opening and Closing Files	826
10.3 Reading and Writing Files	828
10.4 Robust Reading and Writing with the RIO Package	830
10.4.1 RIO Unbuffered Input and Output Functions	830
10.4.2 RIO Buffered Input Functions	831
10.5 Reading File Metadata	834
10.6 Sharing Files	839
10.7 I/O Redirection	842
10.8 Standard I/O	843
10.9 Putting It Together: Which I/O Functions Should I Use?	844
10.10 Summary	845
11 Network Programming	849
11.1 The Client-Server Programming Model	849
11.2 Networks	850
11.3 The Global IP Internet	855
11.3.1 IP Addresses	856
11.3.2 Internet Domain Names	858
11.3.3 Internet Connections	862
11.4 The Sockets Interface	863
11.4.1 Socket Address Structures	863
11.4.2 The socket Function	864
11.4.3 The connect Function	865
11.4.4 The open_clientfd Function	866

11.4.5	The <code>bind</code> Function	867
11.4.6	The <code>listen</code> Function	867
11.4.7	The <code>open_listenfd</code> Function	867
11.4.8	The <code>accept</code> Function	869
11.4.9	Example Echo Client and Server	870
11.5	Web Servers	873
11.5.1	Web Basics	873
11.5.2	Web Content	874
11.5.3	HTTP Transactions	875
11.5.4	Serving Dynamic Content	878
11.6	Putting it Together: The TINY Web Server	879
11.7	Summary	889
12	Concurrent Programming	895
12.1	Concurrent Programming With Processes	896
12.1.1	A Concurrent Server Based on Processes	897
12.1.2	Pros and Cons of Processes	898
12.2	Concurrent Programming With I/O Multiplexing	900
12.2.1	A Concurrent Event-Driven Server Based on I/O Multiplexing	903
12.2.2	Pros and Cons of I/O Multiplexing	905
12.3	Concurrent Programming With Threads	908
12.3.1	Thread Execution Model	909
12.3.2	Posix Threads	909
12.3.3	Creating Threads	910
12.3.4	Terminating Threads	911
12.3.5	Reaping Terminated Threads	912
12.3.6	Detaching Threads	912
12.3.7	Initializing Threads	913
12.3.8	A Concurrent Server Based on Threads	913
12.4	Shared Variables in Threaded Programs	915
12.4.1	Threads Memory Model	915
12.4.2	Mapping Variables to Memory	917
12.4.3	Shared Variables	917

12.5	Synchronizing Threads with Semaphores	918
12.5.1	Progress Graphs	921
12.5.2	Semaphores	923
12.5.3	Using Semaphores for Mutual Exclusion	924
12.5.4	Using Semaphores to Schedule Shared Resources	926
12.5.5	Putting It Together: A Concurrent Server Based on Prethreading	930
12.6	Using Threads for Parallelism	931
12.7	Other Concurrency Issues	938
12.7.1	Thread Safety	938
12.7.2	Reentrancy	940
12.7.3	Using Existing Library Functions in Threaded Programs	941
12.7.4	Races	942
12.7.5	Deadlocks	944
12.8	Summary	947
A	Error Handling	959
A.1	Error Handling in Unix Systems	959
A.2	Error-Handling Wrappers	961