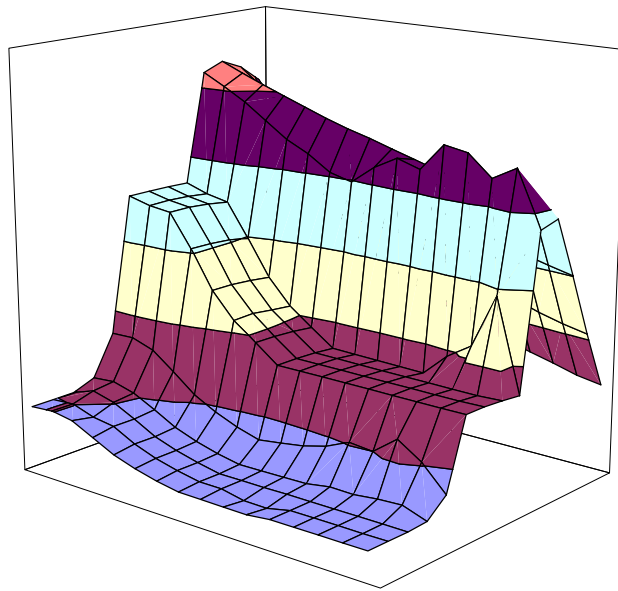


Computer Systems

*A Programmer's Perspective*¹



Randal E. Bryant
David R. O'Hallaron

June 7, 2002

¹Copyright © 2002, 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	4
1.4 Processors Read and Interpret Instructions Stored in Memory	5
1.4.1 Hardware Organization of a System	6
1.4.2 Running the <code>hello</code> Program	8
1.5 Caches Matter	10
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	14
1.7.3 Virtual Memory	14
1.7.4 Files	16
1.8 Systems Communicate With Other Systems Using Networks	17
1.9 The Next Step	18
1.10 Summary	18
I Program Structure and Execution	21
2 Representing and Manipulating Information	25
2.1 Information Storage	27
2.1.1 Hexadecimal Notation	27

2.1.2	Words	31
2.1.3	Data Sizes	31
2.1.4	Addressing and Byte Ordering	32
2.1.5	Representing Strings	39
2.1.6	Representing Code	39
2.1.7	Boolean Algebras and Rings	40
2.1.8	Bit-Level Operations in C	44
2.1.9	Logical Operations in C	46
2.1.10	Shift Operations in C	47
2.2	Integer Representations	48
2.2.1	Integral Data Types	48
2.2.2	Unsigned and Two's-Complement Encodings	48
2.2.3	Conversions Between Signed and Unsigned	52
2.2.4	Signed vs. Unsigned in C	54
2.2.5	Expanding the Bit Representation of a Number	56
2.2.6	Truncating Numbers	59
2.2.7	Advice on Signed vs. Unsigned	60
2.3	Integer Arithmetic	61
2.3.1	Unsigned Addition	61
2.3.2	Two's-Complement Addition	64
2.3.3	Two's-Complement Negation	68
2.3.4	Unsigned Multiplication	69
2.3.5	Two's-Complement Multiplication	70
2.3.6	Multiplying by Powers of Two	71
2.3.7	Dividing by Powers of Two	72
2.4	Floating Point	75
2.4.1	Fractional Binary Numbers	75
2.4.2	IEEE Floating-Point Representation	78
2.4.3	Example Numbers	79
2.4.4	Rounding	84
2.4.5	Floating-Point Operations	85
2.4.6	Floating Point in C	86
2.5	Summary	91

3	Machine-Level Representation of Programs	115
3.1	A Historical Perspective	116
3.2	Program Encodings	119
3.2.1	Machine-Level Code	119
3.2.2	Code Examples	120
3.2.3	A Note on Formatting	123
3.3	Data Formats	125
3.4	Accessing Information	126
3.4.1	Operand Specifiers	127
3.4.2	Data Movement Instructions	128
3.4.3	Data Movement Example	131
3.5	Arithmetic and Logical Operations	133
3.5.1	Load Effective Address	133
3.5.2	Unary and Binary Operations	134
3.5.3	Shift Operations	135
3.5.4	Discussion	135
3.5.5	Special Arithmetic Operations	136
3.6	Control	138
3.6.1	Condition Codes	138
3.6.2	Accessing the Condition Codes	139
3.6.3	Jump Instructions and their Encodings	141
3.6.4	Translating Conditional Branches	146
3.6.5	Loops	147
3.6.6	Switch Statements	156
3.7	Procedures	160
3.7.1	Stack Frame Structure	160
3.7.2	Transferring Control	162
3.7.3	Register Usage Conventions	162
3.7.4	Procedure Example	164
3.7.5	Recursive Procedures	168
3.8	Array Allocation and Access	171
3.8.1	Basic Principles	171
3.8.2	Pointer Arithmetic	172

3.8.3	Arrays and Loops	173
3.8.4	Nested Arrays	175
3.8.5	Fixed Size Arrays	176
3.8.6	Dynamically Allocated Arrays	179
3.9	Heterogeneous Data Structures	181
3.9.1	Structures	181
3.9.2	Unions	184
3.10	Alignment	188
3.11	Putting it Together: Understanding Pointers	190
3.12	Life in the Real World: Using the GDB Debugger	194
3.13	Out-of-Bounds Memory References and Buffer Overflow	194
3.14	*Floating-Point Code	200
3.14.1	Floating-Point Registers	201
3.14.2	Stack Evaluation of Expressions	201
3.14.3	Floating-Point Data Movement and Conversion Operations	204
3.14.4	Floating-Point Arithmetic Instructions	206
3.14.5	Using Floating Point in Procedures	209
3.14.6	Testing and Comparing Floating-Point Values	211
3.15	*Embedding Assembly Code in C Programs	213
3.15.1	Basic Inline Assembly	214
3.15.2	Extended Form of asm	216
3.16	Summary	219
4	Processor Architecture	243
4.1	The Y86 Instruction Set Architecture	245
4.2	Logic Design and the Hardware Control Language HCL	257
4.2.1	Logic Gates	258
4.2.2	Combinational Circuits and HCL Boolean Expressions	258
4.2.3	Word-Level Combinational Circuits and HCL Integer Expressions	260
4.2.4	Set Membership	265
4.2.5	Memory and Clocking	266
4.3	Sequential Y86 Implementations	267
4.3.1	Organizing Processing into Stages	267

4.3.2	SEQ Hardware Structure	276
4.3.3	SEQ Timing	281
4.3.4	SEQ Stage Implementations	284
4.3.5	SEQ+: Rearranging the Computation Stages	291
4.4	General Principles of Pipelining	294
4.4.1	Computational Pipelines	294
4.4.2	A Detailed Look at Pipeline Operation	296
4.4.3	Limitations of Pipelining	298
4.4.4	Pipelining a System with Feedback	300
4.5	Pipelined Y86 Implementations	301
4.5.1	Inserting Pipeline Registers	301
4.5.2	Rearranging and Relabeling Signals	305
4.5.3	Next PC Prediction	307
4.5.4	Pipeline Hazards	308
4.5.5	Avoiding Data Hazards by Stalling	313
4.5.6	Avoiding Data Hazards by Forwarding	315
4.5.7	Load/Use Data Hazards	319
4.5.8	PIPE Stage Implementations	324
4.5.9	Pipeline Control Logic	331
4.5.10	Performance Analysis	340
4.5.11	Unfinished Business	341
4.6	Summary	345
4.6.1	Y86 Simulators	346
5	Optimizing Program Performance	363
5.1	Capabilities and Limitations of Optimizing Compilers	364
5.2	Expressing Program Performance	367
5.3	Program Example	369
5.4	Eliminating Loop Inefficiencies	372
5.5	Reducing Procedure Calls	376
5.6	Eliminating Unneeded Memory References	378
5.7	Understanding Modern Processors	380
5.7.1	Overall Operation	381

5.7.2	Functional Unit Performance	384
5.7.3	A Closer Look at Processor Operation	385
5.8	Reducing Loop Overhead	393
5.9	Converting to Pointer Code	398
5.10	Enhancing Parallelism	400
5.10.1	Loop Splitting	401
5.10.2	Register Spilling	405
5.10.3	Limits to Parallelism	407
5.11	Putting it Together: Summary of Results for Optimizing Combining Code	408
5.11.1	Floating-Point Performance Anomaly	409
5.11.2	Changing Platforms	410
5.12	Branch Prediction and Misprediction Penalties	410
5.13	Understanding Memory Performance	413
5.13.1	Load Latency	414
5.13.2	Store Latency	416
5.14	Life in the Real World: Performance Improvement Techniques	421
5.15	Identifying and Eliminating Performance Bottlenecks	422
5.15.1	Program Profiling	422
5.15.2	Using a Profiler to Guide Optimization	424
5.15.3	Amdahl's Law	427
5.16	Summary	428
6	The Memory Hierarchy	439
6.1	Storage Technologies	440
6.1.1	Random-Access Memory	440
6.1.2	Disk Storage	447
6.1.3	Storage Technology Trends	458
6.2	Locality	458
6.2.1	Locality of References to Program Data	460
6.2.2	Locality of Instruction Fetches	462
6.2.3	Summary of Locality	462
6.3	The Memory Hierarchy	464
6.3.1	Caching in the Memory Hierarchy	465

6.3.2	Summary of Memory Hierarchy Concepts	467
6.4	Cache Memories	468
6.4.1	Generic Cache Memory Organization	469
6.4.2	Direct-Mapped Caches	471
6.4.3	Set Associative Caches	477
6.4.4	Fully Associative Caches	479
6.4.5	Issues with Writes	482
6.4.6	Instruction Caches and Unified Caches	483
6.4.7	Performance Impact of Cache Parameters	484
6.5	Writing Cache-friendly Code	486
6.6	Putting it Together: The Impact of Caches on Program Performance	491
6.6.1	The Memory Mountain	491
6.6.2	Rearranging Loops to Increase Spatial Locality	495
6.6.3	Using Blocking to Increase Temporal Locality	499
6.7	Putting It Together: Exploiting Locality in Your Programs	502
6.8	Summary	502

II Running Programs on a System 517

7 Linking 521

7.1	Compiler Drivers	522
7.2	Static Linking	523
7.3	Object Files	524
7.4	Relocatable Object Files	525
7.5	Symbols and Symbol Tables	526
7.6	Symbol Resolution	529
7.6.1	How Linkers Resolve Multiply Defined Global Symbols	530
7.6.2	Linking with Static Libraries	533
7.6.3	How Linkers Use Static Libraries to Resolve References	536
7.7	Relocation	538
7.7.1	Relocation Entries	538
7.7.2	Relocating Symbol References	539
7.8	Executable Object Files	543

7.9	Loading Executable Object Files	544
7.10	Dynamic Linking with Shared Libraries	546
7.11	Loading and Linking Shared Libraries from Applications	548
7.12	*Position-Independent Code (PIC)	551
7.13	Tools for Manipulating Object Files	554
7.14	Summary	554
8	Exceptional Control Flow	565
8.1	Exceptions	566
8.1.1	Exception Handling	567
8.1.2	Classes of Exceptions	569
8.1.3	Exceptions in Intel Processors	571
8.2	Processes	572
8.2.1	Logical Control Flow	573
8.2.2	Private Address Space	574
8.2.3	User and Kernel Modes	574
8.2.4	Context Switches	575
8.3	System Calls and Error Handling	577
8.4	Process Control	578
8.4.1	Obtaining Process ID's	578
8.4.2	Creating and Terminating Processes	579
8.4.3	Reaping Child Processes	584
8.4.4	Putting Processes to Sleep	589
8.4.5	Loading and Running Programs	590
8.4.6	Using fork and execve to Run Programs	593
8.5	Signals	594
8.5.1	Signal Terminology	594
8.5.2	Sending Signals	598
8.5.3	Receiving Signals	601
8.5.4	Signal Handling Issues	604
8.5.5	Portable Signal Handling	609
8.5.6	Explicitly Blocking Signals	611
8.6	Nonlocal Jumps	613

8.7	Tools for Manipulating Processes	618
8.8	Summary	618
9	Measuring Program Execution Time	629
9.1	The Flow of Time on a Computer System	630
9.1.1	Process Scheduling and Timer Interrupts	631
9.1.2	Time from an Application Program's Perspective	632
9.2	Measuring Time by Interval Counting	635
9.2.1	Operation	635
9.2.2	Reading the Process Timers	636
9.2.3	Accuracy of Process Timers	637
9.3	Cycle Counters	640
9.3.1	IA32 Cycle Counters	640
9.4	Measuring Program Execution Time with Cycle Counters	642
9.4.1	The Effects of Context Switching	642
9.4.2	Caching and Other Effects	643
9.4.3	The <i>K</i> -Best Measurement Scheme	647
9.5	Time-of-Day Measurements	656
9.6	Putting it Together: An Experimental Protocol	659
9.7	Looking into the Future	659
9.8	Life in the Real World: An Implementation of the <i>K</i> -Best Measurement Scheme	660
9.9	Lessons Learned	660
9.10	Summary	661
10	Virtual Memory	667
10.1	Physical and Virtual Addressing	668
10.2	Address Spaces	669
10.3	VM as a Tool for Caching	670
10.3.1	DRAM Cache Organization	671
10.3.2	Page Tables	671
10.3.3	Page Hits	672
10.3.4	Page Faults	673
10.3.5	Allocating Pages	674

10.3.6	Locality to the Rescue Again	675
10.4	VM as a Tool for Memory Management	675
10.4.1	Simplifying Linking	676
10.4.2	Simplifying Sharing	676
10.4.3	Simplifying Memory Allocation	677
10.4.4	Simplifying Loading	677
10.5	VM as a Tool for Memory Protection	678
10.6	Address Translation	679
10.6.1	Integrating Caches and VM	682
10.6.2	Speeding up Address Translation with a TLB	682
10.6.3	Multi Level Page Tables	683
10.6.4	Putting it Together: End-to-end Address Translation	686
10.7	Case Study: The Pentium/Linux Memory System	690
10.7.1	Pentium Address Translation	690
10.7.2	Linux Virtual Memory System	695
10.8	Memory Mapping	698
10.8.1	Shared Objects Revisited	699
10.8.2	The <code>fork</code> Function Revisited	701
10.8.3	The <code>execve</code> Function Revisited	701
10.8.4	User-level Memory Mapping with the <code>mmap</code> Function	702
10.9	Dynamic Memory Allocation	704
10.9.1	The <code>malloc</code> and <code>free</code> Functions	705
10.9.2	Why Dynamic Memory Allocation?	708
10.9.3	Allocator Requirements and Goals	709
10.9.4	Fragmentation	710
10.9.5	Implementation Issues	711
10.9.6	Implicit Free Lists	712
10.9.7	Placing Allocated Blocks	713
10.9.8	Splitting Free Blocks	714
10.9.9	Getting Additional Heap Memory	714
10.9.10	Coalescing Free Blocks	714
10.9.11	Coalescing with Boundary Tags	715
10.9.12	Putting it Together: Implementing a Simple Allocator	717

10.9.13 Explicit Free Lists	725
10.9.14 Segregated Free Lists	726
10.10 Garbage Collection	728
10.10.1 Garbage Collector Basics	729
10.10.2 Mark&Sweep Garbage Collectors	730
10.10.3 Conservative Mark&Sweep for C Programs	731
10.11 Common Memory-Related Bugs in C Programs	733
10.11.1 Dereferencing Bad Pointers	733
10.11.2 Reading Uninitialized Memory	733
10.11.3 Allowing Stack Buffer Overflows	734
10.11.4 Assuming that Pointers and the Objects they Point to Are the Same Size	734
10.11.5 Making Off-by-one Errors	735
10.11.6 Referencing a Pointer Instead of the Object it Points to	735
10.11.7 Misunderstanding Pointer Arithmetic	736
10.11.8 Referencing Nonexistent Variables	736
10.11.9 Referencing Data in Free Heap Blocks	736
10.11.10 Introducing Memory Leaks	737
10.12 Recapping Some Key Ideas About Virtual Memory	737
10.13 Summary	738

III Interaction and Communication Between Programs 749

11 System-Level I/O 753

11.1 Unix I/O	754
11.2 Opening and Closing Files	754
11.3 Reading and Writing Files	756
11.4 Robust Reading and Writing with the R10 Package	758
11.4.1 R10 Unbuffered Input and Output Functions	758
11.4.2 R10 Buffered Input Functions	759
11.5 Reading File Metadata	762
11.6 Sharing Files	767
11.7 I/O Redirection	770
11.8 Standard I/O	771

11.9 Putting It Together: Which I/O Functions Should I Use?	772
11.10 Summary	773
12 Network Programming	777
12.1 The Client-Server Programming Model	777
12.2 Networks	778
12.3 The Global IP Internet	783
12.3.1 IP Addresses	784
12.3.2 Internet Domain Names	786
12.3.3 Internet Connections	790
12.4 The Sockets Interface	791
12.4.1 Socket Address Structures	791
12.4.2 The <code>socket</code> Function	792
12.4.3 The <code>connect</code> Function	793
12.4.4 The <code>open_clientfd</code> Function	794
12.4.5 The <code>bind</code> Function	795
12.4.6 The <code>listen</code> Function	795
12.4.7 The <code>open_listenfd</code> Function	795
12.4.8 The <code>accept</code> Function	797
12.4.9 Example Echo Client and Server	798
12.5 Web Servers	801
12.5.1 Web Basics	801
12.5.2 Web Content	802
12.5.3 HTTP Transactions	803
12.5.4 Serving Dynamic Content	806
12.6 Putting it Together: The TINY Web Server	807
12.7 Summary	817
13 Concurrent Programming	823
13.1 Concurrent Programming With Processes	824
13.1.1 A Concurrent Server Based on Processes	825
13.1.2 Pros and Cons of Processes	826
13.2 Concurrent Programming With I/O Multiplexing	828

13.2.1	A Concurrent Event-Driven Server Based on I/O Multiplexing	831
13.2.2	Pros and Cons of I/O Multiplexing	835
13.3	Concurrent Programming With Threads	835
13.3.1	Thread Execution Model	836
13.3.2	Posix Threads	837
13.3.3	Creating Threads	838
13.3.4	Terminating Threads	838
13.3.5	Reaping Terminated Threads	839
13.3.6	Detaching Threads	839
13.3.7	Initializing Threads	840
13.3.8	A Concurrent Server Based on Threads	840
13.4	Shared Variables in Threaded Programs	842
13.4.1	Threads Memory Model	844
13.4.2	Mapping Variables to Memory	844
13.4.3	Shared Variables	845
13.5	Synchronizing Threads with Semaphores	845
13.5.1	Progress Graphs	848
13.5.2	Using Semaphores to Access Shared Variables	851
13.5.3	Posix Semaphores	852
13.5.4	Using Semaphores to Schedule Shared Resources	853
13.6	Putting It Together: A Concurrent Server Based on Prethreading	856
13.7	Other Concurrency Issues	859
13.7.1	Thread Safety	859
13.7.2	Reentrancy	861
13.7.3	Using Existing Library Functions in Threaded Programs	862
13.7.4	Races	863
13.7.5	Deadlocks	866
13.8	Summary	868
A	HCL Descriptions of Processor Control Logic	877
A.1	HCL Reference Manual	877
A.1.1	Signal Declarations	878
A.1.2	Quoted Text	878

A.1.3	Expressions and Blocks	878
A.1.4	HCL Example	880
A.2	SEQ	882
A.3	SEQ+	885
A.4	PIPE	889
B	Error Handling	897
B.1	Error Handling in Unix Systems	897
B.2	Error-Handling Wrappers	899
B.3	The <code>csapp.h</code> Header File	902
B.4	The <code>csapp.c</code> Source File	906