

CONTENTS IN DETAIL

ACKNOWLEDGMENTS	xv
------------------------	-----------

INTRODUCTION	xvii
---------------------	-------------

1	
THINKING LOW-LEVEL, WRITING HIGH-LEVEL	1

1.1	Misconceptions About Compiler Quality	2
1.2	Why Learning Assembly Language Is Still a Good Idea	2
1.3	Why Learning Assembly Language Isn't Absolutely Necessary.....	3
1.4	Thinking Low-Level.....	3
1.4.1	Compilers Are Only as Good as the Source Code You Feed Them.....	4
1.4.2	Helping the Compiler Produce Better Machine Code	4
1.4.3	How to Think in Assembly While Writing HLL Code	5
1.5	Writing High-Level	7
1.6	Assumptions	7
1.7	Language-Neutral Approach	8
1.8	Characteristics of Great Code	8
1.9	The Environment for This Text.....	9
1.10	For More Information.....	10

2	
SHOULDN'T YOU LEARN ASSEMBLY LANGUAGE?	11

2.1	Roadblocks to Learning Assembly Language.....	12
2.2	<i>Write Great Code, Volume 2</i> , to the Rescue.....	12
2.3	High-Level Assemblers to the Rescue	13
2.4	The High-Level Assembler (HLA)	14
2.5	Thinking High-Level, Writing Low-Level.....	15
2.6	The Assembly Programming Paradigm (Thinking Low-Level)	16
2.7	<i>The Art of Assembly Language</i> and Other Resources	18

3	
80X86 ASSEMBLY FOR THE HLL PROGRAMMER	21

3.1	Learning One Assembly Language Is Good, Learning More Is Better	22
3.2	80x86 Assembly Syntaxes	22
3.3	Basic 80x86 Architecture.....	23
3.3.1	Registers	23
3.3.2	80x86 General-Purpose Registers	24
3.3.3	The 80x86 EFLAGS Register.....	25

3.4	Literal Constants.....	26
3.4.1	Binary Literal Constants.....	26
3.4.2	Decimal Literal Constants.....	27
3.4.3	Hexadecimal Literal Constants.....	27
3.4.4	Character and String Literal Constants.....	28
3.4.5	Floating-Point Literal Constants.....	29
3.5	Manifest (Symbolic) Constants in Assembly Language.....	30
3.5.1	Manifest Constants in HLA.....	30
3.5.2	Manifest Constants in Gas.....	30
3.5.3	Manifest Constants in MASM and TASM.....	31
3.6	80x86 Addressing Modes.....	31
3.6.1	80x86 Register Addressing Modes.....	31
3.6.2	Immediate Addressing Mode.....	32
3.6.3	Displacement-Only Memory Addressing Mode.....	33
3.6.4	Register Indirect Addressing Mode.....	35
3.6.5	Indexed Addressing Mode.....	36
3.6.6	Scaled-Indexed Addressing Modes.....	38
3.7	Declaring Data in Assembly Language.....	39
3.7.1	Data Declarations in HLA.....	40
3.7.2	Data Declarations in MASM and TASM.....	41
3.7.3	Data Declarations in Gas.....	41
3.8	Specifying Operand Sizes in Assembly Language.....	44
3.8.1	Type Coercion in HLA.....	44
3.8.2	Type Coercion in MASM and TASM.....	45
3.8.3	Type Coercion in Gas.....	45
3.9	The Minimal 80x86 Instruction Set.....	46
3.10	For More Information.....	46

4 POWERPC ASSEMBLY FOR THE HLL PROGRAMMER 47

4.1	Learning One Assembly Language Is Good; More Is Better.....	48
4.2	Assembly Syntaxes.....	48
4.3	Basic PowerPC Architecture.....	49
4.3.1	General-Purpose Integer Registers.....	49
4.3.2	General-Purpose Floating-Point Registers.....	49
4.3.3	User-Mode-Accessible Special-Purpose Registers.....	49
4.4	Literal Constants.....	52
4.4.1	Binary Literal Constants.....	52
4.4.2	Decimal Literal Constants.....	53
4.4.3	Hexadecimal Literal Constants.....	53
4.4.4	Character and String Literal Constants.....	53
4.4.5	Floating-Point Literal Constants.....	53
4.5	Manifest (Symbolic) Constants in Assembly Language.....	54
4.6	PowerPC Addressing Modes.....	54
4.6.1	PowerPC Register Access.....	54
4.6.2	The Immediate Addressing Mode.....	54
4.6.3	PowerPC Memory Addressing Modes.....	55
4.7	Declaring Data in Assembly Language.....	56
4.8	Specifying Operand Sizes in Assembly Language.....	59
4.9	The Minimal Instruction Set.....	59
4.10	For More Information.....	59

5
COMPILER OPERATION AND CODE GENERATION **61**

5.1	File Types That Programming Languages Use.....	62
5.2	Programming Language Source Files	62
5.2.1	Tokenized Source Files	62
5.2.2	Specialized Source File Formats.....	63
5.3	Types of Computer Language Processors.....	63
5.3.1	Pure Interpreters.....	64
5.3.2	Interpreters	64
5.3.3	Compilers	64
5.3.4	Incremental Compilers	65
5.4	The Translation Process.....	66
5.4.1	Lexical Analysis and Tokens	68
5.4.2	Parsing (Syntax Analysis)	69
5.4.3	Intermediate Code Generation.....	69
5.4.4	Optimization	70
5.4.5	Comparing Different Compilers' Optimizations	81
5.4.6	Native Code Generation	81
5.5	Compiler Output.....	81
5.5.1	Emitting HLL Code as Compiler Output	82
5.5.2	Emitting Assembly Language as Compiler Output.....	83
5.5.3	Emitting Object Files as Compiler Output	84
5.5.4	Emitting Executable Files as Compiler Output	85
5.6	Object File Formats	85
5.6.1	The COFF File Header.....	86
5.6.2	The COFF Optional Header	88
5.6.3	COFF Section Headers	91
5.6.4	COFF Sections.....	93
5.6.5	The Relocation Section.....	94
5.6.6	Debugging and Symbolic Information.....	94
5.6.7	Learning More About Object File Formats	94
5.7	Executable File Formats	94
5.7.1	Pages, Segments, and File Size	95
5.7.2	Internal Fragmentation.....	97
5.7.3	So Why Optimize for Space?.....	98
5.8	Data and Code Alignment in an Object File.....	99
5.8.1	Choosing a Section Alignment Size.....	100
5.8.2	Combining Sections	101
5.8.3	Controlling the Section Alignment	102
5.8.4	Section Alignment and Library Modules.....	102
5.9	Linkers and Their Effect on Code.....	110
5.10	For More Information.....	113

6
TOOLS FOR ANALYZING COMPILER OUTPUT **115**

6.1	Background.....	116
6.2	Telling a Compiler to Produce Assembly Output.....	117
6.2.1	Assembly Output from GNU and Borland Compilers	118
6.2.2	Assembly Output from Visual C++	118
6.2.3	Example Assembly Language Output.....	118
6.2.4	Analyzing Assembly Output from a Compiler	128

6.3	Using Object-Code Utilities to Analyze Compiler Output.....	129
6.3.1	The Microsoft dumpbin.exe Utility	129
6.3.2	The FSF/GNU objdump.exe Utility	142
6.4	Using a Disassembler to Analyze Compiler Output.....	146
6.5	Using a Debugger to Analyze Compiler Output	149
6.5.1	Using an IDE's Debugger	149
6.5.2	Using a Stand-Alone Debugger.....	151
6.6	Comparing Output from Two Compilations	152
6.6.1	Before-and-After Comparisons with diff	153
6.6.2	Manual Comparison	162
6.7	For More Information.....	163

7 CONSTANTS AND HIGH-LEVEL LANGUAGES **165**

7.1	Literal Constants and Program Efficiency	166
7.2	Literal Constants Versus Manifest Constants	168
7.3	Constant Expressions	169
7.4	Manifest Constants Versus Read-Only Memory Objects.....	171
7.5	Enumerated Types	172
7.6	Boolean Constants	174
7.7	Floating-Point Constants	176
7.8	String Constants.....	182
7.9	Composite Data Type Constants	186
7.10	For More Information.....	188

8 VARIABLES IN A HIGH-LEVEL LANGUAGE **189**

8.1	Runtime Memory Organization	190
8.1.1	The Code, Constant, and Read-Only Sections.....	191
8.1.2	The Static Variables Section	193
8.1.3	The BSS Section	194
8.1.4	The Stack Section	195
8.1.5	The Heap Section and Dynamic Memory Allocation	196
8.2	What Is a Variable?	196
8.2.1	Attributes	197
8.2.2	Binding.....	197
8.2.3	Static Objects	197
8.2.4	Dynamic Objects	197
8.2.5	Scope.....	198
8.2.6	Lifetime	198
8.2.7	So What Is a Variable?	199
8.3	Variable Storage	199
8.3.1	Static Binding and Static Variables.....	199
8.3.2	Pseudo-Static Binding and Automatic Variables.....	203
8.3.3	Dynamic Binding and Dynamic Variables	206
8.4	Common Primitive Data Types	210
8.4.1	Integer Variables	210
8.4.2	Floating-Point/Real Variables	213
8.4.3	Character Variables	214
8.4.4	Boolean Variables.....	215

8.5	Variable Addresses and High-level Languages	215
8.5.1	Storage Allocation for Global and Static Variables	216
8.5.2	Using Automatic Variables to Reduce Offset Sizes.....	217
8.5.3	Storage Allocation for Intermediate Variables	223
8.5.4	Storage Allocation for Dynamic Variables and Pointers.....	224
8.5.5	Using Records/Structures to Reduce Instruction Offset Sizes.....	226
8.5.6	Register Variables	228
8.6	Variable Alignment in Memory	229
8.6.1	Records and Alignment.....	235
8.7	For More Information.....	239

9

ARRAY DATA TYPES **241**

9.1	What Is an Array?	242
9.1.1	Array Declarations	242
9.1.2	Array Representation in Memory.....	246
9.1.3	Accessing Elements of an Array.....	250
9.1.4	Padding Versus Packing.....	252
9.1.5	Multidimensional Arrays	255
9.1.6	Dynamic Versus Static Arrays	270
9.2	For More Information.....	279

10

STRING DATA TYPES **281**

10.1	Character String Formats	282
10.1.1	Zero-Terminated Strings	283
10.1.2	Length-Prefixed Strings	300
10.1.3	7-Bit Strings.....	302
10.1.4	HLA Strings	303
10.1.5	Descriptor-Based Strings	306
10.2	Static, Pseudo-Dynamic, and Dynamic Strings.....	307
10.2.1	Static Strings	308
10.2.2	Pseudo-Dynamic Strings	308
10.2.3	Dynamic Strings.....	308
10.3	Reference Counting for Strings.....	309
10.4	Delphi/Kylix Strings	310
10.5	Using Strings in a High-Level Language	310
10.6	Character Data in Strings.....	312
10.7	For More Information.....	314

11

POINTER DATA TYPES **315**

11.1	Defining and Demystifying Pointers	316
11.2	Pointer Implementation in High-Level Languages.....	317
11.3	Pointers and Dynamic Memory Allocation	320
11.4	Pointer Operations and Pointer Arithmetic.....	320
11.4.1	Adding an Integer to a Pointer.....	322
11.4.2	Subtracting an Integer from a Pointer.....	323

11.4.3	Subtracting a Pointer from a Pointer	324
11.4.4	Comparing Pointers.....	325
11.4.5	Logical AND/OR and Pointers.....	327
11.4.6	Other Operations with Pointers.....	328
11.5	A Simple Memory Allocator Example	329
11.6	Garbage Collection	332
11.7	The OS and Memory Allocation.....	332
11.8	Heap Memory Overhead.....	333
11.9	Common Pointer Problems.....	335
11.9.1	Using an Uninitialized Pointer.....	336
11.9.2	Using a Pointer That Contains an Illegal Value.....	337
11.9.3	Continuing to Use Storage After It Has Been Freed.....	337
11.9.4	Failing to Free Storage When Done with It.....	338
11.9.5	Accessing Indirect Data Using the Wrong Data Type.....	339
11.10	For More Information.....	340

12 **RECORD, UNION, AND CLASS DATA TYPES** **341**

12.1	Records	342
12.1.1	Record Declarations in Various Languages.....	342
12.1.2	Instantiation of a Record	344
12.1.3	Initialization of Record Data at Compile Time	350
12.1.4	Memory Storage of Records	355
12.1.5	Using Records to Improve Memory Performance	358
12.1.6	Dynamic Record Types and Databases	359
12.2	Discriminant Unions.....	360
12.3	Union Declarations in Various Languages	360
12.3.1	Union Declarations in C/C++.....	361
12.3.2	Union Declarations in Pascal/Delphi/Kylix	361
12.3.3	Union Declarations in HLA	362
12.4	Memory Storage of Unions	362
12.5	Other Uses of Unions	363
12.6	Variant Types	364
12.7	Namespaces	369
12.8	Classes and Objects.....	371
12.8.1	Classes Versus Objects	371
12.8.2	Simple Class Declarations in C++	371
12.8.3	Virtual Method Tables.....	373
12.8.4	Sharing VMTs.....	377
12.8.5	Inheritance in Classes.....	377
12.8.6	Polymorphism in Classes.....	380
12.8.7	Classes, Objects, and Performance	381
12.9	For More Information.....	382

13 **ARITHMETIC AND LOGICAL EXPRESSIONS** **385**

13.1	Arithmetic Expressions and Computer Architecture	386
13.1.1	Stack-Based Machines	386
13.1.2	Accumulator-Based Machines	391

13.1.3	Register-Based Machines.....	393
13.1.4	Typical Forms of Arithmetic Expressions.....	394
13.1.5	Three-Address Architectures.....	395
13.1.6	Two-Address Architectures.....	395
13.1.7	Architectural Differences and Your Code.....	396
13.1.8	Handling Complex Expressions.....	397
13.2	Optimization of Arithmetic Statements.....	397
13.2.1	Constant Folding.....	398
13.2.2	Constant Propagation.....	399
13.2.3	Dead Code Elimination.....	400
13.2.4	Common Subexpression Elimination.....	402
13.2.5	Strength Reduction.....	406
13.2.6	Induction.....	410
13.2.7	Loop Invariants.....	413
13.2.8	Optimizers and Programmers.....	416
13.3	Side Effects in Arithmetic Expressions.....	416
13.4	Containing Side Effects: Sequence Points.....	421
13.5	Avoiding Problems Caused by Side Effects.....	425
13.6	Forcing a Particular Order of Evaluation.....	425
13.7	Short-Circuit Evaluation.....	427
13.7.1	Short-Circuit Evaluation and Boolean Expressions.....	428
13.7.2	Forcing Short-Circuit or Complete Boolean Evaluation.....	430
13.7.3	Efficiency Issues.....	432
13.8	The Relative Cost of Arithmetic Operations.....	436
13.9	For More Information.....	437

14 CONTROL STRUCTURES AND PROGRAMMATIC DECISIONS 439

14.1	Control Structures Are Slower Than Computations!.....	440
14.2	Introduction to Low-Level Control Structures.....	440
14.3	The goto Statement.....	443
14.4	break, continue, next, return, and Other Limited Forms of the goto Statement.....	447
14.5	The if Statement.....	448
14.5.1	Improving the Efficiency of Certain if/else Statements.....	450
14.5.2	Forcing Complete Boolean Evaluation in an if Statement.....	453
14.5.3	Forcing Short-Circuit Boolean Evaluation in an if Statement.....	460
14.6	The switch/case Statement.....	466
14.6.1	Semantics of a switch/case Statement.....	467
14.6.2	Jump Tables Versus Chained Comparisons.....	468
14.6.3	Other Implementations of switch/case.....	475
14.6.4	Compiler Output for switch Statements.....	486
14.7	For More Information.....	486

15 ITERATIVE CONTROL STRUCTURES 489

15.1	The while Loop.....	489
15.1.1	Forcing Complete Boolean Evaluation in a while Loop.....	492
15.1.2	Forcing Short-Circuit Boolean Evaluation in a while Loop.....	501

15.2	The repeat..until (do..until/do..while) Loop.....	504
15.2.1	Forcing Complete Boolean Evaluation in a repeat..until Loop.....	507
15.2.2	Forcing Short-Circuit Boolean Evaluation in a repeat..until Loop.....	510
15.3	The forever..endfor Loop.....	515
15.3.1	Forcing Complete Boolean Evaluation in a forever Loop.....	518
15.3.2	Forcing Short-Circuit Boolean Evaluation in a forever Loop.....	518
15.4	The Definite Loop (for Loops)	518
15.5	For More Information.....	520

16 FUNCTIONS AND PROCEDURES 521

16.1	Simple Function and Procedure Calls.....	522
16.1.1	Storing the Return Address	525
16.1.2	Other Sources of Overhead	529
16.2	Leaf Functions and Procedures	530
16.3	Macros and Inline Functions	534
16.4	Passing Parameters to a Function or Procedure	540
16.5	Activation Records and the Stack	547
16.5.1	Composition of the Activation Record	549
16.5.2	Assigning Offsets to Local Variables.....	552
16.5.3	Associating Offsets with Parameters	554
16.5.4	Accessing Parameters and Local Variables.....	559
16.6	Parameter-Passing Mechanisms.....	567
16.6.1	Pass-by-Value	568
16.6.2	Pass-by-Reference.....	568
16.7	Function Return Values.....	570
16.8	For More Information.....	577

ENGINEERING SOFTWARE 579

APPENDIX A BRIEF COMPARISON OF THE 80X86 AND POWERPC CPU FAMILIES 581

A.1	Architectural Differences Between RISC and CISC.....	582
A.1.1	Work Accomplished per Instruction.....	582
A.1.2	Instruction Size	583
A.1.3	Clock Speed and Clocks per Instruction.....	583
A.1.4	Memory Access and Addressing Modes	584
A.1.5	Registers.....	585
A.1.6	Immediate (Constant) Operands	585
A.1.7	Stacks	585
A.2	Compiler and Application Binary Interface Issues.....	586
A.3	Writing Great Code for Both Architectures.....	587

ONLINE APPENDICES 589

INDEX 591