

# CONTENTS IN DETAIL

## ACKNOWLEDGMENTS xxiii

## INTRODUCTION xxv

Who This Book Is For . . . . .	.xxvi
Why Write a C Compiler? . . . . .	xxvii
Compilation from 10,000 Feet . . . . .	xxvii
What You'll Build . . . . .	xxviii
How to Use This Book . . . . .	xxxii
The Test Suite . . . . .	xxxii
Extra Credit Features . . . . .	xxxii
Some Advice on Choosing an Implementation Language . . . . .	xxxiii
System Requirements . . . . .	xxxiv
Installing GCC and GDB on Linux . . . . .	xxxiv
Installing the Command Line Developer Tools on macOS . . . . .	xxxv
Running on Apple Silicon . . . . .	xxxv
Validating Your Setup . . . . .	xxxv
Additional Resources . . . . .	xxxvi
Let's Go! . . . . .	xxxvii

## PART I: THE BASICS 1

### 1 **A MINIMAL COMPILER** 3

The Four Compiler Passes . . . . .	3
Hello, Assembly! . . . . .	4
The Compiler Driver . . . . .	7
The Lexer . . . . .	8
The Parser . . . . .	10
An Example Abstract Syntax Tree . . . . .	11
The AST Definition . . . . .	13
The Formal Grammar . . . . .	14
Recursive Descent Parsing . . . . .	15
Assembly Generation . . . . .	17
Code Emission . . . . .	19
Summary . . . . .	21
Additional Resources . . . . .	21

### 2 **UNARY OPERATORS** 25

Negation and Bitwise Complement in Assembly . . . . .	26
The Stack . . . . .	27
The Lexer . . . . .	31

The Parser . . . . .	33
TACKY: A New Intermediate Representation . . . . .	35
Defining TACKY . . . . .	36
Generating TACKY . . . . .	37
Generating Names for Temporary Variables . . . . .	38
Updating the Compiler Driver . . . . .	38
Assembly Generation . . . . .	39
Converting TACKY to Assembly . . . . .	39
Replacing Pseudoregisters . . . . .	42
Fixing Up Instructions . . . . .	42
Code Emission . . . . .	43
Summary . . . . .	45
Additional Resources . . . . .	45

### **3 BINARY OPERATORS 47**

The Lexer . . . . .	48
The Parser . . . . .	48
The Trouble with Recursive Descent Parsing . . . . .	50
The Adequate Solution: Refactoring the Grammar . . . . .	51
The Better Solution: Precedence Climbing . . . . .	51
Precedence Climbing in Action . . . . .	55
TACKY Generation . . . . .	58
Assembly Generation . . . . .	60
Doing Arithmetic in Assembly . . . . .	60
Converting Binary Operations to Assembly . . . . .	61
Replacing Pseudoregisters . . . . .	64
Fixing Up the idiv, add, sub, and imul Instructions . . . . .	64
Code Emission . . . . .	65
Extra Credit: Bitwise Operators . . . . .	67
Summary . . . . .	67
Additional Resources . . . . .	68

### **4 LOGICAL AND RELATIONAL OPERATORS 71**

Short-Circuiting Operators . . . . .	72
The Lexer . . . . .	72
The Parser . . . . .	73
TACKY Generation . . . . .	75
Adding Jumps, Copies, and Comparisons to the TACKY IR . . . . .	75
Converting Short-Circuiting Operators to TACKY . . . . .	76
Generating Labels . . . . .	77
Comparisons and Jumps in Assembly . . . . .	78
Comparisons and Status Flags . . . . .	78
Conditional Set Instructions . . . . .	82
Jump Instructions . . . . .	83
Assembly Generation . . . . .	85
Replacing Pseudoregisters . . . . .	87
Fixing Up the cmp Instruction . . . . .	88
Code Emission . . . . .	88

Summary . . . . .	90
Additional Resources . . . . .	91

## **5 LOCAL VARIABLES 93**

Variables, Declarations, and Assignment. . . . .	94
The Lexer . . . . .	97
The Parser. . . . .	97
The Updated AST and Grammar. . . . .	97
An Improved Precedence Climbing Algorithm. . . . .	101
Semantic Analysis . . . . .	103
Variable Resolution . . . . .	105
The --validate Option. . . . .	109
TACKY Generation. . . . .	109
Variable and Assignment Expressions . . . . .	110
Declarations, Statements, and Function Bodies . . . . .	110
Functions with No return Statement . . . . .	111
Extra Credit: Compound Assignment, Increment, and Decrement . . . . .	113
Summary . . . . .	114

## **6 IF STATEMENTS AND CONDITIONAL EXPRESSIONS 117**

The Lexer . . . . .	118
The Parser. . . . .	118
Parsing if Statements . . . . .	118
Parsing Conditional Expressions . . . . .	121
Variable Resolution. . . . .	125
TACKY Generation. . . . .	126
Converting if Statements to TACKY . . . . .	126
Converting Conditional Expressions to TACKY . . . . .	127
Extra Credit: Labeled Statements and goto . . . . .	128
Summary . . . . .	128

## **7 COMPOUND STATEMENTS 131**

The Scoop on Scopes . . . . .	132
The Parser. . . . .	135
Variable Resolution. . . . .	136
Resolving Variables in Multiple Scopes . . . . .	137
Updating the Variable Resolution Pseudocode. . . . .	138
TACKY Generation. . . . .	140
Summary . . . . .	140

## **8 LOOPS 143**

Loops and How to Escape Them . . . . .	144
The Lexer . . . . .	148
The Parser. . . . .	149
Semantic Analysis . . . . .	151

Extending Variable Resolution . . . . .	151
Loop Labeling . . . . .	152
Implementing Loop Labeling . . . . .	154
TACKY Generation . . . . .	155
break and continue Statements . . . . .	155
do Loops . . . . .	156
while Loops . . . . .	157
for Loops . . . . .	157
Extra Credit: switch Statements . . . . .	159
Summary . . . . .	159

## 9

### FUNCTIONS

**161**

Declaring, Defining, and Calling Functions . . . . .	162
Declarations and Definitions . . . . .	162
Function Calls . . . . .	165
Identifier Linkage . . . . .	166
Compiling Libraries . . . . .	169
The Lexer . . . . .	170
The Parser . . . . .	170
Semantic Analysis . . . . .	174
Extending Identifier Resolution . . . . .	175
Writing the Type Checker . . . . .	178
TACKY Generation . . . . .	182
Assembly Generation . . . . .	183
Understanding Calling Conventions . . . . .	184
Calling Functions with the System V ABI . . . . .	185
Converting Function Calls and Definitions to Assembly . . . . .	194
Replacing Pseudoregisters . . . . .	200
Allocating Stack Space During Instruction Fix-Up . . . . .	200
Code Emission . . . . .	201
Calling Library Functions . . . . .	203
Summary . . . . .	205

## 10

### FILE SCOPE VARIABLE DECLARATIONS AND STORAGE-CLASS SPECIFIERS

**207**

All About Declarations . . . . .	208
Scope . . . . .	208
Linkage . . . . .	209
Storage Duration . . . . .	212
Definitions vs. Declarations . . . . .	214
Error Cases . . . . .	217
Linkage and Storage Duration in Assembly . . . . .	220
The Lexer . . . . .	223
The Parser . . . . .	224
Parsing Type and Storage-Class Specifiers . . . . .	225
Distinguishing Between Function and Variable Declarations . . . . .	226
Semantic Analysis . . . . .	227
Identifier Resolution: Resolving External Variables . . . . .	227
Type Checking: Tracking Static Functions and Variables . . . . .	229

TACKY Generation . . . . .	234
Assembly Generation . . . . .	235
Generating Assembly for Variable Definitions . . . . .	236
Replacing Pseudoregisters According to Their Storage Duration . . . . .	237
Fixing Up Instructions. . . . .	237
Code Emission. . . . .	238
Summary . . . . .	240

## **PART II: TYPES BEYOND INT 241**

### **11 LONG INTEGERS 243**

Long Integers in Assembly . . . . .	244
Type Conversions . . . . .	244
Static Long Variables . . . . .	246
The Lexer . . . . .	247
The Parser. . . . .	247
Semantic Analysis . . . . .	251
Adding Type Information to the AST . . . . .	252
Type Checking Expressions. . . . .	253
Type Checking return Statements. . . . .	256
Type Checking Declarations and Updating the Symbol Table . . . . .	257
TACKY Generation. . . . .	258
Tracking the Types of Temporary Variables. . . . .	260
Generating Extra Return Instructions . . . . .	261
Assembly Generation . . . . .	261
Tracking Assembly Types in the Backend Symbol Table . . . . .	266
Replacing Longword and Quadword Pseudoregisters . . . . .	267
Fixing Up Instructions. . . . .	267
Code Emission. . . . .	269
Summary . . . . .	271

### **12 UNSIGNED INTEGERS 273**

Type Conversions, Again . . . . .	274
Converting Between Signed and Unsigned Types of the Same Size. . . . .	274
Converting unsigned int to a Larger Type . . . . .	274
Converting signed int to a Larger Type. . . . .	275
Converting from Larger to Smaller Types . . . . .	275
The Lexer . . . . .	275
The Parser. . . . .	276
The Type Checker. . . . .	279
TACKY Generation. . . . .	281
Unsigned Integer Operations in Assembly . . . . .	283
Unsigned Comparisons . . . . .	283
Unsigned Division . . . . .	286
Zero Extension . . . . .	286

Assembly Generation . . . . .	287
Replacing Pseudoregisters . . . . .	289
Fixing Up the Div and MovZeroExtend Instructions . . . . .	290
Code Emission . . . . .	290
Summary . . . . .	292

## 13

### **FLOATING-POINT NUMBERS** **295**

IEEE 754, What Is It Good For? . . . . .	296
The IEEE 754 Double-Precision Format . . . . .	297
Rounding Behavior . . . . .	299
Rounding Modes . . . . .	299
Rounding Constants . . . . .	300
Rounding Type Conversions . . . . .	300
Rounding Arithmetic Operations . . . . .	301
Linking Shared Libraries . . . . .	301
The Lexer . . . . .	302
Recognizing Floating-Point Constant Tokens . . . . .	302
Matching the End of a Constant . . . . .	303
The Parser . . . . .	305
The Type Checker . . . . .	308
TACKY Generation . . . . .	309
Floating-Point Operations in Assembly . . . . .	310
Working with SSE Instructions . . . . .	311
Using Floating-Point Values in the System V Calling Convention . . . . .	312
Doing Arithmetic with SSE Instructions . . . . .	315
Comparing Floating-Point Numbers . . . . .	317
Converting Between Floating-Point and Integer Types . . . . .	317
Assembly Generation . . . . .	324
Floating-Point Constants . . . . .	326
Unary Instructions, Binary Instructions, and Conditional Jumps . . . . .	327
Type Conversions . . . . .	328
Function Calls . . . . .	329
Return Instructions . . . . .	333
The Complete Conversion from TACKY to Assembly . . . . .	333
Pseudoregister Replacement . . . . .	336
Instruction Fix-Up . . . . .	336
Code Emission . . . . .	338
Formatting Floating-Point Numbers . . . . .	338
Labeling Floating-Point Constants . . . . .	339
Storing Constants in the Read-Only Data Section . . . . .	339
Initializing Static Variables to 0.0 or -0.0 . . . . .	340
Putting It All Together . . . . .	340
Extra Credit: NaN . . . . .	342
Summary . . . . .	343
Additional Resources . . . . .	343

<b>14</b>		
<b>POINTERS</b>		<b>347</b>
Objects and Values . . . . .		348
Operations on Pointers . . . . .		349
Address and Dereference Operations . . . . .		349
Null Pointers and Type Conversions . . . . .		351
Pointer Comparisons . . . . .		352
& Operations on Dereferenced Pointers . . . . .		353
The Lexer . . . . .		353
The Parser . . . . .		354
Parsing Declarations . . . . .		356
Parsing Type Names . . . . .		361
Putting It All Together . . . . .		363
Semantic Analysis . . . . .		364
Type Checking Pointer Expressions . . . . .		365
Tracking Static Pointer Initializers in the Symbol Table . . . . .		369
TACKY Generation . . . . .		370
Pointer Operations in TACKY . . . . .		371
A Strategy for TACKY Conversion . . . . .		372
Assembly Generation . . . . .		375
Replacing Pseudoregisters with Memory Operands . . . . .		378
Fixing Up the lea and push Instructions . . . . .		378
Code Emission . . . . .		379
Summary . . . . .		380

<b>15</b>		
<b>ARRAYS AND POINTER ARITHMETIC</b>		<b>383</b>
Arrays and Pointer Arithmetic . . . . .		384
Array Declarations and Initializers . . . . .		384
Memory Layout of Arrays . . . . .		385
Array-to-Pointer Decay . . . . .		386
Pointer Arithmetic to Access Array Elements . . . . .		387
Even More Pointer Arithmetic . . . . .		389
Array Types in Function Declarations . . . . .		390
Things We Aren't Implementing . . . . .		391
The Lexer . . . . .		392
The Parser . . . . .		392
Parsing Array Declarators . . . . .		394
Parsing Abstract Array Declarators . . . . .		395
Parsing Compound Initializers . . . . .		396
Parsing Subscript Expressions . . . . .		396
The Type Checker . . . . .		398
Converting Arrays to Pointers . . . . .		398
Validating lvalues . . . . .		399
Type Checking Pointer Arithmetic . . . . .		400
Type Checking Subscript Expressions . . . . .		401
Type Checking Cast Expressions . . . . .		402
Type Checking Function Declarations . . . . .		402
Type Checking Compound Initializers . . . . .		403
Initializing Static Arrays . . . . .		404
Initializing Scalar Variables with ZeroInit . . . . .		405

TACKY Generation . . . . .	406
Pointer Arithmetic . . . . .	407
Subscripting . . . . .	408
Compound Initializers . . . . .	410
Tentative Array Definitions . . . . .	411
Assembly Generation . . . . .	411
Converting TACKY to Assembly . . . . .	414
Replacing PseudoMem Operands . . . . .	417
Fixing Up Instructions. . . . .	418
Code Emission. . . . .	418
Summary . . . . .	419

## **16 CHARACTERS AND STRINGS 423**

Character Traits . . . . .	424
String Literals . . . . .	425
Working with Strings in Assembly . . . . .	426
The Lexer . . . . .	429
The Parser. . . . .	431
Parsing Type Specifiers . . . . .	433
Parsing Character Constants. . . . .	433
Parsing String Literals. . . . .	433
Putting It All Together . . . . .	433
The Type Checker. . . . .	435
Characters . . . . .	435
String Literals in Expressions . . . . .	436
String Literals Initializing Non-static Variables . . . . .	437
String Literals Initializing Static Variables . . . . .	437
TACKY Generation. . . . .	440
String Literals as Array Initializers . . . . .	440
String Literals in Expressions . . . . .	441
Top-Level Constants in TACKY. . . . .	442
Assembly Generation . . . . .	443
Operations on Characters . . . . .	443
Top-Level Constants . . . . .	446
The Complete Conversion from TACKY to Assembly . . . . .	446
Pseudo-Operand Replacement . . . . .	448
Instruction Fix-Up. . . . .	449
Code Emission. . . . .	449
Hello Again, World!. . . . .	451
Summary . . . . .	454

## **17 SUPPORTING DYNAMIC MEMORY ALLOCATION 457**

The void Type . . . . .	458
Memory Management with void * . . . . .	460
Complete and Incomplete Types. . . . .	461
The sizeof Operator . . . . .	462
The Lexer . . . . .	463
The Parser. . . . .	463



The Type Checker . . . . .	467
Conversions to and from void * . . . . .	467
Functions with void Return Types . . . . .	469
Scalar and Non-scalar Types . . . . .	470
Restrictions on Incomplete Types . . . . .	471
Extra Restrictions on void . . . . .	473
Conditional Expressions with void Operands . . . . .	476
Existing Validation for Arithmetic Expressions and Comparisons . . . . .	476
sizeof Expressions . . . . .	477
TACKY Generation . . . . .	478
Functions with void Return Types . . . . .	479
Casts to void . . . . .	479
Conditional Expressions with void Operands . . . . .	479
sizeof Expressions . . . . .	480
The Latest and Greatest TACKY IR . . . . .	481
Assembly Generation . . . . .	482
Summary . . . . .	483

## 18

### STRUCTURES

**485**

Declaring Structure Types . . . . .	486
Structure Member Declarations . . . . .	488
Tag and Member Namespaces . . . . .	489
Structure Type Declarations We Aren't Implementing . . . . .	490
Operating on Structures . . . . .	491
Structure Layout in Memory . . . . .	492
The Lexer . . . . .	494
The Parser . . . . .	494
Semantic Analysis . . . . .	498
Resolving Structure Tags . . . . .	498
Type Checking Structures . . . . .	500
TACKY Generation . . . . .	512
Implementing the Member Access Operators . . . . .	513
Converting Compound Initializers to TACKY . . . . .	517
Structures in the System V Calling Convention . . . . .	519
Classifying Structures . . . . .	519
Passing Parameters of Structure Type . . . . .	522
Returning Structures . . . . .	525
Assembly Generation . . . . .	528
Extending the Assembly AST . . . . .	529
Copying Structures . . . . .	531
Using Structures in Function Calls . . . . .	532
Putting It All Together . . . . .	546
Replacing Pseudo-operands . . . . .	550
Code Emission . . . . .	551
Extra Credit: Unions . . . . .	552
Summary . . . . .	553
Additional Resources . . . . .	553

**19****OPTIMIZING TACKY PROGRAMS****557**

Safety and Observable Behavior . . . . .	558
Four TACKY Optimizations . . . . .	560
Constant Folding . . . . .	561
Unreachable Code Elimination . . . . .	561
Copy Propagation . . . . .	563
Dead Store Elimination . . . . .	564
With Our Powers Combined . . . . .	566
Testing the Optimization Passes . . . . .	569
Wiring Up the Optimization Stage . . . . .	570
Constant Folding . . . . .	573
Constant Folding for Part I TACKY Programs . . . . .	573
Supporting Part II TACKY Programs . . . . .	574
Control-Flow Graphs . . . . .	576
Defining the Control-Flow Graph . . . . .	577
Creating Basic Blocks . . . . .	578
Adding Edges to the Control-Flow Graph . . . . .	579
Converting a Control-Flow Graph to a List of Instructions . . . . .	580
Making Your Control-Flow Graph Code Reusable . . . . .	580
Unreachable Code Elimination . . . . .	581
Eliminating Unreachable Blocks . . . . .	581
Removing Useless Jumps . . . . .	582
Removing Useless Labels . . . . .	583
Removing Empty Blocks . . . . .	583
A Little Bit About Data-Flow Analysis . . . . .	584
Copy Propagation . . . . .	585
Reaching Copies Analysis . . . . .	589
Rewriting TACKY Instructions . . . . .	598
Supporting Part II TACKY Programs . . . . .	599
Dead Store Elimination . . . . .	603
Liveness Analysis . . . . .	604
Removing Dead Stores . . . . .	608
Supporting Part II TACKY Programs . . . . .	608
Summary . . . . .	610
Additional Resources . . . . .	610

**20****REGISTER ALLOCATION****613**

Register Allocation in Action . . . . .	614
Take One: Put Everything on the Stack . . . . .	615
Take Two: Register Allocation . . . . .	616
Take Three: Register Allocation with Coalescing . . . . .	618
Updating the Compiler Pipeline . . . . .	619
Extending the Assembly AST . . . . .	620
Converting TACKY to Assembly . . . . .	621
Register Allocation by Graph Coloring . . . . .	622

Detecting Interference . . . . .	626
Spilling Registers . . . . .	627
The Basic Register Allocator . . . . .	630
Handling Multiple Types During Register Allocation . . . . .	631
Defining the Interference Graph . . . . .	631
Building the Interference Graph . . . . .	631
Calculating Spill Costs . . . . .	638
Coloring the Interference Graph . . . . .	638
Building the Register Map and Rewriting the Function Body . . . . .	646
Instruction Fix-Up with Callee-Saved Registers . . . . .	648
Code Emission . . . . .	649
Register Coalescing . . . . .	651
Updating the Interference Graph . . . . .	653
Conservative Coalescing . . . . .	656
Implementing Register Coalescing . . . . .	663
Summary . . . . .	668
Additional Resources . . . . .	669

**NEXT STEPS 671**

Add Some Missing Features . . . . .	671
Handle Undefined Behavior Safely . . . . .	672
Write More TACKY Optimizations . . . . .	672
Support Another Target Architecture . . . . .	672
Contribute to an Open Source Programming Language Project . . . . .	673
That's a Wrap! . . . . .	673

**A  
DEBUGGING ASSEMBLY CODE WITH GDB OR LLDB 675**

The Program . . . . .	676
Debugging with GDB . . . . .	677
Configuring the GDB UI . . . . .	678
Starting and Stopping the Program . . . . .	679
Printing Expressions . . . . .	682
Examining Memory . . . . .	684
Setting Conditional Breakpoints . . . . .	685
Getting Help . . . . .	686
Debugging with LLDB . . . . .	687
Starting and Stopping the Program . . . . .	688
Displaying Assembly Code . . . . .	691
Printing Expressions . . . . .	692
Examining Memory . . . . .	694
Setting Conditional Breakpoints . . . . .	695
Getting Help . . . . .	697

**B  
ASSEMBLY GENERATION AND CODE EMISSION TABLES 699**

Part I . . . . .	699
Converting TACKY to Assembly . . . . .	700
Code Emission . . . . .	701

Part II .....	704
Converting TACKY to Assembly .....	704
Code Emission .....	711
Part III .....	715

<b>REFERENCES</b>	<b>725</b>
-------------------	------------

<b>INDEX</b>	<b>731</b>
--------------	------------