

Table of Contents

Preface

xxv

Part 1: Getting Started with LLVM

1

Chapter 1: Building LLVM and Understanding the Directory Structure

3

Getting the most out of this book – get to know your free benefits	4
Technical requirements	5
Getting ready for LLVM’s world	6
Prerequisites • 7	
Identifying the right version of the tools • 8	
Installing the right tools • 9	
Building a compiler	11
What is a compiler? • 11	
Opening Clang’s hood • 12	
Building Clang • 13	
Experimenting with Clang • 14	
Building LLVM	15
Configuring the build system • 15	
Crash course on Ninja • 18	
Building the core LLVM project • 20	
Testing a compiler	21
Crash course on the Google test infrastructure • 21	
Crash course on the LLVM Integrated Tester • 21	
<i>Testing in lit</i> • 22	
<i>Directives</i> • 22	
<i>Describing the RUN command</i> • 23	

<i>The lit driver – llvm-lit</i>	• 24
Crash course on FileCheck	• 25
<i>FileCheck by example</i>	• 26
LLVM unit tests	• 28
<i>Finding the source of a test</i>	• 29
<i>Running unit tests manually</i>	• 29
<i>The unit tests pass, what now?</i>	• 32
LLVM functional tests	• 32
<i>The LLVM test suite</i>	• 33
<i>The functional tests fail – what do you do?</i>	• 34
Understanding the directory structure	36
High-level directory structure	• 36
Focusing on the core LLVM project	• 36
A word on the include files	• 37
<i>Private headers</i>	• 37
<i>What is the deal with <project>/include/<project>?</i>	• 37
<i>What is include/<project>-c?</i>	• 38
Overview of some of the LLVM components	• 38
<i>Generic LLVM goodness</i>	• 38
<i>Working with the LLVM IR</i>	• 38
<i>Generic backend infrastructure</i>	• 39
<i>Target-specific constructs</i>	• 39
Summary	39
Quiz time	40
Chapter 2: Contributing to LLVM	45
<hr/>	
Reporting an issue	45
Engaging with the community	47
Reviewing patches	• 48
Contributing patches	49
Understanding patch contribution in a nutshell	• 50
Following up with your contribution	• 51
A word on adding tests	• 51
Summary	52
Quiz time	52

Technical requirements	55
A word on APIs	56
Understanding compiler jargon	57
Target • 58	
Host • 58	
Lowering • 58	
Canonical form • 58	
Build time, compile time, and runtime • 58	
Backend and middle-end • 59	
Application binary interface • 59	
Encoding • 60	
Working with basic structures	60
Module • 61	
<i>A module at the LLVM IR level • 61</i>	
<i>A module at the Machine IR level • 62</i>	
Function • 63	
<i>A function in the LLVM IR • 64</i>	
<i>A function in the Machine IR • 64</i>	
Basic block • 65	
<i>A basic block in the LLVM IR • 66</i>	
<i>A basic block in the Machine IR • 67</i>	
Instruction • 68	
An instruction in the LLVM IR • 69	
<i>An instruction in the Machine IR • 69</i>	
Control flow graph • 70	
<i>Reverse post-order traversal • 71</i>	
<i>Backedge • 72</i>	
<i>Critical edge • 72</i>	
<i>Irreducible graph • 73</i>	
Building your first IRs	75
Building your first LLVM IR • 76	
<i>A walk over the required APIs • 76</i>	
<i>Your turn • 77</i>	

Building your first Machine IR • 78	
<i>A walk over the required APIs • 78</i>	
<i>Your turn • 80</i>	
Summary	81
Quiz time	82
Chapter 4: Writing Your First Optimization	85
Technical requirements	85
The concept of value	86
SSA • 87	
Constructing the SSA form • 88	
Dominance • 91	
Def-use and use-def chains • 94	
<i>Def-use and use-def chains in the LLVM IR • 94</i>	
<i>Def-use and use-def chains in the Machine IR • 97</i>	
Tackling optimizations	99
Legality • 99	
<i>Integer overflow/underflow • 100</i>	
<i>Fast-math flags • 100</i>	
<i>Side effects • 101</i>	
Profitability • 102	
<i>Instruction lowering – TargetTransformInfo and TargetLowering • 104</i>	
<i>Library support – TargetLibraryInfo • 105</i>	
<i>Datatype properties – DataLayout • 105</i>	
<i>Register pressure • 105</i>	
<i>Basic block frequency • 106</i>	
<i>More precise instruction properties – scheduling model and instruction description • 107</i>	
Transformation jargon • 107	
<i>Instcombine • 107</i>	
<i>Fixed point • 108</i>	
<i>Liveness • 108</i>	
<i>Hoisting • 108</i>	
<i>Sinking • 108</i>	
<i>Folding • 108</i>	

Loops	109
Terminology • 109	
Preheader • 109	
Header • 110	
Exiting block • 110	
Latch • 110	
Exit block • 110	
Where to get loop information • 110	
Writing a simple constant propagation optimization	110
The optimization • 111	
Simplifying assumptions • 111	
Missing APIs • 111	
The Constant class • 111	
The APInt class • 112	
Creating a constant • 112	
Replacing a value • 113	
Your turn • 113	
Going further • 113	
Legality • 113	
Profitability • 114	
Propagating constants across types • 114	
Summary	115
Quiz time	115
Chapter 5: Dealing with Pass Managers	119
Technical requirements	119
What is a pass?	120
What is a pass manager?	121
The legacy and new pass manager	122
Pass managers' capabilities • 122	
Populating a pass manager • 123	
Inner workings of pass managers • 124	
Creating a pass	125
Writing a pass for the legacy pass manager • 125	
Using the proper base class • 125	
Expressing the dependencies of a pass • 127	

<i>Preserving analyses</i> • 130	
<i>Specificities of the Pass class</i> • 131	
Writing a pass for the new pass manager • 132	
<i>Implementing the right method</i> • 132	
<i>Registering an analysis</i> • 133	
<i>Describing the effects of your pass</i> • 134	
Inspecting the pass pipeline	135
Available developer tools • 135	
Plumbing up the information you need • 136	
Interpreting the logs of pass managers • 138	
<i>The pass pipeline structure</i> • 138	
<i>Time profile</i> • 141	
Your turn	142
Writing your own pass • 142	
Writing your own pass pipeline • 143	
Summary	144
Further reading	144
Quiz time	145

Chapter 6: TableGen – LLVM Swiss Army Knife for Modeling **147**

Technical requirements	148
Getting started with TableGen	148
The TableGen programming language	150
Types • 151	
Programming with TableGen • 152	
Defining multiple records at once • 153	
Assigning fields • 155	
Discovering a TableGen backend	157
General information on TableGen backends for LLVM • 157	
Discovering a TableGen backend • 159	
<i>The implementation of intrinsics</i> • 159	
<i>The content of a generated file</i> • 159	
<i>The source of a TableGen backend</i> • 160	
Debugging the TableGen framework	162
Identifying the failing component • 162	
Cracking open a TableGen backend • 163	

Summary	164
Further reading	164
Quiz time	165
Part 2: Middle-End: LLVM IR to LLVM IR	167
Chapter 7: Understanding LLVM IR	169
Technical requirements	170
Understanding the need for an IR	170
What an IR is • 170	
Why use an IR? • 170	
Introducing LLVM IR	171
Identifiers • 171	
Functions • 172	
Basic blocks • 175	
Instructions • 175	
Types • 176	
Single-value types • 177	
The label type • 180	
Aggregate types • 180	
Types in the LLVM IR API • 183	
Walking through an example	184
Target-specific elements in LLVM IR	186
Intrinsic functions • 186	
Triple • 188	
Function attributes • 189	
Data layout • 189	
Application binary interface • 193	
Textual versus binary format	195
LLVM IR API – cheat sheet	196
Summary	198
Further reading	198
Quiz time	199

Chapter 8: Survey of the Existing Passes 201

Technical requirements	202
How to find the unknown	202
Leveraging opt • 202	
Using the LLVM code base • 205	
Starting from the implementation • 206	
Survey of the helper passes	207
The verifier • 207	
The printer • 208	
Analysis passes • 210	
<i>Target transformation information</i> • 210	
<i>Loop information</i> • 211	
<i>Alias analysis</i> • 211	
<i>Block frequency info</i> • 212	
<i>Dominator tree information</i> • 212	
<i>Value tracking</i> • 213	
Canonicalization passes	213
The instruction combiner • 215	
<i>An example of a canonical rewrite</i> • 215	
<i>An example of an optimization</i> • 216	
<i>How to use instcombine</i> • 216	
The memory to register rewriter • 217	
The converter to loop-closed-SSA form • 217	
Optimization passes	220
Interprocedural optimizations • 221	
Scalar optimizations • 224	
Vectorization • 233	
Summary	236
Further reading	236
Quiz time	237

Chapter 9: Introducing Target-Specific Constructs 239

Technical requirements	240
Adding a new backend in LLVM	241
Connecting your target to the build system • 241	
Registering your target with Clang • 245	

<i>Adding a new architecture to the Triple class</i> • 246	
<i>Populating the Target instance</i> • 247	
<i>Plumbing your Target through Clang</i> • 251	
Creating your own intrinsics	256
The pros and cons of intrinsics • 256	
Creating an intrinsic in the backend • 258	
<i>Defining our intrinsics</i> • 259	
<i>Hooking up the TableGen backend</i> • 261	
<i>Teaching LLVM IR about our intrinsics</i> • 262	
Connecting an intrinsic to Clang • 262	
<i>Writing the .def file by hand</i> • 263	
<i>Using the TableGen capabilities</i> • 265	
<i>Hooking up the built-in information</i> • 266	
Establishing the code generation link • 267	
Adding a target-specific TargetTransformInfo implementation	269
<i>Establishing a connection to your target-specific information</i> • 269	
Introducing target-specific costs • 270	
Customizing the default middle-end pipeline	272
Using the new pass manager • 272	
Using the legacy pass manager • 274	
A one-time setup – assembling a codegen pipeline • 275	
<i>Faking the instruction selector</i> • 276	
<i>Faking the lowering of the object file</i> • 277	
<i>Creating a skeleton for the assembly information</i> • 278	
Using the right abstraction	281
Summary	282
Further reading	282
Quiz time	283
Chapter 10: Hands-On Debugging LLVM IR Passes	285
<hr/>	
Technical requirements	285
The logging capabilities in LLVM	286
Printing the IR between passes • 286	
Printing the debug log • 287	
Printing high-level information about what happened • 289	

Reducing the input IR size	291
Extracting a subset of the input IR • 291	
Shrinking the IR automatically • 293	
Using sanitizers	296
A crash course on LLDB	297
Starting a debugging session • 298	
Controlling the execution • 299	
<i>Stopping the program</i> • 299	
<i>Command resolution</i> • 301	
<i>Resuming the execution</i> • 302	
<i>Inspecting the state of a program</i> • 303	
The LLVM code base through a debugger	305
Summary	307
Further reading	307
Quiz time	308

Part 3: Introduction to the Backend **311**

Chapter 11: Getting Started with the Backend **313**

Technical requirements	313
Introducing the Machine IR	314
Here comes the Machine IR	314
The Machine IR textual representation	315
The .mir file format • 315	
A primer on the YAML syntax • 316	
The semantics of the different fields • 318	
<i>Mapping the content of a .mir file to the C++ API</i> • 318	
<i>A deep dive into the body of a MachineFunction instance</i> • 319	
Working with a .mir file • 322	
<i>Generating a .mir file</i> • 322	
<i>Running passes</i> • 323	
<i>Shrinking a .mir file</i> • 325	
The anatomy of a MachineInstr instance	327
Introducing the MC layer • 328	
Working with MachineOperand instances • 329	
<i>Unboxing a MachineOperand instance</i> • 329	

<i>Dealing with explicit and implicit operands</i> • 331	
<i>Understanding the constraints of an operand</i> • 332	
Working with registers	333
The concept of the register class • 335	
The concept of sub-registers • 335	
The concept of register tuples • 336	
The concept of register units • 337	
The registers and SSA and non-SSA forms • 339	
Interacting with registers in the debugger • 340	
Creating MachineInstr objects	342
Describing registers	344
Writing the target description • 344	
Describing instructions	350
Summary	353
Further reading	353
Quiz time	353
Chapter 12: Getting Started with the Machine Code Layer	357
Technical requirements	358
The use of the MC layer	358
Connecting the MC layer	359
What instructions to describe • 360	
Augmenting the target description with MC information • 360	
<i>Defining the MC layer for the registers</i> • 361	
<i>Defining the MC layer for the instructions</i> • 361	
Enabling MC-based tools	365
Leveraging TableGen • 366	
Implementing the missing pieces • 368	
<i>Implementing your own MCInstPrinter class</i> • 368	
<i>Implementing your own MCCodeEmitter class</i> • 370	
<i>Implementing your own XXXAsmParser class</i> • 371	
Summary	372
Quiz time	372

Chapter 13: The Machine Pass Pipeline **375**

Technical requirements	376
The Machine pass pipeline at a glance	376
Injecting passes	377
Using the generic Machine optimizations	378
Generic passes worth mentioning	380
The CodeGenPrepare pass • 380	
The PeepholeOptimizer pass • 381	
The MachineCombiner pass • 381	
Summary	382
Further reading	382
Quiz time	382

Part 4: LLVM IR to Machine IR **385**

Chapter 14: Getting Started with Instruction Selection **387**

Technical requirements	388
Overview of the instruction selection frameworks	388
How does instruction selection work? • 389	
Framework complementarity • 389	
Overall differences between the selectors • 391	
<i>Compile time</i> • 391	
<i>Modularity and testability</i> • 391	
<i>Scope</i> • 392	
Which selector to use?	394
FastISel • 394	
SDISel • 394	
GlobalISel • 395	
Selectors' inner workings	395
Understanding the DAG representation	397
Textual representation of the SelectionDAG class • 398	
Manipulating a DAG • 400	
Understanding the generic Machine IR	401
Textual representation of generic attributes • 402	
Lowering constraints of the generic Machine IR • 403	
APIs to work with the generic Machine IR • 404	

Groundwork to connect the codegen pipeline	405
Instantiating the codegen pass pipeline • 406	
Providing the key target APIs to the codegen pipeline • 408	
Connecting SDISel to the codegen pipeline • 408	
Connecting FastISel to the codegen pipeline • 410	
Connecting GlobalISel to the codegen pipeline • 412	
Choosing between different selectors • 413	
Summary	414
Further reading	414
Quiz time	414

Chapter 15: Instruction Selection: The IR Building Phase **417**

Technical requirements	418
Overview of the IR building	418
Describing the calling convention	420
Writing your target description of the calling convention • 421	
Connecting the gen-callingconv TableGen backend • 423	
Anatomy of the CCValAssign class • 423	
Lowering the ABI with SDISel	424
Implementing the lowering of formal arguments • 425	
Providing custom description for the SDNode class • 428	
Handling of stack locations • 431	
Lowering the ABI with FastISel	432
Lowering the ABI with GlobalISel	434
Summary	439
Further reading	439
Quiz time	440

Chapter 16: Instruction Selection: The Legalization Phase **441**

Technical requirements	442
Legalization overview	442
Legalization actions	443
Legalization in SDISel	446
Describing your legal types • 446	
Describing your legalization actions • 447	
Implementing a custom legalization action • 448	

Legalization in GlobalISel	450
Describing your legalization actions with the LegalizeRuleSet class • 451	
Custom legalization in GlobalISel • 454	
Summary	456
Quiz time	456
Chapter 17: Instruction Selection: The Selection Phase and Beyond	459
<hr/>	
Technical requirements	460
Register bank selection	460
The goal of the register bank selection • 460	
Describing the register banks • 461	
Implementing your RegisterBankInfo class • 463	
Instruction selection	467
Expressing your selection patterns • 467	
<i>Introduction to the selection patterns</i> • 468	
<i>Advanced selection patterns</i> • 470	
Selection in SDISel • 474	
Selection in FastISel • 474	
Selection in GlobalISel • 475	
<i>Setting up the InstructionSelector class</i> • 475	
<i>Importing the selection patterns</i> • 477	
<i>Going beyond patterns</i> • 479	
Finalizing the selection pipeline	481
Using custom inserters • 481	
Customizing the TargetLowering::finalizeLowering method • 482	
Optimizations	483
Using the DAGCombiner framework • 483	
Leveraging the combiner framework • 484	
Debugging the selectors	489
Debugging SDISel • 489	
Debugging the GlobalISel match table • 491	
Summary	493
Quiz time	494

Chapter 18: Instruction Scheduling**499**

Technical requirements	500
Overview of the instruction scheduling framework	500
The ScheduleDAGInstrs class	502
Changing the scheduling algorithm	503
The scheduling model	507
The scheduling events • 509	
The processing units • 510	
The scheduling bindings • 511	
Gluing everything together • 514	
Implementing your scheduling model	515
Connecting your scheduling model • 515	
<i>Describing a processor model • 515</i>	
<i>Instantiating your subtarget • 516</i>	
Guidelines to get started with your scheduling model • 516	
Summary	518
Quiz time	518

Chapter 19: Register Allocation**521**

Technical requirements	522
Overview of register allocation in LLVM	522
Enabling the register allocation infrastructure	524
Introducing the slot indexes	527
Introducing the live intervals	530
Maintaining the live intervals	533
Summary	534
Further reading	534
Quiz time	534

Chapter 20: Lowering of the Stack Layout**537**

Technical requirements	538
Overview of stack lowering	538
Handling of stack slots • 538	
From frame index to stack slot • 539	

The lowering of the stack frame	540
Introducing the reserved call frame • 540	
Implementing the frame-lowering target hooks • 542	
The expansion of the frame indices	544
Introducing register scavenging • 545	
Provisioning an emergency spill slot • 546	
Expanding the frame indices • 547	
Summary	549
Quiz time	550

Chapter 21: Getting Started with the Assembler **553**

Technical requirements	554
Overview of the lowering of a textual assembly file	554
Assembling with the LLVM infrastructure	556
Implementing an assembler	558
Providing the MCodeEmitter class • 558	
Handling the fixups with the MCodeBackend class • 559	
Recording the relocations with the MCodeTargetWriter class • 561	
Summary	564
Further reading	564
Quiz time	565

Chapter 22: Unlock Your Book's Exclusive Benefits **567**

How to unlock these benefits in three easy steps	567
--	-----

Other Books You May Enjoy **570**

Index **575**
