Table of Contents

1 Introduction **1**
1.1 Abstract 2
1.2 Goals 3
1.3 Planning 4

2 Research **5**
2.1 Choosing a suitable language frontend 6
2.1.1 Clang (C) 7
2.1.2 GCC (GNU Compiler Collection, C) 8
2.1.3 Other options 9
2.1.4 Custom Frontend 10
2.1.5 Conclusion 11
2.2 Choosing a suitable target platform 12
2.2.1 ARM 13
2.2.2 X86/x86_64 14
2.2.3 Conclusion 15
2.3 Basic Compiler Theory 16
2.3.1 The 3 Stages of Compilation 17
2.3.1.1 Front end 18
2.3.1.2 Middle end 19
2.3.1.3 Back end 20
2.3.2 Intermediate Representations (IR) 21
2.3.2.1 SSA 22
2.4 Investigation into existing optimising compilers 23
2.4.1 Clang/LLVM 24
2.4.2 GCC 25
2.5 Code Optimisation Techniques 26
2.5.1 Register Allocation 27
2.5.2 Peephole optimisations 28
2.5.3 Constant folding 29
2.6 Tools 30

3 Development **31**
3.1 Compiler Structure & Design 32
3.2 Implementing a language frontend 33
3.2.1 Using the LibClang libraries 34
3.3 Implementing a basic ARM code generator 35
3.3.1 Considerations for the target platform 36
3.3.2 Getting from IR to an executable binary 37
3.4 Benchmarking and Optimising 38
3.4.1 Register Allocation 39
3.4.1.1 Ad Hoc (Spill Only) 40
3.4.1.2 Linear Scan 41
3.4.2 Analysis and improving the performance of specific benchmarks 42
3.4.2.1 String Length (strlen) 43
3.2.2.2 <Any Others> 44
3.5 Testing 45
3.5.1 Integration Tests 46
3.5.2 Unit Tests 47
3.5.3 External Testsuites 48
3.5.4 Debugging 49
3.6 Custom Tools 50
3.6.1 Testify (Integration Testing) 51
3.6.2 MachineDescription (C++ code generator for defining machine instructions) 52
3.6.3 Benchmarking & native testing infrastructure 53

4 Evaluation **54**
4.1 Retrospective on the structure and design of the compiler 55
4.1.1 IR 56
4.2 Development process 57
4.3 Testing & correctness 58
4.4 Limitations and Future developments 59
4.5 Conclusion 60
4.5.1 Personal Development 61

5 References **62**
6 Glossary **63**
7 Appendices **64**