Table of Contents

**1 Introduction1**

1.1 Abstract 2

1.2 Goals3

1.3 Planning4

**2 Research5**

2.1 Choosing a suitable language frontend6

2.1.1 Clang (C)7

2.1.2 GCC (GNU Compiler Collection, C)8

2.1.3 Other options9

2.1.4 Custom Frontend10

2.1.5 Conclusion11

2.2 Choosing a suitable target platform12

2.2.1 ARM13

2.2.2 X86/x86\_6414

2.2.3 Conclusion15

2.3 Basic Compiler Theory16

2.3.1 The 3 Stages of Compilation17

2.3.1.1 Front end18

2.3.1.2 Middle end19

2.3.1.3 Back end20

2.3.2 Intermediate Representations (IR)21

2.3.2.1 SSA22

2.4 Investigation into existing optimising compilers23

2.4.1 Clang/LLVM24

2.4.2 GCC25

2.5 Code Optimisation Techniques26

2.5.1 Register Allocation27

2.5.2 Peephole optimisations28

2.5.3 Constant folding29

2.6 Tools30

**3 Development31**

3.1 Compiler Structure & Design32

3.2 Implementing a language frontend33

3.2.1 Using the LibClang libraries34

3.3 Implementing a basic ARM code generator35

3.3.1 Considerations for the target platform36

3.3.2 Getting from IR to an executable binary 37

3.4 Benchmarking and Optimising38

3.4.1 Register Allocation39

3.4.1.1 Ad Hoc (Spill Only)40

3.4.1.2 Linear Scan41

3.4.2 Analysis and improving the performance of specific benchmarks42

3.4.2.1 String Length (strlen)43

3.2.2.2 <Any Others>44

3.5 Testing45

3.5.1 Integration Tests46

3.5.2 Unit Tests47

3.5.3 External Testsuites48

3.5.4 Debugging49

3.6 Custom Tools50

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 infrastructure53

**4 Evaluation54**

4.1 Retrospective on the structure and design of the compiler55

4.1.1 IR56

4.2 Development process57

4.3 Testing & correctness58

4.4 Limitations and Future developments59

4.5 Conclusion 60

4.5.1 Personal Development61

**5 References62**

**6 Glossary63**

**7 Appendices64**