Writing a Compiler

Andrew DeSimone
Samford University
800 Lakeshore Drive
Birmingham, AL 35229
+1 201 252 5848
adesimon@samford.edu

Brian Toone
Samford University
800 Lakeshore Drive
Birmingham, AL 35229
+1 205 726 2960
brtoone@samford.edu

Abstract

Computer programs written by humans need to be translated into a machine readable assembly language. This project will carefully define a simple programming language, implement a compile, and build more complex language features.

Keywords: compilers, lexing, parsing, ir, assembly

1 Background

A compiler is a computer program which translates one programming language to another, most often this is from a human readable language to a machine readable language. These compilers allow programmers to abstract away hardware details and write powerful programs with simpler language.

2 Methodology

2.1 Language Design

Programming languages can be defined by their context free grammar and their lexical grammar. Context free grammars describe sentences in the language and lexical grammars describe lexemes. For example a number is a lexeme described by the regular expression [0-9]+ and a program is a sentence described by the context free grammar begin block end. This project will entail building a grammar for a bare bones language.

2.2 Frontend

The frontend of a compiler is made up of a lexer and parser. The lexer takes a program which is represented by a string and turns it into a stream of lexemes which fully describe the program. The parser takes the stream of lexemes and transforms it into an abstract syntax tree which represents the program.

2.3 Backend

The first part of a compiler's backend will involve translating the abstract syntax tree into an intermediate representation. This intermediate representation will be optimized and then used to produce machine code. In this project the machine code will be RISC-V due to its relatively small instruction set. For the purpose of this project the RISC-V machine will be a docker container running Debian.

2.4 Benchmark

In order to measure the performance of this language two factors will be considered. Firstly, the performance of the optimizer. By measuring how long an optimized program takes compared to an optimized program we can verify that the optimizer improves runtime significantly. The other factor will be a comparison with other languages. Runtime will be compared with an interpreted language like Python, compiled program like C, and a jit language like Java.

3 Summary

A successful project would result in a program which can take human written code and transform it to machine readable code. This would be done through a compilation process of lexing, parsing, intermediate representation, optimization, and code generation. These human written programs would be able to be run on a risc-v linux machine.

4 References

[1] Keith Cooper and Linda Torczon. Engineering a Compiler. [2] davidburela/riscv-emulator - Docker Image — Docker Hub. Retrieved December 9, 2024 from https://hub.docker.com/r/davidburela/riscv-emulator [3] Debian – The Universal Operating System. Retrieved December 9, 2024 from https://www.debian.org/