University of Southern Denmark | IMADA
Saturday 15th April, 2023

Compiler for Panda
BADM500: Bachelor Project

Author
KIAN BANKE LARSEN
kilar20@student.sdu.dk

Supervisor
KIM SKAK LARSEN
Professor

Abstract

English This is my very good abstract

Danish Et fantastisk abstract

Contents

1 Introduction

This report examines how to make a simple compiler in Python. The compiler is simple in the sense that some decisions have been made to ease the process, despite the choice, although the decisions are not necessarily optimal. The aim is to learn different compiler techniques and get a hands-on feel for the different compiler phases by actually implementing a working compiler, targeting X86 assembler, from scratch, using a Flex/Bison equivalent package such as `PLY` for scanning and parsing.

The language to be compiled is a subset of the imperative language C. This has been chosen because of its simpler syntax and easy to read curly bracket enclosed static scopes. In this project, we are interested in making a language having integers, Booleans and preferably some kind of floats. The language must have control flow constructs in form of `if-else` statements and functions, and iterative constructs such as `for`- and `while`-loops.

A modern compiler is, as is well known, divided into phases. These phases relate to lexical and syntactic analysis, resulting in an abstract syntax tree. Subsequent phases analyze and adorn the abstract syntax tree, building a symbol table and finally generating assembler code.

The main focus in regard to advanced techniques will be local register allocation, using techniques described in Copper and Torczon 2022. Handling this efficiently requires data flow analysis via control flow-graph, construction of interference graph, graph coloring and translation back to instructions using a combination of the registers and the stack, when the available registers do not suffice.

Initially, a stack machine will be prepared, which will form the basis for developing a compiler that uses CPU registers. We will take advantage of the split phases property when replacing the stack code generation phase in benefit for one that uses register allocation. This allows us to only worry about ensuring that subsequent phases cope with the changes made in the former phases.

When adding extra complexity such as register allocation, it is important to document the benefit of this choice. Performance of the stack machine and the register machine will therefore be constructively compared.

2 Project Basics

2.1 Project Structure

2.2 Design Patterns

2.3 Compiler Usage

```
Compiler$ python3.10 main.py --help

usage: Compiler for Panda [-h] [-o OUTPUT] [-c] [-d] [-f FILE] [-t] [-r]

Compiles source code to assembly

options:
    -h, --help            show this help message and exit
    -o OUTPUT, --output OUTPUT
                          Specify name of assembly output file
    -c, --compile         Set this flag if the output file should be compilled
    ↪  with gcc
    -d, --debug           Set this flag for debugging information, i.e., ILOC and
    ↪  Graphviz
    -f FILE, --file FILE  Path to input file, otherwise stdin will be used
    -t, --runTests        Run tests
    -r, --run             Run compilled program
```

3 Phases

4 Testing

5

Performance Comparison

6 Evaluation

7 Conclusion

Bibliography

Copper, Keith D. and Linda Torczon (Oct. 2022). *Engineering a Compiler*. 3. Morgan Kaufmann publishers. ISBN: 978-0-12-815412-0.