CS335: Milestone 3 Documentation

Roll no.- 210750, 210557, 210543

April 17, 2024

1 Introduction

This document provides an overview of the implementation of the x86_64 assembly code generator for the Python compiler developed in Milestone 2.

2 Supported Language Features

All features are supported till 3ac generation of code, but we haven't implemented the x86 conversion for classes and lists due to time constraints.

2.1 Register Allocation

The assembly code generator uses a local register allocation scheme to efficiently utilize the available registers. It keeps track of the variables currently stored in registers and spills them to the stack when necessary to make room for new variables.

2.2 Function Calls

The assembly code generator supports function calls by correctly setting up the stack frame, preserving the necessary registers, and handling the function return value.

2.3 Handling Data Types

The assembly code generator correctly handles the different data types (integers and strings) by using the appropriate x86_64 instructions and memory layouts.

3 Compilation Instructions

To compile the project, use the `make` command. It involves the following steps:

1. Generate the parser using Bison:

   ```
   bison -d -o parser.cpp parser.y
   ```

2. Generate the scanner using Flex:

   ```
   flex -o scanner.cpp lexer.l
   ```

3. Compile the source files:

   ```
   g++ parser.cpp scanner.cpp ast.cpp symtab.cpp -o parse -std=c++11
   ```

4 Execution Instructions

To execute the compiled program, following command line tools are supported By default, we can execute using:

1. `./parser path-to-input-file`
2. `g++ -o chd x86.cpp`
3. `./chd --input=3ac.txt --output=5.s`
4. `gcc 5.s`

5 Testing and Evaluation

The assembly code generator has been tested with the following test cases:

1. `test1.py`
2. `test2.py`
3. `test3.py`
4. `test4.py`
5. `test5.py`

6 Modifications from Milestone 2

The 3AC representation generated in Milestone 2 has been modified to remove specific errors for some manual test cases, and also it's little bit optimised;

7 Manual Changes to the Assembly Code

Minor changes were required to the generated assembly code to run it successfully with the GNU Assembler (as) or the GCC compiler.

8 Unsupported Features and Challenges

The current implementation does not support the following features:

- Floating-point data type
- Lists, tuples
- Classes and object-oriented programming

9 Effort Sheet

The total effort for Milestone 3 was approximately 120 person-hours, distributed as follows:

- Prashant (210750) : 33.33
- Lakshvant (210557): 33.33
- Harsh (210543) : 33.33