README

Group Information

Group Name: [T06_G14]

Student Number: UP202108876

Full Name: Lara Inês Alves Cunha

Contribution

Lara Inês Alves Cunha: 100%

Project Overview

This project is written in Haskell and consists of two main parts.

Part 1: Assembly Language Interpreter

In this part, the program provides an interpreter for a simple assembly language. The assembly language includes basic arithmetic and logical operations, control flow instructions such as conditional statements and loops, and the ability to manipulate a stack and variables.

Key Definitions

Inst: Data type representing assembly language instructions.

Code: Type alias for a list of assembly instructions.

Stack: Type alias for a stack of integers.

State: Type alias for a list of variable-value pairs.

Functions

run: Executes a given assembly code, stack, and state, producing a new code, stack, and state.

performBinaryOperation: Helper function to perform binary operations on the stack.

lookupVar, lookupVarBool: Helper functions to look up variables in the state.

updateVar: Helper function to update a variable in the state.

store: Helper function to store a variable in the state.

testAssembler: Function to test the assembler with predefined code examples.

Example Usage

– Example 1: Arithmetic operations testAssembler [Push 10, Push 4, Push 3, Sub, Mult] – Output: (“-10”,““)

– Example 2: Conditional statements and variable manipulation testAssembler [Fals, Push 3, Tru, Store “var”, Store “a”, Store “someVar”] – Output: (““,”a=3,someVar=False,var=True”)

Part 2: Simple Imperative Language Compiler

This part involves defining types and functions for a simple imperative language. The language includes arithmetic expressions, boolean expressions, and statements such as assignment, sequence, conditional statements, and loops.

Key Definitions

Aexp: Data type representing arithmetic expressions.

Bexp: Data type representing boolean expressions.

Stm: Data type representing statements.

Functions

compA: Compiles an arithmetic expression into assembly code.

compB: Compiles a boolean expression into assembly code.

compile: Compiles a list of statements into assembly code.

Example Usage

– Example: Compile and run a simple program testParser “x := 5; x := x - 1;” – Output: (““,”x=4”)

Strategy

Part 1

Defined data types (Inst, Code, Stack, State) to represent the assembly language components. Implemented functions for basic operations (arithmetic, logical) and control flow. Created helper functions for stack and variable manipulation. Tested the assembler with predefined examples.

Part 2

Defined data types (Aexp, Bexp, Stm) to represent the imperative language components. Implemented functions to compile arithmetic and boolean expressions. Developed a compiler function to translate a list of statements into assembly code. Tested the compiler with predefined examples.

Conclusion

The project provides a comprehensive solution for interpreting a simple assembly language and compiling a basic imperative language into assembly code. The defined data types and functions facilitate code organization and readability. The examples demonstrate the correctness and functionality of the implemented interpreter and compiler.