WACC Compiler: Project Report

Paul Vidal, Saturnin Pugnet, Gregoire Yharrassarry and Corentin Herbinet

December 14, 2015

1 Introduction

- Present WACC Language
- Objective of the Project

2 Product: Quality of our WACC Compiler

2.1 Functional Correctness

- Valid programs compile correctly, create Assembly file that produces expected output when run
- Syntactically or semantically invalid programs do not compile, corresponding error message is displayed

2.2 Future Development

- Does our compiler allow easy implementation of new WACC language features?
- Does our compiler allow easy implementation of Assembly code optimisation?

2.3 Performance Issues

- Any slow, redundant processes?

3 Project Management

3.1 Organisation of our Group

- Splitting up the work

- Group meetings and coding sessions

3.2 Use of Project Management Tools

- Version control with Git
- Communication through Slack
- Tasks using Trello

3.3 Reflecting on our Project Management

- What went well
- What we would do differently

4 Design Choices

4.1 Syntax Analysis Design Choices

- Using a Listener to detect syntax errors and print out a precise error message
- Extending the basic ErrorStrategy to personalize the error messages
- Using a Visitor to check all functions have a return statement

4.2 Semantic Analysis Design Choices

- Using a visitor to check semantic consistency of the program
- Creating two HashMaps to link all functions to number of arguments and type, and to link all variables to their type

4.3 Code Generation Design Choices

- Hardware Manager singleton pattern to manage registers and position of variables in memory/stack
- Command interface with hierarchy of superclasses to be able to build all useful ARM commands
- Factory pattern to build the statements and expressions
- Expression, AssignRhs, AssignLhs and Statement interfaces which all have common generateCommands() method

- FileCreator and FileWriter classes which take a list of Commands and write them in the right Assembly file

5 Our Extensions

5.1 Function Overloading

5.2 Optimisation: Instruction Evaluation

5.3 Optimisation: Control Flow Analysis

5.4 Future Extensions

6 Conclusion