Cack Assembler

Source code, report, presentation: Chia-Wen Fan (B09303028)

Overview

In this short report, I would share how I implement the assembler for Hack machine language in C++, and some insights in the process of building this assembler.

Implementation

Modules

The Hack assembler implementation in C++ is modular, consisting of several key components that each handle a specific part of the assembly process. Below is an outline of the major classes:

- **Parser**: This class is responsible for reading the input assembly file (.asm) and breaking it into several tokens. It handles the parsing of A-instructions, C-instructions, and labels.

- **Symbol Table**: This class maintains a mapping between symbols (such as variable names or labels) and their corresponding memory addresses or line numbers. It provides functionality for adding new symbols and resolving symbols during translation.

- **Code Translator**: This class converts parsed instructions into their binary representation. It includes methods for translating comp, dest, and jump fields in C-instructions.

- **Assembler Driver**: This class orchestrates the entire assembly process by coordinating the Parser, Symbol Table, and Code Translator.

It ensures that the input file is processed in two passes: one for populating the symbol table and another for generating the final machine code.

Workflow

The workflow of the Hack assembler follows these steps:

1. **Initialization**: The assembler initializes necessary components, such as the Symbol Table and input/output file streams.

2. **First Pass**: The assembler scans through the assembly file to identify all label declarations and populate the Symbol Table with their respective memory addresses.

3. **Second Pass**: The assembler processes each instruction line by line:

    - For A-instructions, it resolves symbols to addresses (if applicable) and generates the corresponding binary code.

    - For C-instructions, it translates the comp, dest, and jump fields into binary code using the Code Translator module.

4. **Output Generation**: The translated binary instructions are written to the output file (.hack), completing the assembly process.

Insights

- **Data Structures Used**:

  - **Symbol Table**: Implemented using an `std::unordered_map`, which provides average constant-time complexity for insertions and lookups.

  - **Code Translator Table**: For translating fields in C-instructions (e.g., comp, dest, jump), a lookup table is implemented also using an `std::unordered_map`.

- **Troubleshooting**: During development, an issue where symbols were incorrectly added to the Symbol Table multiple times was fixed. This avoids having identical key values being stored the symbol table and ensures that each symbol is mapped correctly to a single memory address.

- **GitHub Repository**: The full implementation, along with detailed documentation and test cases, is available on GitHub: `https://github.com/valeriefan/CackAssembler`.

References

- Nisan, N., & Schocken, S. (2021). *The Elements of Computing Systems: Building a Modern Computer from First Principles*. MIT Press.

- Hack Assembly Language Specification. Available at: `https://www.nand2tetris.org/_files/ugd/44046b_89a8e226476741a3b7c5204575b8a0b2.pdf`