Assembler Design Notes

CSCI 6461 Team 10 Project

February 7, 2025

1 Overview

The assembler is designed to process assembly language input and generate machine-readable octal instructions. It follows a two-pass assembly approach to ensure proper label resolution and binary translation.

2 Architecture

The assembler consists of the following key components:

- **Opcode Table:** Maps assembly mnemonics to binary opcode values.
- **Symbol Table:** Stores labels and their corresponding memory addresses.
- **Two-Pass Processing:**
  - **Pass One:** Builds the symbol table by scanning labels and addresses.
  - **Pass Two:** Converts instructions into binary, resolves labels, and generates octal output.

3 Processing Logic

3.1 Pass One: Symbol Table Construction

- Reads each line from the assembly source file.
- Identifies labels and stores their memory locations.
- Increments the location counter for each instruction.

3.2 Pass Two: Instruction Translation and Output

- Reads the assembly file again.
- Converts mnemonics to binary using the opcode table.
- Resolves label references using the symbol table.
- Formats the binary instruction and converts it to octal.
- Writes the final machine-readable code to ‘listing.txt‘ and ‘load.txt‘.

4 Key Functions and Responsibilities

- **parseOperands():** Converts instruction operands into binary.
- **binaryToOctal():** Converts 16-bit binary strings to octal representation.
- **passOne():** Constructs the symbol table.
- **passTwo():** Processes instructions and generates output files.

5 Error Handling and Edge Cases

- Undefined labels trigger an error in ‘passTwo()‘.
- Extra or missing operands are flagged.
- Instructions are padded to ensure exact 16-bit representation.

6 Conclusion

This document outlines the assembler’s design, core logic, and processing workflow. The two-pass structure ensures proper translation and label resolution, making the assembler reliable for ISA-based execution.