Two Pass Assembler - Project Report/ User Manual

- Aaron Tom Viji
- Roll no. 1

This document provides an overview and detailed analysis of the Two Pass Assembler application.

1. Introduction

The Two Pass Assembler application is an Android tool designed to convert assembly language code into machine-readable object code using a two-pass methodology. This application simplifies the assembly process for users, allowing them to transform their code efficiently.

This document outlines the features, design, and functionality of the app, along with instructions on how to use it effectively.

2. Overview of Two Pass Assembler

This assembler processes assembly code in two distinct phases:

- **Pass 1:** Constructs a symbol table by reading the source code and tracking all defined labels.
- **Pass 2:** Utilizes the symbol table and instruction set to generate the final object code for execution.

The app supports basic assembly instructions, error detection, and has a user-friendly interface for easy code input and processing.

3. Design and Implementation

The application follows a modular architecture:

- **Pass 1:** Responsible for intermediate file and symbol table generation.
- **Pass 2:** Responsible for generating object code and output using the optab, intermediate file and symbol table.

- **Error Handling Module:** Identifies syntax errors, undefined symbols, and provides appropriate error messages.
- **File Input/Output Module:** Reads and writes files for the intermediate, symbol table, and object code outputs.

The app was developed using **Kotlin, with Jetpack Compose** for the user interface and native Android file management for loading and saving files.

4. Features

- Symbol Table Generation
- Object Code Generation
- Output Code Generation
- Error Detection and Reporting
- User-friendly Interface
- All files are saved on device
- Directly upload files from device or cloud

5. User Interface

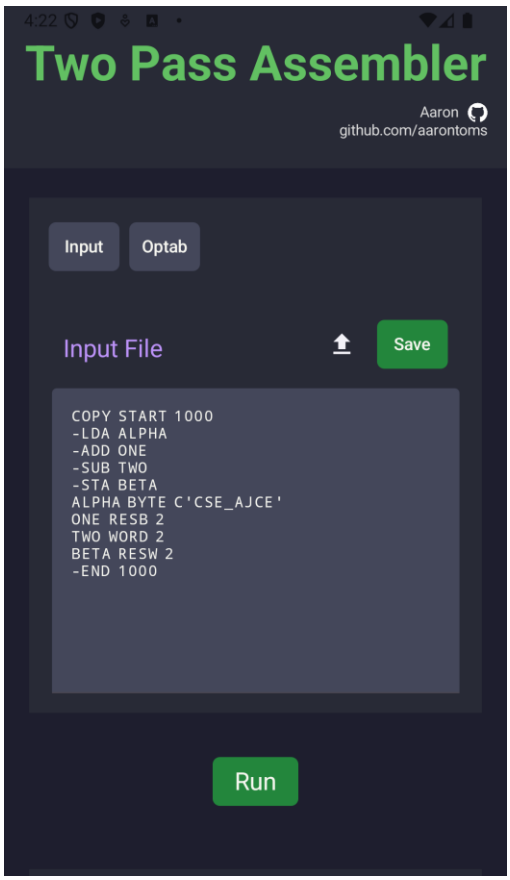
The app features a clean, user-friendly interface with the following components:

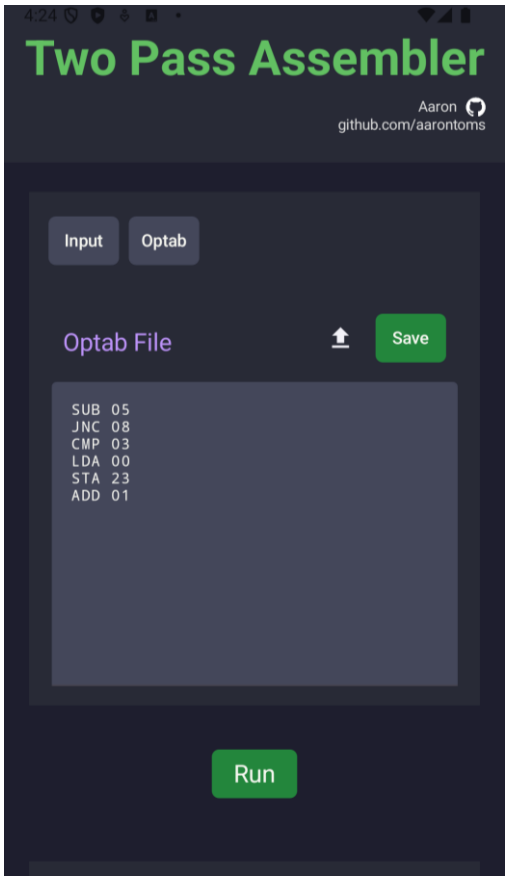
- **Text Field for Code Input:** Users can manually enter or load assembly code from a file.
- **Buttons for Assembling:** Users can click buttons to start the assembly process, clear files, or load source code from a file.
- **Display Areas for Output:** The results of the assembly process, including symbol tables, intermediate files, and final object code, are displayed within the app.

The user interface is divided into clear sections for easy navigation:

- **Header Section:** Displays the app's title, “Two Pass Assembler,” and a clickable GitHub link with an icon for quick access to the project repository.
- **Input Section:** A fixed-height text input field where users can either manually enter their assembly code or load a file from device or cloud. This area supports

large programs, with a scrollable interface and syntax highlighting to differentiate comments and instructions.





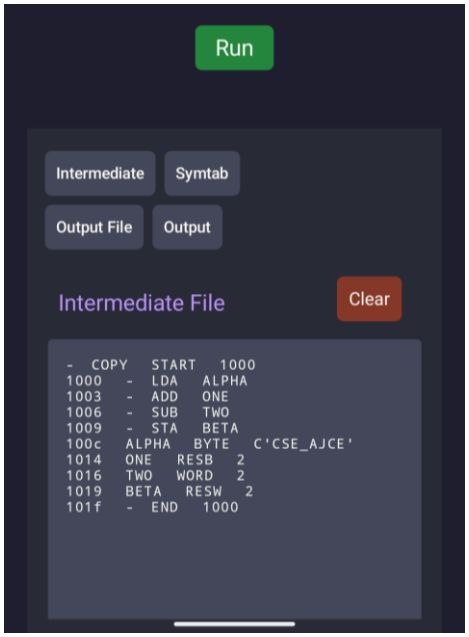
- **Button Section:**
    - **Load File Button:** Click this button to select and upload a .txt file containing your assembly code from your device. The file content will automatically load into the input field.
    - **Assemble Button:** Starts the two-pass process to generate the symbol table and object code.
    - **Clear Button:** Resets the text field and clears any previously generated output files for a fresh start.
- **Output Section**

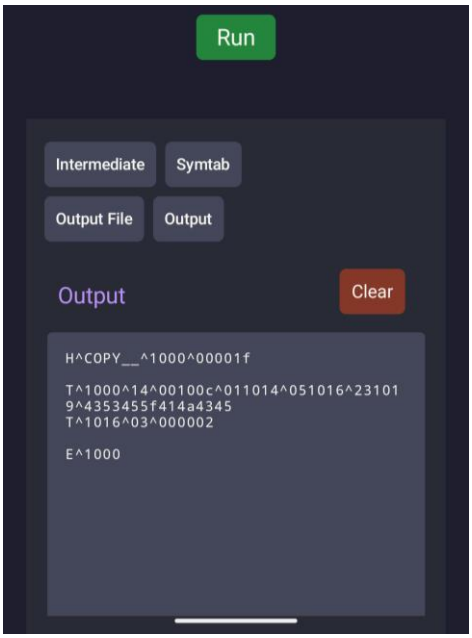
Displays the results of the assembly processes:

- **Symbol Table:** Lists symbols and their addresses from Pass 1.

- **Intermediate Code:** Shows the processed source code with resolved labels.
- **Object Code:** Outputs the final machine code generated in Pass 2. The object code is also saved to an external .txt file for easy access and use.
- **Output Code:** Outputs the final machine code generated in Pass 2.

All files, including the symbol table, intermediate code, and object code, are automatically saved to the user's device for easy access and retrieval.





6. Requirements

User Requirements:

These requirements define what users expect and need from the app in terms of functionality and usability.

- **Platform:** The app should run on Android devices, ideally supporting versions from Android 8.0 (Oreo) and above.
- **Input:** Users must be able to upload assembly code files (in .txt format) or manually input code directly into a text field within the app.

Developer Requirements:

These requirements define what developers need to consider while developing the app.

- **Programming Language:** Kotlin should be used as the primary language for development in Android Studio.
- **Development Environment:**
    - Android Studio (latest stable version recommended).
    - Gradle for dependency management.
- **Minimum SDK:** The app should target at least API level 26 (Android 8.0 Oreo) or higher to ensure compatibility across a wide range of devices.
- **Libraries/Frameworks:**
    - **Jetpack Compose:** For building a modern, declarative UI.
    - **Activity Result API:** To handle file uploads via intent-based actions.
    - **File I/O:** Use standard Kotlin or Android libraries for reading and writing files (e.g., fileReader and fileWriter functions).
    - **Dialog Components:** Jetpack Compose dialogs (e.g., Dialog, Surface, Button) for file selection and confirmation dialogs.

7. App

The Two Pass Assembler successfully implements the two-pass assembly process for converting assembly language into machine code. The app is effective in symbol management, error detection, and code generation.

To download the app, visit GitHub Repository for access to the source code and installation instructions.