Project Report on Python GUI-based SIC Assembler

By Advaitha Sujith Roll No.12

1. Introduction

This project report details the implementation of a Python-based SIC (Simplified Instructional Computer) assembler utilizing a graphical user interface (GUI) created with the Tkinter library. The assembler follows a two-pass methodology, enabling users to convert assembly language code into machine code efficiently. The application is designed to provide a user-friendly interface for code input, assembly processing, and output display, all within a single window.

2. Features of the Assembler

2.1 Two-Pass Assembly Process

• Pass 1:

o The assembler reads the source code and generates a symbol table that maps labels to their corresponding memory addresses. It also creates an intermediate representation of the source code with address placeholders for any unresolved labels.

• Pass 2:

o The assembler utilizes the symbol table and operation code (OP) table to resolve any placeholders, generating the final object code that can be executed by the SIC architecture.

2.2 Key Functionalities

• Code Input Area:

oUsers can input assembly language code directly into a scrollable text area.

• OP Table Input Area:

o Users can define the operation codes in a separate scrollable text area for easy access and modification.

• Output Areas:

o The application displays the intermediate file, symbol table, output file, and object code in dedicated text areas, allowing users to view results immediately after assembly.

• Generate Button:

o A button to initiate the assembly process, which executes both Pass 1 and Pass 2,displaying results in real time.

3. Tech Stack

• Python: The programming language used for developing the assembler.

• Tkinter: A standard GUI toolkit for Python, providing the graphical interface.

• ScrolledText: A widget from Tkinter for creating scrollable text areas.

4.Requirements

User Requirements:

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

- **Platform Compatibility**:
  The app should run on major desktop operating systems, including Windows 10 and above.
  - For Windows, it should support 64-bit systems.

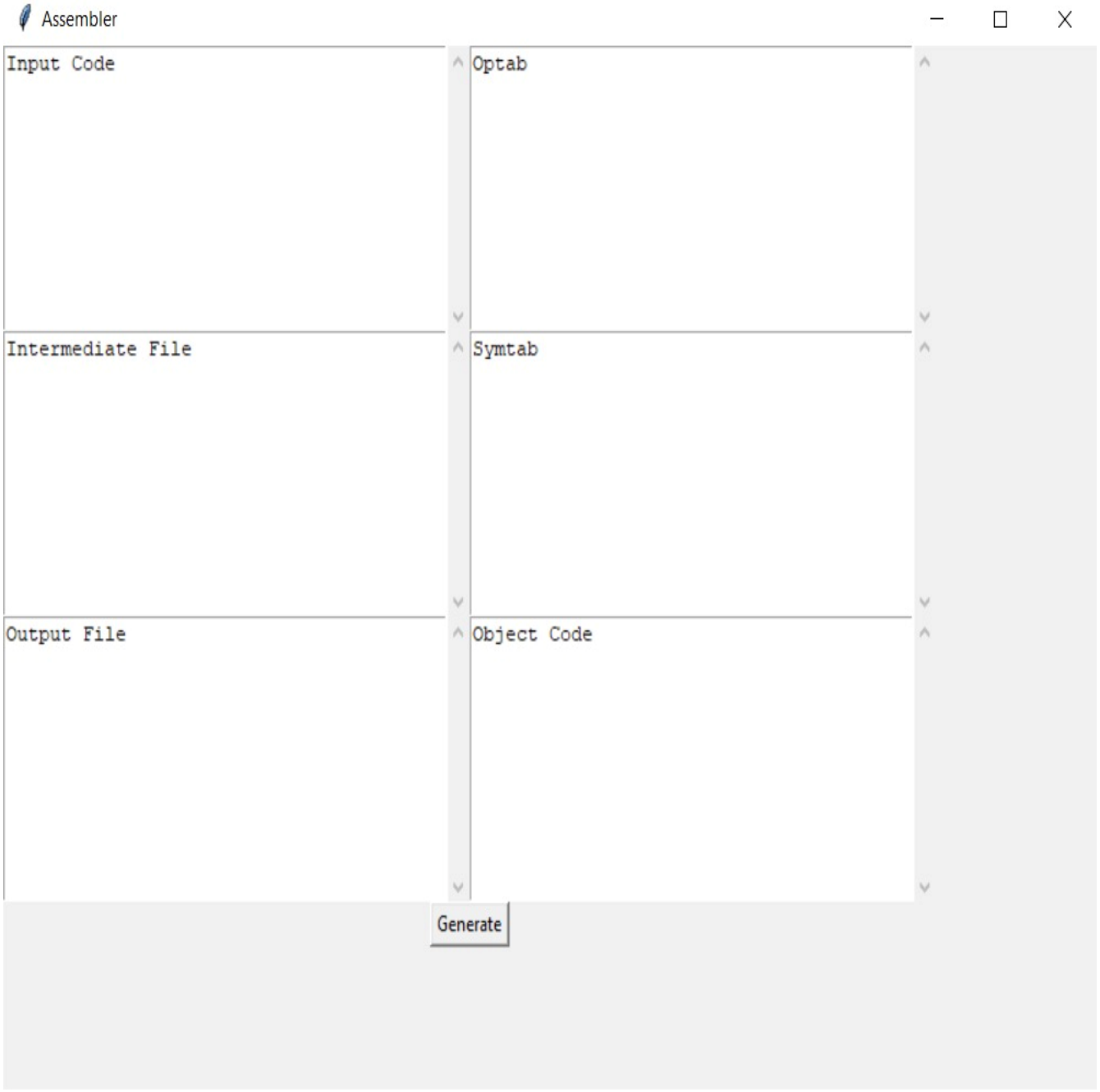
- **Input Options**:
  Users must be able to manually input SIC assembly code directly into the app .

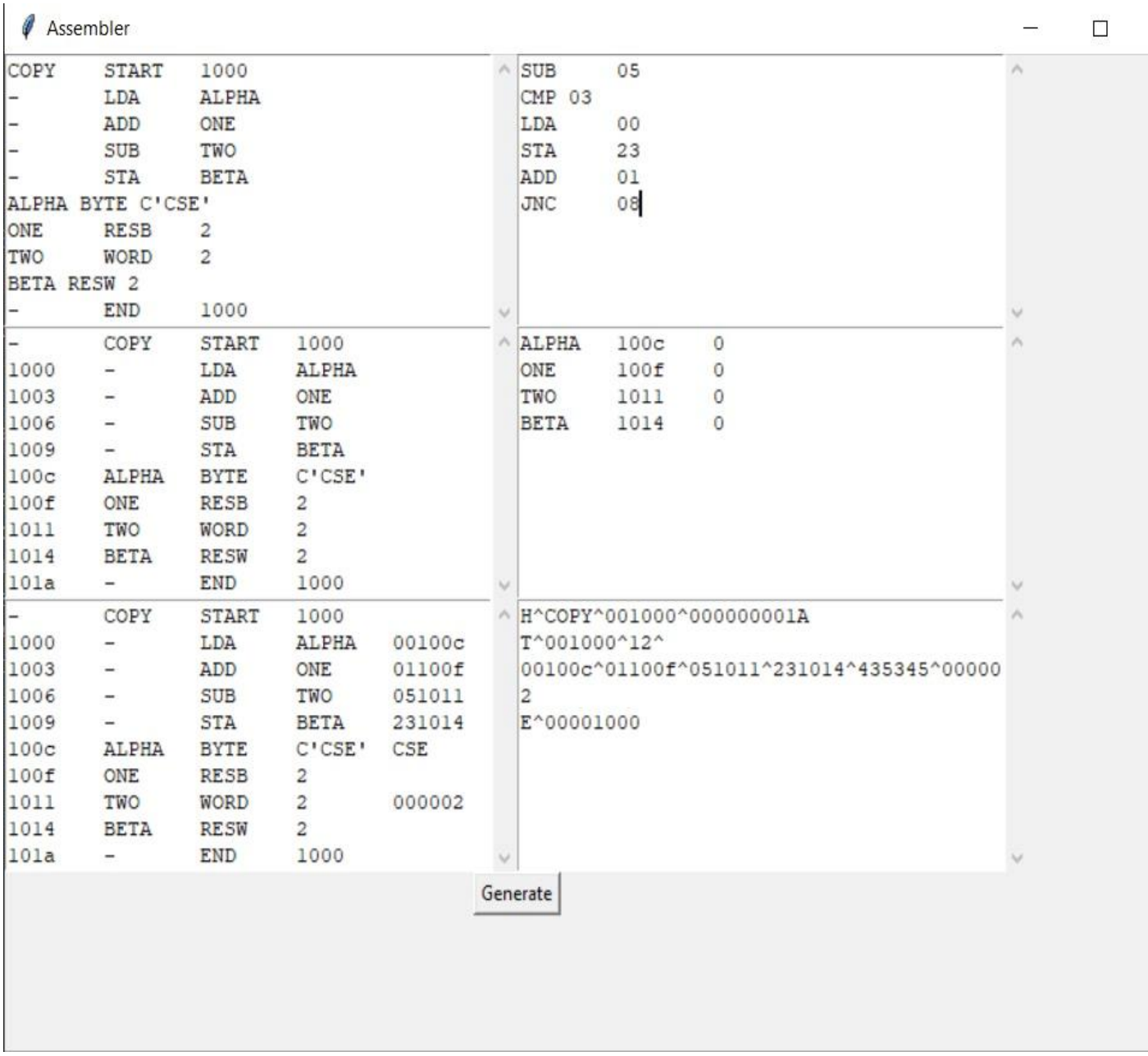
Developer Requirements:

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

- **Programming Language**:
  Python should be used as the primary programming language, and the GUI should be built using the Tkinter library.
- **Development Environment**:
  - **IDE**: Any Python-compatible IDE, such as PyCharm or VS Code, can be used for development.
  - **Python Version**: Use Python 3.7 or higher.
  - **Dependencies**: Ensure Tkinter is installed (usually included in most Python distributions). Optionally, use pyinstaller or cx_Freeze for packaging the app as an executable.
- **Code Modularity**:
  The assembler logic should be separated from the GUI. Use an Assembler class to handle the assembler passes and logic, ensuring it is decoupled from the user interface code.

5. Screenshots





6. How to Use the Assembler

Step 1: Input Assembly Code and OP Table

• Input Code:

o Enter the assembly code in the designated input area. The code should follow the

SIC format.

• OP Table:

o Define the operation codes in the OP table input area, mapping each opcode to its

corresponding machine code.

Example Input Code:

COPY START 1000

- LDA ALPHA

- ADD ONE

- SUB TWO

- STA BETA

ALPHA BYTE C'CSE'

ONE RESB 2

TWO WORD 2

BETA RESW 2

- END 1000

Example OP Table:

SUB 05

CMP 03

LDA 00

STA 23

ADD 01

JNC 08

Step 2: Click 'Generate'

After entering the input code and OP table, click the "Generate" button to start the assembly

process. The application will:

• Pass 1: Generate the symbol table and intermediate file with placeholders for unresolved

labels.

• Pass 2: Resolve these placeholders and generate the final object code.

Step 3: View the Output

The output will be displayed in the following sections:

• Intermediate File:

o Shows the processed assembly code along with memory addresses and any

placeholders.

• Symbol Table:

o Lists all labels and their corresponding memory addresses.

• Output File:

o Displays the complete assembly output including labels, opcodes, and object codes.

• Object Code:

o Displays the final machine code that can be executed by the SIC machine.

Step 4: Reset (Optional)

If you want to start over or enter new code, simply refresh the input areas manually or close and

reopen the application.

7. Conclusion

The Python GUI-based SIC assembler effectively implements a two-pass assembly process, allowing users to input assembly code and OP tables seamlessly. The application provides immediate feedback through its output areas, enhancing the user experience by simplifying the process of converting assembly language into machine code. This project serves as an educational tool for those learning assembly programming and provides a practical application of GUI development in Python.