User Manual for Two-Pass Assembler GUI Program

Introduction

This Two-Pass Assembler GUI program is designed to process assembly language source code through two passes to generate machine-level code, symbol tables, and other outputs. The program helps users understand how assemblers work by providing real-time feedback and visual outputs in a simple graphical interface.

Overview

Assemblers convert assembly language code into machine code that the computer can understand. In this two-pass assembler:

- **Pass 1** scans the assembly code to determine addresses for all labels, producing a symbol table and an intermediate file.
- **Pass 2** uses the symbol table and intermediate file to generate the final object code.

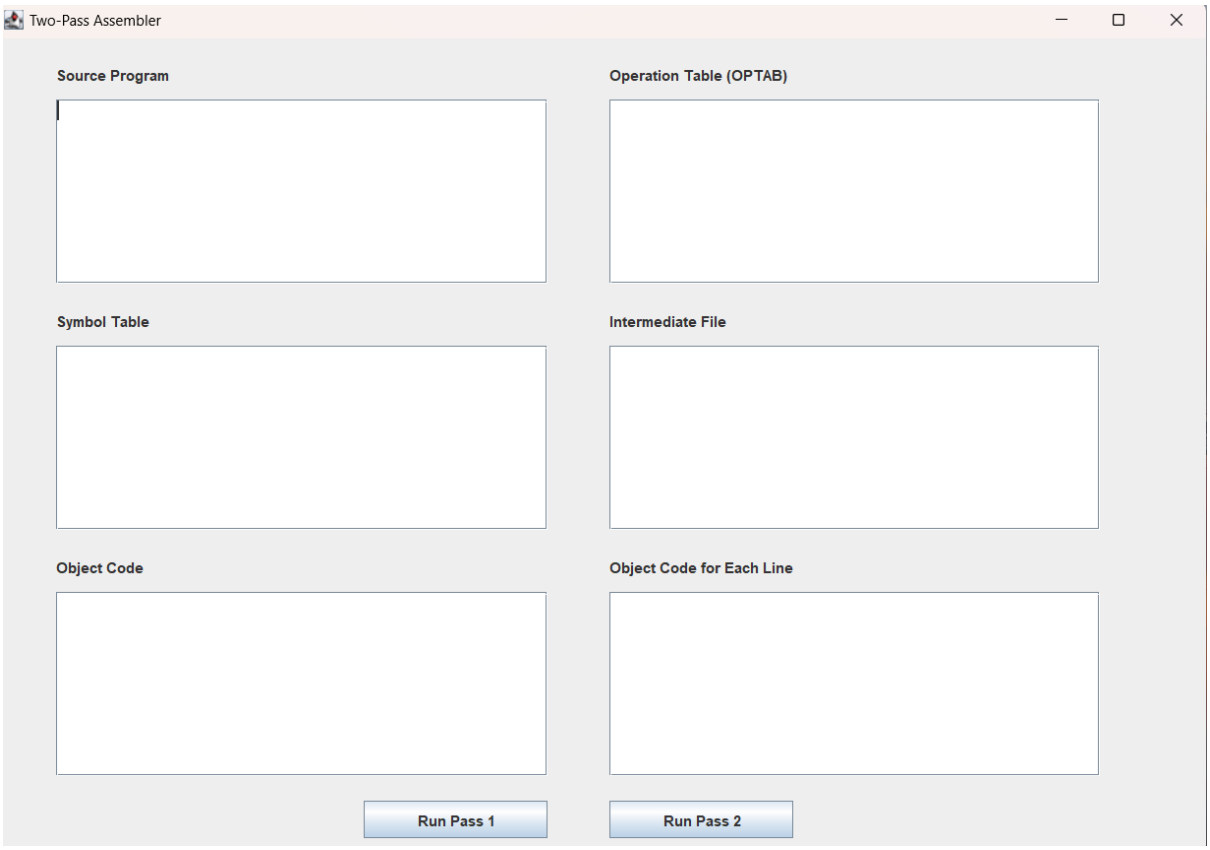
This program allows users to input their assembly code in a text area, and with the click of buttons, perform pass1 and pass2 of the assembler to generate the required output.

Program Components:

This program consists of several Java Swing components:

- **Source Program Area:** A text area for users to input or paste their assembly language source code.
- **OPTAB:** A table that lists machine instructions and their corresponding opcode (entered by the user).
- **SYMTAB:** Displays the labels encountered during Pass 1 and their associated addresses.

- **Intermediate File Area:** Shows the intermediate representation of the source program after Pass 1.
- **Object Code Area:** Displays the final object code in record format after Pass 2.
- **Object Code per Line Area:** Provides object code for each individual line of the source program.



Pass 1:

- **Input:** The user enters the source assembly program and the OPTAB (operation table).
- **Tasks Performed in Pass 1:**
    1. **Read the Source Program:** Line by line, each instruction is read and processed.
    2. **Symbol Table Generation:** If a label is found, it is added to the symbol table with its memory address.

3. **LOCCTR (Location Counter) Calculation:** The program calculates memory addresses for each instruction using the location counter.
4. **Intermediate File Generation:** This file records each line of the source program along with its address.
5. **Program Length Calculation:** After processing the entire program, the length is calculated based on the last address.

Output: Symbol Table and Intermediate File are generated and displayed in the respective text areas

Pass 2:

- **Input:** Uses the symbol table and intermediate file generated in Pass 1.
- **Tasks Performed in Pass 2:**
    1. **Object Code Generation:** For each instruction, the program generates the corresponding object code using the OPTAB and symbol table.
    2. **Text Records:** Object code is grouped into records with appropriate start addresses and lengths.
    3. **Object Code Per Line:** Shows the object code for each line of the source program.
- **Output:** Object Code and Object Code per Line are displayed in the respective text areas.

Outputs:

- **Symbol Table:** Contains labels and their associated memory addresses.
- **Intermediate File:** Lists the source program instructions along with their addresses.

- **Object Code:** The final machine code, formatted in header, text, and end records.

  Object Code per Line: Displays the object code for each instruction line from the source program

Requirements

System Requirements:

- Operating System: Windows, macOS, or Linux
- Java Development Kit (JDK) 8 or later installed
- Minimum of 2GB RAM (for smooth execution)
- Screen resolution of at least 1024x768

Technical Requirements:

- **Java Swing** is used for the GUI.
- Users should provide assembly code written in the SIC format.
- The operation table (OPTAB) should be provided with correct operation-opcode pairs.

How to Use:

1. **Launch the Application:**
    - Open the program by running the AssemblerGUI class, which will start the graphical user interface.
2. **Input the Source Program:**
    - In the “Source Program” area, enter the assembly language code.

Enter the OPTAB

- In the “Operation Table (OPTAB)” area, input the operation codes and their corresponding machine instructions.
- Click on ‘Run Pass 1’ to generate the symtab and Intermediate file.
- Click on ‘Run Pass 2’ to generate the object code.

Sample Input:

```
SAMPLE  START   1000
FIRST     LDA     NUM1
          ADD     NUM2
          STA     RESULT
          RSUB
NUM1    WORD    5
NUM2    WORD    10
RESULT  RESW    1
          END     FIRST
```

Sample Optab:

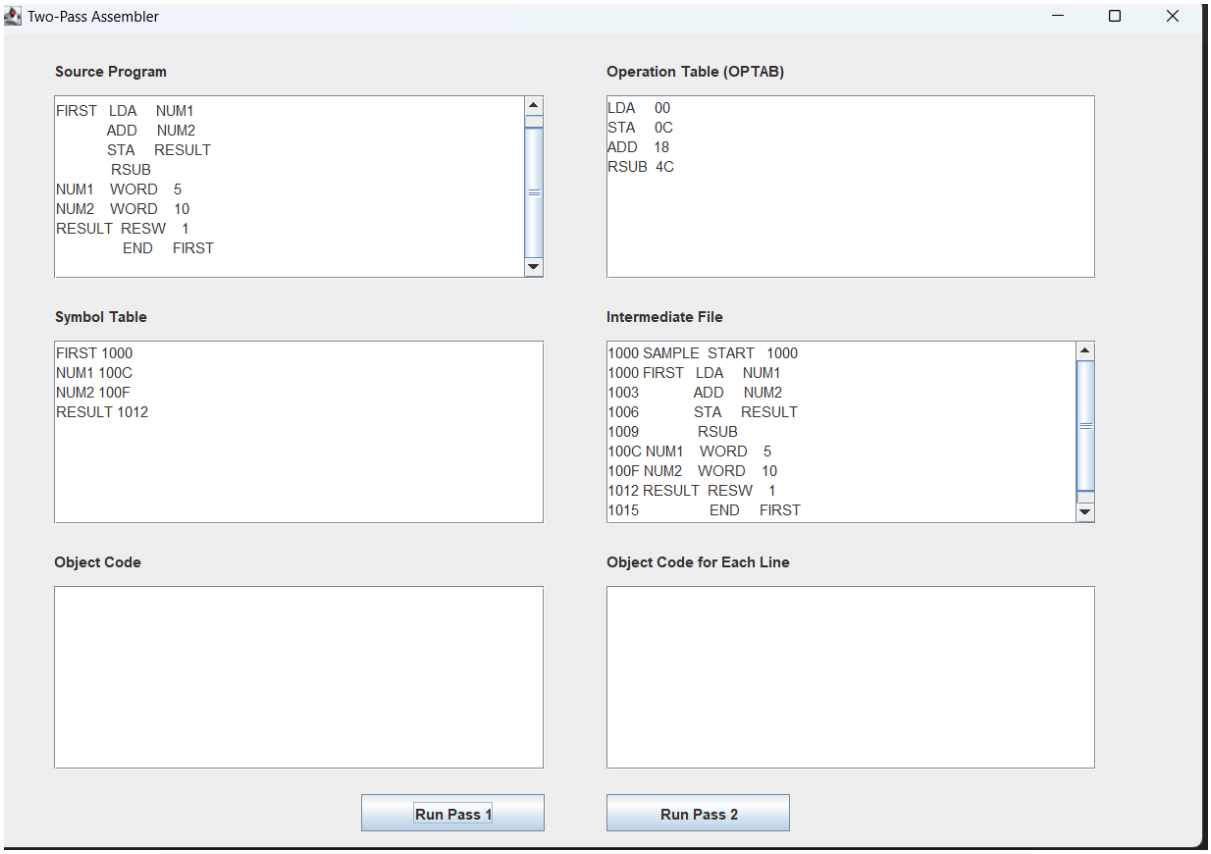
LDA 00

ADD 18

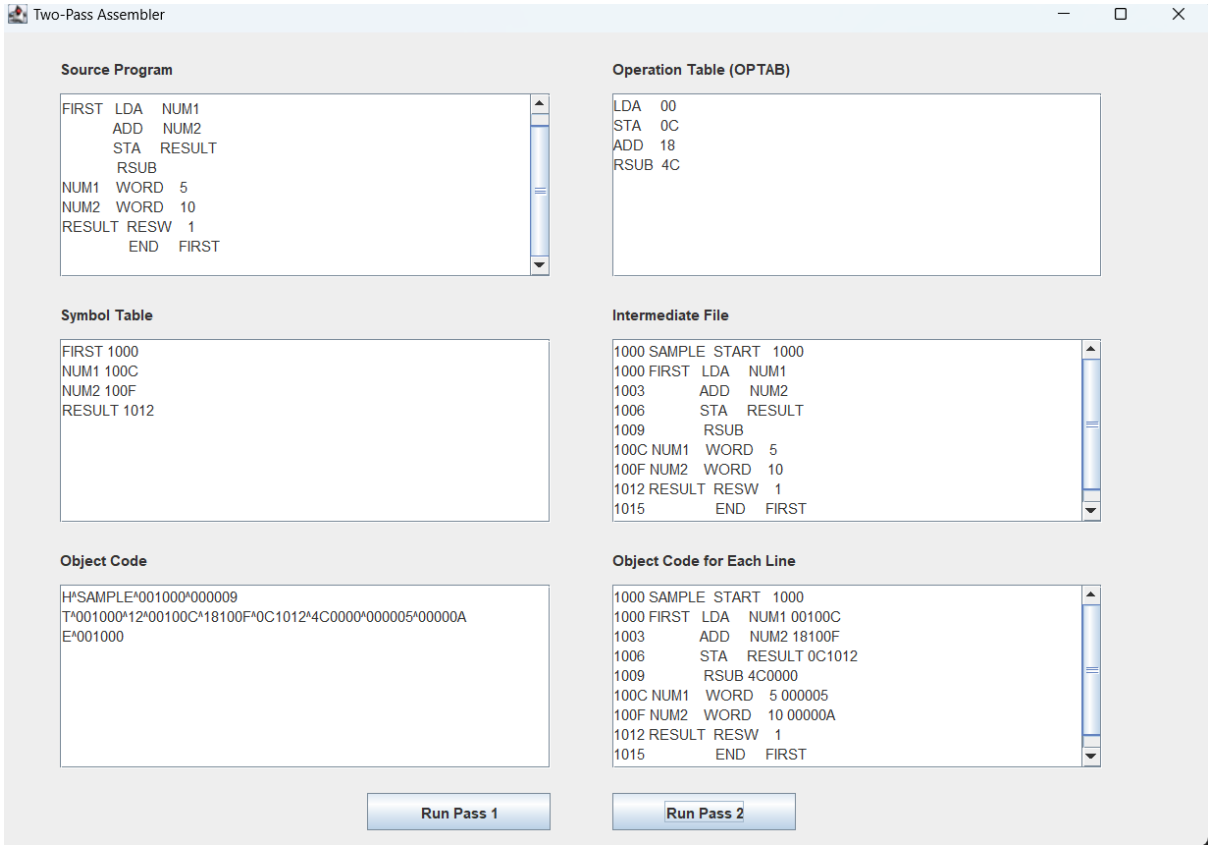
STA 0C

RSUB 4C

PASS1 OUTPUT:



PASS2 OUTPUT:



Conclusion

The Two-Pass SIC Assembler GUI simplifies the process of assembling SIC programs. Users can easily input assembly code, visualize the symbol table and intermediate file, and generate object code with minimal effort. Its clean interface and functionality make it a valuable tool for students and professionals learning about assembler design or working with the SIC architecture.

Ardra Sajeevan

Roll no:51