NAME:ANAKHA SD
CLASS: CSE A
ROLL NO:35

Project Report: Two-Pass Assembler

Introduction

The Two-Pass Assembler project aims to develop an assembly language translator that processes an input program written in assembly language and converts it into machine code. The program performs this translation in two passes:

- The first pass constructs a symbol table by scanning the source code and calculating the addresses of labels and symbols.
- The second pass generates the actual object code using the symbol table created in the first pass.

This assembler is designed to handle basic assembly instructions, generate intermediate code, and produce machine-readable object code. It offers an intuitive graphical user interface (GUI) for users to input assembly code and view the intermediate code and final object code.

Objectives The primary goals of the Two-Pass Assembler project are:

- To develop a tool that demonstrates the functioning of a two-pass assembler.
- To provide a clear visualisation of the intermediate code and the final object code.
- To allow reassembly of input to ensure the correct conversion process.

Features

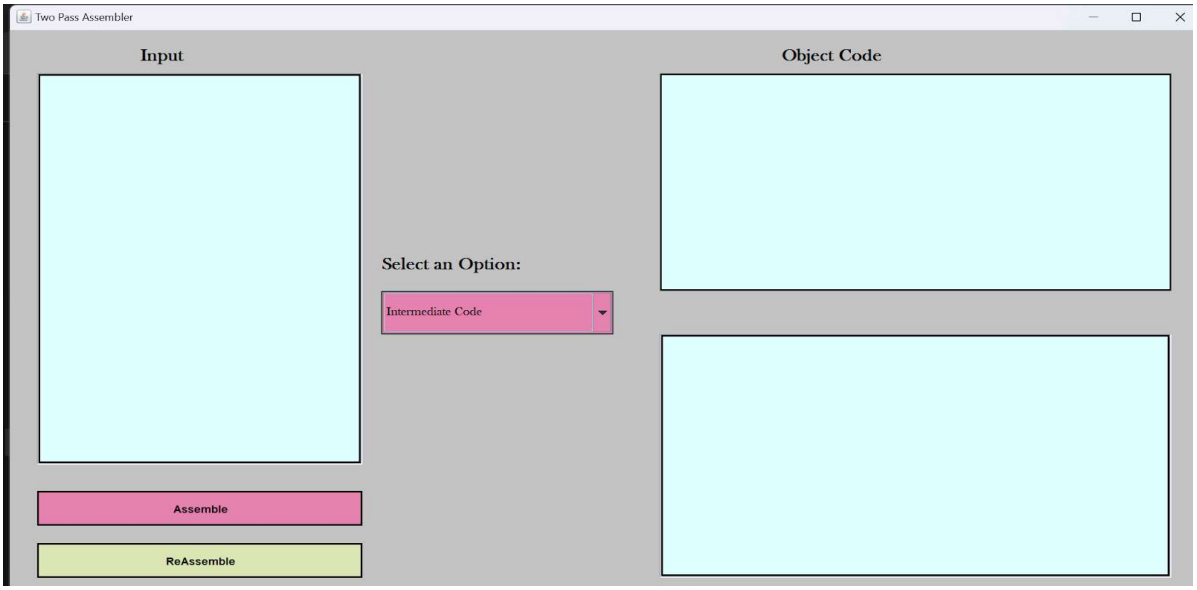
- **Input Window:** Users can input assembly language code into the program.
- **Intermediate Code Generation:** The system generates intermediate code after processing the input and displays it.
- **Object Code Generation:** Once the intermediate code is generated, the system produces machine-level object code.
- **Reassemble Option**: A feature allowing users to reassemble the input code, ensuring flexibility and correctness.
- **Graphical User Interface:** A user-friendly GUI where users can interact with the assembler easily.

System Requirements:

- Java Runtime Environment (JRE) or Java Development Kit (JDK) installed.
- The system should have adequate memory to handle the graphical interface and backend processing.

Design and Implementation:

- **First Pass**: The assembler reads the source program line by line, creating the symbol table. The symbol table keeps track of the addresses assigned to labels and variables.
- **Second Pass**: Using the symbol table from the first pass, the assembler generates the final object code.



The GUI for the assembler is divided into several key areas:

- **Input Section:** A text box where users can input the assembly language code.
- **Option Selector:** A dropdown that allows users to select between intermediate code and object code for display.
- **Object Code Section:** Two boxes where the intermediate and object codes are displayed after the respective pass.
- **Buttons:** Two main buttons - "Assemble" to start the translation process, and "Reassemble" to reset and restart the process.

Execution Flow:

1. The user inputs the assembly code in the input box.
2. The user clicks the "Assemble" button to begin the two-pass assembly process.
3. The first pass generates the intermediate code and constructs the symbol table.
4. The second pass generates the final object code using the symbol table.
5. Both the intermediate code and object code are displayed in their respective sections.

Sample Input Code:

```
COPY START 1000

** LDA ALPHA

** ADD ONE

** SUB TWO

** STA BETA

ALPHA BYTE C'CZXE'

ONE RESW 2

TWO WORD 5

BETA RESW 1

** END -
```

Conclusion :

The Two-Pass Assembler project successfully demonstrates the functionality of an assembler using a two-pass approach. The GUI-based design makes it user-friendly and accessible for students and educators interested in understanding how assemblers work. This project serves as both a learning tool and a practical application in the field of low-level programming and system design.