TWO PASS ASSEMBLER
GUI FORMAT

ALISHA ANN SUBASH

CSE A

ROLL NO : 27

AssemblerGUI Documentation

Overview

The `AssemblerGUI` is a Java Swing application designed to simulate a two-pass assembler. It allows users to input assembly code and opcode definitions, process the assembly code through two passes, and generate a symbol table and object program. The assembler supports basic directives and instructions, including `START`, `END`, `BYTE`, `WORD`, `RESB`, and `RESW`.

Features

- **User-Friendly GUI**: Easy-to-use graphical interface for inputting assembly code and opcode definitions.
- **Two-Pass Assembly**: Processes assembly code in two passes:
  - **Pass 1**: Generates a symbol table and intermediate code.
  - **Pass 2**: Generates the final object program.
- **Hexadecimal Support**: Handles addresses and values in hexadecimal format.
- **Dynamic Opcode Table**: Users can input opcode definitions that map mnemonics to machine codes.

Components

1. **Input Area**: A text area where users can input assembly code.
2. **Opcode Area**: A text area for users to define opcodes and their corresponding machine codes (format: `opcode machine_code`).
3. **Output Area**: Displays the results of the assembly process, including the symbol table and object program.
4. **Buttons**:
   - **Run Pass 1**: Processes the assembly code for the first pass.
   - **Run Pass 2**: Processes the intermediate code to generate the object program.

Usage Instructions

Step 1: Define Opcodes

1. In the **Opcode Area**, input the opcode definitions in the format:

   ```
   Copy code
   opcode machine_code
   ```

   For example:

   ```
   LDA 00
   STA 23
   ADD 01
   ```

Step 2: Input Assembly Code

1. In the **Input Area**, enter the assembly code. Each line can include an optional label, an opcode, and an operand, formatted as follows,

   ```
   [label] opcode [operand]
   ```

   Example:

   ```
   START 1000
   LDA 2000
   STA 2001
   END
   ```

Step 3: Run Pass 1

1. Click the **Run Pass 1** button.
2. The output will show the **Symbol Table** and **Intermediate Code** in the **Output Area**.

Step 4: Run Pass 2

1. Click the **Run Pass 2** button.
2. The output will display the **Object Program** in the **Output Area**.

Example

Opcode Definitions

```
LDA 00
STA 01
ADD 02
SUB 03
```

Assembly Code

```
START 1000
LABEL1 LDA 2000
LABEL2 STA 2001
END
```

Output After Pass 1

```
Symbol Table:
LABEL1: 1000
LABEL2: 1003

Intermediate Code:
1000 LABEL1 LDA 2000
1003 LABEL2 STA 2001
1006 END
```

Output After Pass 2

```
Object Program:
H LABEL1 1000
T 1000 06 00 0002 01
E 1000
```

User Perspective Requirements

Software Requirements

1. **Operating System:**
   - Windows 10 or later, macOS, or a Linux distribution (e.g., Ubuntu).
2. **Java Runtime Environment (JRE):**
   - Java 8 or later installed to run Java applications.
3. **Assembler Application:**
   - The two-pass assembler application must be available in an executable format (e.g., JAR file) that users can easily download and run.
4. **Text Editor:**
   - A text editor (e.g., Notepad, Sublime Text, or any IDE like IntelliJ IDEA or Eclipse) for users to write assembly code.
5. **Internet Connection (Optional):**
   - For downloading the assembler application and related documentation.

Hardware Requirements

1. **Processor:**
   - A modern multi-core processor (e.g., Intel i3 or AMD Ryzen 3 or equivalent).
2. **RAM:**
   - At least 4 GB of RAM (8 GB recommended for better performance).
3. **Storage:**
   - Minimum 100 MB of free disk space for the application and additional space for user-generated files.
4. **Display:**
   - Minimum screen resolution of 1366x768 pixels for optimal user interface experience.

Programmer Perspective Requirements

Software Requirements

1. **Development Environment:**
   - An Integrated Development Environment (IDE) like IntelliJ IDEA, Eclipse, or NetBeans configured for Java development.
2. **Java Development Kit (JDK):**
   - JDK 8 or later installed for compiling and running Java code.
3. **Version Control System:**
   - Git or another version control system to manage source code and collaboration.
4. **Testing Framework:**
   - JUnit or another testing framework for writing and running unit tests.

5. **Build Tools (Optional):**
    - Apache Maven or Gradle for managing project dependencies and building the application.

Hardware Requirements

1. **Processor:**
    - A modern multi-core processor (e.g., Intel i5 or AMD Ryzen 5 or equivalent) to facilitate smooth development and testing.
2. **RAM:**
    - At least 8 GB of RAM (16 GB recommended for running multiple applications and IDEs).
3. **Storage:**
    - Minimum 500 MB of free disk space for development tools, libraries, and the project files.
4. **Display:**
    - A minimum screen resolution of 1920x1080 pixels for effective multitasking and code visibility.

Notes

- Ensure that the input format is strictly followed to avoid errors.
- Labels must start with a letter and can contain letters, digits, and underscores.
- The assembler does not validate opcode definitions; users should ensure the machine codes correspond to the defined opcodes.

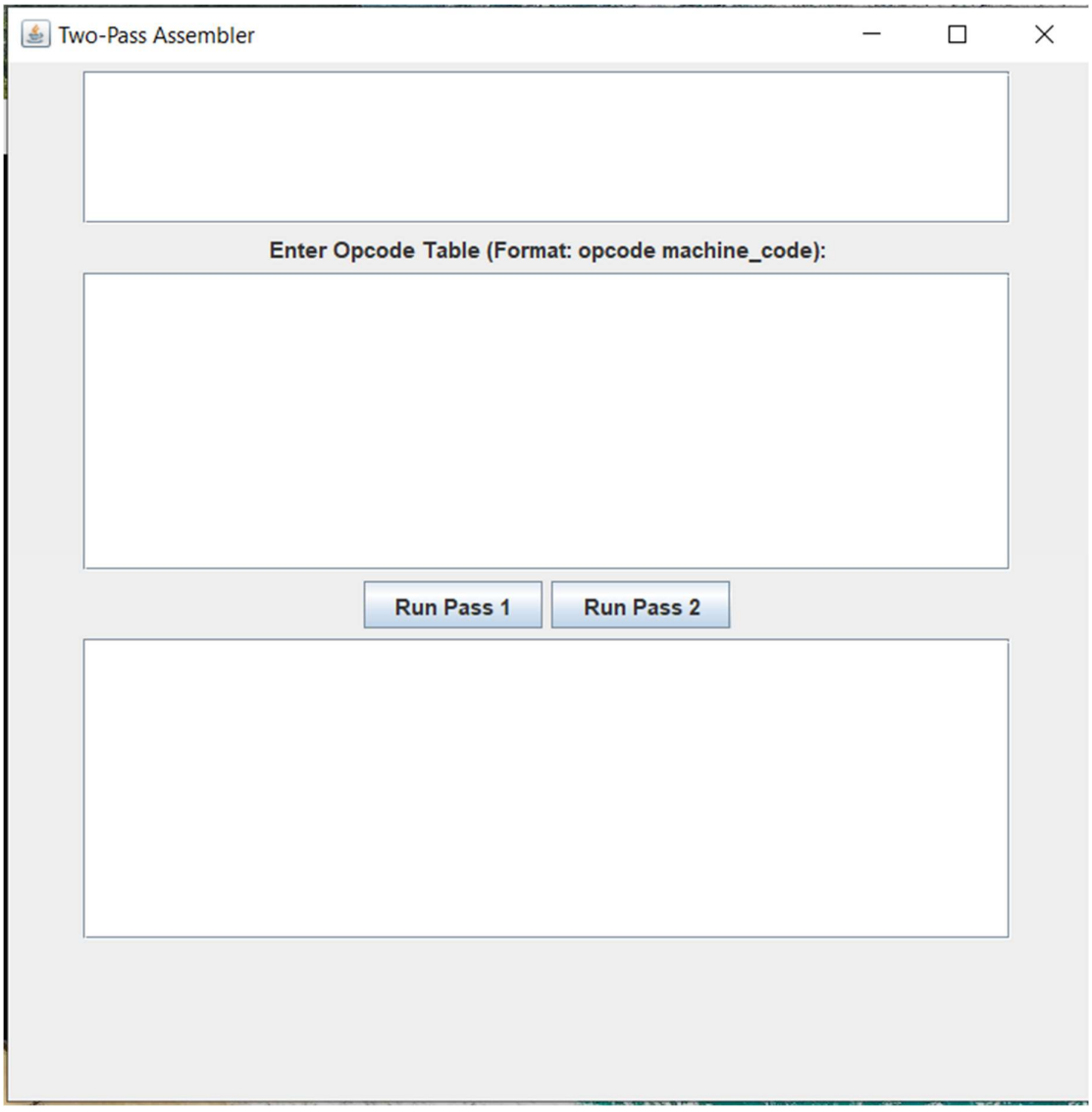
Limitations

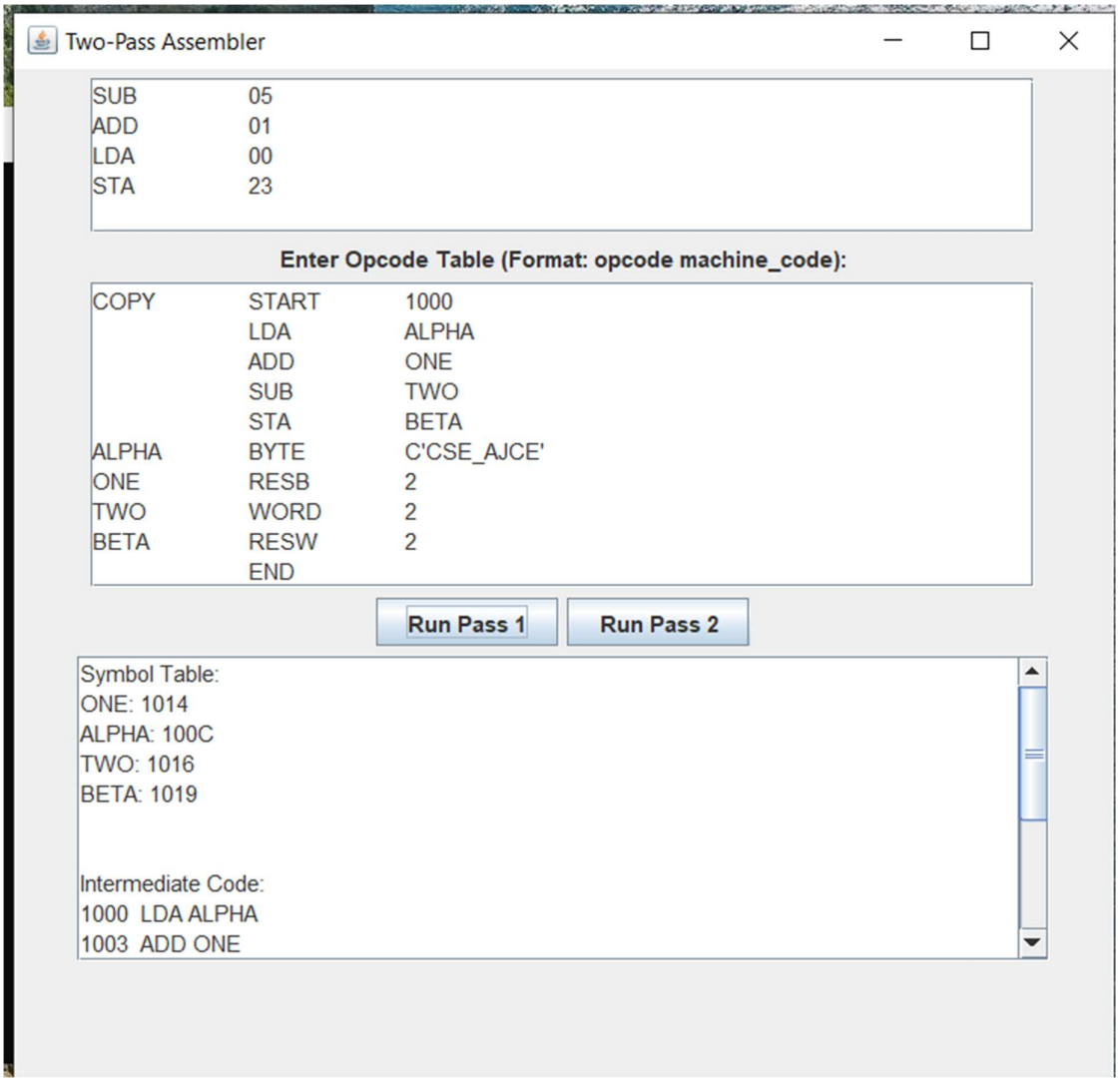
- This assembler is intended for educational purposes and may not support all assembly language features or opcodes.
- Error handling for invalid input formats is minimal.

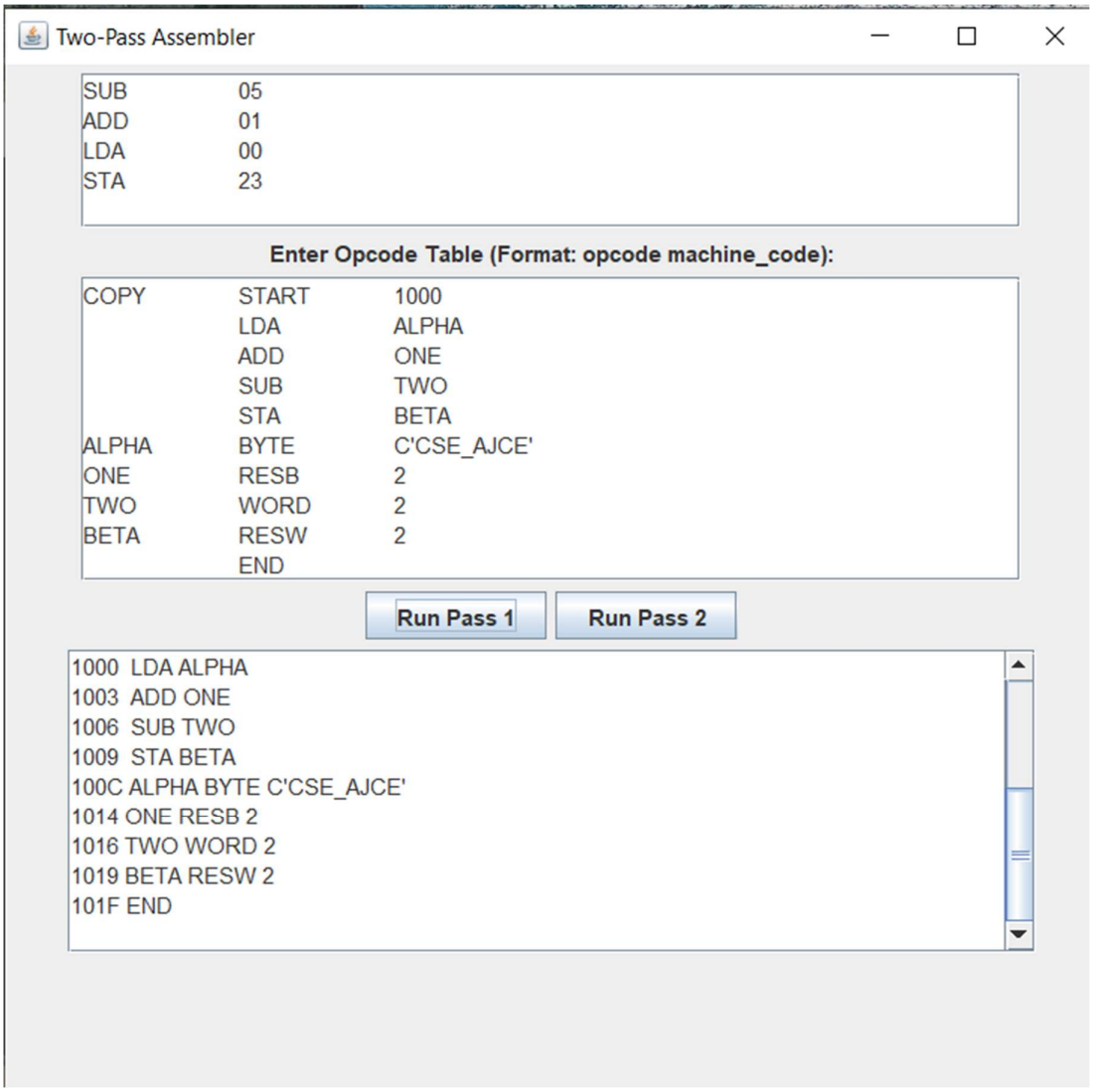
Conclusion

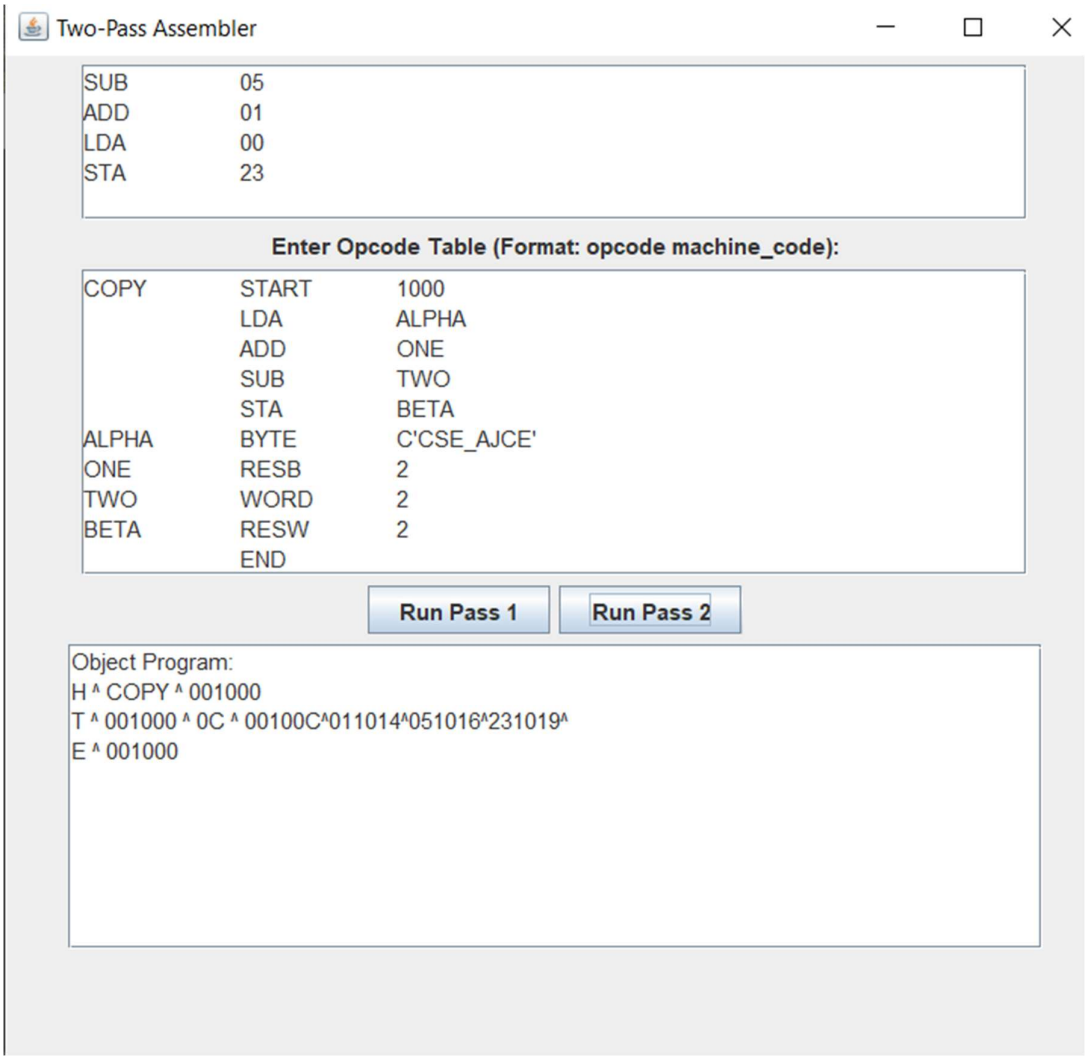
The `AssemblerGUI` serves as an effective tool for learning the fundamentals of assembly language processing and can be used in educational settings for teaching assembly language concepts. Users can expand its capabilities by adding more complex features or additional opcode definitions.

Screenshots




Two-Pass Assembler
SUB 05
ADD 01
LDA 00
STA 23
Enter Opcode Table (Format: opcode machine_code):
COPY START 1000
LDA ALPHA
ADD ONE
SUB TWO
STA BETA
ALPHA BYTE C'CSE_AJCE'
ONE RESB 2
TWO WORD 2
BETA RESW 2
END
Run Pass 1
Run Pass 2
Symbol Table:
ONE: 1014
ALPHA: 100C
TWO: 1016
BETA: 1019
Intermediate Code:
1000 LDA ALPHA
1003 ADD ONE


Two-Pass Assembler
SUB 05
ADD 01
LDA 00
STA 23
Enter Opcode Table (Format: opcode machine_code):
COPY START 1000
LDA ALPHA
ADD ONE
SUB TWO
STA BETA
ALPHA BYTE C'CSE_AJCE'
ONE RESB 2
TWO WORD 2
BETA RESW 2
END
Run Pass 1
Run Pass 2
1000 LDA ALPHA
1003 ADD ONE
1006 SUB TWO
1009 STA BETA
100C ALPHA BYTE C'CSE_AJCE'
1014 ONE RESB 2
1016 TWO WORD 2
1019 BETA RESW 2
101F END



https://github.com/AlishaSubash/2_pass