User Manual for PASS Assembler Program

Overview:

This program simulates a simple two-pass assembler for SIC (Simplified Instructional Computer). The assembler reads assembly language code from a text area, processes it in two passes (Pass 1 and Pass 2), and generates intermediate code, a symbol table, and final object code records.

The graphical user interface (GUI) allows users to enter the input assembly code, OPTAB (operation table), and displays the results of both passes.

Main Components:

1. **Input Assembly Code Area**:
    - Text area to enter the assembly code (instructions).
2. **OPTAB Entries Area**:
    - Text area to enter operation codes (opcodes) and their respective machine code equivalents.
3. **Intermediate/Final Output Area**:
    - Displays the intermediate file generated during Pass 1.
    - Displays the final object code generated during Pass 2.
4. **Symbol/Object Code Table Area**:
    - Displays the symbol table after Pass 1.
    - Displays the object code records after Pass 2.
5. **Buttons**:
    - **PASS1 Button**: Executes Pass 1, which creates an intermediate file and a symbol table.
    - **PASS2 Button**: Executes Pass 2, which generates final object code based on the intermediate file and symbol table.

Instructions for Using the Program:

1. **Input the Assembly Code**:
    - In the Input Assembly Code area, enter your assembly code
2. • Ensure that the code follows proper SIC assembly language syntax.

- **Input the OPTAB Entries**:

3. In the `OPTAB Entries` area, enter the operation codes and their corresponding machine code
4. **Running Pass 1**:
5.
    - Click on the **PASS1** button to execute Pass 1.
    - Pass 1 will perform the following:
        1. Parse the input assembly code.
        2. Generate the intermediate code (addressed lines of the code).
        3. Create a symbol table (SYMTAB) for all labels encountered.
    - After Pass 1 is complete:
        1. The `Intermediate/Final Output` area will show the intermediate file, which contains the addresses and corresponding assembly code.
        2. The `Symbol/Object Code Table` area will display the symbol table, which maps labels to addresses.
6. **Running Pass 2**:
7.
    - Click on the **PASS2** button to execute Pass 2.
    - Pass 2 will perform the following:
        1. Parse the intermediate file from Pass 1.
        2. Look up instructions in OPTAB and symbols in SYMTAB.
        3. Generate the final object code (machine code).
    - After Pass 2 is complete:
        1. The `Intermediate/Final Output` area will show the final assembly with object codes.
        2. The `Symbol/Object Code Table` area will display the object code records in the format:
            1. Header (H-record): Program name, starting address, and length.
            2. Text (T-record): Object codes for the program.
            3. End (E-record): Address of the first executable instruction.

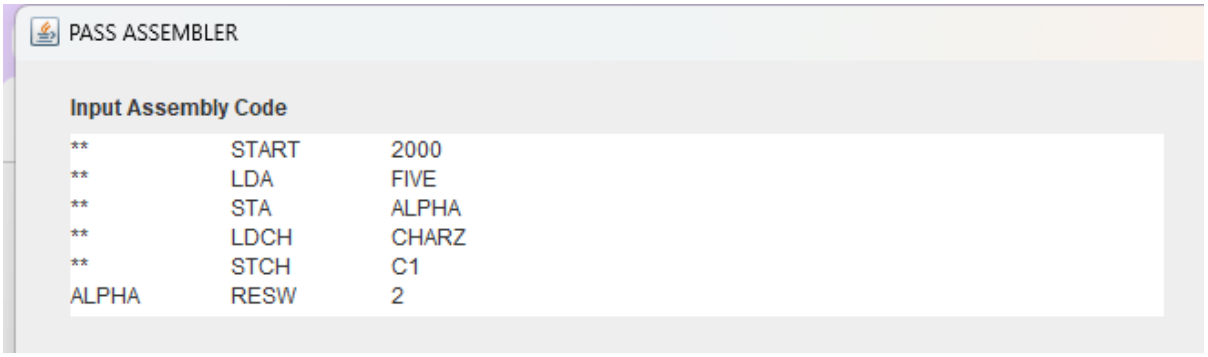
Features:

- **Input Assembly Code**: Enter assembly language code for processing.
- **OPTAB Entries**: Define operation codes and corresponding machine codes.
- **Intermediate Output**: Displays the results of Pass 1, including the generated intermediate file.
- **Symbol Table**: Shows the table of symbols and addresses after Pass 1.
- **Final Object Code**: Outputs the final object code generated in Pass 2.
- **Object Code Records**: Displays the header, text, and end records of the object code after Pass 2.
- **Pass 1 and Pass 2**: Buttons to execute each pass.

HOW TO USE THE PROGRAM

- **Step 1: Enter Assembly Code**

- **** START 2000**
- **** LDA FIVE**
- **** STA ALPHA**
- **** LDCH CHARZ**
- **** STCH C1**
- **ALPHA RESW 2**
- **FIVE WORD 5**



Step 2: Enter OPTAB Entries

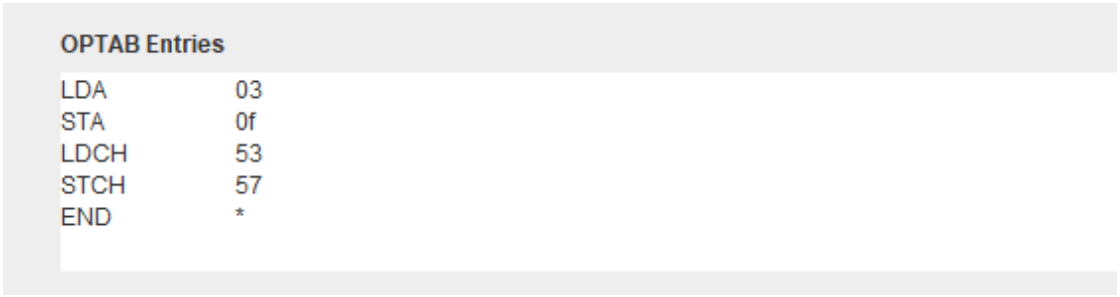
LDA 03

STA 0f

LDCH 53

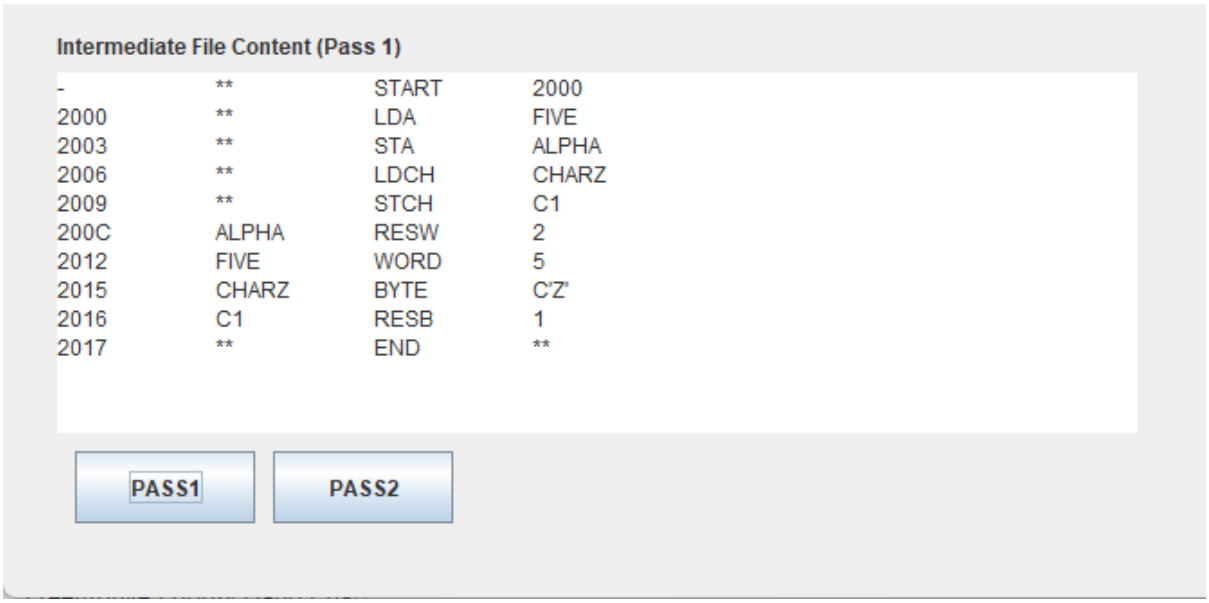
STCH 57

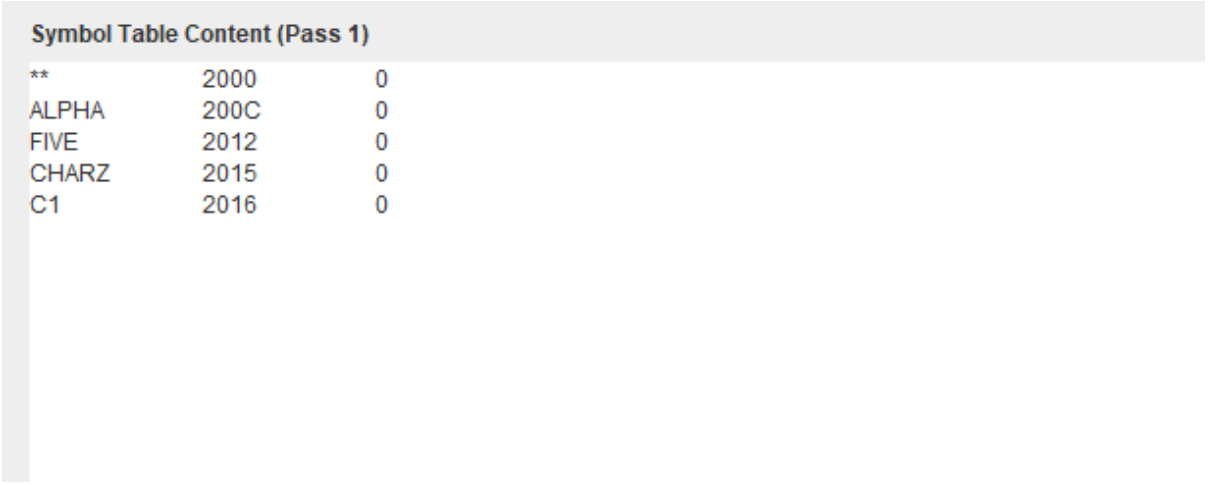
END *



Step 3: Execute Pass 1

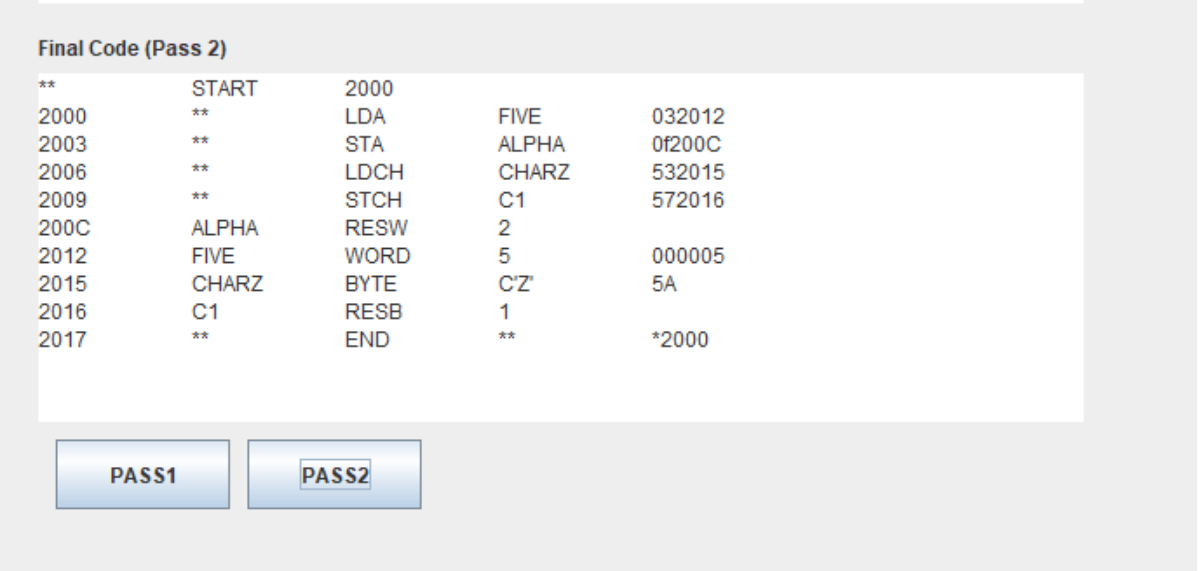
- Click the **PASS1** button to perform the first pass. This will:
    - Generate the intermediate file, which includes the address for each line of assembly code.
    - Build the symbol table (SYMTAB) mapping labels to memory addresses.
- **Output of Pass 1**:
    - The Intermediate/Final Output area will display the intermediate file content with addresses.
    - The Symbol/Object Code Table area will show the symbol table with labels and their respective addresses.





Step 4: Execute Pass 2

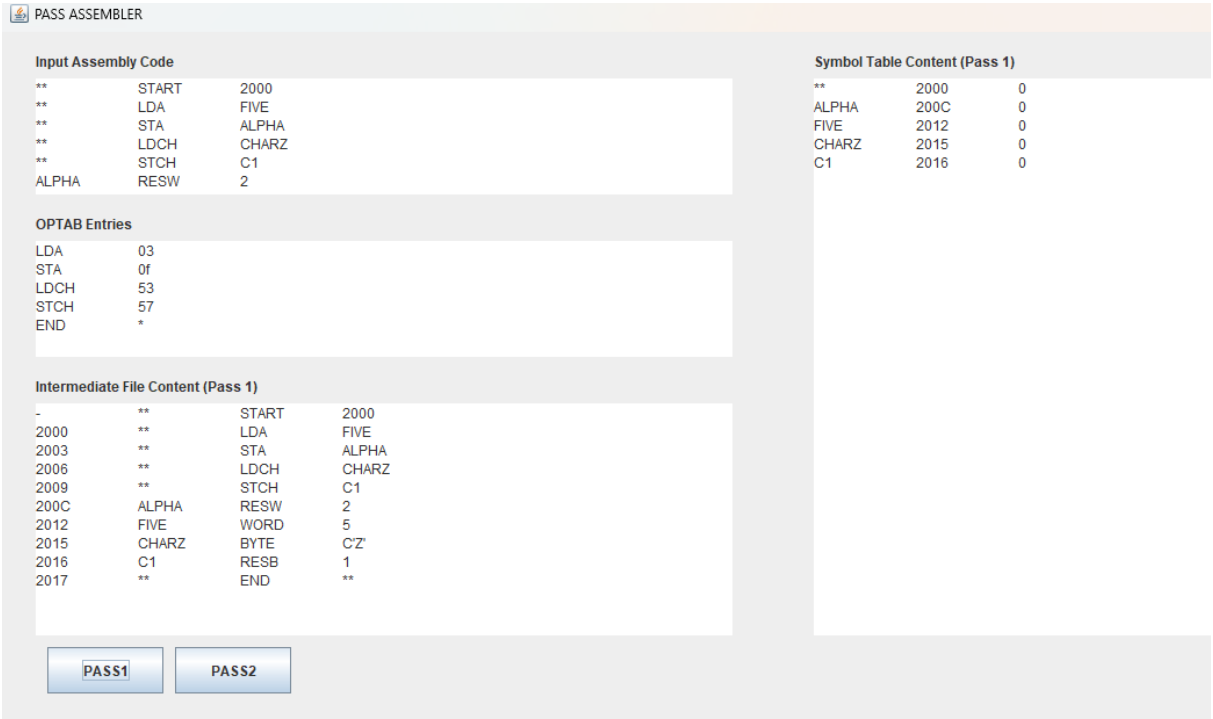
- Click the **PASS2** button to perform the second pass. This will:
    - Use the intermediate file and symbol table to generate object code for each instruction.
    - Display the final machine code (object code) for the program.
    - Create object code records, including the Header (H), Text (T), and End (E) records.
- **Output of Pass 2**:
    - The Intermediate/Final Output area will display the final object code along with the original assembly code.
    - The Symbol/Object Code Table area will display the object code records.



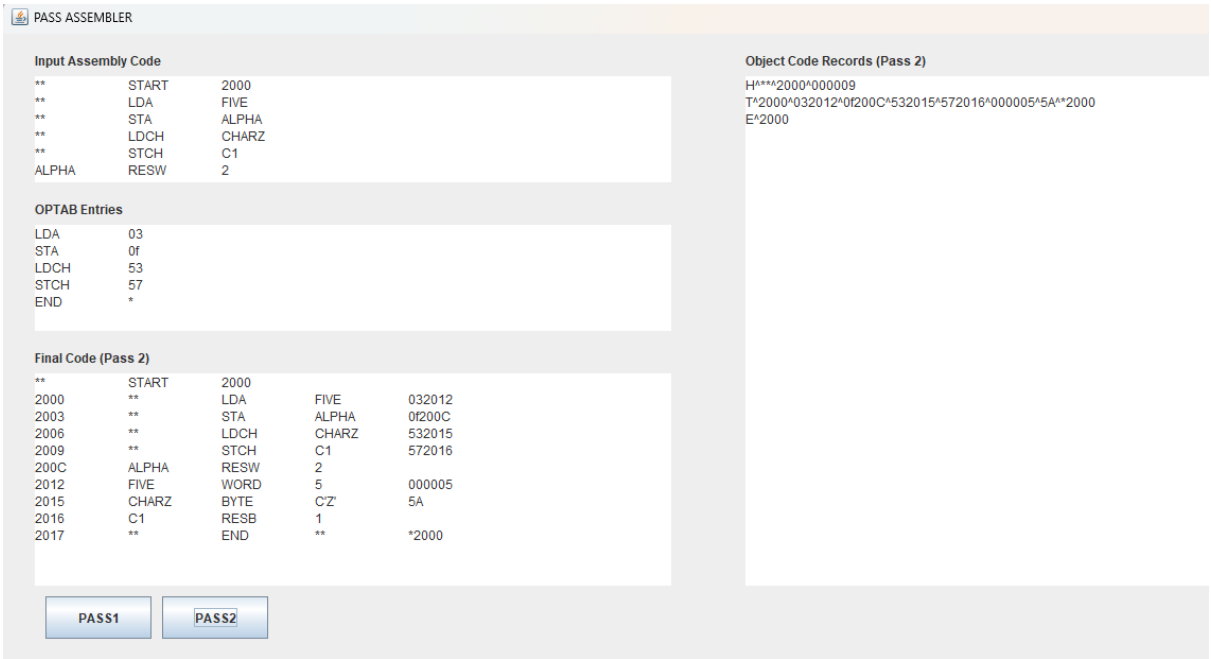
Object Code Records (Pass 2)

H^**^2000^000009
T^2000^032012^0f200C^532015^572016^000005^5A^*2000
E^2000

OUTPUT PASS 1



OUTPUT PASS 2



User Requirements

1. **Input Assembly Code**: The user should be able to input assembly language code for processing.
2. **OPTAB Entries**: The user should provide a list of operation codes (optab) that map mnemonics to their corresponding machine code.
3. **Execution of Passes**: The user can execute Pass 1 and Pass 2 of the assembler to generate intermediate code and object code respectively.
4. **Output Display**: The application must display intermediate output, symbol table, and final object code.
5. **Error Handling**: The application should gracefully handle errors and display appropriate messages to the user.

Developer Requirements

1. **Java Development Kit (JDK)**: The program must be developed using JDK 8 or higher.
2. **IDE**: An Integrated Development Environment (IDE) such as Eclipse or IntelliJ IDEA for easier development and debugging.
3. **Swing Library**: Use Java Swing for the graphical user interface (GUI) components.
4. **Version Control**: Using Git or any other version control system for managing code changes.

Software Requirements

1. **Operating System**: Windows, macOS, or Linux.
2. **JDK**: Version 8 or higher.
3. **Java Runtime Environment (JRE)**: Required for running Java applications.

Hardware Requirements

1. **Processor**: Minimum dual-core processor.
2. **RAM**: At least 4GB (8GB recommended).
3. **Storage**: Minimum of 500MB available for software and dependencies.
4. **Display**: Monitor with at least 1024x768 resolution for proper GUI visibility.

Pass 1 Logic Explanation

1. **Initialization**: Start by initializing the location counter (`LOCCTR`) to 0 and create data structures for symbol tables.
2. **Reading Input**: Split the input assembly code into lines and further split each line into label, opcode, and operand.
3. **Handling START Directive**: If the opcode is `START`, set the `LOCCTR` to the specified starting address.
4. **Symbol Table Population**: If a label is encountered, store it in the symbol table with its corresponding address.
5. **Locating Opcode**: Increment `LOCCTR` based on the type of opcode:
    - `WORD` increments by 3.
    - `RESW` increments by `3 * operand`.
    - `RESB` increments by the operand value.
    - `BYTE` increments by the length of the operand (minus any formatting characters).
6. **Output**: Construct and display the intermediate code and symbol table.

Pass 2 Logic Explanation

1. **Reading Intermediate Output**: Retrieve the intermediate code and symbol table from Pass 1.
2. **Opcode and Symbol Table Parsing**: Parse the OPTAB and SYMTAB to map mnemonics to their respective machine codes.
3. **Generating Object Code**: For each line in the intermediate code:
    - Translate the opcode into machine code using the OPTAB.
    - Resolve operands using the SYMTAB to find their addresses.
    - Construct the object code record.
4. **Creating Text Records**: Format and assemble the text records and header record for the final output.
5. **Output**: Display the final code and object code records.

Conclusion

The Pass Assembler program allows users to convert assembly language code into machine-readable object code through a two-pass process. Pass 1 generates an intermediate representation and a symbol table, while Pass 2 transforms the intermediate code into object code using an operation table. This application emphasizes modular design and error handling, making it a valuable tool for assembly language programmers. Future improvements could include support for more assembly language features and enhanced user interaction for ease of use.