GUI PASS ASSEMBLER

USER MANUAL

The **PASS ASSEMBLER** program is a simple assembler simulator that performs two passes over an assembly language input file. The first pass (`PASS1`) generates a symbol table and an intermediate file containing assembly instructions along with their locations. The second pass (`PASS2`) translates the assembly instructions into machine code, generating the final object code and a record file.

Components

Project Structure

- Main Class:`Passing.java`

- GUI Components:

- `JFrame`: Main window titled "PASS ASSEMBLER".

- `JButton`: Two buttons (`b1` for Pass1, `b2` for Pass2).

- `JTextArea`: Two text areas (`t1` and `t2`) for displaying outputs.

- `JLabel`: Two labels (`label1` and `label2`) for descriptions.

- **Functional Components:**

- ActionListener: Handles button click events to trigger `runPass1()` and `runPass2()`.

- Pass1 (`runPass1()`): Processes input and OPTAB files to generate `intermediate.txt` and `symtab.txt`.

- Pass2 (`runPass2()`): Processes `intermediate.txt` and `symtab.txt` to generate `final.txt` and `record.txt`.

Development Environment

- Programming Language: Java (JDK 8 or higher recommended)

- IDE:Visual Studio Code (VS Code) with Java extensions

-Libraries Used:

- `javax.swing.*`: For GUI components

- `java.awt.event.*`: For event handling

- `java.io.*`: For file operations

- `java.util.*`: For data structures like `HashSet` and `Scanner`

Code Structure and Workflow

1. GUI Initialization (`Passing()` Constructor):

- Sets up the JFrame and its components.

- Configures layout, visibility, and event listeners.

2. Event Handling (`actionPerformed` Method):

- Detects which button is clicked (`PASS1` or `PASS2`).

- Updates labels to reflect the current pass.

- Calls the corresponding pass method.

3. Pass1 (`runPass1()` Method):

- File Selection:

- Prompts the user to select `input.txt` and `optab.txt` using `JFileChooser`.

- Processing:

- Reads the input and OPTAB files.

- Generates `intermediate.txt` and `symtab.txt` with appropriate formatting.

- Output Display:

- Reads and displays the contents of the generated files in the text areas.

4. Pass2 (`runPass2()` Method):

- File Processing:

- Reads `intermediate.txt` and `symtab.txt`.

- Generates `final.txt` and `record.txt` containing the final code and object code records.

- Output Display:

- Reads and displays the contents of the generated files in the text areas.

Classes and Imports

- Imports: The program uses classes from the following packages:

- `javax.swing.*`: For creating the GUI components (JFrame, JButton, JTextArea).

- `java.awt.event.*`: For handling button click events.

- `java.io.*`: For reading from and writing to files.

- `java.util.*`: For utilizing `Scanner` for input and `HashSet` for managing unique symbols.

Main Class: `Passing`

The main class `Passing` encapsulates the GUI and the logic for the assembler's functionality.

Attributes

- JFrame f: Main window of the application.
- JButton b1, b2: Buttons to trigger `PASS1` and `PASS2`.
- JTextArea t1, t2: Text areas to display results and outputs.
- String result: Holds the length of the object code.
- StringBuilder objectCodeBuilder: Accumulates the object code during `PASS2`.

Constructor: `Passing()`

The constructor initializes the GUI components and sets up the layout:

- Creates the main frame, text areas, and buttons.
- Sets bounds for each component.
- Adds components to the frame.
- Configures the frame properties (size, visibility, close operation).
- Attaches action listeners to the buttons.

Methods

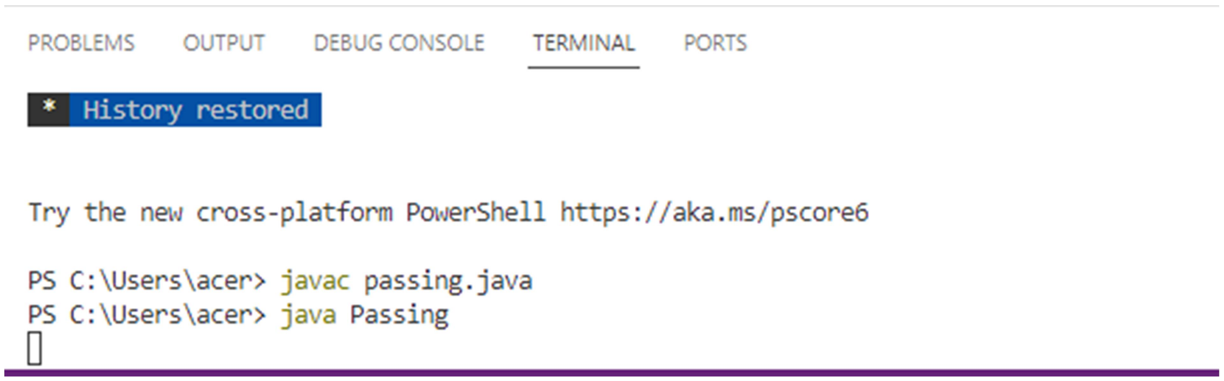
`actionPerformed(ActionEvent e)`

Handles button click events to determine which pass to run:

- Calls `runPass1()` if `b1` is clicked.

- Calls `runPass2()` if `b2` is clicked.

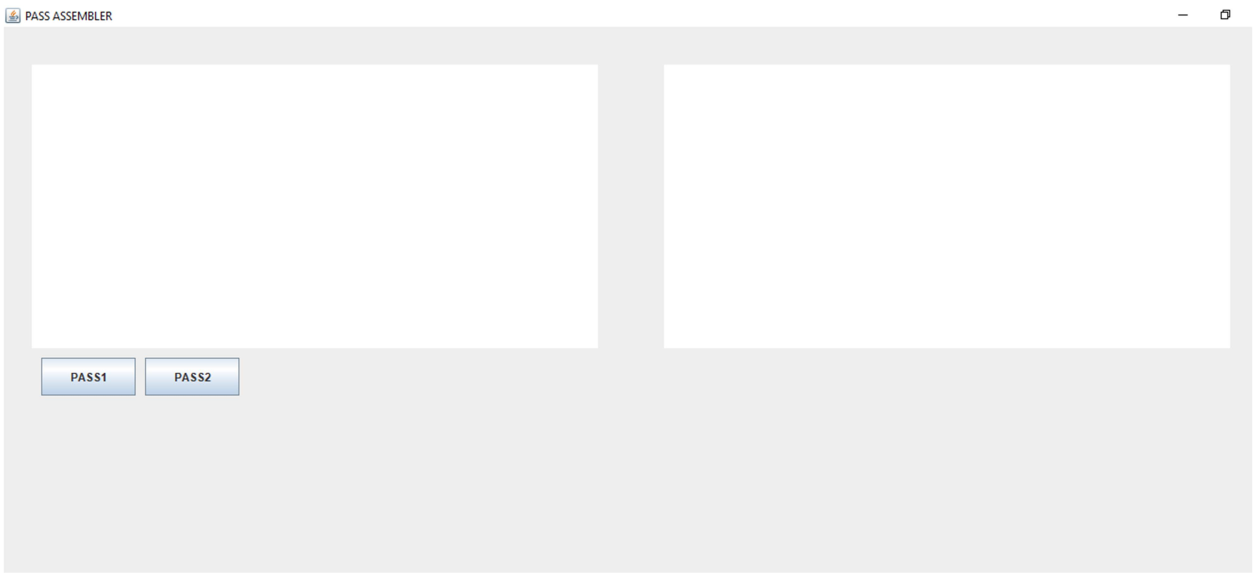
`runPass1()`



```
* History restored

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\acer> javac passing.java
PS C:\Users\acer> java Passing
```



This method performs the first pass of the assembler:

1. **File Selection**: Prompts the user to select an input file (assembly code) and an OPTAB file (opcode table).

2**. Initialization**: Initializes data structures and variables, including `HashSet<String> symtabSet` for storing unique symbols.

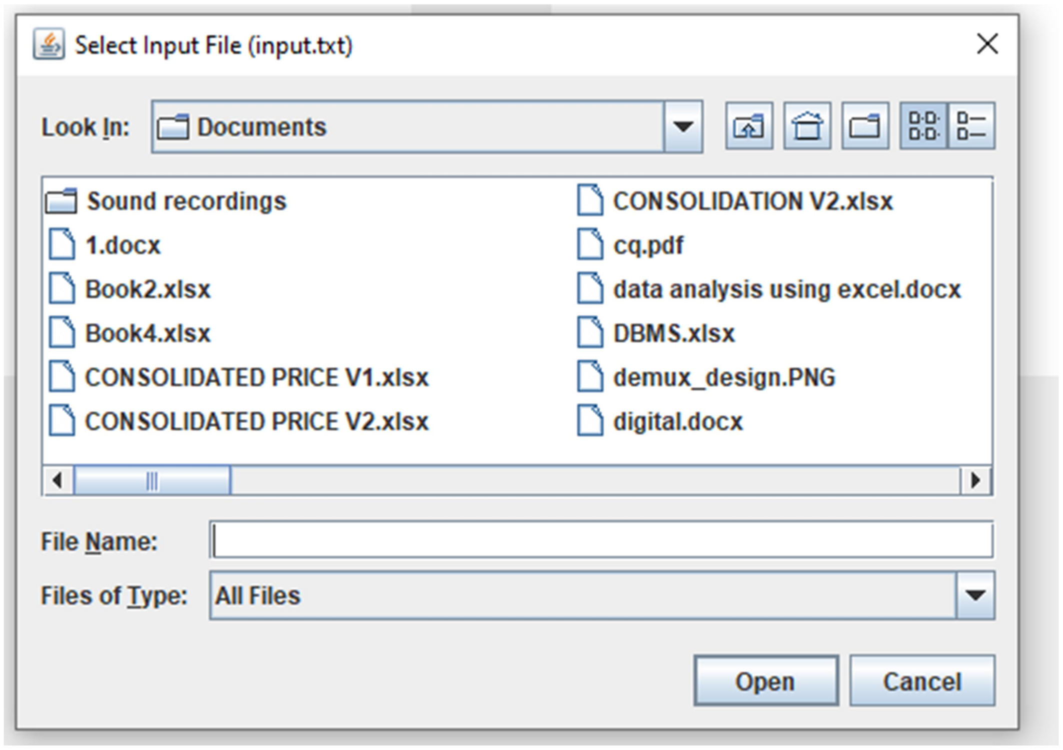
3. **Reading Input**: Reads the input file line by line:

- Processes labels, opcodes, and operands.

- Handles the `START` directive to set the starting address.

- Generates intermediate code and updates the symbol table (SYMTAB).

4**. Output:** Writes the intermediate file and symbol table to separate files and displays the intermediate content in `t1`.



`runPass2()`

This method performs the second pass of the assembler:

1. **File Handling**: Opens the previously generated files (OPTAB, SYMTAB, and intermediate file) for reading.

2. **Translation**: Reads the intermediate file to generate object code:

- Looks up opcodes in the OPTAB.

- Resolves operands using the SYMTAB.

- Handles special cases for directives like `WORD`, `BYTE`, and others.

3. **Output Generation:** Writes the final object code to `final.txt` and records to `record.txt`, then displays the results in `t1` and `t2`.

`main(String[] args)`

The entry point of the program. It creates an instance of the `Passing` class, initializing the GUI.

User Point of View Instructions

1. **Launch the Application**

- Navigate to the directory containing `passing.jar`.

- Double-click `passing.jar` to launch the application, or run the following command in the terminal:

```bash
java -jar passing.jar
```

2. **User Interface Components**

- **Intermediate File Content Area (Left Text Area):** Displays the content of the intermediate file or final code based on the pass.

- **Symbol Table Content Area (Right Text Area):** Shows the symbol table or object code records corresponding to the selected pass.

- Buttons:

- PASS1: Initiates the first pass of the assembler.

- PASS2: Initiates the second pass of the assembler.

Performing Assembly Passes

Pass 1: Generating Intermediate File and Symbol Table**

1. Click on the "PASS1" Button

- File Selection:

- Input File: A file containing the assembly code (e.g., `input.txt`).

- OPTAB File: A file containing the opcode table (e.g., `optab.txt`).

- Process:

- The assembler reads the input and OPTAB files.

- It generates an `intermediate.txt` file and a `symtab.txt` symbol table.

- Output:

- Intermediate File Content Area: Displays the contents of `intermediate.txt`.

- Symbol Table Content Area: Displays the contents of `symtab.txt`.

Pass 2: Generating Final Code and Object Code Records

1. Click on the "PASS2" Button

- Process:

- The assembler reads the previously generated `intermediate.txt` and `symtab.txt` files.

- It creates `final.txt` (final code) and `record.txt` (object code records).

- Output:

- Intermediate File Content Area: Displays the contents of `final.txt`.

- Symbol Table Content Area:Displays the contents of `record.txt`.

Closing the Application

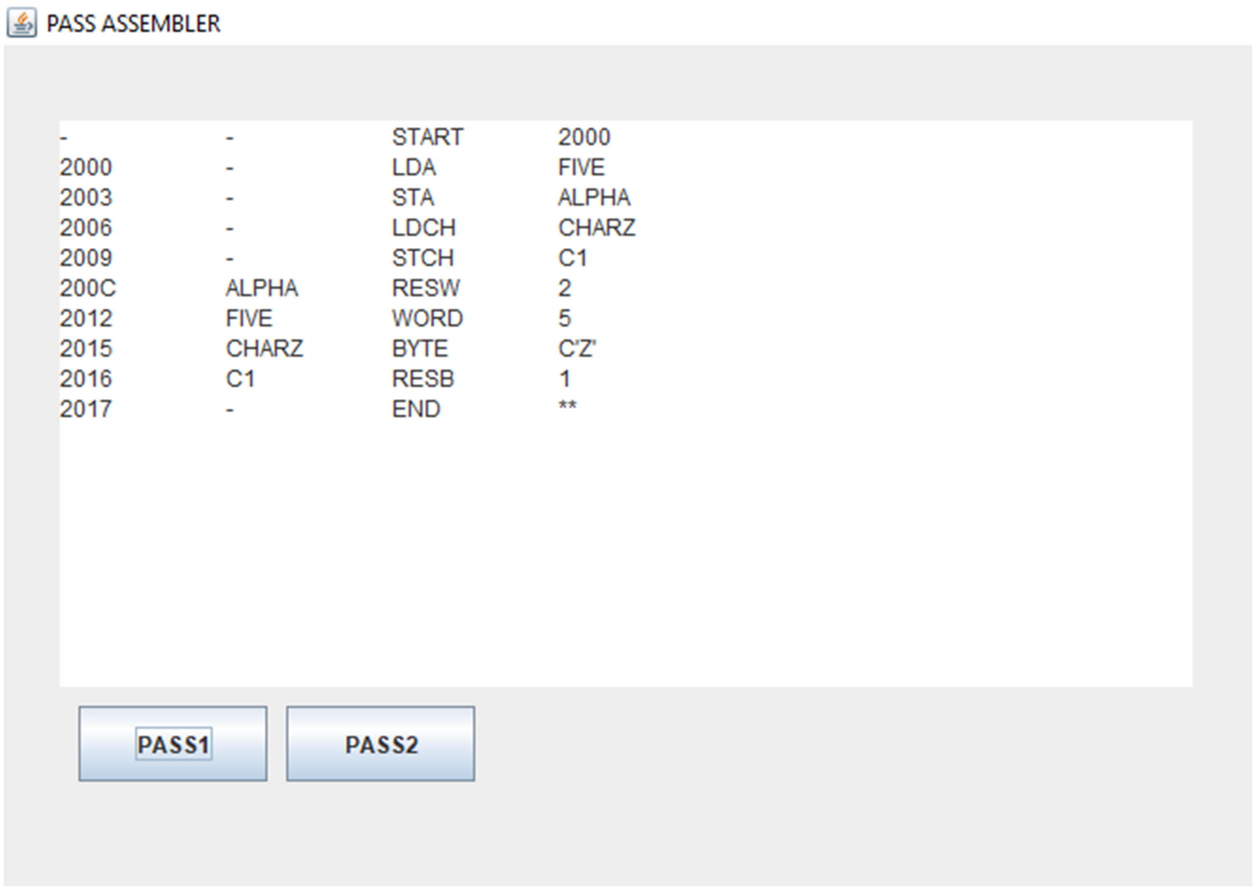
- Click the close button (typically the "X" at the top-right corner) of the application window to exit.

Example Files

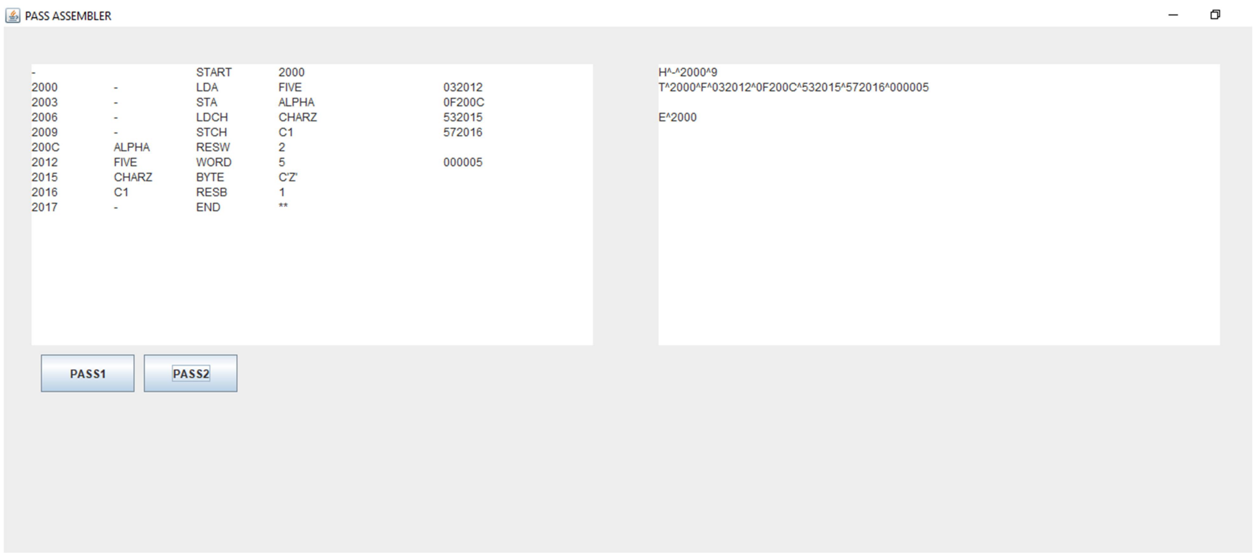
- input.txt: Should contain the assembly language instructions.
- optab.txt: Should define the opcode and their corresponding machine codes.

Conclusion

The PASS ASSEMBLER program serves as a basic assembler for educational purposes, allowing users to understand the process of assembling assembly language into machine code through a two-pass approach.


PASS ASSEMBLER

```
-       -       START   2000
2000    -       LDA     FIVE
2003    -       STA     ALPHA
2006    -       LDCH    CHARZ
2009    -       STCH    C1
200C    ALPHA   RESW    2
2012    FIVE    WORD    5
2015    CHARZ   BYTE    C'Z'
2016    C1      RESB    1
2017    -       END     **
```


PASS ASSEMBLER

```
-       -       START   2000
2000    -       LDA     FIVE            032012
2003    -       STA     ALPHA           0F200C
2006    -       LDCH    CHARZ           532015
2009    -       STCH    C1              572016
200C    ALPHA   RESW    2
2012    FIVE    WORD    5               000005
2015    CHARZ   BYTE    C'Z'
2016    C1      RESB    1
2017    -       END     **
```

```
H^-^2000^9
T^2000^F^032012^0F200C^532015^572016^000005

E^2000
```

INPUT

```
- START   2000
- LDA FIVE
- STA ALPHA
- LDCH    CHARZ
- STCH    C1
ALPHA   RESW    2
FIVE    WORD    5
CHARZ   BYTE    C'Z'
C1  RESB    1
-  END **
```

OPTAB

```
START   *
LDA 03
STA 0f
LDCH    53
STCH    57
END *
```

Github link: https://github.com/adithyasalesh/gui-pass-assembler.git