INTRODUCTION

An **assembler** is a tool that translates an assembly language program into machine code. A **two-pass assembler** processes the source code in two stages, or passes. This ensures the assembler can handle forward references and generate accurate object code.

In the **first pass (Pass One)**, the assembler:

1. Scans through the source code line by line to establish the locations of labels and instructions.
2. Creates a **symbol table** (`symtab`), associating labels with their corresponding memory addresses (location counter).
3. Generates an **intermediate file**, storing essential details (location counter, label, opcode, operand) for the second pass.
4. Calculates the total length of the program by updating the **location counter** (`locctr`), and handles different directives (e.g., `WORD`, `BYTE`, `RESW`, `RESB`) by updating memory accordingly.

In the **second pass (Pass Two)**, the assembler:

1. Reads the **intermediate file** and **symbol table** generated during Pass One.
2. Uses this information to generate the **object code** by converting mnemonics into their corresponding machine code, resolving memory addresses of symbols, and creating object program records (header, text, and end records).
3. Outputs an **object program** that can be loaded into memory and executed by the machine.

The two-pass approach is advantageous because it allows the assembler to handle forward references (labels or variables used before they are defined). By scanning the entire code first (Pass One), the assembler can ensure all symbol definitions are resolved when generating the final machine code (Pass Two).

This combined program demonstrates both passes of a two-pass assembler:

- **Pass One** handles the creation of the symbol table and intermediate file.
- **Pass Two** uses the intermediate data to generate the final object program in machine code.

The output includes multiple files representing each stage of the translation process, such as:

- The **intermediate file**, which stores partially processed instructions.
- The **symbol table**, mapping symbols to their addresses.
- The **object program**, which is the final machine code ready for execution

This program implements a two-pass assembler with a graphical user interface (GUI) using Java. It allows users to input assembly language code, process it through two passes, and generate corresponding outputs such as the symbol table, intermediate file, and object code. The assembler performs two main tasks: in Pass 1, it scans the assembly code to build a symbol table and generate an intermediate representation; in Pass 2, it uses this data to produce the final object code. The GUI facilitates user interaction, displaying the input, intermediate steps, and final output in separate text areas.

H/W REQUIREMENT

The hardware requirements for running an assembler project with a Java-based GUI and assembly logic integration are relatively modest, as both Java applications and assemblers are lightweight compared to more resource-intensive software. However, to ensure smooth development and execution, the following hardware specifications are recommended:

1. Processor (CPU):

- **Minimum**: Dual-core processor (e.g., Intel Core i3 or AMD equivalent)
- **Recommended**: Quad-core processor (e.g., Intel Core i5 or AMD Ryzen 5)
- For a smooth development experience, especially when running the assembler logic and Java GUI simultaneously, a multi-core processor is preferred.

2. Memory (RAM):

- **Minimum**: 4 GB of RAM
- **Recommended**: 8 GB of RAM or higher
- More memory is helpful for handling multiple processes like the Java IDE, testing, and running the assembler at the same time, especially if your project involves large input files or multiple components.

3. Storage (HDD/SSD):

- **Minimum**: 250 GB of free disk space
- **Recommended**: 500 GB of free disk space or more
- **SSD (Solid-State Drive)**: For faster read/write speeds, project builds, and application performance, an SSD is highly recommended. It will also improve the speed of compiling Java code and running assembly files.

4. Graphics Card (GPU):

- **Minimum**: Integrated graphics (e.g., Intel HD Graphics or AMD Radeon integrated GPUs)
- **Recommended**: Dedicated GPU is not necessary unless you are working with high-performance visualization or advanced JavaFX GUI elements, in which case an entry-level dedicated GPU (e.g., NVIDIA GeForce GTX 1050 or AMD Radeon RX 560) would be sufficient.
- Java GUI components (Swing or JavaFX) do not generally require high-end graphical processing power.

5. Monitor/Display:

- **Minimum Resolution**: 1280x720 pixels
- **Recommended Resolution**: 1920x1080 pixels or higher (Full HD)
- A higher resolution monitor can help with development, allowing you to view more code and manage different windows like the IDE, GUI design, and terminal.

6. Input Devices:

- **Keyboard and Mouse/Trackpad**: Standard input devices are required for coding and GUI testing. You may also use additional devices for testing the user interface, but nothing special is required for basic development.

7. Network:

- **Internet Connection**: Required for downloading software (JDK, IDE, etc.), managing dependencies, and possibly using version control (Git). It's also necessary for online collaboration or accessing repositories like GitHub.
- While not critical for running the project, a stable internet connection is helpful during the development process.

8. Optional Peripherals:

- **External Storage (USB/External SSD)**: If you need to backup your project files, external storage can be useful, especially for large or multiple versions of your project.
- **Dual Monitor Setup**: For increased productivity, using dual monitors can help you manage the IDE, GUI design, and terminal windows simultaneously.

9. Cooling:

- A basic cooling system should suffice since the project isn't very resource-intensive. However, if your machine tends to overheat during heavy usage (e.g., compiling large codebases), consider using a laptop cooling pad or ensuring proper ventilation for desktop systems.

10. Power Supply:

- **Laptop**: Ensure your laptop has sufficient battery life or stay connected to a power source during long development sessions, especially if you're using an integrated development environment (IDE).
- **Desktop**: A standard power supply unit (PSU) is sufficient, but for a system with higher-end components (e.g., SSDs, multi-core CPUs), ensure your PSU can provide adequate power.

Summary of Recommended Hardware:

- **Processor**: Quad-core (e.g., Intel i5 or AMD Ryzen 5)
- **RAM**: 8 GB or more
- **Storage**: 500 GB SSD (preferred) or HDD
- **GPU**: Integrated graphics (dedicated GPU optional)
- **Display**: 1920x1080 resolution
- **Internet**: Stable connection for downloads and collaboration

These hardware requirements ensure that the project runs smoothly, including the Java IDE for development, the assembler logic, and the Java GUI components.

S/W REQUIREMENT

To successfully run a project that involves **Pass One** and **Pass Two** of an assembler program, combined with a **Java GUI** for managing the user interface and assembler logic, the following software requirements must be fulfilled:

1. Operating System:

- **Windows, macOS, or Linux**: The project can be developed and executed on any major operating system, depending on your development environment preference.
- Ensure that your OS supports both Java and the assembler toolchain you plan to use.

2. Java Development Kit (JDK):

- **JDK 8 or later**: A complete Java development environment is necessary to compile and run the Java code that provides the GUI and integrates with the assembler logic.
- You can download the JDK from Oracle JDK or use an open-source distribution like OpenJDK.

3. Java Integrated Development Environment (IDE):

- **Eclipse, IntelliJ IDEA, or NetBeans**: An IDE will help you write, debug, and manage your Java project efficiently.
- Choose an IDE that supports GUI development, so you can easily design and integrate the components for the project.

4. Assembler Tool:

- **Assembler Software**: If you are using actual assembly code (e.g., NASM, MASM, or any other assembler), ensure you have an assembler tool installed for testing the assembler code independently.
- **Assembler Library (Optional)**: If the assembler logic is integrated into the Java project, you may use or develop an assembler library in Java that handles the two-pass assembly internally.

5. Java GUI Libraries:

- **Swing or JavaFX**: If your project involves Java GUI components for visualizing tables, interacting with user input, and displaying the assembler's output, you will need libraries like Swing (built into the JDK) or JavaFX for building the GUI.
    - **Swing**: Part of the Java Standard Library, good for lightweight desktop applications.
    - **JavaFX**: A more modern option for building richer UIs in Java. It may require additional installation depending on your JDK version.

6. Assembler Logic Integration:

- **Custom Assembler Classes in Java**: You may need to write custom Java classes that handle the assembler logic for both Pass One and Pass Two. These classes will likely involve file handling, parsing, symbol tables, and memory address calculations.
- Ensure you have appropriate file handling capabilities in Java for managing input assembly files and generating output object code files.

7. Testing and Debugging Tools:

- **JUnit (for Java Unit Testing)**: To ensure the correctness of both your Java code and the assembler logic, you can use JUnit to write and run test cases.
- **Debugger in IDE**: For debugging Java and assembler logic, your chosen IDE will likely have a built-in debugger to help identify and fix issues during development.

8. Assembler Input Files:

- **Assembly Language Source Files**: You need sample or actual assembly code files to test the assembler. These files will serve as input for Pass One and Pass Two of the assembler.

9. Version Control:

- **Git**: For source code management and collaboration, use Git for version control. Hosting platforms like GitHub or GitLab will allow you to manage your project and work with others more effectively.

Optional Software:

- **Text Editors**: Simple text editors like Notepad++ or Sublime Text can be used to write and modify assembly code.
- **Build Tools (Maven/Gradle)**: If your project involves dependencies or follows a modular structure, a build tool like Maven or Gradle can help manage dependencies and automate building the project.

By ensuring that all these software requirements are met, you will be able to develop, compile, and run your assembler project successfull

USER MANAUL/DOCUMENTS

This user manual provides step-by-step instructions for setting up, running, and interacting with the assembler project that integrates **Pass One** and **Pass Two** logic with a Java-based Graphical User Interface (GUI).

Table of Contents:

---

1. Introduction

The assembler project is designed to process assembly language programs using a two-pass assembler method. It reads assembly code and converts it into machine-readable object code. The project is built with a **Java-based GUI** to help users interact with the assembler more intuitively. The GUI allows users to input assembly code, view the generated symbol table, and see the final object code.

---

2. System Requirements

Software Requirements

- **Java Development Kit (JDK)**: Version 8 or higher.
- **Java IDE**: (e.g., IntelliJ IDEA, Eclipse, NetBeans) for compiling and running the project.
- **Assembler Tool (Optional)**: NASM/MASM, if you need to run assembly code outside the project.
- **Swing or JavaFX**: For the GUI components of the project.

Hardware Requirements

- **Processor**: Dual-core (minimum), Quad-core (recommended)
- **RAM**: 4 GB (minimum), 8 GB (recommended)
- **Storage**: 250 GB of free space, preferably on SSD
- **Display**: Minimum 1280x720 resolution; recommended 1920x1080 resolution

---

3. Installation Guide

Installing Java and IDE

1.

Install the JDK:

2.

   - Download the latest Java Development Kit (JDK) from Oracle or install OpenJDK.
   - Follow the instructions to install Java, and set the environment variables if needed (e.g., set the `JAVA_HOME` and add it to your system's `PATH`).

3.

Install an IDE:

4.

   - Download and install any Java IDE like **Eclipse**, **IntelliJ IDEA**, or **NetBeans**.
   - Ensure the IDE is configured with the JDK you installed.

Installing Assembler Tools (Optional)

- If you plan to run assembly code separately, install an assembler like NASM or MASM from their respective websites.

---

4. Running the Project

Starting the Program

1.

Open the Project in the IDE:

2.

- Clone or download the project repository.
- Open it in your IDE by selecting **File → Open Project** and navigating to the project folder.

3.

Build the Project:

4.

- Use the IDE's build system to compile the project. For example, in **IntelliJ IDEA** or **Eclipse**, right-click on the project and select **Build** or **Run**.

5.

Run the Main Class:

6.

- The main entry point for the project is in the `Main.java` file.
- Run this file to launch the program. The Java GUI window should open, presenting the assembler interface.

Using the GUI Interface

1.

Input Assembly Code:

2.

- In the GUI, there is a text area labeled "Input Assembly Code".
- Paste or type in the assembly code you want to assemble.

3.

Execute Pass One:

4.

- Click the **Pass One** button to begin the first pass of the assembler.
- The system will parse the assembly code and generate the **symbol table**.

5.

Execute Pass Two:

6.

- Click the **Pass Two** button to proceed to the second pass.
- This will resolve the memory addresses and generate the final object code.

7.

View Results:

8.

- The results will be displayed in two sections:
    - **Symbol Table**: Shows the labels and their corresponding memory addresses.
    - **Object Code**: Shows the generated machine code or object code.

9.

Save Output (Optional):

10.

- The GUI allows you to save the output to a file by clicking the **Save** button and choosing a destination.

---

5. Understanding the Input and Output

Input: Assembly Code

- Input is expected to be valid assembly language instructions. Labels, mnemonics, literals, and comments are supported.
- Example input:
-

sql

-
-

Copy code

•
•

```
START 100
LOOP  MOVER AREG, ='5'
      ADD AREG, ='1'
      MOVEM AREG, X
X     DS 1END
```

•
•

Output: Symbol Table and Object Code

•

Symbol Table: A list of symbols (labels) defined in the code, with their memory addresses. Example:

•
•

css

•
•

Copy code

•
•

```
Symbol   Address
LOOP     100
X        104
```

•
•
•

Object Code: The final machine code generated from the assembly code. Example:

•
•

Copy code

-
-

```
00 01 05
01 01 01
02 02 00 104
```

-
-

---

6. Troubleshooting

- **Error: JDK Not Found**: Ensure that the JDK is properly installed and the `JAVA_HOME` environment variable is set correctly.
- **GUI Not Launching**: Ensure that all necessary Java libraries (Swing or JavaFX) are available. Rebuild the project if necessary.
- **Incorrect Object Code**: Double-check your input assembly code for syntax errors. Ensure that labels are defined properly, and all instructions are valid.
- **Performance Issues**: If the program is slow, ensure that your system meets the minimum hardware requirements and close unnecessary background applications.

---

7. FAQ

Q1: Can I run this project on any operating system?

- Yes, as long as you have Java installed, the project should run on Windows, macOS, or Linux.

Q2: How can I test my assembly code independently of the project?

- You can use NASM or MASM assemblers to test assembly code independently, but ensure the syntax matches the one expected by your assembler.

Q3: Can I modify the GUI layout?

- Yes, you can modify the GUI by editing the Java Swing/JavaFX components in the project. These components are located in the `Main.java` or associated files.

This concludes the user manual for running the assembler project. Follow these steps to set up and run your project smoothly. If you encounter issues, refer to the troubleshooting section or consult online resources for additional support.

NECESSARY ITEMS FOR RUNNING THE PROJECT

To run your assembler project (which integrates **Pass One** and **Pass Two** logic with a Java GUI), you need several necessary items. Below is a list of essential components:

1. Assembly Code File (Input)

- **Assembly Language Source File**: This is the input file that contains the assembly code you want to assemble. The file typically contains labels, instructions, and directives written in assembly language.
  - Example:

asm

Copy code

```
START 100
LOOP  MOVER AREG, =’5’
      ADD AREG, =’1’
      MOVEM AREG, X
X     DS 1
END
```

2. Java Development Kit (JDK)

- **JDK 8 or Later**: The Java Development Kit is necessary for compiling and running the Java code, especially if the project uses Java Swing or JavaFX for GUI development.
- Install from Oracle or use **OpenJDK**.

3. Integrated Development Environment (IDE)

- **IDE (Eclipse, IntelliJ IDEA, or NetBeans)**: A Java IDE will help you build, debug, and run the project. It allows you to manage the Java source files, handle dependencies, and execute the project.

- Install your preferred IDE and ensure it's configured with the correct JDK.

4. Assembler Logic Code (Java Classes)

- **Pass One & Pass Two Implementation**: These are Java classes that contain the logic for the two-pass assembler. The code reads the assembly input, processes it, generates the symbol table, and outputs the object code.
- Make sure the project includes these essential Java files:
  - `PassOne.java`: Handles the first pass of the assembler, including symbol table generation.
  - `PassTwo.java`: Handles the second pass, resolving addresses and generating object code.

5. Java GUI Components

- **Swing/JavaFX Components**: If the project uses a graphical user interface (GUI), the Java GUI components should be part of the project. These components allow users to input the assembly code and view the symbol table and object code.
- Common GUI files may include:
  - `Main.java`: The entry point of the project with the GUI implementation.
  - Other helper classes like `AssemblerGUI.java`, depending on how your project is structured.

6. Assembler Tables (Symbol Table, Literal Table, etc.)

- **Symbol Table**: A data structure that stores the labels and their corresponding memory addresses.
- **Literal Table**: A table that stores literal values and their assigned memory locations (if applicable).
- These tables are usually created dynamically by the Java classes during the execution of **Pass One**.

7. Opcode Table

- **Opcode Table (Internal or External)**: This is a reference table that maps assembly mnemonics to their corresponding machine code instructions. It is either stored within the project as part of the assembler logic or accessed via an external file.
- Example:
-

```
text
```

-
-

Copy code

-
-

```
MOVER  00
MOVEM  01
ADD    02
```

-
-

8. Text Editor (Optional)

- **Text Editor**: While the IDE handles Java code, you might use a simple text editor like **Notepad++, Sublime Text**, or **VS Code** to edit or create assembly code files.

9. Testing Files

- **Sample Assembly Code Files**: Have sample assembly programs prepared for testing. These should include different instructions and labels to test all functionalities of the assembler.
- Example input files:
  - `test1.asm`: Basic assembly code with a few instructions.
  - `test2.asm`: A more complex assembly program with loops and literals.

10. Version Control (Optional)

- **Git**: To manage your project's source code, you can use Git for version control. Set up a repository on GitHub, GitLab, or another version control system to track changes and collaborate with others.

11. Dependencies (if using external libraries)

- **JavaFX Library (if applicable)**: If you're using JavaFX for the GUI, ensure the JavaFX libraries are correctly installed and linked to your project.
- **Maven or Gradle (optional)**: If your project involves dependencies or uses a build tool, ensure that **Maven** or **Gradle** is set up to handle these dependencies.

12. External Assembler Tool (Optional)

- **NASM, MASM, or Other Assemblers**: If you want to verify your assembly code independently of your Java project, having an external assembler (like NASM or MASM) installed is useful. This can help test your assembly programs in a real-world environment.

Summary of Necessary Items

1. **Assembly Code File** (Input)
2. **Java Development Kit (JDK)**
3. **IDE (Eclipse, IntelliJ IDEA, NetBeans)**
4. **Pass One & Pass Two Java Classes**
5. **Java GUI Components (Swing or JavaFX)**
6. **Symbol Table & Literal Table**
7. **Opcode Table (Internal or External)**
8. **Text Editor (Optional)**
9. **Testing Files** (Sample assembly programs)
10. **Git/Version Control (Optional)**
11. **JavaFX or Other Dependencies (if applicable)**
12. **External Assembler Tool (Optional)**

These items ensure you can develop, run, and test your assembler project with both the Java-based GUI and assembler logic.