Two Pass Assembler

This document provides an overview and information about Two pass Assembler.

1.Introduction

Two pass assembler is used to convert the input file(assembly file) to machine code with the help of a optab,symtab and intermediate file where the optab is provided as input and the symtab and intermrdiate files are intermediate outputs.This program helps the users to generate the machine code easely.

2.Overview of Two pass Assembler

Two pass assembler contains two passes pass1 and pass2.

Pass1:Creates Symbol table and generates address for each instruction in the input file.

Pass2:Uses the symbol and generates the machine code.

3.An Overview of the code

1.It contains 3 Buttons

1. Assembly Code: Allows the user to select and load the input Code file
2. Load OPTAB: Allows the user to select and load the optab
3. Assemble: Executes both Pass 1 and Pass 2 to generate machine code

2.Action Listeners:

- **LoadAssemblyAction**: Handles loading the assembly code file using a JFileChooser.
- **LoadOptabAction**: Handles loading the OPTAB file using a JFileChooser. The OPTAB is loaded into a HashMap where each operation is mapped to its corresponding machine code.
- **AssembleAction**: Executes Pass 1 and Pass 2 when the "Assemble" button is clicked. It ensures both files are loaded before proceeding.

3.Data Structures:

- symtab: A HashMap that stores symbol names (labels) and their corresponding addresses (location counter values) during Pass 1.
- optab: A HashMap that stores operation mnemonics and their corresponding machine code.
- intermediateFile: A StringBuilder that stores the intermediate file generated during Pass 1.
- machineCode: A StringBuilder that stores the generated machine code during Pass 2.

4.Pass 1:

- In Pass 1, the assembler processes the assembly code line by line to build the symbol table and an intermediate file with the addresses for each instruction.
- **Location Counter:** A hexadecimal counter that keeps track of memory locations.
- **SYMTAB Generation:** Each label in the assembly code is associated with its memory address (location counter) and stored in the SYMTAB.
- **Start Address:** If the START directive is found, the assembler initializes the location counter to the specified address.

5.Pass 2:

- In Pass 2, the assembler reads the intermediate file generated in Pass 1 and generates the corresponding machine code.
- **Object Code Generation:** The assembler looks up each instruction in the OPTAB to get the machine code and uses the SYMTAB to resolve operand addresses.
- **Text Records:** Object code is organized into text records with a maximum of 30 bytes.
- **Header Record and End Record:** A header record is generated at the beginning, and an end record marks the end of the program.
- **Key Methods:**
- **loadOptab(File file):** Reads the OPTAB file and populates the optab HashMap with mnemonic-opcode pairs.
- **executePass1():** Processes the assembly code, updating the location counter and generating the intermediate file and symbol table.
- Skips comments and handles different types of directives (START, END, WORD, RESW, RESB, and BYTE).
- Updates the location counter based on the instruction type.

6.updateLocationCounter(String opcode, String operand):

Updates the location counter based on the instruction type and operand.

7.executePass2():

- Reads the intermediate file and generates machine code.
- Builds text records and handles the generation of object code for different instructions (WORD, BYTE, etc.).
- Constructs the final object program with a header, text, and end records.

8.displayOutput():

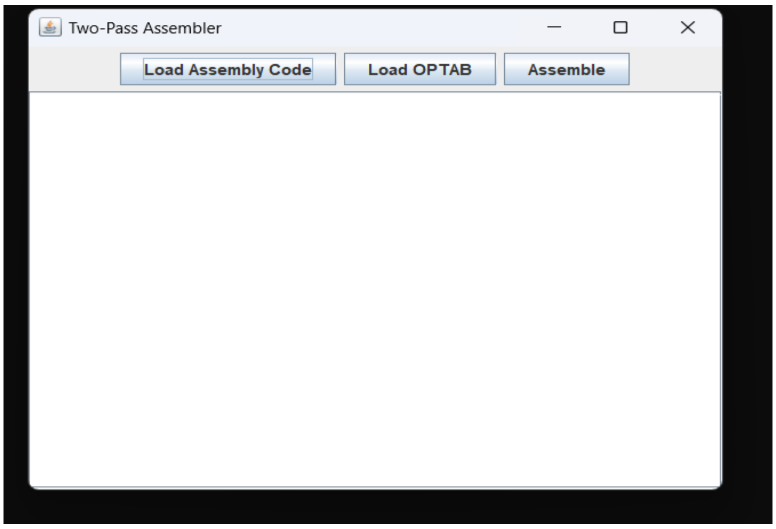
- Displays the intermediate file, symbol table, and machine code in the text area.

4.Steps to run the program

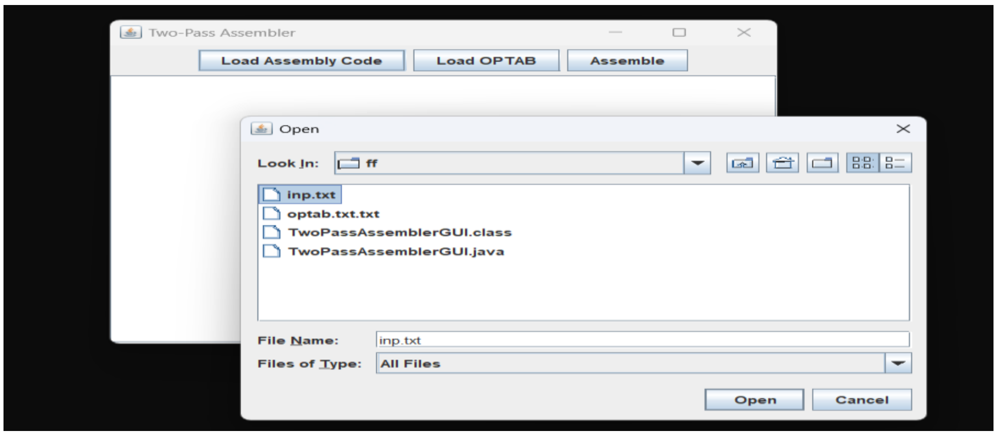
1.Open the code in any of the text editor like sublime,visual studio etc

2.compile and run the program(Ensure that you have downloaded jdk In your system).

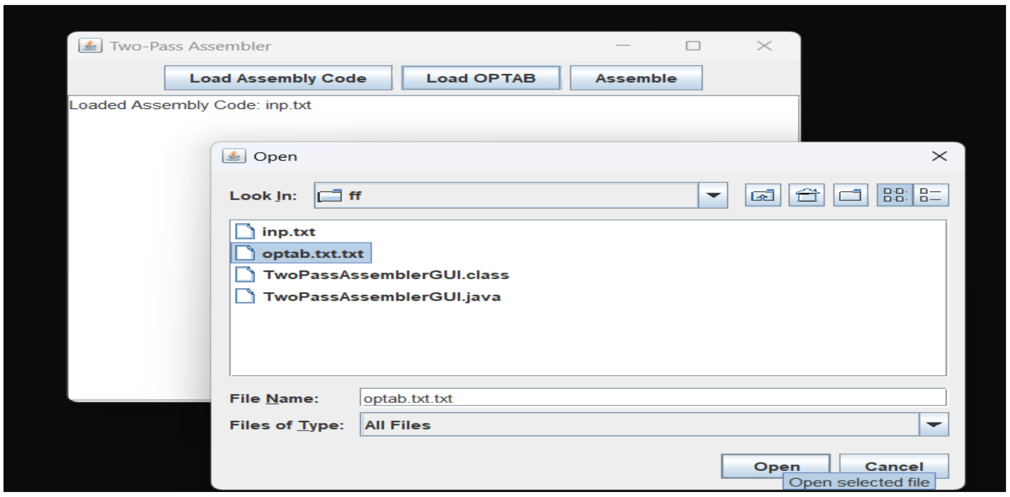
3.Then the following GUI Interface will appear on your screen



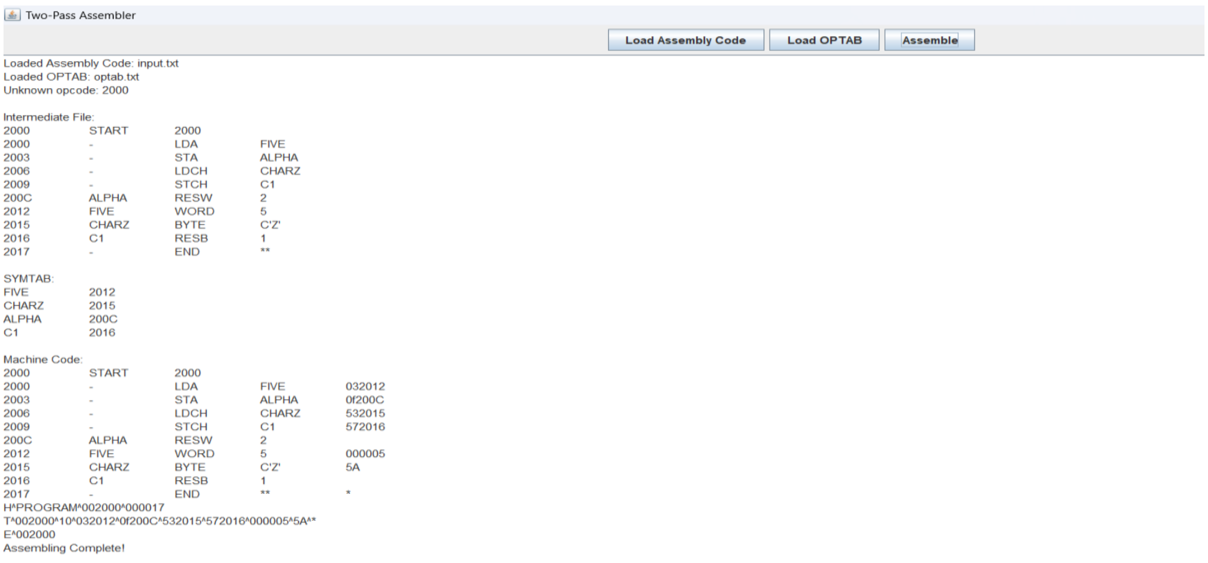
4.Now click on Load Assembly code to load the input file .



5.After loading input file click on load OPTAB



6.After loading these two files click on ASSEMBLE. On clicking on ASSEMBLE you will be able to see the interface with intermediate file,symtab and machine code according with the input file and optab provided.



```
Loaded Assembly Code: input.txt
Loaded OPTAB: optab.txt
Unknown opcode: 2000

Intermediate File:
2000        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         **

SYMTAB:
FIVE        2012
CHARZ       2015
ALPHA       200C
C1          2016

Machine Code:
2000        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'        5A
2016        C1          RESB        1
2017        -           END         **          *
H^PROGRAM^002000^000017
T^002000^10^032012^0f200C^532015^572016^000005^5A^*
E^002000
Assembling Complete!
```

5.Requirements

1.User Requirements

1.There must appropriate text editors for running the program.

2.JDK must be installed in system(JDK 21)

3.input files and optab must be saved in a folder where you wish to save the java source program

2.Devoloper requirements

1. The code should be well-structured and commented for ease of understanding and future maintenance.

2. The GUI should be simple, intuitive, and easy for users with basic assembler knowledge to interact with.

3. The program should run on any machine with a standard Java Runtime Environment

4. The assembler must handle reasonably large assembly files and OPTABs efficiently.

5. Display a clear message if either the assembly file or OPTAB file fails to load.

Conclusion

The TwoPassAssemblerGUI program simulates a two-pass assembler with a graphical interface, allowing users to assemble simple programs. It reads an OPTAB and assembly code, processes them in two passes, and generates machine code while displaying all intermediate steps.

Github: https://github.com/aidashaji/Two-pass-assembler-GUI-JAVA