USER MANUAL TO

PASS 1 AND PASS 2 ASSEMBLER

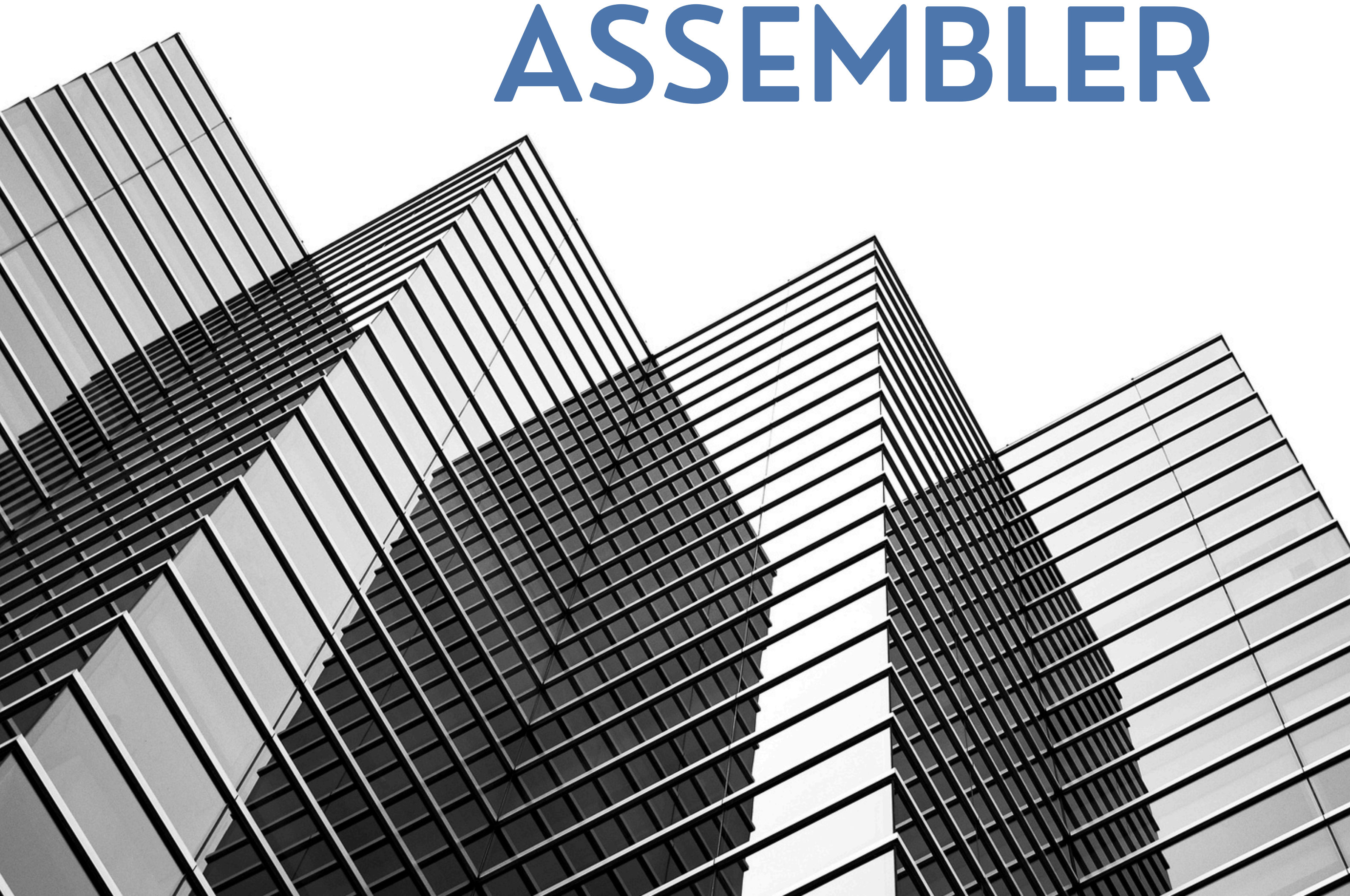


Table of Contents

Introduction

This assembler program is a two-pass assembler that processes SIC (Simplified Instructional Computer) assembly language files. It takes an input assembly file (input.asm) and an opcode table (optab.txt) to generate the intermediate code, symbol table, and final object code. The assembler is designed to handle basic assembly instructions and directives, such as START, WORD, BYTE, RESW, and RESB.

The graphical interface is built using Python's Tkinter library for ease of use, enabling users to select input files, run the assembler, and view the results.

Features

- Two-Pass Assembler: The assembler performs two passes over the source file:
- Pass 1: It generates the symbol table and intermediate code.
- Pass 2: It generates the object code based on the symbol table and opcode table.
- Object Code Generation: The assembler produces object code in the SIC format, including:
- Header Record (H): Contains the program name, starting address, and program length.
- Text Record (T): Contains the starting address, length, and the object code.
- End Record (E): Specifies the starting execution address.
- User-friendly GUI: The program offers an easy-to-use interface for selecting the input files, running the assembler, and viewing intermediate code, symbol table, and object code.

Installation

- Requirements:
- Python 3.x
- Tkinter (comes pre-installed with Python on most systems)
- A text editor to create the assembly program and opcode table (e.g., Notepad, Sublime Text, etc.)
- Running the Program:
- Download the assembler code.
- Run the Python script using:

```
python assembler_gui.py
```

Input Files

1. Assembly File (input.asm): This file contains the assembly language code. It should follow the format where each line contains:

- A label (optional)
- An opcode (instruction)
- An operand

2. Opcode Table File (optab.txt): This file contains the list of opcodes and their corresponding machine codes in the following format needed

OUTPUT FORMAT

- Header Record (H):
  - H^program_name^starting_address^program_length
  - program_name: The name of the program
  - starting_address: The starting address of the program (hexadecimal).
  - program_length: Total length of the program (hexadecimal).
- Text Record (T):
  - T^starting_address^record_length^object_code
  - starting_address: The address where this segment starts (hexadecimal).
  - record_length: The length of the object code in bytes (hexadecimal).
  - object_code: The assembled object code in hexadecimal.
- End Record (E):
  - E^starting_address
  - starting_address: The address where the program execution starts (hexadecimal).

GUI Instructions

1. Opening the GUI:
   - Run the program using the command:

```
python assembler_gui.py
```

   - A window titled ---BELDA'S PASS1 AND PASS2 ASSEMBLER--- will open.
2. Loading Input Files:
   - Input Assembly File: Click the Browse button next to the INPUT FILE field to select your assembly source file (e.g., input.asm).
   - Opcode Table File: Click the Browse button next to the OPCODE FILE field to select your opcode table file (e.g., optab.txt).
3. Running the Assembler:
   - After selecting the files, click the Run Assembler button. The assembler will process the files in two passes and generate:
     - Intermediate Code: The intermediate representation of the assembly code.
     - Symbol Table: The symbol table that maps labels to memory addresses.
     - Object Code: The final object code including the header, text, and end records.

4. Viewing Results:
   - The GUI displays the results in three sections:
     - Intermediate Code: Shows the intermediate file with location counters, labels, opcodes, and operands.
     - Symbol Table: Displays labels and their corresponding memory addresses.
     - Object Code: Displays the final object code formatted for SIC assembly.

Error Handling

- Missing Input Files: If you try to run the assembler without selecting both input files, an error dialog will appear prompting you to provide the required files.
- Invalid Opcodes: If the assembly code contains opcodes not found in the optab.txt file, an error message will notify you of the missing opcode.
- Unsupported Directives: The program supports common directives such as WORD, BYTE, RESW, and RESB. Ensure correct usage and formatting in the assembly file.
- Syntax Errors: The assembler expects correctly formatted assembly files. Errors like missing operands or labels will trigger error messages.

Example Walkthrough

1. Input Assembly (input.asm):

```
COPY   START   1000
        LDA    ALPHA
        ADD    ONE
        SUB    TWO
        STA    BETA
ALPHA  BYTE    C'CSE'
ONE    RESB    2
TWO    WORD    2
BETA   RESW    2
        END    1000
```

2. Opcode Table (optab.txt):

```
SUB 05
CMP 03
LDA 00
STA 23
ADD 01
JNC 08
```

Output:

1.Intermediate Code:

```
001000  COPY    START   1000
00100C          LDA     ALPHA
00100F          ADD     ONE
001011          SUB     TWO
001014          STA     BETA
001017  ALPHA   BYTE    C'CSE'
001020  ONE     RESB    2
001022  TWO     WORD    2
001025  BETA    RESW    2
```

2. Symbol Table:

```
ALPHA   001017
ONE     001020
TWO     001022
BETA    001025
```

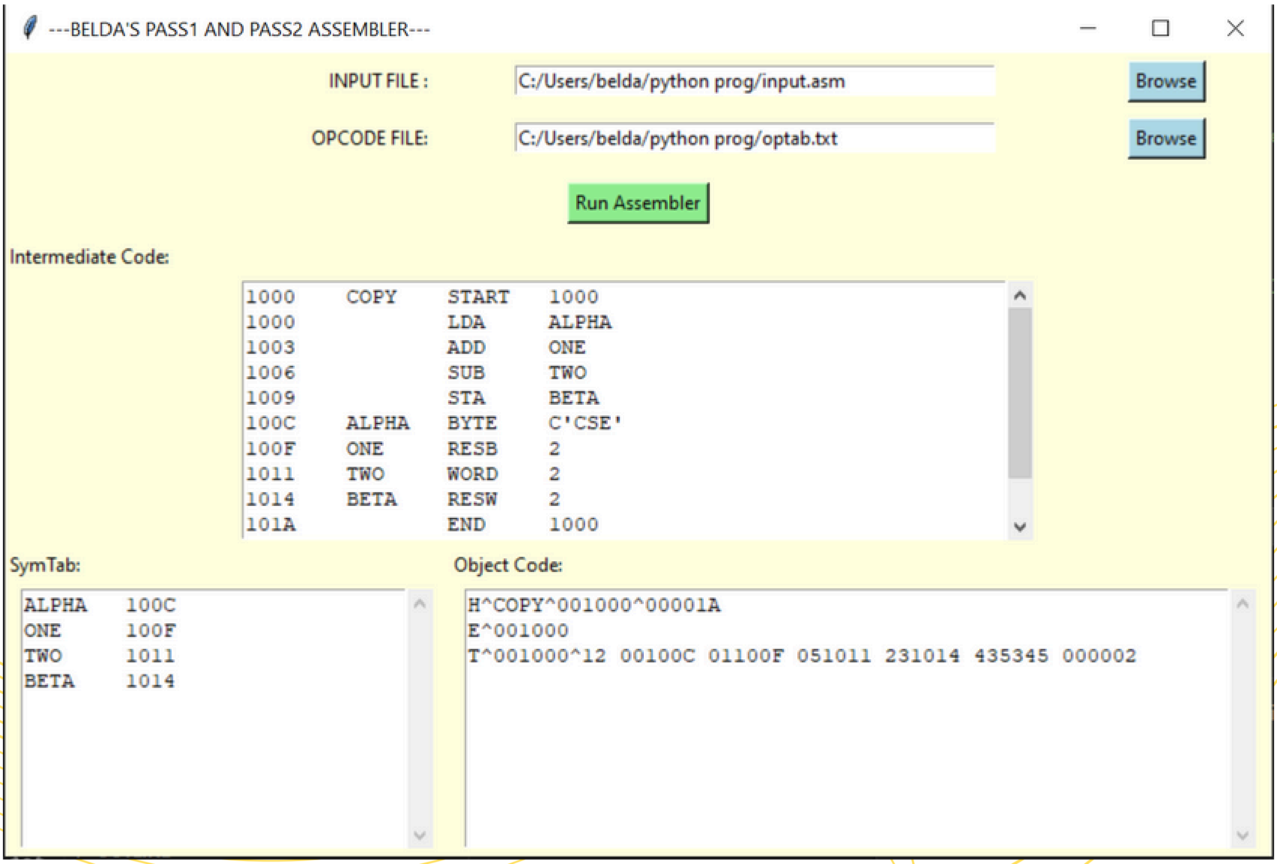
3. Object Code

```
H^COPY^001000^00001A
T^001000^12 00100C 01100F 051011 231014 435345 000002
E^001000
```

Conclusion

This assembler simplifies the process of converting assembly code to object code by automating both Pass 1 and Pass 2. The GUI offers an easy way to interact with the program, and the generated output adheres to the standard SIC object code format.
If errors occur, refer to the Error Handling section for troubleshooting.

Here is a representation of the final window with the output



INPUT FILE : C:/Users/belda/python prog/input.asm

OPCODE FILE: C:/Users/belda/python prog/optab.txt

Intermediate Code:

```
1000    COPY    START   1000
1000            LDA     ALPHA
1003            ADD     ONE
1006            SUB     TWO
1009            STA     BETA
100C    ALPHA   BYTE    C'CSE'
100F    ONE     RESB    2
1011    TWO     WORD    2
1014    BETA    RESW    2
101A            END     1000
```

SymTab:

```
ALPHA    100C
ONE      100F
TWO      1011
BETA     1014
```

Object Code:

```
H^COPY^001000^00001A
E^001000
T^001000^12 00100C 01100F 051011 231014 435345 000002
```

THANK YOU

BELDA BEN THOMAS
ROLL NO: 64