

System Software

Assembler for SIC/XE

Introduction

This project implements a two-pass SIC/XE assembler. It supports the following features:

- **Literals:** Data values directly included in the source code.
- **Program Blocks:** Allow dividing the program into logical blocks.
- **Expressions:** Enable calculations within operand fields.
- **Assembler Directives:** Provide instructions for the assembler itself.
- **Symbols:** Represent memory locations with meaningful names.

For optimal code quality, the assembler enforces these guidelines:

- **No whitespace in EXTDEF/EXTREF operands:** Operands in these directives (e.g., external references) must not have spaces between commas.

Note: The assembler focuses on the first six characters of the filename during assembly. This behaviour avoids complete program failure but prioritises shorter filenames.

Steps to run

Linux

- Clone the repository from [Github](#) or extract the contents of the Zip File.
 - `cd ./Assembler`
 - `make`
 - `cp test.txt build/`
 - Specify the name of the file (the file must be in the build directory).
 - `cd build`
 - `./assembler`
-

NOTE: 5.15.146.1-microsoft-standard-WSL2 was used along with g++ (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0

Files Generated

- Object Program
- Listing File
- Error File
- Intermediate File
- A file showing various tables created by the Assembler

Testing and Verification

A text file named test.txt is provided. This is question 3 of section 2.2 in the prescribed textbook. As mentioned, the assembler will be tested on this. Please note the input formatting required. This file demonstrates assembler usage.

Design of Assembler

My two-pass assembler functions as follows:

Pass 1:

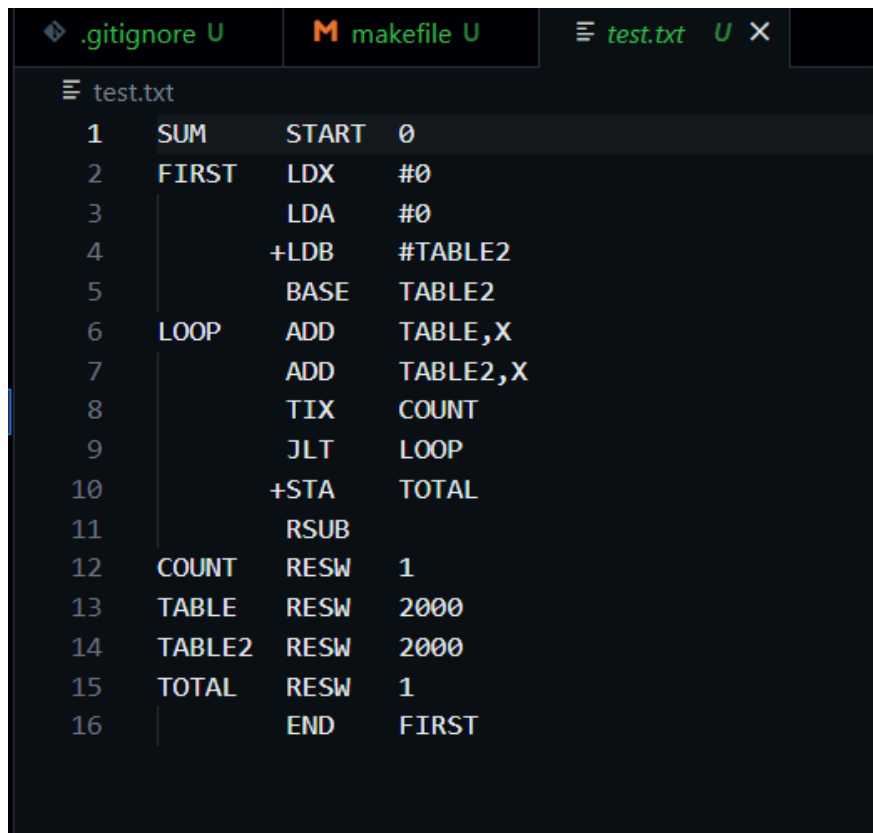
- **Intermediate File Generation:** The assembler creates and updates an intermediate file containing information for pass 2.
- **Error Handling:** Errors like duplicate symbols are identified and reported to an error file.
- **Symbol Table Construction:** Symbols encountered in the source code are declared and stored in a symbol table for future reference.
- **Comment Handling:** Comments are ignored during processing.
- **Start Directive Recognition:** Processing begins upon encountering the START directive. The Location Counter (LOCCTR) is set based on the value provided in the START directive or defaults to zero if absent.
- **Program Loop:** Nested loops iterate through the source code until the END directive is found.

-
- **Robust Execution:** The assembler is designed to continue execution despite non-critical errors, reporting them to the error file.

Pass 2:

- **Intermediate File Processing:** The second pass reads the intermediate file generated in pass 1 using the `readIntermediateFile()` function.
- **Listing and Object File Generation:** The assembler creates both a listing file (containing the source code with annotations) and an object file (containing machine code instructions).
- **Symbol Resolution:** The symbol table is used to resolve references to symbols used as operands, ensuring correct memory addresses are used.
- **Assembler Directive Processing:** Assembler directives are interpreted and handled appropriately during object file creation.
- **Program Block Support:** The assembler recognises and handles program blocks effectively.
- **Error Reporting:** Errors encountered during pass 2 are reported to the error file as needed.

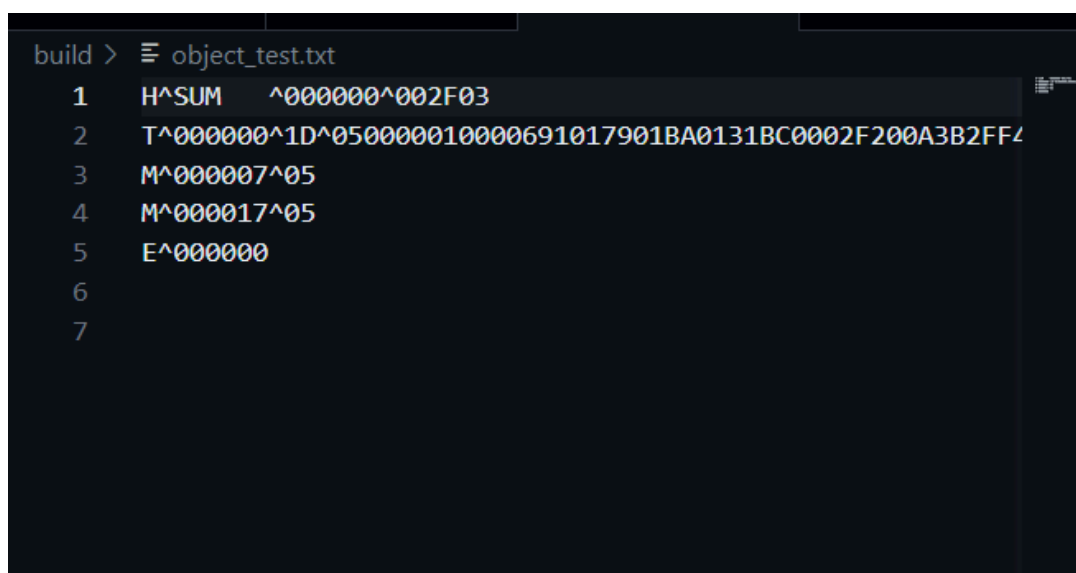
Screenshots



A screenshot of a code editor with a dark theme. The top bar shows three tabs: `.gitignore` (green), `makefile` (orange), and `test.txt` (green). The `test.txt` tab is active. The editor displays assembly code with line numbers 1 through 16. The code includes labels like `SUM`, `FIRST`, `LOOP`, `COUNT`, `TABLE`, `TABLE2`, and `TOTAL`, along with instructions like `LDX`, `LDA`, `+LDB`, `BASE`, `ADD`, `TIX`, `JLT`, `+STA`, `RSUB`, `RESW`, and `END`.

```
1  SUM      START  0
2  FIRST    LDX    #0
3           LDA    #0
4           +LDB   #TABLE2
5           BASE   TABLE2
6  LOOP     ADD    TABLE,X
7           ADD    TABLE2,X
8           TIX    COUNT
9           JLT    LOOP
10          +STA   TOTAL
11          RSUB
12  COUNT   RESW   1
13  TABLE  RESW   2000
14  TABLE2 RESW   2000
15  TOTAL   RESW   1
16          END    FIRST
```

Sample Input



A screenshot of a terminal window with a dark theme. The prompt is `build >`. The file `object_test.txt` is open, showing a hex dump of its contents. The dump consists of seven lines of hexadecimal data.

```
build > object_test.txt
1  H^SUM   ^000000^002F03
2  T^000000^1D^050000010000691017901BA0131BC0002F200A3B2FF4
3  M^000007^05
4  M^000017^05
5  E^000000
6
7
```

```
.gitignore U  M makefile U  tables_test.txt X
build > tables_test.txt
1  *****SYMBOL TABLE*****
2
3  :- name:undefined |address:0 |relative:00000
4  0:- name: |address:0 |relative:00000
5  COUNT:- name:COUNT |address:0001D |relative:00001
6  FIRST:- name:FIRST |address:00000 |relative:00001
7  LOOP:- name:LOOP |address:0000A |relative:00001
8  TABLE:- name:TABLE |address:00020 |relative:00001
9  TABLE2:- name:TABLE2 |address:01790 |relative:00001
10 TOTAL:- name:TOTAL |address:02F00 |relative:00001
11
12 *****LITERAL TABLE*****
13
14
15 *****EXTREF TABLE*****
16
17
18 *****EXTDEF TABLE*****
19
20
21
```