CSN-252 Project

An assembler for sic-xe

ANVADYA KHARE (21114015)

2023

**INTRODUCTION**

The Objective of the project is to implement a version of two-pass SIC/XE assembler.

Features implemented:

* Literals
* Control Sections
* Expressions
* Assembler Directives
* Symbols

Some conditions required for successfully running the code (to enforce best practices):

* A comment must follow after a **CSECT** command (by specifying a . after the CSECT instruction).
* Operands separated by commas (such as the ones found in EXTDEF and EXTREF) must not contain whitespace between them.

**NB:** Instead of failing to assemble the object program the assembler will only consider the first six characters of the file name

How to run the code (on Windows):

1. Extract the contents of the zip file
2. cd ./21114015
3. g++ pass2.cpp -o ./assembler
4. ./assembler.exe
5. Specify the name of the file (the file must be in the current working directory)

**NB:** Windows 10 Home (19044.2728) was used alongwith g++ 6.3.0

How to run the code (on Linux):

1. Extract the contents of the zip file
2. cd ./21114015
3. g++ pass2.cpp -o ./assembler.out
4. ./assembler.out
5. Specify the name of the file (the file must be in the current working directory)

**NB:** 5.10.16.3-microsoft-standard-WSL2 kernel was used alongwith g++ 11

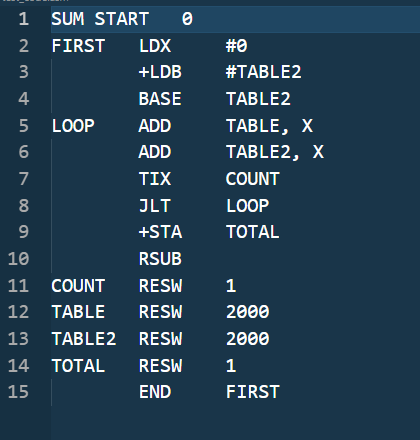
Output Files generated:

1. Object Program
2. Listing File
3. Error File
4. Intermediate File
5. A file showing various tables created by the Assembler

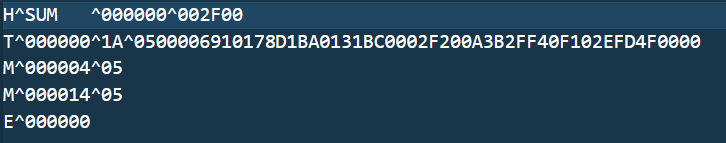
Files provided to demonstrate assembler usage:

1. test\_code.txt: 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.
2. csect\_test.txt: This provides a sample code given in the textbook (Fig 2.15) to demonstrate the usage of control sections.

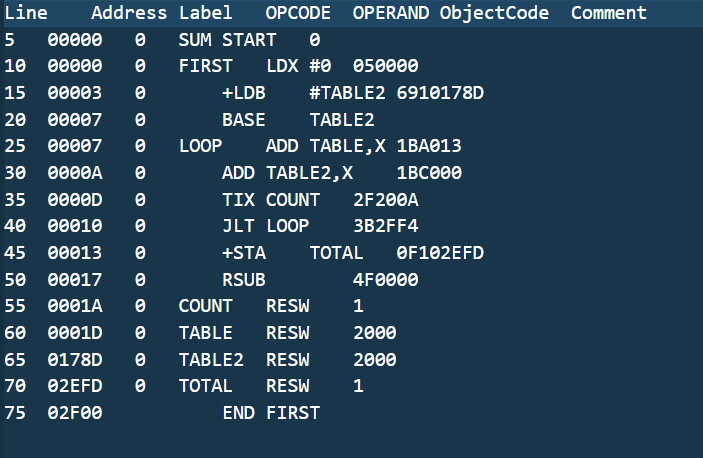
Results on executing test\_code.txt:



^Sample Test Input

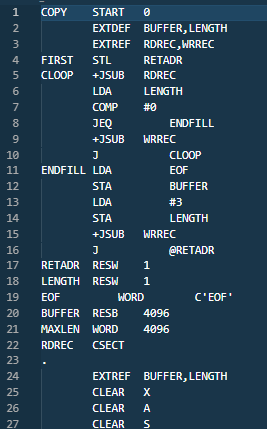
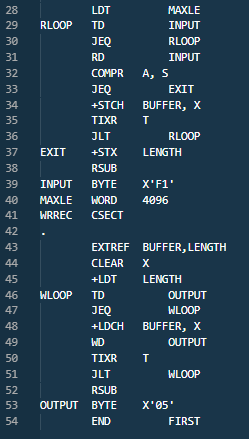


^Object File of the sample input

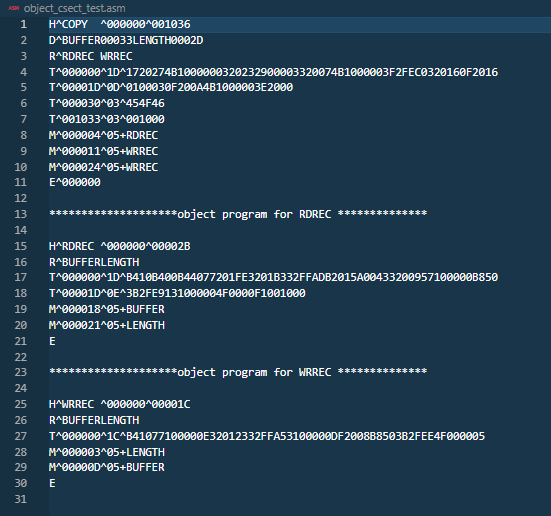


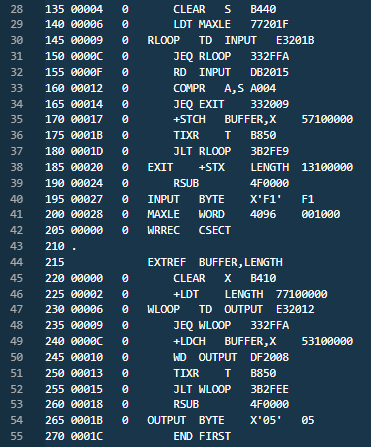
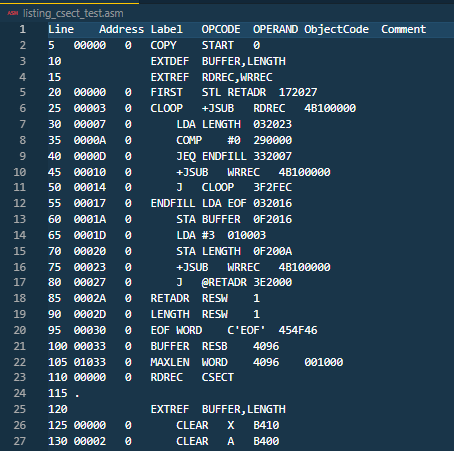
^Listing file for the sample input

Results on executing csect\_test.txt:

^csect\_test.txt file



^Object File for csect\_test.txt

^Listing File for csect\_test.txt

**BRIEF EXPLANATION OF DESIGN OF ASSEMBLER**

I have a designed a standard two pass assembler. The function of each pass is as follows:

* Pass 1: The intermediate file is created and updated and the error file is also updated if the need arises. The required symbols are declared in the symbol table. Comments are dealt with accordingly and are ignored, the actual processing of the input starts when the START symbol is encountered. The LOCCTR is set as per the value given in the start directive, otherwise it is default initialised to zero. Two nested loops are at the heart of the program and keep executing till the directive END is encountered. Errors such as duplicate symbols are mentioned in the error file. The assembler is very robust and does not stop execution unless a critical error is encountered.
* Pass 2: The second pass acts on the intermediate file generated by pass 1, the file is processed via the readIntermediateFile() function and we then proceed to generate the listing file and the object program. The error file is updated as and when necessary. The symbol table is used to resolve any issues created when symbols are used as operands. The various assembler directives are also taken into account while creating the object file. If control sections are present, we will use the writeRRecord() and writeDRecord() to write the external references and the external definitions.