CSN 252

SIC-XE Assembler project report

Kulkarni Sourabh Shrinivasrao (21114053)

- SIC/XE stands for Simplified Instructional Computer Extra Equipment or Extra Expensive . This computer is an advance version of SIC . Both SIC and SIC/XE are closely related to each other and are upward compatible .

System configurations:

- Memory is comprised of 1 megabyte (2^20 bytes) of 8-bit bytes, which is relatively small in the standard SIC memory size. Due to this change in memory size, both the instruction formats and addressing modes are affected. In the SIC/XE architecture, a word is formed by 3 consecutive bytes (24 bits), with all addresses being byte addresses and words being addressed by the lowest numbered byte location.

- The registers in SIC/XE architecture contain a total of 9 registers, including 5 SIC registers and 4 additional registers: B (base register), S and T (general working registers), and F (floating point accumulator).

- Different data formats are used for different types of data. Integers are represented using binary numbers, characters using ASCII codes, and floating points using 48 bits.

- In the SIC/XE architecture, there are four available instruction formats. Format 3 and format 4 are distinguished by a bit (e). A value of 0 for e indicates format 3, while a value of 1 indicates format 4.

Assembler:

- An assembler is a software tool that converts assembly language code into machine language code. Assembly language is a low-level programming language that uses symbolic instructions instead of binary instructions to represent machine operations
- The assembler works by reading the assembly language code line by line and converting each line into its equivalent machine language code. The resulting machine code is then saved into an object file, which can be loaded and executed by the computer.
- My assembler includes all Machine-Independent Assembler Features-

1.Literals

2.Symbol Defining Statements

3.Expressions

4.Control Sections and Program Linking

Implementation:

- The Assembler is implemented our assembler using C++ programming language
- The Assembler uses C++ library fstream to read input from a file and write output om another file.

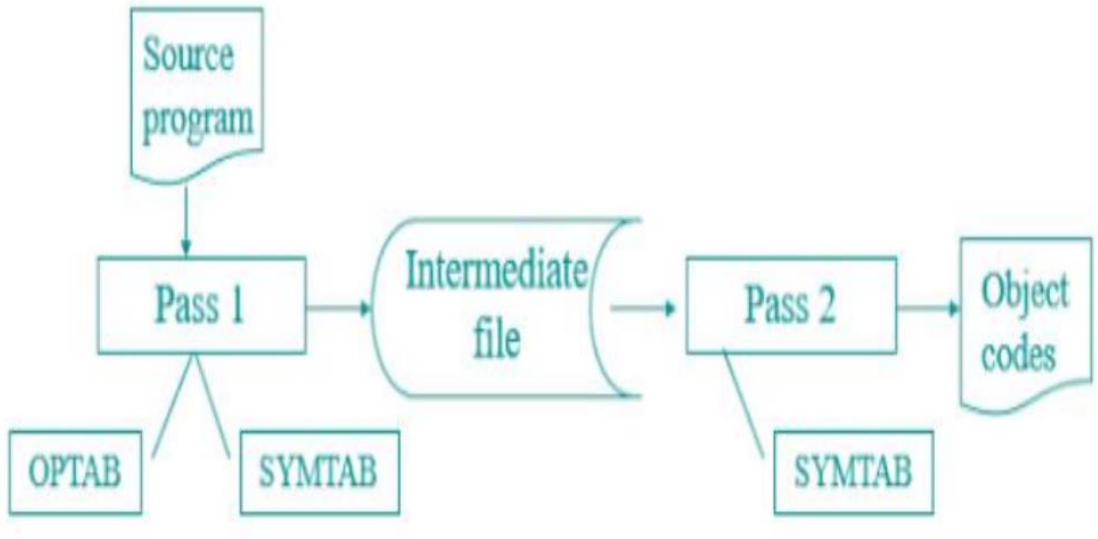
- File structure:

  There exist following functions distributed within files with given purpose:

  1. Pass1():

the input source code is first processed in Pass 1 to create an intermediate file, SYMTAB, LITTAB, and CSECT_TAB. In Pass 1, the input code is scanned and processed line by line, the LOCCTR is incremented based on the opcode, and symbol addresses are stored in SYMTAB. If an error is found, it is written to the error file. The handle_LTORG() function is used to print the literals in the LITTAB. The evaluateExpression() function is used to validate

expressions in EQU statements. In Pass 2, the intermediate file is used to generate the object code.



2.Tables():

It contains all the data structures required for our assembler to run. It contains the structs for labels, opcode, literal, blocks, extdef, extref, and control sections. The CSECT_Tab contains Maps are defined for various tables with their indices as strings with the names of the labels or opcodes as required.

3.pass2():

The intermediate file is read using the readIntermediateFile() function, and if there are errors in opening the intermediate file, the error message is printed in the error file. The first line of the intermediate file is then read, and the program length and start address are initialized. The header record is written in the object program, and the input lines from the intermediate file are taken until the opcode comes as 'END' or 'CSECT'. Object codes are generated based on different types of opcodes, and external references and definitions are written using the writeRRecord() and writeDRecord() functions. The end record for the program is written using the writeEndRecord() function. The functions readTillTab(), createObjectCodeFormat34(), writeDRecord(), writeRRecord(), and writeEndRecord() are used for various purposes. After the execution of

Pass 1, the Tables like SYTAB and LITTAB are printed in a separate file before executing Pass 2.

4.functions():

It has different useful functions such as

getString()- takes in input as a character and returns a string.

intToStringHex()- takes in input as int and then converts it into its hexadecimal equivalent with string data type.

stringToHexString()- takes in string as input and then converts the string into its hexadecimal equivalent and then returns the equivalent as string.

checkWhiteSpace()- checks if blanks are present. If present, returns true or else false.

And some more which are used frequently in other files.

How to execute the code:

Firstly, put all files in the same folder. Preferably keep input in .txt file.

Then run 'pass2.cpp' with following commands (execute sequentially):

```
PS C:\Users\Dell\Desktop\252final> g++ pass2.cpp -o pass2
PS C:\Users\Dell\Desktop\252final> ./pass2
```

If build is successful, it will demand you the input file. After giving it,you should see following log:

```
 OPTAB loaded...

  Pass1 done...
Writing intermediate File to 'intermediate_i1.txt'
Writing errors (if present) to 'error_i1.txt'
 SYMBOL TABLE loaded...
 LITERAL TABLE loaded
 EXTREF TABLE loaded...
 EXTDEF TABLE loaded...

Performing PASS2
Writing object code to 'object_i1.txt'
Writing lists to 'listing_i1.txt'
```

Output (object code for given program) will be written in object_<*inputFileName>*

Intermediate files are written in intermediate_<*inputFileName>*

Errors are written in error_<inputFileName>

Sample output looks like this:

object_i1.txt

```
H^COPY  ^000000^001036
D^BUFFER00033LENGTH0002D
R^RDREC WRREC
T^000000^1D^1720274B10000003202329000033200 74B1000003F2FEC0320160F2016
T^00001D^0D^0100030F200A4B1000003E2000
T^000030^03^454F46
T^001033^03^001000
M^000004^05+RDREC
M^000011^05+WRREC
M^000024^05+WRREC
E^000000

_____object code for RDREC_____


H^RDREC ^000000^00002B
R^BUFFERLENGTH
T^000000^1D^B410B400B44077201FE3201B332FFADB2015A00433200957100000B850
T^00001D^0E^3B2FE9131000004F0000F1001000
M^000018^05+BUFFER
M^000021^05+LENGTH
E

_____object code for WRREC_____


H^WRREC ^000000^00001C
R^BUFFERLENGTH
T^000000^1C^B41077100000E32012332FFA53100000DF2008B8503B2FEE4F000005
M^000003^05+LENGTH
M^00000D^05+BUFFER
E

```