

CSN-252

SYSTEM SOFTWARE

Tutorial – 8

Design of a SIC-XE Assembler

(Includes Program blocks)

Submitted By - Sanidhya Bhatia

Enrollment No. – 21114090

Introduction:

The designed SIC/XE assembler is a two-pass assembler written in the C++ programming language. The assembler accepts an assembly language input file (source programme using the SIC/XE instruction set) and creates the object code file ("object.txt"), as well as tables ("tables.txt"), intermediate file ("intermediate.txt"), and error files ("error_pass_one.txt" and "error_pass_two.txt"). The assembler does two passes:

- Pass 1 - The assembler generates a symbol table and an intermediate file for usage in Pass 2.
- Pass 2 - The assembler generates a listing file ("listing.txt") that includes the input assembly code as well as the address, block number, and object code for each instruction. The object programme and error file indicating the faults (if any) in the input assembly programme are also included.

This assembler considers all SIC/XE instructions and machine-independent assembler features like literals, program blocks, and symbol-define statements.

Steps to compile and generated output:

1. Open the folder containing the assembler code, the folder should contain four files initially (the input file, "PASS1SICXE.cpp", "PASS2SICXE.cpp" and "required.cpp").

```
PS C:\Users\Sanidhya\Desktop\SIC-XE Assembler> ls

Directory: C:\Users\Sanidhya\Desktop\SIC-XE Assembler

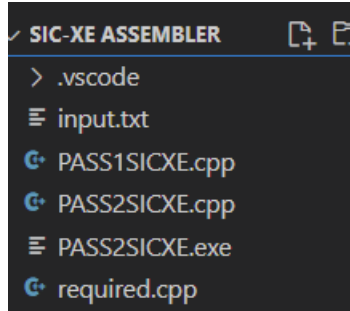
Mode                LastWriteTime         Length Name
----                -
d-----          06-04-2023    16:48             .vscode
-a-----          07-04-2023    13:55          1107 input.txt
-a-----          07-04-2023    20:04       16148 PASS1SICXE.cpp
-a-----          07-04-2023    20:06       29535 PASS2SICXE.cpp
-a-----          07-04-2023    20:02       16481 required.cpp

PS C:\Users\Sanidhya\Desktop\SIC-XE Assembler> |
```

2. Compile the "PASS2SICXE.cpp" file using the following command.

```
PS C:\Users\Sanidhya\Desktop\SIC-XE Assembler> g++ PASS2SICXE.cpp -o PASS2SICXE.exe
PS C:\Users\Sanidhya\Desktop\SIC-XE Assembler> █
```

3. We can see that an executable file "PASS2SICXE.exe" is generated.



4. Now we can execute this file using the command "./PASS2SICXE.exe". And we can see the following changes in the terminal.

```
PS C:\Users\Sanidhya\Desktop\SIC-XE Assembler> g++ PASS2SICXE.cpp -o PASS2SICXE.exe
PS C:\Users\Sanidhya\Desktop\SIC-XE Assembler> ./PASS2SICXE.exe
** NOTE: The Input file and executable assembler file should be in same folder **

Enter name of the Input File=█
```

5. Now enter the name of the input text file and execute.

```
PS C:\Users\Sanidhya\Desktop\SIC-XE Assembler> ./PASS2SICXE.exe
** NOTE: The Input file and executable assembler file should be in same folder **

Enter name of the Input File=input.txt

Loading OPTAB

Performing Pass 1
Writing the Intermediate File to 'intermediate.txt'
Writing the Error File to 'pass_one_errors.txt'
4096-0
Creating Symbol Table in 'tables.txt'
Creating Literal Table in 'tables.txt'
Creating Block Table in 'tables.txt'

Performing Pass 2
Writing the Object File to 'object.txt'
Writing the Listing File to 'listing.txt'
PS C:\Users\Sanidhya\Desktop\SIC-XE Assembler> █
```

6. The following output files are generated.

We can now see that many new text files are generated, these are.

- "object.txt": contains the object code of the program (header record, text record, modification record, and end record).
- "intermediate.txt": this file is generated as an output to pass 1, it contains. line number, address, label, operands, comment, and the opcode of all the instructions.

- “pass_one_errors.txt”: contains all the errors found during pass1 of the assembler.
- “pass_two_errors.txt”: contains all the errors found during pass2 of the assembler.
- “tables.txt”: contains the Symbol Table, Literal Table, and the Block Table generated during the assembly of the program. Two more tables for registers and operations are already loaded during the start of the program.
- “listing.txt”: this file is generated as an output to pass 2 along with the object code file, it contains the line number, address, opcode, operand, block number, object code, and comments.

```
PS C:\Users\Sanidhya\Desktop\SIC-XE Assembler> ls

Directory: C:\Users\Sanidhya\Desktop\SIC-XE Assembler

Mode                LastWriteTime         Length Name
----                -
d-----          06-04-2023         16:48         .vscode
-a----          07-04-2023         13:55         1107 input.txt
-a----          07-04-2023         21:07         1422 intermediate.txt
-a----          07-04-2023         21:07         1707 listing.txt
-a----          07-04-2023         21:07         348 object.txt
-a----          07-04-2023         20:04        16148 PASS1SICXE.cpp
-a----          07-04-2023         20:06        29535 PASS2SICXE.cpp
-a----          07-04-2023         21:03       521176 PASS2SICXE.exe
-a----          07-04-2023         21:07          37 pass_one_errors.txt
-a----          07-04-2023         21:07          37 pass_two_errors.txt
-a----          07-04-2023         20:02       16481 required.cpp
-a----          07-04-2023         21:07        1274 tables.txt

PS C:\Users\Sanidhya\Desktop\SIC-XE Assembler> S
```

Assembler Architecture and working of the code:

All the functions and table generation code are present in the “required.cpp” file, all the code related to the pass 1 of the assembler is present in the “PASS1SICXE.cpp” file and all the code related to the pass 2 of the assembler is present in the “PASS2SICXE.cpp” file.

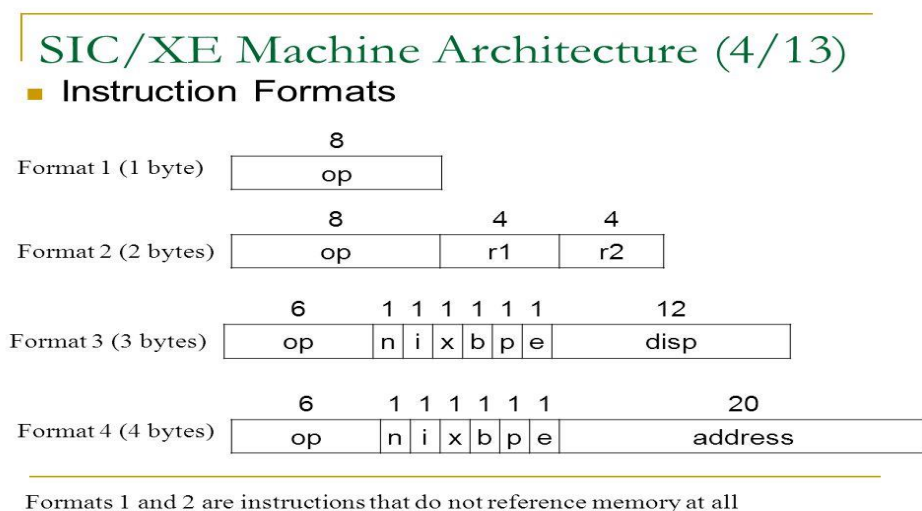
In pass 1, the input source file is used to make the intermediate and error files. In case the intermediate file does not open, or the source file cannot be in the folder, the corresponding error messages get displayed.

The basic approach used is that the assembler starts reading from the first line of the assembly code and checks whether it is a comment line. If it is a comment line, it is written to the “intermediate.txt” file and the line number gets updated. This process is repeated until a non-comment line is identified, after which the assembler begins checking for the opcode.

If the opcode is "START" the assembler updates the line number, LOCCTR, and START Address, else the START Address and LOCCTR are initialized to 0. A single while loop is a run that ends when the END opcode is found. Same thing happens as stated above if another comment line is encountered, else for the given instruction assembler starts to check whether the label in the instruction matches with a label in the SYMTAB, if it is found in the SYMTAB, an error is printed in the "pass_one_errors.txt" file that a "Duplicate Symbol" is found. Otherwise, the symbol is assigned a name, an address, block number, etc. and then it is stored in the SYMTAB.

Then for the same instruction, the assembler checks the opcode, if the opcode is present in the OPTAB, if it is present, then its format is found and the LOCCTR is incremented accordingly to the format used/

There are four types of formats in SIC-XE, these are:



If the opcode is not found in the OPTAB then it is checked in other opcodes like 'RESW', 'BYTE', 'RESBYTE', 'WORD', 'LTORG', 'ORG', 'BASE', 'USE', 'EQU'. Then, accordingly, the assembler inserts symbols in the map defined in the tables file. For instance whenever it encounters 'USE' other than the default case then it inserts a new BLOCK entry in the BLOCK map, for 'ORG' it points out the LOCCTR to the operand value given, for 'LTORG' it calls LTORGsupport() function which is defined in pass1.cpp, for EQU it checks if the operand is an expression and then checks if it is a valid expression, if valid it enters the symbol in the SYMTAB.

After checking all this if the opcode does not match then the assembler prints an error message in the error file. After the while loop ends, the length of the program is stored for generating the record, and Literal Table (LITTAB) and the Symbol Table (SYMTAB) are printed in the "tables.txt" file.

Pass 1 also generates an intermediate file storing all this information, the line number, address, etc., and then this file is passed on to the Pass 2 assembler for the next step.

The evaluateExpression() function of the pass 1:

The evaluateExpression() function utilizes the pass-by reference feature. Here the while loop is used to fetch out the symbols from the expression and if the symbols which are fetched

are not present in SYMTAB then the error message is sent to the error file. To check on the nature of the expression i.e., based on whether the expression is absolute or relative, a variable named pairCount is used, which keeps the track of whether the expression is absolute or relative. If the pair count gives some unexpected value, an error message is printed.

The intermediate file which was generated as the output of pass 1 is now utilized by pass 2 of the assembler to generate a listing file and the object program as the output. If any of the files do not open, an error message is printed.

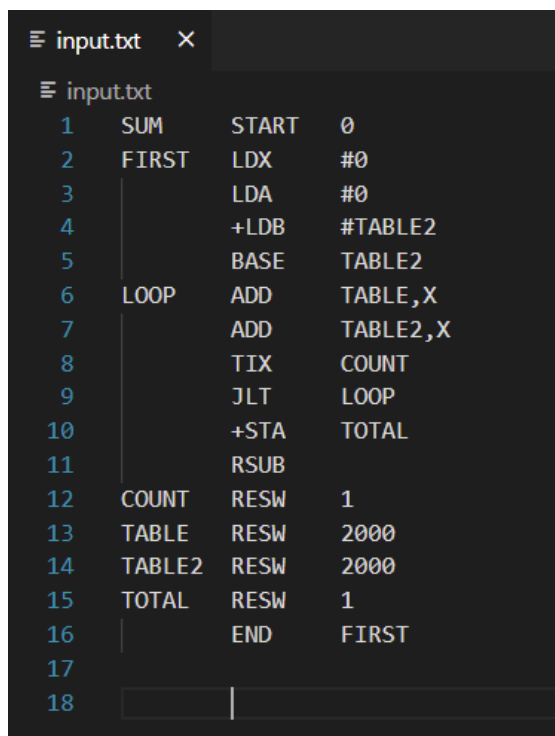
After all the files have opened successfully, the assembler starts reading from the first line of the intermediate file, it first checks for any comments if present, and if found they are written to the listing file else it starts checking for opcodes.

If it gets START, it initializes the start address in LOCCTR and writes the line into the listing file. First, it writes the header record. Next, it reads from the intermediate file until it encounters the opcode: END. Using the textrecord() function it writes on the object program and updates the listing file. The object code is written based on the types of formats used in instruction. For formats 3 and 4 the assembler uses createObjectCodeFormat34() function. After completing all the text records, it writes the end record.

The functions, and data structures (tables) can be found in "required.cpp" file along with the corresponding explanation.

Testing the program:

The following test program is used as the input in "input.txt" file.



```
input.txt
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
17
18
```

The received "intermediate.txt" file after pass 1:

	input.txt	intermediate.txt	
		intermediate.txt	
1	Line	Address	Label
2	5	00000	0
3	10	00000	0
4	15	00003	0
5	20	00006	0
6	25	0000A	0
7	30	0000A	0
8	35	0000D	0
9	40	00010	0
10	45	00013	0
11	50	00016	0
12	55	0001A	0
13	60	0001D	0
14	65	00020	0
15	70	01790	0
16	75	02F00	0
17	80	02F03	
18			

The received "object.txt" file after pass 2:

	input.txt	object.txt	
		object.txt	
1	H^SUM	^000000^002F03	
2	T^000000^1D^050000010000691017901BA0131BC0002F200A3B2FF40F102F004F0000		
3	M^000007^05		
4	M^000017^05		
5	E^000000		
6			
7			

The received "listing.txt" file after pass 2:

≡ input.txt

≡ listing.txt X

≡ listing.txt

	Line	Address	Label	OPCODE	OPERAND	ObjectCode	Comment
2	5	00000	0	SUM	START	0	
3	10	00000	0	FIRST	LDX #0	050000	
4	15	00003	0	LDA #0	010000		
5	20	00006	0	+LDB	#TABLE2	69101790	
6	25	0000A	0	BASE	TABLE2		
7	30	0000A	0	LOOP	ADD TABLE,X	1BA013	
8	35	0000D	0	ADD	TABLE2,X	1BC000	
9	40	00010	0	TIX	COUNT	2F200A	
10	45	00013	0	JLT	LOOP	3B2FF4	
11	50	00016	0	+STA	TOTAL	0F102F00	
12	55	0001A	0	RSUB		4F0000	
13	60	0001D	0	COUNT	RESW	1	
14	65	00020	0	TABLE	RESW	2000	
15	70	01790	0	TABLE2	RESW	2000	
16	75	02F00	0	TOTAL	RESW	1	
17	80	02F03		END	FIRST		
18							

Since there are no errors in the input code, both the error files “pass_one_errors.txt” and “pass_two_errors.txt” are empty.

input.txt

tables.txt

tables.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

BLOCK TABLE

16

17

DEFAULT:- value:DEFAULT |address:00000

18

19

Example of error detection:

```
pass_two_errors.txt
1 #####PASS2#####
2 Line 35 : Symbol doesn't exists. Found TABLE3
3
```