**SIC-XE COMPILER**

**With program blocks**

-NAQIYAH KAGZI

-21114064

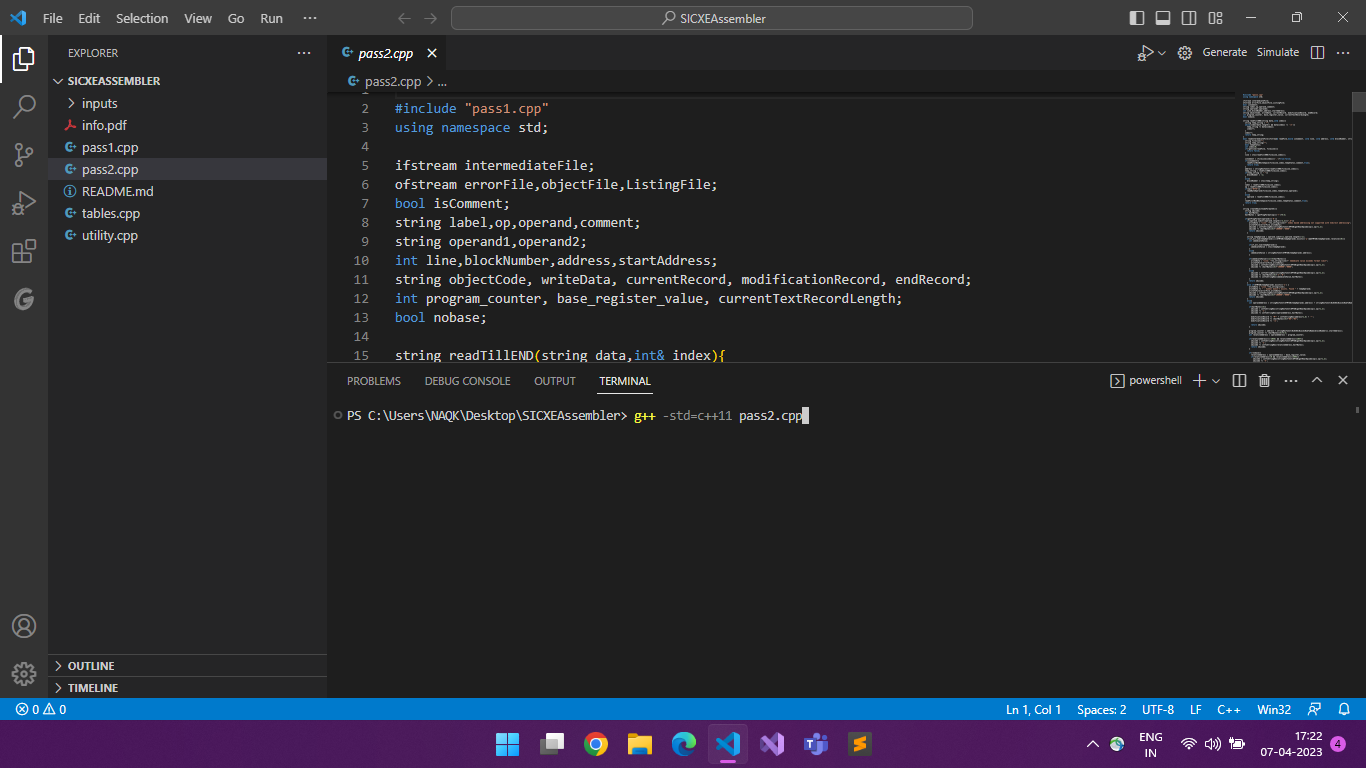
The Objective of the project is to implement a version of two-pass SIC/XE assembler: Pass 1 and Pass 2.

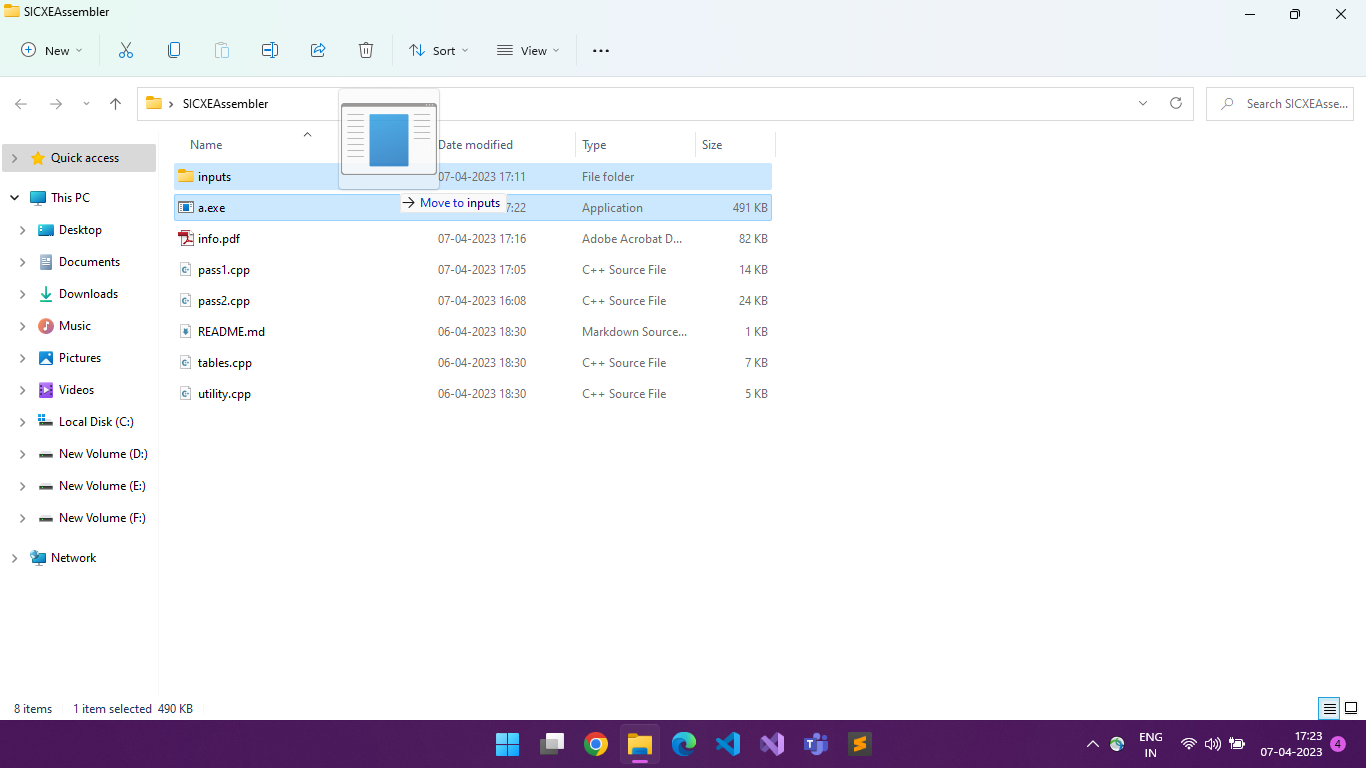
The Assembler we implemented includes all the SIC/XE instructions and supports all four formats 1, 2, 3, 4, addressing modes and program relocation.

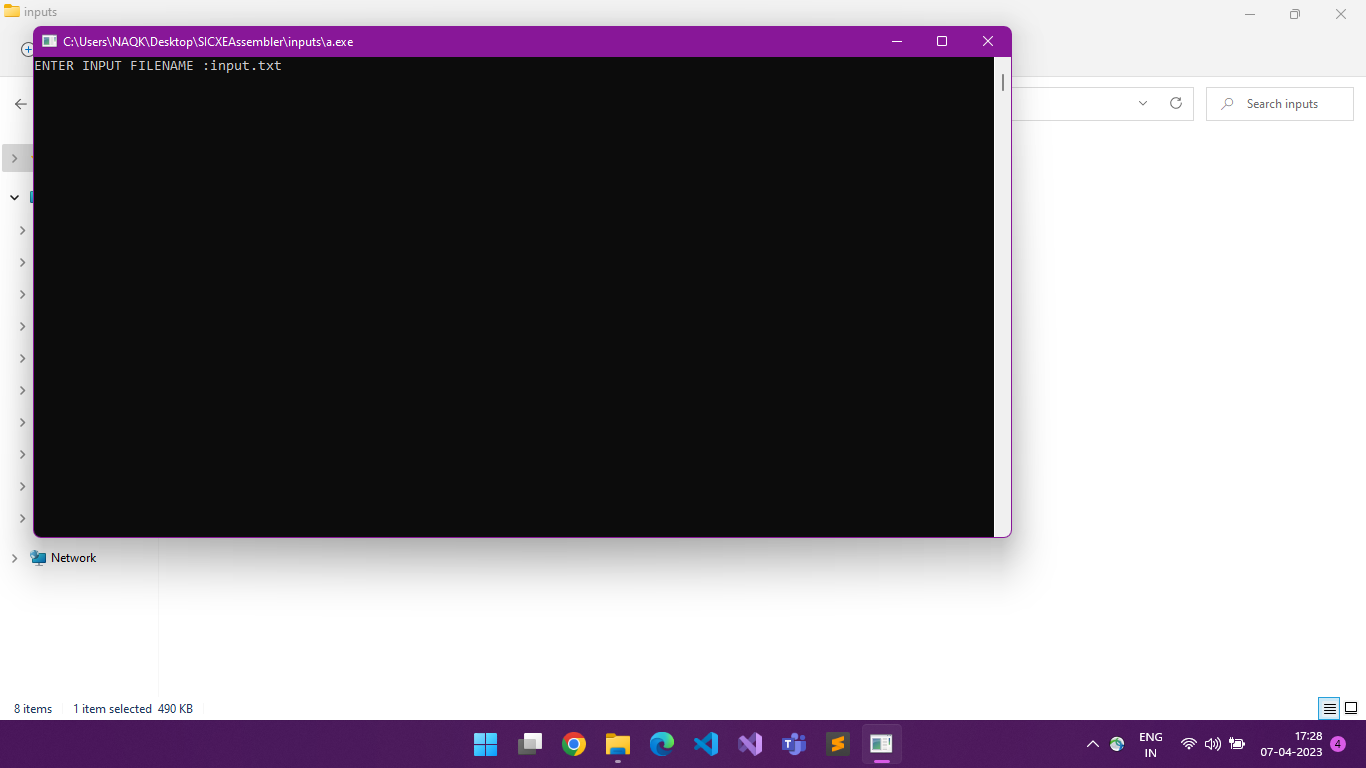
**STEPS TO RUN THE PROGRAM:-**

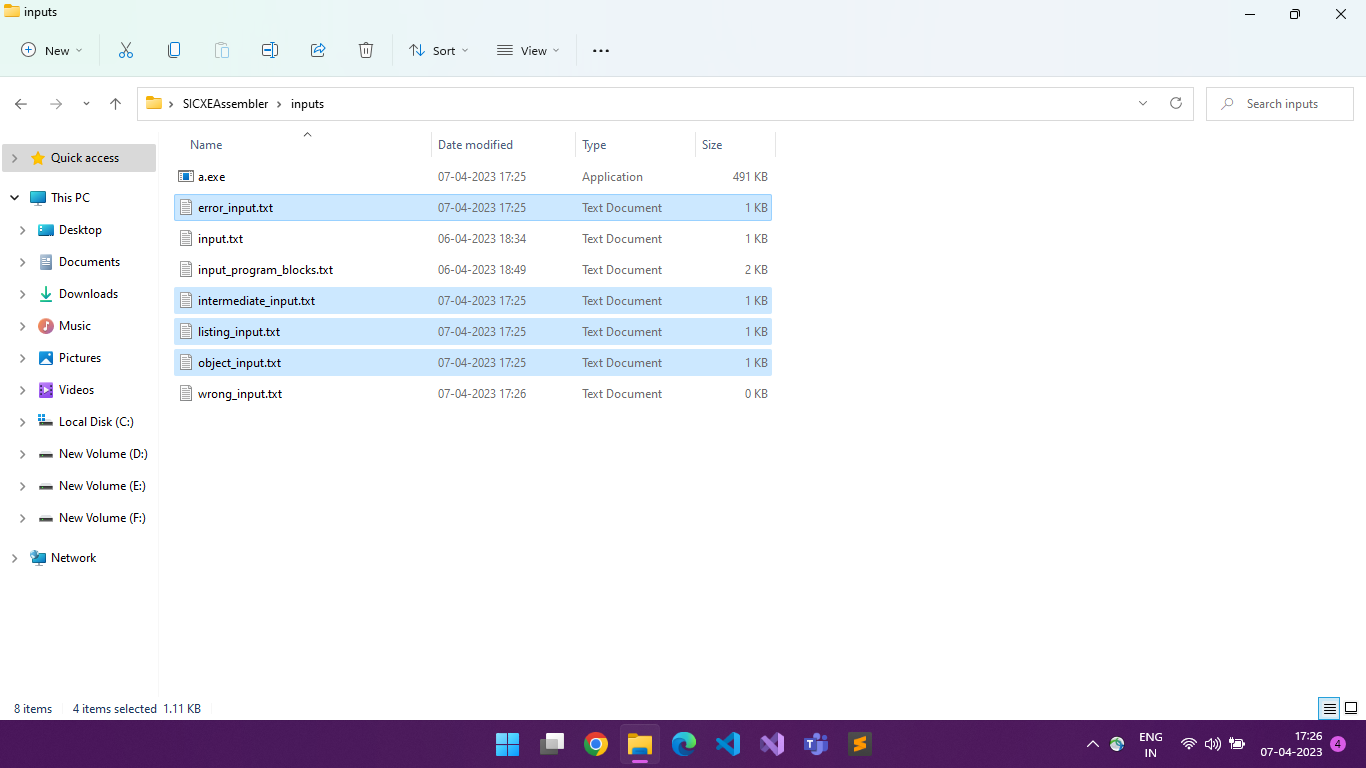
1. Download the zip file and open the terminal
2. Compile the file pass2.cpp using the command **g++ -std=c++11 pass2.cpp**
3. Put the executable a.out and the inputs in the same folder
4. Now open the a.exe executable file.
5. Write the name of the file you want to run in the given space
6. Four folders will be created corresponding to error file, intermediate file, listing file, and object code in the same folder.

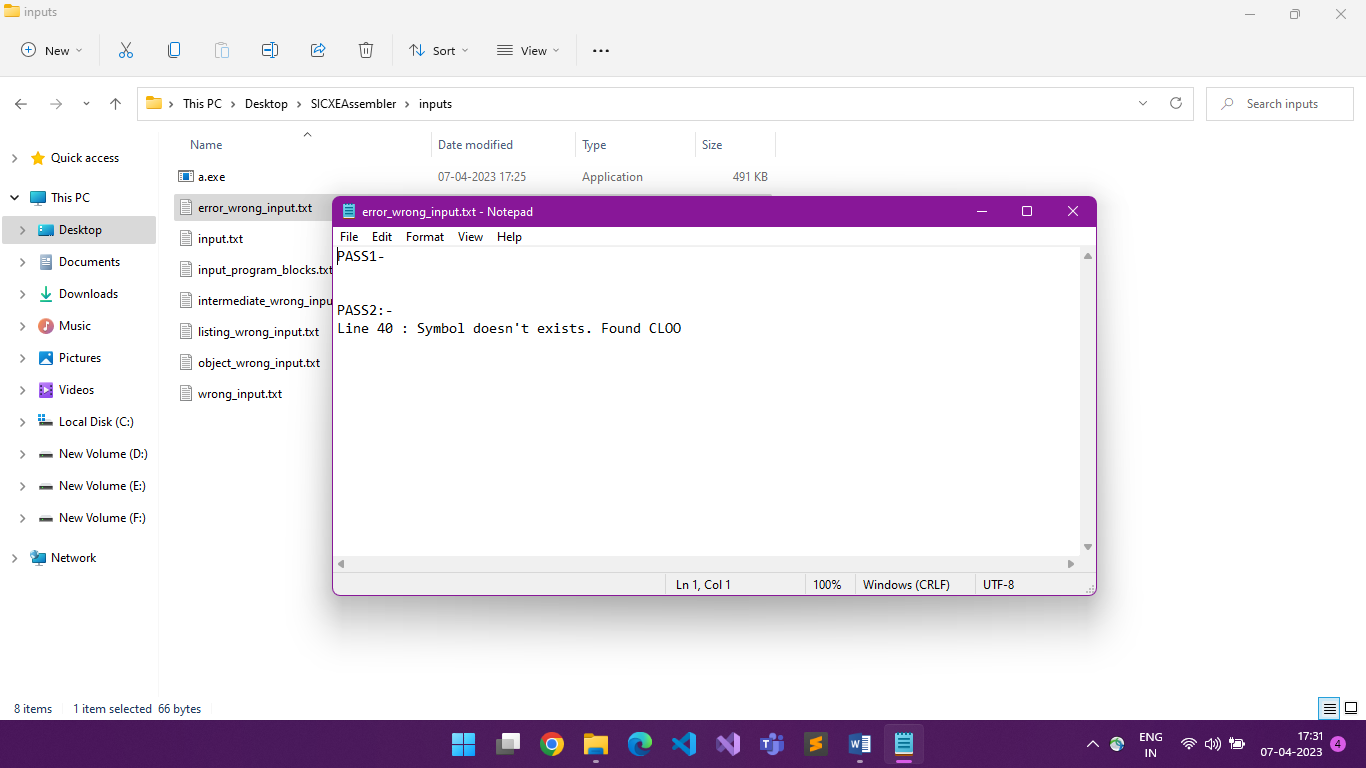
Some Screenshots for Reference:-

Running pass2.cpp (step 1 of instructions)

Moving a.exe to inputs

Open a.exe and enter filename

intermediate, listing, error and object file are produced

Error shown for wrong input file

Structure of the program:-

PASS1-

Using the source ﬁle, we update the intermediate and error ﬁles. If the source ﬁle cannot be found or the intermediate ﬁle cannot be opened, the relevant error is written to the error ﬁle, and if the error ﬁle cannot be opened, it is printed to the console. The variables that are required are declared. The ﬁrst line is then used as input, and it is checked to see if it is a comment line. We treat the lines as input until they become comments, at which point we print them to our intermediate ﬁle and update our line number. If the line isn't a comment, we check if the opcode is 'START,' and if it is, we update the line number, LOCCTR, and start address. If the opcode isn't found, we set the start address and LOCCTR to zero. Then we use two nested while() loops, with the outer loop iterating until the opcode equals 'END' and the inner loop iterating until the opcode equals 'END' or 'CSECT'. We check if the line is a comment inside the inner loop. If there is a comment, we print it to our intermediate ﬁle, update the line number, and read the next line of input. If there isn't a comment in the line, we look for a label and see if it's in the SYMTAB; if it is, we print an error saying 'Duplicate symbol' in the error ﬁle, or we give the symbol a name, address, and other required information and store it in the SYMTAB. Then we look to see if the opcode is present. If it is present, we determine its format and then increase the LOCCTR correspondingly. If the opcode 'WORD', 'RESW', 'BYTE', 'RESBYTE', 'LTORG', 'ORG, 'BASE', 'USE', 'EQU', 'EXTREF', or 'EXTDEF' is not present in OPTAB, we check it with other opcodes like 'WORD', 'RESW', 'BY As a result, we place the symbols, external references, and external deﬁnitions in the SYMTAB or map that we established for the control section. For example, we place a new BLOCK item in the BLOCK map as provided in the utility for opcodes like USE. We call the handle LTORG() function described in pass1 for LTORG in the cpp ﬁle. cpp, for 'ORG,' we use LOCCTR to refer to the operand value, and for EQU, we check if the operand is an expression, then we use the validator to see if the expression is valid.

TABLES –

It contains all of the data structures that our assembler needs to function. Labels, opcode, literal, blocks, extdef, extref, and control sections are all represented as structs. The CSECT Tab contains Maps for various tables that have their indices deﬁned as strings with the names of the labels or opcodes as needed.

UTILITY-

It contains useful functions that will be required by the other ﬁles.

PASS2-

We take in the intermediate ﬁle as input using the readIntermediateFile() function and generate the listing ﬁle and the object program. Similar to pass1, if the intermediate ﬁle is unable to open, we will print the error message in the error ﬁle. Same with the object ﬁle if unable to open. We then read the ﬁrst line of the intermediate ﬁle. Until the lines are comments, we take them as input and print them to our intermediate ﬁle and update our line

number. If we get an opcode as ‘START’, we initialize our start address as the LOCCTR, and write the line into the listing ﬁle. Then we check whether the number of sections in our intermediate ﬁle was greater than one, if so, then we update our program length as the length of the ﬁrst control section or else we keep the program length unchanged. We then write the ﬁrst header record in the object program. Then until the opcode comes as ‘END’ or ‘CSECT’ if the control sections are present, we take in the input lines from the intermediate ﬁle and then update the listing ﬁle and then write the object program in the text record using the textrecord() function.

We will write the object code on the basis of the types of formats used in the instruction. Based on diferent types of opcodes such as ‘BYTE’,’WORD’,’BASE’,’NOBASE’,’EXTDEF’,’EXTREF’,’CSECT’, we will

generate diferent types of object codes.

Data Structures used-

SYMTAB

The struct contains information of labels like name, address, block number, a character representing whether the label exits in the symbol table or not, an integer representing whether label is relative or not.

OPTAB

The struct contains information of the opcode like name, format, a character representing whether the opcode is valid or not.

LITTAB

The struct contains information of literals like its value, address, block number, a character representing whether the literal exits in the literal table or not.

REGTAB

The struct contains information of registers like its numeric equivalent, a character representing whether the registers exist or not.

BLOCKS

The struct contains information of blocks like its name, start address, block number, location counter value for end address of block, a character representing whether the block exits or not.

CSECT

The struct contains information of diferent control sections like its name, start address, section number, length, location counter value for end address of section. It also contains two maps for extref and extdef of particular section.