SIC/XE Assembler

(Phase 1)

Made with JAVA

George Amgad Wadie 5002

# Requirements specification

1. A parser that is capable of handling source lines that are instructions, storage declaration, comments, and assembler directives.

2. For instructions, the parser is to minimally be capable of decoding 2, 3 and 4-byte instructions as follows:

a) 2-byte with 1 or 2 symbolic register reference (e.g., TIXR A, ADDR S, A)

b) 3-byte PC-relative with symbolic operand to include immediate, indirect, and indexed addressing

c) 3-byte absolute with non-symbolic operand to include immediate, indirect, and indexed addressing

d) 4-byte absolute with symbolic or non-symbolic operand to include immediate, indirect, and indexed addressing

3. The parser is to handle all storage directives (BYTE, WORD, RESW, and RESB).

4. The output of this phase should contain (at least):

a) The symbol table.

b) The source program in a format like the listing file described in your text book except that the object code is not generated as shown below.

c) A meaningful error message should be printed below the line in which the error occurred.

# Design:

The main idea of the design is to encapsulate each type of data in class and each one handles its own errors that it can detect and define the type of its data for example the main classes are the following:

* Assembler
* Statement
* OpCode
* Tables
* Error

When defining any statement, the statement starts to split the String and detect of there is a label or not and add the opcode and define its types if there Is anything wrong the statement add the suitable error for itself

Also, the assembler assigns errors that it can handle when found.

# Main data structure:

1. Symbol table: used as a hash map with the label as the key and the address as the value.
2. Object table: is assigned to the tables class as a static list of type op code which is filled with the data in the object code file on the start of the program
3. Error table: is a list of type error that is loaded in the start of the program and is used to define errors
4. Directives: are loaded inside the object table too and the isDirective Boolean var is assigned to true
5. A list of the recorded statements after the first path for farther validations and easier object code generation in path 2

# Algorithms description

The main algorithms as somehow like the main flow chart of the book except that some loop conditions are changed to make error handling easier for example:

The stopping condition of the first pass of the assembler is not finding the end statement, it stops when it reaches the end of the file to make it easier to handle the absence of end statement when the end of file is reached

A close up of a screen

Description automatically generated

Also, after finishing the normal first path algorithms with some errors handled the code re-loop on all statements to find more errors that the main path may not be able to handle.

For example:

The (undefined symbol in operand) cannot be detected until the first pass is completed and all the symbols are added to the symbol table.

for (Statement statement : statements) {  
 if (!statement.isComment() && statement.getOpCode() != null) {  
 if (!statement.getOpCode().isRegisterType() && !statement.getOpCode().isDirective() &&  
 statement.hasFirstOperand()) {  
 if (statement.getFirstOperand().startsWith("#") && !statement.getFirstOperand().startsWith("#0")) {  
 String[] spliter = statement.getFirstOperand().split("#");  
 if (!*SYMTABLE*.containsKey(spliter[1])) {  
 statement.setError(*ERRORS*.get(5));  
 }  
 }  
 if (statement.hasSecondOperand()) {  
 if (statement.getSecondOperand().startsWith("#")&& statement.getFirstOperand().startsWith("#0")) {  
 String[] spliter = statement.getFirstOperand().split("#");  
 if (!*SYMTABLE*.containsKey(spliter[1])) {  
 statement.setError(*ERRORS*.get(5));  
 }  
 }  
 }  
 }  
 }  
}

NOTE: this is one of the undefined label definitions also there is other full loops that check for the same error but with different conditions.

After finishing all the validations, the assembler loop one last time on the statements, write them with there assigned addresses in the list file and loop on the symbol table and print the data inside it with its addresses.

# Assumptions:

1. Only the errors on the errors file are handled
2. Any literal (hex or dec) must start with zero to distinguish it from symbols.
3. BONUS: The code is designed to handle free form assembly language so there is no need for the misplaced operand or opcode errors so they are removed ( missing op code is handles ) it also detects if there is a label or not and detect of the label is legal or not.
4. BONUS ERRORS:
5. missing START statement
6. not a data type
7. end label must be the same as start label
8. illegal label ( label that is the same as opcode )

# Sample runs:

Source:

.234567890123456789  
SORT START 1000  
 LDX #0  
 LDA #0a48p  
 LDT #0  
LOOP LDA #STR  
 ADDR X,A  
 TIX LEN  
 JEQ BREAK  
 LDS #STR  
 ADDR X,w  
 STA P1  
 STS P2  
 LDCH @P3  
 RMO A,S  
 LDCH @P1  
 COMPR A,S  
 JGT SWAP  
 J LOOP  
 lda RMO T,A  
 ADD #01  
 COMP LEN  
BREAK LDX #0  
 RMO A,T  
 J LOOP  
SWAP LDCH @P  
 STCH TEMP

List file:

.234567890123456789  
001000 SORT START 1000  
001000 LDX #0  
 \*\*\* not a hexadecimal string  
001003 LDA #0a48p  
001006 LDT #0  
001009 LOOP LDA #STR  
00100c ADDR X,A  
00100e TIX LEN  
001011 JEQ BREAK  
001014 LDS #STR  
 \*\*\* illegal address for a register  
001017 ADDR X,w  
001019 STA P1  
00101c STS P2  
 \*\*\* undefined symbol in operand  
00101f LDCH @P3  
001022 RMO A,S  
001024 LDCH @P1  
001027 COMPR A,S  
001029 JGT SWAP  
00102c J LOOP  
 \*\*\*\* illegal label  
000000 LDA RMO T,A  
00102f ADD #01  
001032 COMP LEN

Source file:

.234567890123456789  
SEAR START 1000  
 LDA #0  
 LDX #0  
LOOP LDA #STR  
 +(ADDR) X,A  
 STA TEMP  
 LDA #0  
 LDCH @TEMP  
 COMP CHAR  
 JEQ FOUND  
 TIX LE  
 JEQ NOTF  
 J LOOP  
FOUND LDA TEMP  
. J DONE  
NOTF LDA XX  
.DONE J \*  
TEMP RESW 1  
STR BYTE C'FFREWQRFG'  
STR BYTE X'FFREWQRFG'  
XX RESW 1  
LEN WORD 9  
CHAR WORD 65  
 END SEARCH

List file:

.234567890123456789  
001000 SEAR START 1000  
001000 LDA #0  
001003 LDX #0  
001006 LOOP LDA #STR  
 \*\*\* can’t be format 4 instruction  
000000 +(ADDR) X,A  
001009 STA TEMP  
00100c LDA #0  
00100f LDCH @TEMP  
001012 COMP CHAR  
001015 JEQ FOUND  
 \*\*\* undefined symbol in operand  
001018 TIX LE  
00101b JEQ NOTF  
00101e J LOOP  
001021 FOUND LDA TEMP  
. J DONE  
001024 NOTF LDA XX  
.DONE J \*  
001027 TEMP RESW 1  
00102a STR BYTE C'FFREWQRFG'  
 \*\*\* not a hexadecimal string  
001033 STR BYTE X'FFREWQRFG'  
001038 XX RESW 1  
00103b LEN WORD 9  
00103e CHAR WORD 65  
 \*\*\*\* end label must be the same as start label  
001041 END SEARCH

Symbol Table:

-----------------------------------------------  
  
 SYMBOL TABLE   
  
 Name Value  
 -----------------------  
 STR 00102a  
 XX 001038  
 TEMP 001027  
 LEN 00103b  
 SEAR 001000  
 LOOP 001006  
 NOTF 001024  
 CHAR 00103e  
 FOUND 001021