SIC/XE Assembler

(Phase 2)

Made with JAVA

George Amgad Wadie 5002

# Requirements specification

# 1. The output of the assembler should include :

# a) Object-code file whose format is the same as the one described in the textbook in section 2.1.1 and 2.3.5.

# b) A report at the end of pass2. Pass1 and Pass2 errors should be included as part of the assembler report, exhibiting both the erroneous lines of source code and the error.

# 2. The assembler should support:

# a) EQU and ORG statements.

# b) Simple expression evaluation. A simple expression includes simple (A <op> B) operand arithmetic, where <op> is one of +, -, \*, / and no spaces surround the operation, e.g. A+B.Design:

After phase 1 is complete and the program length is counted all the statements that are comments or directives with no instructions are removed from the list then the header record is written followed by a loop to write the text records, separating directives from code then finished with the end record. No additional classes are used.

# 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
6. No additional data structures are used for phase 2, just some string builders to help concatenate the records of the object file

# Algorithms description

At first if there is any error in the statements, phase 2 returns and doesn’t build the object file to avoid exceptions and useless faulty records then remove all the statements that are comments or directives and doesn’t contain instruction

statements.removeIf(Statement::isComment);  
statements.removeIf(statement -> statement.getMnemonic().isDirective() && statement.getInstruction() == null);

then a loop on each instruction starts creating the instruction for each statement also compute the displacement if PC relative or Base relative or immediate and concatenate all of this to a string builder and make sure to deal with various bases ( binary, hex... ) types properly also the String.format method make sure that the String is formed in a proper length of characters to fit the instruction or address size

for example :

instruction.append(String.*format*("%05X", displacement));

instruction.append(String.*format*("%03X", displacement));

each one of these append the displacement to the instruction with 5 or 3 number of chars and X indicate an uppercase hex decimal number

after that the header record is written in the file with the program size and the starting address calculated from phase 1 then a final loop on statements with proper validations to make sure that the text record is written in the right format and the directives are separated from the operations then finish all of that with the end record

# Assumptions:

1. The base relative addressing is not handled but could easily be implemented
2. The literals are implemented in WORD and BYTE only with all the proper validations and errors and the object code creation
3. The EQU and ORG are fully implemented with the expression evaluation
4. Simple expression evaluation is implemented for any operand type without changing the errors handled in phase 1 also the expression evaluation method applies some errors if found
5. BONUS ERRORS OF PHASE 2:

\*\*\*\* can't be indexed instruction  
\*\*\*\* unspecified source register  
\*\*\*\* this statement requires a label  
\*\*\*\* this statement cannot have a label  
\*\*\*\* cannot evaluate expression  
\*\*\*\* odd length of hex string

# Sample runs:

Source:

.234567890123456789  
SEARCH START 1000  
 LDA #0  
 LDX #0  
LOOP LDA #STR  
 ADDR X,A  
 +STA TEMP  
 LDA #345  
 LDCH @TEMP  
 COMP CHAR  
 JEQ FOUND+2  
 +TIX LEN  
 JEQ NOTF  
 J LOOP  
FOUND LDA TEMP  
 J LOOP  
NOTF LDA XX  
TEMP RESW 1  
STR BYTE C'FFREWQRFG'  
STRA BYTE X'123AB6F8'  
XX RESW 3  
LEN WORD 9  
CHAR WORD 65  
 END SEARCH

List:

.234567890123456789  
001000 SEARCH START 1000  
001000 LDA #0  
001003 LDX #0  
001006 LOOP LDA #STR  
001009 ADDR X,A  
00100B +STA TEMP  
00100F LDA #345  
001012 LDCH @TEMP  
001015 COMP CHAR  
001018 JEQ FOUND+2  
00101B +TIX LEN  
00101F JEQ NOTF  
001022 J LOOP  
001025 FOUND LDA TEMP  
001028 J LOOP  
00102B NOTF LDA XX  
00102E TEMP RESW 1  
001031 STR BYTE C'FFREWQRFG'  
00103A STRA BYTE X'123AB6F8'  
00103E XX RESW 3  
001047 LEN WORD 9  
00104A CHAR WORD 65  
00104D END SEARCH  
  
  
-----------------------------------------------  
  
 SYMBOL TABLE   
  
 Name Value  
 -----------------------  
 STR 001031  
 XX 00103E  
 TEMP 00102E  
 LEN 001047  
 SEARCH 001000  
 LOOP 001006  
 NOTF 00102B  
 STRA 00103A  
 CHAR 00104A  
 FOUND 001025

Object:

HSEARCH^001000^00004C  
T001000^1B^01000005000001202890100F10102E0101595220192B203233200C  
T00101B^13^2F1010473320093F2FE10320063F2FDB032010  
T001031^09^464652455751524647  
T00103A^04^123AB6F8  
T001047^03^000009  
T00104A^03^000041  
E001000

Same code assembled in the SIC simulator:

H ^001000^00004C

T001000^1B^01000005000001202890100F10102E0101595220192B203233200C

T00101B^13^2F1010473320093F2FE10320063F2FDB032010

T001031^0D^464652455751524647123AB6F8

T001047^06^000009000041

E001000

EQU & ORG:

Source:

.234567890123456789  
 START 1000  
STAB RESB 1100  
SYMBOL EQU STAB  
VALUE EQU STAB+6  
FLAGS EQU STAB+9  
 END

List:

.234567890123456789  
001000 START 1000  
001000 STAB RESB 1100  
00144C SYMBOL EQU STAB  
00144C VALUE EQU STAB+6  
00144C FLAGS EQU STAB+9  
00144C END  
  
  
-----------------------------------------------  
  
 SYMBOL TABLE   
  
 Name Value  
 -----------------------  
 001000  
 STAB 001000  
 SYMBOL 001000  
 FLAGS 001009  
 VALUE 001006

Source:

.234567890123456789  
STAW START 1000  
STAB RESB 1100  
 ORG STAB  
SYMBOL RESB 6  
VALUE RESW 1  
FLAGS RESB 2  
 ORG STAB+1100  
 END STAW

List:

.234567890123456789  
001000 STAW START 1000  
001000 STAB RESB 1100  
00144C ORG STAB  
001000 SYMBOL RESB 6  
001006 VALUE RESW 1  
001009 FLAGS RESB 2  
00100B ORG STAB+1100  
00144C END STAW  
  
  
-----------------------------------------------  
  
 SYMBOL TABLE   
  
 Name Value  
 -----------------------  
 STAW 001000  
 STAB 001000  
 SYMBOL 001000  
 FLAGS 001009  
 VALUE 001006