CS222 Systems Programming

SIC/XE Assembler

Phase II Report

Team Members

Mohamed Esmail	(53)
Mohamed Kamal Elshazly	(58)
Mohamed Mashaal	(60)
Mahmoud Tarek Samir	(62)
Youssef Ali	(73)

Table of Contents

Introduction **2**

Pass II Specifications **2**

Requirement Specifications **2**

Design **3**

Pass 1 Modules 3

Source Program Module 3

Validator Module 3

Assembly Listing Module 3

Operation Table Module 4

Symbol Table Module 4

Pass 2 Modules 4

Pass 2 Module 4

Literal Table Module 4

Expression Evaluator Module 4

Main Data Structures **5**

Map 5

Set 5

Dynamic array (Vector) 5

Algorithms Description **6**

Assumptions **7**

Sample Runs **8**

Source Code **9**

Introduction

An **assembler** is a type of computer program that interprets software programs written in assembly language into machine language, code and instructions that can be executed by a computer. It enables software and application developers to access, operate and manage a computer's hardware architecture and components.

An assembler primarily serves as the bridge between symbolically coded instructions written in assembly language and the computer processor, memory and other computational components. An assembler works by assembling and converting the source code of assembly language into object code or an object file that constitutes a stream of zeros and ones of machine code, which are directly executable by the processor.

Assemblers are classified based on the number of times it takes them to read the source code before translating it; there are both single-pass and multi-pass assemblers.

SIC (Simplified Instructional Machine) is a hypothetical computer that has been carefully designed to include the hardware features most often found on real machines, while avoiding unusual or irrelevant complexities. XE “extra equipment” is a version of the SIC that have additional features such as different addressing modes, larger memory, more registers, etc.

This report provides a description of the implementation details of phase II of SIC/XE two-pass assembler in C/C++. This phase includes using the output of phase I to implement pass 2 in which the instruction are assembled. The output of this phase is the object program.

Pass II Specifications

In pass 2, the following should be accomplished:

1. Assemble instructions (translating operation codes and looking up addresses).
2. Generate data values defined by BYTE, WORD, etc.
3. Perform processing of assembler directives not done during Pass 1.
4. Write the object program and the assembly listing.

Requirement Specifications

In Phase II, It’s required to implement **pass 2** of the assembler. It includes the following :

- Supporting:
    1. EQU and ORG statements.

2. Simple expression evaluation. A simple expression includes simple (A <op> B) operand arithmetic, where <op> is one of +,-,*,/ and no spaces surround the operation, eg. A+B.

3. Literals (Including LTORG)

=C’<ASCII-TEXT>’, =X’HEX-TEXT’, =<DECIMAL-TEXT> forms

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

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

  2. A report at the end of pass2. Pass1 and Pass2 errors should be included as part of the assembler report, exhibiting both the offending line of source code and the error.

Design

Pass 1 Modules

We have five main modules

Source Program Module

It parses all the file. For each line it gets each word inside and put it in class called “Source Line” then it passes it to validator module to check the syntax and if it is okay, it increases the location counter and save the symbols if exist by operation table module and symbol table module. Then it calls assembly listing module to write in the file the line, location counter, and error if exist.

Validator Module

It takes the source line which has all the data about one statement and check its syntax, return true if it is right otherwise return a meaningful error message.

Assembly Listing Module

It writes inside each line and the value of location counter on this line and the error if exist inside the listing file.

Operation Table Module

It stores all the information (Operation Code, Format Type and Number of Operands) about each operation in map.

Symbol Table Module

It stores each symbol in the code. Source Program uses it to insert new symbol (location and type) and to search if this symbol is already inserted in the table. It used also in expressions validation and evaluation.

Pass 2 Modules

We have three main modules

Pass 2 Module

It takes the programs source lines and generates the corresponding object code to each source line using the *Object Code Generator* helper module and writes the final object program's header, text, end and modification records generated with the help of the *Object Program* helper module

Literal Table Module

It stores each literal in the program code in map. The program have only one instance of this module so it's designed to be a singleton module.

Expression Evaluator Module

It evaluates the expressions written in EQU and ORG statements. It takes a simple expression in the form of <operand><operator><operand> and returns its evaluated address value and type (relative or absolute) if it's a valid expression. Otherwise, if the expression contains some error(s) (i.e.: undefined symbol or invalid expression), it returns a suitable error message.

Main Data Structures

Map

To save all symbols we find and to check if there is no symbol exist more than one time.
The second use is to save the information about each operation like format type, number of operands and operation code.
The third use is to save all literals used in the program. The fourth use is to save all SIC/XE registers.

Set

To save all directives so that if there is operation which is not in operation table and not in directive set then it is undefined operation and error will appear.

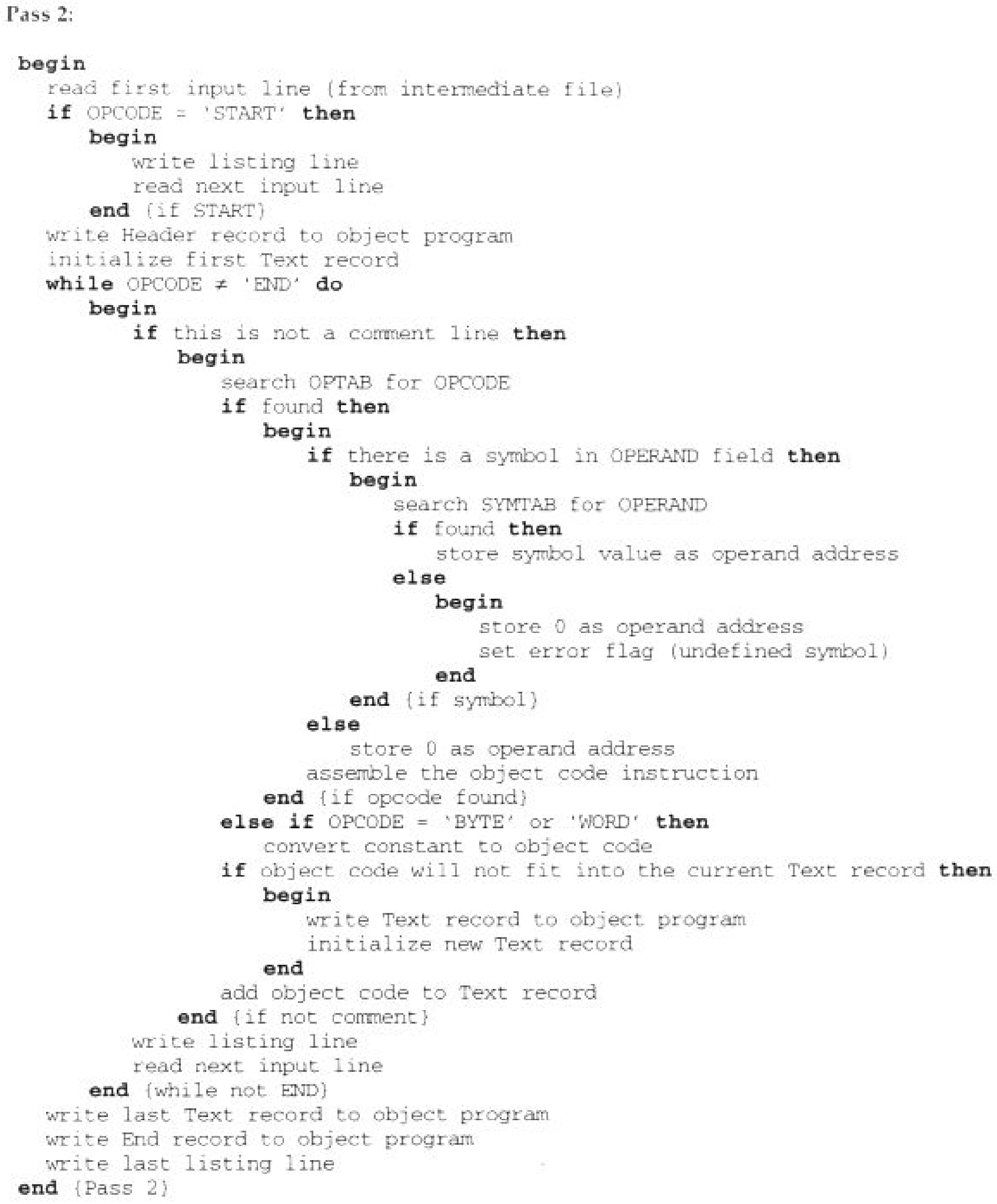
Dynamic array (Vector)

We use it to store each word in the statement we make it dynamic because each line have different numbers of words.

It's also used in many places throughout the program modules and functions.

Algorithms Description

The main work of the assembler in pass 2 is the following



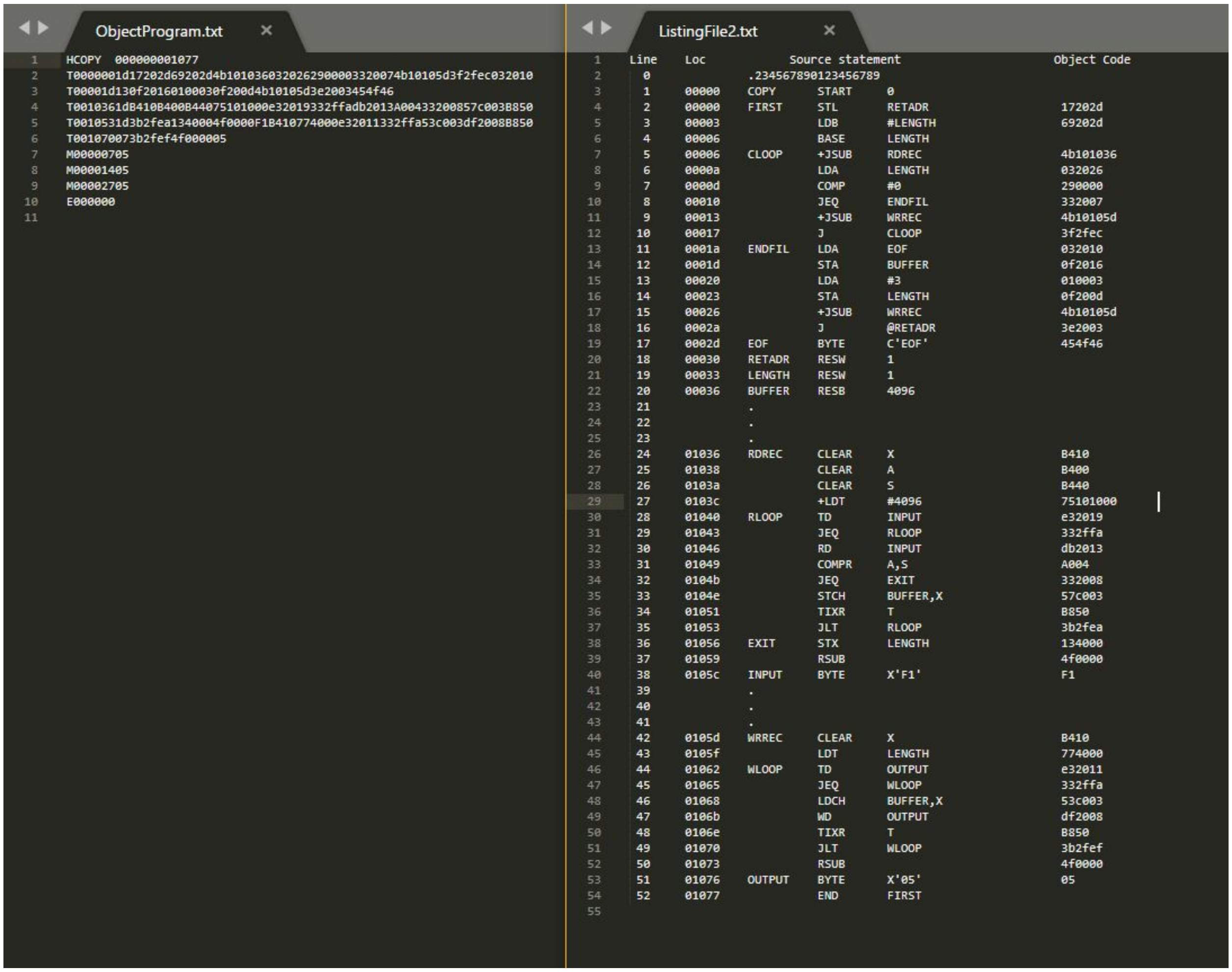
Pass 2:

```
begin
  read first input line (from intermediate file)
  if OPCODE = 'START' then
     begin
        write listing line
        read next input line
     end {if START}
  write Header record to object program
  initialize first Text record
  while OPCODE ≠ 'END' do
     begin
        if this is not a comment line then
           begin
              search OPTAB for OPCODE
              if found then
                 begin
                    if there is a symbol in OPERAND field then
                       begin
                          search SYMTAB for OPERAND
                          if found then
                             store symbol value as operand address
                          else
                             begin
                                store 0 as operand address
                                set error flag (undefined symbol)
                             end
                       end {if symbol}
                    else
                       store 0 as operand address
                    assemble the object code instruction
                 end {if opcode found}
              else if OPCODE = 'BYTE' or 'WORD' then
                 convert constant to object code
              if object code will not fit into the current Text record then
                 begin
                    write Text record to object program
                    initialize new Text record
                 end
              add object code to Text record
           end {if not comment}
        write listing line
        read next input line
     end {while not END}
  write last Text record to object program
  write End record to object program
  write last listing line
end {Pass 2}
```

Assumptions

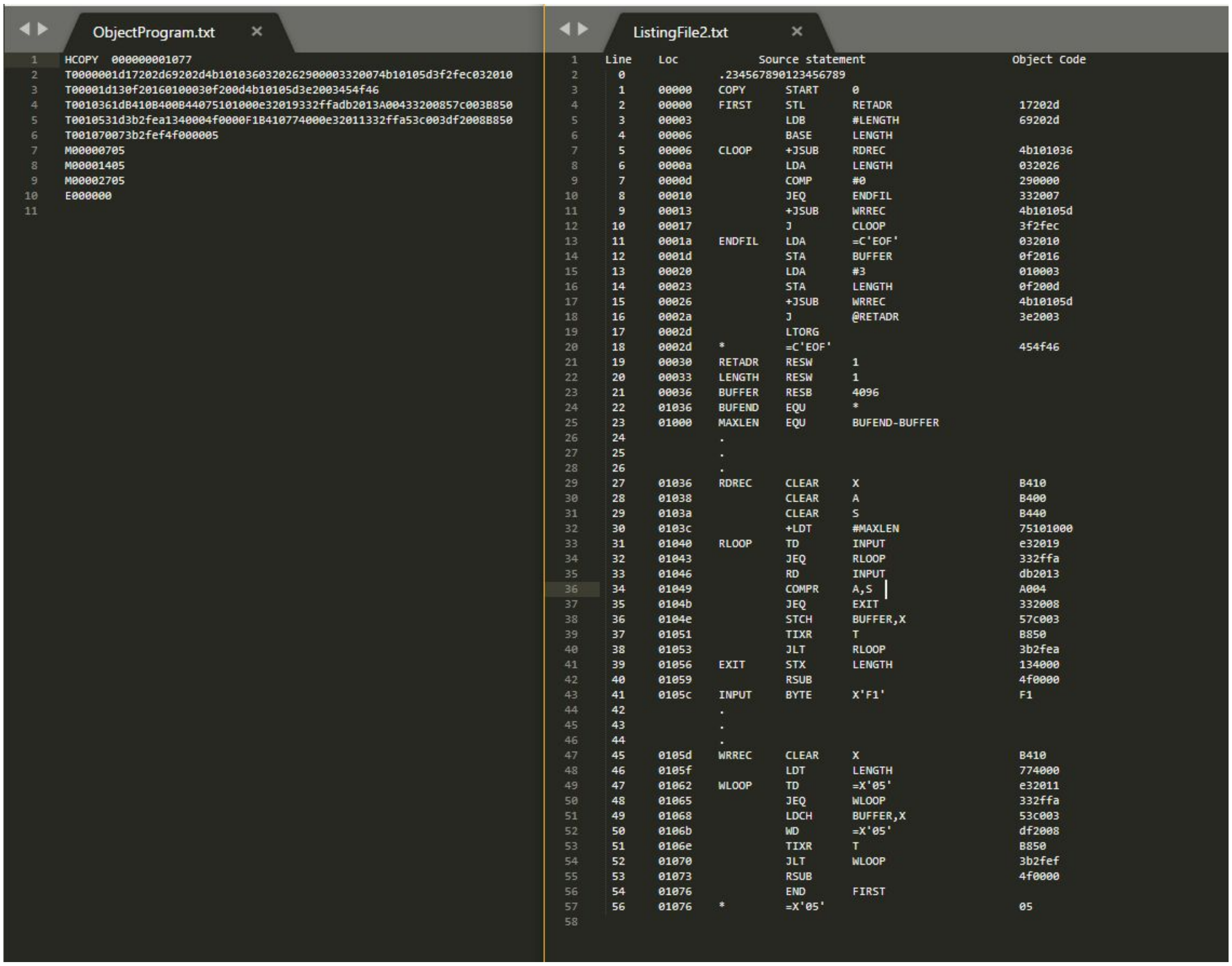
- User will use free format.
- There is no empty line in the source program.
- We support the following directives
  - Storage directives (i.e. BYTE, WORD, RESB, and RESW)
  - START and END directives.
  - EQU and ORG.
  - LTORG
  - BASE
- We support case insensitive so the user can write the source program in either capital case or small case.
- We support using literals and simple expressions.
- Label must start with a letter and consist of letters, numbers, or '_'.

Sample Runs



```
HCOPY  000000001077
T0000001d17202d69202d4b101036032026290000332007 4b10105d3f2fec032010
T00001d130f20160100030f200d4b10105d3e2003454f46
T0010361dB410B400B44075101000e32019332ffadb2013A00433200857c003B850
T0010531d3b2fea1340004f0000F1B410774000e32011332ffa53c003df2008B850
T001070073b2fef4f000005
M00000705
M00001405
M00002705
E000000
```

```
Line   Loc          Source statement                      Object Code
 0            .234567890123456789
 1     00000  COPY      START     0
 2     00000  FIRST     STL       RETADR                   17202d
 3     00003            LDB       #LENGTH                  69202d
 4     00006            BASE      LENGTH
 5     00006  CLOOP     +JSUB     RDREC                    4b101036
 6     0000a            LDA       LENGTH                   032026
 7     0000d            COMP      #0                       290000
 8     00010            JEQ       ENDFIL                   332007
 9     00013            +JSUB     WRREC                    4b10105d
10     00017            J         CLOOP                    3f2fec
11     0001a  ENDFIL    LDA       EOF                      032010
12     0001d            STA       BUFFER                   0f2016
13     00020            LDA       #3                       010003
14     00023            STA       LENGTH                   0f200d
15     00026            +JSUB     WRREC                    4b10105d
16     0002a            J         @RETADR                  3e2003
17     0002d  EOF       BYTE      C'EOF'                   454f46
18     00030  RETADR    RESW      1
19     00033  LENGTH    RESW      1
20     00036  BUFFER    RESB      4096
21            .
22            .
23            .
24     01036  RDREC     CLEAR     X                        B410
25     01038            CLEAR     A                        B400
26     0103a            CLEAR     S                        B440
27     0103c            +LDT      #4096                    75101000
28     01040  RLOOP     TD        INPUT                    e32019
29     01043            JEQ       RLOOP                    332ffa
30     01046            RD        INPUT                    db2013
31     01049            COMPR     A,S                      A004
32     0104b            JEQ       EXIT                     332008
33     0104e            STCH      BUFFER,X                 57c003
34     01051            TIXR      T                        B850
35     01053            JLT       RLOOP                    3b2fea
36     01056  EXIT      STX       LENGTH                   134000
37     01059            RSUB                               4f0000
38     0105c  INPUT     BYTE      X'F1'                    F1
39            .
40            .
41            .
42     0105d  WRREC     CLEAR     X                        B410
43     0105f            LDT       LENGTH                   774000
44     01062  WLOOP     TD        OUTPUT                   e32011
45     01065            JEQ       WLOOP                    332ffa
46     01068            LDCH      BUFFER,X                 53c003
47     0106b            WD        OUTPUT                   df2008
48     0106e            TIXR      T                        B850
49     01070            JLT       WLOOP                    3b2fef
50     01073            RSUB                               4f0000
51     01076  OUTPUT    BYTE      X'05'                    05
52     01077            END       FIRST
```



```
HCOPY  000000001077
T0000001d17202d69202d4b1010360320262900003320074b10105d3f2fec032010
T00001d130f20160100030f200d4b10105d3e2003454f46
T0010361dB410B400B44075101000e32019332ffadb2013A00433200857c003B850
T0010531d3b2fea1340004f0000F1B410774000e32011332ffa53c003df2008B850
T001070073b2fef4f000005
M00000705
M00001405
M00002705
E000000
```

```
Line    Loc          Source statement                       Object Code
  0             .23456789012345678 9
  1     00000   COPY      START     0
  2     00000   FIRST     STL       RETADR                   17202d
  3     00003             LDB       #LENGTH                  69202d
  4     00006             BASE      LENGTH
  5     00006   CLOOP     +JSUB     RDREC                    4b101036
  6     0000a             LDA       LENGTH                   032026
  7     0000d             COMP      #0                       290000
  8     00010             JEQ       ENDFIL                   332007
  9     00013             +JSUB     WRREC                    4b10105d
 10     00017             J         CLOOP                    3f2fec
 11     0001a   ENDFIL    LDA       =C'EOF'                  032010
 12     0001d             STA       BUFFER                   0f2016
 13     00020             LDA       #3                       010003
 14     00023             STA       LENGTH                   0f200d
 15     00026             +JSUB     WRREC                    4b10105d
 16     0002a             J         @RETADR                  3e2003
 17     0002d             LTORG
 18     0002d   *         =C'EOF'                            454f46
 19     00030   RETADR    RESW      1
 20     00033   LENGTH    RESW      1
 21     00036   BUFFER    RESB      4096
 22     01036   BUFEND    EQU       *
 23     01000   MAXLEN    EQU       BUFEND-BUFFER
 24             .
 25             .
 26             .
 27     01036   RDREC     CLEAR     X                        B410
 28     01038             CLEAR     A                        B400
 29     0103a             CLEAR     S                        B440
 30     0103c             +LDT      #MAXLEN                  75101000
 31     01040   RLOOP     TD        INPUT                    e32019
 32     01043             JEQ       RLOOP                    332ffa
 33     01046             RD        INPUT                    db2013
 34     01049             COMPR     A,S                      A004
 35     0104b             JEQ       EXIT                     332008
 36     0104e             STCH      BUFFER,X                 57c003
 37     01051             TIXR      T                        B850
 38     01053             JLT       RLOOP                    3b2fea
 39     01056   EXIT      STX       LENGTH                   134000
 40     01059             RSUB                               4f0000
 41     0105c   INPUT     BYTE      X'F1'                    F1
 42             .
 43             .
 44             .
 45     0105d   WRREC     CLEAR     X                        B410
 46     0105f             LDT       LENGTH                   774000
 47     01062   WLOOP     TD        =X'05'                   e32011
 48     01065             JEQ       WLOOP                    332ffa
 49     01068             LDCH      BUFFER,X                 53c003
 50     0106b             WD        =X'05'                   df2008
 51     0106e             TIXR      T                        B850
 52     01070             JLT       WLOOP                    3b2fef
 53     01073             RSUB                               4f0000
 54     01076             END       FIRST
 56     01076   *         =X'05'                             05
```

Source Code

https://github.com/MahmoudTarek97/SIC-XE-Assembler/