

SIC/XE Assembler

Names

Afnan Mousa	15
Khadija Assem	27
Shimaa Kamal	34
Nourhan Magdy	69

Overview

The SIC machine has basic addressing, storing most memory addresses in hexadecimal integer format. Similar to most modern computing systems, the SIC architecture stores all data in binary and uses the two's complement to represent negative values at the machine level. Memory storage in SIC consists of 8-bit bytes, and all memory addresses in SIC are byte addresses. Any three consecutive bytes form a 24-bit 'word' value, addressed by the location of the lowest numbered byte in the word value. There is also a more complicated machine built on top of SIC called the Simplified Instruction Computer with Extra Equipment (SIC/XE). The XE expansion of SIC adds a 48-bit floating point data type, an additional memory addressing mode, and extra memory to the original machine. All SIC assembly code is upwards compatible with SIC/XE.

Goals

The project is to implement **One Pass SIC/XE Assembler**.assembler, producing code for the absolute loader.

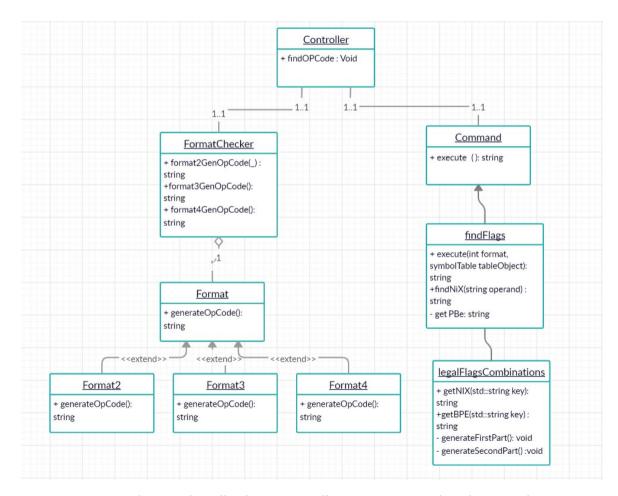
Requirements Specification

- 1. Reading the source lines from a file.
- 2. A parser that is capable of handling source lines that are instructions, storage declaration, comments, and assembler directives.
- 3. The parser is to minimally be capable of decoding 2, 3 and 4-byte instructions.
- 4. The parser is to handle all storage directives (BYTE, WORD, RESW, and RESB).
- 5. The assembler should support:
 - a- EQU and ORG statements.
 - b- Simple expression evaluation. A simple expression includes simple (A B) operand
 - arithmetic, where is one of +, -, *, / and no spaces surround the operation, eg.

 A+B.
- 6. The output of the assembler should include (at least):
 - a- Intermediate file that contains the symbol table and a definition for any error.
 - b- Object-code file.

Design

- The user inserts the name of the file followed by (.txt) which contain the statements which need to be assembled, then:
 - The <u>Main</u> class calls the class <u>ReadFromFile</u> to read the statements one by one by for loop, in this loop the <u>Parser</u> is called to check that the statement is valid and doesn't have any syntax error. The Parser does that in two parts, first part by calling a function on it, and the second by calling Class <u>DynamicTable</u> which checks that the statement doesn't have any logical error. And if it's valid then it calls class <u>Make Address</u> to generate the Address as location counter of this line, then setting all (Location Counter, Label, Operation, Operant) in map SymbolMap by call class <u>SymbolTable</u>.
 - After reading the line and checking if this is a valid one, class



<u>ReadFromFile</u> calls class <u>Controller</u> to generate the object code.

Main data structures

I. Map (Hash map)

- ➤ Map to store the definition of data which follow the key word such as(WORD, BYTE, RESW, REWB, ORG, EQU).
- ➤ Map to store all Register names as key and his object code as parameters which Register are valid to use in Sic/XE.
- ➤ Map to store all Operation names as keys and his object code as parameters which Operation are valid to use in Sic/XE.

 \triangleright

II. Vector

- > Vector to store the Operant which is defined as follower reference .
- > Vector to split the operand which content of two parts one is Data and other is Register X or operand can be two register the vector store parts to check if the operant is valid .

III. Array One Dimensional

- ➤ One dimension array in size seven elements to store the static label such as(WORD, BYTE, RESW, REWB, ORG, EQU) as String.
- One dimension array of size three to store the Label, Operation , Operand.

Algorithms description

- > First of all, the assembler reads the source lines from a file line by line and parsing it then sends it to the object code generator.
- ➤ Function **CheckWithRegex** in **Parsing** Class checks that the syntax of the line is correct, and splits the line to (Label Operation Operand), and then checks that all of them are valid, for example check the operation belongs to **SIC/XE**.

- ➤ Class <u>DynamicTables</u> contains function <u>BuildDataTable</u> which check that statement don't have any logic error or any sentence error by many conditions such as:
 - Check that the Operand which after the keyword "WORD" the length less than 5 if the Operand is number, the length less than 8 and not contain white space if Operand is string and in two case must the length be more than zero.
 - Check that the Operand which after the keyword <u>"BYTE"</u> the length less than 15 if the Operand is a hexadecimal number, is even number and contain character "X" and single quote, the length less than 16 and check that contain character "C" and single quote in the first and in the last and in two case must the length be more than zero.
 - Check that the Operand which after the keyword <u>"RESB"</u> or <u>"RESW"</u> the length less than 5 number ,must be more than zero.
 - Check that the Operand which after the keyword <u>"END"</u> or <u>"RESW"</u> the length be zero or in case "END" label not equal the label in the directive start.
 - Check that the Operand which after the keyword <u>"START"</u> the length more than zero and less than 5 and the Operand must be a hexadecimal number.
 - If the first character of Operand be '@', '#', '*' must check that Operation not be any type of which deal with register and the operand size in case of '*' must be one if the Operand after this symbol is word must check if have defined already if not store it to deal on it as forward reference if Operand is number the length must be less than 5 more than zero.

If Operand is two registers or register x and and word define or not must check that the Operation is at the end of character <u>'R'</u> or is equal <u>"STCH"</u>, "LDCH", and otherwise which deal on registers.

- Class <u>MakeAddress</u> contain function <u>LocationCounter</u> which return the (LOC) of every line as string of hexadecimal number that make by:
 - Define int variable <u>LOC</u> as static the initial value of it is the Operant which became after directive <u>Start</u>.
 - Then to calculate the value of location counter two any line take the value of LOC and ADD on it according to every situation:
 - > If Operation is "RESB" then add Loc=Loc+Length of Operand
 - ➤ If Operation is "RESW" then add Loc=Loc+(Length of Operand) *3
 - ➤ If Operation in format 4 then add LOC = 4 + LOC

- ➤ If Operation is "WORD" then add Loc=Loc+3
- ➤ If Operation is "BYTE" then add Loc=Loc+(Length of Operand-3) if the Operant is character if is hexadecimal then Loc=Loc+(Length of Operand-3)/2.
- ➤ Otherwise Loc=Loc+3 and for all that return the string equal the value of LOC at hexadecimal.
- ➤ If Operation is "ORG", then the next address will be the address of the operand. And if there is no operand, then it returns the default address.
- ➤ If Operation is "EQU", then the current address becomes equal to the value in the operand field or the address of the operand.

> For the object code generation :

- The <u>Controller</u> specifies which format is the instruction then calls the <u>formatchecker</u> that decides which pass to take from the three formats passess whether format 2 or 3 or 4
 - The 3 formats computes the first part of the object code by relatively the same way then for format 2 it finds the hexadecimal corresponding to each register used for format 3 and 4 takes the next step towards "find flags" class.
 - Find flags class specify which addressing mode the instruction uses and gets the corresponding flags combination (NIX) from "legalFlagsCombinationsClass" then performs some calculations to find the proper (BPE) part of the object code and gets it from "legalFlagsCombinationsClass".
 - After that the "<u>CalculateDisplacement</u>" does its work by calculating the displacement.
 - The controller combines the parts of the object code returned by each class and converts it to hexadecimal and stores the object code in the symbol table.

> For the forward referencing part :

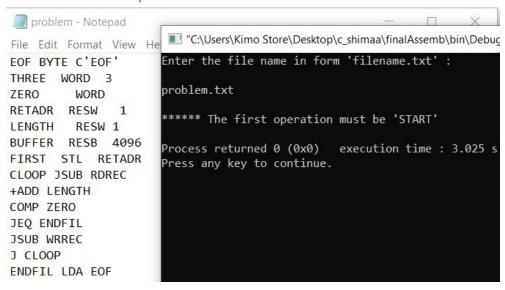
- If the <u>CalculateDisplacement</u> class found an operand symbol that isn't in the symbol table(pre defined) it stores it in the "<u>LocationsTable</u>" singleton class and stores the address of the instruction which is missing this operand.
- When the parser finds this symbol it gives it's address to "AddSymbol" class and it automatically completes the computation of the incomplete object codes.

Assumptions

> Owr program dealing with forward path technique for all Operand Not only in <u>JSUB</u> instructions but in any Operandand label except the Operand Which comes before Operation (Base) the Operandmust be defined before using it.

Sample runs

First the parser check that the operation of the first instruction is "START" **Ex.1:** The first line operation isn't "Start"

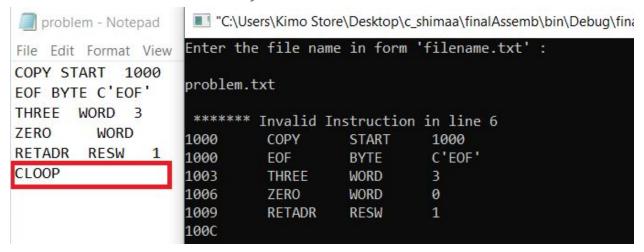


Then the parser checks that the instruction is a valid one with "regex". We can say the instruction is correct if:

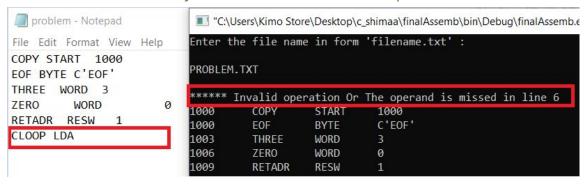
- It contains 3 words, which are label, operation and operand.
- It contains 2 words, operation and operand.
- It contains 1 word, like (RSUB, ORG, END).

So anything else will be considered as an invalid instruction. Let's try that.

Ex.2: The instruction is "CLOOP" only:



Ex.3: The instruction is only two words but it misses the operand.



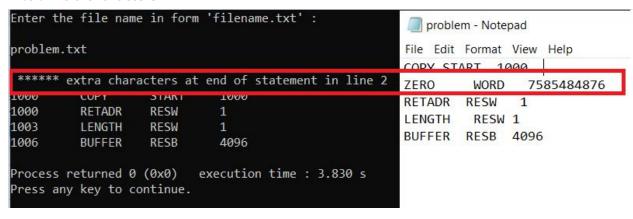
Ex.4: Check that the operation is existing in SIC/XE "STORE" isn't an operation in SIC/XE

```
problem - Notepad
                       "C:\Users\Kimo Store\Desktop\c_shimaa\finalAssemb\bin\Debug\finalAssemb.e
                     Enter the file name in form 'filename.txt' :
File Edit Format View
COPY START 1000
                     problem.txt
STORE LENGTH
                      1000
                                COPY
                                          START
                                                     1000
                      1000
                                          STORE
                      ****** Invalid operation Or The operand is missed in line 2
                      Process returned 0 (0x0)
                                                  execution time : 5.579 s
```

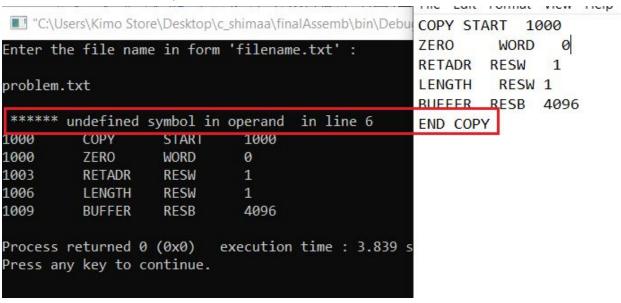
Ex.5: When RSUB has an operand

```
"C:\Users\Kimo Store\Desktop\c_shimaa\finalAssemb\bin\L
                                                  problem - Notepad
Enter the file name in form 'filename.txt' :
                                                 File Edit Format View Help
                                                 COPY START 1000
problem.txt
                                                 EOF BYTE C'EOF'
                                                 THREE WORD 3
1000
          COPY
                    START
                               1000
                                                 ZERO
                                                           WORD
1000
          EOF
                    BYTE
                               C'EOF'
1003
          THREE
                    WORD
                                                 RSUB CLOOP
                               3
1006
          ZERO
                    WORD
                               0
                                                          RESW
                                                 RETADR
1009
                    RSUB
                                         400000
                                                 LENGTH
                                                            RESW 1
****** RSUB doesn't need an operand
                                                 BUFFER RESB 4096
100C
          RETADR
                    RESW
                              1
100F
          LENGTH
                    RESW
                               1
1012
          BUFFER
                    RESB
                               4096
Process returned 0 (0x0)
                            execution time: 42.
Press any key to continue.
```

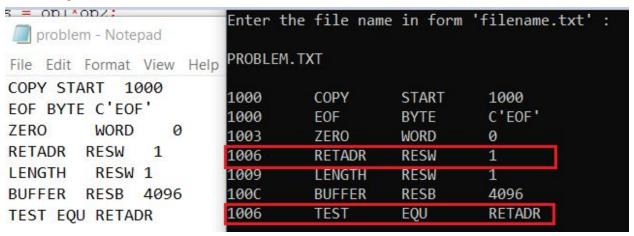
Ex.6: Extra characters



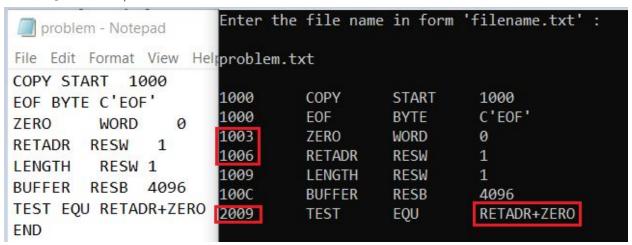
Ex.7: When End has an Operand



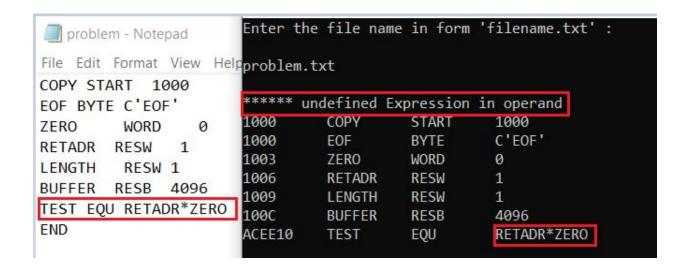
Ex.8: EQU



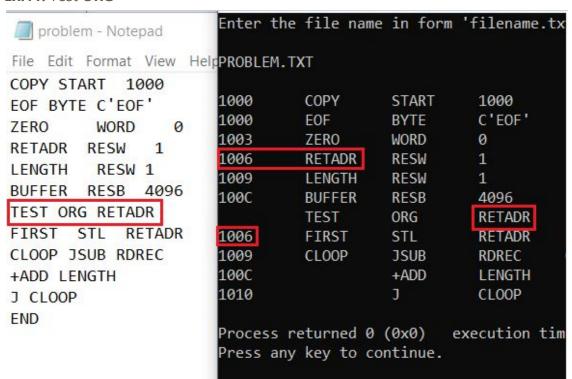
Ex.9: EQU with Expression



Ex.10: Invalid expression, multiplication is illegal for type relative



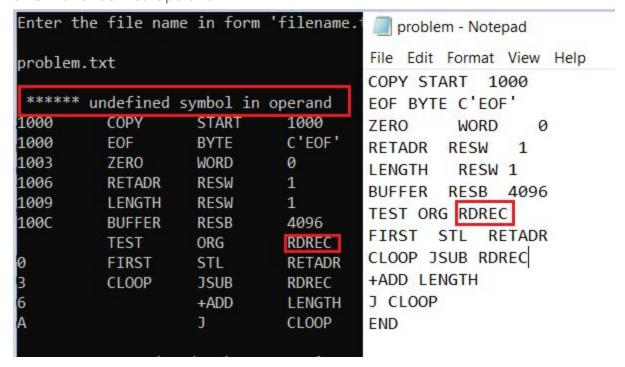
Ex.11: Test ORG



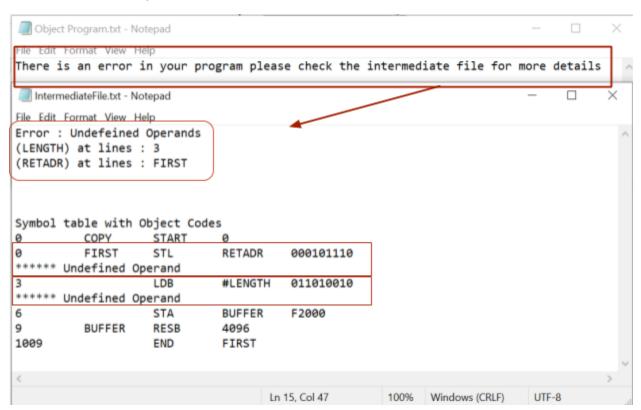
Two **ORG**

problem - Notepad	Enter th	ne file nam	e in form	n 'filename
File Edit Format View F	_{le} PROBLEM.	TXT		
COPY START 1000 EOF BYTE C'EOF' ZERO WORD 0 RETADR RESW 1 LENGTH RESW 1 BUFFER RESB 4096 TEST ORG RETADR FIRST STL RETADR CLOOP JSUB RDREC ORG +ADD LENGTH J CLOOP END	1000 1000 1003 1006 1009 100C 1006 1009	COPY EOF ZERO RETADR LENGTH BUFFER TEST FIRST CLOOP		1000 C'EOF' 0 1 1 4096 RETADR RETADR RDREC LENGTH CLOOP

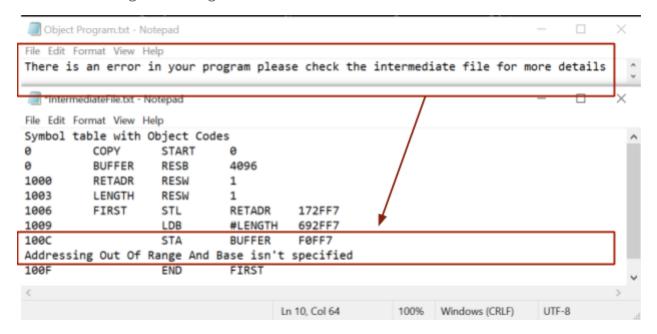
ORG with undefined Operand



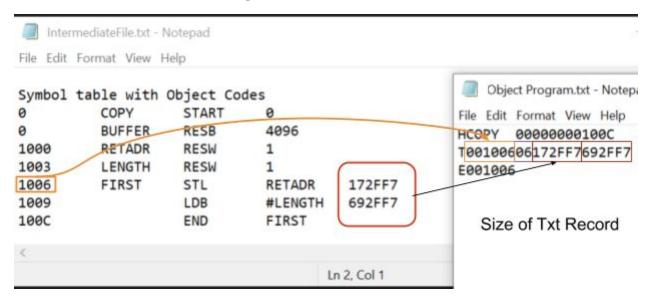
Ex.12: Undefined Operands:



Ex.13: Addressing out of range:



Ex.13: Without forward referencing:



Ex.13: The Book's Example:

5	0000	COPY	START	0	MANAGEMENT AND
10	0000	FIRST	SIL	RETADR	17202D
12	0003		LDB	#LENGTH	69202D
13			BASE	LENGTH	
15	0006	CLOOP	+JSUB	RDREC	4B101036
20	000A		LDA	LENGTH	032026
25	000D		COMP	#0	290000
30	0010		JEQ	ENDFIL	332007
35	0013		+JSUB	WRREC	4B10105D
40	0017		J	CLOOP	3F2FEC
45	001A	ENDFIL	LDA	EOF	032010
50	001D		-7STA	BUFFER	0F2016
55	0020		LDA	#3	010003
60	0023		STA	LENGTH	0F200D
65	0026		*JSUB	WRREC	4B10105D
70	002A		J	GRETADR	3E2003
80	002D	EOF	BYTE	C'EOF'	454F46
95	0030	RETADR	RESW	1	
100	0033	LENGTH	RESW	1	1 1 1 1 1 1 1 1 1 1 1 1
105	0036	BUFFER	RESB	4096	
110	-		1400		
115		-	SUBBOUT	TINE TO READ RE	CORD INTO BUFFER
120				10 10 100	
125	1036	RDREC	CLEAR	X	B410
130	1038		CLEAR	A	B400
132	103A		CLEAR	S	B440
133	103C		+LDT	#4096	75101000
135	1049	RLOOP	TD	INPUT	E32019
140	1043	10001	JEQ	RLOOP	332FFA
145	1046		RD	INPUT	DB2013
150	1049		COMPR	A,S	A004
155	104B		JEQ	EXIT	332008
160	104E		STCH	BUFFER, X	57C003
165	1051		TIXR	T	B850
170	1053		JLT	RLOOP	3B2FEA
175	1056	EXIT	STX	LENGTH	134000
180	1059	Ent.	RSUB	and Total a	4F0000
185	105C	INPUT	BYTE	X'F1'	F1
195	1030	THEOT	DITE	2 27	
200			CHEDOLE	TIME ON MOTTE D	ECORD FROM BUFFER
205			SUDENOU.	THE TO HELLE IN	and the second
210	105D	WRREC	CLEAR	×	B410
212	105F	MINIST	LDT	LENGTH	774000
212	1062	WLOOP	TD	OUTPUT	E32011
220		HLOOP		MILOOP	332FFA
225	1065 1068		JEQ LDCH	BUFFER, X	530003
230			WD	OUTPUT	DF2008
	106B		TIXR	T	B850
235	106E		JUT "		3B2FEF
240	1070		RSUB	MLOOP	4F0000
245 250	1073	CUTPUT	BYTE	X'05'	05
255	1076	OUTPUT		FIRST	03
633			END	E41031	

Symbol	table with Oh	oject Cod	des	
0	COPY	START	0	
0	FIRST	STL	RETADR	17202D
3		LDB	#LENGTH	69202D
6	CLOOP	+JSUB	RDREC	4B101036
A		LDA	LENGTH	32026
D		COMP	#0	290000
10		JEQ	ENDFIL	332007
13		+JSUB	WRREC	4B10105D
17		J	CLOOP	3F2FEC
1A	ENDFIL	LDA	EOF	32010
1D		STA	BUFFER	F2016
20		LDA	#3	10003
23		STA	LENGTH	F200D
26		+JSUB	WRREC	4B10105D
2A		J	@RETADR	3E2003
2D	EOF	BYTE	C'EOF'	
30	RETADR	RESW	1	
33	LENGTH	RESW	1	
36	BUFFER	RESB	4096	
1036	RDREC	CLEAR	X	B410
1038		CLEAR	A	B400
103A		CLEAR	S	B440
103C		+LDT	#4096	75101000
1040	RLOOP	TD	INPUT	E32019
1043		JEQ	RLOOP	332FFA
1046		RD	INPUT	DB2013
1049		COMPR	A,S	A004
104B		JEQ	EXIT	332008
104E		STCH	BUFFER, X	570003
1051		TIXR	T	B850
1053		JLT	RLOOP	3B2FEA
1056	EXIT	STX	LENGTH	134000
1059		RSUB		4F0000
*****	RSUB doesn't	need an	operand	
105C	INPUT	BYTE	X'F1'	
105D	WRREC	CLEAR	X	B410
105F		LDT	LENGTH	774000
1062	WLOOP	TD	OUTPUT	E32011
1065		JEQ	WLOOP	332FFA
1068		LDCH	BUFFER, X	53C003
106B		WD	OUTPUT	DF2008
106E		TIXR	T	B850
1070		JLT	WLOOP	3B2FEF
1073		RSUB	9276 FT 075	4F0000
*****	RSUB doesn't		operand	CONTRACTOR OF STREET
1076	OUTPUT	BYTE	X'05'	
1077		END	FIRST	



Object Program.txt - Notepad

File Edit Format View Help

HCOPY 000000001077

T0000001D17202D69202D4B101036320262900003320074B10105D3F2FEC32010 T00001D13F201610003F200D4B10105D3E2003

T0010361DB410B400B44075101000E32019332FFADB2013A00433200857C003B850 T0010531D3B2FEA1340004F0000B410774000E32011332FFA53C003DF2008B850 T001070073B2FEF4F0000

E000000



