

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 Main class calls the class ReadFromFile to read the statements one by one by for loop, in this loop the Parser 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 DynamicTable which checks that the statement doesn't have any logical error. And if it's valid then it calls class Make Address to generate the Address as location counter of this line, then setting all (Location Counter, Label ,Operation, Operant) in map SymbolMap by call class SymbolTable.
 - ◆ After reading the line and checking if this is a valid one, class ReadFromFile calls class Controller to generate the object code.
- For the object code generation :
 - ◆ The Controller specifies which format is the instruction then calls the formatchecker 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 "CalculateDisplacement" 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 CalculateDisplacement class found an operand symbol that isn't in the symbol table(pre defined) it stores it in the "LocationsTable" 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.

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 .
-

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 operand 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 the operation belongs to **SIC/XE**.
- Class **DynamicTables** contains function **BuildDataTable** 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 "BYTE" 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 "RESB" or "RESW" the length less than 5 number ,must be more than zero.
- Check that the Operand which after the keyword "END" or "RESW" 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 "START" 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 'R' or is equal "STCH", "LDCH", and otherwise which deal on registers.

- Class **MakeAddress** contain function **LocationCounter** which return the (LOC) of every line as string of hexadecimal number that make by:
- Define int variable LOC as static the initial value of it is the Operant which became after directive Start.
 - 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 $Loc = Loc + \text{Length of Operand}$
 - If Operation is "RESW" then add $Loc = Loc + (\text{Length of Operand}) * 3$
 - 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 $Loc = Loc + (\text{Length of Operand} - 3)$ if the Operant is character if is hexadecimal then
 $Loc = Loc + (\text{Length of Operand} - 3) / 2$.
 - 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.

Assumptions

- Our program dealing with forward path technique for all Operand Not only in JSUB instructions but in any Operand and label except the Operand Which comes before Operation (Base) the Operand must 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"

```

problem - Notepad
File Edit Format View Help
EOF BYTE C'EOF'
THREE WORD 3
ZERO WORD
RETADR RESW 1
LENGTH RESW 1
BUFFER RESB 4096
FIRST STL RETADR
CLOOP JSUB RDREC
+ADD LENGTH
COMP ZERO
JEQ ENDFIL
JSUB WRREC
J CLOOP
ENDFIL LDA EOF

"C:\Users\Kimo Store\Desktop\c_shimaa\finalAssemb\bin\Debug
Enter the file name in form 'filename.txt' :
problem.txt

***** The first operation must be 'START'

Process returned 0 (0x0) execution time : 3.025 s
Press any key to continue.

```

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:

The screenshot shows a Notepad window titled "problem - Notepad" with the following assembly code:

```

COPY START 1000
EOF BYTE C'EOF'
THREE WORD 3
ZERO WORD
RETADR RESW 1
CLOOP

```

The "CLOOP" instruction is highlighted with a red box. To the right, a command prompt window shows the command to assemble the file:

```

"C:\Users\Kimo Store\Desktop\c_shimaa\finalAssemb\bin\Debug\finalAssemb.exe" /f "problem.txt"

```

The output of the command prompt shows the error:

```

***** Invalid Instruction in line 6
1000 COPY START 1000
1000 EOF BYTE C'EOF'
1003 THREE WORD 3
1006 ZERO WORD 0
1009 RETADR RESW 1
100C

```

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

The screenshot shows a Notepad window titled "problem - Notepad" with the following assembly code:

```

COPY START 1000
EOF BYTE C'EOF'
THREE WORD 3
ZERO WORD 0
RETADR RESW 1
CLOOP LDA

```

The "CLOOP LDA" instruction is highlighted with a red box. To the right, a command prompt window shows the command to assemble the file:

```

"C:\Users\Kimo Store\Desktop\c_shimaa\finalAssemb\bin\Debug\finalAssemb.exe" /f "PROBLEM.TXT"

```

The output of the command prompt shows the error:

```

***** Invalid operation Or The operand is missed in line 6
1000 COPY START 1000
1000 EOF BYTE C'EOF'
1003 THREE WORD 3
1006 ZERO WORD 0
1009 RETADR RESW 1

```

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

The screenshot shows a debugger window with the title bar "C:\Users\Kimo Store\Desktop\c_shimaa\finalAssemb\bin\Debug\finalAssemb.a". The main window displays assembly code for a file named "problem.txt". The code includes instructions like "COPY START 1000" and "STORE LENGTH". A red box highlights the error message: "***** Invalid operation Or The operand is missed in line 2". The process returned 0 (0x0) and the execution time was 5.579 s.

```

Enter the file name in form 'filename.txt' :
problem.txt

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

The screenshot shows a debugger window with the title bar "C:\Users\Kimo Store\Desktop\c_shimaa\finalAssemb\bin\Debug\finalAssemb.a". The main window displays assembly code for a file named "problem.txt". The code includes instructions like "COPY START 1000", "EOF BYTE C'EOF'", "THREE WORD 3", "ZERO WORD 0", and "RSUB". A red box highlights the error message: "***** RSUB doesn't need an operand". The process returned 0 (0x0) and the execution time was 42.0 s. A second window titled "problem - Notepad" shows the source code for "problem.txt", which includes instructions like "COPY START 1000", "EOF BYTE C'EOF'", "THREE WORD 3", "ZERO WORD 0", "RSUB CLOOP", "RETADR RESW 1", "LENGTH RESW 1", and "BUFFER RESB 4096".

```

Enter the file name in form 'filename.txt' :
problem.txt

1000      COPY      START      1000
1000      EOF       BYTE      C'EOF'
1003      THREE     WORD      3
1006      ZERO      WORD      0
1009      RSUB
***** RSUB doesn't need an operand
100C      RETADR    RESW      1
100F      LENGTH    RESW      1
1012      BUFFER    RESB      4096

Process returned 0 (0x0)   execution time : 42.0 s
Press any key to continue.

```


Ex.6 : Extra characters

Enter the file name in form 'filename.txt' :
problem.txt
***** extra characters at end of statement in line 2
1000 COPY START 1000
1000 RETADR RESW 1
1003 LENGTH RESW 1
1006 BUFFER RESB 4096
Process returned 0 (0x0) execution time : 3.830 s
Press any key to continue.

problem - Notepad
File Edit Format View Help
COPY START 1000
ZERO WORD 7585484876
RETADR RESW 1
LENGTH RESW 1
BUFFER RESB 4096

Ex.7 :When End has an Operand

"C:\Users\Kimo Store\Desktop\c_shimaa\finalAssemb\bin\Debug
Enter the file name in form 'filename.txt' :
problem.txt
***** undefined symbol in operand in line 6
1000 COPY START 1000
1000 ZERO WORD 0
1003 RETADR RESW 1
1006 LENGTH RESW 1
1009 BUFFER RESB 4096
Process returned 0 (0x0) execution time : 3.839 s
Press any key to continue.

File Edit Format View Help
COPY START 1000
ZERO WORD 0
RETADR RESW 1
LENGTH RESW 1
BUFFER RESB 4096
END COPY

Ex.8: EQU

problem - Notepad

File Edit Format View Help

```

COPY START 1000
EOF BYTE C'EOF'
ZERO WORD 0
RETADR RESW 1
LENGTH RESW 1
BUFFER RESB 4096
TEST EQU RETADR

```

Enter the file name in form 'filename.txt' :

PROBLEM.TXT

1000	COPY	START	1000
1000	EOF	BYTE	C'EOF'
1003	ZERO	WORD	0
1006	RETADR	RESW	1
1009	LENGTH	RESW	1
100C	BUFFER	RESB	4096
1006	TEST	EQU	RETADR

Ex.9: EQU with Expression

problem - Notepad

File Edit Format View Help

```

COPY START 1000
EOF BYTE C'EOF'
ZERO WORD 0
RETADR RESW 1
LENGTH RESW 1
BUFFER RESB 4096
TEST EQU RETADR+ZERO
END

```

Enter the file name in form 'filename.txt' :

problem.txt

1000	COPY	START	1000
1000	EOF	BYTE	C'EOF'
1003	ZERO	WORD	0
1006	RETADR	RESW	1
1009	LENGTH	RESW	1
100C	BUFFER	RESB	4096
2009	TEST	EQU	RETADR+ZERO

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

The screenshot shows a Notepad window titled 'problem - Notepad' with the following assembly code:

```
File Edit Format View Help
COPY START 1000
EOF BYTE C'EOF'
ZERO WORD 0
RETADR RESW 1
LENGTH RESW 1
BUFFER RESB 4096
TEST EQU RETADR*ZERO
END
```

Next to it is a command prompt window titled 'Enter the file name in form 'filename.txt' :'. It shows the file 'problem.txt' has been loaded. The assembly listing is as follows:

Address	Label	OpCode	Operand
1000		COPY	START 1000
1000		EOF	BYTE C'EOF'
1003		ZERO	WORD 0
1006		RETADR	RESW 1
1009		LENGTH	RESW 1
100C		BUFFER	RESB 4096
ACEE10	TEST	EQU	RETADR*ZERO

An error message is displayed: '***** undefined Expression in operand'. The expression 'RETADR*ZERO' is highlighted in red in both the code and the listing.

Ex.11: Test **ORG**

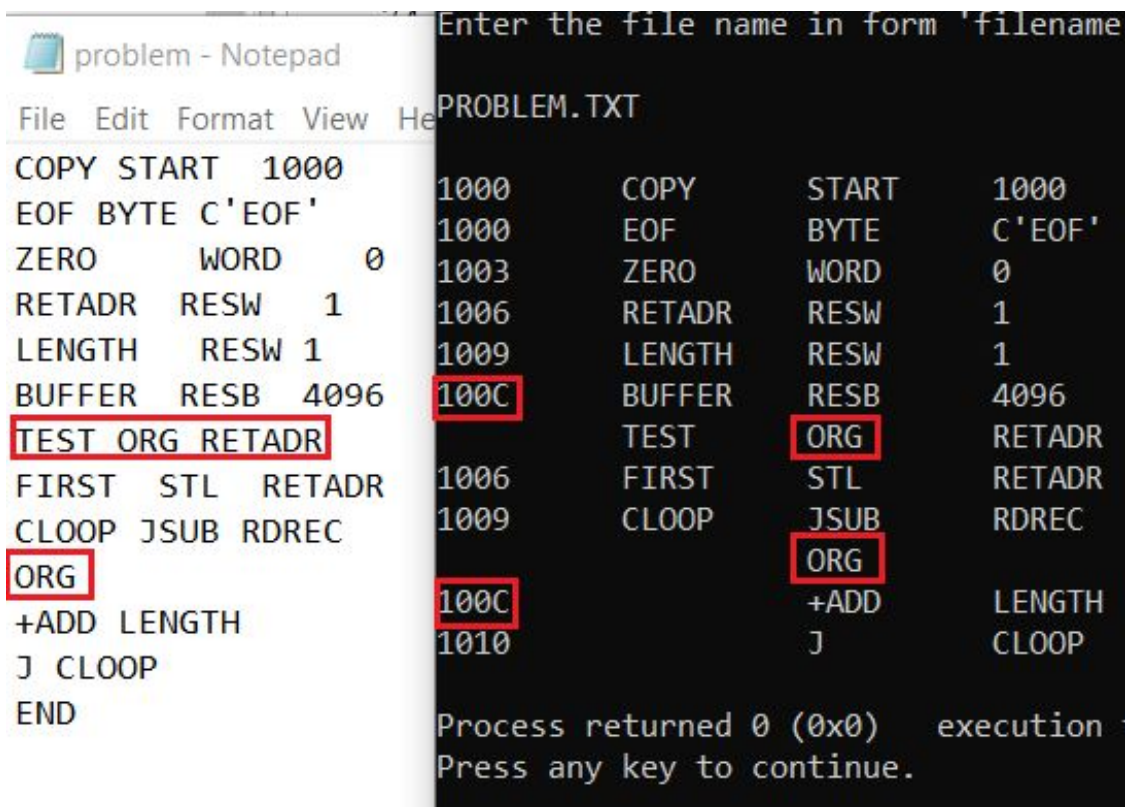
The screenshot shows a Notepad window titled 'problem - Notepad' with the following assembly code:

```
File Edit Format View Help
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
+ADD LENGTH
J CLOOP
END
```

Next to it is a command prompt window titled 'Enter the file name in form 'filename.tx'. It shows the file 'PROBLEM.TXT' has been loaded. The assembly listing is as follows:

Address	Label	OpCode	Operand
1000		COPY	START 1000
1000		EOF	BYTE C'EOF'
1003		ZERO	WORD 0
1006		RETADR	RESW 1
1009		LENGTH	RESW 1
100C		BUFFER	RESB 4096
	TEST	ORG	RETADR
1006		FIRST	STL RETADR
1009		CLOOP	JSUB RDREC
100C		+ADD	LENGTH
1010		J	CLOOP

The output shows 'Process returned 0 (0x0) execution time' and 'Press any key to continue.' The instructions 'TEST ORG RETADR' and 'FIRST STL RETADR' are highlighted in red in the code, and 'RETADR' is highlighted in red in the listing.

Two **ORG**


The image shows a Notepad window on the left with assembly code and a command prompt window on the right showing the execution of that code. In the Notepad code, the instructions `TEST ORG RETADR` and `ORG` are highlighted with red boxes. In the command prompt output, the addresses `100C` and the instruction `ORG` are also highlighted with red boxes.

```

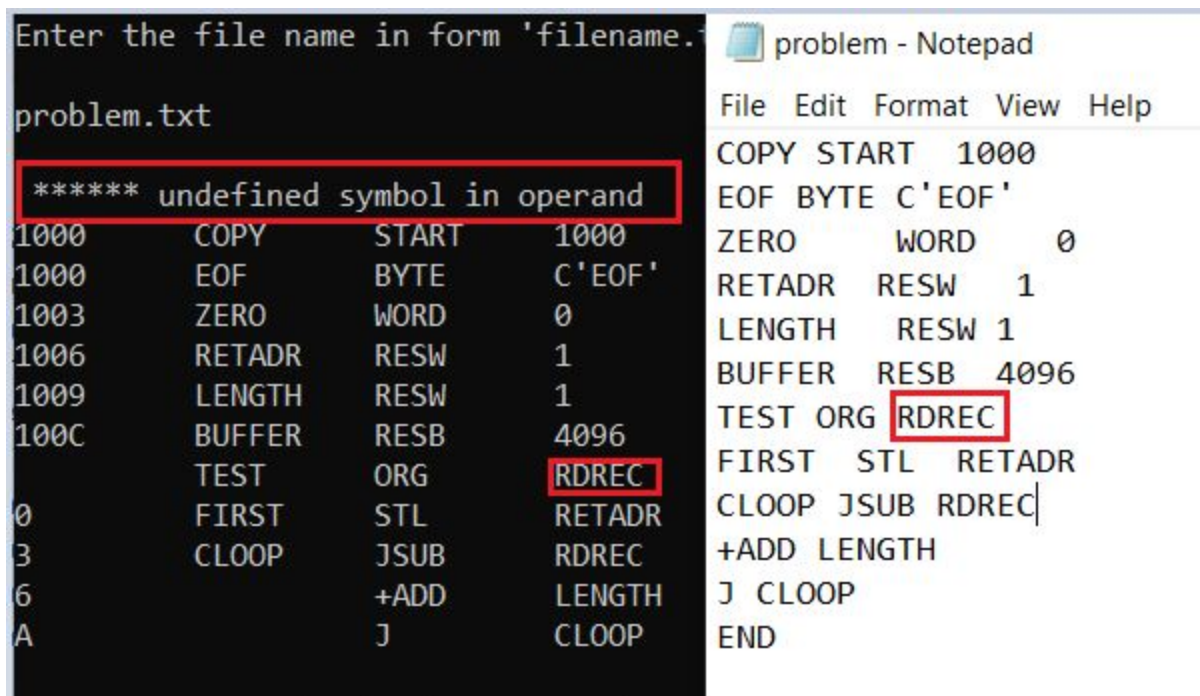
problem - Notepad
File Edit Format View Help
COPY START 1000
EOF BYTE C'EOF'
ZERO WORD 0
RETADR RESW 1
LENGTH RESW 1
BUFFER RESB 4096
TEST ORG RETADR
FIRST STL RETADR
CLOOP JSUB RDREC
ORG
+ADD LENGTH
J CLOOP
END

```

```

Enter the file name in form 'filename'
PROBLEM.TXT
1000 COPY START 1000
1000 EOF BYTE C'EOF'
1003 ZERO WORD 0
1006 RETADR RESW 1
1009 LENGTH RESW 1
100C BUFFER RESB 4096
100C TEST ORG RETADR
1006 FIRST STL RETADR
1009 CLOOP JSUB RDREC
100C +ADD LENGTH
1010 J CLOOP
Process returned 0 (0x0) execution
Press any key to continue.

```

ORG with undefined Operand


The image shows a Notepad window on the right with assembly code and a command prompt window on the left showing the execution of that code. In the Notepad code, the instruction `TEST ORG RDREC` is highlighted with a red box. In the command prompt output, the error message `***** undefined symbol in operand` is highlighted with a red box.

```

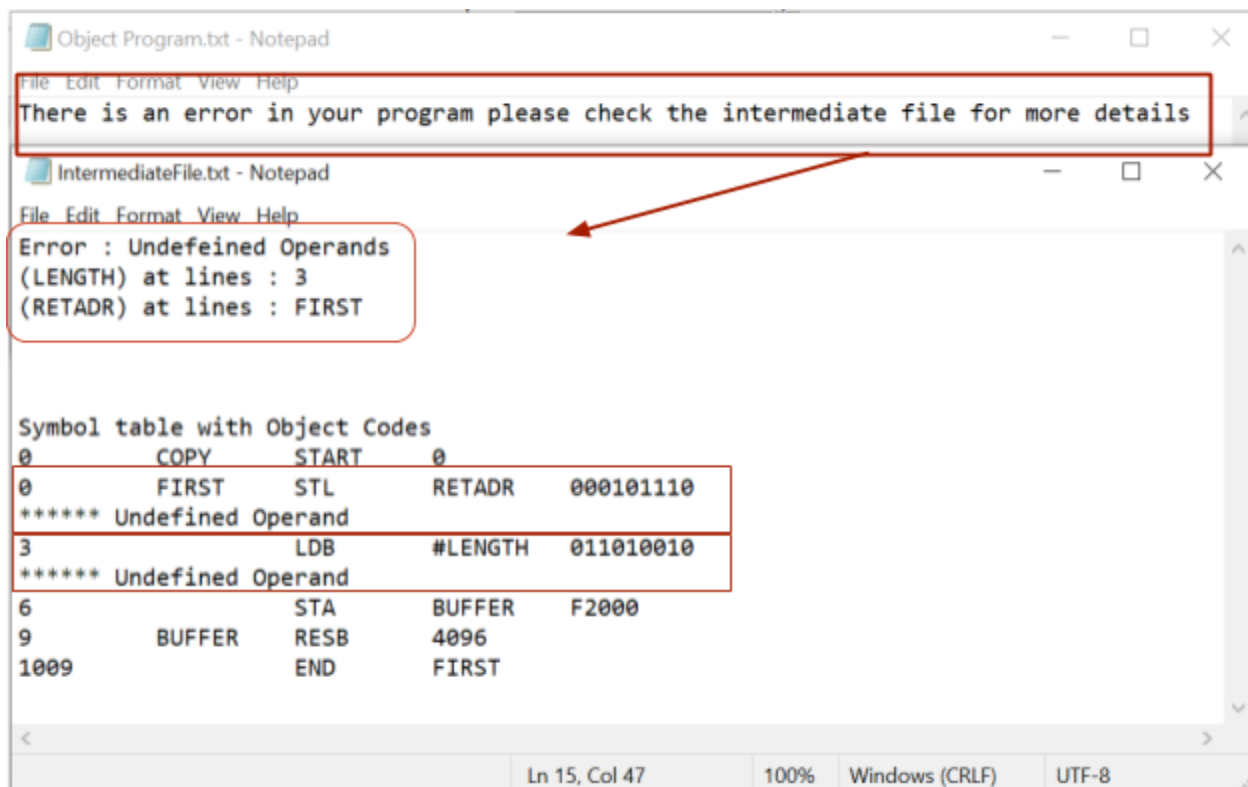
problem - Notepad
File Edit Format View Help
COPY START 1000
EOF BYTE C'EOF'
ZERO WORD 0
RETADR RESW 1
LENGTH RESW 1
BUFFER RESB 4096
TEST ORG RDREC
FIRST STL RETADR
CLOOP JSUB RDREC
+ADD LENGTH
J CLOOP
END

```

```

Enter the file name in form 'filename.'
problem.txt
***** undefined symbol in operand
1000 COPY START 1000
1000 EOF BYTE C'EOF'
1003 ZERO WORD 0
1006 RETADR RESW 1
1009 LENGTH RESW 1
100C BUFFER RESB 4096
100C TEST ORG RDREC
0 FIRST STL RETADR
3 CLOOP JSUB RDREC
6 +ADD LENGTH
A J CLOOP

```

Ex.12: Undefined Operands :


Object Program.txt - Notepad

File Edit Format View Help

There is an error in your program please check the intermediate file for more details

IntermediateFile.txt - Notepad

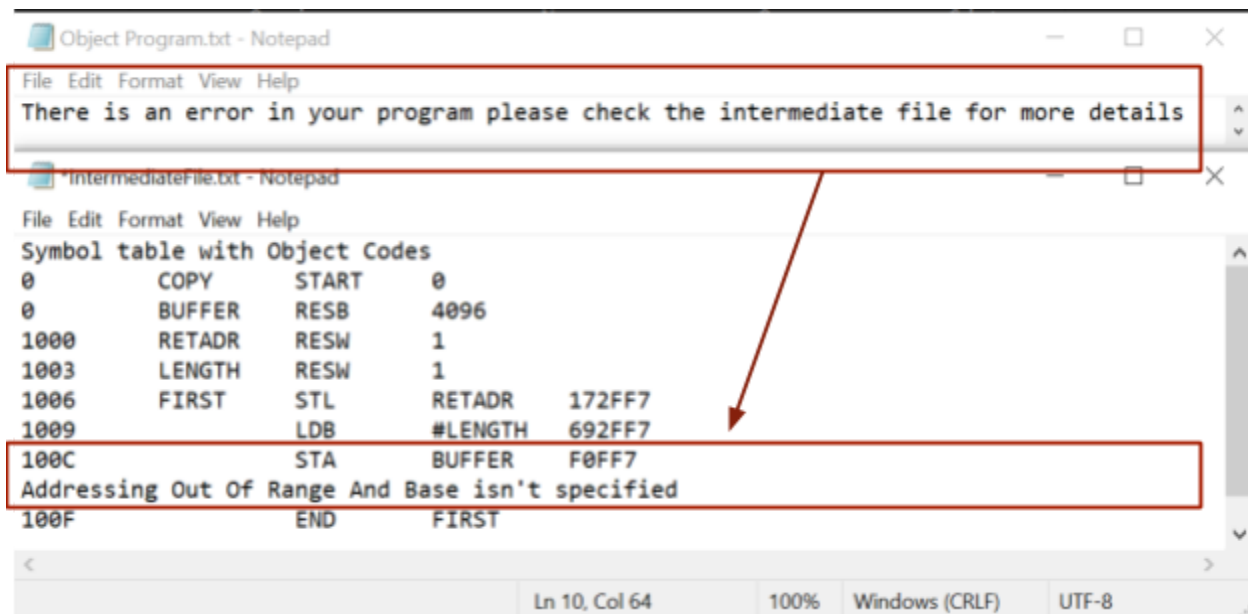
File Edit Format View Help

Error : Undefined Operands
(LENGTH) at lines : 3
(RETADR) at lines : FIRST

Symbol table with Object Codes

0	COPY	START	0
0	FIRST	STL	RETADR 000101110
*****	Undefined Operand		
3	LDB	#LENGTH	011010010
*****	Undefined Operand		
6	STA	BUFFER	F2000
9	BUFFER	RESB	4096
1009	END	FIRST	

Ln 15, Col 47 100% Windows (CRLF) UTF-8

Ex.13: Addressing out of range :


Object Program.txt - Notepad

File Edit Format View Help

There is an error in your program please check the intermediate file for more details

IntermediateFile.txt - Notepad

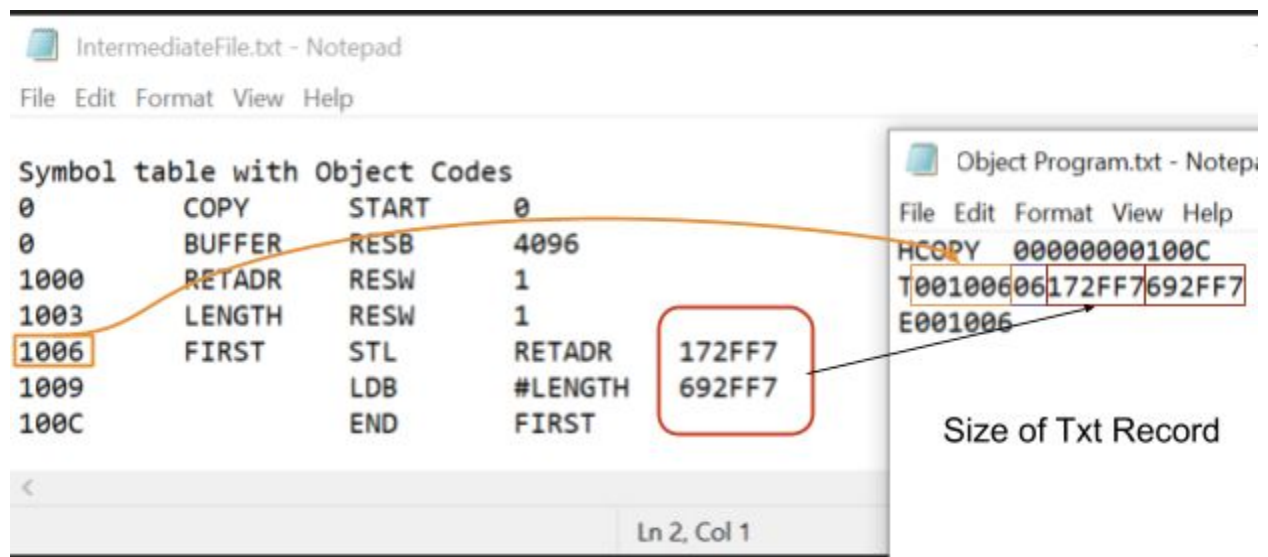
File Edit Format View Help

Symbol table with Object Codes

0	COPY	START	0
0	BUFFER	RESB	4096
1000	RETADR	RESW	1
1003	LENGTH	RESW	1
1006	FIRST	STL	RETADR 172FF7
1009	LDB	#LENGTH	692FF7
100C	STA	BUFFER	F0FF7
Addressing Out Of Range And Base isn't specified			
100F	END	FIRST	

Ln 10, Col 64 100% Windows (CRLF) UTF-8

Ex.13: Without forward referencing :



Ex.13: The Book's Example :

5	0000	COPY	START	0	
10	0000	FIRST	STL	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		→STA	BUFFER	0F2016
55	0020		LDA	#3	010003
60	0023		STA	LENGTH	0F200D
65	0026		+JSUB	WRREC	4B10105D
70	002A		J	@RETADR	3E2003
80	002D	EOF	BYTE	C'EOF'	454F46
95	0030	RETADR	RESW	1	
100	0033	LENGTH	RESW	1	
105	0036	BUFFER	RESB	4096	
110		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		
120		.			
125	1036	RDREC	CLEAR	X	B410
130	1038		CLEAR	A	B400
132	103A		CLEAR	S	B440
133	103C		+LDT	#4096	75101000
135	1040	RLOOP	TD	INPUT	E32019
140	1043		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		RSUB		4F0000
185	105C	INPUT	BYTE	X'F1'	F1
195		.			
200		.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205		.			
210	105D	WRREC	CLEAR	X	B410
212	105F		LDT	LENGTH	774000
215	1062	WLOOP	TD	OUTPUT	E32011
220	1065		JEQ	WLOOP	332FFA
225	1068		LDCH	BUFFER,X	53C003
230	106B		WD	OUTPUT	DF2008
235	106E		TIXR	T	B850
240	1070		JLT	WLOOP	3B2FEF
245	1073		RSUB		4F0000
250	1076	OUTPUT	BYTE	X'05'	05
255			END	FIRST	

The image shows three Notepad windows side-by-side, displaying assembly code, a symbol table, and object code.

problem.txt - Notepad

```

COPY START 0
FIRST STL RETADR
LOB WLENGTH
BASE LENGTH
CLOOP +TSUB RPREC
LOA LENGTH
COMP #0
REQ ENDFIL
+TSUB MPREC
J CLOOP
ENDFIL LOA EOF
STA BUFFER
LOA #3
STA LENGTH
+TSUB MPREC
J RETADR
EOF BYTE C'EOF'
RETAOR RESW 3
LENGTH RESW 1
BUFFER RESW ARNG
RPREC CLEAR X
      CLEAR A
      CLEAR S
+JLT NARNG
RLOOP TO INPUT
REQ RLOOP
RD INPUT
COMP A,S
REQ EXIT
STCH BUFFER,X
TDOR T
JLT RLOOP
EXIT STX LENGTH
RSUB
INPUT BYTE X'F1'
MPREC CLEAR X
LOA LENGTH
WLOOP TO OUTPUT
REQ WLOOP
LOCH BUFFER,X
WD OUTPUT
TDOR T
JLT WLOOP
RSUB
OUTPUT BYTE X'05'
END FIRST

```

***IntermediateFile.txt - Notepad**

Symbol table with Object Codes

Address	OpCode	OpName	OpCode	OpName
0	COPY	START	0	
0	FIRST	STL	RETADR	172003
3	LOB	WLENGTH		062003
6	CLOOP	+TSUB	RPREC	48103034
A	LOA	LENGTH		53016
D	COMP	#0		290000
10	REQ	ENDFIL		312007
13	+TSUB	MPREC		48103033
17	J	CLOOP		3F2FEC
1A	ENDFIL	LOA	EOF	53016
1D	STA	BUFFER		F3016
20	LOA	#3		10003
23	STA	LENGTH		F3000
26	+TSUB	MPREC		48103033
2A	J	RETAOR		3E2003
2D	EOF	BYTE	C'EOF'	
30	RETAOR	RESW	3	
33	LENGTH	RESW	1	
36	BUFFER	RESW	ARNG	
3936	RPREC	CLEAR	X	0410
3938		CLEAR	A	0400
393A		CLEAR	S	0440
393C	+JLT	NARNG		75103000
3940	RLOOP	TO	INPUT	E12029
3943	REQ	RLOOP		312FFA
3946	RD	INPUT		082013
3949	COMP	A,S		A000
394B	REQ	EXIT		312000
394E	STCH	BUFFER,X		53C003
3951	TDOR	T		0850
3953	JLT	RLOOP		302F1A
3956	EXIT	STX	LENGTH	114000
3959	RSUB			4F0000
***** RSUB doesn't need an operand				
395C	INPUT	BYTE	X'F1'	
395D	MPREC	CLEAR	X	0410
395F	LOA	LENGTH		774000
3962	WLOOP	TO	OUTPUT	E12011
3965	REQ	WLOOP		312FFA
3968	LOCH	BUFFER,X		53C003
396B	WD	OUTPUT		0F2000
396E	TDOR	T		0850
3970	JLT	WLOOP		302F1F
3973	RSUB			4F0000
***** RSUB doesn't need an operand				
3976	OUTPUT	BYTE	X'05'	
3977	END	FIRST		

Object Program.txt - Notepad

```

HCOPY 000000001077
T0000001D1720D6920D48101036320262900003320074810105D3F2FEC32010
T00001D13F201610003F200D4810105D3E2003
T0010361D04100400044075101000E32019332FFAD02013A00433200857C003B850
T0010531D3B2FEA1340004F00000410774000E32011332FFA53C003DF20088850
T001070073B2FEF4F0000
E0000000

```