Term Project - SIC/XE Assembler
Phase (1)

The term project is to implement a (cross) assembler for (a subset of) SIC/XE assembler, written in **C/C++,** producing code for the absolute loader used in the SIC/XE programming assignments.
In phase 1 of the project, it is required to implement **Pass1** of the assembler. The output of this phase should be used as input for subsequent phases.

Specifications

1. The pass1 is to execute by entering
   pass1 <source-file-name>
2. The source file for the main program for this phase is to be named pass1.c
3. You should build a parser that is capable of handling source lines that are instructions, storage declaration, comments, and assembler directives (a directive that is not implemented should be ignored possibly with a warning)
   - For instructions, the parser is to be minimally 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. RSUB (ignoring any operand or perhaps issuing a warning)
     c. 3-byte PC-relative with symbolic operand to include immediate, indirect, and indexed addressing
     d. 3-byte absolute with non-symbolic operand to include immediate, indirect, and indexed addressing
     e. 4-byte absolute with symbolic or non-symbolic operand to include immediate, indirect, and indexed addressing.
   - The parser is to handle all storage directives (BYTE, WORD, RESW, and RESB), in addition to START and END directives.
   - Hexadecimal addresses that would begin with ‘A’ through ‘F’ must start with a leading ‘0’ to distinguish them from labels.
   - Instructions and assembler directives in the source program may be written using either uppercase or lowercase letters.
   - The source program to be assembled must be in fixed format as follows:
     1. bytes 1–8 label
     2. 9 blank
     3. 10–15 operation code
     4. 16–17 blank
     5. 18–35 operand
     6. 36–66 comment
   - If a source line contains “.” in the first byte, the entire line is treated as a comment

- A list of required instructions along with their op-codes are found in the appendix at the end of this statement, for ease of reference.

4. The output of this phase should contain (at least):
   1. The symbol table.
   2. The source program in a format similar to the listing file described in your text book except that the object code is not generated as shown below. A meaningful error message is printed below the line in which the error occurred.

Example input

```
TERMPROJ START 3A0
.THIS IS A COMMENT LINE
LBL1 BYTE C'ABCDEF'
LBL2 RESB 4
LBL2 RESW 1
TOP LDA ZERO
LDX #INDEX
```

Output

Line no.	Address	Label	Mnemonic Op-code	Operands	Comments
1	0003A0	TERMPROJ	START	3A0	
2	0003A0	.THIS IS A COMMENT LINE			
3	0003A0	LBL1	BYTE	C'ABCDEF'	
4	0003A6	LBL2	RESB	4	
5	0003AA	LBL2	RESW	1	
		**** Error: Symbol 'LBL2' already defined			
6	0003AD	TOP	LDA	ZERO	
7	0003B2		LDX	#INDEX	

Your project write-up should include:

1. Requirements specifications.
2. Main data structures.
3. Algorithms description.

Bonus

The input is a free-formatted assembly language program. In a free-formatted assembly program, statements are not restricted to begin at a given position in the line. Many consecutive white spaces or tabs should be treated as a single space. (You may use regular expressions)

Example input (Free formatted code) :

```
      TERMPROJ  START        3A0
     .THIS     IS    A COMMENT LINE
    LBL1          BYTE     C'ABCDEF'
   LBL2                RESB       4
  LBL2     RESW                    1
 TOP         LDA                 ZERO
LDX                               #INDEX
```

Notes:

- Assigned: Sunday, April 5th.
- Due date: Sunday, April 19th.
- You should work in groups of **4-5** members.
- For the bonus part, you can use external libraries for string parsing. (Hint: you can look into regular expressions)
- Begin as early as possible and ask questions early.
- All members should work together. There is a grade on distributing the load evenly.
- All member should understand all components in the project, not just the parts they implemented.
- Cheating will be severely penalized. Both copies will be graded zero. So, delivering a partially functional implementation is much better than delivering a copy.

Deliverables:

- Source Code
- Executable file for pass1 which runs by entering: “pass1 <source_file_name>”
- Report that contains:
  - Requirements specification.
  - Design
  - Main data structures
  - Algorithms description
  - Assumptions (if any)
  - Sample runs.
- You should submit the deliverables in a zipped file with the format: groupNumber_phase1.[rar/zip/...etc]. (for example: "1_phase1.rar") to csed.system.programming@gmail.com

Appendix:

Mnemonic	**Format**	**Opcode**
ADD m	3 / 4	18
ADDR r1,r2	2	90
CLEAR r1	2	B4
COMP m	3 / 4	28
COMPR r1,r2	2	A0
DIV m	3 / 4	24
DIVR r1,1r	2	9C
J m	3 / 4	3C
JEQ m	3 / 4	30

JGT m	3 / 4	34
JLT m	3 / 4	38
JSUB m	3 / 4	48
LDA m	3 / 4	00
LDB m	3 / 4	68
LDCH m	3 / 4	50
LDL m	3 / 4	08
LDS m	3 / 4	6C
LDT m	3 / 4	74
LDX m	3 / 4	04
MUL m	3 / 4	20
MULR r1,r2	2	98
RD m	3 / 4	D8
RMO r1,r2	2	AC
RSUB	3 / 4	4C
SHIFTL r1,n	2	A4
SHIFTR r1,n	2	A8
STA m	3 / 4	0C
STB m	3 / 4	78
STCH m	3 / 4	54
STL m	3 / 4	14
STS m	3 / 4	7C
STT m	3 / 4	84
STX m	3 / 4	10
SUB m	3 / 4	1C
SUBR r1,r2	2	94
TD m	3 / 4	E0
TIX m	3 / 4	2C
TIXR r1	2	B8
WD m	3 / 4	DC