**CSED 2014**

[Year]

**SIC/XE Assembler**

**Pass 1**

Quick introduction :

The required 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 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. You can work in groups of **3-4**

members.

Problem statement 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 declaratives, comments, and assembler directives (a directive that is not

implemented should be ignored possibly with a warning)

1. For instructions, the parser is to minimally be 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

2. The parser is to handle all storage directives (BYTE, WORD, RESW, and RESB).

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.

Assembler phase 1 design:

* Assembler is divided into different modules supporting different processes needs such as:

1. Literal table:

* The basic data structure needed to handle literal operands, for each literal used, this table contains the literal name, the operand value, and length, and the address assigned to the operand when it’s placed in the literal pool.

1. Symbol Table:

* The first pass stores in the symbol table label definitions, such as its name, its value and its type ( relative or absolute ), in case of a program relocation needed.

1. Operation code Table:

* A table in which each operand is stored, other than its name, its opcode is stored ( to be used in pass 2 while generating object code for the program, and the format of each opcode.

1. Directive pool :

* Stores each directive and some information about it, like if its format is 1, then no need for an operand, if its format is greater, then it needs an operand to be declared, it also contains the “START” and “END” directives which declares the start and the end of a program.

1. Pass1 :

* The main component of the assembler, which executes the pass1 of the program, it reads the file, and processes the pass1 on it, advancing addresses used in location counter, labels, inserts labels within their addresses to be able to generate object codes after the pass2.

K,

Bonus :

* Supporting Free formatted instructions
* Supporting Literals
* Evaluating Expressions
* EQU statement
* Org statement
* Ability to change labels’ comments’ and operands’ width (no of columns)

Handling Opcodes

Using a map<string, opcode> where opcode is a class used to carry