University of North Florida

School of Computing

Introduction to Software

Instructor: Anirban Ghosh

**Assembler in Java**

Team Members:

Cina, Kim – N01034237

Onybe, Uche – N01241024

Wessolossky, Alejandra – N01045827

November 30, 2017

**How to use it**

1. Have the SIC/XE code in the same folder where the Assembler in Java program is located.
2. Run the program
3. A .lst and .obj files will be created and stored in the same folder
   1. the listing file is the one having the .lst extension
   2. the object file is the one having the .obj extension. It contains a complete object program for your SIC/XE code.

**Data Structures**

* Source file: SIC/XE code provided by user
* OPTAB: hash-table that contains mnemonics and related information
* SYMTAB: hash-table that contains symbols
* Intermediate file: file used as input for pass 2
* .lst file: listing file
* .obj file: object program

**Restrictions**

1. EQU, USE, and CSECT directives are not implemented. You will get an error message in the .lst file indicating if one of those directives were found in your SIC/XE code.
2. Floating points, literals and division are not supported by the program. You will get an error message in the .lst file indicating if they were found in your SIC/XE code
3. HIO, LPS, SSL, STI, STSW, SVC, SIO, and TIO instruction are not supported by the program. You will get an error message in the .lst file indicating if they were found in your SIC/XE code

**\***The Assembler in Java program will continue to run even if one of the restriction is found, but it would be reported in the .lst file

**Implementation**

In our assembler, we implemented two hash-tables, OPTAB and SYMTAB. The OPTAB is a static table that stores mnemonics (used as the key) and their corresponding machine language. On the other hand, the SYMTAB table stores addresses assigned to labels. In order to get labels, mnemonics, addresses, and opcode, the file is read line by line. Each line is split based on the number of variables (length) and comments and whitespaces are ignored when using the splitter method.

In *Pass 1*, as the requirements mentioned, a variable LOCCTR is created to store addresses, and it is initialized to the value of the “START” variable. At the beginning addresses are increased by 3, but they might change later when comparing it to the OPTAB table. A “IF” statement is implemented in order to know if the OPTAB table contains any “WORD, “RESW”, and “RESB”. In case yes, addresses are updated according to them. During Pass 1, an intermediate file is also created; it stores everything that was read from the original file, and it is used as input for Pass 2.

In *Pass 2,* the intermediate file is used as input, and it is read one more time in order to obtain addresses which are gotten by looking up symbols that are used as operands in the SYMTAB table. During this pass, data values defined by BYTE, WORD, and RSEW are generated as well. Finally, the object code and assembly listing are written.