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

December 1, 2017

How to use it

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

Restrictions

1. EQU, USE, and CSECT directives are not implemented.
2. Floating points, literals and division are not supported by the program.
3. HIO, LPS, SSL, STI, STSW, SVC, SIO, and TIO instruction are not supported by the program.
4. The length of each text record is not displayed in the object program
5. The LOCCTR for text records is not implemented
6. 2's Complements bits are not converted exactly as they should be; it is off for 1 byte
7. Side comments implementation is not handled

The Assembler in Java program continues to run even if one of the restriction is found and an error message is displayed at the end of the .lst file.

Data Structures

- Source file: file that contains SIC/XE code and is provided by user.
- OPTAB: hash-table that contains mnemonics and related information.
- SYMTAB: hash-table that contains symbols.
- Intermediate file: file that contains information obtained from source file.
- .lst file: listing file.
- .obj file: object program.

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. The SYMTAB table stores addresses assigned to

labels. In order to get labels, mnemonics, addresses, and opcode, the file is read line by line. Then, each line is split based on the number of variables (length) and comments and whitespaces are ignored.

In *Pass 1*, the Assembler in Java program read line by line the entire source program and all information is collected. A variable LOCCTR is created to store addresses, and it is initialized to the value of the "START". Then, a "IF" statement is implemented in order to check if the OPTAB table contains any "WORD", "RESW", and "RESB". In case yes, addresses are calculated according to them. During this pass, an intermediate file is also created. This file stores everything that was read from the source file, and it is used as input for Pass 2.

In *Pass 2,* instructions are assembled using the SYMTAB table. During this pass, Assembler in Java reads line by line the intermediate file, and generates a machine code for each instruction. It is done in SIC format first, and then values for N, I, X, B, P, E are change and Program Counter or Base Relative calculations are implanted to produce the SIC/XE version of the OPCODE. At the same time, the ".lst" and ".obj" files are written.