Assignment description – Project 5

The project spans Chapter 6 of the book, about the implementation of the Assembler. It also builds on Chapter 4, where the Hack Assembly and Machine languages are defined. Please read these two chapters of the book thoroughly before you start your project.

We provide a large part of the Assembler in Python, and ask you to extend this implementation. Read and understand the provided code before you start to extend it. If you prefer, you can implement the entire Assembler yourself. We can accept solutions in Python, C, C++ or Java.

Objective The purpose of this project is to get the first experience with writing a compiler, by completing the implementation of the Assembler that reads as input a text file with extension .asm, containing a Hack Assembly program, and produces an output file with extension .hack, containing the translated Hack Machine Language code.

Contract Implement the functions in charge of completing the parsing with finding the *dest* field of the C-Instruction, and completing the code generation by translating the *dest* filed in Assembly to the appropriate bits for the Machine Language code. Compete the part of the Symbol Table generation by inserting the entries of the *Labels* along with their ROM addresses.

Resources The main tool in this assignment is the partially implemented Hack Assembler written in Python. Chapter 6 of the book contains detailed plan for the Assembler implementation and the API of all the main routines used in the Assembler. You can also find test programs for testing the translation of Assembly programs without and with symbols (in nand2tetris/projects/06). Of course you also need a programming environment for your selected programming language.

Background

The Assembler supplied with this assignment is documented in section 6.3 of the text book. It is strongly suggested to have a full understanding of this section before peeking into the code supplied with the assignment.

Details

As suggested in section 6.5, you can build the Assembler in two stages, first a symbol-less version, that can translate programs that do not contain labels and variables. Then a complete Assembler, that includes the symbol handling.

The following Python codes are provided:

- *hasm.py*: is the main file, controlling the two passes of the Assembly process;
- *hasmParser.py*: containing the routines for parsing the Assembly instructions;
- *hasmCode.py*: containing the routines to translate the Assembly mnemonics to Machine Language codes;
- *hasmSymbols.py*: with the routines needed for symbol handling.
- *hasmError.py:* for error handling.

You need to write 3 pieces of codes to complete the Assembler. Start with implementing the routines:

- *_ParseDest(self)* in hasmParser.py and
- *Dest(self, mnemonic)* in hasmCode.py.

With this the Assembler is able to translate codes that do not use symbols. After carefully testing this version, you can implement a part of:

- *Pass1(sourceFile)* in hasm.py,

that controls the handling of *Labels* in the symbol table. Note that you need to understand and use the available routines in *hasmSymbols.py* to complete the code.

Hints

Understand the process of translation suggested in the book and the supplied codes. Try to answer the following questions before you start:

- What is the role of the functions *Pass1* and *Pass2*? What does variable *address* represent?
- What is the result of the parsing? What does the Parser provide to the Code writer?
- How can a C-instruction look like? How do we know whether the optional *dest* and *jump* fields are present?
- How can a new *Label* symbol be recognized? What is the address of a *Label* symbol in Hack assembly?
- What is the result of the assembly process if you try to translate an Assembly program (without symbols) with the provided, not completed code?
- What is the result of the assembler process if you translate an Assembly code that includes labels, without completing *Pass1* in *hasm.py*?

Submission

Please submit the three files, *hasm.py*, *hasmParser.py* and *hasmCode.py*, which should include your implementation of the above named functions, as well as the other two files, *hasmSymbols.py* and *hasmError.py*.

Your project must be mailed to your group leader by the time given on the webpage. Make sure that the subject field states *EP1200-groupN*, if you are in seminar group N. All projects should be done individually.

One submission per student, in a zipped file named EP1200-seminarM-GroupN-firstname-familyname.zip including:

- The solutions (typically code).
- The list of the sub-projects you solved successfully and are able to present. Use the provided form.

Copying and other forms of plagiarism and cheating will be reported for disciplinary action!