Assembler

Prof. Naga Kandasamy
ECE Department, Drexel University

Develop an assembler that translates programs written in Hack assembly language into the binary code understood by the Hack computer. This problem is due November 15, 2023, by 11:59 pm. Please submit original work.

The name of the Hack assembly-language file is supplied to the assembler as a command-line argument. If you choose to implement the assembler in Python, usage is as follows:

```
> Python assembler.py some-program.asm
```

Your assembler must translate the `some-program.asm` file into machine code and store it in a text file called `some-program.hack`. Use the skeleton code provided in `assembler.py` as a starting point for your solution.

The `.asm` file consists of text lines, each representing either an instruction or a symbol declaration. Key specifications of the Hack assembly language are summarized below:

- *Instruction type*: an A-instruction or a C-instruction.
- (*Symbol*): This pseudo-command binds the Symbol to the memory location into which the next command in the program will be stored. It generates no machine code.
- *Constants and symbols*: Constants are non-negative, written in decimal notation. A user defined symbol can be any sequence of letters, digits, underscore (_), dot (.), dollar sign ($), and colon (:) that does not begin with a digit.
- *Comments*: Text beginning with two slashes (//) in a line is considered a comment and is ignored.
- *White space*: Space characters are ignored and so are empty lines.

All assembly mnemonics must be written in uppercase whereas user-defined labels and variable names are case sensitive. Use uppercase for labels and lowercase for variable names.

The Hack assembly program is allowed to use these predefined symbols:

Label	RAM address (in decimal)
`SP`	0
`LCL`	1
`ARG`	2
`THIS`	3
`THAT`	4
`R0 - R15`	0 - 15
`SCREEN`	16384
`KBD`	24576

Variables are mapped to consecutive memory locations as they are first encountered, starting at RAM address 16 (0x0010).

Testing Your Assembler

Test the correctness of the machine code generated by your assembler using the following test programs:

1. `Add.asm` adds constants 2 and 3, and stores the result in register `R0`.
2. `Max.asm` computes the maximum of values stored in `R0` and `R1`, and stores the result in `R2`.
3. `Rect.asm` draws a rectangle of dimensions 16 pixels wide and `R0` pixels high at the top left corner of the screen.
4. `mult.asm` multiplies two numbers stored in `R0` and `R1`, and stores the result in `R2`.
5. `series_sum.asm` calculates the sum of $1 + 2 + \ldots + 10$ and stores the result in `RAM[17]`.

Execute the `.hack` files generated by your assembler using the CPUEmulator and verify that the correct result is generated in each case.

Use the supplied Assembler tool (which produces the correct binary code) to convert each of the `.asm` files to the corresponding `.hack` file. Compare the output of your assembler to the correct output. You may compare the files using the TextComparer tool provided with the nand2tetris toolkit, or the 'diff' command if in UNIX, or using the 'FC' command if operating in Windows.

Submission Instructions

Submit via BBLearn, the source code for the assembler along with a README file that details how to build and execute it.