COSC 341/342 Programming Languages WINTER 2015
Python Project Part 1: Build a Symbol Table

Distributed 22 January 2015 Due: 5 February 2015

Assignment:

Write a Python program that will build a symbol table for an arbitrary, but correct, LC3 assembler language program.

The symbol table data structure is up to you.

You must provide functions to insert and to retrieve data from the symbol table. You must provide a function to pretty-print the symbol table (i.e., nicely formatted).

You are allowed to import any libraries that are available on the Web – cite the URL plus whatever is necessary for a complete citation.

Background information

The LC3 instruction set architecture is described in Appendix A of this link:
http://highered.mheducation.com/sites/0072467509/student_view0/appendices_a__b__c__d___e.html

A description of the LC3 assembler language (if you don't have the COSC 221 text) can be found at this link:
https://classes.soe.ucsc.edu/cmpe012/Fall06/notes/08_LC3_Assembly.pdf

Slides #21 and 22 describe the process of building the symbol table.

There are 2^{16} memory locations in an LC3 machine. Each instruction occupies 2 bytes (aka, one word). The LC3 machine is word-addressable. Thus, memory address 0x3000 contains 2 bytes (one instruction, or one integer, or one character). The next memory address 0x3001 contains 2 bytes. And so on.

Labels (symbols, identifiers) will reference a memory location. Labels are used to specify targets of a jump instruction (BR, JSR) , a load instruction (LD, LDI, LEA), a store instruction (ST, STI).

A label starts in the first position of a line of code. A label can be from 1 – 20 (alphanumeric, beginning with alphabetic) characters. There are two kinds of labels: (1) a target of a jump, (2) location of a variable for load or store.

A variable can be either (1) an integer, stored as 16 bit 2s complement, or (2) character, also 16 bits, though only the bottom 8 bits are used for ASCII encoding.

The following assembler directives important for computing the correct addresses for the symbols: .ORIG, .FILL, .BLKW, .STRINGZ, .END.
.FILL specifies the content of two bytes (one memory location)

.BLKW specifies the amount of space reserved (e.g., for an array). Each location specified will occupy two bytes.
.STRINGZ specifies a string of characters. This will reserve two bytes per character, plus one more byte for the string terminator 0x0000

Turn in:
- Hardcopy of source code that you wrote.
- Citation of any imported libraries,
- Sample run on multiply program (see below), ending with a listing of the symbol table.
- Sample run on program to be supplied on 2/3/2015 (it will be a long program), ending with listing of the symbol table.

Grade based on:

(1) Satisfies specs – runs on multiply program	30%
(2) Satisfies specs – runs on test program	50%
(3) Readability (commenting, variable names, white space, design of functions, …)	20%

```
                .ORIG  x3000
x3000           LD     R2, Zero
x3001           LD     R0, M0
x3002           LD     R1, M1
        ; begin multiply
x3003 Loop      BRz    Done
x3004           ADD    R2, R2, R0
x3005           ADD    R1, R1, #-1
x3006           BR     Loop
        ; end multiply
x3007 Done      ST     R2, Result
x3008           HALT
x3009 Result    .FILL  x0000
x300A Zero      .FILL  x0000
x300B M0        .FILL  x0007
x300C M1        .FILL  x0003
                .END
```

PostScript:
It looks like there is a partial interpreter for LC3 in Python here:
http://cs.brynmawr.edu/cs240/lc3.py
It won't be useful for this assignment, because ADD, ADDI, AND, ANDI, NOT instructions do not use symbols.

This write up on how to write an assembly language program from a high-level program may be helpful in the future – but not, I think for this project:
http://cs.brynmawr.edu/cs240/highlevelprogramming.html