Assignment 2

Deadline

11:55 PM on 5th February, 2018

Problem

Write an ***(Assembly) Code Generator*** from three-address code. The output of the assignment must be ***the assembly code for the respective machine architecture*** (that you have chosesn). This assignment requires you to:

- design your three address code and symbol table (as in-memory data-structures);
- load the intermediate language program from a text file into your three-address code data-structure and symbol table;
- implement a basic-block local register allocator;
- implement the translator to translate statements in three-address code to assembly instructions;
- set up the *data* regions to handle global data and constants;
- provide some (minimal) library support to allow for writing of useful programs.

Make sure that your assembly code can be *assembled* using the GNU assembler (invoked using *as* or simply *gcc*) to an executable binary (for target x86) or can *execute* on the SPIM simulator (for target MIPS).

Input

You can accept a textual description of your three-address code. The input can be in comma-separated form so that it is easy to parse and load into an in-memory array (as discussed in class). The each line in the input is a tuple <line-number, operation, destination-variable, source-variable(s)>. For example:

```
1, =, a, 2
2, =, b, 7
3, +, a, a, b
4, ifgoto, leq, a, 50, 2
5, call, foo
6, ret
7, label, foo
8, print, a
9, ret
```

It would stand for your representation for the following program:

```
a=2
L1:  b=7
a=a+b
if (a<=50) goto L1
foo(a)
return

foo():
  print a
  return
```

However, you are free to choose your own mnemonics/instructions for your three-address code. Do pay attention that the three-address code is the bridge between the source and machine code. So, keep your three addess code

near your source, or else, you will have a problem while attempting to construct your designed three-address code in the future assignments. You may assume:

1. All variables have the "integer" type.
2. The programs have only global variables. Even the temporaries can be allocated globally.
3. Function calls have no arguments and return only a single value (using a designated register).
4. No floating point operations are present.

Details

- Your implementation should read the source filename as its first command-line parameter; it should produce its output on the standard output as a listing of the assembly code.
- You will only be allowed **minor** modifications in this assignment submission in future (for integrating with the rest of the phases).
- You have to submit a zipped folder (name the folder "asgn2") with:
  - the source of the implementation (in a folder called "src" within "asgn2";
  - a Makefile to build the implementation (it should generate an executable called "codegen" in the folder "asgn2/bin";
  - a set of at least 5 test cases that you have used to check your implementation (in a folder "asgn2/test");
  - a README file with a brief description for building and running it.

  Binaries should NOT be part of the submission. Clean the folder of all object and executable files before submission.
- Note that all elements of your submission, like code quality and readability, quality of documentation (README file), quality of test-cases etc. carry marks.
- We will apply the following set of commands to build and run your implementation; make sure that your implemantation works correctly with these sequence of commands:
  - cd asgn2
  - make
  - bin/codegen test/test1.ir (to execute the first test-case file test1.ir; the listing of assembly code should be displayed on the standard output)

References

SPIM simulator. http://spimsimulator.sourceforge.net/