Compilers Fall 2015 Project 5

Due Wed 12/09/2015 at 8:30am.

Project Milestone 5: Code generator

Implement an x64 code generator for TACK. Your program should translate a TACK source file to x64 assembly. For this milestone, you can assume that the TACK source file does not contain any syntactic or semantic errors. The TACK language specification is here:

http://www.inf.usi.ch/faculty/soule/teaching/2015-fall/cc/tack-spec.pdf

And the following document introduces the subset of x64 that you need for our project:

http://www.inf.usi.ch/faculty/soule/teaching/2015-fall/cc/x64-intro.pdf

The following file contains the intrinsic functions written in C. In other words, this file provides the runtime that you will need to link your generated code against:

http://www.inf.usi.ch/faculty/soule/teaching/2015-fall/cc/pr5/x64runtime.c

See the x64-intro for an example for how to use `gcc` as an assembler. For example, assuming the driver for your compiler is the Java class `Main`, you would first run your compiler:

```
java -ea -cp .:/usr/local/lib/antlr-4.5.1-complete.jar Main hello.tack > hello.s
```

Second, you would run `gcc` as an assembler and linker:

```
gcc -m64 -masm=intel -o hello hello.s x64runtime.c
```

Third, you would run the generated binary program:

```
./hello
```

The output from that should be the `Hello, world!` message.

Please turn in the entire, self-contained code for your code generator, including a `README` with instructions for how to compile and run.

Hints and tips

1. Get started early.

2. It is easiest to do the code generation starting from an in-memory representation of the IR. For example, each instruction could be represented by one object, and each function could contain an array of instructions. The example solutions for pr4 create such a representation:
   http://www.inf.usi.ch/faculty/soule/teaching/2015-fall/cc/pr4-example-solutions.tar

3. Reuse code from the previous milestone (either your own code, or the example solutions, or a mixture of the two).

4. Start by getting a simple test working, then incrementally implement more cases. You can use tests from earlier project milestones, as well as write your own.

5. It is up to you whether you want to implement any sophisticated register allocation. Whatever you decide, it is strongly recommended that you start by implementing a "spill-all" or "no register allocation" code generator, as a fall-back in case you run out of time.
