Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.035, Spring 2016 Handout — Code Generation Project Friday, Oct 3

DUE: Friday, Mar 18, 11:59 pm

Code Generation involves producing correct x86-64 assembler code for all Decaf programs. The two following projects will involve code optimizations. For now, we are not interested in whether your generated code is efficient.

By the end of code generation, you should have a fully working Decaf compiler. You'll be able to write, compile, and execute real programs on a real machine!

Project Assignment

For Code Generation, your compiler will translate your high-level IR into a low-level IR. Your low-level IR will include structures that more closely match the machine instructions of a modern architecture. Your compiler will then translate your low-level IR into x86-64 assembly code to be run on an AMD Opteron machine. You should target the subset of the x86-64 ISA defined in the *x86-64 Architecture Guide*, which is an appendix to this document.

Your generated code must include all runtime checks listed in the Decaf language specification. Additional checks such as integer overflow are not required.

The two final assignments, *Dataflow Analysis* and *Optimization*, will focus on improving the efficiency of the target code generated by your compiler. For this assignment, you are not expected to produce great code. In fact, even horrendous code is acceptable. When considering tradeoffs, always choose simplicity of implementation over performance.

You are not constrained as to how you go about generating your final assembly code listing. However, we suggest that you follow the general approach presented in lecture.

You will have a number of opportunities to do some creative design work for the code optimization projects. For this assignment, you should focus your creative energies on designing your low-level IR, familiarizing yourself with our target ISA, your machine-code representations of the run-time structures, and generating correct assembly code. Do not try to produce an improved register allocation scheme; you will be addressing these issues later.

Compiling and Libraries

Your compiler will create a `.s` file, which can be compiled to create an executable file. You can use `gcc` to compile your assembly code.

Decaf does not have any input/output functions. Part of the assignment is to implement the standard x86-64 calling convention for `callout` statements, so that you can interface with the outside world. Any function that is called using `callout` needs to be linked in separately. `gcc`

will link against any standard libraries (you may need to use the `-l` argument for `gcc` to link some libraries). The testing files provided to you link against the standard C library. If you want to use functions that are not easy to use in Decaf (handle pointers, etc), you are welcome to write your own library calls in `C`, compile them to object files (using `gcc -c`) and then link them in by hand when compiling your assembly.

What to Hand In

Follow the directions given in project overview handout when writing up your project. Your design documentation should include a description of how your IR and code generation are organized, as well as a discussion of your design for generating code. Submit your design document in pdf or plaintext format in the doc directory.

Submitted tarballs should have the following structure:

```
GROUPNAME-codegen.tar.gz
|
‘-- GROUPNAME-codegen
    |
    |-- AUTHORS            (list of students in your group, one per line)
    |
    |-- src
    |   |
    |   ...                (full source code, can build by running ‘./build.sh‘)
    |
    |-- doc
    |   |
    |   ...                (write-up, described in project overview handout)
    |
    |-- build.sh
    |-- run.sh
    |-- (etc)
```

You should be able to run your compiler from the command line with:

```
./run.sh --target=assembly <filename>
```

Your compiler should then write a x86-64 assembly listing to standard out or the output file specified by `-o`.

Nothing should be written to standard error for a syntactically and semantically correct program (or to standard out, if an output file is specified) unless the `--debug` flag is present. If the `--debug` flag is present, your compiler should still run and produce the same resulting assembly listing. Any debugging output is left to your own discretion.

Project Submission

You should submit the project using the course's submission web site by the due date. Place your design documentation in the directory `docs/codegen` in your git repository.

We will run your compilers on the test cases in the tests repository and on a set of hidden tests. We strongly recommend you write additional tests of your own, because the provided tests are nowhere near comprehensive.

The design document should describe your design and implementation for the code generation stage. This description will include the design of your low-level IR and a discussion of your code generation scheme and implementation. This document also counts towards the project grade.

Related Handouts

The *X86-64 Architecture Guide*, provided as a supplement to this handout on the course reference materials page, presents a not-so-gentle introduction to the x86-64 ISA. It presents a subset of the x86-64 architecture that should be sufficient for the purposes of this project. You should read this handout before you start to write the code that traverses your IR and generates x86-64 instructions.