Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.035, Fall 2005 Handout 9 — Code Generation Project Monday, October 10

DUE: Tuesday, October 31

Code Generation involves producing correct x86-64 assembler code for all Decaf programs. The next two project assignments will involve code optimizations. For now, we're 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!

Project Assignment

For CG (Code Generation), your compiler will translate your IR tree into x86-64 assembly code to be run under Linux on an AMD64-based computer. For a given input file containing a Decaf program, your compiler must generate an assembly language listing (⟨*filename*⟩`.s`).

Your code must include instructions to perform the runtime checks listed in Handout 6. Additional checks such as integer overflow are not required. You may have implemented runtime checks for IR generation. If not, it needs to be done by the end of CG.

The two later assignments, *Dataflow optimizations* and *Instruction optimizations*, will focus on improving the efficiency of the target code generated by your compiler. For this assignment, you will use a "two-register" form for all arithmetic and logical operations. This means that you will store all variables on the stack, and will use registers only as temporary storage.

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 first assignment, you should focus your creative energies on your machine-code representations of the run-time structures and (possibly) of the procedure call / return sequences presented in lecture. Do not try to produce an improved register allocation scheme; you will be addressing these issues later.

System Usage

You will develop your compiler on Athena or your own computer, perhaps using your group locker as a shared repository. Your group also has an account on the x86-64 test machine.

Once you generate an assembly output file, say @output.s@, you need to copy it to your group's space on the , assemble and link it using @gcc@, and run it:

```
athena% scp output.s <test machine>:
athena% ssh <test machine>
<test machine>'s password:
-bash-3.00$ ls
output.s
-bash-3.00$ gcc output.s <library directory>.s -o output
-bash-3.00$ ./output
Hello, world!
-bash-3.00$ logout
Connection to <test machine> closed.
athena%
```

What to Hand In

As always, follow the directions given in Handout 3 when writing the hardcopy documentation for your project. The electronic portion of the hand-in procedure should also be familiar: Provide a gzipped tar file named `leNN-codegen.tar.gz` in your group locker, where `NN` is your group number. This file should contain all relevant source code and a `Makefile`. Also provide a Java archive named `leNN-codegen.jar`.

Unpacking the tar file and running `make` should produce the same Java archive. With the `CLASSPATH` set appropriately, you should be able to run your compiler from the command line with one of the following:

```
java Compiler <filename>
java Compiler <filename> -target assembly
```

Your compiler should write an x86-64 assembly listing to a file of the same name with a `.s` extension.

Nothing should be written to standard out or standard error for a syntactically and semantically correct program unless the `-debug` flag is present. If the `-debug` flag is present, your compiler should still run and produce the same resulting assembly listing.

Test Cases

We will run your compilers on the revealed test cases on the course server,
and on a set of hidden tests.

Related Handouts

The *X86-64 Architecture Guide*, provided as a supplement to this handout, describes the restricted view that we will take of the x86-64 architecture for the purposes of this project. You must read this handout before you start to write the code that traverses your IR and generates x86-64 instructions.