Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.035, Fall 2000	Handout 16 Segment IV: Code Generation Project Assignment	Wednesday, October 11

DUE: Thursday, November 2

This project assignment, **Code Generation**, involves generating correct MIPS assembler code for all Decaf/Espresso programs. The next two project assignments will involve code optimizations, so for now, as with the previous assignment, we're not interested in whether your generated code is very efficient.

By the end of code generation, you should have a fully working Decaf/Espresso compiler and be able to write and execute real programs you write in Decaf/Espresso!

Project Assignment

For CG (Code Generation) your compiler will translate your IR tree into MIPS R2000 assembly code (Handout 18) to be run on the SPIM simulator (Handout 19). You should target the virtual machine described in the SPIM and MIPS handouts. If you want to generate code for the real MIPS processor, you may do that too. In your documentation, specify which target you choose, and why you choose it. For a given input file containing a Decaf/Espresso program, your compiler must generate an assembly language listing ($\langle filename \rangle$`.s`).

Your code must include instructions to perform the runtime checks listed in Handout 6/Handout 7(additional runtime checks such as integer overflow are not required). You should have already implemented this for IR generation, but if not then it needs to be done by the end of CG.

The two later assignments, *Data-flow optimizations* and *Instruction optimizations*, will focus on improving the efficiency of the target code generated by your compiler. For this assignment, therefore, you are not expected to produce great code. (Even horrendous code is acceptable. In any 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 use the supplied ASM toolkit (described in Handout 17), and that you follow the general approach being presented in lecture. We strongly recommend using the ASM toolkit; it will aid you in running programs natively, and greatly simplifies writing out the final assembly code.

You will have a number of opportunities to do some creative design work for the code generation 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 sorts of issues later.

What to Hand In

Follow the directions given in Handout 4 when writing up your project. Your design documentation should include a description of how your IR and symbol data table structures are organized, as well as discussion of your design for performing the semantic checks.

The electronic copy portion of the hand-in procedure is similar to that of the semantic analysis segment. 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`. Additionally, you should provide a Java archive, produced with the `jar` tool, named `leNN-codegen.jar` in the same directory.

Unpacking the tar file and running `make` should produce the same Java archive. With the `CLASSPATH` set to `leNN-codegen.jar:/mit/6.035/classes`, you should be able to run your compiler from the command line with:

`java Compiler` $\langle filename \rangle$ or
`java Compiler` $\langle filename \rangle$ `-target assembly`

Your compiler should write a MIPS assebly listing to a file of the same name with a `.s` extention.

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 (in `/mit/6.035/tests/codegen`) and hidden test cases.

Related Handouts

There are three other handouts relevant to CG.

1. **ASM Toolkit Documentation** - Handout 17 — describes how to use the ASM toolkit abstraction to output MIPS assembly code.
2. **MIPS R2000 Code** - Handout 18 — describes the 6.035 Instruction Subset of the MIPS R2000 assembly language. You must read this handout before you start to write the code that traverses your IR and generates R2000 instructions.
3. **SPIM Interface** - Handout 19 — describes how to use SPIM to execute the assembly files your compiler will produce.