Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.035, Fall 2005 Handout 12 – Low-level Optimizations Monday, November 28

DUE: Monday, December 12

In the final phase of the project, you will implement some instruction-level optimizations. You are *required* to implement register allocation. *Any remaining optimizations are optional.*

- **Register Allocation** – Your compiler must implement a graph coloring based register allocator. An "always or never" allocator is not adequate, but we do not require you to implement web splitting and combining. See Chapter 16 of the Whale book and §9.7 of the Dragon book.

  Your register allocator should take advantage of the full set of x86 general purpose registers, not just r10 and r11. It should also respect the caller-save and callee-save properties of this register.

  In addition to allocating registers to program variables and temporaries, you must use them to pass function arguments as specified in the full calling convention.

- **List Scheduling** – Instruction scheduling minimizes pipeline stalls for long latency operations such as loads, multiplies, and divides. See Chapter 17 of Whale book.

- **Instruction Selection** – So far, we have been using a very restricted subset of the x86-64's instructions. As a peephole optimization, you might replace a sequence of old instruction with a single new one taht does the same thing efficiently. For instance, it is possible to push and pop values directly from memory, so no intermediate temporary register is needed.

See the full AMD64 reference (linked from the abridged version) for details regarding the architecture. To speed-test your code on `chocura`, use the `-pg` option to gcc while compiling in order to generalte profiling information, and use `gprof` to do the actual profiling.

What to Hand In

Your compiler must provide the following command line flags:

- `-opt regalloc`: turns on register allocation
- `-opt instsel`: turn on instruction selection peephole optimizations (optional)
- `-opt sched`: turns on list scheduling (optional)

As in the last assignment, you must provide a `-opt all` flag to turn on all optimizations. If you choose to implement list scheduling, you will need to think carefully about the order in which your optimizations are performed. You may even wish to perform some optimizations twice.

As before, your generated code must perform the runtime checks listed in the language specification. These may be optimized (or removed in some cases) as long as they report a runtime error whenever the unoptimized program reports an error.

Follow the instructions in Handout 3 on what to turn in. For the electronic portion, you should submit files named `leNN-instruction.jar` and `leNN-instruction.tar.gz`. Your written documentation should present a discussion of how you determined the order in which your optimizations are performed. Give clear examples of MIPS excerpts showing what your optimizations do, and describe any hacks or solutions to tricky problems.

Test Cases

No new test cases have been provided for this segment. However, you should make sure that any valid program continues to run successfully. We will test your compiler on a set of hidden test cases. Grades will be given based on the effectiveness and correctness of your optimizations.