15-411 Compiler Design, Fall 2014
Lab 5

Instructor: Frank Pfenning
TAs: Flávio Cruz, Tae Gyun Kim, Rokhini Prabhu, Max Serrano

Compilers due 11:59pm, Thursday, November 13, 2014
Papers due 11:59pm, Tuesday, November 18, 2014

1 Introduction

The goal of the lab is to implement some optimization for the language L4, which remains unchanged from Lab 4. This includes an *unsafe* mode in which your compiler may assume that no exception will be raised during the execution of the program (except due to assert). This affects operations such as integer division, arithmetic shift, array access, and pointer dereference.

2 Preview of Deliverables

In this lab, you are not required to hand in any test programs, since there is no change in language specification. Instead, we will be testing your compilers on the test suites from Labs 1–4. We will also test the efficiency of your generated code on benchmark tests created by the course staff. Since you have a choice between multiple different optimizations, the benchmarks are designed as realistic programs rather than to verify that any particular optimization is taking place.

You are required to turn in a complete working compiler that translates L4 source program into correct target programs in x86-64. In addition, you have to submit a PDF file which describes and evaluates the optimizations that you implemented.

3 Compilation to Unsafe Code

The `--unsafe` flag to your compiler allows it to assume that no exceptions will be raised during the execution of the program except ones due to `assert`. This means you can eliminate some checks from the code that you generate. You are *not* required to eliminate all checks, but it will make your compiled code slower if you do not take advantage of this opportunity at least to some extent.

Note that this does not in any way constitute an endorsement of unsafe compilation practices. It will, however, give you a more level playing field to compare the efficiency of your generated code against gcc and others in the class.

4 Optimizations

Beside unsafe mode, you are required to implement and evaluate at least four optimizations from the list below.

1. **Instruction selection.** You may optimize instruction selection to generate more compact or faster code. This includes generating good code for conditionals (e.g., avoiding `set` instructions), loops (e.g., enabling good branch prediction or aligning jump targets), and other improvements on your code.
2. **Constant propagation and folding.** Implement constant propagation together with constant folding and eliminating constant conditional branches.
3. **Dead code elimination.** Implement dead code elimination using the analysis described in the lecture notes.
4. **Eliminating register moves.** Explore techniques for eliminating register moves such as improved register allocation, copy propagation, register coalescing, and peephole optimization. We suggest coalescing registers in a single pass after register allocation as suggested by Pereira and Palsberg and the notes to Lecture 3.
5. **Common subexpression elimination.** Implement common subexpression elimination, with or without type-based alias analysis to avoid redundant loads from memory.
6. **Loop optimizations.** Hoisting invariant computations out of loops and strength reduction enabled by basic and derived induction variables are good targets for optimizations. Another important loop optimization is loop unrolling.
7. **Tail call optimization.** While not discussed in lecture, the tail call optimization was covered in Assignment 3.
8. **Inlining.** While not discussed in lecture, inlining was also covered in Assignment 3. For inlining, it will be important to develop a good heuristic for when to perform it.
9. **Other optimizations.** Feel free to add other optimizations as you see fit, although we strongly recommend first completing basic ones before you go for more advanced ones.

If you have already implemented any of the optimizations, you may revisit and describe them, empirically evaluate their impact, and perhaps improve them further.

Your compiler must take a new option, `-O`n (dash capital-O), where `-O0` means no optimizations, `-O1` performs some optimizations, and `-O2` performs the most aggressive optimizations. Part of our evaluation may be based on the performance improvements you achieve. At least `-O0` and `-O2` should be different. Keeping intermediate (and incompletely optimized) versions of your compiler functional will be important so you can reliably assess the effects of your optimizations.

5 Regression Testing

As you are implementing optimizations, it is extremely important to carry out regression testing to make sure your compiler remains correct. We will call it with the `--safe` and `--unsafe` flags at various levels of optimization to ascertain its continued correctness.

6 Deliverables and Deadlines

For this project, you are required to hand in a complete working compiler for L4 that produces correct target programs written in Intel x86-64 assembly language, and a description and assessment of your optimizations. The compiler must accept the flags `--unsafe`, `--safe` (default), `-O`n with default $n = 2$. When we grade your work, we will use the `gcc` compiler to assemble and link the code you generate into executables using the provided runtime environment on the lab machines.

Note that this time we will not just call the `_c0_main` function in the assembly file you generate, but other, internal functions in order to obtain cycle counts that are as precise as possible. So it is critical that your code follow the standard calling conventions and function naming conventions from Labs 1–4.

Compiler

The sources for your compiler should be handed in via Autolab as usual, and must contain documentation that is up to date. Your last submission will also be tested for correctness. Remember that optimizations are required to preserve memory safety. You may use any remaining late days for the compiler.

Compilers are due **11:59pm on Thursday, Nov 13, 2014**.

Project Report

The project report should be a PDF file of approximately 3–5 pages, and should also be handed in on Autolab. Your report should describe the effect of `--unsafe` as well as your optimizations and assess how well they worked in improving the code, over individual tests and the benchmark suite.

If you use algorithms that have not been covered in class, cite any relevant sources, and briefly describe how they work. If the algorithms have been covered in class, cite the appropriate lecture notes, and do not waste any space on describing the optimizations. The interactions of the optimizations with each other and the effect of optimizations on the produced code should be given adequate treatment.

Project reports are due on **11:59pm on Tuesday, Nov 18, 2014**.

You may not use any late days for the paper. This means if you hand in the compiler two days late, then the paper will be due on the same day.