1 Optimization

Compile primes.cool with and without optimization and see what difference it makes, using the following instructions:

1. Add a line in the loop in Main to print statistics when i reaches 1000. Compile (without optimization) and run the code. Record the data. (The total number of instructions should be around $3.5 \times 10^6$)

2. Now compile with optimization (-O), and do the whole process again. How many fewer instructions are executed? Give the drop overall and also individual for branches, loads and stores. How many more primes are found? (The answer may be surprising.) Explain the results! (You have the information you need to explain them.)

3. Now change the condition from
   
   prime * (n / prime) == n

   to

   (n / prime) * prime == n

   and repeat steps 1 and 2.

4. Explain the results; did the change make any difference in unoptimized code? In optimized code? Why? Please explain both the difference in number of instructions executed and in number of primes produced.

5. Now change the condition one more time to
   
   n <= (n / prime) * prime

   and repeat steps 1 and 2.

6. Again, explain the results (as in step 4).

7. As a programmer, what do these results make you think about optimization?

2 Spilling

The code generated by Figure 14.6 (3rd ed.) (Fig. 14.5 in 2nd and Fig. 9.8 in the 1st edition) is very poor. In particular it may spill the same register over and over even when the saved value is not changing. A better way to do spilling is to use the “maximum register needed” attribute you defined in your attribute grammar, to generate code that spills an
intermediate result at the point it is defined, if we know we’re going to need the register for a subcomputation. In other words, rather than spilling a register only when we need it for a later computation, the code should be changed to spill a register as soon as it has been assigned a value, if we know that it will have to be spilled. That way, the intermediate value will be spilled and restored only once.

This improvement can be accomplished through a few changes in the macro handle_op. Remember: we are changing the place where spills happen! Read the last paragraph again if you didn’t catch that.

(a) Assuming the number of available registers $k$ is 4, give an expression that requires 6 registers and will cause register 1 to be spilled multiple times between its original definition and eventual use.

(b) Indicate in the code which temporary values will be spilled, perhaps multiple times. Draw the AST and circle the binary operators whose left operands will be spilled while evaluating the right operand. (This is not the operator for which the code currently includes a spill!)

(c) Figure out how to determine which expressions will be spilled “ahead of time,” that is, at the point the temporary is originally defined. Change the macro handle_op and explain what you are doing. You may assume the existence of attribute max_reg. Think clearly; don’t just push symbols around. You must understand what you are doing!

3 Testing

Write a test case test8.cool that tests the optimized code generator. Think about situations that might be done wrong. What are likely bugs? Then write code that will cause such bugs to be exposed. Make sure the code executes and produces results that depend on correct execution.

CS 754: Homework #7
Due: 2020/5/6

Instead of the above assignment, investigate the optimization steps (a)-(j) from our class example. Don’t assume that the optimization consists of changing the assembly code—it probably will consist of being smarter in generating code in the first place.

For each of the optimizations added (b) through (j):

• Briefly say how this optimization could be accomplished.

• Cite at least one academic paper that discusses one of the issues raised in the optimization (either how to determine that the optimization is possible or how to carry out the optimization). Cite a different paper for each step. If the step is too simple to have a paper, find a paper on something more general or advanced or broad.