THE UNIVERSITY OF ARIZONA
DEPARTMENT OF COMPUTER SCIENCE

CSc 553 : Principles of Compilation

Programming Assignment 3 (Register Allocation)

Start: Fri Oct 27, 2023
Due: 11:59 PM, Fri Nov 10, 2023

1. General

This assignment involves the improvement of the final code generated by your compiler via global (function-level) register allocation, as specified below. (The text about reaching definitions is crossed out because this analysis is not necessary due to the simplification mentioned in Section 2 of having a single live range for each variable.)

(a) If the command-line option `-Oregalloc` is specified, for each function in the input program, your compiler should do the following:

- Construct a control flow graph for the function.
- Use intra-procedural liveness analysis (implemented in Assignment 2) to construct a register interference graph for each scalar local variable of the function.
- Use the interference graph to perform register allocation for scalar local variables. You can, but are not required to, put globals or array elements in registers.

(b) You should test the effectiveness of your optimization by testing five C-- programs of your choice compiled with and without optimization. Use 'spim -keepstats' to measure the number of instructions executed, and report the results in a file EVALUATION.txt that you upload along with your code.

2. Optimizations

You are to use the information obtained via liveness analysis and reaching definitions to perform register allocation within functions.

Acceptable simplifications

- It is OK to perform register allocation for just the scalar local variables of functions, i.e., map all globals and arrays to memory.
- For the purposes of this assignment, it is OK to have a single live range for each variable. This means that each vertex of the interference graph is a variable, and there is an edge between any pair of variables that are simultaneously live. This simplification also means that it is not necessary to implement reaching definitions in order to construct live ranges.

3. Execution behavior

Your program will be called **compile**. It will take three optional command-line flags, `-Olocal`, `-Oglobal`, and `-Oregalloc`, that will control code optimization. It will take its input from **stdin** and generate all output on **stdout**.

Note: Register allocation should generate correct code both with and without global and local optimizations from the previous assignment. In other words, all of the following combinations of command-line options should generate code that has the same behavior as the original unoptimized code (the order in which they are specified should not matter):

- `-Oregalloc`
- `-Olocal -Oregalloc`
- `-Oglobal -Oregalloc`
- `-Olocal -Oglobal -Oregalloc`

4. Evaluating your optimizations

In addition to the source code for your compiler, you should additionally submit a file EVALUATION.txt that describes your experiments with your optimizations for at least five programs of your choice of reasonable size. At a minimum, this should contain the following:

1. The *cost* and *benefit* of register allocation. This is similar to the cost/benefit analysis in Assignment 2.

   Cost: In assignment 2, some students used worst-case asymptotic analysis (big-O) to analyze the cost of an analysis. One drawback with this approach is that it does not give a concrete idea of just how much time the analysis and optimization take. Please measure the actual execution time cost, e.g., using something like the `time` command.

   Benefit: Use instruction counts as in Assignment 2.

2. Experiments on the impact of register allocation under different circumstances:
   - How well does register allocation work depending on whether the local and/or global optimizations from Assignment 2 are enabled?
   - What is the impact of the different spill strategies on the cost and benefit of register allocation? I.e., think of some other criteria for choosing which node to spill instead of Chaitin's algorithm of minimizing cost/degree and see how they do.

Note: For Assignment 2, some students carried out their experiments using the programs used in GradeScope for evaluating your optimizations. This is legal, but it misses the point of these experiments: those programs are constructed to check that optimizations are being done and have very artificial code structure as a result; no programmer would write code like that. So it isn't clear that the results of experiments on such programs help us get a better understanding of the cost and benefit of these optimizations. If you can, it would be better to try and write programs that are not so artificial.

5. Grading

Your optimization should be *correct*, i.e., it must not change the behavior of any program; and *effective*, i.e., for programs that contain code instances where the optimization is applicable, it should in fact optimize those code instances. These will be evaluated as for Assignment 2:

- As with Assignment 2, it is expected that your compiler will generate correct code for all of the test inputs used for Assignment 1 (they are available on lectura under the directory **/home/cs553/fall23/TestCases/all-codegen/**). Generating correct code for these programs will not, in itself, earn you points; however you will lose points if your compiler fails any of the correctness tests.

- Assuming that your compiler passes the correctness requirement, I will evaluate your implementation of register allocation using a selection of test inputs constructed specifically to demonstrate the impact of register allocation. Points will be awarded based on whether these optimizations are being carried out as expected. The autograder checks the following for each o the following: no other optimizations; local optimizations only; and local+global optimizations.

    - [Baseline] The total no. of instructions executed should be at most 120% of that for the reference implementation (similar to assignment 2).

    - [Instructions executed]
      (1) The total no. of instructions executed with register allocation enabled should be less than that without register allocation.
      (2) The total no. of load instructions executed with register allocation enabled should be less than that without register allocation.
      (3) The total no. of store instructions executed with register allocation enabled should be less than that without register allocation.

    - [Instructions eliminated: Loads] Either:
        - the no. of loads eliminated with register allocation enabled is at least 80% of either (a) the no. of loads eliminated by the reference implementation; or (b) the no. of loads obtained using your compiler without register allocation;

      OR
        - the no. of loads executed with register allocation enabled is not more than the no. of loads executed by the reference implementation.

    - [Instructions eliminated: Stores] Either:
        - the no. of store eliminated with register allocation enabled is at least 80% of either (a) the no. of stores eliminated by the reference implementation; or (b) the no. of stres obtained using your compiler without register allocation;

      OR
        - the no. of stores executed with register allocation enabled is not more than the no. of stores executed by the reference implementation.

- Finally, your evaluation of your optimizations, as given in your EVALUATION.txt file, will count for 10% of the grade for this assignment.

6. Submitting your work

Submit your work in GradeScope in the submission area created for this assignment. You should submit the following files:

- All files needed to build an executable of your compiler from scratch;
- a Makefile that provides (at least) the following targets:

  make clean:
  Deletes any object files (*.o) as well as the file 'compile'

  make compile:
  Compiles all the files from scratch and creates an executable file named 'compile'; and
- a file EVALUATION.txt evaluating your optimizations as discussed in Section 4.

7. Gotchas to watch out for

7.1. Implicit type conversion

Register allocation may cause problems in code that does implicit type conversion for reasons similar to those for Assignment 2. As in Assignment 2, it is OK for this project to not perform the optimization (in this case, register allocation) in this situation.

7.2. Register allocation and formal parameters

Since we are using a parameter-passing convention where function arguments are passed on the stack, if a formal parameter is allocated to a register it will need additional code to initialize that register from the appropriate stack location before it is used.

It is OK for this project to not place formal parameters in registers.