

In this assignment you will implement different register allocators and compare the generated MIPS code in terms of the number of cycles required for execution. This assignment should be done with groups of two, but can be done individually if necessary. Groups will implement four variations of register allocation and individuals will implement two.

1 Motivation

Register allocation is arguably the most important program optimization in the compiler. The MiniJava compiler version being provided spills all variables to memory. The implications of spilling all the variables is that a MIPS template instruction like

```
"add    'd0, 's0, 's1"
```

will be expanded into four MIPS template instructions using the load-load-compute-store concept

```
"lw     'd0, -16( 's0)", uses = [ t60 --> "%fp" ], defs = [ t77 --> "%t0" ]  
"lw     'd0, -20( 's0)", uses = [ t60 --> "%fp" ], defs = [ t78 --> "%t1" ]  
"add    'd0, 's0, 's1,  uses = [ t77 --> "%t0" , t78 --> "%t1" ],  
                           defs = [ t79 --> "%t2" ]  
"sw     's0, -24( 's1)", uses = [ t79 --> "%t2", t60 --> "%fp" ].
```

The code shown above assumes that the temporaries for the original add instruction's s0 and s1 slots are put into memory locations -16(%fp) and -20(%fp) and the temporaries for the original d0 slot is stored at memory location -24(%fp). The temporaries used above are just made up to illustrate what happens. In the code, the temporary for the frame pointer is provided by the MipsFrame on an FP() call. The MipsFrame tempReg() method provides temporary registers for use when generating load-load-compute-store code.

Due to the major code expansion that occurs when spilling all temporaries, register allocation can reduce the number of cycles executed by up to 75%!

2 The Assignment

Your assignment is to design implement four different versions of register allocation, compare the performance of the generated MIPS code amongst the four versions and the provided MiniJava compiler, and report on your design decisions, testing strategies, and performance results. Of the four versions of register allocation at least one of the versions must be based on graph coloring techniques and all four versions must be able to handle spilling. You will design each of the four version by selecting amongst the following options:

- Implement register allocation for the IR Tree data structure using Algorithm 11.11 in the Tiger book.
- Use Algorithm 11.10 in the Tiger book before performing graph coloring.
- Possible spilling heuristics: random spill choice, the temporary with the fewest uses and defs, some combination of the uses and defs and degree in the interference graph, other possibilities that you design
- Type of spilling: optimistic or pessimistic
- Coalescing options: no coalescing, George, Briggs, other possibilities that you design
- Other options that you design

It should be possible to select one of your four versions on the command line. For example,

```
% java -jar MiniJavaCompiler.jar -v1 file.java
% java -jar MiniJavaCompiler.jar -v2 file.java
...
```

Your group will need to create at least two test input programs. To check correctness, compare the output of the MIPS code running with SPIM to the output of the same program run with the JVM. Design your test cases so that they will break an incorrect implementation of register allocation. Feel free to share such test cases with other groups in the class, but each group must submit their own two test cases. Individuals not working with a partner must also generate two test cases. Also, performance in terms of dynamic cycles executed should be better when using register allocation than the provided MiniJava compiler.

Your group will also need to create two benchmark input programs. The benchmark programs should be designed to show off one or more of your register allocation implementations. Individuals not working with a partner must also generate two benchmarks. The benchmark programs from the whole class will be run through each submitted compiler using the version of register allocation that you predict will perform the best overall. We will compare the “best” register allocator from

each group in class. There are things you can do that are not described here that will result in better code than what is currently generated. If you discover these and decide to do them, they can be considered as “other options that you design”.

Graph the results of your four versions of register allocation and the provided MiniJava compiler for both of your benchmark programs. Explain the results.

3 Getting Started

1. The MiniJava compiler was sent to the class mailing list. You will be adding functionality to this provided compiler. The tree-based register allocation schemes will operate on the linked list of statement Trees generated with the following statement in Main.java:

```
LinkedList<Stm> traced = (new TraceSchedule(b)).stms;
```

The graph-coloring based register allocators can be called in Codegen.java where currently all registers are spilled:

```
// spill all the registers
retlist = frame.spillAll(retlist);
```

Alternate organizations of the code in the compiler are welcome.

2. Read chapters 10 and 11 in the Tiger book. Some of the code discussed in those chapters is available at <http://www.cambridge.org/resources/052182060X/>.
3. Work out some examples by hand before jumping into the implementation of any one version.
4. I have installed a version of SPIM that counts cycles on the CS machines.

```
% /s/bach/b/class/cs553/spim-7.2.1-keepstats/spim -keepstats file.s
```

4 Your Report

The report is an essential part of your completed assignment. Use it to describe your reasoning behind each of your register allocation designs, assumptions, difficulties, insights, and results. Organize and present your document as if it were the only basis for your assignment’s grade. The format of your writeup is up to you, but it should minimally include the following:

- Introduce the main goals of the project and in a couple of sentences summarize what you have accomplished.

- Describe the options that you used in each of your register allocation implementations. Motivate your selection of the various options.
- Present and explain graphs that exhibit the performance difference between MIPS programs generated by the provided MiniJava compiler and the four versions of register allocation in your extended MiniJava compiler.
- How did you test your register allocation implementations?
- What problems did you encounter while developing your program? If you knew someone who was just about to start work on this assignment, what advice would you give them?
- What, if any, outside sources did you use (e.g., articles, books, other students)? This is particularly important. It's OK to look at books and articles and speak with your professor and fellow students (although sharing code and working together is strictly forbidden), but as with any scientific document, you should always cite your references and collaborations. You can either cite collaborations in footnotes or in a separate Acknowledgment section.

There's no exact number of pages you should write, but if you've got between two and four then you're in the right ballpark.

5 Hints for doing the assignment

I can not emphasize the importance of your report enough. It is entirely possible that I will grade your whole project based solely on your report.

You should start this assignment early!! You have three weeks so it is tempting to put it off, however consider a possible schedule for completing this project:

1. Use the Graph class provided by Appel in a test driver. You might want to modify the implementation so that it uses Java generics.
2. Implement and test the AssemFlowGraph class, which upon construction will create a control-flow graph where each node contains one Assem.Instr.
3. Design and writeup your first register allocation version.
4. Write your test cases.
5. Implement and test your first register allocation version.
6. Write your benchmarks.
7. Figure out how to grab the cycle count information from SPIM, design the graphs, and generate the graphs.

8. Writeup the results from your first register allocation version.
9. Design, writeup, implement, and test your second register allocation version.
10. Design, writeup, implement, and test your second register allocation version.
11. Design, writeup, implement, and test your second register allocation version.
12. Finish your report.

You will have to probably redo steps when you find errors or problems. Start Early!! Also, notice in the above suggested sequence that you could turn in a report after only doing one register allocation implementation. It would be MUCH better to do that than to get all the register allocation implementations done, but not finish your report in time.

I can look at your code and help point you in the right direction, but the amount of help I can give may be inversely proportional to the amount of time until the due date. By no means should you spend several hours trying to figure out a weird bug; consult me for help. When e-mailing me about the project, send a copy of the relevant section of your code (not as an attachment; send it as text pasted into your message). Give a good description of the problem including information about the stack in a debugger.

Use dot to visualize the control-flow graph and the interference graph.

6 What to turn in

Turn in a hard copy of the report and email a copy in pdf format to mstrout@cs.colostate.edu.

Create a jar file that includes all of the bytecode files, the source files, your test cases, and your benchmarks. The jar file should also include a README that gives specific command-lines for running the jar file on your provided test cases. The README and the report should indicate which of your register allocation versions you predict will perform the best when compared to other register allocators in the class.

7 Due date

This assignment is due Friday September 22nd, **at 2:10pm** and is worth 10% of your final grade. Late assignments will be penalized 10% per day.