California Institute of Technology
Department of Computer Science
Computer Architecture

CS184b, Winter 2000

Assignment 2: ISA Characteristics

Monday, January 8

Due: Tuesday, January 16, 5:00PM

This assignment is mostly to get you started using simplescalar. I'm asking you to pick a suitable application, compile it, and do some preliminary profiling using simplescalar. In later assignments we'll use additional features of the simplescalar suite to measure the impact of various effects which we'll be studying.

To Do

1. Pick a moderate sized program to use in this, and perhaps future, assignments.
   - Program should be something which can reasonable run for one or a few million dynamic instructions. (I want it to be large enough to get statistically significant results, but small enough that you're not waiting forever for runs.)
   - Program should have reasonable tight source so that building it is easy.
   - Aim for a small application (bigger than a kernel, smaller than gcc). I used rijndael as an example; it's probably towards the small end. gzip is another example of about the right size.
   - Your program should be able to run non-interactively (input from file or scripted), since you'll want to set it off and run it many times.
   - Don't pick something which is completely IO dominated; you'll get one result (that it's IO dominated) and the rest won't be too interesting to study.
   - Rijndael is the newly selected (AES) encryption standard. I built and tested rijndael in `/cs/courses/cs184/assign/Rijndael` and have left it there. (I might play with it more.)
2. Build a simplescalar "PISA" executable using simplescalar's gcc cross compiler.
   - simplescalar support lives in `/cs/courses/cs184/software/simplescalar/`
   - binary executables live in the `bin/` subdirectory
   - you will be using `bin/sslittle-na-sstrix-gcc`, at least, and made need other binary utilities from that directory
3. You probably want to read a bit about simplescalar:
   - Note that we are using simplesim-3.0 (the simplesim-2.0.broken subdirectory should be ignored)

- Technical Report `/cs/courses/cs184/software/simplescalar/TR_1342.ps`
- Slides on using `/cs/courses/cs184/software/simplescalar/simplesim-3.0/hack_guide.pdf`

4. Verify your program runs under simplescalar using `simplesim-3.0/simple-safe`.
5. Run your program under the profiler `simplesim-3.0/simple-profile`.
   - you probably want to run it with the `-all` option to start with and begin getting familiar with all the things it can profile; later, you may want to focus which profiling results you request.
6. Compile your program at two different optimization levels (probably none and the highest which will work); Note the difference in dynamic instructions executed and distribution of operations.

Turnin

This lab must be returned electronically (HTML preferred). I will link all the labs off of the web page so the class can share the results to see how the statistics vary among applications.

1. Identify and briefly describe (1-2 paragraph) selected application. Make sure you include:
   - What does it do? (external function, major algorithmic elements, what kind of computation do you expect makes up most of the dynamic execution)
   - Where did it come from? (proper bibliographic citation, if appropriate)
   - Why did you choose it?
   - Benchmark dynamic instructions executed (for chosen input sequence—identify input sequence as well)
2. Summarize the compiler optimization effects (total instructions before and after optimization, breakdown by instruction class before and after optimization).
3. Identify the top 10 instructions by dynamic frequency for the before and after optimization cases and compare these with the data in the text (Figure 2.11) — use a table or tables comparable to 2.11. If appropriate, comment on the differences among the three cases.