CS152 Laboratory Exercise 2

Instructor: John Lazzaro
TA: Eric Love Authors: Yunsup Lee, Andrew Waterman, Eric Love
Department of Electrical Engineering & Computer Science
University of California, Berkeley

March 2014

1 Introduction and goals

The goal of this laboratory assignment is to conduct some simple memory hierarchy experiments in the RISC-V simulation environment. Using the cache simulator module, you will collect cache statistics and make architectural recommendations based on the results.

The lab has two sections, a directed portion and an open-ended portion. Everyone will perform the directed portion the same way, and grades will be assigned based on correctness. The open-ended portion will allow you to pursue more creative investigations, and your grade will be based on the effort made to complete the task or the arguments you provide in support of your ideas.

Students are encouraged to discuss solutions to the lab assignments with other students, but must complete the lab by themselves and turn in their own lab report.

You are only required to do one of the open-ended assignments. These assignments are generally starting points or suggestions. Alternatively, you can propose and complete your own open-ended project as long as it is sufficiently rigorous. If you feel uncertain about the rigor of a proposal, feel free to consult the instructor or the TA.

1.1 Tools and Benchmarks

The processors that you will be studying in this lab implement the RISC-V ISA, recently developed at UC Berkeley for use in education and research.

An entire tool-chain is provided. The `riscv-gcc`, `riscv-g++` cross-compilers build new binaries from RISC-V assembly, `C`, and `C++` source codes. The `riscv-objdump` tool can disassemble existing RISC-V binaries to show the exact sequence of instructions being executed.

The ISA simulator `riscv-isa-sim` can execute RISC-V binaries. We consider it to be the golden reference for the ISA. The ISA simulator executes RISC-V code rapidly, but does not model pipeline timing and so is not cycle-accurate.

In Lab 2, we will use an ISA simulator that has been extended with a cache simulator. The cache simulator will run memory addresses through a simulated cache (with a given size, associativity, and block size), and record the number of accesses, hits, and misses. With the simulated miss rate and miss penalty, we can estimate the impact on CPI. We also use CACTI (`http://quid.hpl.hp.com:9081/cacti`) to see how various cache parameters will impact the cycle time of the pipeline.

The ISA simulator is used in this lab, not the Chisel-generated emulator, even though the ISA simulator doesn't give accurate cycle counts: 1) Typically, we need to run a couple billion cycles to

get a realistic view of the cache behavior, and the Chisel-generated emulator runs a few orders of magnitudes slower than the ISA simulator, 2) the AMAT/CPI performance estimate for a simple pipeline turns out to be reasonably accurate.

We will use 4 benchmarks (`bzip2, mcf, sjeng, lbm`) from the SPEC CPU2006 benchmark suite (`http://www.spec.org`). SPEC is a standardized set of benchmarks to evaluate the performance of modern computer systems. The test results are published on the SPEC website. To read more about SPEC, please consult the SPEC website.

1.2 Graded Items

You will turn in a hard copy of your results to the instructor or TA. Please label each section of the results clearly. The directed items need to be turned in for evaluation. You only need to turn in *one* of the problems found in the open-ended portion.

1. (Directed) Problem 2.2: simple cache statistics for each benchmark and answers
2. (Directed) Problem 2.3: suggested working sets and evidence
3. (Directed) Problem 2.4: optimal I$ configuration and evaluation
4. (Directed) Problem 2.5: optimal D$ configuration and evaluation
5. (Directed) Problem 2.6: optimal D$ configuration with an L2$ and evaluation
6. (Open-ended) Problem 3.1: optimal memory hiearchy that fits in a 5mm^2 area budget
7. (Open-ended) Problem 3.2: suggested victim cache configuration, modifications, and evaluation (include source code if required)
8. (Open-ended) Problem 3.3: suggested prefetching algorithm, modifications, and evaluation (include source code if required)

Lab reports must be in *readable* English and not raw dumps of log-files. It is *highly* recommended that your lab report be typed. Charts, tables, and figures - when appropriate - are great ways to succinctly summarize your data.

2 Directed Portion

2.1 Setting Up Your Workspace

To complete this lab you will log in to an instructional server to run the RISC-V ISA simulator and compiler tool chain. We will provide you with an instructional computing account for this purpose.

The tools for this lab were set up to run on any of the twelve instructional Linux servers `t7400-1.eecs`, `t7400-2.eecs`, `...`, `t7400-12.eecs`. (see `http://inst.eecs.berkeley.edu/cgi-bin/clients.cgi?choice=servers` for more information about available machines).

First, download the lab materials[1] into your private github repo's directory:

```
inst$ cd ~/cs152-[github-id]
inst$ cp -R ~cs152/sp14/lab2 .
inst$ cd lab2
inst$ echo $PWD
```

[1]The capital "R" in "cp -R" is critical, as the -R option maintains the symbolic links found in the ${LAB1ROOT} directory.

The path you've just output to the terminal (probably `~/cs152-[..]/lab2`) we will refer to as `${LAB2ROOT}` in the rest of this document to denote that location of the Lab 2 directory. *You need to add the following line to your* `.bashrc` *file BEFORE the line that sources the cs152 configure script*:

```
export LAB2ROOT=~/cs152-[github-id]/lab2  # or whatever your lab2 path was
```

This will ensure that the script automatically adds the ISA simulator that you will build to your path. After you make this change, you should run

```
inst$ exec bash
```

to ensure that this script will have run properly before you continue.

The structure of `${LAB2ROOT}` is as follows:

- `${LAB2ROOT}`/
  - riscv-isa-sim/ Source code for the RISC-V ISA Simulator
    - riscv/ Source code for the RISC-V ISA Simulator
  - spec-cpu2006-riscv
    - 401.bzip2/ Source and data files for the bzip2 benchmark
      - src/ Source files
      - data/ Data files
    - 429.mcf/ Source and data files for the mcf benchmark
    - 458.sjeng/ Source and data files for the sjeng benchmark
    - 470.lbm/ Source and data files for the lbm benchmark

Build the ISA simulator with the following commands.

```
inst$ cd ${LAB2ROOT}/riscv-isa-sim
inst$ mkdir build
inst$ cd build
inst$ ../configure --prefix=${LAB2ROOT}/install
inst$ make -j 4
inst$ make install
```

Please use -j 4 or another reasonable number so as not to overuse the instructional machines.
To check whether the ISA simulator is in your path, run the following command.

```
inst$ which spike
```

2.2 Collecting statistics from a simple cache

You should first build the benchmarks.

```
inst$ cd ${LAB2ROOT}/spec-cpu2006-riscv
inst$ make -j 4
inst$ ls -ls build.riscv/
total 17120
  932 -rwxr-xr-x 1 yunsup grad  1014224 Feb 15 23:31 401.bzip2
 1348 -rwxr-xr-x 1 yunsup grad  1440538 Feb 15 23:31 429.mcf
11940 -rwxr-xr-x 1 yunsup grad 12285464 Feb 15 23:31 450.soplex
 1632 -rwxr-xr-x 1 yunsup grad  1730854 Feb 15 23:31 458.sjeng
 1268 -rwxr-xr-x 1 yunsup grad  1357611 Feb 15 23:31 470.lbm
```

Execute the target binary on the ISA simulator with an L1 instruction cache.[2]

```
inst$ cd ${LAB2ROOT}/spec-cpu2006-riscv
inst$ mkdir test
inst$ cd test
inst$ spike --ic=128:2:64 pk ../build.riscv/401.bzip2 \
      ../401.bzip2/data/test/input/dryer.jpg
spec_init
Loading Input Data
...
Tested 64KB buffer: OK!
I$ Bytes Read:            4539708644
I$ Bytes Written:         0
I$ Read Accesses:         1134927161
I$ Write Accesses:        0
I$ Read Misses:           1534
I$ Write Misses:          0
I$ Writebacks:            0
I$ Miss Rate:             0.000%
```

The 3 parameters given to the instruction cache `--ic` are `number of sets:associativity:line size`. If you multiply all the numbers $128 \times 2 \times 64$, you will get cache size (16KB). Note that "I$ Read Accesses" equals the number of instructions executed.

You can also run the simulation with an L1 data cache, and with an unified L2 cache with the following commands.

```
inst$ spike --dc=128:2:64 ...
inst$ spike --dc=128:2:64 --ic=128:2:64 ...
inst$ spike --dc=128:2:64 --ic=128:2:64 --l2=1026:4:64 ...
```

We wrote a Makefile that launches the benchmarks. Execute all the tests in parallel with the following command. It will take a couple minutes to run all 4 benchmarks.

```
inst$ cd ${LAB2ROOT}/spec-cpu2006-riscv
```

[2]The pk is necessary in the position shown because it invokes the *proxy kernel*, whose responsibility it is to handle basic syscalls needed by the C standard library to run the SPEC benchmarks.

```
inst$ make -j run
inst$ ls -ls run.riscv/
total 36
 4 -rw-r--r-- 1 yunsup grad  1398 Feb 15 23:58 401.bzip2.out
 4 -rw-r--r-- 1 yunsup grad  1353 Feb 15 23:59 429.mcf.out
 4 -rw-r--r-- 1 yunsup grad  2382 Feb 15 23:58 458.sjeng.out
 4 -rw-r--r-- 1 yunsup grad  1469 Feb 15 23:58 470.lbm.out
16 -rw-r--r-- 1 yunsup grad 14800 Feb 15 23:59 mcf.out
```

Answer the following 7 questions.

(Q1) For each benchmark, look at the corresponding output file and record the miss rate for the L1 I$, L1 D$, and the L2$. Which benchmark has the best cache performance? Which has the worst? We highly recommend you to automate this process by scripting as we are going to analyze a lot data in the subsequent sections.

(Q2) What is the cache access time for the L1 I$, L1 D$, and the L2$? Refer to Table 1.

(Q3) What is the cycle time of the pipeline? Assume that the critical path among all non-memory pipeline stages is 600ps long.

(Q4) Calculate the average CPI (in cycles) across all benchmarks without the L2$. Use the following formula to calculate CPI. (MP=Miss Penalty, CT=Cycle Time. Assume that the backside of the L1$s are connected to a DRAM, and it takes 100ns to access the DRAM. Use $CPI_{base} = 1.2$)

$$CPI = CPI_{base} + \frac{L1\ I\$\ misses + L1\ D\$\ misses}{\#\ of\ insts} \times \lceil \frac{MP}{CT} \rceil$$

(Q5) What is the AMAT (in ns) of the L2$ for all benchmarks? Use the following formula to calculate AMAT. (HT=Hit Time, MR=Miss Rate, MP=Miss Penalty. Assume that the backside of the L2$ is connected to a DRAM, and it takes 100ns to access the DRAM)

$$AMAT_{L2} = HT_{L2} + MR_{L2} \times MP_{L2}$$

(Q6) Calculate the average CPI (in cycles) across all benchmarks with the L2$. Use the following formula to calculate CPI, assuming the L2$ is running asynchronously on its own clock domain. (Use $CPI_{base} = 1.2$)

$$CPI = CPI_{base} + \frac{L1\ I\$\ misses + L1\ D\$\ misses}{\#\ of\ insts} \times \lceil \frac{AMAT_{L2}}{CT} \rceil$$

(Q7) Compare the average CPI value with and without the L2$. Does the L2$ help performance?

2.3 Determining benchmark working-set size

Your task in this section is to determine the working-set size of each of the benchmarks by varying the size and associativity of the L2$ cache used in the previous section. Record the measurements you make that support your claim. Which benchmark seems to have the largest working set, and how big is it? We have provided you a python script that will help you launch many simulation jobs. Take a look at `$LAB2ROOT/spec-cpu2006-riscv/explore.py`. Simply write a loop that populates the `design_space` dictionary. Finally, run the script to kick off the design-space exploration.

```
inst$ cd ${LAB2ROOT}/spec-cpu2006-riscv
inst$ ./explore.py
```

2.4 Find the Optimal L1 I$ Configuration

For this section, you will find the optimal L1 I$ configuration that maximizes performance of the 5 SPEC benchmarks. Assume the data memory accesses all hit in the cache, and hence don't affect the CPI.

Note:

- Limit the L1 cache design space to: capacity[16KB,32KB,64KB] × associativity[1,2,4,8] × cache line size[64].
- To calculate CPI, come up with a similar formula you have used in 2.2.
- Consult Table 1 to see how cache access time scales with different cache configurations.
- Think carefully how the cache access time will affect the cycle time of the processor, and the overall performance.
- We encourage you to modify the design-space exploration script used in 2.3.
- We put a text version of Table 1 in `$LAB2ROOT/spec-cpu2006-riscv/cacti` just in case you would like to read it from your analysis script.

Which configuration do you recommend for the L1 I$? Show your work. Based on your findings, can you provide any intuition behind sizing the L1 I$? If you account the silicon area for different cache configurations (see Table 2), would your recommendation change? If so, why?

2.5 Find the Optimal L1 D$ Configuration

For this section, you will find the optimal L1 D$ configuration that maximizes performance of the 5 SPEC benchmarks. Pick one configuration from the following cache design space: capacity[16KB,32KB,64KB] × associativity[1,2,4,8] × cache line size[64]. Assume your processor has the L1 I$ you have recommended in Section 2.4.

Which configuration do you recommend for the L1 D$? Show your work. Based on your findings, can you provide any intuition behind sizing the L1 D$? If you account the silicon area (see Table 2), would your recommendation change? If so, why?

2.6 Find the Optimal L1 D$ Configuration with an L2$

This question is similar to Section 2.5, but now we assume that we have an L2$ between the L1$s and the DRAM. Pick one configuration for the L1 D$ from the following cache design space that maximizes performance: capacity[16KB,32KB,64KB] × associativity[1,2,4,8] × cache line size[64]. Assume your processor has the L1 I$ you have recommended in Section 2.4. Also assume a fixed 256KB 8-way set-associative L2 cache that uses 64-byte cache lines.

Which configuration do you recommend for the L1 D$? Show your work. How does the L2$ affect your cache design decisions?

assoc — size	8KB	16KB	32KB	64KB	128KB	256KB	512KB	1MB
1	0.31	0.37	0.44	0.53	0.62	0.79	0.98	1.31
2	0.51	0.57	0.62	0.65	0.74	0.84	1.17	1.52
4	0.56	0.60	0.65	0.70	0.74	0.85	1.17	1.52
8	0.77	0.78	0.86	0.89	0.95	1.03	1.16	1.52
16	N/A	1.21	1.24	1.30	1.35	1.42	1.53	1.69
32	N/A	N/A	2.10	2.12	2.19	2.30	2.45	2.46
64	N/A	N/A	N/A	3.90	3.92	3.95	4.02	4.13

(a) cache line size = 32

assoc — size	8KB	16KB	32KB	64KB	128KB	256KB	512KB	1MB
1	0.32	0.34	0.41	0.53	0.61	0.79	1.04	1.31
2	0.60	0.61	0.64	0.68	0.74	0.85	1.09	1.44
4	0.83	0.84	0.86	0.91	0.94	0.99	1.09	1.44
8	N/A	1.28	1.30	1.34	1.37	1.41	1.48	1.60
16	N/A	N/A	2.05	2.21	2.24	2.29	2.34	2.44
32	N/A	N/A	N/A	3.97	4.01	4.04	4.10	4.18
64	N/A	N/A	N/A	N/A	7.20	7.25	7.28	7.35

(b) cache line size = 64

assoc — size	8KB	16KB	32KB	64KB	128KB	256KB	512KB	1MB
1	0.37	0.45	0.46	0.53	0.62	0.85	1.04	1.37
2	0.83	0.84	0.87	0.90	0.94	0.98	1.06	1.42
4	N/A	1.29	1.29	1.32	1.35	1.39	1.44	1.53
8	N/A	N/A	2.16	2.18	2.25	2.27	2.33	2.36
16	N/A	N/A	N/A	3.92	3.96	4.02	4.02	4.12
32	N/A	N/A	N/A	N/A	7.42	7.46	7.49	7.54
64	N/A	N/A	N/A	N/A	N/A	13.84	13.88	13.92

(c) cache line size = 128

Table 1: Cache access time (in ns) for various cache configurations in 45nm technology. Data obtained from CACTI.

assoc — size	8KB	16KB	32KB	64KB	128KB	256KB	512KB	1MB
1	0.09	0.23	0.35	0.54	1.12	1.53	3.11	6.78
2	0.12	0.16	0.23	0.46	0.65	1.30	2.94	5.49
4	0.18	0.34	0.40	0.41	0.65	1.30	2.94	4.88
8	0.37	0.49	0.59	0.73	1.32	1.36	2.62	5.55
16	N/A	0.80	0.99	1.08	1.34	2.29	3.37	5.51
32	N/A	N/A	1.77	2.06	2.23	3.40	4.07	6.46
64	N/A	N/A	N/A	4.03	4.32	4.79	5.83	7.96

(a) cache line size = 32

assoc — size	8KB	16KB	32KB	64KB	128KB	256KB	512KB	1MB
1	0.21	0.25	0.33	0.73	1.25	1.95	3.24	6.98
2	0.34	0.37	0.44	0.57	1.10	1.47	3.12	6.99
4	0.55	0.64	0.71	0.84	1.48	2.03	3.12	6.99
8	N/A	1.12	1.26	1.39	2.22	2.75	3.78	5.81
16	N/A	N/A	3.11	2.54	2.78	4.35	5.36	7.34
32	N/A	N/A	N/A	4.89	5.14	5.64	8.66	10.92
64	N/A	N/A	N/A	N/A	10.23	10.75	15.44	17.37

(b) cache line size = 64

assoc — size	8KB	16KB	32KB	64KB	128KB	256KB	512KB	1MB
1	0.70	0.75	0.82	0.98	2.03	3.13	4.60	7.38
2	1.12	1.27	1.35	1.48	1.77	3.30	4.50	7.39
4	N/A	2.18	2.36	2.49	2.74	4.65	5.72	7.85
8	N/A	N/A	4.29	4.57	5.52	6.04	7.16	10.63
16	N/A	N/A	N/A	8.65	9.16	10.69	10.58	13.72
32	N/A	N/A	N/A	N/A	17.90	18.36	19.35	21.84
64	N/A	N/A	N/A	N/A	N/A	36.17	37.17	39.01

(c) cache line size = 128

Table 2: Cache area (in mm^2) for various cache configurations in 45nm technology. Data obtained from CACTI.

3 Open-ended Portion

Pick *one* of the following questions. The open-ended portion is worth a large fraction of the grade of the lab, and the grade depends on how complex and interesting a project you complete, so spend the appropriate amount of time and energy on it. Also, have fun with it!

3.1 Design a memory hierarchy that fits within a 5mm^2 area budget

For this question, we want to figure out how we can make the best use out of our 5mm^2 area budget for caches. In the directed portion of the lab, we have asked you to explore the design space of L1 caches. However, we have constrained the design space to minimize the busy work. But for this study, you have the freedom to change any cache parameter. The only constraint you have is to *fit in a 5mm^2 area budget*. Propose the best memory hierarchy for the 5 SPEC benchmarks we used in this lab.

Use Table 2 to estimate area, and Table 1 to estimate cache access time for different cache configurations. If the tables don't have an estimate for your cache configuration, please use the CACTI web interface (`http://quid.hpl.hp.com:9081/cacti`) to obtain them. Use 1 bank and technology node of 45nm.

Make sure to report all the statistics you gathered and calculations you made to reach your conclusions.

3.2 Design a victim cache

Although direct-mapped caches have an advantage of smaller access time than set-associative caches, they have more conflict misses due to their lack of associativity. In order to reduce these conflict misses, N. Jouppi proposed victim caching where a small fully-associative back up cache, called a victim cache, is added to a direct-mapped L1 cache to hold recently evicted cache lines.

Given a 32KB direct-mapped L1 D$, design your own victim cache. Assume that the backside of the L1 D$ is directly hooked up to the DRAM. To read more about victim caches, please consult problem the victim cache paper linked to on the course website

`http://www-inst.eecs.berkeley.edu/~cs152/sp14/handouts/local_only/victim.pdf`

The only constraint you have is to *add fewer than 2K flip-flops* to the cache design.

First sketch out a block diagram. Then modify the cache simulator to model your victim cache. You will have to modify the `cache_sim_t::access()` function in `$LAB2ROOT/riscv-isa-sim/riscv/cachesim.cc`. Recompile the ISA simulator with the steps described in Section 2.1. Run the benchmarks and record the cache statistics. Analyze the impact on AMAT and CPI. Estimate the impact on critical path, and performance. Change parameters in your design and see which configuration works the best.

Make sure to report all the statistics you gathered and calculations you made to reach your conclusions.

Feel free to email your TA or attend his office hours if you need help understanding the ISA simulator, the cache simulator, or anything else regarding this problem.

3.3 Design your own hardware prefetcher

For this question, we want to investigate whether hardware data prefetching would improve performance of the 5 SPEC benchmarks we have used in the directed portion. Please take a look at

the "Cache II" lecture for more information on hardware prefetching.

Assume you are building a hardware prefetcher for a 32KB 4-way set-associative L1 D$. The backside of the L1 D$ is directly hooked up to the DRAM. The only constraint you have is to *add fewer than 2K flip-flops* to the cache design.

First sketch out a block diagram. Then modify the cache simulator to model your victim cache. You will have to modify the `cache_sim_t::access()` function in `$LAB2ROOT/riscv-isa-sim/riscv/cachesim.cc`. Recompile the ISA simulator with the steps described in Section 2.1. Run the benchmarks and record the cache statistics. Analyze the impact on AMAT and CPI. Estimate the impact on critical path, and performance. Change parameters in your design and see which configuration works the best.

Make sure to report all the statistics you gathered and calculations you made to reach your conclusions.

Feel free to email your TA or attend his office hours if you need help understanding the ISA simulator, the cache simulator, or anything else regarding this problem.

4 Acknowledgments

Many people have contributed to versions of this lab over the years. This lab was originally developed for CS152 at UC Berkeley by Yunsup Lee and Andrew Waterman, and heavily inspired by the previous set of CS 152 labs (which targeted the Simics emulators) written by Henry Cook. This lab was made possible through the work of Andrew Waterman, Yunsup Lee, David Patterson, and Krste Asanović who developed the RISC-V ISA.