Assignment 2: Understanding and Optimizing the Cache

Computer Architecture

Issued:	Friday, March 4, 2016
Due:	Friday, March 25, 2016 at 4:00 PM

1 Introduction

This assignment represents the second practical component of the Computer Architecture module. This component contributes 12.5% of the overall mark for the module. It consists of a programming exercise culminating in a written report. Assessment of this assignment will be based on the correctness of the code and the clarity of the report (see more details below). The assignment is to be solved individually to assess your competence on the subject. Please bear in mind the School of Informatics guidelines on plagiarism. You must submit your solutions before the due date shown above. Follow the instructions provided in Section 6 for submission details.

In this assignment, you are required to implement and integrate 1) a data cache and 2) prefetching techniques in a processor simulator modeling a 5-stage pipeline[1]. You are strongly advised to commence working on the assignment as soon as possible.

2 Assignment

Cache is a special memory that is used to store instructions and/or data so that the processor can access them at a very high speed. In this assignment, you will investigate the effect of data caching and prefetching on the performance. Towards this end, you have to implement and integrate the following into the 5-stage pipeline simulator you are provided with:

1. Data Cache: You need to implement a parameterized data cache in the simulator.
2. Prefetcher: Your need to implement prefetchers and tune their parameters.

[1]The simulator is a modified version of https://github.com/clord/MIPS-CPU-Simulator.

2.1 Date Cache

Following are the parameters for the data cache:
Command line Cache Parameters:

1. Cache Size: 1 KB, 2 KB, 4 KB or 8 KB.

The variables for command line arguments are already provided in the baseline code. You can find the corresponding variables in "cache.h".

```
extern uint32_t cache_size;     //in bytes
extern int prefetcher_type;
```

Fixed parameters:

1. Associativity: 2-way
2. Cache Line (Block) Size: 32 bytes.
3. Cache Policies: Write-back, Write-allocate.
4. Replacement Policy: Least Recently Used (LRU).
5. Cache Hit Latency: 1 cycle (no stall).
6. Cache Miss Penalty: 5 cycles (4 stall cycles).
7. Maximum Outstanding Miss Requests: 1. There can be just one pending miss request (pendingMissReq). In other words, if the cache has requested a block from memory it cannot request another block until the first one has been received. This is important for the prefetching aspect of the assignment. However, hit under a pending miss is allowed.

2.2 Prefetcher

Prefetching is a technique aimed at fetching the required data into the cache before it is needed, thereby (potentially) reducing cache miss rate and improving performance. Prefeteching techniques are available in both hardware and software flavors. In this assignment you have to implement two hardware prefetchers.

2.2.1 Next-N-Line Prefetcher:

The simplest hardware prefetcher is a "Next-N-Line Prefetcher[2]" that prefetches N cache lines next to the one being accessed. There are several parameters in the Next-N-Line prefetcher such as when to issue a prefetch, how many next lines to prefetch, etc. Different policies can be used to decide when to issue a prefetch such as:

[2]§4.1 Sequential prefetching (Next-N-Line Prefetcher) in http://www.ece.lsu.edu/tca/papers/vanderwiel-00.pdf.

1. Policy 1:
   a) Only on a cache miss.
2. Policy 2:
   a) On a cache miss.
   b) On a cache hit in previously prefetched blocks.
3. Policy 3:
   a) On a cache miss.
   b) On a cache hit in any block.

Moreover, the number of next blocks to prefetch can vary between different implementations of Next-N-Line Prefetcher. The simplest Next-N-Line Prefetcher prefetches just 1 next block (N=1). However, more aggressive prefetcher can prefetch more next blocks. Furthermore, if the next blocs are already in the cache, they need not to be prefetched.

2.2.2 Better Prefetcher:

You need to implement a prefetcher to improve the performance over the Next-N-Line prefetcher. You can choose any prefetcher you want; one good option is a Stride prefetcher[3]. You will need to describe the prefetcher, implementation details, and its performance in the report.

3 Simulator

Please download the *latest code* for the simulator from the course homepage for this coursework (do not reuse the code from the first coursework as there have been some changes made).
Compile. You can compile the simulator with the following command on a DICE machine:

```
tar zxvf sim-car2.tar.gz
cd sim-car2
make   # you can ignore warnings during the compilation of "rasm"
```

Run. You can run the simulation with the following command:

```
# you can try different workloads (linear, matrix, matrix_big)
./rsim -t programs/matrix.t -d programs/matrix.d
```

Debug. You can print pipeline state in each cycle by adding “-v” command:

```
./rsim -t programs/matrix.t -d programs/matrix.d -v
```

[3]§4.2 Prefetching with arbitrary strides (Stride Prefetcher) in http://www.ece.lsu.edu/tca/papers/vanderwiel-00.pdf.

Other Information. In case you want to test the simulator with other assembly programs (e.g. assembly.s), you first need to assemble them using the following command:

```
./rasm -f <assembly.s>
```

which will produce two files assembly.t and assembly.d, which you can then feed to the simulator.
Command-line Parameters. Following are the parameters that you need to pass from the command-line:

1. Cache Size (-s): The number after "-s" flag indicates cache size in KB. Possible values: *0 (No Cache), 1 (1 KB), 2 (2 KB), 4 (4 KB) and 8 (8 KB).*

2. Prefetcher (-x): Which prefetcher to use. Options: 0 (No prefetcher), 1 (Next-N-Line prefetcher), 2 (Better prefetcher)

An example command line will look like:

```
./rsim -t programs/matrix.t -d programs/matrix.d -s 2 -x 1
```

The above command line invokes the simulator with 2KB cache size and will also enable the Next-N-Line prefetcher.

3.1 Reference results

The reference results for data cache (without prefetching) are shown in Table 1 and Table 2.

Table 1: Miss Rate

Cache Size	linear.s	matrix.s	matrix_big.s
1 KB	81.51 %	51.78 %	56.31 %
2 KB	81.51 %	18.42 %	54.86 %
4 KB	73.96 %	9.01 %	18.18 %
8 KB	33.59 %	5.51 %	11.76 %

Table 2: Processor Cycle Count

Cache Size	linear.s	matrix.s	matrix_big.s
1 KB	22803	5170738	40482112
2 KB	22803	4844443	40366648
4 KB	22255	4758084	37501677
8 KB	19162	4723588	37067148

3.2 Advice and Hints

1. You need not model the latency of the writeback when you replace a dirty block. In another words, writeback happens effectively in zero time.
2. You do not need to simulate the data in the cache. CacheCtrl::responseToCPU provides a hack to get the real data from memory directly.
3. <CAR-PA2-REF> (in cache.h and cache.cc) provides reference code for you to integrate your cache implementation to the simulator.
4. For the prefetcher, you should place the prefetched block into the cache directly.
5. Please put comments in the code that you modify/add to make it easy for the markers to understand.

4 Report

Your report should be divided into two parts within the same PDF document:

1. Cache Implementation: You need to write a section (max 1 page) on how you have implemented the cache, where the cache code is (source file, line numbers) and how you integrated it to the CPU pipeline.
2. Prefetcher Details: There is no page limit for this one. You need to organize the details of each prefetcher in following sections:
    a) **Technique:** Provide an abstract description of the prefetching technique.
    b) **Implementation:** Provide the details of how you implemented the prefetcher and where exactly the corresponding code is (source file, line numbers).
    c) **Results:** Compare the results (in bar/line charts) with and without prefetching with different cache sizes and workloads (linear.s, matrix.s, and matrix_big.s).

    Also, write a **discussion section** on what you learned from this assignment.

5 Marking Scheme

Part 1: Cache (50 marks).

1. **Correctness (30 marks).** Testing of your cache implementation.
2. **Report (20 marks).** A brief report (max. 1 page) on cache implementation.

Part 2: Prefetcher (50 marks).

1. **Next-N-Line Prefetcher (10 marks)** We do not provide reference results on the prefetcher implementation. Instead, you will need to justify your design in the report.

2. **Tuning Next-N-Line Prefetcher (10 marks).** The effectiveness of the Next-N-Line prefetcher depends on how well tuned it is. Therefore, to maximize the performance gain (in terms of processor cycles), you need to tune the parameters described in Section 2.2.1 (when to issue a prefetch and how many blocks to prefetch). You are required to document your prefetching scheme in the report.

3. **Better Prefetcher (10 marks).**

4. **Report (20 marks).** Please reference to Section 4.

6 Format of your submission

Remember that your simulator will be compiled/executed on a DICE machine, so you must ensure it works on DICE.
Archive. You can archive the source code using the make command:

```
$ make deliverable
```

Submission. Submit an archive of the simulator (sim-car2.tar.gz) *and* a soft copy of your report as follows, using the DICE environment:

```
$ submit car 2 sim-car2.tar.gz report.pdf
```

7 Reporting Problems

A discussion forum is available on https://piazza.com/class/ikc6va2pmdt1a7?cid=31. It is worth noting that this forum is not for you to ask questions such as “My results are wrong. How to fix it?”. You should only use them to clarify things which are unclear in the handout.