Assignment 1: Understanding Branch Prediction

Computer Architecture

Issued:	Thursday, February 4, 2016
Due:	Monday, February 22, 2016 at 4:00 PM

1 Introduction

This assignment represents the first practical component of the Computer Architecture module. This practical contributes 12.5% of the overall mark for the module. It consists of a programming exercise culminating in a brief 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 practical 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 4 for submission details.

In this assignment, you are required to explore Branch Prediction techniques in a processor simulator modeling a 5-stage pipeline[1]. You are strongly advised to commence working on the simulator as soon as possible. Note that the second assignment will require further (and more invasive) modifications to the simulator, so the time invested now in understanding the code will pay off later.

2 Experiments

A branch predictor anticipates the outcome of a branch, before it is executed, to improve the instruction flow in the pipeline. In this assignment you are will investigate the accuracy of various branch predictors. Towards this end, you have to implement each branch predictor and integrate it into the 5-stage pipeline simulator you are provided with. The branch prediction techniques you need to examine are:

1. Static Branch Prediction: Always Not Taken (already implemented in the baseline simulator) and Always Taken.

2. Dynamic Branch Prediction: 2-Bit Predictor and 2-Level Predictor.

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

2.1 Simulator

Compile. You can compile the simulator with the following command on a DICE machine:

```
unzip sim-base.zip
cd sim-base
make   # you can ignore warnings during the compilation of "rasm"
```

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

```
./rsim -t programs/test.t -d programs/test.d
```

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

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

Other Information. test.t and test.d files are the assembler outputs we have already assembled. 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.

2.2 Simulator and Branch Predictor Parameters

Specifying the predictor: The predictor type (-b) should be passed as command line arguments and the number after "-b" will indicate which predictor to use: *0 (Alwasys Not Taken), 1 (Always Taken), 2 (2-Bit Predictor), and 3 (2-Level Predictor).* You can find the tag (<CAR_PA1_HOOK1>) in simulator.cc for adding the parameter.
A sample command line would look like:

```
./rsim -t programs/test.t -d programs/test.d -b 2   # 2-Bit Predictor
```

Parameters for dynamic branch predictors: The parameters for 2-Bit and 2-Level predictors should be fixed to the following values.

1. Entry (Counter) Size: 2 bits.
2. Predictor Size: 2048 bits. (i.e, 2048/2 = 1024 entries)
3. Global History Buffer Length: 10 bits (i.e, the outcome of last 10 branches). It should be initialized to 0.
4. Each instruction is 8 bytes.

Note: All the 2-bit (saturation) counters in 2-Bit Predictor and 2-Level Predictor should be initialized to zero.

2.3 Advice and Hints

1. In this assignment, the branch predictor will make predictions at the instruction fetch stage and branches are resolved and branch predictor table is updated at the execution stage. Furthermore, you can assume a perfect BTB (Branch Target Buffer); that is, in the fetch stage , you should assume knowledge of whether the current instruction is a branch or not and its taken target address. The taken branch target address is available in "right.immediate". Please read the comments in stage.cc (<CAR_PA1_HOOK2>) for more details.

2. 2-Bit Predictor: As all the instructions are 8 byte aligned, the last 3 bits of instruction addresses are always zeros. Therefore, you can right shift the instruction address by 3 bits for looking up and updating the branch predictor.

3. 2-Level Predictor: You need to update the Global History Buffer in the Fetch stage (before the actual branch outcome is known) and the 2-bit counter in the execution stage (when the outcome is known). To do so, shift-in the prediction made by the 2-Level Predictor into the global history buffer register. However, before shifting in the prediction you need to make a copy of the Global History Buffer and pass it to the next pipeline stages. This copy would serve following purposes in the execution stage: 1) In case the branch was mispredicted, you need to copy it back to the Global History Buffer to the restore the correct history and then shift-in the correct branch direction. 2) To calculate the index for the branch predictor entry (i.e., the 2-bit counter) that needs to be updated at the execution stage.

2.4 Reference results

Here are some reference results for test.s.

Always Not Taken

```
stat.processorCycles: 2942
stat.BPHits: 396
stat.BPMisses: 357
```

Always Taken

```
stat.processorCycles: 3020
stat.BPHits: 357
stat.BPMisses: 396
```

2-Bit

```
stat.processorCycles: 2270
stat.BPHits: 732
stat.BPMisses: 21
```

2-Level

```
stat.processorCycles: 2320
stat.BPHits: 707
stat.BPMisses: 46
```

3 Marking Scheme

Correctness ***(80 marks).*** Includes testing and code quality for the following 3 predictors:

1. **Always-Taken Predictor** ***(20 marks).***
2. **2-Bit Predictor** ***(30 marks).***
3. **2-Level Predictor** ***(30 marks).***

Report ***(20 marks).*** You should write a short report (max 2 pages) on how you have implemented each predictor, where the predictor code is (source file, line numbers) and how you integrated the predictors into the cpu pipeline.

4 Format of your submission

Your submission should clearly indicate (in both the report and the code) which branch prediction techniques you have simulated completely and which you have only partially completed. Please put comments in the code that you modify/add to make it easy for the markers to understand. **Remember that your simulator will be compiled/executed on a DICE machine, so you must ensure it works on DICE**.

Submit an archive of your simulator source code *and* a soft copy of your report as follows, using the DICE environment:

```
$ submit car 1 <branch-predictor.tar.gz> <report.pdf>
```

5 Reporting Problems

Send an email to cheng-chieh.huang@ed.ac.uk and rakesh.kumar@ed.ac.uk for any issues regarding the assignment.