MIPS Simulator: Branch Predictor
Project 6 – CS 3339 – Spring 2016

Due: Wednesday, 4/27/16 @ noon
100 points

PROBLEM STATEMENT

In this project, you will attempt to reduce the number of pipeline flushes by adding a branch predictor to your simulator. This predictor will predict only conditional branch instructions, i.e. `beq` and `bne`.

You should begin with a copy of your Project 5 submission. However, to make this milestone fair to students who did not get Project 5 working, we will use Project 4's baseline cycle count (i.e., we will assume there is no cache and zero latency associated with reading/writing memory). I have provided a Makefile that will turn the cache off in this manner, as well as a class skeleton for a BranchPred class intended to be instantiated inside your existing CPU class. You may add any variables and functions to the BranchPred class that you'd like.

You should simulate a dynamic branch predictor that uses a 64-entry direct-mapped table with no tag and no valid bits. Each entry stores a 2-bit saturating counter (described below) used to generate a “taken” or “not-taken” prediction, as well as a target PC (i.e., the PC the branch is predicted to jump to if it is taken). The table should be indexed as follows: `index = (PCbranch >> 2) % BPRED_SIZE`. (I have defined BPRED_SIZE for you in BranchPred.h).

All counters should be initialized to zero. Whenever a branch is taken, update the corresponding 2-bit saturating counter by incrementing it, unless it is already at its maximum value (3). Whenever a branch is not taken, update the counter by decrementing it, unless it is already at its minimum value (0). The prediction should be “branch taken” if `PCbranch`’s count is at least 2 and “branch not taken” otherwise.

You may structure the interaction between your CPU and branch predictor however you'd like. However, be careful: for any branch instruction, first make a prediction, then update the predictor. In other words, you can only use past information to make a prediction. To make sure, it's simplest to have separate BranchPred::predict() and BranchPred::update() functions.

Up until now, your processor has flushed the pipeline behind any taken branch instruction. Modify your code so that a flush occurs only when a branch has been mispredicted.

Here's my recommended approach:
1.) Based on the PC, predict whether the operation will be a taken branch and to what address.
2.) Once the operation has been decoded as a branch (beq or bne), determine its actual direction (taken vs. not-taken) and target PC.
3.) Determine if there was a misprediction. A misprediction can be one of two types:
   a. Direction mispredict: compare the predicted and actual taken vs. not-taken determination
b. Target mispredict: *if the branch was correctly predicted taken*, compare the predicted and actual target PCs.
   If there was a misprediction, flush the pipeline behind the branch.

4.) Update the 2-bit saturating counter for the branch’s PC, based on its actual taken or not-taken determination. *If the branch was taken*, also update the predictor’s target PC.

Your branch predictor model will calculate and report the following statistics:

- The total number of **predicted branches**
- The number of branches that were **predicted taken vs. predicted not-taken**
- The total number of **mispredictions**
- The number of **direction mispredictions** (i.e., T vs. NT decision was wrong)
- The number of **target mispredictions** (i.e., the branch was correctly predicted taken but the target PC was wrong)
- The **overall predictor accuracy**

I have provided the following new files:

- A `BranchPred.h` class specification file, which you’ll need to add member variables and function prototypes to
- A `BranchPred.cpp` class implementation file, which you should enhance to model the described branch predictor and count predictions and mispredictions
- A new Makefile, which disables the Project 5 data cache and sets the bus latencies to zero

In addition to enhancing the `BranchPred.h/.cpp` skeleton, you will also need to modify `CPU.h` to instantiate a `BranchPred` object, and `CPU.cpp` to call your `BranchPred` class functions and call `Stats::flush()` only on mispredictions. You’ll also need to change `CPU::printStats()` to match my expected output format (see below).

---

**ASSIGNMENT SPECIFICS**

All provided files are on TRACS under Resources → Project Files → cs3339_project6.tgz.

Begin by copying all of your Project 5 files into a new Project 6 directory. Then extract the additional files in cs3339_project6.tgz and add them to your project6 directory. You can compile and run the simulator program identically to previous projects, and test it using the same *.mips inputs.

Begin by making sure that my Project 6 Makefile successfully disables the cache with no memory latency. Below is a table of expected pre-branch-predictor cycle, flush, and CPI metrics for some of the inputs. **You should not begin implementing your branch predictor until your baseline code matches these numbers!** (The other 3 inputs have un-interesting branch behavior and will not be tested).

<table>
<thead>
<tr>
<th>Input</th>
<th>Cycle Count</th>
<th>Flushes</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>hash.mips</td>
<td>89788</td>
<td>6526</td>
<td>2.3</td>
</tr>
<tr>
<td>nqueens.mips</td>
<td>432723233</td>
<td>24671130</td>
<td>2.1</td>
</tr>
<tr>
<td>prime.mips</td>
<td>676319</td>
<td>54450</td>
<td>2.5</td>
</tr>
<tr>
<td>qsort.mips</td>
<td>106731</td>
<td>8334</td>
<td>2.3</td>
</tr>
<tr>
<td>sssp.mips</td>
<td>983220</td>
<td>51990</td>
<td>2.2</td>
</tr>
</tbody>
</table>
The following is the expected output for sssp.mips. Your output must match this format verbatim:

CS 3339 MIPS Simulator  
Cache Config: cache disabled  
Branch Predictor Entries: 64  
Running: sssp.mips

7 1

Program finished at pc = 0x400440  (449513 instructions executed)

Cycles: 942018  
CPI: 2.1

Bubbles: 481710  
Flushes: 10788  
Stalls: 0

Branches predicted: 42954  
Pred T: 25851 (60.2%)  
Pred NT: 17103

Mispredictions: 5393 (12.6%)  
Mispredicted direction: 2803  
Mispredicted target: 2590

Predictor accuracy: 87.4%

The below table shows the expected cycle count, flush count, CPI, and predictor accuracy after the inclusion of the simulated branch predictor.

<table>
<thead>
<tr>
<th>Input</th>
<th>Cycle Count</th>
<th>Flushes</th>
<th>CPI</th>
<th>Predictor Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>hash.mips</td>
<td>83720</td>
<td>458</td>
<td>2.1</td>
<td>98.0%</td>
</tr>
<tr>
<td>nqueens.mips</td>
<td>415228123</td>
<td>7176020</td>
<td>2.0</td>
<td>92.8%</td>
</tr>
<tr>
<td>prime.mips</td>
<td>628351</td>
<td>6482</td>
<td>2.3</td>
<td>91.6%</td>
</tr>
<tr>
<td>qsort.mips</td>
<td>99403</td>
<td>1006</td>
<td>2.2</td>
<td>95.6%</td>
</tr>
<tr>
<td>sssp.mips</td>
<td>942018</td>
<td>10788</td>
<td>2.1</td>
<td>87.4%</td>
</tr>
</tbody>
</table>

Additional Requirements:

- **Your code must compile with the given Makefile and run on zeus.cs.txstate.edu**
- Your code must be well-commented, sufficient to prove you understand its operation
- Make sure your code doesn’t produce unwanted output such as debugging messages. (You can accomplish this by using the D(x) macro defined in Debug.h)
- Make sure your code’s runtime is not excessive
- Make sure your code is correctly indented and uses a consistent coding style
- Clean up your code before submitting: i.e., make sure there are no unused variables, unreachable code, etc.
SUBMISSION INSTRUCTIONS

Submit all of the code necessary to compile your simulator (all of the .cpp and .h files as well as the Makefile) as a compressed tarball. You can do this using the following Linux command:

```
$ tar czvf project6.tgz *.cpp *.h Makefile
```

Do not submit the executables (*.mips files). Any special instructions or comments to the grader, including notes about features you know do not work, should be included in a separate text file (not inside the tarball) named README.txt.

All project files are to be submitted using TRACS. Please follow the submission instructions here: [http://tracsfacts.its.txstate.edu/trainingvideos/submitassignment/submitassignment.htm](http://tracsfacts.its.txstate.edu/trainingvideos/submitassignment/submitassignment.htm)

Note that files are only submitted if TRACS indicates a successful submission.

You may submit your file(s) as many times as you'd like before the deadline. Only the last submission will be graded. **TRACS will not allow submission after the deadline**, so I strongly recommend that you don't come down to the final seconds of the assignment window. Late assignments will not be accepted.