LSU EE 4720 Homework 8 Due: 24 November 2010

Problem 1: Do Spring 2010 Final Exam Problem 2 (branch prediction). The grading criteria used for the final appear below. Positive numbers indicate total credit for a category, negative numbers are specific deductions.

```
Problem 2 (20 pts)
~~~~~~~~~

 5 Bimodal B2
   -2 Accuracy based on predicting after update.

 1 Local B2
 4 B2 min local history size
   -2 Nine, since pattern repeats every 10 executions.
   -2 Three, with "example"
   -0 Four, with "example"
 1 B2 local pht count

 1 Bimodal B3
 1 Local B3

 1 Global B3
 4 B3 min global history
   -3 Three, needs b1, b2, b3

 2 B3 warmup
   -1 2^5, 32 * 18 cycles
```

Problem 2: Do Spring 2010 Final Exam Problem 3 using some additional information provided here: The BHT is usually indexed with a subset of PC address bits, in class we like to use `11:2` (bits 11 to 2) for a 2^{10}-entry table but real systems use more bits. Even so, the entry retrieved from the BHT might not be for the branch being predicted. For example, a branch at PC `0x1234` and `0x9234` would have the same index (BHT address) bits, `0x234` (leaving the two least-significant bits on for convenience), but the higher bits, `31:12`, would be different: `0x1` for the first branch and `0x9` for the second. An ordinary branch predictor would treat these two branches as the same branch, with a possible loss of performance. A solution is to place some or all of the high-order PC bits in the BHT entry, along size the 2-bit counter, target, and other info. The high-order PC bits used this way are referred to as a *tag*. A full-size tag for MIPS-I and our default predictor would be `31:12` or 20 bits. When predicting a branch the tag in the entry that is retrieved is compared to the corresponding bits in the branch being predicted. If they match the BHT lookup is said to *hit*, if they don't match it is said to *miss*. Tags, along with target and other data, are stored when the predictor is updated. If we were predicting `0x1234` there would be a hit if the tag were `0x1`, if the tag were `0x9` that would be a miss. By using tags a *collision* between the two branches has been detected. Without tags the predictor would have to assume that the retrieved BHT entry was correct. For structures like branch predictors that don't have to always be right, it's possible to get away with fewer than full-size tags. So in this example, one could use just two tag bits and detect the collision. If one tag bit were used the tags would be the same for the example branches and so the collision would not be detected.

With the information above, solve Problem 3.

Problem 3: Do Spring 2010 Final Exam Problem 7 (short answers). Part (a) is easy, part (b) requires a small amount of thinking, parts (c) and (d) are easy.