Computer Design and Organization

Problem Set #3

Due: Monday February 7

This assignment is meant to make you understand the operation of a BTB. At the same time it serves as a refresher on the operation of caches.

You will design a BTB (branch target buffer) with 16 entries. The BTB is two-way set associative, i.e., it contains 8 sets of 2 entries each. Each entry consists of the following fields (cf Figure):

1. Branch Instruction Tag (BrIT) is the tag allowing the BTB to recognize which branch instruction is to be predicted or whose statistics have to be updated. We assume that all instructions are aligned on word boundaries, hence while an instruction address is 32 bits long the least significant 2 bits are always 0. The 3 bits needed to index the BTB will therefore be bits 4-2. This leaves the BrIT to be 27 bits long.
2. Prediction bits (Pr). This is a 2-bit field used for a 2-bit saturating counter (cf. H&P page 502 and Assignment #2, Problem 3).
3. Branch Target field (BrTa). This field is 30 bits long. It stores the target address for a branch predicted taken, minus the two rightmost bits since they are always 0.
4. Some mechanism to indicate the LRU entry in a set (LRU). When an entry has to be put in the BTB and the corresponding set is full, the replacement algorithm is as follows:
   - If one (and only one) of the 2 entries has a counter which predicts "not taken", this entry is replaced
   - If both entries predict "taken" or both predict "not taken", the LRU entry is replaced.

Inputs to the BTB are:

- An operation field (see below), called OP: 2 bits.
- The PC of the instruction to be predicted or of the branch instruction whose fields in the BTB must be updated, called PC: 32 bits
- The Branch Target Address of the branch instruction whose fields in the BTB must be updated, called BTIN: 32 bits

Outputs from the BTB are:

- BTB hit/miss : HIT (0 = miss, 1 = hit)
- BTB prediction TAK (0 = not taken, 1 = taken)
- Branch Target Address BTOUT : 32 bits

The operations are:

- No-op OP = 0; The need for this OP will become clear for the second part of the assignment
- Predict OP = 1; Check the entry corresponding to PC.
  The possible outputs for the triple (HIT, TAK, BTOUT) are ('x' means undefined)
  - (0,x,x)
  - (1,0,x)
  - (1,1,{BrTa,2'b00})
- Update not taken OP = 2;
  If the entry corresponding to PC exists in the BTB, "decrement" the saturating counter and set the LRU mechanism to "point" to the other entry in the set
  Else do nothing
- Update Taken OP = 3;
  If the entry corresponding to PC exists in the BTB, "increment" the saturating counter, set the LRU mechanism to "point" to the other entry in the set, and set BrTa to BTIN
  Else replace one entry (following the algorithm given above) with an entry BrIT (corresponding to PC), Pr (corresponding to "weak taken"), BrTa (corresponding to BTIN), and the LRU mechanism pointing to the other entry.

What you have to do

1. Write a Verilog program for the design of the BTB. You should write your program at the behavioral level. In the first design, assume that the BTB handles only one operation in a given clock period. Test your design with various sequences of operations.
   The module should be called *btbseq* and have parameters listed as below:
   module btbseq(clk,op,pc,btin,hit,tak,btout);
   Use the clock module (11 cycles per period) that was e-mailed for assignment #2.

A typical testing sequence would consist of a list of operations such as (I have not included the timings and clearly it's a bogus sequence since we have a taken branch and no jump to it!):

```
        1,32'H0000_0800,32'Hxxxx_xxxx \\_ is here for readability
                                      \\ should return (0,x,32'Hxxxx_xxxx)
        1,32'H0000_0804,32'Hxxxx_xxxx \\won't be updated; not a branch instruction
                                      \\ should return (0,x,32'Hxxxx_xxxx)
        3,32'H000_08000,32'H0000_1234 \\update taken
        1,32'H0000_0800,32'Hxxxx_xxxx \\should return (1,1,32'H0000_1234)
        1,32'H0000_0808,32'Hxxxx_xxxx \\should return (0,x,32'Hxxxx_xxxx)
        2,32'H0000_0808,32'Hxxxx_xxxx \\update not taken
        1,32'H0000_0808,32'Hxxxx_xxxx  \\should return (0,x,32'Hxxxx_xxxx)
etc..
```

Be sure to test cases where you predict taken and not-taken and where your update confirms or does not confirm your prediction. Also be sure to test the LRU replacement, the setting of the saturating counters etc.

2. Comment your design by explaining what instructions can trigger the various outputs of "Predict" (OP = 1). Give reasons (that might be design choices or correctness necessities) why nothing is done when OP =2 and there is a BTB miss and why BrTa has to be set to BTIN when OP = 3 and the entry already exists.

3. Now assume that the BTB can handle concurrently one "Predict" and one "Update" operation in the same clock period. You can assume that an "Update" operation (if any) follows its corresponding "Predict" after 1 clock period (why?) and that there is no two consecutive "Predict" with the same PC (this is a reasonable assumption; why?). (Note: you don't have to answer the "why"s but it would not hurt to think about it.)

   Modify your first design so that your new design reflects the concurrency. **The module should be called *btbcon* and have parameters listed as below:**
   module btbcon(clk,op1,pc1,btin1,hit,tak,btout,op2,pc2,btin2);
   Note that **op1** is always 1 (corresponding to the ID stage of the pipeline) and **op2** is never equal to 1 (corresponding to the updating). The output will be the output corresponding to the first operation.

   A typical testing sequence would consist of a list of operations such as:

```
        1,32'H0000_0800,32'Hxxxx_xxxx,0,32'Hxxxx_xxxx,32'Hxxxx_xxxx
        1,32'H0000_0804,32'Hxxxx_xxxx,0,32'Hxxxx_xxxx,32'Hxxxx_xxxx
        1,32'H0000_0808,32'Hxxxx_xxxx,3,32'H0000_0804,32'H000_1234
etc...
```