LSU EE 7700-2 **Homework 1** Solution **Due: 20 October 2003**

For this assignment read Eric Rotenberg, Steve Bennett, and James E. Smith, "A trace cache microarchitecture and evaluation," *IEEE Transactions on Computers,* vol. 48, no. 2, pp. 111-120, February 1999. For authorized users the paper can be found at
`http://www.ece.lsu.edu/tca//s/roten-99.pdf`.

For the last problem look at Glenn Reinman, Brad Calder, and Todd Austin, "Optimizations enabled by a decoupled front-end architecture," *IEEE Transactions on Computers,* vol. 50, no. 4, pp. 338-355, April 2001. For authorized users the paper can be found at
`http://www.ece.lsu.edu/tca//s/reinman-01.pdf`. Concentrate on Section 5, up to and including 5.2.

Problem 1: In a system using a trace cache there is a possibility that a branch within a trace can be mispredicted.

Consider a system using the trace cache described in Rotenberg 99. The system has predicted a trace, trace A, which contains three branches. Trace A was found in the trace cache and its instructions are executing. The first of its three branches was correctly predicted but the second one was mispredicted by the next trace predictor.

(a) Write a code fragment in MIPS or some other RISC ISA. Based on that code fragment make up a trace ID for trace A. (Make up instruction addresses.) *Hint: This part is straightforward and has nothing to do with the misprediction.*

The code is shown below. The paper gives no limit on the number of branches per trace and so a trace id could have room for sixteen branch outcomes but that is more than would almost ever be needed. Instead, a limit of four was chosen. The x in the last position of the trace ID indicates that the trace only contains three branches; the x can be either a 0 or a 1 (but it can't really be an x otherwise more than one bit would be needed to encode each branch). Note that the trace ends with a jump which is how it could be kept short and still only contain three branches.

```
# Solution:
0x1000:
 addi $s0, $0, 0
 beq $t1, $t2, SKIP1  # Predicted taken, correct.
 nop
 addi $s0, $s0, 1
SKIP1: # 0x1010
 beq $t3, $t4, SKIP2  # Predicted taken, wrong.
 nop
 addi $s1, $s1, 1
SKIP2: # 0x101c
 beq $t5, $t6, SKIP3  # Predicted not taken, correct.
 nop
 jr $ra
 nop
SKIP3:

# Trace ID:  {0x1000,1,1,0,X} (Parts shown separately.)
# Trace ID:  Binary encoding:  0000 0000 0000 0000 0001 0000 0000 00 110x
```

(b) When the mispredicted branch is resolved there are two ways in which a prediction for the third branch can be obtained. What are they?

The next-trace predictor can provide an alternate trace id. If that alternate trace ID matches with the current one up to the mispredicted branch, then the prediction for the third branch can be taken from the alternate trace ID.

For example, if the alternate trace ID were `0x1000,1,0,1,X` then it could be used to predict the third branch, and that prediction is taken. The following alternate trace IDs could not be used because it mispredicts the first branch: `0x1000,0,0,1,X`.

If the alternate trace ID cannot be used then the system in the paper uses a bimodal predictor (PC-indexed 2-bit counter) to predict the third branch.

(c) Adding to your code fragment if necessary, show a trace ID for the trace that is constructed after the mispredicted branch resolves. Use hierarchical sequencing, as did the system in the paper.

With hierarchical sequencing a new trace is constructed, starting with the portion of the old trace up to the misprediction. A possible new trace ID would be `0x1000,1,0,0,X`.

(d) Show the trace ID for the trace that would be constructed if hierarchical sequencing had not been used. (See Section 2.5.)

Without hierarchical sequencing the new trace starts after the mispredicted branch. A possible new trace ID would be `0x1018,1,0,X,X`.

Problem 2: Estimate the amount of memory (in bytes) used by the correlated prediction table, the main table used by the next-trace predictor.

According to the paper the table has 2^{16} entries. Each entry holds a trace id, a 2-bit counter, and an alternate trace id. Assuming four branches per trace, and aligned instructions in a 32-bit address space, that would be 34 bits for the trace id. Since a misprediction of a branch within a trace won't change the address of the first instruction in the trace, the alternate trace id need only be 4 bits. The total size of an entry is then $2 + 34 + 4$, the size of the table is 40×2^{16} bits or about 328 kB.

Problem 3: There is an important difference in prediction capabilities between the SEQ.n model of Rotenberg 99 and the superblock predictor described in Reinman 01. What is it? Which is better? Assume that the superblocks that are being predicted in SEQ.n (Rotenberg) and the FTB (Reinman) (See the reference at the beginning of the assignment.) When looking at Reinman 01 concentrate on Section 5, up to and including 5.2.

The two predictors are similar in that they predict a superblock, which does not contain taken branches. Both predictors use a global history of branches (for trace cache they are in the trace ids) and so have similar information available.

Consider a block A ending with a branch that is not highly biased but that can be accurately predicted. If it is not taken it is followed by block B, if it is taken it is followed by block T.

Since the branch in A can be accurately predicted seq.n would predict trace AB when the branch is not taken and A when the branch is taken. Since A is not highly biased the FTB would predict block A in one cycle and then block B in the next cycle. (It could only predict AB in one cycle if A were highly biased not taken.) So the FTB would not be able to predict as many blocks per cycle.