EE 4720 **Homework 5** Solution **Due: 24 April 2000**

Problem 1: The code below is run on three machines each using a slightly different one-level branch predictor. Each machine's branch predictor uses a 1024-entry BHT. The first machine uses 2-bit saturating counters (as described in class), the second machine uses the 2-bit prediction scheme illustrated in Figure 4.13 of the text, and the third uses a 3-bit saturating counter. (The scheme illustrated in Figure 4.13 uses two bits, but it's not a saturating counter.) Find the prediction accuracy for each scheme on each branch instruction for a large number of iterations.

```
! r1 is initially set to a large value.
LOOP1:
 subi r1, r1, #1
 beqz r1, EXIT
 andi r2, r1, #6
 bneq r2, SKIP1
 add  r3, r3, #1
SKIP1:
 andi r2, r1, #2
 bneq r2, SKIP2
 add  r3, r3, #1
SKIP2:
 j LOOP1
EXIT:
```

Solution on next page.

The branch outcomes are shown below, the horizontal position indicates the order in which the branches are executed (the distance between them is not drawn to scale). Beneath the branch outcomes are the values of the branch counter (or state) for the prediction scheme indicated. An **x** appears beneath a branch outcome if it was mispredicted. If the prediction accuracy for a branch and scheme depends on how the counter is initialized then counter values and outcomes are shown for several possible initializations. The prediction scheme illustrated in Figure 4.13 is called *2-bit state*; the states are numbered 0 through 3, with 0 being the state illustrated in the lower right, 1 lower left, 2 upper right, and 3 upper left. At least enough outcomes are shown to reveal a repeating sequence.

```
LOOP1:
 subi r1, r1, #1
 beqz r1, EXIT      N   N   N   N   N   N   N   N   N
 ! 2-bit counter  3 x 2 x 1   0   0   0   0   0   0   0 ... 100%
 ! 2-bit state    3 x 2 x 0   0   0   0   0   0   0   0 ... 100%
 ! 3-bit counter  7 x 6 x 5 x 4 x 3   2   1   0   0   0 ... 100%
 andi r2, r1, #6
 bneq r2, SKIP1      N   N   T   T   T   T   T   T   N   N   T   T   T
 ! 2-bit counter   3 x 2 x 1 x 2   3   3   3   3   3 x 2 x 1 x 2   3   3 ... 5/8 = 62.5%
 ! 2-bit state     3 x 2 x 0 x 1 x 3   3   3   3   3 x 2 x 0 x 1 x 3   3 ... 4/8 = 50%
 ! 3-bit counter   7 x 6 x 5   6   7   7   7   7   7 x 6 x 5   6   7   7 ... 6/8 = 75%

 add  r3, r3, #1
SKIP1:
 andi r2, r1, #2
 bneq r2, SKIP2      N   N   T   T   N   N   T   T   N
 ! 2-bit counter   3 x 2 x 1 x 2   3 x 2 x 1 x 2   3 ... 1/4 = 25%
 ! 2-bit counter   0   0   0 x 1 x 2 x 1   0 x 1 x 2 ... 1/4 = 25%
 ! 2-bit state     3 x 2 x 0 x 1 x 3 x 2 x 0 x 1 x 3 ... 0/4 = 0%
 ! 2-bit state     0   0   0 x 1 x 3 x 2 x 0 x 1 x 3 ... 0/4 = 0%
 ! 3-bit counter   7 x 6 x 5   6   7 ...                 2/4 = 50%
 ! 3-bit counter   5 x 4 x 3 x 4   5 ...                 1/4 = 25%
 ! 3-bit counter   4 x 3   2 x 3 x 4 ...                 1/4 = 25%
 ! 3-bit counter   0   0   0 x 1 x 2   1   0 x 1 x 2 ... 2/4 = 50%

 add  r3, r3, #1
SKIP2:
 j LOOP1
EXIT:
```

Problem 2: What is the largest BHT size (number of entries) for which there will be collisions between at least two branches in the code above?

For there to be a collision the BHT address must be the same for at least two branches. If the BHT had just one entry it would use zero address bits and so all branches would have the same BHT address. But the problem asked for the maximum table size. The address of each instruction (using a made-up value of LOOP1) is shown below next to the instruction. The low-order bits of branch instruction addresses (skipping alignment) are shown to the left. If the BHT used three address bits then these would be the addresses and there would be no collisions. If two bits were used (the two least significant bits) the three address would still be different. If one address bit were used then the first two branches would be indistinguishable, there'd be collisions. So the answer is two entries.

Note that the BHT address is constructed using the address (PC) of the branch instruction, **not** the branch target address.

```
! r1 is initially set to a large value.
       LOOP1: 0x1000
        0x1000: subi r1, r1, #1
 001     0x1004: beqz r1, EXIT
         0x1008: andi r2, r1, #6
 011     0x100c: bneq r2, SKIP1
         0x1010: add  r3, r3, #1
       SKIP1:
         0x1014: andi r2, r1, #2
 110     0x1018: bneq r2, SKIP2
         0x101c: add  r3, r3, #1
        SKIP2:
         0x1020: j LOOP1
        EXIT:
```

Problem 3: The program below runs on a system using a gselect branch predictor with a 14-bit branch history and a 2^{22}-entry BHT.

Show the value of the global branch history just before executing each branch after a large number of iterations. (The branch can be taken or not taken.) Also show the address used to index (lookup the value in) the BHT.

Determine the prediction accuracy of each branch assuming no collisions in the BHT.

```
 ! r2 is initially set to a large value.
 add r1, r0, r0
LOOP1: ! LOOP1 = 0x1000

 addi r1, r1, #2
LOOP2: ! LOOP2 = 0x1080
 subi r1, r1, #1
 bneq r1, LOOP2
A: ...  ! Nonbranch instructions.
 addi r1, r1, #3
LOOP3:   LOOP3 = 0x1100
 subi r1, r1, #1
 bneq r1, LOOP3
B: ...  ! Nonbranch instructions.
 subi r2, r2, #1
LINE:  ! LINE = 0x1180
 bneq r2, LOOP1
```

The branch outcomes are shown in the diagram below. Each branch outcome appears twice, once in the line labeled "Global history," and once in the line holding the corresponding branch. The global branch history at a particular cycle consists of the last 14 branch outcomes.

```
 ! r2 is initially set to a large value.
LOOP1: ! LOOP1 = 0x1000
 ! Cycle:                                a  b                                x     y        z
 ! Global history:       T  N  T  T  N  T  T  N  T  T  N  T  T  N  T  T  N  T  T  N  T  T  N  T
 addi r1, r1, #2
LOOP2: ! LOOP2 = 0x1080
 subi r1, r1, #1
 bneq r1, LOOP2          T  N              T  N              T  N              T  N
A: ...
 addi r1, r1, #3
LOOP3:   LOOP3 = 0x1100
 subi r1, r1, #1
 bneq r1, LOOP3                T  T  N           T  T  N           T  T  N           T  T  N
B: ...
 subi r2, r2, #1
LINE:  ! LINE = 0x1180
 bneq r2, LOOP1                           T                 T                 T                 T
```

For the first branch the global history at cycle `x` (see diagram above) is NTTNTTNTTNTTNT. The address of the first branch is 0x1084. The BHT address is constructed by concatenating the branch history with $22 - 14 = 8$ bits of the branch address: $00100001_2 : 01101101101101_2$, where 00100001_2 is the 8 low bits of 0x1084 skipping alignment and 01101101101101_2 is the binary representation of the branch history (obtained by changing "N"s to 0 and "T"s to 1), and ":" is a concatenation operator.

For the second branch the global history at cycle `y` is TNTTNTTNTTNTTN and the BHT address is $01000001_2 : 10110110110110_2$.

For the third branch the global history at cycle `z` is TNTTNTTNTTNTTN and the BHT address is $01100000_2 : 10110110110110_2$.

Because the outcome is always the same (before the outer loop is exited) the last branch will be predicted with 100% accuracy.

Consider the first branch at positions `a` and `b`. Position `a` is at the first iteration of LOOP2 and position `b` is at the second (last) iteration. As can be seen the global history is the same whenever execution is at the first iteration of LOOP2 (except for the first few iterations of the outer loop). The same holds for the second iteration.

At position `a` the last two outcomes in the global history are **NT**, at position `b` the last two outcomes are **TT**, and so different BHT entries will be used. At the first iteration the branch is always taken so that BHT entry will saturate at 3, at the second iteration the branch is never taken so that entry will saturate at 0; both branches will be predicted perfectly (after warmup). Here the global branch history holds 14 outcomes, if it held only one outcome the same BHT entry would be used for both iterations and prediction accuracy would suffer.

A similar argument holds for the second branch. Therefore the branch prediction accuracy will approach 100% for a large number of iterations.

Problem 4: Suppose the problem above ran on a gshare branch predictor with a 10-bit branch history and a 2^{10}-entry BHT. Determine addresses for LOOP1, LOOP2, LOOP3, and LINE for which there would be collisions in the BHT after a large number of iterations. (Please retain program order.)

The global history at positions `y` and `z` are the same. Therefore if the index part of the address of the second and third branches were the same the same BHT entry would be used. Keep LOOP3 at 0x1100 (and the second branch at 0x1104) and change LINE to $1104_{16} + 2^{10+2} = 2104_{16}$.