CSE 560 – Practice Problem Set 4 Solution

1. In this question, you will examine several different schemes for branch prediction, using the following code sequence for a simple load-store ISA with no branch delay slot (i.e., the branch instruction effects the immediately following instruction):

```
                        addi    r2 ← r0, #45     ; initialize r2 to 101101, binary
                        addi    r3 ← r1, #6      ; initialize r6 to 6, decimal
                        addi    r4 ← #10000      ; initialize r4 to a big number
                top:    andi    r1 ← r2, #1      ; extract the low-order bit of r2
PC1 →                   bnez    r1, skip1        ;branch if the bit is set
                        xor     r0 ← r0, r0      ; dummy instruction
                skip1:  srli    r2 ← r1, #1      ; shift the pattern in r2
                        subi    r3 ← r3, 1       ; decrement r3
PC2 →                   bnez    r3, skip2
                        addi    r2 ← r0, #45     ; reinitialize pattern
                        addi    r3 ← r0, #6
                skip2:  subi    r4 ← r4, 1       ; decrement loop counter
PC3 →                   bnez    r4, top
```

   This sequence contains 3 branches, labeled PC1, PC2, and PC3. The following are the steady-state taken/not-taken patterns for each of these branches (T indicates taken, N indicates not-taken):

```
PC1:   TNTTNT TNTTNT …
PC2:   TTTTTN TTTTTN …
PC3:   TTTTTT TTTTTT …
```

   Here is one branch predictor structure:

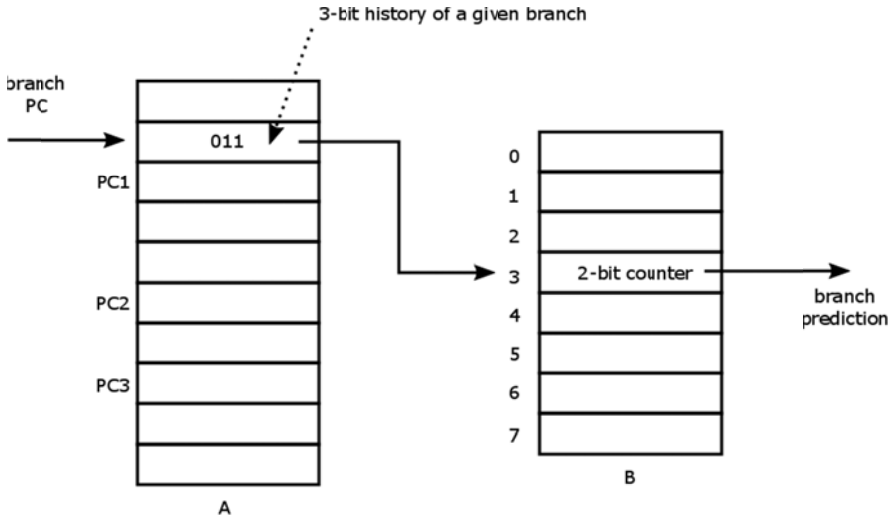
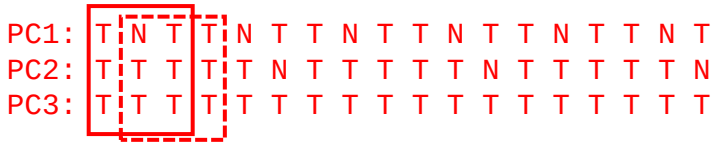


Table A is indexed by the program counter. Table B contains two-bit saturating up-down counters as described in class (state values N, n, t, T); they predict taken when the counter value is t or T. You can assume that PC1, PC2, and PC3 map to distinct slots in Table A.

(a) What is the prediction success rate (that is, the ratio of correctly predicted branches to total branches) for each of the three branches (PC1, PC2, and PC3) using this predictor, in steady-state, after the loop has been running for many iterations? If you want, you may assume that all counters and histories are initialized to zero, but this should not impact your answer.

The correct answer is:
PC1: 100%
PC2: 67%
PC3: 100%

To determine the above, we first determine the steady-state prediction of the relevant counters in table B. The following is the history of the branches:



Imagine a sliding window, illustrated by the box above. At some point during steady-state operation, the branch history for PC1 will be TNT, for PC2 will be TTT, and for PC3 TTT. This is illustrated by the solid box, above. At this point, the next set of branch patterns is given by the next column of the chart, or T for PC1, T for PC2, and T for PC3. So we build a table that holds the branch behavior for each entry in Table B. For example, PC1's history is TNT, and the branch is taken, so we put a T in entry 101 (decimal 5) of the table, etc.:

000	
001	
010	
011	
100	
101	T
110	
111	TT

We now slide the window one branch to the right, indicated by the dotted box, and update the table again. This continues until we hit a repeating pattern (after we enter 18 branches into the table). The result is shown below, with the repeat pattern in italics:

000	
001	
010	
011	NNT *NNT* …
100	
101	TTT *TTT* …
110	TTT *TTT* …
111	TTTTNTTTT *TTTTNTTTT* …

We now imagine each of these patterns being fed into a 2-bit saturating counter, and we observe that counter 011 will always predict not-taken, counter 101 will predict taken, counter 110 will predict taken, and counter 111 will predict taken in steady-state.

Going back to the original branch patterns, we see that PC1 goes through the patterns TNT, NTT, TTN, with next-branch behavior T, N, T, respectively. These correspond to counters 101, 011, and 110, which predict T, N, and T, respectively. These predictions match the branch behavior exactly, so PC1 is predicted with 100% accuracy.

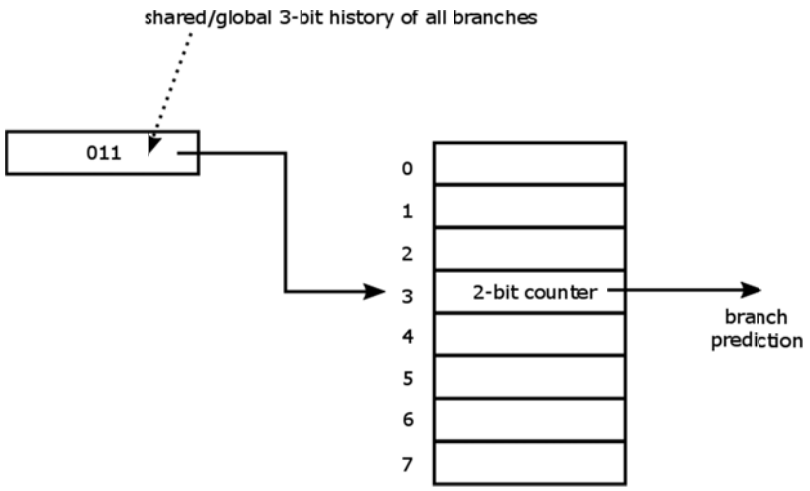
PC2 goes through the states TTT, TTT, TTT, TTN, TNT, and NTT, with next-branch behavior T, T, N, T, T, T. The states correspond to entries 111, 111, 111, 110, 101, and 011, and thus predictions T, T, T, T, T, and N respectively. The third and sixth predictions don't match the actual branch behavior, so 2 out of 6 branches are mispredicted, so PC2 has a prediction accuracy of 4/6 or 67%.

Finally, PC3 only has one pattern, TTT, and the next-branch behavior is always T. It is thus predicted correctly 100% of the time, since entry 111 predicts T.

(b) What is the overall prediction rate for the given code sequence on this predictor?

Above, we saw that the branch pattern repeats every eighteen branches. Of those eighteen, two are mispredicted (the two mispredicts of PC2). Thus, the overall prediction rate is 16/18, or 89%.

(c) Now let's consider another approach to branch prediction. Below is the structure of a predictor that uses 3 global history bits and a table of 2-bit saturating up-down counters.



What is the overall prediction rate for the given code sequence on this predictor?

We proceed as in part (a), but now we look at a global window. Here is the global branch pattern:

T T T N T T T T T T T T N T T T N T

This time, our sliding history window (the solid box) only covers three branches. In this case, with a global pattern of TTT, the next branch encountered is not taken. So we can build another table as in part (a); the dotted box shows how the window advances from the first position. After the pattern repeats (18 branches), the table looks like:

000	
001	
010	
011	TTT *TTT* …
100	
101	TTT *TTT* …
110	TTT *TTT* …
111	NTTTTTNNT *NTTTTTNNT* …

Counters 011, 101, and 110 will saturate to T. Counter 111 is trickier. Let's assume that the counter starts off with a value of 0 (i.e., strong predict-not-taken, or N). Other starting values will produce the same steady-state prediction, an exercise for the reader. Then, the counter evolves as follows:

Counter value:	N	N	n	t	T	T	T	t	n	t	n	t	T	T	T	T	t	n	t	n	t	…
Prediction:	N	N	N	T	T	T	T	T	N	T	N	T	T	T	T	T	T	N	T	N	T	…
Branch behavior:	N	T	T	T	T	T	N	N	T	N	T	T	T	T	T	N	N	T	N	T	T	…

So, what's the overall prediction rate? Of the 18 branches in the above table, those in entries 011, 101, and 110 will always predict correctly. This accounts for 9 of the branches. The other 9 branches correspond to those in the box above (for entry 111). We see that only four of those 9 entries are predicted correctly (the third, fourth, fifth, and sixth). So, out of the eighteen branches, 9 + 4 = 13 are predicted correctly, giving us a 72% overall prediction rate.

2. Use the following RISC code fragment:

```
loop:   load    r1, 0(r2)          (1)
        addi    r1 ← r1, #1        (2)
        store   0(r2), r1          (3)
        addi    r2 ← r2, #4        (4)
        sub     r4 ← r3, r2        (5)
        bnez    r4, loop           (6)
        next instruction           (7)
```

You may assume that the initial value of r3 is r2 + 396.

(a) Show the timing of this instruction sequence (i.e., draw a pipeline diagram) for our 5-stage pipeline without any forwarding or bypassing hardware but assuming a register read and a write in the same clock cycle "forwards" through the register file. Assume that the branch is handled by flushing the pipeline (use the notation **f** in the pipeline diagram). If all memory references hit in the cache, how many cycles does this loop take to execute?

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
(1)	F	D	X	M	W													
(2)		F	D	d*	d*	X	M	W										
(3)			F	p*	p*	D	d*	d*	X	M	W							
(4)						F	p*	p*	D	X	M	W						
(5)									F	D	d*	d*	X	M	W			
(6)										F	p*	p*	D	d*	d*	X	M	W
(7)													F	p*	p*	D	f*	
(1)																	F	D

17 – 1 cycles = 16 cycles to execute

(b) Show the timing of this instruction sequence for our 5-stage pipeline with normal forwarding and bypassing hardware. Assume that the branch is handled by predicting it as not taken. If all memory references hit in the cache, how many cycles does this loop take to execute?

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
(1)	F	D	X	M	W													
(2)		F	D	d*	X	M	W											
(3)			F	p*	D	X	M	W										
(4)					F	D	X	M	W									
(5)						F	D	X	M	W								
(6)							F	D	X	M	W							
(7)								F	D	f*								
(1)										F	D							

10 – 1 cycles = 9 cycles to execute