Department of Computer Science
Computer Architecture Qualifying Exam
Spring, 2007 Solutions

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no electronic aids** are allowed. Please note the following:

- show your work whenever appropriate. There can be no partial credit unless you show how you derived your answers
- be succinct. You may lose points for facts that, while true, are not relevant to the question at hand

1. (30 points) Consider a processor with the following two-level branch predictor:

   The first level predictor is a one-bit predictor, which always predicts a branch will be taken if it was taken on the last execution of the instruction. The first level predictor is initialized to branch-taken. Branches correctly predicted by this predictor have 0 cost.

   The second level predictor is a two-bit predictor using a saturated counter; its states are 00=strongly don't branch, 01=weakly don't branch, 10=weakly branch, and 11=strongly branch. This predictor is initialized to state 10. Predictions changed by this predictor will incur a 1 cycle cost.

   Finally, corrections made by the actual branch evaluation unit have a 2 cycle cost.

   These costs are not cumulative: if the first level predictor makes a correct prediction, the second level predictor incorrectly "corrects" it, but the branch evaluation unit validates the first level predictor, there is no cost.

   Now consider the following code:

```
i = 0;
do {
    j = 0;
    do {
        j = j + 1;
    } while (j < 10);
    i = i + 1;
} while (i < 10);
```

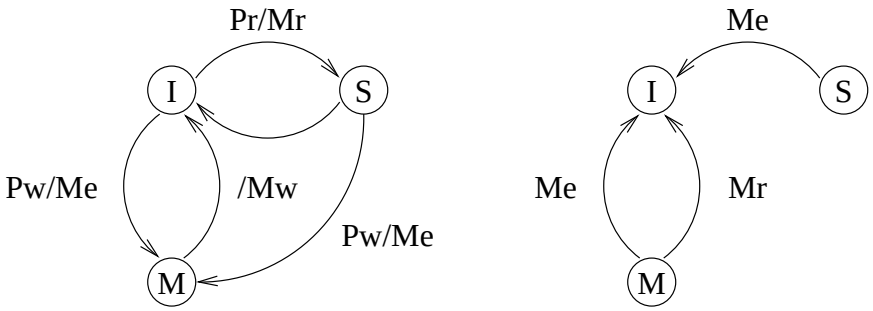
   Questions:

   (a) **What is the mean branch cost for the branch at the end of the outer loop?**
   This branch is executed a total of ten times, with values of i from 1 to 10. The first nine times it isn't taken, the tenth time it is. The first and second level predictors both predict correctly for the first nine iterations, and mispredict on the tenth (which costs two cycles). So, the mean branch cost is 2/10 or .2.

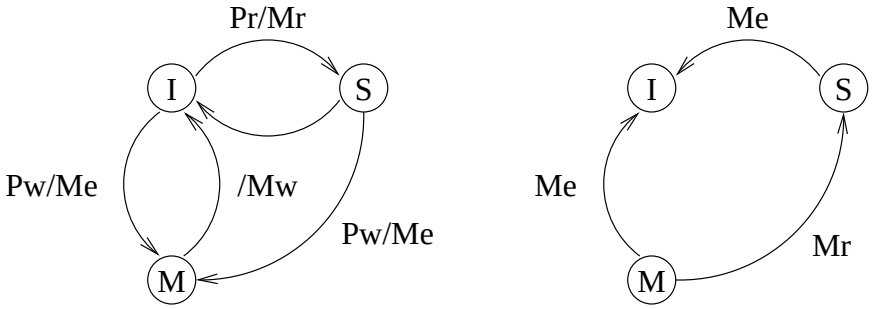
   (b) **What is the mean branch cost for the branch at the end of the inner loop?**
   This branch is executed ten times for every iteration of the outer loop. Here, we have to consider the first outer loop iteration separately from the remaining nine.
   On the first iteration of the outer loop, both predictors are correct for the first nine inner loop iterations, and both mispredict on the tenth. So on the first outer loop iteration, we have a cost of 2 in the ten iterations.
   On the remaining iterations of the outer loop, the first-level predictor will mispredict on the first inner loop iteration, both predictors will be correct for the next eight iterations, and both will mispredict on the final iteration, for a cost of 3 in ten branches.
   *The total cost will be 2 + 9*3 in 100 iterations, or .29*

   (c) **What is the average branch cost for all branches in the program?**
   There are a total of 110 branch instructions, and a total cost of 31. So the mean cost is 31/110, or .28

2. (25 points) Figure 1 shows state diagrams for two different variations on an MSI snoopy cache coherence protocol.



(a)



(b)

Figure 1: Cache Coherence State Transitions

For clarity, the figures show state transitions in response to CPU operations separately from transitions in response to snooped memory operations. The responses to CPU operations are on the left, and the responses to memory operations are on the right.

The two CPU operations are Pr (processor read) and Pw (processor write).

The three memory operations are Mr (memory read), Mw (memory write), and Me (memory read exclusive).

The transitions show the state change that occurs in response to processor (or memory) operations, and the memory operations which may occur as a result. For instance, the arc from state I to state S labelled Pr/Mr in figure 1(a) says that if a cache block is Invalid, and the processor executes a read operation, the cache block will change to state S and a memory read operation will occur as a result.

The arcs with no processor operation are cache flushes. A block in state S can transition to state I without writing to memory (so these arcs don't show any label at all); a block in state M must write its data to memory if it is to transition to state I.

Questions:

(a) **Suppose processor P1 performs a test-and-set-bit instruction. In this instruction, the processor reads a memory location to determine its old contents, and then writes new contents to it. If the memory location was previously not valid in any cache block, what will its state be in P1's cache block following the operation?**

It reads the location, causing a state transition to S. Then it writes it, changing the state to M. So it ends up in state M.

(b) **Now, following the operation in part (a), suppose P2 performs the same instruction on the same memory location. What will the block's state be in the two processor's caches now? Will the variation chosen make any difference to the final state?**

P2 reads it, so P2 has it in state S (in the top variation, it goes to state I in P1, while in the bottom it goes to state S in P1). Now P2 writes it, so it goes to state M in P2 and state I in P1.

(c) **Which variant will require fewer memory operations in part (b)?**

It'll be the same. Urrk – there was an error in the question...

3. (25 points) Suppose you have a computer system with the following characteristics:

   The virtual memory system uses a 4K page size. Its TLB has 16 sets (we don't need to know how associative it is, nor anything else about the virtual memory system, for this question).

   The first-level cache subsystem is a 32K, 8-way set-associative cache with a 16 byte line size. It uses physical addresses.

   An attempt is made to access address `0xabcd1234`.

   Questions:

   (a) **What will the page offset field for this address contain?**

   Since it has a 4K page size, the page offset field is the least significant 12 bits of the address (as 2^{12} is 4K). So it contains `0x234`.

   (b) **Which set in the TLB will this address map to?**

   As the TLB has 16 sets, four bits are used to select one of them (both students taking the exam wanted to know how many entries the TLB has, and how associative it was. That information wasn't necessary to solve the question: the width of the TLB set index field is given by the number of sets. The number of entries and the associativity is used to calculate the number of sets; I was actually trying to simplify the question by not making them calculate this!).

   The next four bits in the address are `0x1`

   (c) **What will the TLB tag have to be for us to have a TLB hit?**

   The remaining bits in the address have to match the TLB tag. So that's `0xabcd`.

   (d) **What will the address's cache offset field contain?**

   The line size is sixteen bytes, so the offset field is the least significant four bits. So it contains `0x4`.

   (e) **Which set in the cache will this address map to?**

   The cache is 32K and has a 16 byte line, so it has 2K lines. Since it's 8-way set-associative, it has 256 sets. This means there are eight bits in the set index field; it contains `0x23`.

   (f) **Assume we had a TLB hit in part (b), and that the page frame number is `0x1247`. What would the cache tag have to be for us to have a cache hit?**

   `0x1247`.

   (g) **Would it have been possible for hardware to solve parts (a) and (c) in parallel, and parts (b) and (d) in parallel (*i.e.*, to perform the TLB lookup and cache lookup simultaneously)?**

   Yes. None of the bits used in the cache lookup are changed by the virtual/physical translation.

4. (20 points) The following is an eight bit data word with ECC and parity: `0100101111010` The bits are in the order $M_8M_7M_6M_5M_4M_3M_2M_1C_8C_4C_2C_1P$.

   Question: is this word correct, does it have a one-bit error, or does it have a two-bit error? If it has a one-bit error, correct it.

Name	*Address*	*Contents*	C_8	C_4	C_2	C_1
M_8	1100	0	0	0		
M_7	1011	1	1		1	1
M_6	1010	0	0		0	
M_5	1001	0	0			0
C_8	1000	1	1			
M_4	0111	1		1	1	1
M_3	0110	0		0	0	
M_2	0101	1		1		1
C_4	0100	1		1		
M_1	0011	1			1	1
C_2	0010	0			0	
C_1	0001	1				1
P	0000	0				
Parities		**1**	**0**	**1**	**1**	**1**

*As the global parity is incorrect, there is a one-bit error. The address of the error is 0111, so the corrected data is **`0100`**`0`**`01111010`***