## **Computer Sciences Department**

## **University of Wisconsin-Madison**

## **CS/ECE 552 – Introduction to Computer Architecture**

## **In-Class Exercise (03/12)**

**Answers to all questions should be uploaded on Canvas.**

1. [1 point] You are building a computer system around a processor with in-order execution that runs at 1 GHz and has a CPI of 1, excluding memory accesses. The only instructions that read or write data from/to memory are loads (20% of all instructions) and stores (5% of all instructions).

The memory system for this computer has a split L1 cache. Both the I-cache and the D-cache hold 32 KB each. The I-cache has a 2% miss rate and 64-byte blocks, and the D-cache has a 5% miss rate and 64-byte blocks. The hit time for both the I-cache and the D-cache is 1 ns.

The L2 cache is a unified cache with a total size of 512 KB and a block size of 64-bytes. The hit time of the L2 cache is 15ns for both read hits and write hits. Tag comparison for hit/miss is included in the 15ns in all cases, do not add hit time to miss time on a miss. The local hit rate of the L2 cache is 80%. On a L2 miss, the penalty for both reads and writes is 100 cycles.

Calculate the value of the AMAT *only* *for data reads* in ns (nanoseconds).

0.98\*1ns+0.02\*0.8\*15ns+0.02\*0.2\*100\*1ns+0.2\*0.95\*1ns+0.2\*0.05\*15ns+0.2\*0.05\*0.2\*100\*1ns = 2.13ns

2. [4 points] Assume you have a direct-mapped cache that uses 8-bit addresses in nibble notation (i.e., base 4), a 32B cache, 4B blocks with the initial cache contents as follows in the *Cache Contents* column and that the initial blocks are accessed in increasing order, and the memory accesses as follows in the *Address* column. Fill in the remaining rows of the table for *Cache Contents*, and in the *Outcome* column specify whether the outcome is a Hit or Miss. When the access is a miss, specify whether the miss is compulsory, capacity, or conflict. We have filled in the first row for you.

|  |  |  |
| --- | --- | --- |
| **Cache Contents** | **Address** | **Outcome** |
| 0000, 0010, 0020, 0030, 0100, 0110, 0120, 0130 | 3020 | Miss (compulsory) |
| 0000, 0010, **3020**, 0030, 0100, 0110, 0120, 0130 | 3030 | Miss |
| 0000 0010 3020 3030 0100 0110 0120 0130 | 2100 | Miss |
| 0000 0010 3020 3030 2100 0110 0120 0130 | 0012 | hit |
| 0000 0010 3020 3030 2100 0110 0120 0130 | 0020 | Miss |
| 0000 0010 0020 3030 2100 0110 0120 0130 | 0030 | Miss |
| 0000 0010 0020 0030 2100 0110 0120 0130 | 0110 | Hit |
| 0000 0010 0020 0030 2100 0110 0120 0130 | 0100 | Hit |
| 0000 0010 0020 0030 0100 0110 0120 0130 | 2100 | Miss |
| 0000 0010 0020 0030 2100 0110 0120 0130 | 3020 | miss |

3. [5 points] Assume a 32-bit memory address space and a 4-way set-associative cache with an 8-bit set index and 8-bit offset, defined as follows:

**assign tag[15:0] = address[31:16];**

**assign set[7:0] = address[15:8];**

**assign offset[7:0] = address[7:0];**

Each entry in the table below describes a potential bug in the cache indexing (all lines in the table are Verilog). Specify which of the bugs would still yield a correct cache design; **each bug is applied individually after the assign statements above**. For example, the first bug yields **set[7] = address[15] | address[31]** and **set[6:0] = address[14:8]**. We define a correct cache as satisfying both of the following requirements: 1) a block with address X must not be able to exist at multiple locations in the cache at any one time; and 2) a block with address X must not be confused with a different block with address Y. If a bug yields an incorrect design, give an example of how it breaks one (or both) of the above requirements.

|  |  |  |
| --- | --- | --- |
| **Bug** | **Correct?** | **If incorrect, give example** |
| set[7] |= address[31] | n | 0x80000000, 0x80008000 |
| set[7] ^= address[31] | Y |  |
| tag[0] &= address[15] | N | 0x80010000 80000000 |
| tag[0] ^= address[15] | Y |  |
| set[7] &= address[2] | N | 0x10008000 0x10008004 |
| set[7] ^= address[2] | N | 0x10000000 0x10000004 |
| set[0] |= address[9] | N | 0x80000200 0x80000300 |
| set[0] ^= address[9] | Y |  |
| tag[15] |= address[16] | N | 0x00010000 0x80010000 |
| tag[15] ^= address[16] | Y |  |
| tag[0] &= address[16] | Y |  |
| tag[0] ^= address[16] | n | 0x80010000 0x80000000 |