## **Computer Sciences Department**

## **University of Wisconsin-Madison**

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

## **In-Class Exercise (04/02)**

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

1. [5 points] Consider the **SECDED** **error correction code** described in the lecture video. The table below summarizes how to compute an 8-bit SECDED codeword (W8W7W6W5W4W3W2W1) from 4-bit data (b4b3b2b1). Note the 1-based indexing (i.e., least-significant bit is position 1).

|  |  |
| --- | --- |
| **8-Bit SECDED Codeword** | |
| **Bit** | **How to Compute** |
| W1 | b1⊕b2⊕b4 |
| W2 | b1⊕b3⊕b4 |
| W3 | b1 |
| W4 | b2⊕b3⊕b4 |
| W5 | b2 |
| W6 | b3 |
| W7 | b4 |
| W8 | W1⊕W2⊕W3⊕W4⊕W5⊕W6⊕W7 |

Consider a **32-bit MIPS instruction** in your program. When stored in memory, each 4-bit segment of the instruction is stored as an 8-bit SECDED codeword. After running the program for a long time, you observe the contents of memory and find the codewords to be as listed in the table below. **Some bits have flipped!** Fill in the table: for each 8-bit codeword, how many bits (if any) have flipped and what is the original (i.e., correct) 4-bit segment? The first three rows have been filled in for you.

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction Bits** | **8-Bit**  **SECDED Codeword**  (in binary) | **How Many Bits**  **Have Flipped?**  (0, 1, 2) | ***Correct* 4 Bits**  **of Instruction**  (in binary) |
| Bits 3-0 | 11100001 | 0 | 1100 |
| Bits 7-4 | 00010000 | 1 | 0000 |
| Bits 11-8 | 01111000 | 0 | 1110 |
| Bits 15-12 | 11111101 | 1 | 1111 |
| Bits 19-16 | 11101100 | 1 | 1001 |
| Bits 23-20 | 00110011 | 1 | 0010 |
| Bits 27-24 | 11100110 | 1 | 1101 |
| Bits 31-28 | 11010011 | 1 | 1010 |

What is the correct **32-bit MIPS assembly instruction**? N/A, if a 2-bit error is detected.

Sb R9, R9, #0xED0C

2. [5 points] Consider a processor with a virtually-indexed physically-tagged (VIPT) cache. You use the standard set indexing scheme on the virtual address, but the tag comes from the physical address as shown (assume the tag is the entire physical block address):

Virtual Address:

offset

set

Physical Address:

tag

Unfortunately, VIPT caches may be prone to **synonyms**. Recall that synonyms are virtual addresses that map to the same physical address. If unhandled, a physical block may exist in the cache at more than one location, which leads to incorrect behavior (i.e., one instance of the block becomes stale if the other instance is modified). Depending on the page size and cache configuration, synonyms may or may not be a problem. Assuming the standard set indexing scheme above, for each of the configurations in the table below, **specify whether or not synonyms can cause a problem.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Page  Size | VIPT Cache | | | Problem?  (Y/N)? |
|  | Capacity | Associativity | Block Size |
| (a) | 64KB | 256KB | 8-way | 32B | N |
| (b) | **32KB** | 256KB | 8-way | 32B | y |
| (c) | 64KB | **1MB** | 8-way | 32B | y |
| (d) | 64KB | 256KB | **2-way** | 32B | y |
| (e) | 64KB | 256KB | 8-way | **8B** | y |