CS/ECE 752 Spring 2025: HOMEWORK # 4
(Due by **12:30 PM** on **Monday, March 17**, via upload of PDF to Canvas)

Contact Rajesh Srivatsav (rsuresh6@wisc.edu)

1. Problem 1 (16 points, 3+4+6+3)

Consider a virtual memory system with the following properties:
•48 bit virtual address (byte addressable)
•4 KB pages
•37 bit physical addresses (byte addressable)
•128 GB physical memory (byte addressable)
•64 KB L1 cache that is 4-way set-associative, has a line size of 64 bytes, and is accessed with virtual addresses
•1 MB L2 cache that is 8-way set-associative, has a line size of 64 bytes, and is accessed with physical addresses
•a 64 entry, 4-way set-associative data TLB

(a) What would be the total size of the page table for each process on this machine, assuming that the valid, protection, dirty, and use bits take a total of 4 bits, and that all of the virtual pages are in use? (Assume that disk addresses are not stored in the page table).

(b) Why might it be infeasible to represent a page table as in (a)? Hierarchical page tables have multiple levels of page tables (segregated by groups of bits in the virtual address). How do hierarchical page tables help in this situation? What are the potential disadvantages of hierarchical page tables?

(c) Draw a diagram of the hardware in the memory system including the L1 and L2 caches and data TLB. Make sure you show how different fields of the address (i.e. which bits) are used to access the caches and the data TLB.

(d) Explain how a memory access proceeds through the memory system for each of the following scenarios: cache hit, cache miss, TLB hit, and TLB miss.

2. Problem 2 (10, 4+4+2 points)

In a virtually indexed, physically tagged cache, the cache set to search is selected using only bits of the virtual address, so virtual-to-physical address translation can proceed in parallel with reading tags for comparison. In the simplest design, the associativity of the cache is large enough so that the cache index and offset bits together fit entirely into the page offset bits. However, page size remains relatively fixed with architectures while cache size grows with semiconductor technology so this may require a high associativity.

(a) If a processor has 4KB pages and a 64KB level 1 cache, what is the minimum associativity required to use the simple virtually-indexed, physically tagged optimization?

(b) Suppose a cache simply forms a longer index using a few of the least significant bits from the virtual page number. Describe a page table and access pattern where this cache will return incorrect data.

(c) Does the MIPS R10000 have problems with synonyms or homonyms? If so, how does it deal with them? [Refer to the Yeager Micro ‘96 paper on MIPS R10K that you had reviewed while answering the question]

3. Problem 3 (6 points)

A cache may use a write buffer to reduce the write latency and a victim cache (or victim buffer) to hold recently evicted blocks. Would there be any advantages to combining the two into a single piece of hardware? Would there be any disadvantages?

4. Problem 4 (18 points, 4+4+4+6)

Consider the following piece of code that transposes a 64x64 integer matrix A, adds it to a 64x64 integer matrix B, and stores the result into a 64x64 integer matrix C. The matrices are stored in row major order, i.e., consecutive elements of a row are in consecutive memory addresses.

```
for (int i = 0; i < 64; i++) {
        for (int j = 0; j < 64; j++) {
                C[j][i] = A[i][j] + B[j][i]
        }
}
```

Assume that this is executed on a system with the following cache:

•fully associative
•size 32KB
•no pre-fetching
•block size of 64 bytes
•write allocate and write back
•LRU replacement policy

Assume that the size of an integer is 4 bytes.

(a) Determine the number of cache misses while executing this piece of code. Do this separately for both read and write misses.

(b) Now consider a simple two-block-lookahead prefetcher which operates as follows: On a cache miss, the prefetches the next two blocks, in addition to fetching the block which missed; on a cache hit the prefetcher does nothing. Determine the number of misses if this simple prefetcher is added to the caching operation. Do this separately for both read and write misses.

(c) A *stride prefetche*r fetches a block that is a stride (S) number of blocks away on a cache miss; on a cache hit the prefetcher does nothing. Would a stride prefetcher be effective here? If so, what stride (S) would the prefetcher use?

(d) Could the number of misses be reduced by a multi-way stream buffer strategy as described in the paper by Jouppi? If so, how? Describe the strategy and the number of misses that would result.