Mariam Sulakian
Professor Sorin
ECE / CS 250
Homework 5

Homework #5 – Caches and Memory

---

1. Why are write-back caches usually also write-allocate?
   WB and write-non-allocate cache is inefficient. WB usually also write-allocate because takes a long time to read DRAM (destroy data each time) so with this you can just read block out of cache and when you write into cache you use SRAM which is easier to write to. Purpose of WB is to hold one fresh copy of data so you can use space / save time in other places.

   A cache with a write-back policy and write-allocate reads an entire block (cacheline) from memory on a cache miss, may need to write dirty cacheline first. Any writes to memory need to be the entire cacheline since there is no way to distinguish which word was dirty with only a single dirty bit. Evictions of a dirty bit cause a write to memory. In write-back, the CPU writes only to cache, not main memory. The cache later writes to main memory when a block is evicted. In write-allocate, if the data is not in the cache then we allocate a cache line for new data.

2. Your L1 data cache has an access latency of 2ns, and your L2 cache has an access latency of 10ns. Assume that 90% of your L1 accesses are hits, and assume that 100% of your L2 accesses are hits. What is the average memory latency as seen by the processor core?
   a. L2 = hit time + %miss = 10ns + 0 = 10ns
   b. tavg = thit + %miss * miss penalty = 2ns + 0.10*(L2) = 2ns + 0.10*(10ns) = 3ns, so the **average memory latency as seen by the processor core is 3ns.**

3. You have a 64-bit machine and you bought 8GB of physical memory. Pages are 32KB. Assume GB = 2^30, KB = 2^10.

   Virtual Address = OS address length
   Physical Address = $\log_2$(RAM size) bits
   Offset = $\log_2$(page size) bits
   Virtual Page Number bits = Virtual Address - Offset
   Physical Page Number bits = Physical Address - Offset

   a. How many virtual pages do you have per process?
      - 64-bit machine → 64-bit address → can address 2^64 bytes in a byte addressable machine.

- Since the size of a page is 32KB (2^10 bits), the number of addressable pages is 2^64 / (32* 2^10) = 2^64 / (2^5 * 2^10) = 2^64 / 2^15 = **2^49 virtual pages**.

b. How many physical pages do you have?
   - Each page is 32KB = 2^15 bits.
   - Physical memory is 8GB = 8 * 2^30 = 2^3 * 2^30 = 2^33
   - 2^33/2^15 = **2^18 physical pages**.

c. In the translation from a virtual address to a physical address, how many bits of VPN are you mapping to how many bits of PPN?
   - Offset = log2(32KB) bits = log2(2^5 * 2^10) = log2(2^15) = 15 bits
   - VPN bits = 64 - 15 = 49 bits, PPN bits = 33 - 15 = 18 bits
   - Thus, you are mapping **49 bits of VPN** to **18 bits of PPN.**

d. How big does a page table entry (PTE) need to be to hold just a single PPN?
   - Assuming we're not adding a valid bit, protection bit, etc. to the calculated PPN, a **PTE would need to be 18 bits** to hold just a single PPN (each PPN is 18 bits). Approximately 3 bytes to hold a single PPN.

e. How many PTEs fit on a page, assuming PTEs are the size computed in part (d)? *Assume each PTE is 16 bits (Sorin's email).
   - Flat page tables = one entry for every page in the table
   - Page size / PTE = 32 kb / 2^1 = 2^15 / 2^1 = **2^14 bits**

f. How many pointers fit on a page?
   - Each pointer is 8 bytes (64 bits).
   - Page size / 8B = 32KB / 8B = 2^15 / 2^3 = 2^12.
   - **2^12 pointers per page.**

g. How big would a flat page table be for a single process, assuming PTEs are the size computed in part (c)? *Assume each PTE is 16 bits.
   - Each PTE = 16 = 2^4 bits
   - 2^45 PTEs per page
   - 2^45 * 2^4 * 2^1 = 2^50 bits, so a flat page table would be **2^50 bits** for a single process.

h. What are the virtual page offset bits for virtual address 25012? What are the physical page offset bits for virtual address 25012 after it has been translated?
   - Virtual POFS = Physical POFS = logs(page size) bits = log2(32KB) bits = log2(2^5 * 2^10) = log2(2^15) = 15 bits
   - 25012{10} = 000…110000110110100{2}, so the page offset bits are the following: **110000110110100**

i. Does a TLB miss always lead to a page fault? Why or why not?
   - No. A TLB miss leads to a page fault when the page is not present in memory and the program needs to transfer control to the OS to deal with it.

From here, you need to generate an exception and copy the page table entry to the TLB. Then, an appropriate replacement algorithm takes place if you need to evict an entry from the TLB. Otherwise, **if the TLB misses, the page may actually be present in memory and just needs to be mapped to the TLB. A page fault occurs when the PTE is neither in the TLB *nor* PT.**

4. In Java [or in C for up to 10 extra credit points], write a simulator of a cache.