Homework #2
Due: Tuesday, April 27th at 11:59pm

Your submission should be comprised of one item: a .pdf file containing answers to the questions. **Legible**, handwritten solutions are acceptable for undergraduate students (472). The handwritten solutions must be electronically scanned and submitted as a PDF file. Graduate students (572) must compose their solutions in MS Word, LaTeX, or some other word processor and submit the results as a PDF file.

1. (7 pts) Suppose the existence of a direct-mapped cache design that utilizes a 26-bit address. Assume that the addressable memory space is byte-indexed. The cache uses the following breakdown of address bits:

Tag	Index	Offset
25-8	7-5	4-0

   a. What is the cache block size (in bytes)?
   b. How many blocks can the cache contain?
   c. Including space for the valid bits, tags, and actual block data, how many bits would be required to implement this hypothetical cache?

2. (18 pts) Using the cache described in problem 1, assume the following byte-addressed cache references are recorded. Assume that the cache is initially empty. Also assume that accesses occurred from left to right (e.g. address 71 was requested, then address 65, followed by address 1927, etc).

Byte Address											
71	65	1927	244	585	225	900	1616	1410	81	590	1942

   a. For each memory access, indicate whether the activity generated a “hit” or a “miss”. If a memory access triggered a block eviction, be sure to indicate this.
   b. Create a table to show the final state of the cache including the value of each Index, Valid bit, and Tag. If a value is unknown, you may leave it blank.

3. (7 pts) Now imagine the existence of a 4-way set-associative cache (n=4) with a 26-bit address. The cache uses a “least recently used” replacement scheme. Assume that the addressable memory space is byte-indexed. The cache uses the following breakdown of address bits (same arrangement as problem #1):

Tag	Index	Offset
25-8	7-5	4-0

   a. What is the cache block size (in bytes)?
   b. How many blocks (in total) can the cache contain?
   c. Including space for the valid bits, tags, and actual block data, how many bits would be required to implement this hypothetical cache?

4. (21 pts) Using the cache described in problem 3, assume the following byte-addressed cache references are recorded. Assume that the cache is initially empty. As before, assume that accesses occurred from left to right (e.g. address 71 was requested, then address 65, followed by address 1927, etc).

Byte Address											
71	65	1927	244	585	225	900	1616	1410	81	590	1942

   a. For each memory access, indicate whether the activity generated a "hit" or a "miss". If a memory access triggered a block eviction, be sure to indicate this.
   b. Create a table to show the final state of the cache including the value of each Index, Valid bit, and Tag. If a value is unknown, you may leave it blank.

5. (4 pts) Suppose we are working with a processor that exhibits an average CPI (cycles per instruction) of 1.2, assuming all references hit the primary cache, and a clock speed of 3 GHz. Assume a main memory access time of 120 cycles (including all miss handling).
Imagine that the miss rate per instruction at the primary cache is 3.5%. How much faster will the system operate (on average) if we add a secondary cache with an access time (hit or miss) of 5 ns? Assume that the new secondary cache exhibits a miss rate of 0.8%.

6. This question is not graded, and does not need to be submitted. It is for your benefit to prepare yourself. As future work for this class it's likely that you will need to implement a model of a cache using C or C++ (your choice).
   a. If you are provided with the number of tag bits, index bits, and offset bits, how could you write a program to automatically calculate the answers for questions 1a through 1c?
   b. Can you envision a strategy to implement a cache model in C using **structs**?
   c. Are you more comfortable implementing a cache model using the available containers in C++?
   d. Take some time to envision your design. If the user provides you with a list of memory accesses, how could your design maintain your model cache?
   e. The end goal is that your code will be able to model the hardware caches that we discuss in class (complete with read accesses and writes).