1) Consider the following access pattern: A, B, C, A. Assume that A, B, and C are memory addresses each of which are in a different block of memory. Further, assume A, B and C are generated in a uniformly random way and that a "true" LRU replacement algorithm is used. Further, assume that any given block has an equal chance of being placed in either "way". To recieve credit you must show your work.

What is the probability that the second instance of "A" will be a hit if:

a) The cache has 4 lines and is direct-mapped [1].
b) The cache has 4 lines and is fully-associative [1].
c) The cache has 8 lines and is direct-mapped [1].
d) The cache has 4 lines and is two-way associative [2].

2) Consider a 4KB direct-mapped cache backed-up by a 1MB 4-way associative cache. The L2 is inclusive of the L1. Say the L1 has a Thit of 2 cycles, the L2 has a Thit of 10 cycles, and the L2 has a Tmiss of 200 cycles. Assume the Tmiss time does not include the time to figure out if we have a hit or not. Answer the following questions:

a) What would be the average memory access time if the L1 has a hit rate of 95% and the L2 has a hit rate of 50%? Assume accesses are only sent to the L2 if they miss in the L1 [2].

b) There is a proposal to add an L0 cache of 1KB. The L0 would have a 1 cycle access time and memory requests would only be passed on to the L1 if the L0 missed. The L1 would be inclusive (See wikipedia for a definition if needed) of the L0 and accesses that hit in the L1 and/or L2 before would continue to do so. What hit-rate would be needed to get a tie with the original configuration? (Use the hit-rate numbers from part "a" as needed.) [3]

c) In part b we said that the L1 is only accessed if the L0 misses. Let's assume we didn't do that and instead accessed both the L1 and L0 in parallel. What would be the disadvantages of doing this (or opportunity costs, however you want to look at it) as compared to the proposal in part b? [3]

3) Adding associativity to a cache usually improves miss rates. However, this is not always the case! Write C code for a loop that produces exactly one miss per loop iteration in a 64KB direct-mapped cache with 32 byte lines, but doesn’t even get a single hit in a 64KB fully-associative cache with 32 byte lines and LRU replacement. Write your code so that all variables are int (or arrays of int), and assume int is 4 bytes wide. Explain why your code gets one miss per iteration in the direct-mapped case, and no hits in the fully-associative case. [5]

4) Consider two L1 cache designs: Design A is a 64KB direct-mapped cache with 256-byte blocks, and Design B is a 64KB sub-blocked cache where the block size is 256 bytes and the sub-block size is 32 bytes. Assume memory addresses are 32 bits wide.

a) Draw a picture of the tag array for Design A. Indicate the number of tags, and width of each tag. Calculate the total storage (in bits) for the entire tag array. [4]

b) Repeat question 4a for Design B. Don’t forget to include the necessary sub-block valid bits! [4]

c) Suppose there is a 1MB array of 4-byte integers in main memory, aligned to a 256-byte boundary. If the processor walks the array from the first element to the last.

i) How many misses does each cache design incur? [1]

ii) How much total data is transferred from main memory by each design? [1]

d) Now suppose the processor walks through the array starting at index zero and accesses only every 64th element.

i) How many misses does each cache design incur? [1]

ii) How much total data is transferred from main memory by each design? [1]