homework 1 - intro to cache

due date: Tues., Sept 5

Please complete the following problems, preparing your submission in .pdf format and uploading to Carmen by 11:59pm on the stated due date. Be sure (especially if scanning handwritten work) that your submitted files are clear and legible.

1. A small 16-bit address space special-purpose processor may be equipped with one of two tiny direct-mapped caches, C1 or C2, each of which has a total capacity of 64 bytes. C1 has a blocksize of 4 bytes (which corresponds to the size of an integer on this system) while C2 has a blocksize of 16 bytes.
   (a) What are the cache parameters (m, C, B, E, S, t, s, b) for C1 and C2?
   (b) Consider that the following sequence of addresses are read, (where the size of the data path equals the blocksize): BA00, BA04, AA08, BA05, AA14, AA11, AA13, AA38, AA09, AA0B, BA04, AA2B, BA05, BA06, AA09, AA11.
   For each cache option, specify which references are hits and which are misses, and show the final data content of the cache.
   (c) Find a sequence of read address references for which C2 has more misses than C1.

2. Given a direct-mapped cache where t = 22, s = 8 and b = 4:
   (a) What is the address space of the processor?
   (b) What is the total usable cache size?
   (c) In what specific ways might your answer in (b) differ from the number of bits necessary to implement your cache?
   (d) How many distinct memory blocks will share each individual cache line?

3. Given an array of 1,000,000 integers of size 4 bytes, we wish to write a *for* loop to find the minimum valued integer in the array. Provide proper pseudo-code, along with all cache parameters for a 64-bit addressable processor which would attain a cache hit rate of at least 87.5%. (assume a cold cache at start)

4. In order to illustrate the concept of row-/column-major, provide two versions of C code for averaging the values stored in a 2-D array (matrix). In the first case, show a nested loop that would provide good locality of reference. Then, contrast that code segment with a similar nested loop that would provide poor locality of reference. Explain the differences in your code fragments, and the implications on cache performance.
   How would your results be impacted if you were to port your examples to Fortran?

5. Consider the in-class exercise from slide 26 of the "cache management" slides. Extend the for loop such that the *sum* computes the total of all values in *valueA[ ]*. Assuming that *sum* is a register variable:
   - assuming no other cache activity, what remains in the cache after completion of the extended for loop? (be specific, include set numbers when describing the element locations)
   - how many RAM (main memory) accesses are performed?