Designing a Cache Simulator

Input: Cache size in Bytes; Block size in Bytes; Associativity: 0 for fully associative, 1 for direct-mapped, 2/4/8/16/32 for set-associative; Replacement Policy: 0 for random, 1 for LRU, 2 for Pseudo-LRU; File containing memory traces (each entry containing 8-digit Hex-decimal number).

Output: (to be written into a file) Cache Size; Block Size; Type of Cache (fully associative/set-associative/direct-mapped); Replacement Policy; Number of Cache Accesses; Number of Read Accesses; Number of Write Accesses; Number of Cache Misses; Number of Compulsory Misses; Number of Capacity Misses; Number of Conflict Misses; Number of Read Misses; Number of Write Misses; Number of Dirty Blocks Evicted;

Methodology: Note that the cache size and the block size must be in the form 2^n, for some n. Given the sizes of the cache and block, compute the number of blocks. Define a struct *cacheBlock* consisting of *tag*, *valid* bit, *dirty* bit, and *pointer* to a cache block. Based on the associativity, compute the number of sets. Create a set using a linked-list of cache blocks. In the cache of LRU policy, whenever a cache block is accessed, bring it to the head of the corresponding linked-list, and a block at the end of the linked-list is considered for victim selection. In a set, if there is an invalid block, give priority for it to be selected as a victim block in the LRU policy. To compute the number of capacity misses, consider fully-associative cache only. Note that the most significant bit in an 8-digit Hex number indicates whether the request is a read or write (0 for read and 1 for write).