Palmer Robins

Programming #3 – Design Document

Overview: This program receives a configuration file, which gives values needed to set up a cache, and an input file, which contains word addresses that need to be accessed from the cache. The first step is to initialize the cache, where the syntax of the configuration file is checked to ensure that those values construct a valid cache. Then, the process of accessing values from the cache begins. As the add access function receives an address, it checks that addresses' possible location to see if it is present inside the cache. If it is not, we insert it, possibly evicting the block that was used least recently. Once this process is completed, we print out the cache state that is present upon completion of the iteration of the add access method. Then, once all addresses have been completed, the cache statistics are printed to the user: total misses, total hits, total accesses, and the miss rate. The program contains a cache class and a driver file named "driver.cpp".

Classes:

1. Cache: This class simulates the cache. It handles both direct-mapping caches as well as set-associative caches. It contains a number of vectors storing the sets within the cache, which in turn are stored in a vector of sets, representing the cache itself. Vectors were chosen to store sets (in the case of the cache vector) and block entries (in the case of the set vectors) because of the importance of location specific locations (indices) within them easily. The set vectors contain Block Entries, which are defined as a struct. Block Entries include the key information that is needed for each block in the cache, including the valid bit, tag, index, addresses, and lru bit(s). The cache class prints the state of the cache as each address is handled, and finally prints the cache statistics upon completion. The cache class also includes variables storing the information needed for cache statistics, including counters for hits, misses, and accesses. Also included as member variables are the values indicating the cache configuration (these are also printed back to the user upon initialization of the cache).
    a. Important Methods:
        i. addAccess() – As stated previously, this method handles accesses addresses from the cache, and places addresses into the cache should they not already be present within it.
        ii. hitChecker() – This method checks whether or not a hit was found for a given address – i.e. whether it was already in the cache.
        iii. updateLRUs() – This method is very important in the case of a set associative cache. It keeps track of the order of usage for blocks within a set, so that we know which one was the least recently used. This block will be evicted if needed so that we can insert a new block. Partially related to the update lru's method is the checkSetContents() method, which allows us to identify if block eviction is necessary – it returns whether or not there is open space in a set.