CSCE 3600

SPRING 2015

Programming Project

Simulation of L2 and L1 Cache Systems

GROUP 23

TEAM MEMBERS

Clement Cole

Elijah Adedapo

Christopher Cruz

Eston, Miranda

Scott Visser

Maintainance Report

Main function:

The main function consist of just an instanstication of the cachemanager class.

Cachemanager. Cpp

The cachemanager.cpp class consist of void inputErrorDetect(int, char\*), void get\_values(char\*) , void get\_write\_policy(string argument), an update\_age() method, write() method, search\_empty() method and several global variables in the global.h file.

Method of implementing the cache system.

We created a structure in the block\_t.h file for both the L1 cache and L1 cache. Then, after initiating the two instances of structures for the L1 and L2 cache, the next step is to actually create the cache itself. In order to create the cache for each level, we needed to know the BLOCK\_SIZE, CACHE\_SIZE and their associativity with the main memory.

To calculate the associativity we used the formula 🡺

W = sizeof(ADDRESS)\*8;

X = log2(CACHE\_SIZE)

Y=log2(BLOCK\_SIZE)

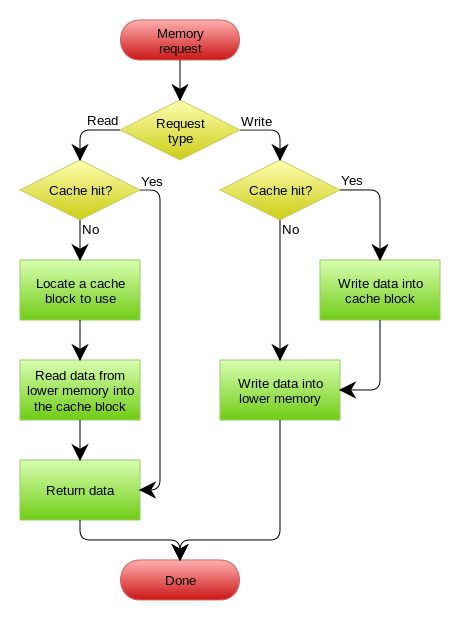
Z=log2(ASSOCIATIVITY)

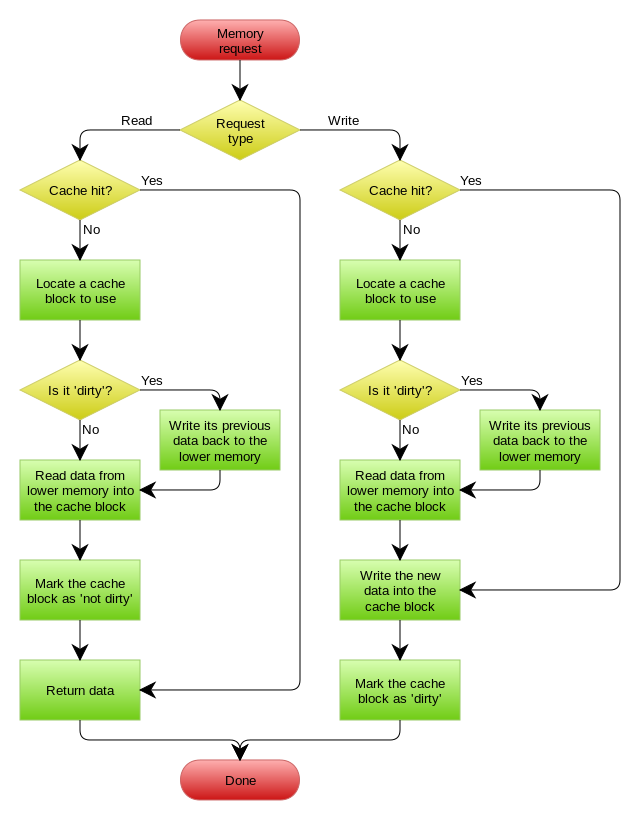
Where the Associativity =CACHE\_SIZE /BLOCK\_SIZE so L1\_SIZE/L2\_SIZE. Finally we use this idea to build the blocks of caches and dynamically store them as they grow or shrink.

The next step is to check for the hit and misses. If it’s a hit at the L1 level nothing happens. However, if it’s a miss on L1 level, that means we have to search for it one level below closer to the physical hard drive or store. If we find the datum at the L2 level, we used the swap function to make a copy of the tag as well as the data to simulate bringing the contents from the L2 level to the L1 level for easy access by the CPU. However, if it’s a miss on both cases then we resulted in the anyone of the replacement algorithm. Dirty or contaminated data was also assumed to be misses for simplicity.

Below is are diagrams showing the flowchart on the abstract implemented in our code for both the write back() and write through without write allocations write policies in case there is a miss or contaminated data .

WRITE-THROUGH CACHE WITH NO-WRITE ALLOCATION



WRITE-BACK CACHE WITH WRITE ALLOCATION

In conclusion: we used the LRU replacement algorithm implemented as a member of the cachemanager class to replace any data that was dirty or contaminated. This meant we had to keep track of time by using the function get age to see which line/block had been recently visited by the system to simulate interaction between memory caches and processing hardware.