Adam Abrams and Matt Jerrard

Cache Overview

The first steps we took in implementing a cache for our single-cycle processor was storing the values given by the user, these include test file, block size, number of sets, and associativity. Our program gives the user the option of including those parameters as command line arguments or as an interactive prompt. We then error check these values to ensure that they are powers of two and in the correct ranges. We used these values to calculate the cache layout and the size of our cache arrays.

Next we call “runInstrs” to execute the machine code read from the input file. Instead of fetching an instruction from memory, we call another function “getCache”, which checks if the desired instruction is in the cache. If it is, the function will return that value and update the LRU when applicable. If it is not already in the cache, the “getCache” function will call our “evictBlock” function. The store word instruction is a special case in this process. For it, we call the “putCache” function in order to properly write the value into the cache. If the instruction was not in the cache, then the “evictBlock” function is called to properly make room for the new instruction. Then the needed block is allocated to the cache, changed, and set as dirty. This function will also write the data of the evicted block to memory if it is dirty.

Once the program executes a HALT, the “writeBack” function is called to ensure that blocks that are still dirty and valid at this time are written to memory.

Challenges

Our first challenge was properly dealing with pointers, structs, and 2D arrays in C. We also struggled with proper allocation for each index and the handling of pointers. We found it helpful to use hand drawn diagrams and asked our professor for assistance.

The second challenge we encountered was eviction with proper block alignment. At first we were not using the first index of the block when writing to memory, causing invalid data. We solved this challenge by zeroing out the block offset before writing to memory and then looping through each word in the block.

The third and last challenge was properly dealing with dirty block in the cache. We had to specifically ensure that dirty blocks are only written to memory upon eviction, not with every write made to them.

We found no major shortcomings with our project.