Alex Palumbo & Ryan Wenner
Overview

Cache Simulator

Our CPU cache simulator was built by extending the provided single cycle processor behavioral simulator. While the simulator executes an assembly language program it accesses instructions and data from the cache which transfers data to and from memory. Addresses are word addressable so blocks are made up of words. The cache simulates a variety of cache configurations provided by the user. The block size, number of sets, and associativity of the cache are inputted by the user at the beginning of the program.

After the user inputs are supplied the program loads all the instructions in memory from the supplied file. Before reading each instruction the cache is initialized. The cache, sets, and entries are all held as structs. The cache struct holds an array of set structs which hold their own array of entry structs. Once the cache has been filled with empty entries, each instruction gets executed. Instructions are fetched from the cache. Any reading of memory is done through the cache. Reading from the cache involves breaking up the address into an offset, set and tag. These are used to identify where in the cache to look for the address. Using these identifiers, the cache is searched to see if an entry for the address exists. The cache looks in the set for the tag. If the entry is found then the offset is used to find the address in the entry data. If there's no entry then it needs to be pulled into the cache from memory. If there is an empty entry in the set then the entry can be placed there, otherwise a different entry will need to be removed. The entry to be removed will be the least recently used entry. If the entry being removed was written to previously, meaning it's dirty, then that entry's data will need to be written to memory first.

After the instruction is fetched, the instruction is performed. Load word and store word are the two instructions that use the cache. The other instructions stay the same. Like fetching an instruction, loading a word also reads from the cache so the same steps are done for loading a word. The only difference is a load word instruction stores the memory into a register. A store word is very similar to load word and instruction fetching. The memory that a store word is writing to needs to be in the cache first so the same steps will happen. However, a store word will write data to an entry in the cache. It will also set that entry to dirty so later it can be written to memory.

Once all the instructions have finished executing, caused by a halt, a couple things need to happen to the cache. Any entry in the cache that is dirty needs to be written. Also, all the entries are invalidated, signifying they are empty.

Difficulties

There were a few difficulties while building the cache simulator. One error we had was setting the dirty bit back to 0 when the data had not been written yet. Also, when there was only one set, unexpected behavior was happening. That was solved by using set index 0. Calculating the addresses for memory transfers took some effort. Finding the least recently used entry also was difficult. The overall design was the most difficult, we chose to use structs for holding all data in the cache instead of a 2d array.

Shortcomings

No shortcomings. Everything was implemented and tested against the solution simulator.