Computer Organization: Instruction Pipeline and Cache (IPLC) Simulator

Members:
Mary Ly Tim Saia
Will Stone Lorelei Wright

Mary and Tim wrote the implementation of the cache. The trap address function, written by Tim, checked if the address was in the cache. A bit mask was used to get the index and the tag, then went through the cache to see if any of the indices matched the tag. If it did, then the update-on-hit function was called, otherwise it was a miss and the replace-on-miss function was called. The replace-on-miss function, written by Mary, puts the address into the cache and makes it the most recently used entry. It replaced the least recently used cache slot and then percolated it up so that it would then be the most recently used cache slot.

Lorelei and Will wrote the implementation of the pipeline processing functions. Will implemented the load word, store word, and branch instruction type functions, Lorelei implemented the jump, syscall, and nop instruction type functions. For each, the structs with the required member variables were provided, so the instructions were pushed to the pipeline and the parameters of the function were set to the member variables. The types of the instructions were also set to the appropriate type.

Will and Tim wrote the implementation of the pipeline. Will did steps 2, 4, and 6, Tim did step 3. Step 2 was implementing the check if the branch prediction was correct or incorrect. It compares the addresses to check if the branch was already predicted or taken. The the addresses are more than a word (4 bits) apart, then it is assumed the branch was taken. The amount of correct branch predictions is incremented if it was correct, otherwise the data is copied, pushed to the next stage, DECODE is reset, and the instruction count is incremented by 1. Step 3 was implementing the load word part of the function. It checked if the word was in the cache, if it's not then it's a miss and a delay is added. If the ALU instruction is an r-type, then the information is written to the write back register and the main register is cleared. Step 4 was implementing the store word part of the function. It calls the trap address function to check if there's a data hit or not. If not, then the pipeline cycles are incremented by the number of cycles the program has to stall for a miss. Lastly, Step 6 was pushing the pipeline data to the next stage at the end of the function.