Comp Org Final Project Writeup

Group Evaluation

Our group consisted of Weston Jones and Eileen Yao

Weston primarily focused on the implementation of the cache and writing up the performance comparison for the code.

Eileen primarily focused on the implementation of the pipeline.

After working on our respective tasks, both of us collaborated on debugging and bug fixing.

All in all, we feel like the work was distributed evenly and that each group member should receive full marks for the part of our grade that is determined by this evaluation.

Summary of Implementation

Cache

First, I completed the implementation of the cache line struct. Each cache line was to hold an array of ints representing the tags for each block, an array of ints for the valid bit of each block, and an array of ints representing the “usage history” (the number of cache accesses since the block was used last) of each block.

These arrays would be sized dynamically at runtime after user input, so I made sure all my future code was general enough to accommodate all associativities (1 being just a direct mapped cache).

When locating something in the cache, the code uses bit masks to splice out the index and tag. The index points to the correct line of the cache. From there, the code loops through the levels of associativity (just 1 if direct mapped) and checks the tags of the blocks to see if there’s a hit.

On a hit, the code increments the relevant counters, then updates the usage histories of all other blocks at that index except for the one just accessed.

On a miss, the code checks all blocks at that index for the highest usage history. That block is updated with the new information, then all other blocks, except the one that was just updated, have their usage histories incremented by 1.

Pipeline

The pipelining was implemented using the given functions. Completed all the pipeline functions that depended on instructions from the given input. Each function was filled in with it's corresponding required values from the struct. Then proceeded to the push_pipeline_stage function. Focused on the second block of code for the branch for a while. I was confused as to how the branch prediction worked at first. I first altered the given branch_taken variable given in the code using the hint in the pdf file. The value of the variable would be 1(true) if the branch was taken. I then figured out that the initial value in the beginning of the program that asked for branch prediction indicated that it would assume that all branches would either be taken or not taken depending on the user input. So I thought that to check if the branch taken was correct by comparing it to whether it was actually taken. Then if the prediction was incorrect, I incremented the cycles and pushed the instructions to the right while inserting a nop in the DECODE stage. If the prediction was correct then it simply incremented the correct number of branch predictions variable. I wasn't sure how to account for forwarding. I considered comparing des_reg of ALU and MEM stages if ALU was rtype or MEM was sw or lw. I wasn't sure how the regs would be forwarded from those regs to the current regs so I left it blank. In 3 I had to check if there were delays in the ALU stage and the only way there would need delays was if the value was being altered which meant it had to be an rtype function altering the value. If there was a conflict, I incremented the nop value and pipeline cycles as a form of stalling. I also pushed the instruction forward and NOP-ed the current stage. If there was a miss in the cache, which was found using the trap_address function then the pipeline cycles would be stalled by the number of inserted nops and the CYCLE_MISS_DELAY -1 because the instructions were being pushed forward anyways. The fourth stage was only affected by a miss so that only incremented the pipeline cycles if it missed. Then the pipeline_cycles was incremented as a whole because it meant that a cycle has finished so it was really for pushing. Then the stages were pushed right and FETCH was nulled to accept a new instruction.

Performance Evaluation

Results																	
Predict Branch Taken									Predict Branch Not taken								
Associativity									**Associativity**								
1			2			4			1			2			4		
Block Size (words)			**Block Size (words)**			**Block Size (words)**			**Block Size (words)**			**Block Size (words)**			**Block Size (words)**		
1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
Results are shown as Maximizing Index (Top), Miss Rate (middle), and CPI (below)																	
7	6	6	6	5	5	5	5	4	7	6	6	6	5	5	5	5	4
0.038759	0.036277	0.021471	0.013998	0.009955	0.004015	0.018208	0.004294	0.054680	0.038759	0.036277	0.021471	0.013998	0.009955	0.004015	0.018208	0.004294	0.054680
1.524991	1.501827	1.364400	1.295284	1.257647	1.202860	1.334532	1.205364	1.672115	1.396915	1.374184	1.236699	1.166978	1.129715	1.074583	1.206140	1.077144	1.058009

Summary of Results