Michael Levitskiy

In our virtual computer project, we tested the value of caching. We have already implemented different parts of a computer, such as the main memory, alu, processor, and an assembler that will take assembly code and coverts it into binary for the processor to run and execute. Many of the instructions given to the processor take time to execute and we calculate this time using clock cycles. Initially we stated that accessing memory is 300 cycles, multiplication is 10 cycles, and all other ALU functions are 2 cycles. Then we implemented an Instruction Cache, where instead of reading instructions from memory, we read then from the Instruction Cache, which contains an 8 Word block from memory. If the instruction address is in this cache, then we retrieve the instruction with a cost of 10 cycles, but if not, then it's 350 cycles for the Instruction Cache to retrieve the new block from Main Memory and return the correct instruction. Finally, we implemented an L2 Cache. This cache contains 4 different 8 Word blocks from memory. We adjusted the processor so whenever we read or write from memory, we read and write from the L2 Cache, except for instruction retrieval which that still reads from the Instruction Cache. We also adjust the Instruction Cache so that if it doesn't contain the desired instruction, it reads from the L2 cache for 50 cycles, instead of main memory for 350 cycles. Our L2 cache is implemented so if our desired address is not in the L2 Cache, it costs 350 cycles to retrieve it from memory, but only 50 cycles if it is contained within the cache.

To test these 3 implementations, I wrote 3 different assembly programs to test to be run by the processor. The first program fills an array of 20 numbers and sums them in order. The second program creates a linked list of 20 numbers and sums them in order. The third program fills an array of 20 numbers and sums them in reverse order. The table on the next page shows the results.

When you analyze the table, you can see the differences between the implementations as well as the efficiency of the 3 methods to sum 20 numbers. Initially, all 3 programs have clock cycles numbers that are nearing 100,000 cycles. The array implementations in tests 1 and 3 performed better than the linked list implementation by more than 11,000 cycles and are within 2 cycles of each other. After implementing the instruction cache, all 3 programs improved drastically. They decreased from about 87,000 and 98,000 cycles down to about 16,000 and 32,000 cycles. Again, the array implementations are within 2 cycles of each other and also outperformed the linked list implementation by about 16,000 cycles, which is about twice as fast. Finally, after applying the L2 Cache, all the clock cycles decreased again. Both array implementations outperformed the linked list implementation by about 16,000 and 17,000 cycles, but this time, test 1 outperformed test 3 by about 1,500 cycles instead of 2 cycles. Summing the numbers in order performed better than summing the numbers in reverse.

Looking at this data overall, the value of caching becomes clear. By having levels of memory that the processor works with, we can save clock cycles and time by retrieving information that is much closer and less costly. Just from implementing an instruction cache and an L2 cache, test 1's performance improved 9.4 times, test 2's performance improved 3.7 times, and test 3's performance improved 8.2 times. On top of the value of caching, we also learned how arrays are quicker to process than linked lists and reading arrays in order are also faster than reading them in reverse.

Michael Levitskiy

	Total Clock Cycles	Instruction Cache Hits	Instruction Cache Misses	L2 Cache Hits	L2 Cache Misses
Initial Test 1:	87,324	-	-	-	-
Initial Test 2:	98,422	-	-	-	-
Initial Test 3:	87,326	-	-	-	-
Instruction Cache Implemented Test 1:	16,524	245	5	-	-
Instruction Cache Implemented Test 2:	32,912	229	18	-	-
Instruction Cache Implemented Test 3:	16,526	245	5	-	-
L2 Cache Implemented Test 1:	9,308	245	5	53	7
L2 Cache Implemented Test 2:	26,804	229	18	147	37
L2 Cache Implemented Test 3:	10,710	245	5	49	11