Lab Report 6

Chris Dobson
EEL4713

Section 1: Book Problems

The Blue non-revised 4th edition was used

6.3.1)

a)

$$Average = seekrate + \frac{.5}{rps} + \frac{data}{rate}$$

$$Average = 11ms + \frac{.5}{\frac{7200}{60}} + \frac{1024}{34 * 2^{20}}$$

$$Average = 15.1954\ ms$$

b) 13.1992 ms

$$Average = seekrate + \frac{.5}{rps} + \frac{data}{rate}$$

$$Average = 9ms + \frac{.5}{\frac{7200}{60}} + \frac{1024}{30 * 2^{20}}$$

$$Average = 13.1992\ ms$$

6.3.2)

a) *minimum assumes the case where there is no seek time or rotational delay.

$$minimum = \frac{dada}{rate}$$

$$minimum = \frac{2048}{34 * 2^{20}}$$

$$minimum = .02872$$

b)

$$minimum = \frac{dada}{rate}$$

$$minimum = \frac{2048}{30 * 2^{20}}$$

$$minimum = .03255$$

6.15.1)

a)

$$parity = FEFE\ xor\ A387\ xor\ F345\ xor\ FF00$$

$$parity = 513C$$

b)

$$parity = AB9C\ xor\ 0098\ xor\ 00FF\ xor\ 2FFF$$

$$parity = 8404$$

6.15.2)

a)

$$parity = FEFE\ xor\ 00FF\ xor\ 4582$$

$$parity = BB83$$

b)

$$parity = AB9C\ xor\ F457\ xor\ A387$$

$$parity = FC4C$$

6.15.3)

Raid 4 is more efficient with small reads, because a single block can be accessed from just 1 disk instead of all the disks like raid 3 requires. Raid 4 also requires less reads to build a new parity block, making writing more efficient. Raid 3 has no real advantage over raid 4.

6.15.4)

Raid 5 constructs its parity blocks the same as raid 4, but it distributes them across the disks instead of storing them all on one disk. This prevents the parity disk from bottlenecking the performance during back to back writes. A single write will not see any improvement.

6.18.1)

a)

$$AFR = \frac{\frac{hours}{drive}}{\frac{hours}{failure}}$$

$$AFR = \frac{8760}{1000000}$$

$$AFR = .876\%$$

$$failed\ disks = \frac{\left(drives * \left(\frac{hours}{drive}\right)\right)}{\frac{hours}{failure}}$$

$$failed\ disks = \frac{\left(1000 * (8760)\right)}{1000000}$$

$$failed\ disks = 8.76$$

b)

$$AFR = \frac{\frac{hours}{drive}}{\frac{hours}{failure}}$$

$$AFR = \frac{10512}{1500000}$$

$$AFR = .7008\%$$

$$failed\ disks = \frac{\left(drives * \left(\frac{hours}{drive}\right)\right)}{\frac{hours}{failure}}$$

$$failed\ disks = \frac{\left(1000 * (10512)\right)}{1500000}$$

$$failed\ disks = 7.008$$

6.18.2)

a)

$$failed\ disks(7) = FR * (1.6667 + 3 + 2 + 4 + 8)$$

$$failed\ disks(7) = 159.14$$

$$failed\ disks(10) = FR * (1.6667 + 3 + 2 + 4 + 8 + 16 + 32 + 64)$$

$$failed\ disks(10) = 1140.26$$

a)

$$failed\ disks(7) = FR * (1.6667 + 3 + 2 + 4 + 8)$$

$$failed\ disks(7) = 127.312$$

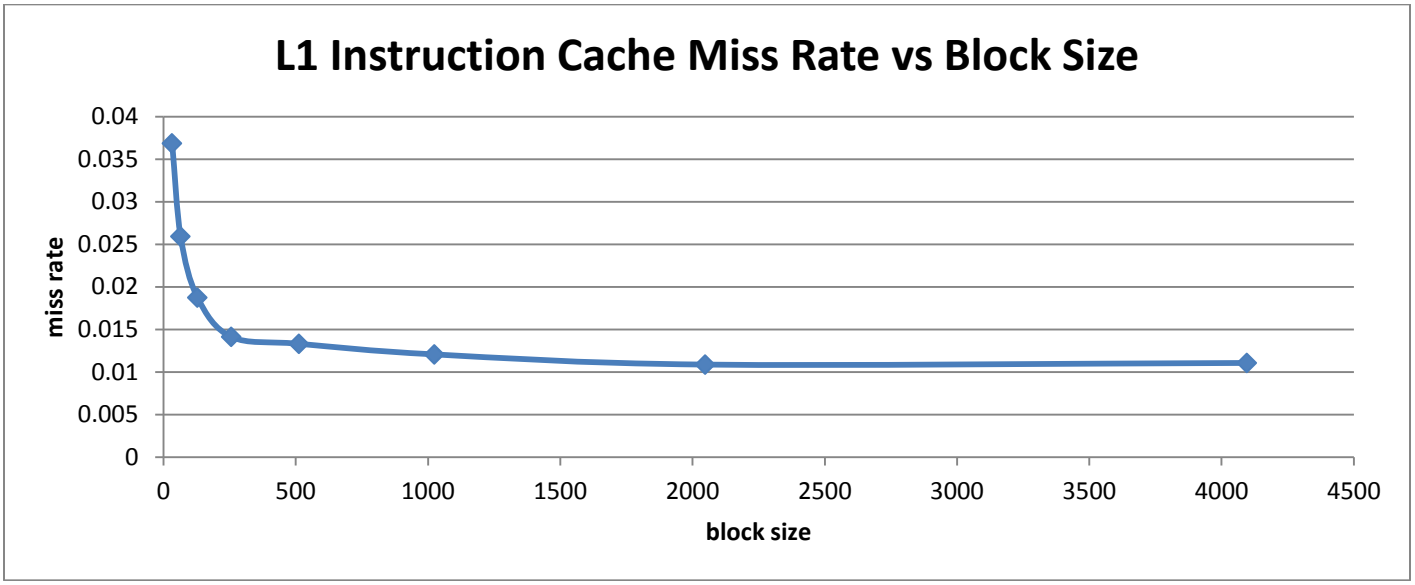
$$failed\ disks(10) = FR * (1.6667 + 3 + 2 + 4 + 8 + 16 + 32 + 64)$$

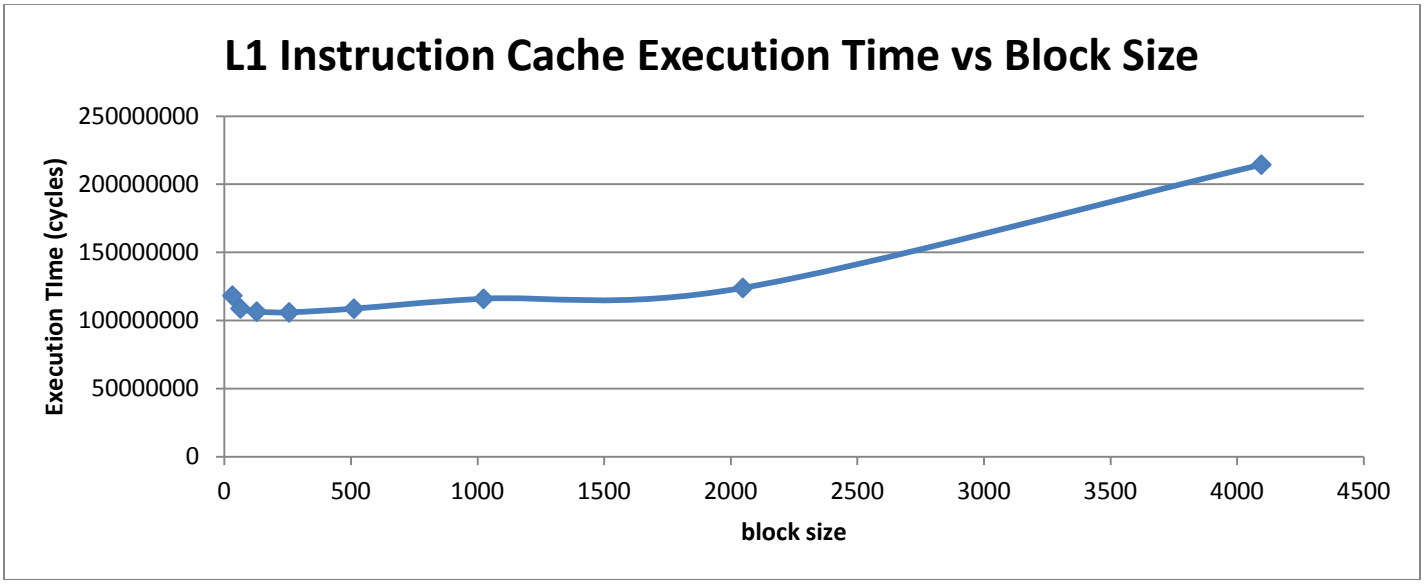
$$failed\ disks(10) = 912.208$$

Section 2: Cacti and SESC simulations

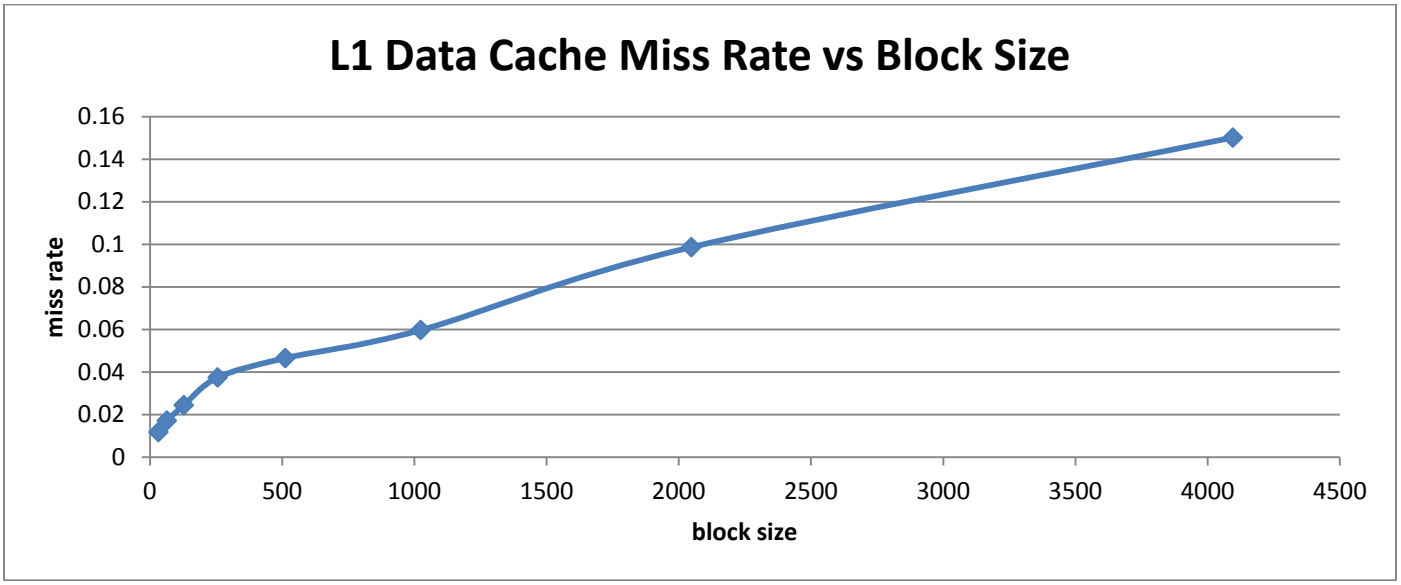
Block Size Tradeoffs

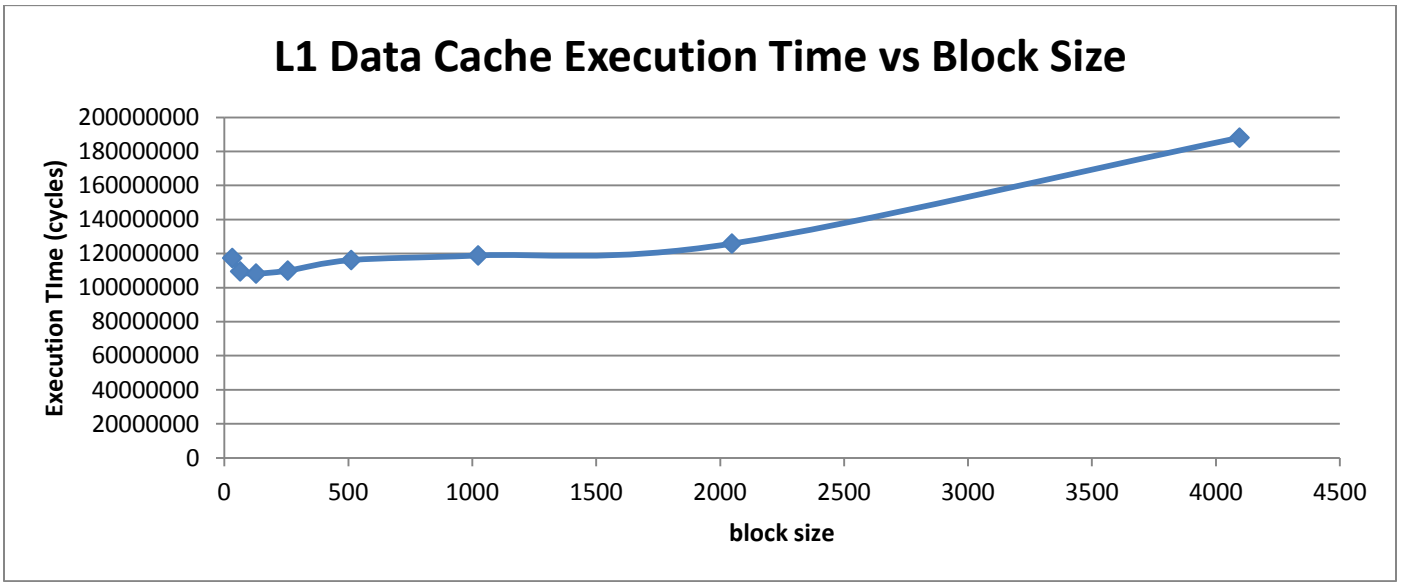
The first simulation involved testing different block sizes for the L1 instruction cache. The two charts below summarize the results from the experiment. The first chart shows the different miss rates depending on the block size, while the second chart shows the change in execution time. The miss rate makes a significant drop right away, with the decline becoming more gradual as time passes. This indicates that the most gain per area will be in the smaller block sizes, with the large ones offering similar miss rates. The execution time follows a different trend. As the block size increases, the execution time starts to go down but turns back up before long. This is caused by the larger latencies that cacti provided for the larger block sizes.





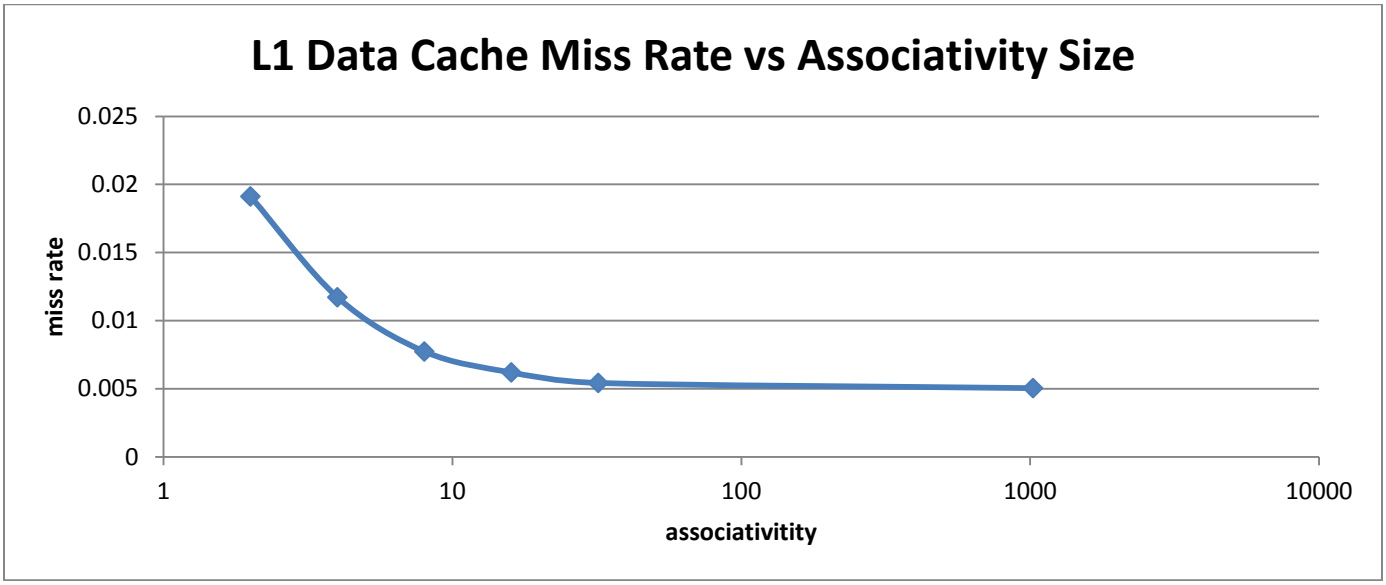
The second simulation involved testing different block sizes of the L1 data cache. The two charts below summarize the results from the experiment. The first chart shows the different miss rates depending on the block size, while the second chart shows the change in execution time. The results are significantly different from before. Increasing the block size actually increase the hit rate. This means the data cache does not have the same type of locality as the instruction cache. The fact that a larger block size decreases the total number of blocks available in the table (a fixed total cache size was used) could also be having a significant effect. The second chart provides some unexpected results. The execution time dips down in the beginning, just like in the previous example even though the miss rate is increasing. This is likely a byproduct of increasing the L2 cache at the same rate as the data cache. This brings the execution time gains from the previous simulation into question. Having the L2 cache independently set to a larger block size was causing SESC to lock up, meaning the configuration file was likely in error. Meaningful information can still be determined from the cache miss charts.

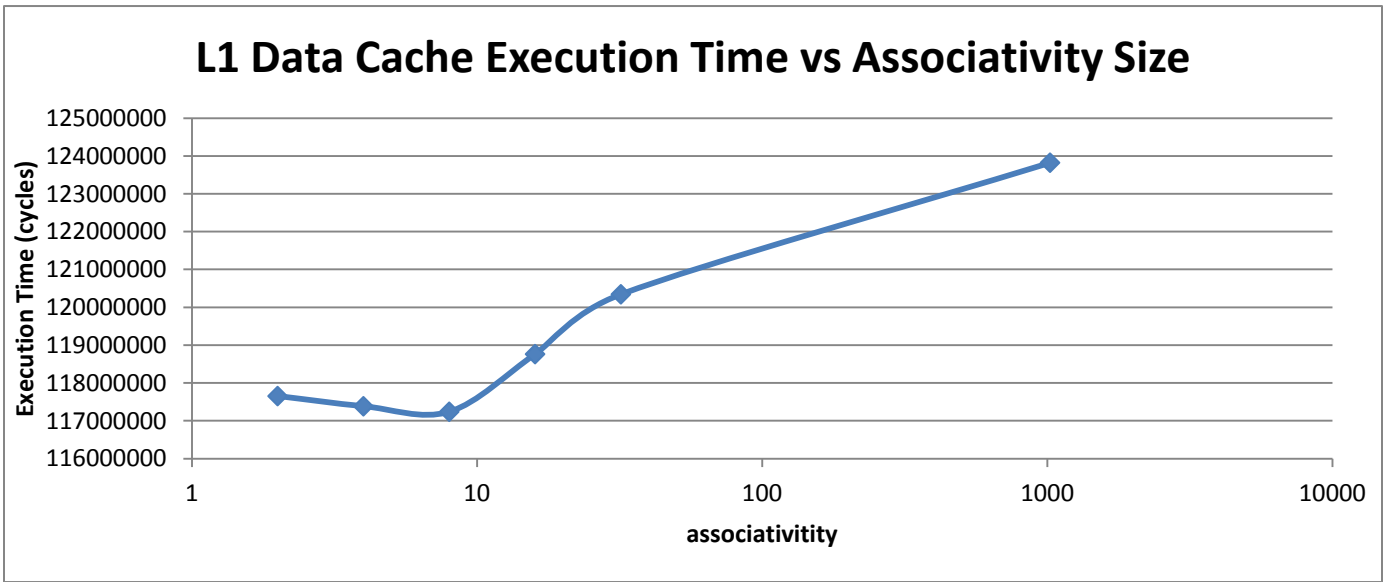




Cache Associativity Tradeoffs

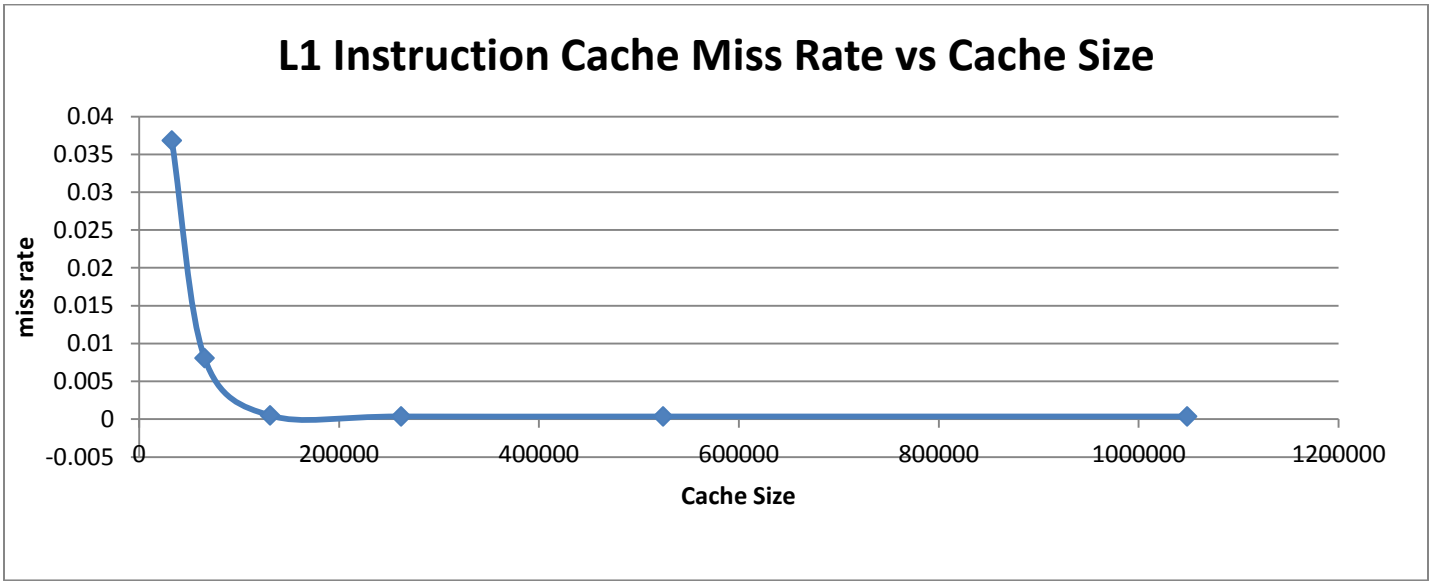
This simulation involves simulating different cache associativities for the L1 data cache. The first chart shows the different miss rates depending on the associativitity, while the second chart shows the change in execution time. Like in the first case, there is a significant initial drop in miss rate which then levels off as the associativitity increases further. The case with an associativitity of 32 has almost the same hit rate as the fully associativite case. The same trend can be seen in the execution time as was present in the previous simulations. As the associativitity gets larger and larger, the cache latency increases. Considering that the hit rate decrease is negligible for the larger cases, it's only natural that the execution time should rise sharply.

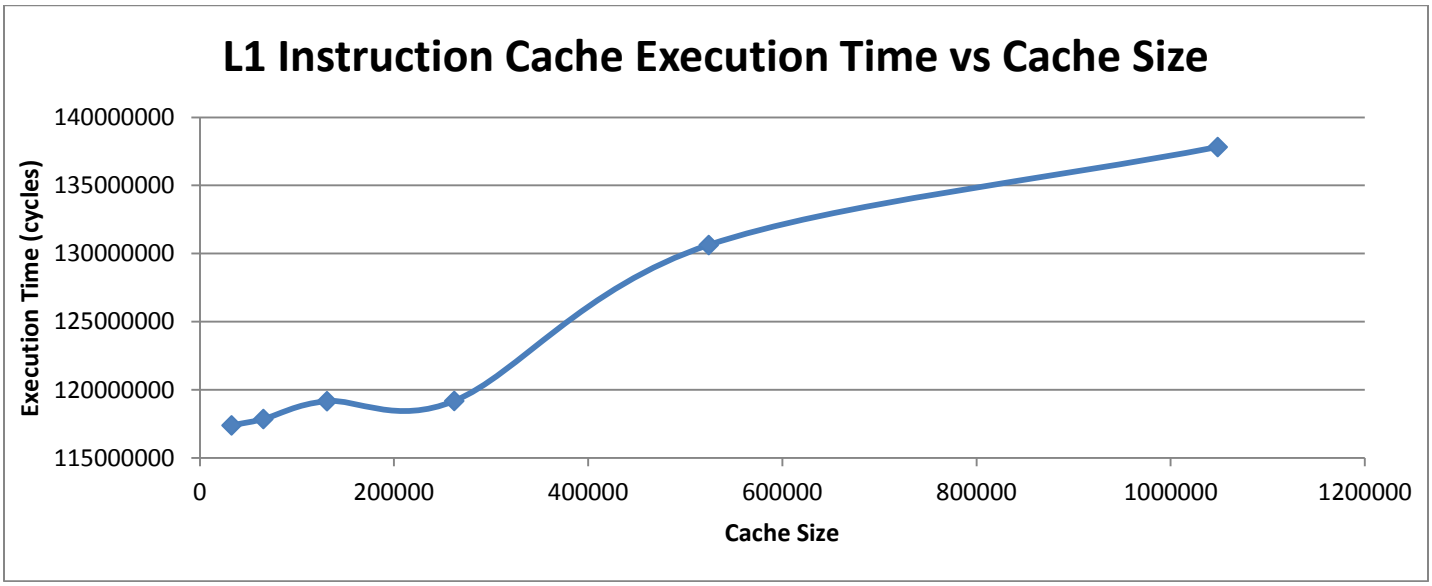




Cache Size Tradeoffs

This simulation involves simulating different cache sizes for the L1 instruction cache. The first chart shows the different miss rates depending on the size, while the second chart shows the change in execution time. The first chart follows the same trend as two of the previous simulations. The hit rate starts out by dropping significantly, and then leveling off. The third test case is approximately as efficient as the largest. The execution time never actually drops however. The change in miss rate never outweighs the hefty increase in cache latency that is associated with increasing the cache size.

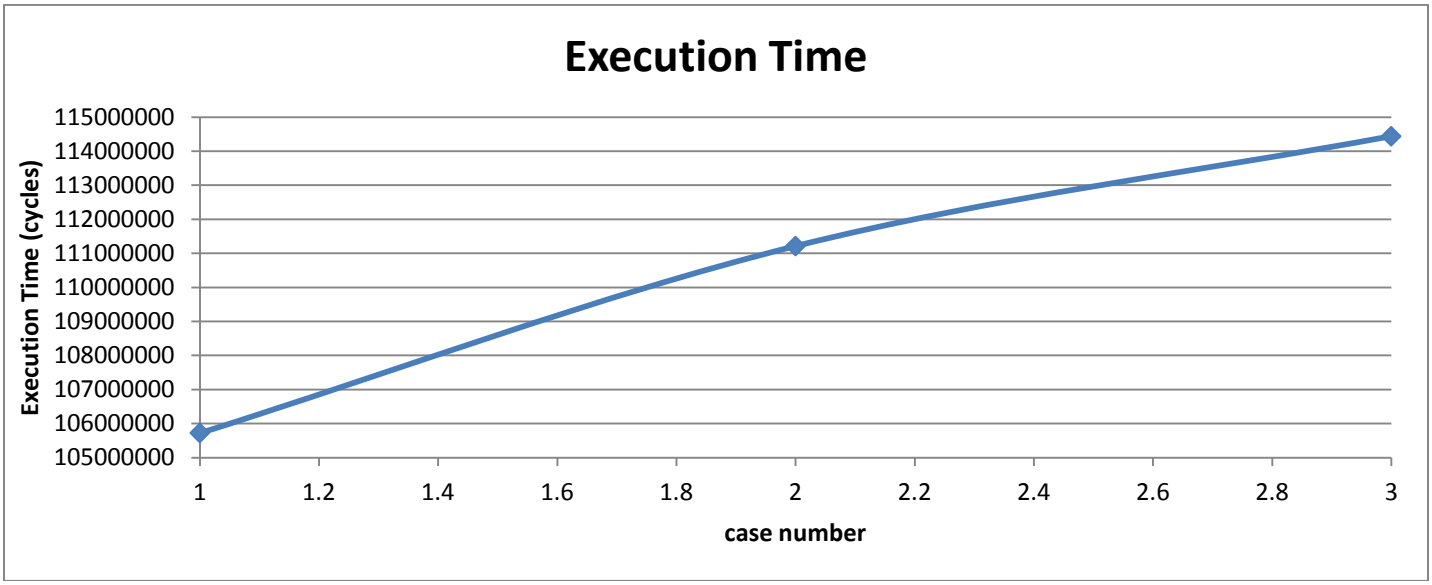




Overall Cache Performance Experiment

This simulation involves simulating three different test cases based of the previous results. Each test case represents an increase in cache latency, while the attributes are maximized to say within that latency. The data cache was first given an increase in associativitity, because it caused the most significant drop in the trials. The instruction cache was first made larger and given a larger block size, because those two attributes made the biggest changes in the previous simulations. The table below summarizes the 3 configurations used, while the chart below shows the relative execution times. The first case was the fastest by far. The smaller cache latencies must have been more important than the increased hit rates that went along with them. The fact that the miss rates were already so low (almost always below 5%) and the miss penalty is not too severe gave the configuration with the smaller latency a clear advantage.

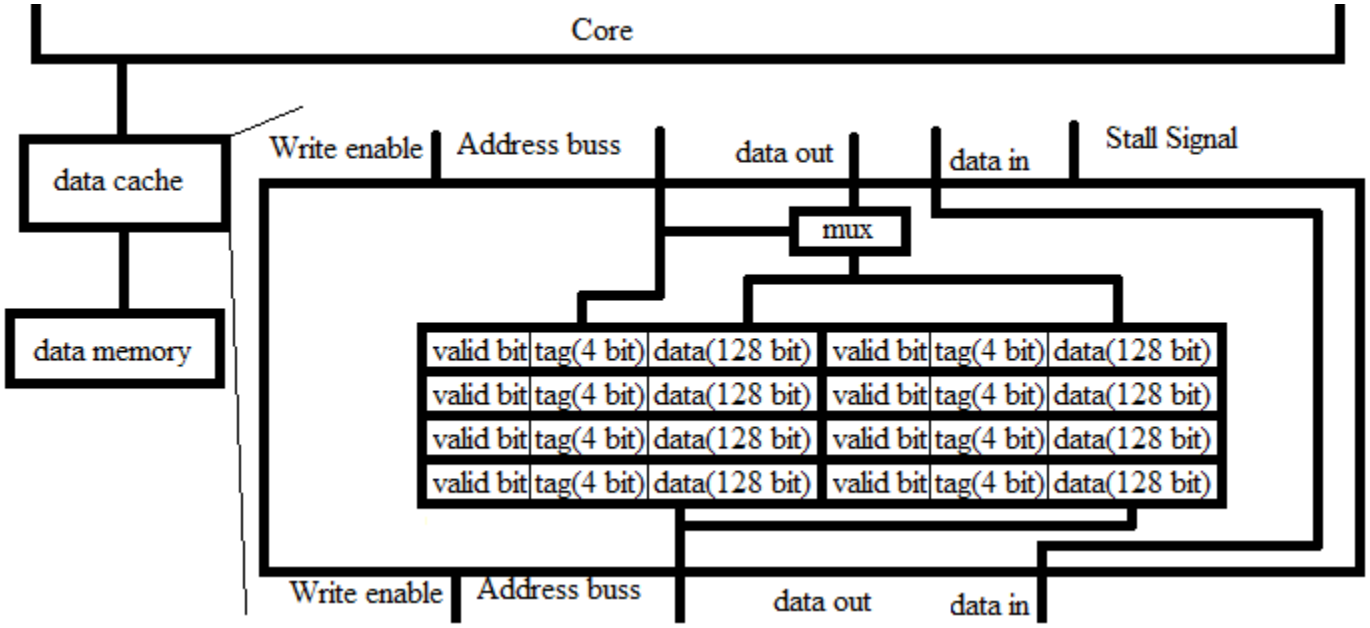
Test Case	Data Cache Size	Data Cache Associativity	Data Cache Block Size	Data Cache Latency	Instruction Cache Size	Instruction Associativity	Instruction Block Size	Instruction Latency
1	32768	8	32	4	65536	2	256	4
2	131072	16	32	5	65536	2	512	5
3	65536	32	32	6	262144	2	512	6



Section 3: Implementing a Simple Data Cache

This section deals with the addition of a basic two way associativite cache for the data memory of the pipelined MIPS processor designed in the previous lab. A write through policy and no-write allocate policy were used, so the cache only effects reads. The diagram below represents the cache system without the control logic. Whenever a write occurs, the data is written directly to ram (and updated in the cache if that address is present) so very little has changed during write cycles. Read cycles are controlled by a simple FSM. Whenever a cache miss, The FSM begins a 4 cycle process that reads each of the 4 words bytes associated with that address into one of the two possible locations that data can be present in the cache. The location is randomly chosen. During this process, the rest processor is stalled. When the data is done being read into the cache, the processor resumes. The total time needed to read the data into the cache is 4 cycles, which is 3 more than the single cycle needed to read from the cache.

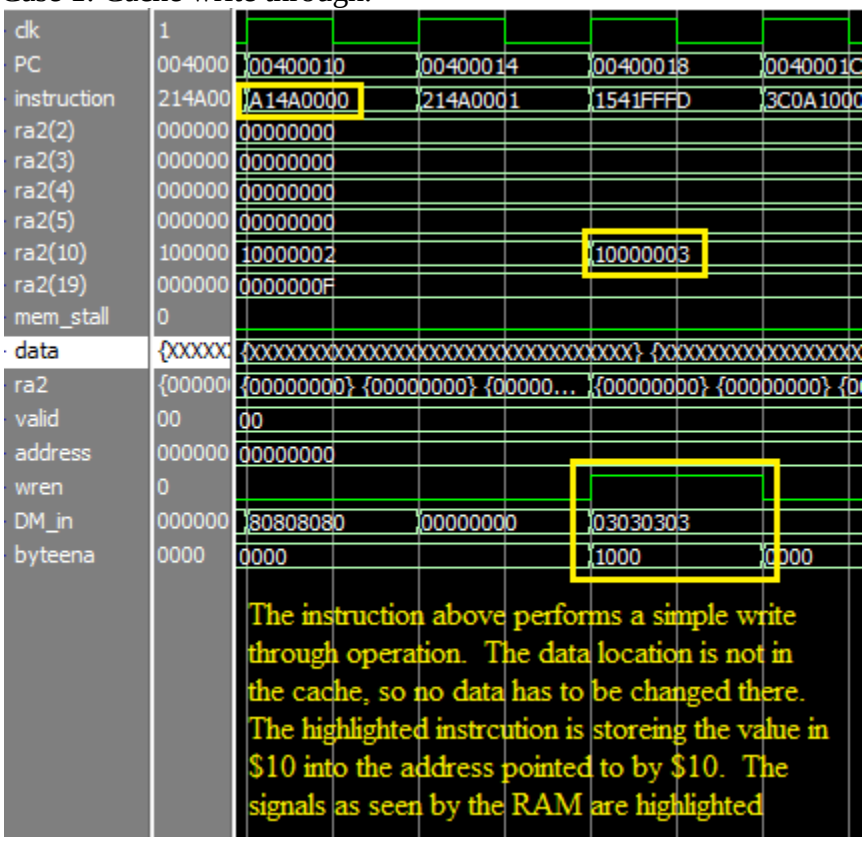
The point of this cache is to reduce the effect of high memory latency. This could be very use full in a system where the processor runs much faster than its main working memory, or when the memory has a relatively long latency. The current implementation on the FPGA would likely see no benefit and only a performance decrease because the memory is capable of running at the same rate as the CPU with no latency. It is likely that other technologies exit where even a simple cache system like this would result in a significant increase in performance (though a write buffer would be necessary to speed up the write process).



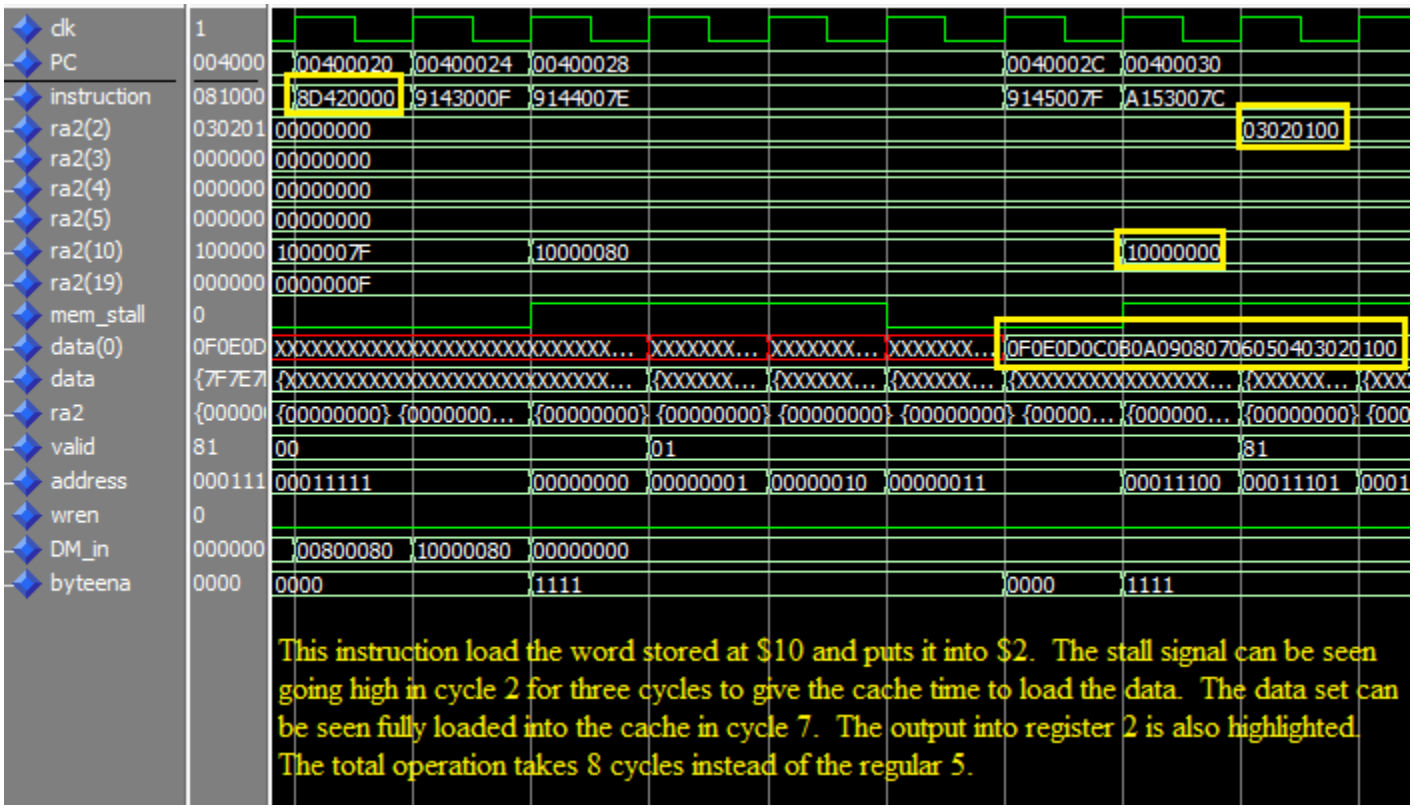
Simulations

Several different simulations are below that show various cache utilizations.

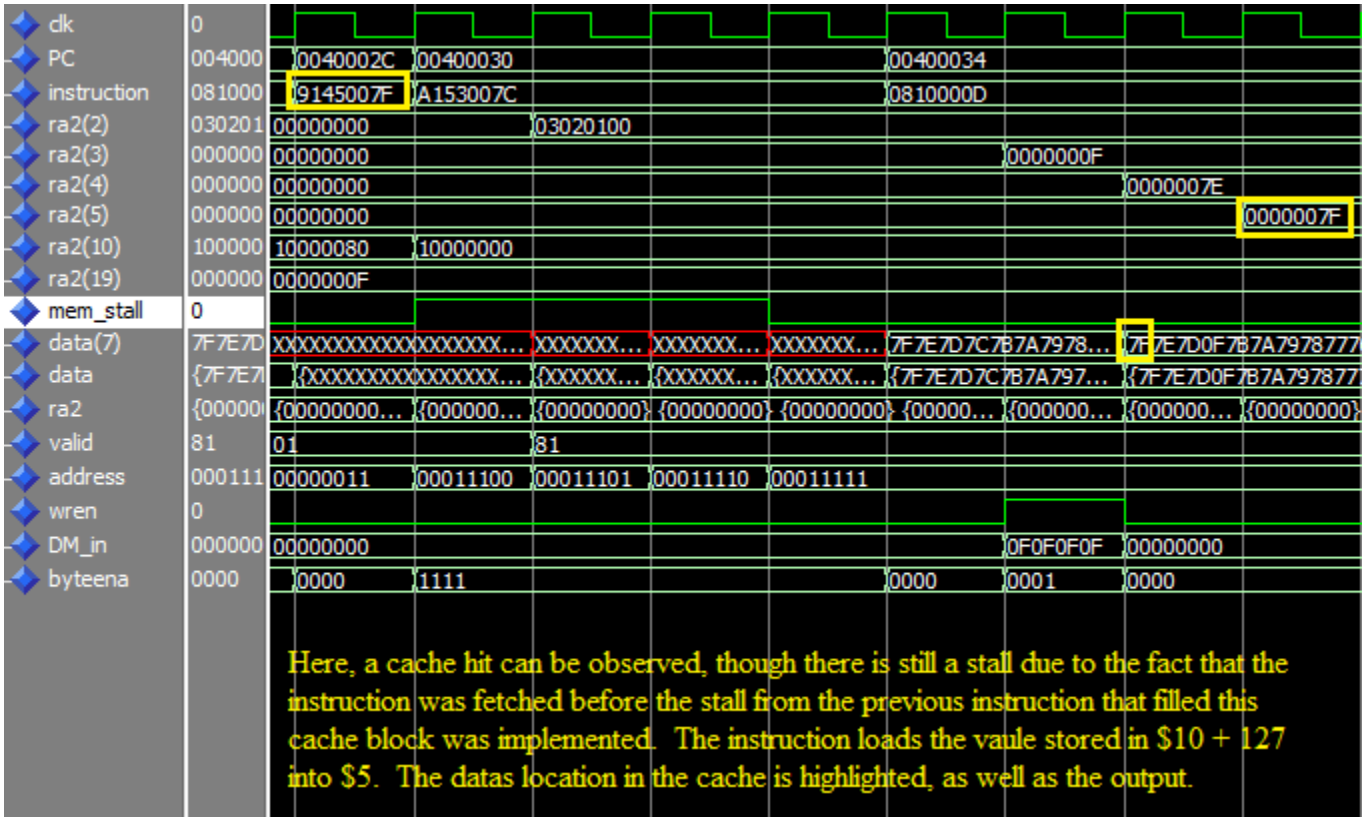
Case 1: Cache write through.



Case 2: Cache Miss



Case3: Cache Hit



Case 4: Cache Write