**Project 1: Cache Simulator**

**Part A (8KB to 128KB, 4-way associativity, write-back, LRU):**

**XSBENCH.t**

At 8KB:



At 16KB:



At 32KB:



At 64KB:



At 128KB:



Plot:

**MINIFE.t**

At 8KB:



At 16KB:



At 32KB:



At 64KB:



At 128KB:



Explanation of Part A:

According to my results, as cache size increase, each recorded element (miss ratio, writes to memory, and reads to memory) slightly decreases in value. This is due to the cache size increasing in size, which, in turn, gives a greater number of sets and more space for address to be held in cache without being evicted. As a result, miss ratio will slightly decrease since there is a higher chance of an element still being within cache, which will also lower your reads on memory, since you read on each miss. Writes to memory will decrease since you have a lower chance of evictions.

**Part B (Write-back and Write through for each cache size):**

**XSBENCH.t**

At 8KB:

Write-Back:



Write-Through:



At 16KB:

Write-Back:



Write-Through:



At 32KB:

Write-Back:



Write-Through:



At 64KB:

Write-Back:



Write-Through:



At 128KB:

Write-Back:



Write-Through:



**MINIFE.t**

At 8KB:

Write-Back:



Write-Through:



At 16KB:

Write-Back:



Write-Through:



At 32KB:

Write-Back:



Write-Through:



At 64KB:

Write-Back:



Write-Through:



At 128KB:

Write-Back:



Write-Through:



Explanation for Part B:

I really do not see much need to explain part B. Much of the information is quite intuitive if you understand what write-back and write-through is. As you can see from the plotted data, write-through is much higher than write-back. The reason for this is that you need to update the cache just as much as you need to update memory on writes, the writes become the same regardless of cache size. With write-through you write to memory exactly how many times you would write to cache. This is probably why write-back tends to be preferred since the number of writes are significantly smaller.

**Part C (Fixed Cache Size of 32KB, 1 to 64-way associativity, write-back, LRU):**

**XSBENCH.t**

1-way associativity:



2-way associativity:



4-way associativity:



8-way associativity:



16-way associativity:



32-way associativity:



64-way associativity:



**MINIFE.t**

1-way associativity:



2-way associativity:



4-way associativity:



8-way associativity:



16-way associativity:



32-way associativity:



64-way associativity:



Explanation for Part C:

The explanation for C is like the explanation of A. As you increase in associativity you are increasing the number of information that can be held within each set. This gives you more data with less chance of eviction. Although, once you reach 8-way associativity, there is not much change after that. Your benefits with increasing associativity go stagnant.

**Part D (Fixed Cache Size of 32KB, 1 to 64-way associativity, write-back, LRU and FIFO):**

**XSBENCH.t**

1-way associativity:

LRU:



FIFO:



2-way associativity:

LRU:



FIFO:



4-way associativity:

LRU:



FIFO:



8-way associativity:

LRU:



FIFO:



16-way associativity:

LRU:



FIFO:



32-way associativity:

LRU:



FIFO:



64-way associativity:

LRU:



FIFO:



**MINIFE.t**

1-way associativity:

LRU:



FIFO:



2-way associativity:

LRU:



FIFO:



4-way associativity:

LRU:



FIFO:



8-way associativity:

LRU:



FIFO:



16-way associativity:

LRU:



FIFO:



32-way associativity:

LRU:



FIFO:



64-way associativity:

LRU:



FIFO:



Explanation to Part D:

With the comparison between FIFO and LRU, LRU tends to come in first place each time. Here is another one of those cases. As with Part B, if you understand how FIFO and LRU work, you can easily understand why the data is the way it is. With FIFO it does not care about repeats, it just cares about what is being called upon. LRU will move an identical tag to the front of the stack. Just this factor alone decreases writes and reads from memory, because, once again, there are fewer evictions (since we are focused on write-back). As you can see from the data, with all of this in mind, LRU reaps more of the benefits of change in associativity.