Paul Francis Lee
A0234681A
Assignment 2 Part 1 Report

Assumptions
A cache miss (without eviction) incurs 100CC for fetching from memory to cache, and 1 additional CC for then reading from the cache.

How to run
I wrote the program in python and am using python 3.11.7 however any somewhat recent python3 version would work as well.

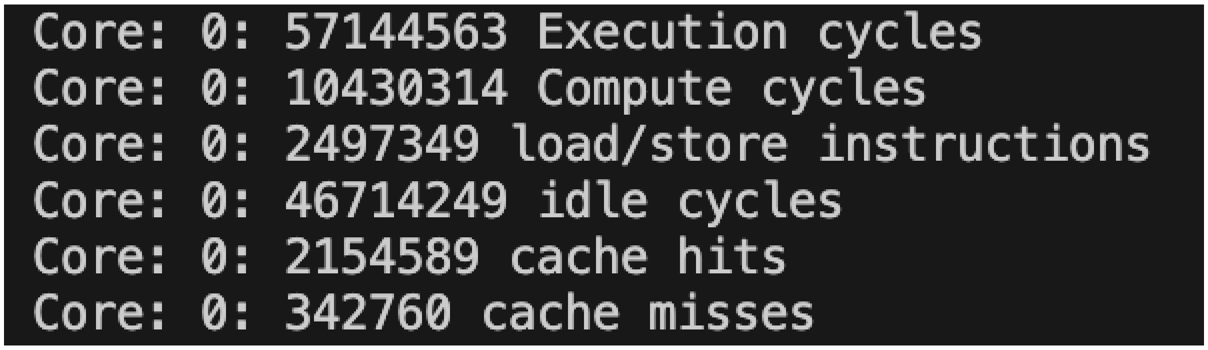
To run, run this command from the root directory (cache-coherence-protocol-sim).
```
python -m coherence [protocol] [input_file] [cache_size]
[associativity] [block_size]
```

For e.g,

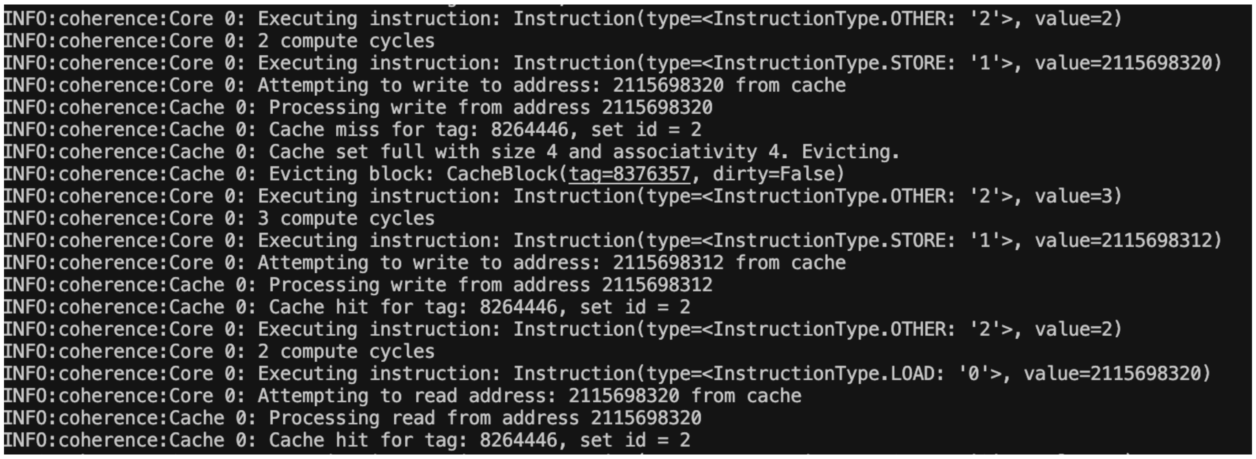
```
python -m coherence Dragon data/blackscholes_0.data 1024 4 64
```

Note that for part 1 coherence protocol is irrelevant. Also note that the -m is important for the packaging to work properly.

Outputs
Running using this configuration:
```
python -m coherence Dragon data/blackscholes_0.data 1024 4 64
```

my program prints the following outputs:

```
Core: 0: 57144563 Execution cycles
Core: 0: 10430314 Compute cycles
Core: 0: 2497349 load/store instructions
Core: 0: 46714249 idle cycles
Core: 0: 2154589 cache hits
Core: 0: 342760 cache misses
```

My program also has logs that show what the cache and core are doing, for e.g what instructions are being run, whether there's a cache hit/miss or write back happening.



```
INFO:coherence:Core 0: Executing instruction: Instruction(type=<InstructionType.OTHER: '2'>, value=2)
INFO:coherence:Core 0: 2 compute cycles
INFO:coherence:Core 0: Executing instruction: Instruction(type=<InstructionType.STORE: '1'>, value=2115698320)
INFO:coherence:Core 0: Attempting to write to address: 2115698320 from cache
INFO:coherence:Cache 0: Processing write from address 2115698320
INFO:coherence:Cache 0: Cache miss for tag: 8264446, set id = 2
INFO:coherence:Cache 0: Cache set full with size 4 and associativity 4. Evicting.
INFO:coherence:Cache 0: Evicting block: CacheBlock(tag=8376357, dirty=False)
INFO:coherence:Core 0: Executing instruction: Instruction(type=<InstructionType.OTHER: '2'>, value=3)
INFO:coherence:Core 0: 3 compute cycles
INFO:coherence:Core 0: Executing instruction: Instruction(type=<InstructionType.STORE: '1'>, value=2115698312)
INFO:coherence:Core 0: Attempting to write to address: 2115698312 from cache
INFO:coherence:Cache 0: Processing write from address 2115698312
INFO:coherence:Cache 0: Cache hit for tag: 8264446, set id = 2
INFO:coherence:Core 0: Executing instruction: Instruction(type=<InstructionType.OTHER: '2'>, value=2)
INFO:coherence:Core 0: 2 compute cycles
INFO:coherence:Core 0: Executing instruction: Instruction(type=<InstructionType.LOAD: '0'>, value=2115698320)
INFO:coherence:Core 0: Attempting to read address: 2115698320 from cache
INFO:coherence:Cache 0: Processing read from address 2115698320
INFO:coherence:Cache 0: Cache hit for tag: 8264446, set id = 2
```

Implementation

The hard part about this assignment for me was implementing the cache (implemented in cache.py), as it had to allow for different configurations such as the associativity, block size and cache size. This meant that given a memory address we had to be able to calculate the tag, set index and offset.

I implemented a CacheSet by using a dictionary of tag to CacheBlock, where CacheBlock contains just tag and dirty fields. To implement LRU eviction policy, I used a doubly linked list data structure where the least recently used cache block is always at the tail of the list. This is similar to the optima solution of the Leetcode LRU question which I am all too familiar with. The dirty field in the CacheBlock is used to know if we need to write to memory when a CacheBlock is evicted, thus allowing for our write back and write allocate policy.