CS211
Yao Shi

Explicity:

- All test cases pass.
- LRU and FIFO work properly.
- Write through and write back work properly.

FIFO and write through:

- If (R)hit, increase the number of hits by one.
- If (W)hit, increase the number of hits, the number of reads and number of writes by one.
- If (R) miss, increase the number of misses and the number of reads by one.
- If (W) miss, increase the number of misses, the number of writes and the number of reads by one.
- If the set is full, remove the last one in the set if we need to and pull the new tag in the front.

LRU and write through:

- If (R)hit, increase the number of hits by one.
- If (W)hit, increase the number of hits, the number of reads and number of writes by one.
- If (R) miss, increase the number of misses and the number of reads by one.
- If (W) miss, increase the number of misses, the number of writes and the number of reads by one.
- If the set is full, remove the last one in the set if we need to and pull the new tag in the front. Since I implement the cache using array of linked list, whenever there is a hit, remove the corresponding node, create a new node with the same tag and place the new node in the front of the list.

FIFO and write back:

- If (R)hit, increase the number of hits by one by one if the dirty bit is 1.
- If (W)hit, increase the number of hits, and the number of reads by one.
- If (R) miss, increase the number of misses and the number of reads by one.
- If (W) miss, increase the number of misses and the number of reads by one.
- In the cache, W always brings a node with the dirty bit as 1.
- If the set is full, remove the last one in the set if we need to and pull the new tag in the front. Whenever writes(W) and whenever modify the cache, mark the corresponding cache's "dirty bit" as 1.

LRU and write back:

- If (R)hit, increase the number of hits by one by one if the dirty bit is 1.
- If (W)hit, increase the number of hits, and the number of reads by one.
- If (R) miss, increase the number of misses and the number of reads by one.
- If (W) miss, increase the number of misses and the number of reads by one.
- In the cache, W always brings a node with the dirty bit as 1.
- If the set is full, remove the last one in the set if we need to and pull the new tag in the front. Since I implement the cache using array of linked list, whenever there is a hit, remove the corresponding node, create a new node with the same tag and place the new node in the front of the list.
- Whenever writes(W) and whenever modify the cache, mark the corresponding cache's "dirty bit" as 1

Regular cache indexing:

```
./c-sim 32 assoc:2 4 FIFO wt trace1.txt
Memory reads:336
Memory writes:334
Cache hits:664
Cache misses:336
```

Miss rate = 0.336 Hit rate = 0.664

```
./c-sim 32 assoc:2 4 LRU wt trace1.txt
Memory reads:336
Memory writes:334
Cache hits:664
Cache misses:336
```

Miss rate = 0.336 Hit rate = 0.664

```
./c-sim 32 assoc:2 4 FIFO wb trace1.txt
Memory reads:336
Memory writes:330
Cache hits:664
Cache misses:336
```

Miss rate = 0.336 Hit rate = 0.664

```
./c-sim 32 assoc:2 4 LRU wb trace1.txt
Memory reads:336
Memory writes:659
Cache hits:664
Cache misses:336
```

Miss rate = 0.336 Hit rate = 0.664

```
./c-sim 32 assoc:2 4 FIFO wt trace2.txt
Memory reads:3499
Memory writes:2861
Cache hits:6501
Cache misses:3499
```

Miss rate = 0.3499      Hit rate = 0.6501

```
./c-sim 32 assoc:2 4 LRU wt trace2.txt
Memory reads:3292
Memory writes:2861
Cache hits:6708
Cache misses:3292
```

Miss rate = 0.3292      Hit rate = 0.6708

```
./c-sim 32 assoc:2 4 FIFO wb trace2.txt
Memory reads:3499
Memory writes:6532
Cache hits:6501
Cache misses:3499
```

Miss rate = 0.3499      Hit rate = 0.6501

```
./c-sim 32 assoc:2 4 LRU wb trace2.txt
Memory reads:3292
Memory writes:2859
Cache hits:6708
Cache misses:3292
```

Miss rate = 0.3292      Hit rate = 0.6708

## High-order cache indexing:

```
./c-sim 32 assoc:2 4 FIFO wt trace1.txt
Memory reads:336
Memory writes:334
Cache hits:664
Cache misses:336
```

Hit rate: 0.664 Miss rate: 0.336

```
./c-sim 32 assoc:2 4 FIFO wb trace1.txt
Memory reads:336
Memory writes:661
Cache hits:664
Cache misses:336
```

Hit rate: 0.664 Miss rate: 0.336

```
./c-sim 32 assoc:2 4 LRU wt trace1.txt
Memory reads:336
Memory writes:334
Cache hits:664
Cache misses:336
```

Hit rate: 0.664 Miss rate: 0.336

```
./c-sim 32 assoc:2 4 LRU wb trace1.txt
Memory reads:336
Memory writes:332
Cache hits:664
Cache misses:336
```

Hit rate: 0.664 Miss rate: 0.336

```
./c-sim 32 assoc:2 4 FIFO wt trace2.txt
Memory reads:3942
Memory writes:2861
Cache hits:6058
Cache misses:3942
Hit rate: 0.6058                    Miss rate: 0.3942

./c-sim 32 assoc:2 4 FIFO wb trace2.txt
Memory reads:3942
Memory writes:6098
Cache hits:6058
Cache misses:3942
Hit rate: 0.6058                    Miss rate: 0.3942

./c-sim 32 assoc:2 4 LRU wt trace2.txt
Memory reads:3943
Memory writes:2861
Cache hits:6057
Cache misses:3943
Hit rate: 0.6057                    Miss rate: 0.3943

./c-sim 32 assoc:2 4 LRU wb trace2.txt
Memory reads:3943
Memory writes:2861
Cache hits:6057
Cache misses:3943
Hit rate: 0.6057                    Miss rate: 0.3943
```

Having the middle bits as index will generally give us the better performance in terms of cache hit rate for larger files. Having high-order bit indexing will results that adjacent memory lines would map to the same cache entry and poor use of spatial locality. If we have middle-order bit indexing, the consecutive memory lines map to different cache lines, then we will have a high chance to get hits(same tags may be in different sets). Middle-order bits tend to change a lot when I traverse the binary string while high-order bits tend to stay the same for those operations. In order to take advantage of different middle-order bits to store a set with good locality in a cache at the same time, so we can get as many as cache hits together, which means that the cache hit ratio is higher when we have the index bits in the middle.