Computer Organization

Detailed description of the implementation:

Direct Mapped Cache

```
1.  利用 struct 當作一個 cache block，存放 valid bit 和 tag
2.  雙層 for 迴圈以不同 block size 和 cache size 執行相同的 memory traces：
3.          先讓 num_of_hit, num_of_miss = 0
4.          計算 num_of_blocks = cache_size / block_size 並初始化 cache
5.          對每一條 memory trace 做下列事情：
6.                  block_addr = byte_addr / block_size
7.                  block_num = block_addr % num_of_blocks
8.                  block_tag = block_addr / num_of_blocks
9.                  如果 valid bit=1 且 tag 相同，num_of_hit++
10.                 如果 valid bit=1 且 tag 不同，num_of_miss++，改掉 tag
11.                 如果 valid bit=0，num_of_miss++，改掉 tag，讓 valid bit=1
12.         計算 hit_rate = num_of_hit / num_of_mem_traces
13.         計算 miss_rate = num_of_miss / num_of_mem_traces
```

N-way Set Associative Cache

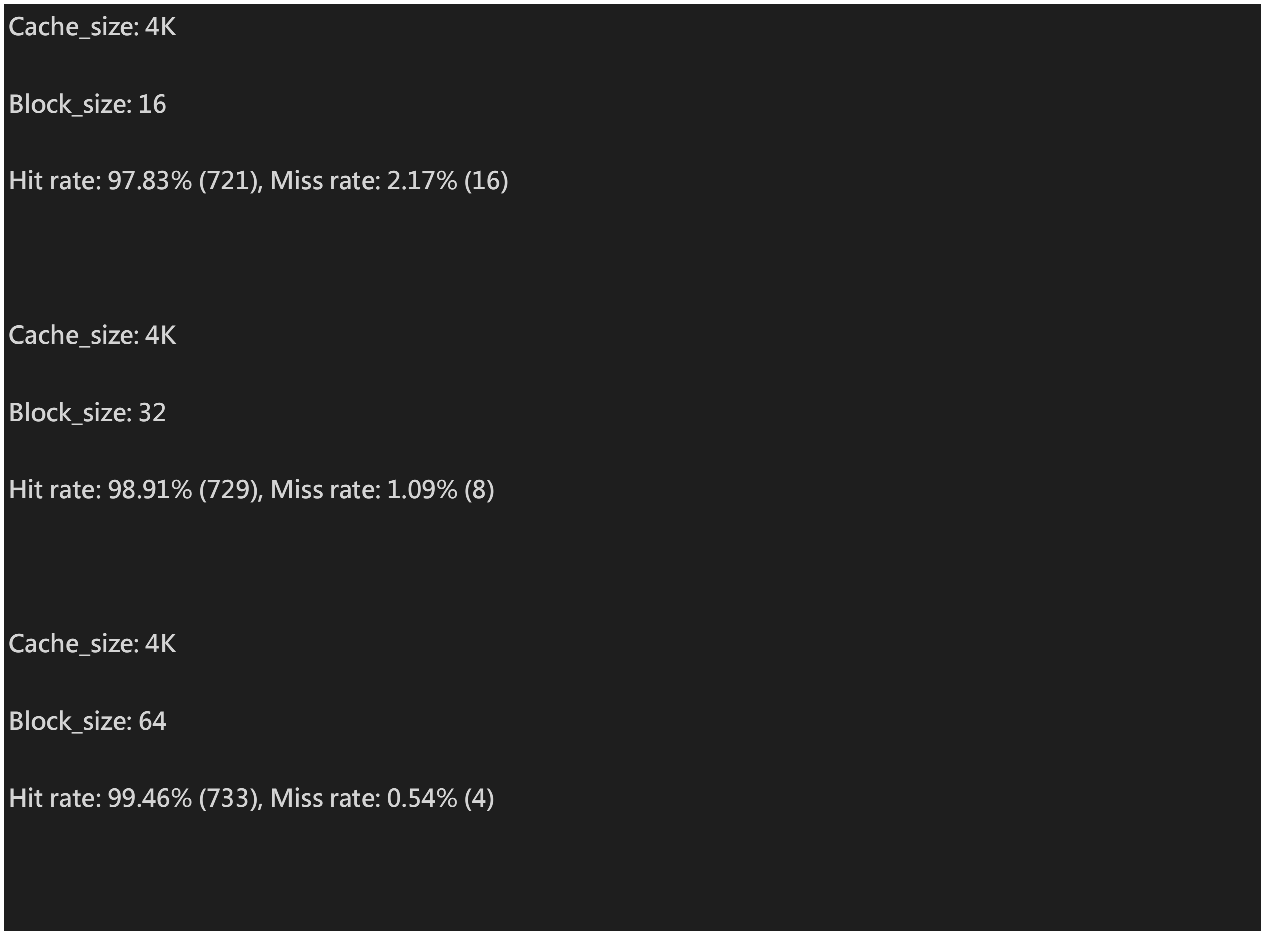
```
1.  利用 struct 當作一個 cache block，存放 valid bit、tag 和 used
2.  雙層 for 迴圈以不同 associativity 和 cache size 執行相同的 memory traces：
3.          先讓 num_of_hit, num_of_miss = 0
4.          計算 num_of_blocks = cache_size / block_size
5.          計算 num_of_sets = num_of_blocks / associativity 並初始化 cache
6.          對每一條 memory trace 做下列事情：
7.                  block_addr = byte_addr / block_size
8.                  set_num = block_addr % num_of_sets
```

```
9.          block_tag = block_addr / num_of_sets
10.         對 set 內每個 block 做下列事情：
11.             若 valid bit=1 且 tag 相同，num_of_hit++，更新 used
12.             同時尋找此 set 內的 least used block
13.         如果 miss，做下列事情：
14.             若有 valid=0 的 block，直接替換，更新 valid bit、tag 和 used
15.             若全都在使用中，替換掉剛剛找到的 least used block
16.     計算 hit_rate = num_of_hit / num_of_mem_traces
17.     計算 miss_rate = num_of_miss / num_of_mem_traces
```

Implementation results:

<u>Direct Mapped - ICACHE</u>

- Output Result



```
Cache_size: 4K

Block_size: 16

Hit rate: 97.83% (721), Miss rate: 2.17% (16)



Cache_size: 4K

Block_size: 32

Hit rate: 98.91% (729), Miss rate: 1.09% (8)



Cache_size: 4K

Block_size: 64

Hit rate: 99.46% (733), Miss rate: 0.54% (4)
```

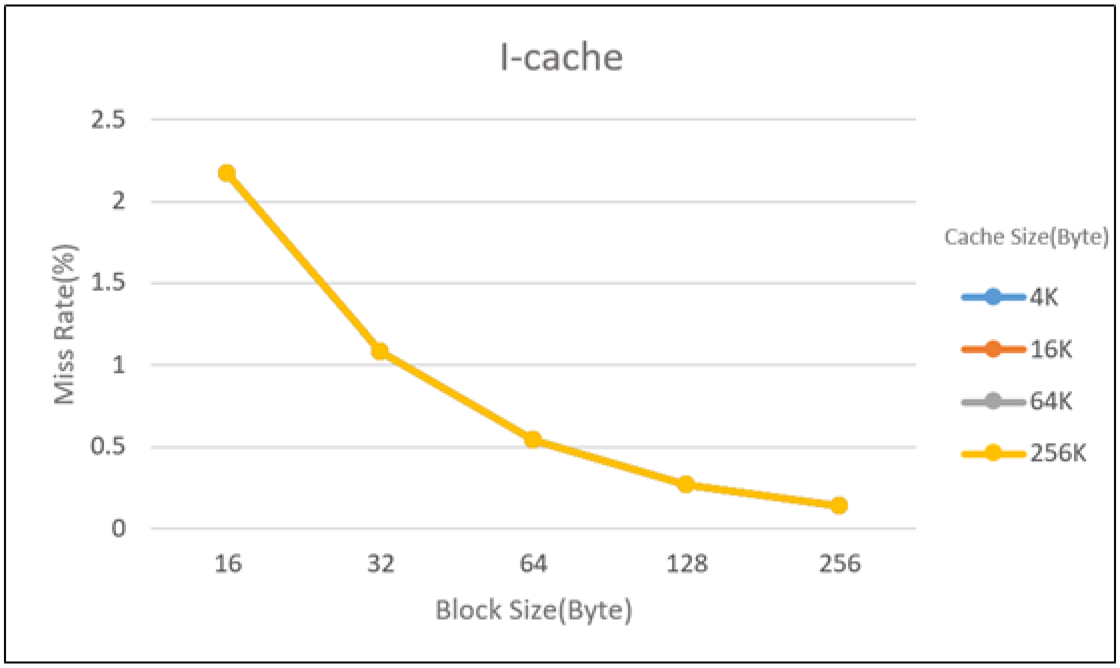
```
Cache_size: 4K

Block_size: 128

Hit rate: 99.73% (735), Miss rate: 0.27% (2)


Cache_size: 4K

Block_size: 256

Hit rate: 99.86% (736), Miss rate: 0.14% (1)


Cache_size: 16K

Block_size: 16

Hit rate: 97.83% (721), Miss rate: 2.17% (16)


Cache_size: 16K

Block_size: 32

Hit rate: 98.91% (729), Miss rate: 1.09% (8)


Cache_size: 16K

Block_size: 64

Hit rate: 99.46% (733), Miss rate: 0.54% (4)


Cache_size: 16K

Block_size: 128
```

```
Hit rate: 99.73% (735), Miss rate: 0.27% (2)


Cache_size: 16K

Block_size: 256

Hit rate: 99.86% (736), Miss rate: 0.14% (1)


Cache_size: 64K

Block_size: 16

Hit rate: 97.83% (721), Miss rate: 2.17% (16)


Cache_size: 64K

Block_size: 32

Hit rate: 98.91% (729), Miss rate: 1.09% (8)


Cache_size: 64K

Block_size: 64

Hit rate: 99.46% (733), Miss rate: 0.54% (4)


Cache_size: 64K

Block_size: 128

Hit rate: 99.73% (735), Miss rate: 0.27% (2)
```

```
Cache_size: 64K

Block_size: 256

Hit rate: 99.86% (736), Miss rate: 0.14% (1)


Cache_size: 256K

Block_size: 16

Hit rate: 97.83% (721), Miss rate: 2.17% (16)


Cache_size: 256K

Block_size: 32

Hit rate: 98.91% (729), Miss rate: 1.09% (8)


Cache_size: 256K

Block_size: 64

Hit rate: 99.46% (733), Miss rate: 0.54% (4)


Cache_size: 256K

Block_size: 128

Hit rate: 99.73% (735), Miss rate: 0.27% (2)


Cache_size: 256K

Block_size: 256
```

```
Hit rate: 99.86% (736), Miss rate: 0.14% (1)
```

- Graph



- The reason for rise and fall

因為 block size 變大，所以 spatial locality 變好，降低 compulsory misses。尤其 Instruction 通常都是 sequential 的執行，所以 block size 越大 miss rate 越低。

Direct Mapped - DCACHE

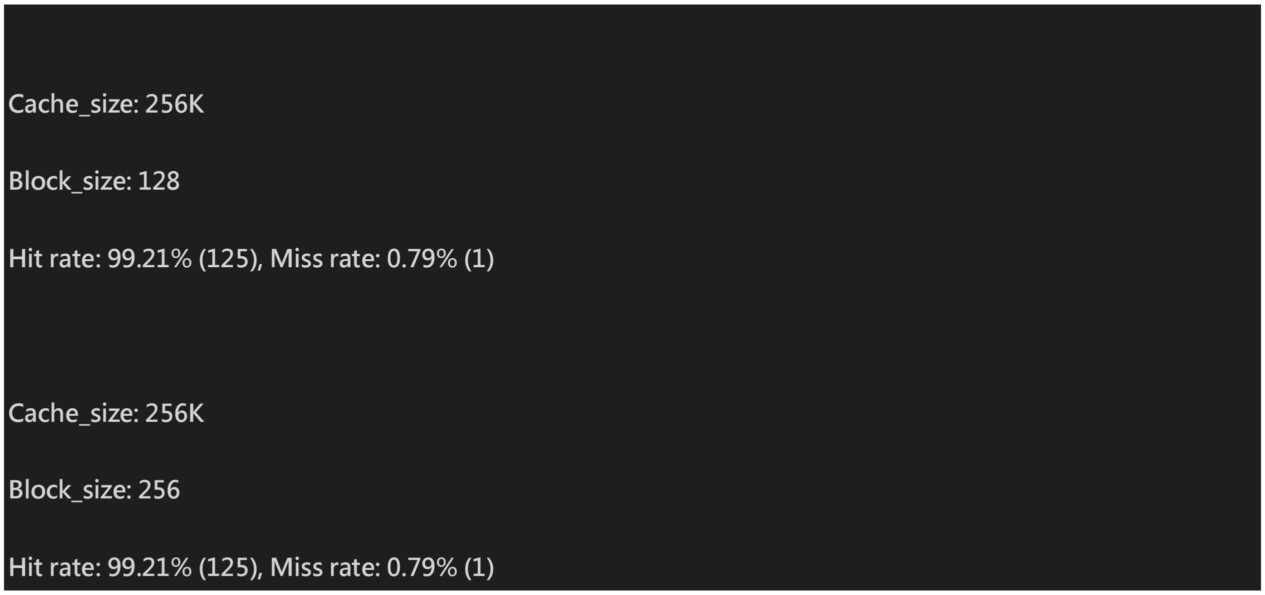
- Output Result

```
Cache_size: 4K

Block_size: 16

Hit rate: 94.44% (119), Miss rate: 5.56% (7)



Cache_size: 4K
```

```
Block_size: 32

Hit rate: 96.83% (122), Miss rate: 3.17% (4)


Cache_size: 4K

Block_size: 64

Hit rate: 98.41% (124), Miss rate: 1.59% (2)


Cache_size: 4K

Block_size: 128

Hit rate: 99.21% (125), Miss rate: 0.79% (1)


Cache_size: 4K

Block_size: 256

Hit rate: 99.21% (125), Miss rate: 0.79% (1)


Cache_size: 16K

Block_size: 16

Hit rate: 94.44% (119), Miss rate: 5.56% (7)


Cache_size: 16K

Block_size: 32

Hit rate: 96.83% (122), Miss rate: 3.17% (4)
```

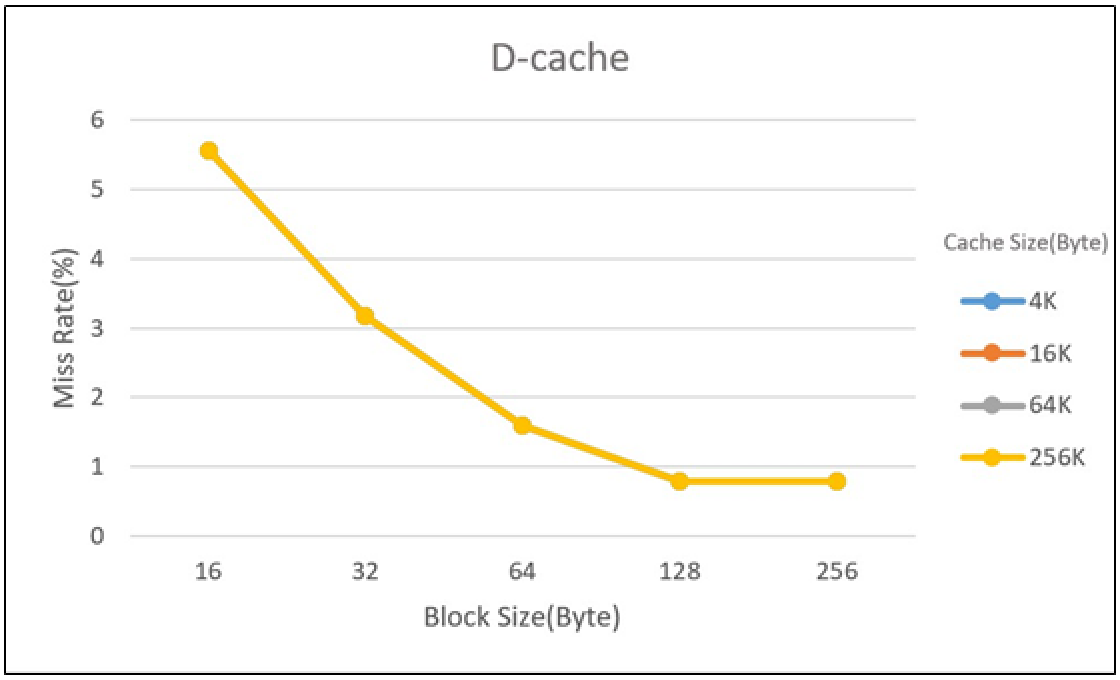
```
Cache_size: 16K

Block_size: 64

Hit rate: 98.41% (124), Miss rate: 1.59% (2)


Cache_size: 16K

Block_size: 128

Hit rate: 99.21% (125), Miss rate: 0.79% (1)


Cache_size: 16K

Block_size: 256

Hit rate: 99.21% (125), Miss rate: 0.79% (1)


Cache_size: 64K

Block_size: 16

Hit rate: 94.44% (119), Miss rate: 5.56% (7)


Cache_size: 64K

Block_size: 32

Hit rate: 96.83% (122), Miss rate: 3.17% (4)


Cache_size: 64K
```

```
Block_size: 64

Hit rate: 98.41% (124), Miss rate: 1.59% (2)



Cache_size: 64K

Block_size: 128

Hit rate: 99.21% (125), Miss rate: 0.79% (1)



Cache_size: 64K

Block_size: 256

Hit rate: 99.21% (125), Miss rate: 0.79% (1)



Cache_size: 256K

Block_size: 16

Hit rate: 94.44% (119), Miss rate: 5.56% (7)



Cache_size: 256K

Block_size: 32

Hit rate: 96.83% (122), Miss rate: 3.17% (4)



Cache_size: 256K

Block_size: 64

Hit rate: 98.41% (124), Miss rate: 1.59% (2)
```



```
Cache_size: 256K

Block_size: 128

Hit rate: 99.21% (125), Miss rate: 0.79% (1)



Cache_size: 256K

Block_size: 256

Hit rate: 99.21% (125), Miss rate: 0.79% (1)
```

- Graph



- The reason for rise and fall

因為 block size 變大，所以 spatial locality 變好，降低 compulsory misses，miss rate 降低。但是 data 可能 sequential 的需求量沒那麼大，所以 block size 是 128 或 256 的 miss rate 是一樣的。若資料的多樣性再大一些，由於 temporal locality 變差，也許會造成 miss rate 上升。

N-way Set Associative Cache

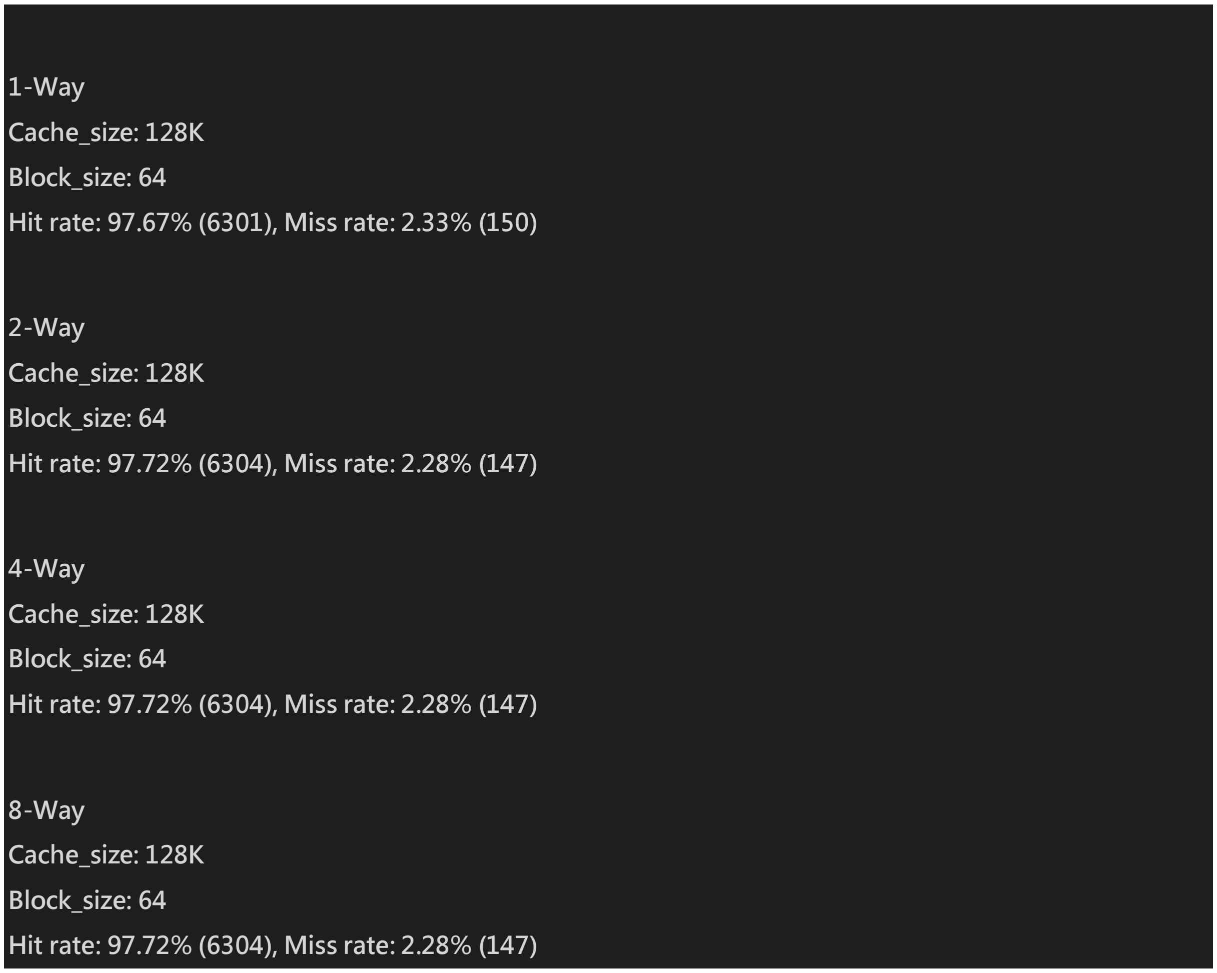
- Output Result

```
1-Way
Cache_size: 1K
Block_size: 64
Hit rate: 88.93% (5737), Miss rate: 11.07% (714)

2-Way
Cache_size: 1K
Block_size: 64
Hit rate: 91.64% (5912), Miss rate: 8.36% (539)

4-Way
Cache_size: 1K
Block_size: 64
Hit rate: 92.22% (5949), Miss rate: 7.78% (502)

8-Way
Cache_size: 1K
Block_size: 64
Hit rate: 92.17% (5946), Miss rate: 7.83% (505)

1-Way
Cache_size: 2K
Block_size: 64
Hit rate: 91.72% (5917), Miss rate: 8.28% (534)

2-Way
Cache_size: 2K
Block_size: 64
Hit rate: 94.82% (6117), Miss rate: 5.18% (334)

4-Way
Cache_size: 2K
Block_size: 64
```

```
Hit rate: 95.81% (6181), Miss rate: 4.19% (270)

8-Way
Cache_size: 2K
Block_size: 64
Hit rate: 96.02% (6194), Miss rate: 3.98% (257)

1-Way
Cache_size: 4K
Block_size: 64
Hit rate: 94.53% (6098), Miss rate: 5.47% (353)

2-Way
Cache_size: 4K
Block_size: 64
Hit rate: 96.37% (6217), Miss rate: 3.63% (234)

4-Way
Cache_size: 4K
Block_size: 64
Hit rate: 96.93% (6253), Miss rate: 3.07% (198)

8-Way
Cache_size: 4K
Block_size: 64
Hit rate: 97.19% (6270), Miss rate: 2.81% (181)

1-Way
Cache_size: 8K
Block_size: 64
Hit rate: 95.97% (6191), Miss rate: 4.03% (260)

2-Way
Cache_size: 8K
Block_size: 64
Hit rate: 97.02% (6259), Miss rate: 2.98% (192)
```

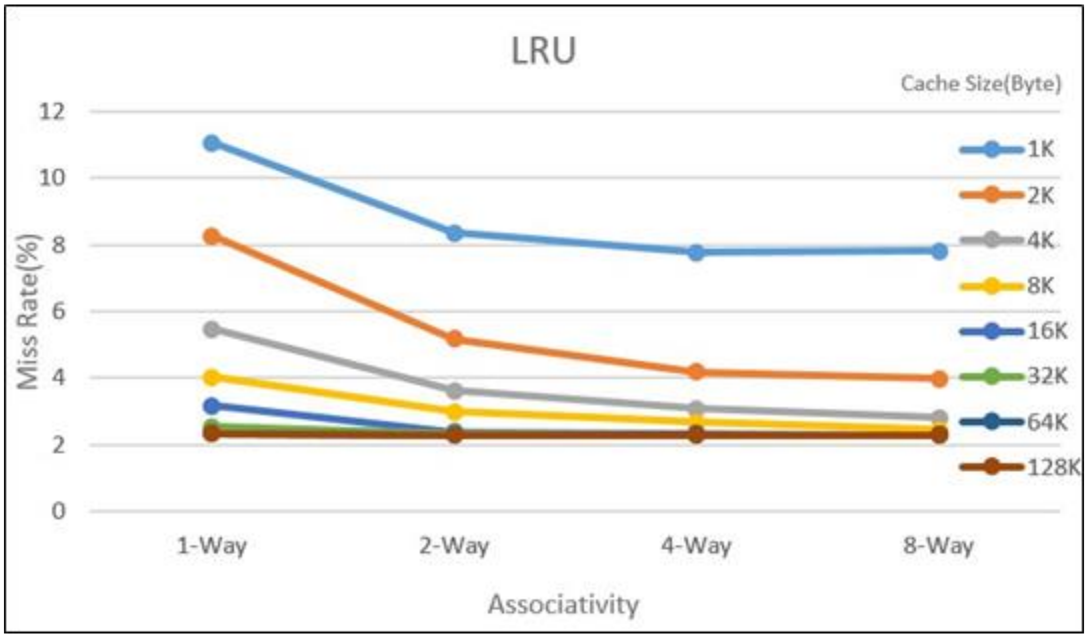
```
4-Way
Cache_size: 8K
Block_size: 64
Hit rate: 97.33% (6279), Miss rate: 2.67% (172)

8-Way
Cache_size: 8K
Block_size: 64
Hit rate: 97.55% (6293), Miss rate: 2.45% (158)

1-Way
Cache_size: 16K
Block_size: 64
Hit rate: 96.84% (6247), Miss rate: 3.16% (204)

2-Way
Cache_size: 16K
Block_size: 64
Hit rate: 97.63% (6298), Miss rate: 2.37% (153)

4-Way
Cache_size: 16K
Block_size: 64
Hit rate: 97.66% (6300), Miss rate: 2.34% (151)

8-Way
Cache_size: 16K
Block_size: 64
Hit rate: 97.71% (6303), Miss rate: 2.29% (148)

1-Way
Cache_size: 32K
Block_size: 64
Hit rate: 97.46% (6287), Miss rate: 2.54% (164)
```

```
2-Way
Cache_size: 32K
Block_size: 64
Hit rate: 97.67% (6301), Miss rate: 2.33% (150)

4-Way
Cache_size: 32K
Block_size: 64
Hit rate: 97.72% (6304), Miss rate: 2.28% (147)

8-Way
Cache_size: 32K
Block_size: 64
Hit rate: 97.72% (6304), Miss rate: 2.28% (147)

1-Way
Cache_size: 64K
Block_size: 64
Hit rate: 97.66% (6300), Miss rate: 2.34% (151)

2-Way
Cache_size: 64K
Block_size: 64
Hit rate: 97.71% (6303), Miss rate: 2.29% (148)

4-Way
Cache_size: 64K
Block_size: 64
Hit rate: 97.72% (6304), Miss rate: 2.28% (147)

8-Way
Cache_size: 64K
Block_size: 64
Hit rate: 97.72% (6304), Miss rate: 2.28% (147)
```



```
1-Way
Cache_size: 128K
Block_size: 64
Hit rate: 97.67% (6301), Miss rate: 2.33% (150)

2-Way
Cache_size: 128K
Block_size: 64
Hit rate: 97.72% (6304), Miss rate: 2.28% (147)

4-Way
Cache_size: 128K
Block_size: 64
Hit rate: 97.72% (6304), Miss rate: 2.28% (147)

8-Way
Cache_size: 128K
Block_size: 64
Hit rate: 97.72% (6304), Miss rate: 2.28% (147)
```

- Graph



● Table

Cache Size \ Associativity	1-way	2-way	4-way	8-way
1KB	11.07%	8.36%	7.78%	7.83%
2KB	8.28%	5.18%	4.19%	3.98%
4KB	5.47%	3.63%	3.07%	2.81%
8KB	4.03%	2.98%	2.67%	2.45%
16KB	3.16%	2.37%	2.34%	2.29%
32KB	2.54%	2.33%	2.28%	2.28%
64KB	2.34%	2.29%	2.28%	2.28%
128KB	2.33%	2.28%	2.28%	2.28%

● The reason for rise and fall

associativity 越大代表一個 set 能容忍彈性空間的寬度越大，代表有更多相同 index 的 block 可以被保留，因此降低 conflict misses 的數量，也降低 miss rate。cache size 變大可以增加容納的空間，block 數量和 set 數量都增加，降低 capacity misses，進而降低整體 miss rate。

Problems encountered and solutions:

Q: 算 miss rate 跟 hit rate 時要用小數來表示，但因為 num_of_hit/miss 與 num_of_mem_traces 都是整數，導致除出來的結果會是整數。

A: 一開始我們將 float 放在商前面還是無法解決，後來發現要先把除數和被除數分別轉成 float 之後再相除。

Q: 第一個版本的 LRU，associativity 變大會導致 miss rate 變大。

A: 一開始我們用一個整數代表使用時間，越久沒用數字越大，第一次放進 cache 或是再次使用都將他歸零，但不知道為什麼沒有成功。後來第二版改成發生 miss 的時候先看有沒有空的位置可以放，以及把用的時間變成 trace id，才做出正確的 LRU。

Comment:

這次的作業不需要用到 verilog 反而是用 c++來寫，一開始覺得不知道要從哪裡開始，後來將步驟先寫下來之後才順利地開始。中途測試的時候一直忘了放 txt 檔導致讀檔失敗，害我們一直以為 code 寫錯了，發現的時候快被氣死。之後因為端午連假回家導致我們後半部的作業寫的時候要一直傳訊息覺得很麻煩，如果之後有團體作業的話一定要寫完再回家，不然真的會導致溝通困難，也會浪費很多時間。