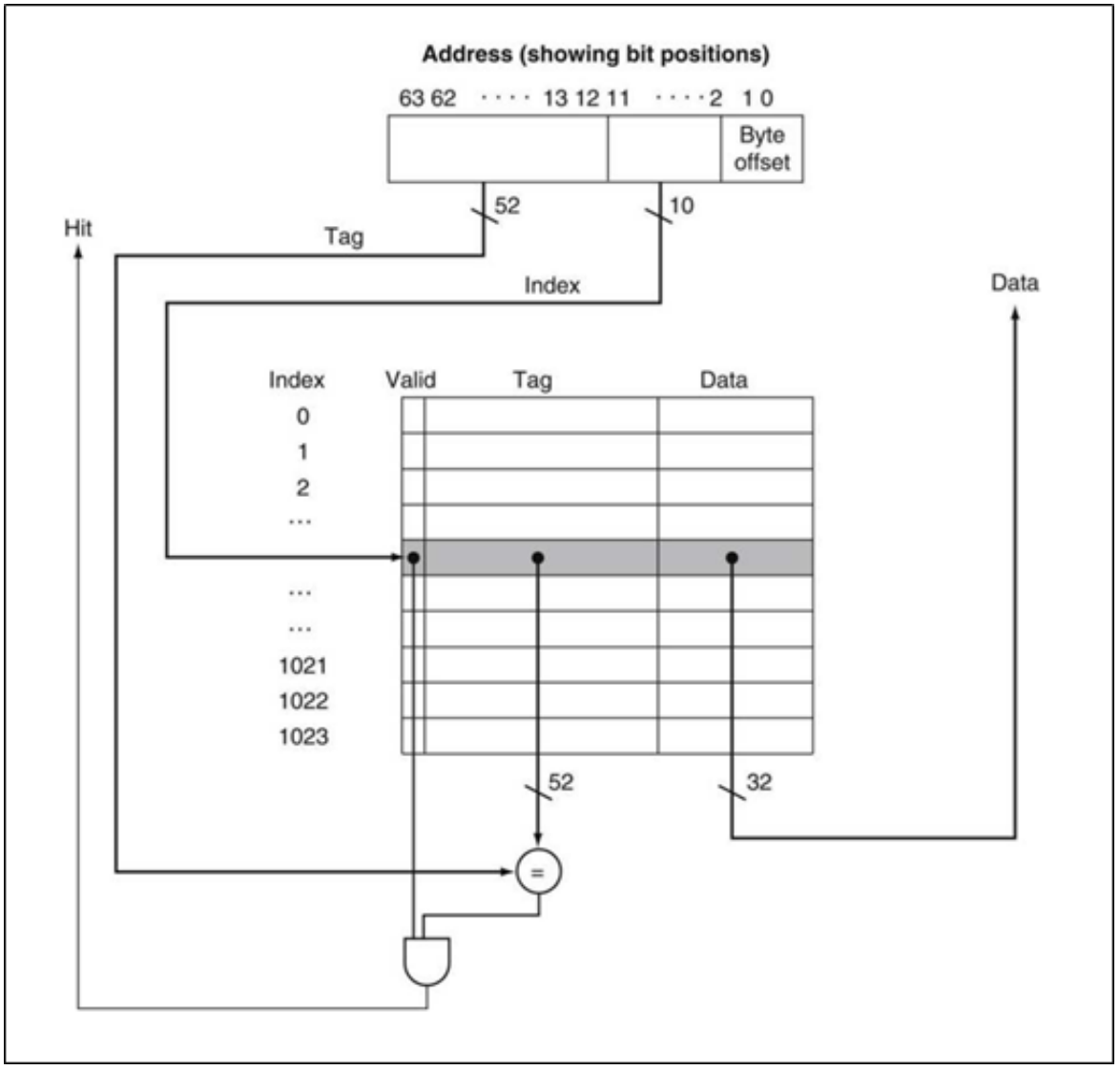
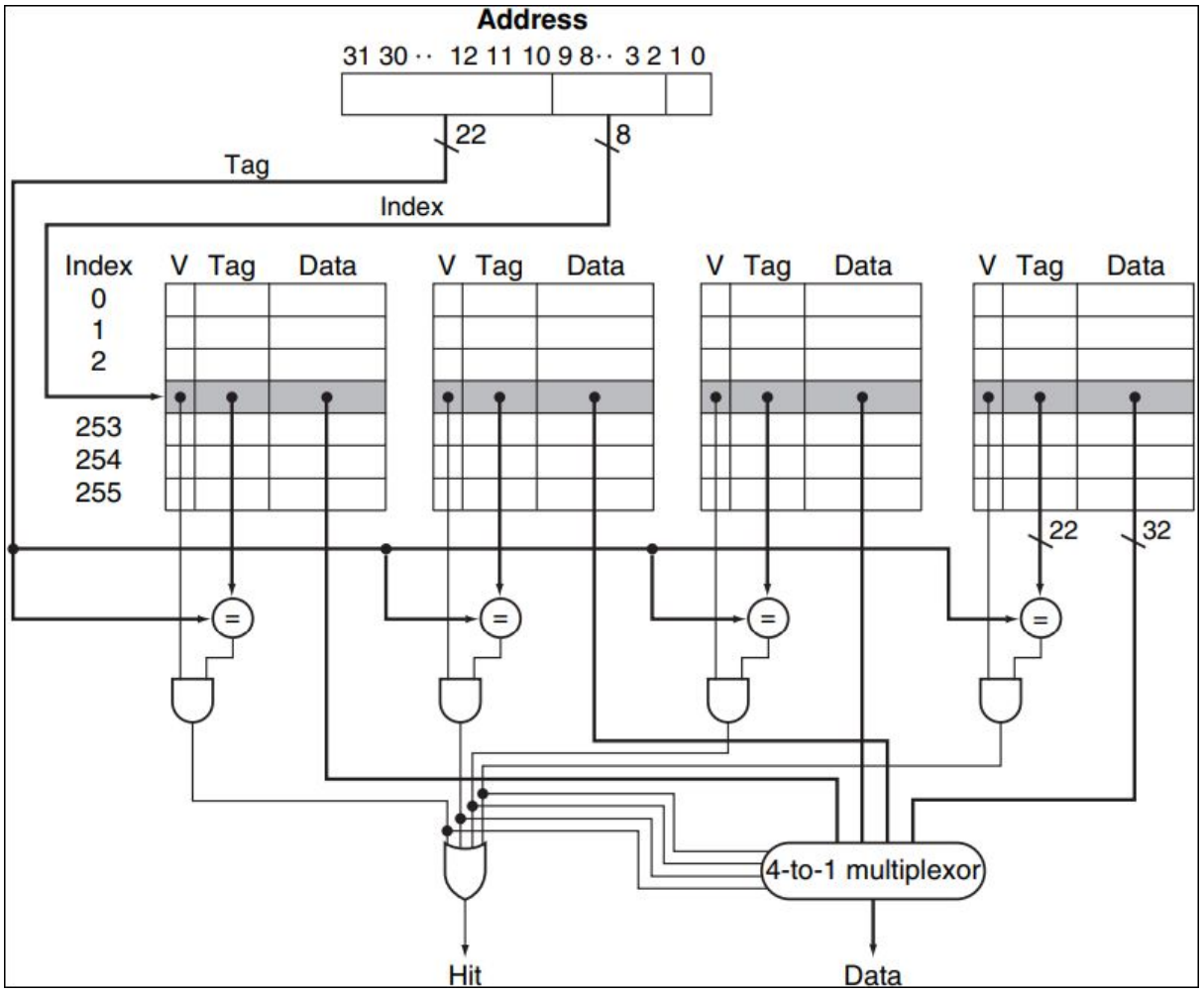
Computer Organization

Architecture diagram：

1. Direct Mapped Cache



2. Set Associative Cache



Detailed description of the implementation：

1. Direct Mapped：

利用Structure去實作Cache，裡面包含Valid bit與tag。

我們先宣告struct Cache的指標，計算完Block lines後（index總數），利用該指標生成出所有的blocks，再去模擬Cache。

2. Set Associative：

我們利用雙層structure去實作Cache，最外層是N_way_Cache，裡面包含sub structure set的指標和LRU List。

(1)先宣告N_way_Cache的指標，根據input的block size、cache size和associativity來決定block lines數目，再利用N_way_Cache的指標生成出需要的block lines（index總數）。

(2)再利用input的Associativity，利用Set指標對每一行的block生成出N個Set（N-way Associative），每一個Set包含1個valid bit與1個tag。1行block包含N個tag & valid bit與1個LRU list。

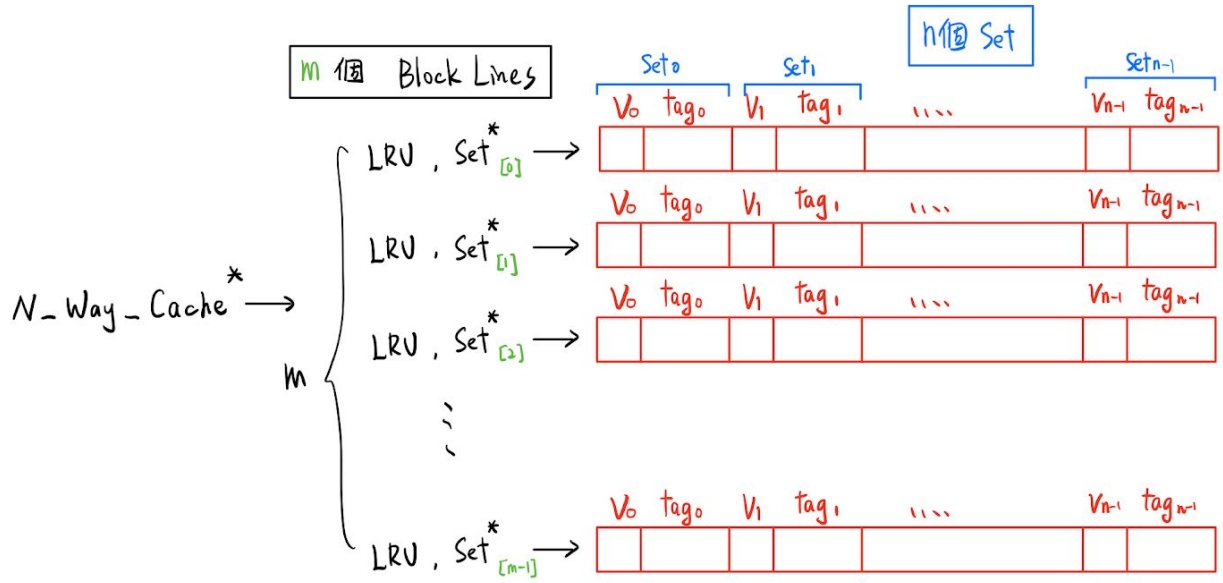
(3)LRU初始化：順序先由0~(N-1)排定，也就是當block的每個set都是空的時候，會由Set0開始填。

(4)讀取檔案我們利用while條件判斷+fscanf讀取每一行測資。

(5)LRU實作

Hit：直接把Hit Set的index從LRU list中remove，再把該index利用push_back()排直LRU list的最後面。

Miss：把LRU list的第一個element 從LRU list中remove（pop_front()是C++ STL中移除第一個element的函式），再把移除掉的element push_back()回LRU list的最後面。




```
Struct N_way_Cache
{
  list LRU;
  Set* set_arr;
}
```

```
Struct Set
{
  bool v;
  int tag;
}
```

Implementation results：

1. Direct Mapped

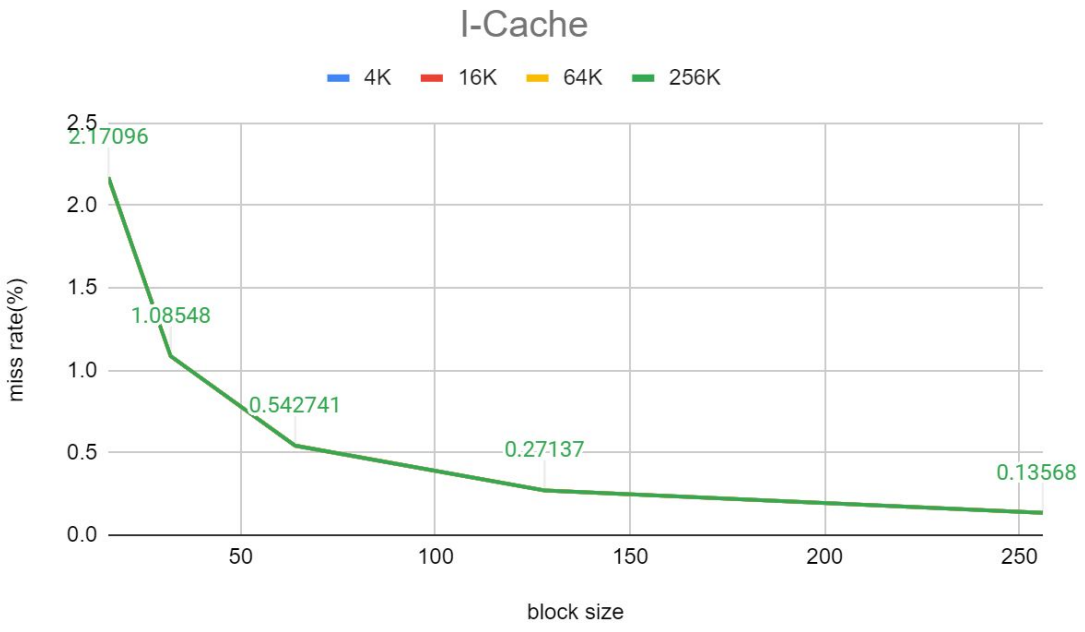
I-Cache

```
~*~*~*~*~ I-Cache ~*~*~*~*~
Cache_Size: 4K
Block_Size: 16
Hit rate: 97.83% (721), Mis rate: 2.17% (16)
Cache_Size: 4K
Block_Size: 32
Hit rate: 98.91% (729), Mis rate: 1.09% (8)
Cache_Size: 4K
Block_Size: 64
Hit rate: 99.46% (733), Mis rate: 0.54% (4)
Cache_Size: 4K
Block_Size: 128
Hit rate: 99.73% (735), Mis rate: 0.27% (2)
Cache_Size: 4K
Block_Size: 256
Hit rate: 99.86% (736), Mis rate: 0.14% (1)

Cache_Size: 16K
Block_Size: 16
Hit rate: 97.83% (721), Mis rate: 2.17% (16)
Cache_Size: 16K
Block_Size: 32
Hit rate: 98.91% (729), Mis rate: 1.09% (8)
Cache_Size: 16K
Block_Size: 64
Hit rate: 99.46% (733), Mis rate: 0.54% (4)
Cache_Size: 16K
Block_Size: 128
Hit rate: 99.73% (735), Mis rate: 0.27% (2)
Cache_Size: 16K
Block_Size: 256
Hit rate: 99.86% (736), Mis rate: 0.14% (1)
```

```
Cache_Size: 64K
Block_Size: 16
Hit rate: 97.83% (721), Mis rate: 2.17% (16)
Cache_Size: 64K
Block_Size: 32
Hit rate: 98.91% (729), Mis rate: 1.09% (8)
Cache_Size: 64K
Block_Size: 64
Hit rate: 99.46% (733), Mis rate: 0.54% (4)
Cache_Size: 64K
Block_Size: 128
Hit rate: 99.73% (735), Mis rate: 0.27% (2)
Cache_Size: 64K
Block_Size: 256
Hit rate: 99.86% (736), Mis rate: 0.14% (1)

Cache_Size: 256K
Block_Size: 16
Hit rate: 97.83% (721), Mis rate: 2.17% (16)
Cache_Size: 256K
Block_Size: 32
Hit rate: 98.91% (729), Mis rate: 1.09% (8)
Cache_Size: 256K
Block_Size: 64
Hit rate: 99.46% (733), Mis rate: 0.54% (4)
Cache_Size: 256K
Block_Size: 128
Hit rate: 99.73% (735), Mis rate: 0.27% (2)
Cache_Size: 256K
Block_Size: 256
Hit rate: 99.86% (736), Mis rate: 0.14% (1)
```



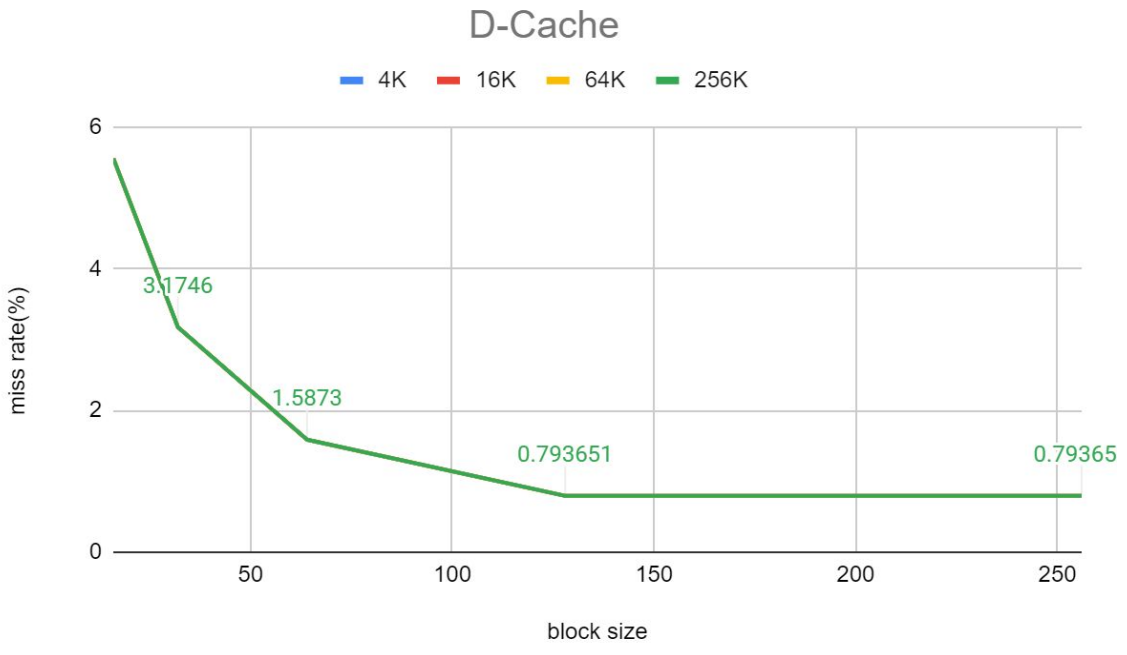
D-Cache

```
~*~*~*~*~ D-Cache ~*~*~*~*~
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)
```



2. Set Associative Cache

1K, 2K

```
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)
```

4K, 8K

```
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)
```

16K, 32K

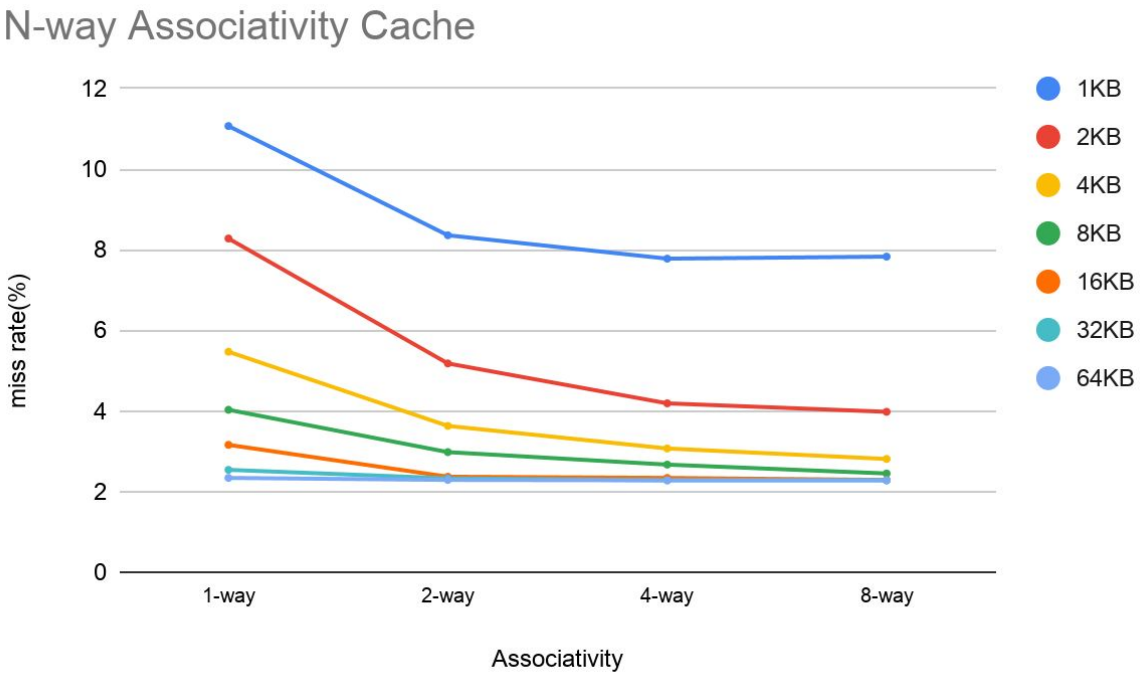
```
1-Way
Cache_Size: 16K
Block_Size: 64
Hit rate: 96.84% (6247), Miss rate: 3.16% (204)
2-Way
Cache_Size: 16K
Block_Size: 64
Hit rate: 97.63% (6298), Miss rate: 2.37% (153)
4-Way
Cache_Size: 16K
Block_Size: 64
Hit rate: 97.66% (6300), Miss rate: 2.34% (151)
8-Way
Cache_Size: 16K
Block_Size: 64
Hit rate: 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)
```

64K

```
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)
```

Miss Rate(%)				
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



Problems encountered and solutions：

1. Direct mapped 不管如何改變 cache size， miss rate 都會相同，也因此所畫出來的圖紙會有一條線。我們一直尋找是哪裡出了問題，後來才知道是因為 test data 的原因。 .

2. Set Associative Cache的LRU實作我們想了很久，一開始想用Array(vector)當作LRU的資料結構記順序，但是發現如果Cache Hit的時後更新LRU會很麻煩，因為Array更改中間元素順序的複雜度很高；後來改用STL裡面的list，雖然不像Array支援隨機存取，但是刪除和修改順序的複雜度比Array低很多，所以後來採用list而非Array實現LRU；另外我們有查到效率更好的資料結構Map去存取LRU，但是不熟悉Map操作怕有很多Bug所以作罷，不然LRU Cache用Map當資料結構去實作會更理想。

3. Set Associative Cache比對Cache是Hit / Miss，因為一地要把所有Set都比對過一遍，所以比對Hit的條件要寫在Loop中，但是Miss就不能寫在Loop中，我們卡了一下子，後來想出用Counter紀錄比對次數，如果比對次數和Set一樣多，那就是全部Set比對完都沒有Hit，這時候就是Miss，再去進行Miss處理。

Comment:

Cache實作過程才發現自己有很多觀念很模糊，例如Block Size和和Associativity會搞混、Direct Mapped Cache的Block怎麼去比對tag、index和計算tag、index和offset的total bit，都逼我們重新檢視自己哪裡觀念不清楚。

我們都很喜歡計算機組織的每次Lab，實作過程中都會發現有些觀念很模糊、不清楚，經過兩人"激烈"的討論後，對整個計算機組織的掌握更高，雖然過程很煎熬、很花時間，但是我們都覺得Lab是這堂課很大的收穫！

這次是這學期最後的一個 Lab，感謝這學期來助教在 Lab 中所提供的一切努力與對我們的幫助，同時後面的 Lab 能夠雙人合作對我們的幫助也非常大，難以想像如果是一個人要面對這些 Lab 會有多麼煎熬。