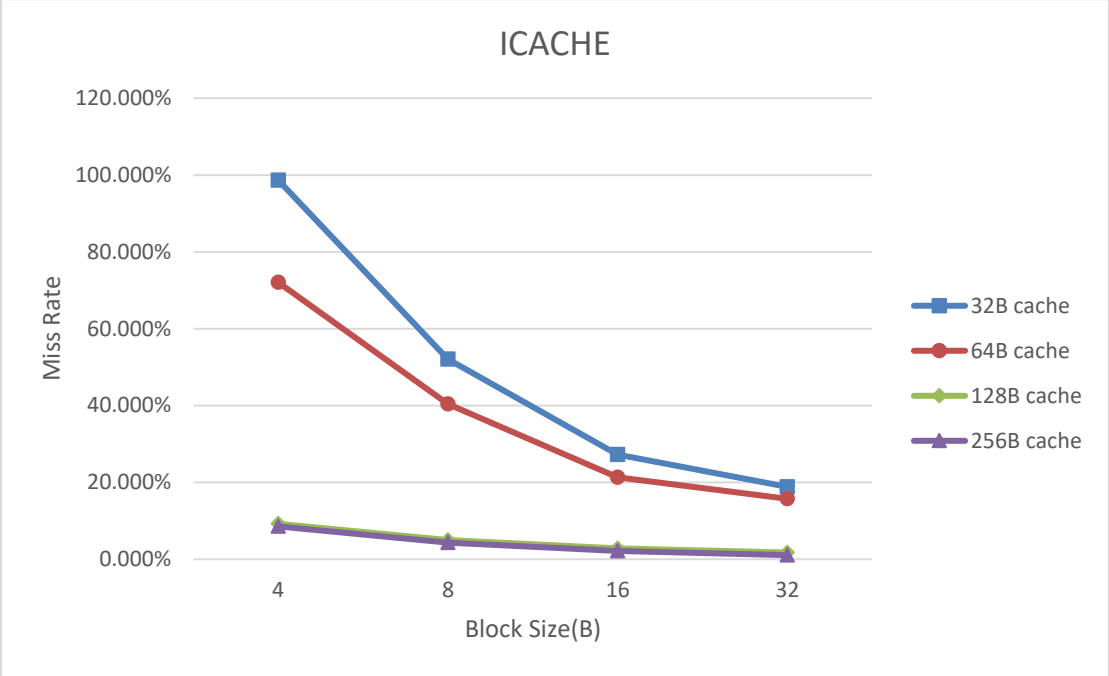
Computer Organization

Basic

1. ICACHE



Miss Rate

Block size(B) / Cache Size(B)	4	8	16	32
32	98.643%	52.103%	27.273%	18.860%
64	72.049%	40.434%	21.303%	15.740%
128	9.227%	5.020%	2.849%	1.764%
256	8.548%	4.342%	2.171%	1.085%

由上面這張圖可以看出，如果增加 Block size，會使得 Miss Rate 減少，但是減少幅度會逐漸趨緩，這是因為 Cache 的大小是固定的，所以會減少 Block 的數量，導致相同 index 的 Block 會有競爭的問題。

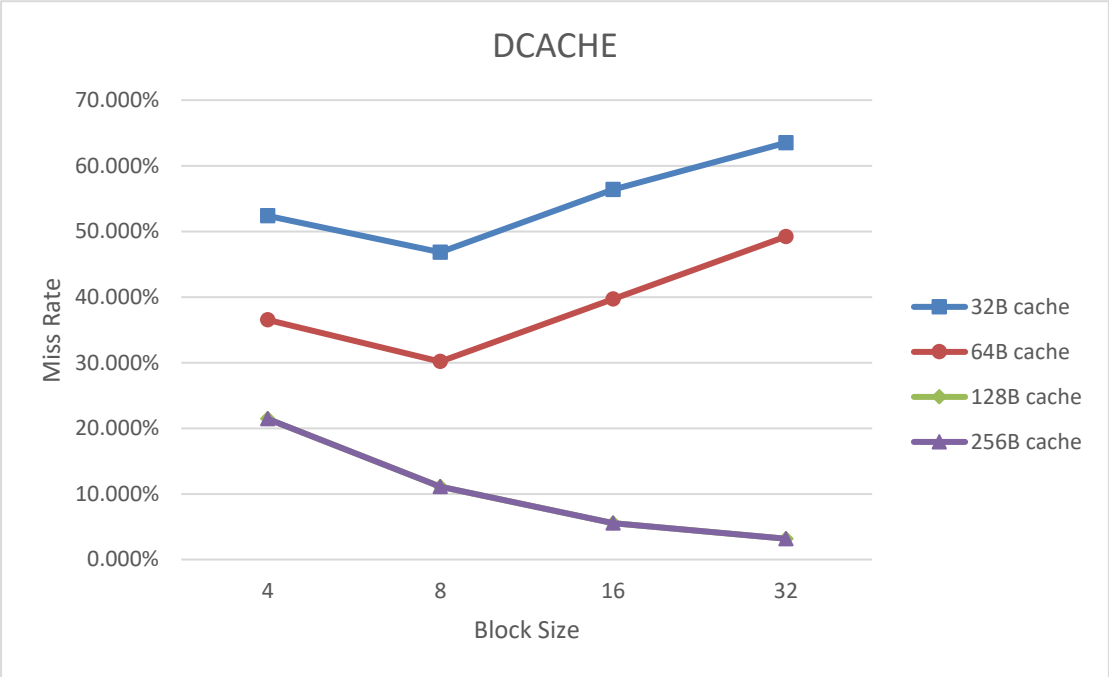
- 增加 Block size 有可能會使 miss rate 降低(Spatial locality)，但影響程度趨緩。

若是增加 Cache 的大小也會使得 Miss Rate 下降(減少 Capacity misses)，不過相對地來說增加 Block size 對於減少 Miss Rate 的影響就沒有那麼大了，因為 Cache size 夠大，因此需要 Replace 的 Block 就會少很多，Miss Rate 的下降幅度也很

小。

- 增加 Cache size 有可能會使 miss rate 降低，但增加到一定程度後影響趨緩。

2. DCACHE



Miss Rate

Block size(B) / Cache Size(B)	4	8	16	32
32	52.318%	46.825%	56.349%	63.492%
64	36.508%	30.159%	39.683%	49.206%
128	21.429%	11.111%	5.556%	3.175%
256	21.429%	11.111%	5.556%	3.175%

在 Cache size 為 32B 與 64B 時，增加 Block size 時 miss rate 會先下降，但後來又會上升，這是因為 D-Cache 相對於 I-Cache 來說，比較不照順序存取，而且 block 的數量也因為 block size 上升而下降，因此常會發生同樣的 index 卻有不同 tag 要存取的狀況，所以 miss rate 就會因此上升。

- 增加 Block size 不一定會使 miss rate 減少，有時反而會上升。

但若是增加 Cache size，就可以降低 miss rate，不過 size 增加到一定程度時，miss rate 就不會再下降了，因為會用到的 Data 有限，只有第一次存取時會 miss，其他時候就不會了，其餘的 index 不會用到。

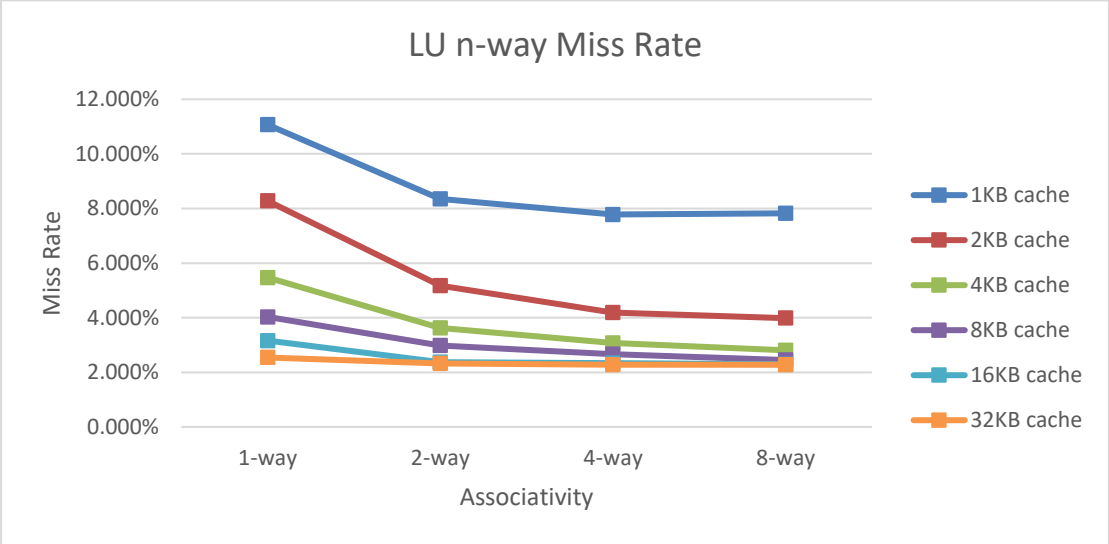
- 增加 Cache size 不一定會使 miss rate 降低。

Advance

direct_mapped_cache_lru.cpp

這個程式主要是修改 direct_mapped_cache.cpp，加了其他 n-way 的 struct，而且還增加了 LRU(Least recently used)，LRU 為此次 cache 中 replacement policy 所使用的方式，如果出現 cache hit 就先記錄原本 hit 的 block 的 LRU(lastLRU)，並將 hit block 的 LRU 設為最高的 bit，並將其餘大於 lastLRU 的 block 的 LRU 減 1。
若是沒有找到相符合的 tags 與 valid 的話，就出現 cache miss 的情況，這樣就必須找到 LRU 最小的(最近沒有被使用到)的 block 做更換，將 tags 改掉，並將 LRU 設為最高的 bit，將其餘 blocks 的 LRU 減 1。

1. LU n-way



LU n-way Miss Rate

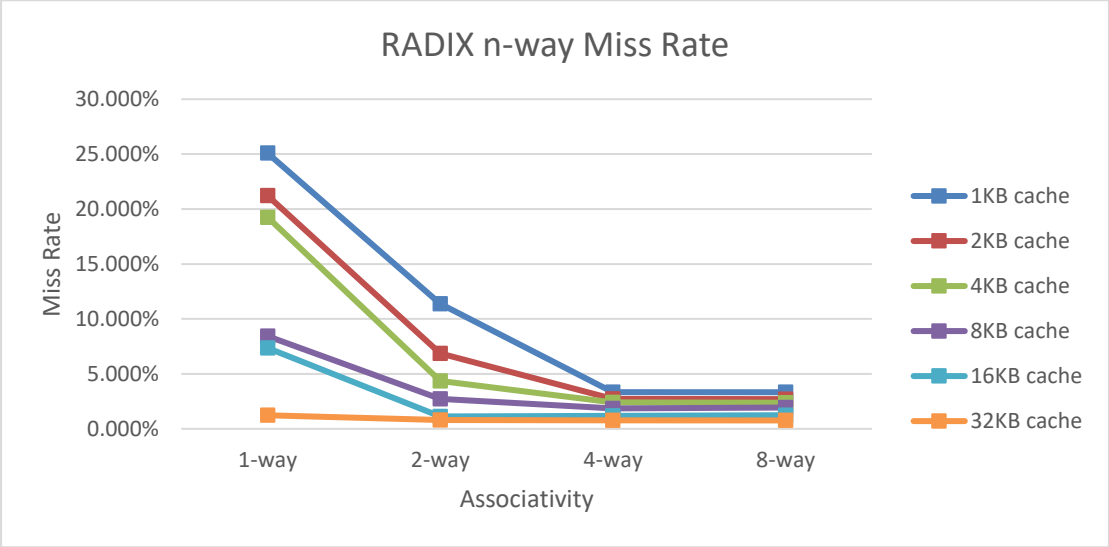
n-way / Cache Size(B)	1	2	4	8
1K	11.068%	8.355%	7.782%	7.828%
2K	8.278%	5.177%	4.185%	3.984%
4K	5.472%	3.627%	3.069%	2.806%
8K	4.030%	2.976%	2.666%	2.449%
16K	3.162%	2.372%	2.341%	2.294%
32K	2.542%	2.325%	2.279%	2.279%

在 LU 的測資裡面，可以看出如果把 1 個 set 裡面的 block 增加，剛開始會減少

miss rate，減少幅度遞減。不過在某些情況下，減少到一定程度後可能會開始上升一點點(不同大小的 Cache 會有不同狀況，不一定會上升)，這是因為固定 Cache Size 時，增加 way 數量的話會減少 index bit，因此 Cache index 容易相同(造成 conflict misses)，不過若再繼續增加 way 數量的話 miss rate 也有可能再繼續降低，因為每一個 index 可以存放的 block 數量又會變多。

- 增加 cache size 後可以使 miss rate 趨近於 2.279%。

2. RADIX n-way

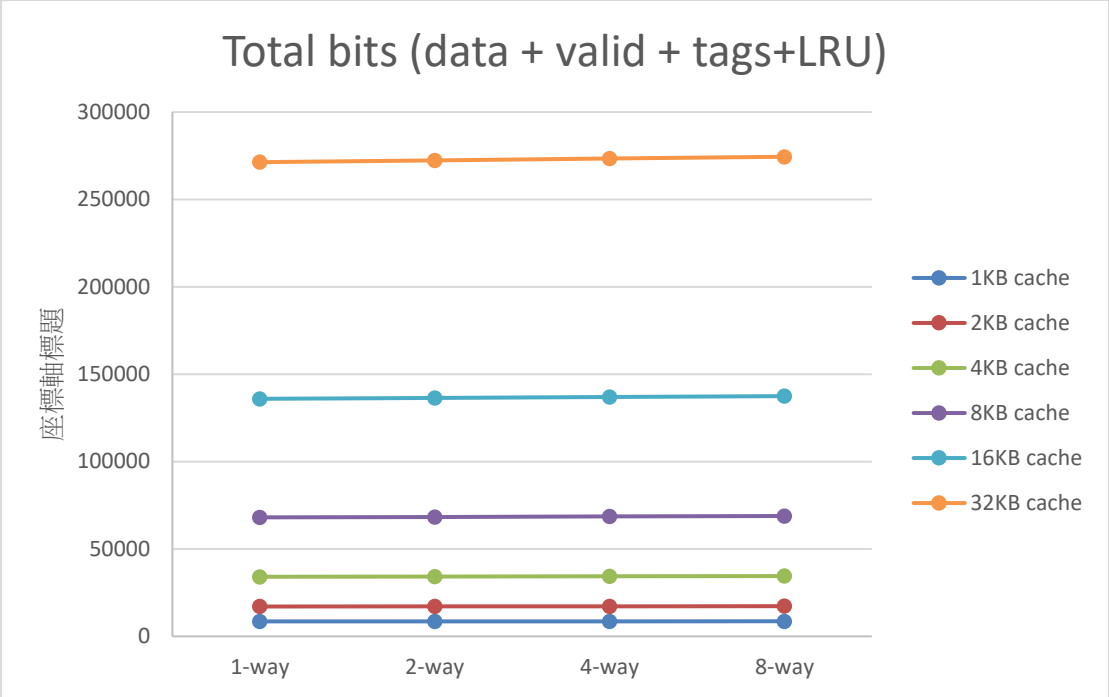


RADIX n-way Miss Rate

n-way Cache Size(B)	1	2	4	8
1K	25.086%	11.374%	3.345%	3.328%
2K	21.217%	6.872%	2.733%	2.702%
4K	19.257%	4.366%	2.396%	2.375%
8K	8.432%	2.712%	1.856%	1.925%
16K	7.356%	1.125%	1.168%	1.237%
32K	1.231%	0.809%	0.776%	0.762%

在 RADIX 的測資裡面，可以看出如果把 1 個 set 裡面的 block 增加，剛開始會減少 miss rate，減少幅度遞減。不過在某些情況下，減少到一定程度後可能會開始上升一點點(不同大小的 Cache 會有不同狀況，不一定會上升)，這是因為固定 Cache Size 時，增加 way 數量的話會減少 index bit，因此 Cache index 容易相同(造成 conflict misses)，不過若再繼續增加 way 數量的話 miss rate 也有可能再繼續降低，因為每一個 index 可以存放的 block 數量又會變多。

➢ 增加 cache size 後可以使 miss rate 趨近於 0.762%。



Total bits (data + valid + tags)

Associativity / Cache Size(B)	1-way	2-way	4-way	8-way
1K	8560	8592	8624	8656
2K	17088	17152	17216	17280
4K	34112	34240	34368	34496
8K	68096	68352	68608	68864
16K	135936	136448	136960	137472
32K	271360	272348	273408	274432

Block size 固定為 64 bytes，因此一個 block 內有 512 bits 的 data，以 1-way associativity Cache size =1kB 為例，Cache 裡可以存放 16 個 blocks($\frac{1024}{64} = 16$)，tags 的算法為$address\ bits - index\ bits - offset\ bits = 32 - \log_2 16 - \log_2 64 = 22$

而 1 way 在做 replacement 時，直接替換不需要 LRU bit，所以 LRU bit = 0。

因此 total bits = $(\text{tags} + \text{valid} + \text{data} + \text{LRU}) \times \text{number of blocks} = 8560$ bits。

而增加 way 數目的話會增加 LRU bits，與 tags bits，並減少 index bits。

其餘的 Associativity 與 Cache Size 也是依照這樣計算出來。

LRU bits per block:

1 way: 0 bit

2 way: 1 bit
4 way: 2 bits
8 way: 3 bits

Lesson learned

這次的 Lab 讓我們對於如何用 C++來計算 miss rate 有更深入的認識，沒想到學計組除了寫 verilog 還可以順便複習一下 C++，真是一舉數得，當初跑 C++ code 時，原本想要像 PDF 檔中的圖那樣子的 Cache size 以及 Block size，但是出現的圖形實在是變化不多，因此我們將 Cache size 以及 Block size 都調小許多，終於看出一些變化，在進階的部分有更認識到 Cache 是如何用 LRU 做替換，最後還有用上課所學計算出整個 cache 所需要的 total bits。