# 專題一報告

課程主題：計算機結構
學號：A1095551
姓名：廖怡誠

本次專題的內容為設計 Cache Controller，模擬在不同 Cache Size、Block Size 與 n 的情況下，Cache 內部會如何操作，以處理 Processor 所給予的記憶位置。

## 1. 架構設計

在 Cache 的設計中，我在 Class cache_controller{}建立時會根據 Cache Size、Block Size 與 n 建立一個二維陣列 Cache，column 為 set 的數量、row 為大小為 n 的一維陣列作為 cache block，每個 cache block 皆會有 valid、tag、data、time 四個變數，其中 time 為記錄 LRU 所設計的計數器。

```
// create cache
cache.resize(num_set, vector<cache_block>(set_degree));
```

圖一、cache 建構

在收到處理器提供的記憶體位置，並計算對應的 block number 後，使用 for-loop 檢查對應 block number 的陣列中是否有對應的 tag 能夠 Hit，反之則為 Miss。

```
bool check_valid(int loc_set_addr, int tag){

    for (int i = 0; i < set_degree; i++){
        if(cache[loc_set_addr][i].valid && cache[loc_set_addr][i].tag == tag){
            dashboard_content += "Hit\n";

            return true;
        }
    }
    dashboard_content += "Miss\n";

    return false;
}
```

圖二、確認 set 中所有 valid 的函式

## 2. 遭遇問題

### 2.1. Cache block 設計

原先 Cache block 的資料結構想要使用 priority queue 來實作，因為 priority queue 能夠以 Time 調整優先度，但如果使用 priority queue 進行節點尋訪以更新 Time 次數與檢查 valid，與使用同為不連續記憶體的 vector 相比較為不便，因此最後決定以 vector 為 Cache block 的資料結構，並加上能夠判斷 LRU 的機制作為輔助。

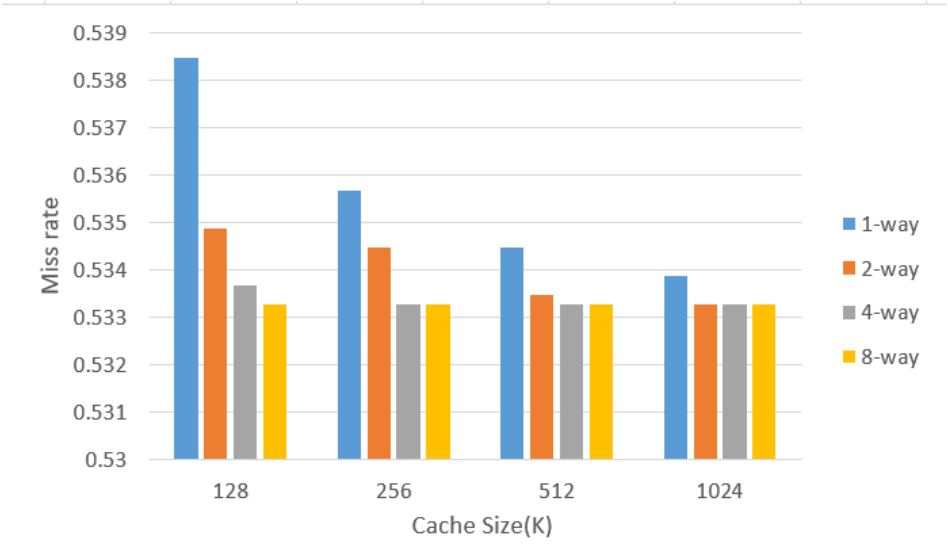
## 2.2. 測資測試問題

本次只需要輸出 miss rate，其他的 LRU 替換與 tag 數值都不需確認，而我注意到假設我寫錯計算 tag 或 LRU 替換 block 的方法，對於 miss rate 的數值是不會有所改變。首先，針對 tag 計算錯誤問題，由於我是先計算要填入哪個 set，因此即使 tag 錯誤，填入的位置也仍會相同，差別只在於紀錄的 tag 與正解不同，並不會影響 miss rate。而 LRU 替換問題，我最初的錯誤作法是將目前 cache block 第一個 block 取代，此方式與 LRU 有些微的區別，因為如果有 hit 的話，第一個位置的 data 不一定會被取代，而碰巧測試資料的 address 都為連續出現，這會使無論取代哪個 data，Hit 的次數都不會被影響。

## 3. 實驗結果分析

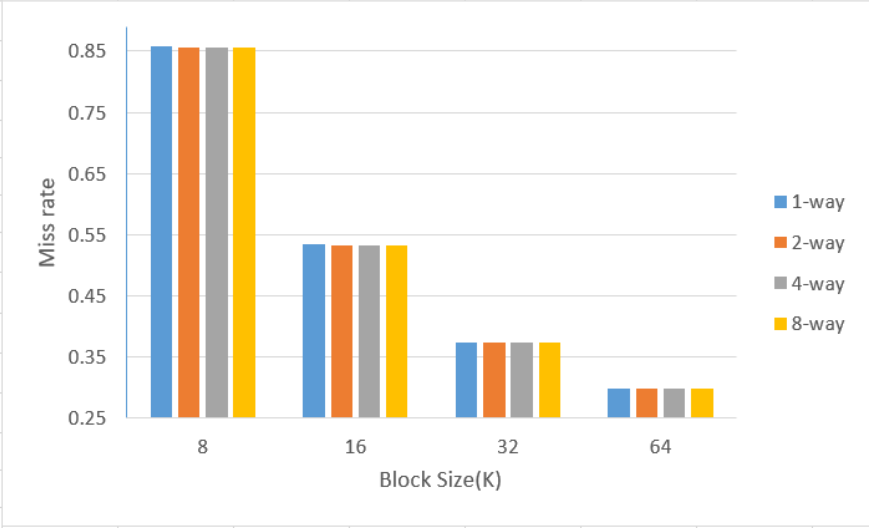
**Case 1: 固定 Block Size 為 16B，不同 Cache size 對 Miss Rate 的影響。**

Cache Size 越大 Miss Rate 越小，way 越大 Miss Rate 越小，但 n 越大，Cache Size 對 Miss Rate 的影響也越小。



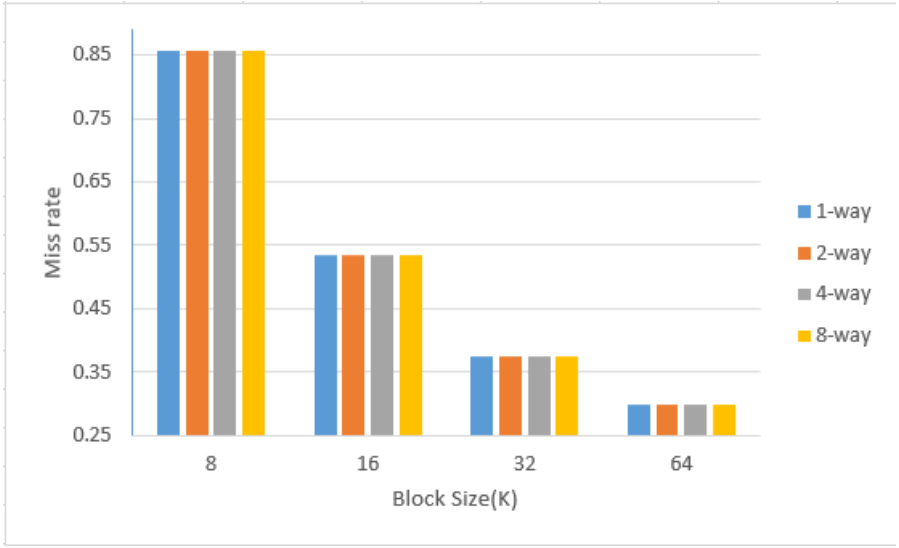

**Case 2: 固定 Cache Size 為 512K，不同 Block size 對 Miss Rate 的影響。**

Block Size 越大 Miss Rate 越小，然後 way 越大對於 Miss Rate 的影響不大。



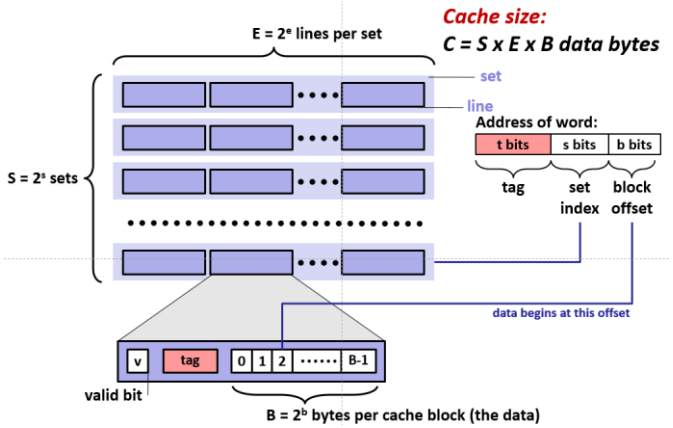

**Case 3: 固定 Cache Size 為 1024K，不同 Block size 對 Miss Rate 的影響。**

與 Cache Size 為 512K 的結果差異不大，Block Size 越大 Miss Rate 越小，然後 way 越大對於 Miss Rate 的影響不大。




## 4. 心得

在這次的實作中，我充分的了解 cache 在存取 data 是如何運作，整體實作的時間並不會花費太長，主要的時間都在理解 cache 內部的架構該如何設計，像是上課投影片的圖片中，一個 set 只有一個 valid，但如果只有一個 valid 的話，就沒辦法判斷 N-way 的情況，因此我花費蠻多時間確認實際的架構為何，最後決定以圖三中架構實作。




圖三、cache 架構

而這次專題，我嘗試使用 C++來實作，因為我認為此類的底層架構使用 Python 實作不太合適，C 語言在 Linux kernel 中被大量運用，且編譯、執行的速度也較 Python 優異，雖然在實作上相較於 Python 複雜，但也使我對於這類的實作更加的熟悉。