2.1. Modules Explanation

(只寫和 lab1 不同的部分)

CPU.v:

增加了和 off-chip memory 的接口，由 dcache_controller 控制後由 memory 傳入 memory data 和 memory ack(表示 memory 資料已準備好，CPU 可讀取)，傳入 memory 的則有 CPU 計算好的 data, address, memory enable(表示需 access memory)和 memory write(表示需寫 memory)

另外增加了 mem_stall 來記錄是否因 access memory 而需 stall，並傳入所有 pipeline stage

dcache_controller.v:

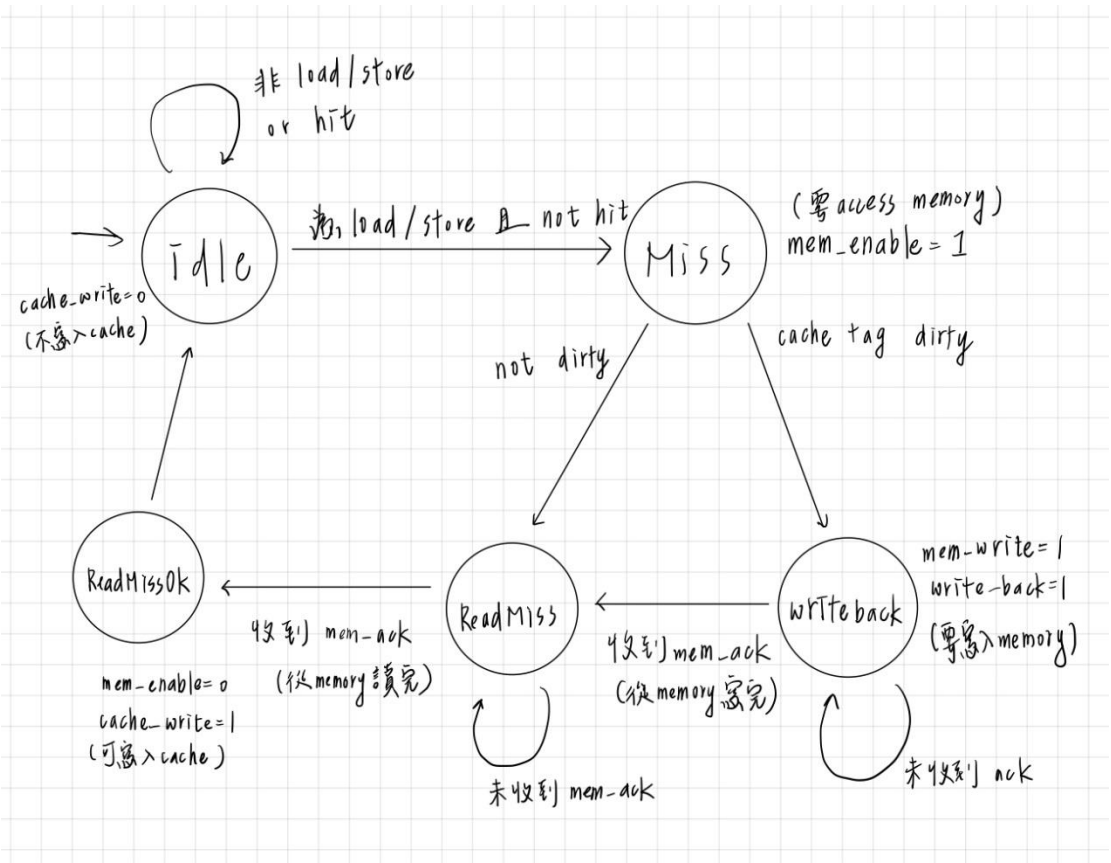
控制 CPU 和 Data memory 的資料傳遞，還有連接 L1 cache

r_hit_data 永遠設成 sram_cache_data(從 cache 拿的資料)，在 hit 前 CPU 不會將資料往下傳，hit 後保證需要的資料一定在 cache 中

cpu_data 則是在 r_hit_data 中找相應 offset 位置的 data，因為所需的資料為 32bit = 4 byte，則將 offset/4*32 後便是在 r_hit_data 中所需資料的開頭，自開頭取 32bit 放入 cpu_data 中

w_hit_data 則是找到 offset 在 w_hit_data 中的哪個位置，填入從 cpu 來的 data

state 轉換:



dcache_sram.v:

hit_o:若 cpu_tag 和該 index 的 cache[0]或 cache[1]相等，則 hit

tag_o:若和 cache[0]tag 相等則為 cache[0]，否則為 cache[1]，都不 hit 則回傳 cache[0](old)

data_o: 若和 cache[0]tag 相等則為 cache[0]，否則為 cache[1]，都不 hit 則回傳 cache[0](old)

*LRC 實作:cache[0]存放舊資料，cache[1]存放新資料，當 write_i 時將 cache[0]設成 cache[1]，cache[1]設成新的資料，若 hit 時且和 cache[0] hit，則將 cache[0]和 cache[1]交換。

Testbench.v:

最後 flush cache 時判斷是否 dirty，若 dirty 的話要將 cache 裡的東西寫回 memory

2.2. Difficulties Encountered and Solutions in This Lab

一開始花很多時間看懂 dcache_controller 助教寫的部分再做甚麼，懂了以後就比較好寫了，還有如何處裡 2 way LRU 的問題一開始造成輸出有很多問題。因為這次作業輸出格式不需要和助教的答案完全相同，debug 的時候也花了滿多時間，會一直擔心自己的寫法是否會有誤

2.3. Development Environment

Window10