Computer Architecture

Project2 Report

Team 米蟲隊

1. Members & Teamwork

B07902005 劉冠鴻

a. 修改 4 個 Pipeline_Register module

b. 修改 CPU.v

c. 一同思考程式邏輯並完成初步的程式碼(dcache_controller.v, dcache_sram.v)

d. 與 b07902127 羅義鈞一同 debug

B07902117 陳漱宇

a. 一同思考程式邏輯並完成初步的程式碼(dcache_controller.v, dcache_sram.v)

b. 撰寫 report

B07902127 羅義鈞

a. 一同思考程式邏輯並完成初步的程式碼(dcache_controller.v, dcache_sram.v)

b. 與 b07902005 劉冠鴻一同 debug

2. Implementation of each module

a. Pipeline_Register_IFID.v

因為在 Project2 需多考慮 memory stall，因此會造成 stall 的可能除了原來的“stall_i”以外還有“MemStall_i”，因此多判斷“MemStall_i”即可。

b. Pipeline_Register_IDEX.v

原本在 Project1 時是直接將 input wire 接收到的值指定給 register，再送給 output wire，但因在 Project2 中需考慮 memory stall，因此多判斷如果 “MemStall_i” 為 1，則給 register 上一次的值，否則就照常給 input 的值。

c. Pipeline_Register_EXMEM.v

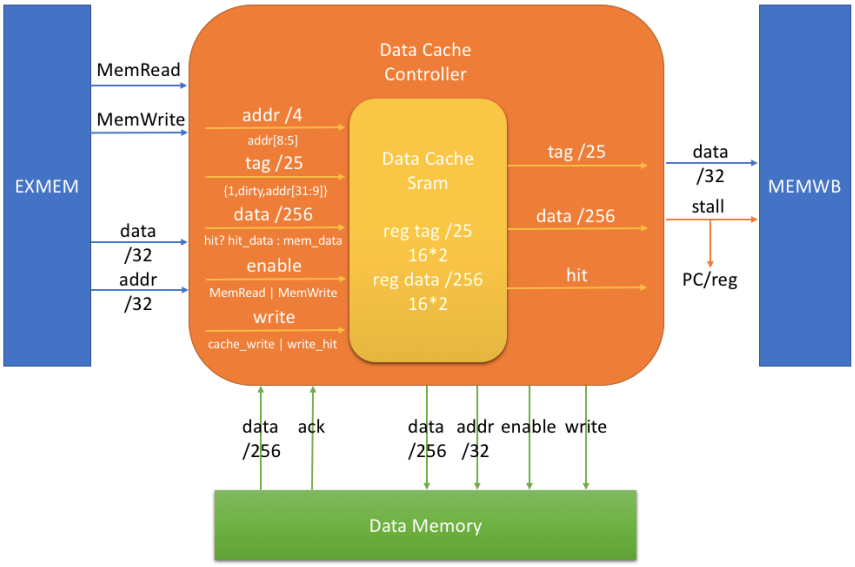
原本在 Project1 時是直接將 input wire 接收到的值指定給 register，再送給 output wire，但因在 Project2 中需考慮 memory stall，因此多判斷如果 “MemStall_i” 為 1，則給 register 上一次的值，否則就照常給 input 的值。

d. Pipeline_Register_MEMWB.v

原本在 Project1 時是直接將 input wire 接收到的值指定給 register，再送給 output wire，但因在 Project2 中需考慮 memory stall，因此多判斷如果 “MemStall_i” 為 1，則給 register 上一次的值，否則就照常給 input 的值。

e. CPU.v

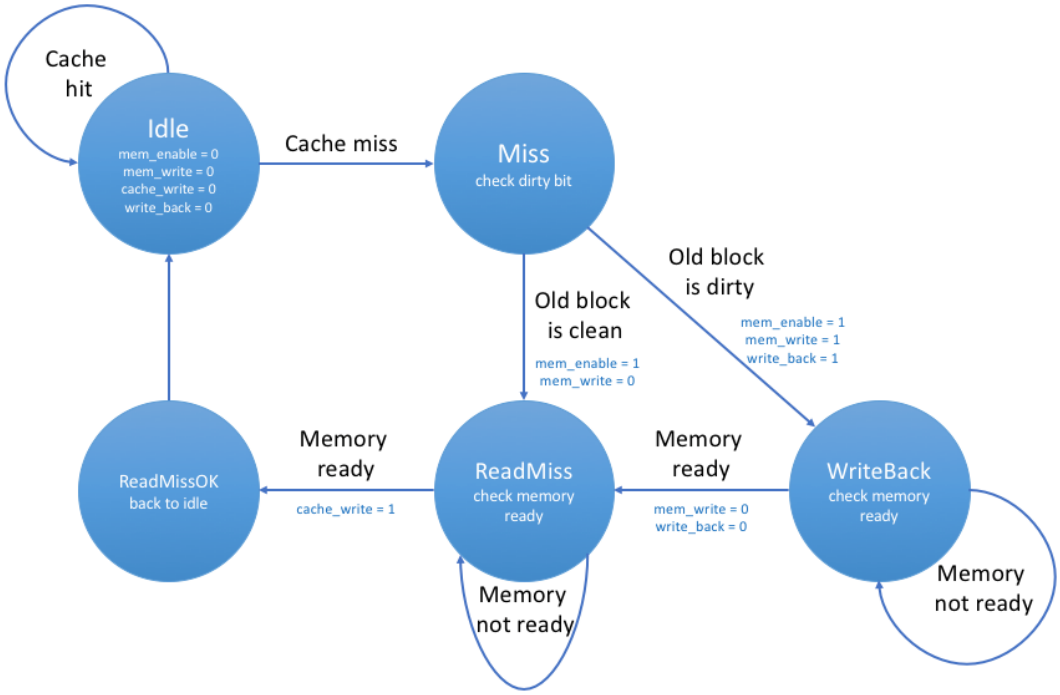
因為 Project2 多加了 Data Cache，因此要將 Data Cache 的 input 及 output 連接至其他 module 上，根據 spec 提供的 datapath 以及下面針對此 project 我們所畫的圖將線接好即可。



f. dcache_controller.v

首先，“r_hit_data” 要拿到從 cache 搬上來的資料，因此將其設為 “sram_cache_data”; 而因為 “r_hit_data” 一次會拿 256 bits，但當 CPU 要 read data 一次只需 32 bits，因此要將 CPU 想要的那 32 bits 從 “r_hit_data” 擷取出來並給 “cpu_data”; 同樣當 CPU 要 write data 時，一次要寫 32 bits，但因為 cache 一次是以一整個 block (256 bits) 做寫入，因此需讓 “w_hit_data” 只有在要被改動的地方被改動，其餘地方維持跟 cache 相同。

而至於 state 的部分，根據以下我們針對此 project 所畫的 FSM，在每個 state 對 “state”, “mem_enable”, “mem_write”, “cache_write”, “write_back” 做適當的改動並使其進到下個 state。



g. dcache_sram.v

對於 write data，用 tag 和 valid bit 來檢查是否在 2-way associative cache 中(為方便說明我們將其定為左右兩邊)，我們用“LRU_bit”表示對於 cache 同個 index entry 的兩個 block 中最久沒被用到的 block，對於一個 index，若在左邊 hit，則 set on dirty bit、將 data 從 cache 中取出，並將“LRU_bit”設為 1(右邊)；在右邊 hit 的情形與左邊相同，只是將“LRU_bit”設為 0(右邊)；若都沒有 hit，則此時代表要把最久沒用的 block 取代掉，也就是把 cache 中“LRU_bit”相對應位置的 tag 和 data 都用 input 進來的 tag 和 data 取代。而我們新增了兩條 wire: ”output_tag”和”output_data”，並判斷哪邊 hit 就設為哪邊的 tag/data 值，若都沒 hit 但 cache 中 tag 的 valid bit 和 dirty bit 是 on 的則設為 cache 的 tag 值，否則設為上一次的 output_tag/output_data。最後 output 的部分，若是要 read data 則將“tag_o”設為“tag_i”，否則設為

“output_tag”; “data_o” 設為 “output_data”; 當左邊或右邊 hit 時將 “hit_o” 設為 1，否則設為 0。

3. Difficulties encountered and Solutions in this project

a. 發現 write_i = 0 時，左右都不會 hit，沒辦法進到正確的 if 判斷。當 write back 發生時理論上要進到 always block 中判斷要 output 出去的 tag 值，但我們原本寫的會直接將 input 進來的 (“tag_i”) 送回去。

解決辦法: 將原本寫在 always block 的部分用 wire 在 always block 外進行 assign，並且做 tag_o 的修正。

4. Development Environment

MacOS 下使用 iverilog -o CPU.out *.v 去編譯，再用 ./CPU.out 執行。