**Computer Architecture Project 2 Report**

**Team: 單生狗聯盟(WannaLaugh)**

**2.1 Module Explanation**

|  |
| --- |
| **dcache\_controller.v**  In **dcache\_controller.v**, there are  8 inputs:  **clk\_i**, **rst\_i**, **mem\_ack\_i**, **cpu\_MemRead\_i** and **cpu\_MemWrite\_i** which all 1 bit.  **cpu\_data\_i** and **cpu\_addr\_i** which both 32 bits.  **mem\_data\_i** which is 256 bits.  6 outputs:  **mem\_enable\_o**, **mem\_write\_o** and **cpu\_stall\_o** which all 1 bit.  **mem\_addr\_o** and **cpu\_data\_o** which both 32 bits.  **mem\_data\_o** which is 256 bits.  There are several codes that needed to add to **dcache\_controller.v**:   1. **r\_hit\_data**     assign **r\_hit\_data** = (hit) ? **sram\_cache\_data** : **mem\_data\_i**;  where  **sram\_cache\_data** is the data read from the **dcache\_sram.v**  **mem\_data\_i** is the data read from the **Data\_Memory.**  If **hit** == 1, it means that the data is in the cache, the 256 bits data should be chosen from the cache. Otherwise, data should be chosen from the data memory.  2. **cpu\_data**    **cpu\_data** <= **r\_hit\_data**[(cpu\_offset\*'d8) +: 'd32];  where **cpu\_data** is the read data that requested.  Since the block size is 32 bytes (256 bits) and the data requested is in size of 32 bits: .    Thus, the block offset should multiply by 8 to get the base offset of the data and extend by 32 bits to get the requested data.  3. **w\_hit\_data**    **w\_hit\_data** <= **r\_hit\_data**;  **w\_hit\_data**[(cpu\_offset\*'d8) +: 'd32] <= **cpu\_data\_i**;  where **w\_hit\_data** is the write data that should write to the cache or the data memory.  Since by TA code, the write process will be  write allocate: write miss = read miss + write hit  Thus, the 256 bits data that needed to write will first held in **r\_hit\_data**. Passed 256 bits **r\_hit\_data** to **w\_hit\_data** and write the modified 32 bits data to **w\_hit\_data.**    The method of base offset calculation is the same as in **section** “**2. cpu\_data”**    **w\_hit\_data** is now ready to be write to cache or data memory.  4. **controller** (actions and states)  Following is the FSM for the controller:    When changing from a state to another, the four variables should also change to the value in the next state as shown in the image above.  Note that after entering **STATE\_READMISSOK**, the next state always be **STATE\_IDLE.**  There are four variables used to determine the actions of **dcache\_sram.v** and **Data\_Memory.v**.    **mem\_enable**  used to determine whether the memory access on data memory is allowed.  **mem\_write**  used to determine whether the write action is allow on data memory when there is data needed to write to the data memory.  **cache\_write**  used to determine whether the write action is allow on cache when there is data needed to write to the cache.  **write\_back**  used to determine whether there is a write back action due to the replacement of a dirty cache in the cache.  There are five states with different value for the four variables stated above that complete the function of controller.  **STATE\_IDLE**    **mem\_enable** <= 1'b0;  **mem\_write** <= 1'b0;  **cache\_write** <= 1'b0;  **write\_back** <= 1'b0;  Wait for request, no actions needed in cache and data memory.  If **cpu\_reg && !hit == 1**, means that there is a request (either read or write) and there is a miss. Thus, the controller state will change from **STATE\_IDLE** to **STATE\_MISS.**  Else the controller state remains in **STATE\_IDLE**.  **STATE\_MISS**    **mem\_enable** <= 1'b0;  **mem\_write** <= 1'b0;  **cache\_write** <= 1'b0;  **write\_back** <= 1'b0;  No action needed in cache and data memory.  If **sram\_dirty == 1**, means that there is dirty cache been replaced by the LRU policy and write back action is needed. Thus, the controller state will change from **STATE\_MISS** to **STATE\_WRITEBACK**.  Else the controller state will change from **STATE\_MISS** to **STATE\_READMISS.**    **STATE\_READMISS**    **mem\_enable** <= 1'b1;  **mem\_write** <= 1'b0;  **cache\_write** <= 1'b0;  **write\_back** <= 1'b0;  When in **STATE\_READMISS**, **mem\_enable = 1’b1** in order to allow memory access in data memory to read data from data memory.  If **mem\_ack\_i == 1**, means that the data is already read from the data memory. Thus, the controller state will change from **STATE\_READMISS** to **STATE\_READMISSOK**.  Else, the controller state remains in **STATE\_READMISS** until the data is read from data memory and an ACK is receive from data memory.    **STATE\_READMISSOK**  **mem\_enable** <= 1'b0;  **mem\_write** <= 1'b0;  **cache\_write** <= 1'b1;  **write\_back** <= 1'b0;  When in **STATE\_READMISSOK**, **cache\_write = 1’b1** so that the data can be written into the cache  The controller state is then change from **STATE\_READMISSOK** to **STATE\_IDLE** and wait for next request.    **STATE\_WRITEBACK**    **mem\_enable** <= 1'b1;  **mem\_write** <= 1'b1;  **cache\_write** <= 1'b0;  **write\_back** <= 1'b1;  When in **STATE\_WRITEBACK**,  **mem\_enable = 1’b1** so that memory access is allowed in data memory  **mem\_write = 1’b1** so that that data in the dirty cache is write to the data memory  **write\_back = 1’b1** to indicate write back action occur.  If **mem\_ack\_i == 1**, means that the data is already write to the data memory. Thus, the controller state change from **STATE\_WRITEBACK** to **STATE\_READMISS**.  Else the controller state remains in **STATE\_WRITEBACK** until the data is write to data memory and an ACK is receive from data memory. |
| **dcache\_sram.v**  **變數解釋**  這個module有7個input和3個output，input分別是clk\_i,rst\_i,addr\_i,tag\_i,data\_i,enable\_i和write\_i,output分別是tag\_o,data\_o和hit\_o。這份module會使用tag和data這兩個register去儲存cache裡的tag和data，然後使用LRU去紀錄每一個set當前的LRU來實作2-way associative cache。LRU是個16bit的register，每一個bit的value代表著每一個set的LRU是第0筆資料還是第1筆資料。  **初始化**  當rst\_i為1的時候，tag,data跟LRU會初始化為0。  **write data**  當enable\_i和write\_i都是1的時候，根據addr\_i查詢到的tag[addr\_i]如果跟tag\_i相同，就會把data\_i寫進去data[addr\_i]，寫入data的同時，也會把相對應的tag替換，並且set up dirty bit，否則就會寫入在data[addr\_i]的LRU的那筆data。  **read data**    根據上圖，我們可以透過addr\_i找到tag[addr\_i]的23-bit tag和1-bit valid bit來決定是否hit，如果hit變把hit\_o 設成1，tag\_o設成hit的tag，data\_o設成hit的那筆data，否則，我們就把hit\_o設成0，tag\_o設成LRU的tag，data\_o設成LRU的data。此外，每次hit了之後我們都會更新LRU。 |
| **CPU.v**  In project 2, CPU.v have  Five inputs:  clk\_i, rst\_i, start\_i and mem\_ack\_i which is 1 bit.  mem\_data\_i which is 256 bits.  Four outputs:  mem\_data\_o which is 256 bits.  mem\_addr\_o which is 32 bits.  mem\_enable\_o and mem\_write\_o which is 1 bit.  A new module, **dcache\_controller dcache()** is added and connected with wires to interact with **Data Memory** and **dcache\_sram**. |
| **Register\_IFID.v & Register\_IDEX.v & Register\_EXMEM.v & Register\_MEMWB.v**  In project 2, there is a small modification for all four pipeline registers.  Every pipeline register has a new input called **stall\_i.**  If **stall\_i** == 1, then the register should undergo stall operation and all the outputs maintain the same value as outputs in previous cycle. |

**2.2. Members & Teamwork**

|  |  |
| --- | --- |
| **Member** | **Works** |
| B07902091 周俊廷 | 主要負責dcache\_controller.v。  更新pipeline registers（共4個）  CPU.v最終修正  撰寫上述module報告 |
| B07902089李智源 | 負責dcach\_sram.v  撰寫dcach\_sram報告 |
| B07902059陳君翰 | Debug、資料測試、報告彙整 |
| B06902098 李恩慈 | CPU.v初步撰寫（由project 1 CPU.v改寫） |

**2.3. Difficulties & Solutions**

|  |  |
| --- | --- |
| B07902091  周俊廷 | 1. read data 和 write data 的查找方式：   在block offset 的使用方式上產生了困難   1. 錯誤： 2. 正確： 3. state的改變：   對mem\_enable, mem\_write, cache\_write和 write\_back的值理解錯誤   1. 錯誤：根據目前的state去更改這四個變數 2. 正確：把四個變數更改成next state時的值   在本報告中的dcache\_controller.v的部分有作更詳盡的説明，並附上Finite State Machine圖做為參考 |
| B07902089  李智源 | 1. 難以直觀地理解助教程式碼，因此不了解controller如何運作，導致不了解sram具體要處理什麼 2. 判斷hit時tag的比對： 3. 錯誤： 25bit全部比對 4. 正確：只需使用tag[22:0]，共23bit即可 5. LRU的實作問題：   應在每次CPU read時即更新 |
| B07902059  陳君翰 | 雖自認對於實作方法有足夠理解，但難以直觀地理解助教程式碼，因此有很長一段時間不知從何處下手 |
| B06902098  李恩慈 | none |

**2.4. Development Environment**

The OS used : CSIE Workstation

The Compiler used : iverilog