Computer Organization

0516013 吳泓寬

1. Source code and the note

```
Pipe_Reg #(.size(71)) MEM_WB(
    .clk_i(clk_i),
    .rst_i(rst_i),
    .data_i({RegWrite_exmem,Write_addr_exmem,MemReg_exmem,result_exmem,mem_data}),
    .data_o({RegWrite_memwb,Write_addr_memwb,MemReg_memwb,result_memwb,mem_data_memwb})
);
```

基本上我覺得這次跟上次lab比起來最大的差異大概就是Pipe_Reg。size的部分就是把data_i有多少bits算出來，而data_i和data_o的寫法算是一個比較tricky的部分，利用對應的bit數去做一個簡單的assign，讓code的可讀性增加不少。

bonus (已附上bonus.txt) - reorder instructions

基本上reorder的概念就是把會產生hazard的code分開，讓後面的instruction能取到正確的值。又因為我們沒有做forwarding所以中間基本上都會有bubble，所以我就之中穿插後面的code(又跟前面沒有dependency)，reorder的結果如下：

00100000000000010000000000010000 # I1

00100000000010010000000001100100 # I10

00100000000000110000000000001000 # I3

10101100000000010000000000000100 # I4

10001100000001000000000000000100 # I5

00100000001000100000000000000100 # I2

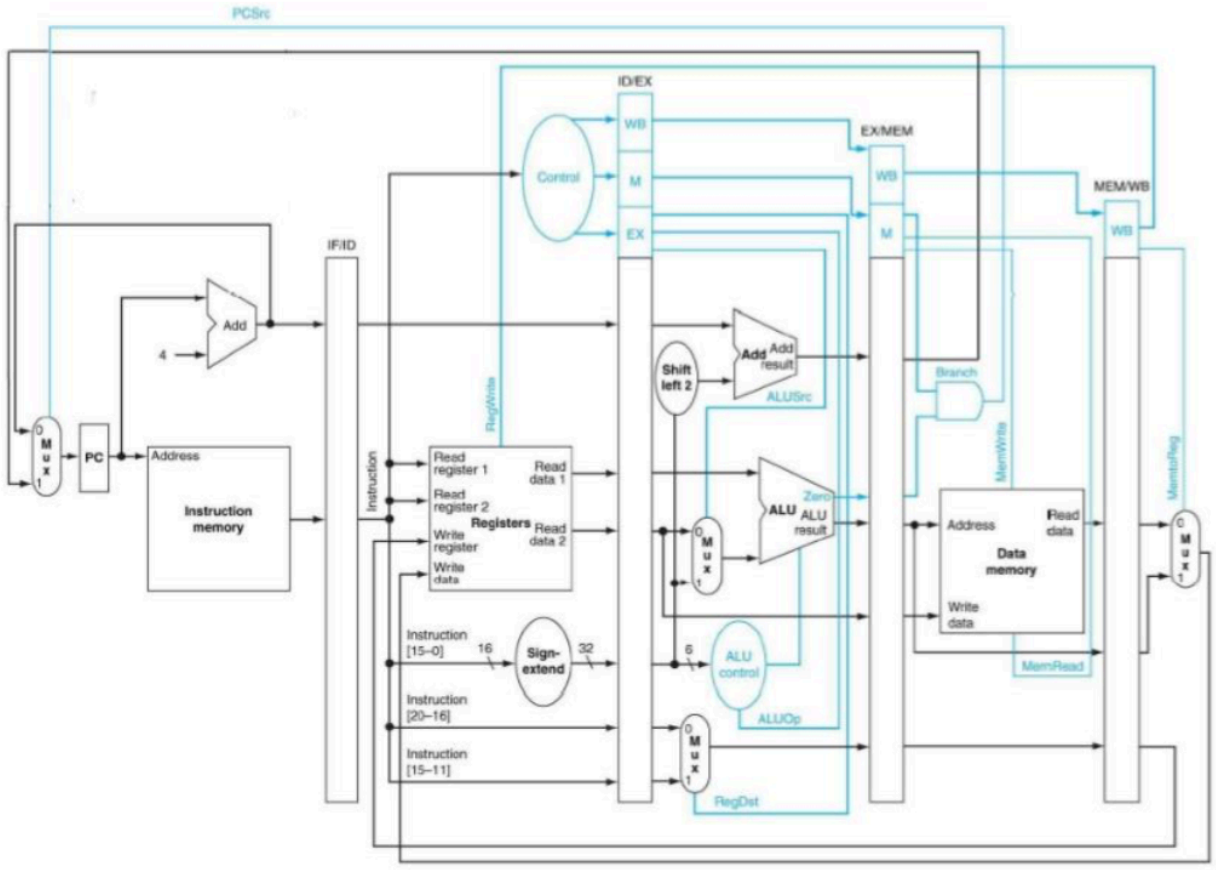
00100000001001110000000000001010 # I8

00000000011000010011000000100000 # I7

00000000100000110010100000100010 # I6

00000000111000110100000000100100 # I9

2. Your architecture



(取自於CO_Lab4.pdf)

3. Hardware module analysis

基本上pipeline的概念就是分解動作讓一個cycle可以變更短，根據上圖的做法便是將原先lab3實作的部分分成5個步驟，而每個步驟間為了保留state(接下來會用到的control)所以會有pipeline register，基本上要用的module都先提供了，所以基本上就是根據上圖的流程完成pipeline_cpu那個檔案。

就如同上課所說的要注意會不會delay的問題，像RegDst為求保險就會放在stage3。

4. Problems you met and solutions

這次一開始有遇到會異常寫入值進入register的情形，之後發現原來是RegWrite有傳到後面但忘記接回Register，所以變成是用現在正在跑的stage2的RegWrite。

5. Summary

這次算是做最基本的pipeline CPU 也還沒做forwarding和一些防hazard的機制，但我覺得分成很多stage後debug變的十分困難，原本都是用$display來看目前的值，但現在還要一直往前看是前面哪個步驟寫壞了，不知道有什麼好用的debuger...

不過隨著學期漸漸到了尾聲，CPU也漸漸成形了，蠻有成就感的。