Lab1 report

By 許庭瑋

1. module 介紹

CPU module是整合所有其他module的。哪個module的input是哪個module的output，哪個module的output是哪個module的input，都在這裡進行連接。

PC module的主要用途是把clock signals, reset bit, start bit讀進來，然後根據這三個數值的變化進行pc_o的更新。當clock signal是positive時，才會更新pc_o。當reset bitis on時，就重置pc_o。當start bitis on 時，PC就會被更新。這次多的PCwrite，是用來回傳hazard結果的。

Register module是用於存放32個32bits registers的地方。讀取clk_i，當clk_i 是positive時會才進行write的動作。讀取RS1addr_i，RS2addr_i，然後根據讀取的address或RegWrite_i把相對應暫存器的值或讀進來的write data讀出來，然後用RS1data_o，RS2data_o輸出。讀取RDaddr_i，RDdata_i，RegWrite_i，根據讀取的address和要寫入的data，然後根據RegWrite_i來判斷要不要進行write。

Instruction_Memory module是用來存放Instruction的。首先

用reg設置好。然後就根據addr_i讀進來的值去判斷要輸出哪個instruction，然後放到instr_o去輸出。

Adder module是用來做加法的。讀取data1_in和data2_i，n然後把這2個值加起來，放到data_o輸出。在這個simple CPU裡，它的作用是做pc=pc+4這個運算。

Control module是根據opcode來判斷是control signal。把instruction 的前7個bits讀進來(opcode)，然後根據R type還是I type還是 lw 還是 sw 還是 beq 來確認輸出。這次多了很多個signal，有7個signal要輸出。NoOp是有hazard的情況，為了製造stall，所以NoOp=1時signal就全部變0。

IFID這個module就是IF/ID pipeline。把pc位址和insruction讀進來，下個clock再丟出去。有2個東西要check，一個是flush，是有branch的情況才會有的值，會把pc位址和insruction設成0避免非預期動作。一個是stall signal，是有hazard的狀況才會有的值，會製造stall(什麼都不做，值不變，control signal全設成0)。

IDEX這個module就是ID/EX pipeline，把signals、register讀出來的值，還有一堆之後的stage會用到的東西全部存進來，下個clock再丟出去。要存的東西就是slide上標註的那些。沒有需要注

意，會造成傳輸要求改變的東西傳進來。

EXMEM這個module就是EX/MEM pipeline，把ALU的結果、剩下的signals、要wrtie back的RD存進來，下個clock再丟出去。沒有需要注意，會造成傳輸要求改變的東西傳進來。

MEMWB這個module就是MEM/WB pipeline，把ALU的結果、memory的data、剩下的signals、要wrtie back的RD存進來，下個clock再丟出去。沒有需要注意，會造成傳輸要求改變的東西傳進來。

Muxforreg1forwardingcheck這個module是在EX stage中，有2個因為forwarding的關係，需要透過forwarding的output來確定要用IDEX_data1、EXMEM_ALUout、WBresult哪個值。這是其中一個，另一個是Muxforreg2forwardingcheck。

Forward_unit這個module就是做forwarding check的。就像老師上課所說的判斷標準，把需要比對的值拉進來，再根據比對萊輸出2個結果，這個2個結果會用來判斷ALU要用哪裡來的值。

MUX32這個module是用來做判斷的，根據ID/EX傳過來的ALUSrc_o是不是0來判斷說我要選Muxforreg2forwardingcheck讀出來的值還是sign extend的值。

Hazard Detection就是做Hazard Detection的。就像老師上課教的，把需要比對的值拉進來，再根據比對萊輸出結果，決定是否

要stall，這會影響到pc、IF/ID、control三個module的行為。

Sign_Extend module是用來做sign extend的。一個值傳進來，要變成32bits傳給ID/EX，而且數值和正負不會變。因為正負的判斷是看最左邊那一位，0是正，1是負。然後做sign extend的話，如果是正的，前面多出來的就填0，如果是負的，前面多出來的就填1。因此可以選擇把最左邊那一位複製20次，就可以做到以上的要求了。這次多了很多東西要判斷，lw還好，sw就會在原本是rd的部分放imm，beq也會。這樣的話最直覺的做法是把整個instruction傳進去，然後看opcode來決定要extend哪些部分。beq的imm就用verilog的{}組合功能來重新排列組合。

ALU_Control module是根據instruction分出來的func7+func3和Control module傳過來的ALUOp_o來判斷ALU要做什麼運算行為。這次跟上次相比只多了一種ALUopu要判斷，問題不大。

ALU module是做運算的。讀取registermodule傳來的data1_i和MUX32選出來的data2_i這2個值和ALU_Control module傳過來的ALUCtrl_o來判斷要對這2個值做什麼運算。然後把結果放到data_o裡，傳回register去write。Beq不會用到，lw和sw在這裡都是做add運算，幾乎不用改

MuxforPC這個module是透過確認是不是branch來決定下一個pc

是+4還是branch。

AddandShift這個module是做branch的地址運算的。在slide上是一個<<1和一個add module，但這可以一起算所以就合併了。

beqcheckequal這個module就是beq中rs1==rs2 確認的部分，單純就是拿register file讀出來的值來比對。

ANDforbranch這個module是確認要不要branch，要符合2個條件:這個instruction是不是branch(control 傳來的branch signal)和值是否相等(beqcheckequal的結果)。

MuxforMEM就是整張datapath圖最右邊的那個mux，來判斷要用memory的值還是ALU的值。

2.所遇困難與解決

剛開始很正常，就按照slide的圖把每個path都連出來，這次lab跟上次有個不一樣是:這次的path會一個source連到多個地方，原本我是直接用module.port的方式去輸入或輸出，但我不知道這在連多個地方會不會成立，verilog不熟的我就只好把每個path都宣告成wire然後再連。Sign_extend有點複雜，因為有beq那個醜醜的imm存在和sw他們的imm都跑到rd的地方，所以原本只傳後面12位就不行了。為了方便思考，就直接傳整個instruction進去，再根據opcode做判斷。接著開始debug。跟上次差不多。有個特難的bug是Register

file要看MEMWB傳回來的RegWrite才能取值，但前4個instruction所得到的MEMWB_RegWrite都是x，這樣取不到值。所以我就額外宣告一個reg，再用always@(*)去讓這個reg隨MEMWB_RegWrite更新，如果MEMWB_RegWrite不是1或0就讓這個reg是0，達成讓Register file的RegWrite初始化的效果。

3.系統

我是用Windows系統和iverilog去編譯