Computer Architecture Project1 Report

組別：羅密歐與傅立葉
組員：黃漢威、陳冠瑜、廖彥綸

分工：

黃漢威：Register.v、testbench.v、Data_Memory.v、EX_MEM.v、MEM_WB.v、部分CPU.v
陳冠瑜：Control.v、Equality.v、Hazard_Detection.v、Shift_Left_1.v、Sign_Extend.v、部分CPU.v
廖彥綸：IF_ID.v、ID_EX.v、Forwarding.v、部分CPU.v

實作方式：

testbench：
CPU.v裡有個flush的bool wire，跟bubble的bool wire，分別對應到flush跟stall。

Adder：
沿用上次作業的Adder.v，將兩個輸入data1_in、data2_in的和相加，輸出至data_o。在兩個地方使用
Adder_PC_IF：將輸入的PC值加4到輸出
Adder_PC_ID：將輸入的PC及從immediate extend來的branch distance相加

ALU：
沿用上次作業的Adder.v。以不同的ALU_Ctrl執行對應的計算。
Ctrl = 0110:　　data1_i - data2_i
Ctrl = 0010　　data1_i + data2_i
Ctrl = 0000　　data1_i & data2_i
Ctrl = 0001　　data1_i | data2_i
Ctrl = 0100:　　data1_i * data2_i

ALU_Control：
沿用上次作業的ALU_Control.v。根據從control unit 來的ALUOp及Op7、Op3來決定ALU應該要做的運算類型。ALUOp=2'b00時做加法，2'b01時做減法，2'b10時則再由Op7及Op3決定，分別做加法、減法、bitwise AND、bitwise OR或乘法。

Control：
根據上次的Control再多加幾個跟memory及branch相關的控制訊號：
　　MemRead：控制data memory，是否要讀取資料。lw時輸出1，其他輸出0。
　　MemWrite：控制data memory，是否要寫資料進memory。sw時輸出1，其他輸出0。
　　MemToReg：控制mux，寫入register的資料是否來自memory。lw時輸出1，其他輸出0。
　　Branch：Operation Code是否為beq。beq時輸出1，其他輸出0。之後再與branch條件判斷的結果 AND在一起來決定是不是要branch。

Immediate_format：控制Sign_Extend，因為不同的instruction format需要取的immediate bit不一樣。ADDI及lw為I-format，sw為S-format，beq則為SB-format。

CPU：
利用 clk_i 驅動程式，並且連接並執行所有module。

DataMemory：
posedge時寫入，偵測到訊號改變時讀取。32bit的資料要拆成四塊存入四個memory address。

Equality：
判斷branch條件是否達成。因為這次只需要實作beq所以就是判斷兩個從register輸出的值是否相等。

EX_MEM：
EX跟MEM間的記憶體。控制DataMemory跟ALU的行為。

Forwarding：
當 Rs1 或 Rs2 所需要的資料發生Hazard時調控，ALU輸入前的MUX，將資料正確地向前傳送。

Hazard_Detection：
判斷是否有hazard產生，條件為EX state要從memory讀取資料，且其destination register與ID state的兩個source register的任一個相同。
3個output：
IF_ID_Write：控制IF_ID register是否寫入新的instruction。hazard發生時輸出0，則不寫；反之則寫。
PC_Write：控制PC是否要更新(+4或branch)。hazard時輸出0，則不更新；反之則更新。
Bubble_Insertion：hazard發生時要stall一個cycle，所以要insert一個bubble到後面的register。hazard發生時輸出1，則insert一個bubble到ID_EX register；反之則正常的將control signal 傳到下個state。

ID_EX：
ID跟EX間的記憶體。
當clk_i為正時驅動。將WB和M的控制碼傳輸到下一層(EX_MEM)，Rs1、Rs2更新到Forwarding中做為Forwarding的判斷基準。同時Rs1、Rs2也輸出至ALU前的MUX依照format和forwarding的判斷後製ALU進行運算。

IF_ID：
IF 層和 ID 層間的記憶體，紀錄PC number和 inst 並往下一層相對應的出口傳輸。當flush發生時，正確將PC number和 inst 歸零。

Instruction_Memory：
沿用上次作業的Instruction_Memory.v。將memory[addr_i << 2]的inst放到instr_o。

MEM_WB：
MEM跟WB之間的記憶體。控制是否要write to register。

MUX7：
用於決定是否要在ID_EX register插入bubble。select接到hazard detection，如果是0則正常的傳遞control signal，1的話則插入全0的bubble。

MUX32：
由 select_i 選擇 32bits 的輸入data1_i 或 data2_i 和者連接至 data_o。
在此作業用於PC_SEL_Branch、PC_SEL_Hazard、MUX_ALUSrc(用於addi之類的運算)和 rightest(最右側的MUX，回傳需要寫入register的data).

MUX32_3:
擁有3個32bits輸入端的MUX，在mux_ALU_RD1及mux_ALU_RD2，此兩處的input都有三個可能性需要處理。select接到forwarding，判斷是否需要forward資料及forward來源的state。

Register：
將clk_i改成*。

Shift_Left_1：
將輸入的訊號往左shift 1bit，用於產生branch distance。

Sign_Extend：
輸入instruction，根據其instruction format取出其中的immediate bits 並extend成32 bits。
I-format：output={{20{data_i[31]}}, data_i[31:20]}
S-format：output={{20{data_i[31]}}, data_i[31:25], data_i[11:7]}
SB-format：output={{20{data_i[31]}}, data_i[31], data_i[7], data_i[30:25], data_i[11:8]}

問題：

1. 對題目的架構不熟悉，線路組裝上出現不少錯誤。
   解法：熟讀課本上的電路圖，避免混淆的情況。
2. Register寫入完來不及讀出。
   解法：問助教，clk_i改成*。
3. 雖然.v檔很多，但最累的部分是CPU.v的接線，難以分工。
   解法：先接個大概，集合起來一起debug。

編譯環境：

Linux上的ncverilog