CA lab1 report

Module explanation:

MUX_PC:

決定下一個 PC 是否需要 branch，由 Add_PC 傳進 Current PC+4，由 Add_PC_branch 傳進 PC+immediate << 的值，再由 Control 裡的 branch 和 Src1 == Src2 的 and 值決定要使用哪個 data.

IF_ID.v:

輸入 PC 和從 instruction memory 讀出來的 instruction 並儲存在 register 裡，下一個 clock cycle 再繼續往下傳，由從 Hazard 傳來的 stall 跟 branch and gate 的 flush 做 control，若 stall == 1 則不改變當前 register 中的 instruction，若 flush == 1 則將目前的 instruction 變成 32’ b0 向下傳

Control:

多了一個 NoOp 來控制是否要將所有 control signal 設成 0(碰到 stall 時)，NoOp 由 Hazard 傳入

Sign_Extend:

將 12 bit 的 immediate padding 最高位(signed)成 32 bit，由各 type 的 instruction 結構決定要取哪幾個 bit 做 immediate

D_EX:

輸入所有 control signal(除了 branch)和 register data, instructions 並存入 register，等待下一個 cycle 往下傳

Forward:

由 ID_EX 傳入的 Src1, Src2 addr 和 EX_MEM, MEM_WB 傳入的 RD addr 決定是否需要 forward，當 ID_EX 中的 Src1 或 Src2 addr 和 EX_MEM 的 RD addr 相同且 EX_MEM RegisterWrite == 1 則輸出 10，若和 MEM_WB 相同則輸出 01，結果輸出至 MUXA 和 MUXB

MUXA/B:

若 00 使用由 ID_EX 傳來的 Src data，10 使用 EX_MEM 儲存的 ALU 結果，01 使用 MEM_WB 通過 MUX 要寫入 register 的結果，將結果傳進 ALU or

MUX_ALUSrc:

由 ID_EX.傳入的 ALUSrc 判斷要使用 MUXB or immediate 傳入 ALU

ALU_Control:

和上一次作業相似，但多加了新的 instruction，有一些額外的判斷式 ，除了 ALUop 也需看 ID_EX 內存的 function7 & function3 來判斷他是哪類型的運算

ALU:

由 ALU_Crtl 判斷需要做哪種 arithmetic operation

EX_MEM:

儲存 control signal 和 ALU result, Src2 data 和 RD address，ALU result 和 Src 傳入 Data memory，ALU result 和 RD address 傳入 MEM_WB

MEM_WB:

儲存 Data from ALU, Data from memory 和 RD address，還有剩下的 control signal MemtoReg 和 RegWrite
將 Data from ALU, Data from memory 和 MemtoReg 傳入 MUX，RD address 和 RegWrite 傳回 register

MemToReg:

決定要使用 ALU result 還是 Memory data 傳回 register

Hazard:

由 IF_ID 得到的 Src1 和 Src2 和 ID_EX 的 RD 比對，若相同且 MemRead == 1(表示為 load instruction)，則發生 Hazard，設定 stall = 1，PC_Write = 0,NoOp = 1，傳入 IF_ID,PC 和 Control，表示下個 cycle 的 instruction 和 PC 都不會變，且這個 cycle 所有的 control signal 都為 0，不會做影響 register 和 data memory 的 operation

Difficult Encounter

覺得碰到最大的問題應該是 verilog 的 compiler 不會像平常寫 C 那樣報錯，比如說某條線的一個字母打錯不管怎麼初始化或傳值進去他都會顯示 z or xxxxxx，一開始也不知道原來只是命名的問題而已，只能手動去檢查每一個變數的名稱。

另外，有些判斷式如果不小心讀取到還未定義的 wire or register，可能會發生錯誤，比如如果 always 偵測的是 clk_i，但裡面某個變數在最一開始的 cycle 還未被定義，我想到兩個解決方法，一個是先用 register 儲存變數，這樣就可以在第一個 cycle 就有值，另一個是不偵測 clk_i，而是偵測需要的變數是否有輸入(但這只能用在 latch 之外的 register，因為 latch 為了要做 pipeline 須配合 clock cycle)

Development Environment:
Window 10