Lab1 Report

資工四 胡材溢 b08208032

- Module Explanation

這次的作業我實作了以下模組。

1. Control：

Input:

(a) Hazard Detection Unit 傳入的 NoOp

(b) IF/ID 傳來的 instruction opcode

Output:

若 NoOp 為 1，則全部輸出為 0。否則規則如下：

(a) RegWrite，只要 instructions 有 rd 就設為 1，否則為 0。

(b) MemtoReg，若 instruction 為 lw 設為 1，否則為 0。

(c) MemRead，若 instruction 為 lw 設為 1，否則為 0。

(d) MemWrite，若 instruction 為 sw 設為 1，否則為 0。

(e) ALUOp，因為 beq 用不到 ALU，因此按其他的 instructions，總共四種編號。

(f) ALUSrc，只要 instructions 有 imm 就設為 1，否則為 0。

(g) Branch，若 instruction 為 beq，則設為 1，否則為 0。

2. ALU control：

Input:

(a) ID/EX 傳入的 funct7 + funct3

(b) ID/EX 傳入的 ALUOp

Output:

ALUCtrl，由 funct7 + funct3 與 ALUOp，判斷要執行的 instruction。將 ADD、SUB、MUL、AND、XOR、SLL、SRAI 分別編號為 0~6，作為輸出。

3. ALU：

根據 ALUCtrl，以及 2.提到的編號方式，執行相應的 operation。

4. Adder：

實作一加法器，將兩個 32-bits 的 inputs 相加並輸出到 output。

5. MUX2/MUX4：

這次會用到 2 取 1 或 4 取 1 的 mux，根據 select 取得相應的輸入。

6. Pipeline Registers：

總共有四個，根據圖表，設相應的 registers 儲存資料。

(a) IF/ID：儲存 PC 以及 instruction。

(b) ID/EX：RegWrite、MemtoReg、MemRead、MemWrite、ALUOp、ALUSrc、RS1data、RS2data、imm、instruction

(c) EX/MEM：RegWrite、MemtoReg、MemRead、MemWrite、ALURet、WriteData、RDaddr

(d) MEM/WB：RegWrite、MemtoReg、ALURet、ReadData、RDaddr

如同 PC.v 藉由傳入 rst 完成初始化，並在 posedge 用 nonblocking 的方式更新值。

7. Flusher：

根據讀出的 Read Data1 與 Read Data2 以及 Branch control signal，看 beq 是否成立要跳到別的 PC address。

8. Hazard Detection Unit：

若要根據某個 lw 讀入的值運算，會產生 hazard。因此先藉由 Execution Stage 的 MemRead 確定執行的 instruction 是否為 lw。若是，比較從 Data Memory 讀資料後要存到的 Register，與接著要使用的 register 是否重複，來偵測是否有 hazard 發生。

9. Forwarding Unit：

根據 Table 1、Listing 1、Listing 2，從輸入的 EX.Rs1、EXRs2、WB.Rd、WB.RegWrite、MEM.RegWrite、MEM.Rd，來判斷是否要 forward 值給 ALU。

10. CPU：

定義更多的 wires，根據 modules 的輸入與輸出對應到的 wire，將整個電路連接起來。另外我在此處定義了一 parameter four = 4，作為常數來更新 pc。

- Difficulties Encountered and Solutions in This Lab

在寫的過程中發現，取變數名稱是非常麻煩的事情，每條線都要寫成 wire，很容易弄混。如果接線的部分寫錯，會跳出 x 或 z，在不熟悉 verilog 的情況下，不太知道會錯在哪，最後發現接線接錯。

另外 pipeline 的初始化也花了很多時間才弄好，最後如同前面所述，用類似 PC.v 的方式傳入 ret 來初始化。

- Development Environment

這次的作業我使用的作業系統為 macOS，在 Visual Studio Code 上寫這次作業的程式，並下載了 Verilog-HDL/SystemVerilog/Bluespec SystemVerilog 延伸套件作為輔助。編譯的部分使用 iverilog，並藉由 iverilog -o CPU.out *.v 指令來編譯，再使用 vvp CPU.out 執行程式。除了照著 spec 編譯會出錯外，沒什麼大問題。