組員分工

組員

- 許正恣，李柚昇，鐘紫育

分工

（1）Module 部分

- 李柚昇：PC, Add_pc, Adder, Data memory, Control, IF/ID
- 鐘紫育：Sign_extend, muxes, Forwarding unit, EX/MEM, MEM/WB
- 許正恣：ALU, ALU control, Hazard Detection unit, Eq, ID/EX

（2）CPU 部分

- 李柚昇：stage 1, 4, 5
- 鐘紫育：stage 3
- 許正恣：stage 2

（3）Report 部分

- 組員分工：許正恣
- 其他部分：李柚昇，鐘紫育，許正恣

CPU Implementation

我們依照投影片最後一張圖，先分別將 `modle` 算出來有多少，每個人認領部分。完成 `modle` 後，再去分 `Stage` 接線。

Stage 1

- Stage 1 使用到的 module 有 PC, mux1, mux2, Add_PC, Instruction_Memory, IF_ID。

- 設定一些 wire 來連接各個 module。
- Branch & Flush 使用邏輯判斷寫在 CPU.v 中，不獨立出 module。

Stage 2

- Stage 2 會用到的 module 有 Control, Sign_extend, Adder, mux8, Registers, Hazard Detection, Eq, ID/EX。
- 設定一些 wire 來連接各個 module。
- Jump 的地址中用到的 shift left 和 concatenate 是直接把 PC 的前 4 位、instruction 的最後 26 位和兩位 0 串起來。
- 同理，shift left 2 位的 immediate number 也是直接把 number 的後 30 位和兩位 0 連在一起。

Stage 3

- Stage 3 使用到的 module 有 mux3, mux4, mux6, mux7, ALU, ALU Control, Fowarding Unit, EX_MEM。
- 設定一些 wire 來連接各個 module。
- 將 IE_EX 中各條訊號線接好。

Stage 4

- Stage 4 使用到的 module 有 Data_Memory, MEM_WB。
- 設定一些 wire 來連接各個 module。
- 這個 Stage 才將 RegWrite 信號送到 forwardingUnit 裡做判斷。
- ALU result 連回 stage 3 做 forwarding。

Stage 5

- Stage 5 使用到的 module 有 mux5。
- 設定一些 wire 來連接各個 module。
- 這個 stage 的 MemWrite 也被送到 forwardingUnit 裡做判斷，以及 Register file 裡決定是否寫入。

Each module

(1) PC

- 四個 input 端：clk_i, rst_i, PCWrite_i(以上 1 bit), pc_i(32 bits)，一個 output 端：pc_o(32 bits)。
- 在 rising edge 時，預設 pc_o 為 32'b0，在 RegWrite=1'b0 時，pc_o 由 pc_i 來 update。

(2) Add_PC

- 一個 input 端：pc_i(32 bits)，一個 output 端：pc_o(32 bits)。 -藉由 wire 將 pc_i + 4 輸出到 pc_o。

(3) IF_ID

- 五個 input 端：clk_i, IFIDWrite_i, IFFlush_i(以上 1 bit), pc4_i, instr_i(32 bits)，兩個 output 端：pc4_o, instr_o(32 bits)。
- 在 rising edge 時，預設 pc4_i 輸出給 pc4_o，instr_i 輸出給 instr_o。
- 若 IFIDWrite_i = 1'b0，維持預設。若 IFFlush_i = 1'b1，兩 output 都輸出 32'b0。

(4) Instruction_Memory

- 一個 input 端：addr_i(32 bits)，一個 output 端：instr_o(32 bits)。
- 將 addr_i 右移 2 bits，作為 memory 陣列的 index，輸出該 address 的 instr_o。

(5) Control

- 一個 input 端：Op_i(6 bits)，三個 output 端：Branch_o, Jump_o(1 bit), ConMux_o(8 bits)。
- 依照 Logic Design 的方式，對應 Op code 的 Truth Table 決定 instruction 類型，接著依此再對照另一個 Truth Table 決定對應的 Control signal，由高到低為 RegWrite, MemtoReg, MemRead, MemWrite, ALUSrc, ALUOp(2 bits), RegDst(all 1 bit except for ALUOp)，輸出到 ConMux_o。

- 若 instruction 類型顯示為 branch 或 jump，Branch_o 或 Jump_o 輸出為 1'b1。

(6) ALU

- 定義 3 個 input 端口（ALUCtrl 以及兩個要進行計算的 data）和一個 output 端口（計算結果）。ALUCtrl 是 4 位， 其他端口都是 32 位。
- 在一個 always block 中，根據 ALUCtrl 的不同，進行不同類型的計算。其中，新增的 lw 和 sw 的 ALUCtrl signal 與 add 相同，因為它們是要計算 offset 和 base register 地址的和。
- ALUCtrl 都是四位二進制數，具體數值來源於課本 Chapter 4。

(7) ALU control

- 定義兩個 input 端口：funct (6 bits), ALUOp (2 bits), 一個 output 端口：ALUCtrl (4 bits)。
- 先判斷 ALUOp。如果 ALUOp 等於 10，說明當前執行的 instruction 是 R type，下面再根據 funct 判斷當前的 instruction 具體要執行哪種運算，并分配相應的 ALUCtrl 的數值。
- 如果 ALUOp 等於 00，說明當前執行的 instruction 是 lw, sw, addi 三個中的某一個，它們都是要執行相加的運算，所以讓 ALUCtrl 對應於 add 的 ALUCtrl 數值。

(8) Hazard Detection unit

- 四個 input 端口：MemRead, ID/EX.Rt, IF/ID.Rs, IF/ID.Rt。四個 output 端口：PCWrite, IF/IDWrite, mux8_select, flush。input 中除了 MemRead 是 1 位，其他都是 5 位。output 都是 1 位。
- 如果 MemRead 等於 1（說明 instruction 是 sw），且 ID/EX.Rt 等於 IF/ID.Rs 或 IF/ID.Rt，說明會發生 load-use hazard，所以把 PCWrite 和 IF/IDWrite 設 0，mux8_select 和 flush 設 1。
- 反之，就把 PCWrtie 和 IF/IDWrite 設 1，mux8_select 和 flush 設 0，讓程式正常運行下去。

(9) Eq

- 兩個 input：data1 和 data2 (32 bits)，output：1 位數值，指示 data1 與 data2 是否相等。
- 如果 data1 等於 data2，output 設 1，否則 output 設 0。

(10) ID/EX

- 定義兩個 input：clk，in (N bits)，一個 output：out (N bits)，另外定義一個 parameter N。
- 在 clk 的 posedge，令 out 等於 in。
- 在 CPU 中會定義不同長度的 input。

(11) Sign_Extend:

- 將輸入的 16bits，用最後面的第 16bit 重複放在 17-32 的位置上。

(12) Mux:

- 在 mux 的部分，我依照三種不同的輸入輸出，將 mux 寫成三種形式分別為兩個輸入(1,2,4,5,8)、三個輸入(6,7)、五個 bits 的兩個輸入(3)。

(13) Forwarding unit

- 依照作業投影片 P.7 上的 pseudocode，改成 verilog。

(14) EX/MEM & MEM/WB

- 幫每一個 output 先設定預設值都為 0,之後將輸入接近輸出。

問題和解決

(1) PC 需要設定初始值為 0

- 在 PC 和 testbench 里加了一個 input：reset。

(2) 前幾個 instruction 執行時，後面 stage 的一些 signal 還沒有有效值

- 更改了相關 control signal 的條件語句，首先設 0，遇到 trigger 條件再修改值。

(3) Forwarding Unit 的條件設置不夠準確

- 修改了條件語句，首先把 forwarding 的 signal 都設 0，遇到 trigger 再改成 01 或 10。

(4) PC 能夠正常運作，但從 register 中輸出的 read data1 & read data2 都會是 32'h0。

(5) 在 debug 的時候發現設定各個 reg 跟 wire 的時候有些缺陷，只有考慮到 1,0 而忽略了 x 的可能，導致第一次跑的時候許多值都出不來，後來經過修改後才有所改善。