NTU CA Lab2 Report
B09901027 賀崇恩

1. Modules Explanation

這個部分我會先給出這次作業和 Lab1 做出的變更，然後下面再給出 Lab1 的 module explanation，這樣可以更一目了然兩次作業的內容有何不同。

ALU_Control
這次的 ALU Control 需要判斷 instruction 是否是 beq type，但因為 beq 的 function code (10 bit)擺放的是 immediate，所以我是先在 ID 的 Control module 決定 instruction 是否是 beq，如果是的話用 ALUOp 通知 ALU Control 該 instruction 是 beq。ALU Control 在收到 instruction 是 beq 時通知 ALU 做減法。

ALU
除了因為 testbench 要 bind ALU.data_o 所以把 output data_o 改成 output signed，其餘沒有不同。

branch_predictor
這個 module 輸入 result (EX stage 的 data_o) 以及 update (ID stage 的 Control 判斷是否為 beq)。如果收到 ID stage 的 instruction 是 beq，它會先把這個狀態存進 reg last_instr_beq，然後在下一個 cycle 根據 result 來 update state。輸出的 predict 即根據 state 來輸出。

Control
如前所述，Control 的不同是加入了判斷 instruction 是否為 beq，如果是的話就令 ALUOp 是 beq 的 type，然後 ALUSrc 取 0 (取來自 register)。

RegIDEX
這個 pipeline register 加入了四個新儲存的東西: Branch (即 ID 的 instruction 是否為 beq)，last_flush (ID stage 是否 take branch)，target PC (如果 evaluate branch 的結果為 taken，branch target 為何)，PC (現在的 PC，如果 evaluate branch 的結果是 not taken 則 branch target 是這個 PC+4) 以及 flush 訊號 (是否 flush 該 pipeline register)。

CPU
CPU 裡不一樣的邏輯主要如下。
1. IF/ID flush
IF/ID flush 的信號變成: ID_Branching && predictor (ID instruction 為 Branching，

且 predictor 為 true)

2. ID/EX flush
ID/EX flush 的信號為 EX_Branching && ~EX_last_flush && (ALU out == 0) 或 EX_Branching && EX_last_flush && (ALU out != 0) (也就是: EX stage instruction 為 beq，但在上個 stage 選擇 not taken 且 EX evaluate result 選擇 taken 或 上個 stage not taken 但 EX evaluate result 選擇 taken)。此信號稱為 EX_BEQ_flush。

3. next PC selection
下一個 PC 選擇的邏輯如下: (兩個 MUX 串接)
(1) 是否由 EX stage 決定 target PC。(即: select 訊號為 EX_BEQ_flush) 如果是的話，EX stage 決定 target PC 的方式是: 若 evaluate 結果為 taken，則選擇 ID/EX pipeline register 儲存的 target address。如果為 not taken，則選擇 ID/EX 儲存的 PC+4。
(2) ID stage 決定 target PC 的內容方式仍和 lab1 相同，只差在 select 訊號是 ID_Branching && predictor (ID stage 是否選擇 taken)。
先決定 ID 再決定 EX 的內容的原因是，EX 的結果一定是對的，但 ID 是預測的，所以如果 EX 跟 ID stage 同時決定要 flush，應該是 EX 優先。

Lab1 的 module explanation:

Adder
這個 module 把 32bit 的兩個 input 加起來 assign 給 output。這是一個純 combinational 的 module。

ALU_control
這個 module 用 function code 決定 R-type 或 I-type instruction (從 ID 的 Control module 過來的)，再決定 ALU 的控制信號(ADD XOR SLL SUB MUL SRA AND 七種)。這是一個純 combinational 的 module。

ALU
這個 module 如同上次 single cycle 的作業，是由 ALU_Control 送過來的信號對兩個 32 bit input 做運算，其中 input data 宣告為 signed，在 shift right arithmetic 才會有 sign extension 的效果。這是一個純 combinational 的 module。

Control
這個 module 由 operation code 決定 ALUSrc, RegWrite, RegWrite, MemRead, MemWrite, MemToReg, Branching 信號。其中比較特別的是 NoOp input，是為了

load use hazard 要加入一個 nop instruction 用的，NoOp 為 1 的時候所有控制信號均為 0 送入 pipeline。這是一個純 combinational 的 module。

CPU
這個 module 是 design 的 top module，基本上就是定義所有 module 的 wire connection，以及提供 stall 和 flush 的 wire 給 testbench 計算。另外，flush pipeline 都直接定義在各個有關 module (PC，PC 的 mux 等等)的腳位，沒有額外的控制 module。

Data_Memory
這個 module 是存放 data 的 memory (MEM stage)，根據 MemRead 讀取 memory，且根據 MemWrite 看要不要寫入 input address 和對應的 data。這是一個 sequential module。

Forwarding_Unit
這個 module 使用 EX stage 的 Register source、WB stage 的 Register destination 和 RegWrite、MEM stage 的 Register destination 和 RegWrite 檢查 pipeline 有沒有 MEM 或 EX data hazard，並 output forwarding signal 讓 EX stage 的 mux 可以切換到正確的 forwarding data。這是一個純 combinational 的 module。

Hazard_Detection
這個 module 使用 EX stage 的 MemRead 跟 RegDest、ID stage 的 Register source 檢查 load use hazard，如果有偵測到的話就會令 pipeline stall (送 NoOp 給 Control、送 stall 給 RegIFID、同時讓 PC register 不更新) 這是一個純 combinational 的 module。

ImmGen
這個 module 吃進整個 instruction，並且根據 instruction 的需要 concatenate + sign extension 出對應的 immediate field。需要 immediate field 的 instruction 包含: I-type, lw, sw, 以及 beq。這是一個純 combinational 的 module。

Instruction_Memory
這個 module 根據 address input 讀取對應的 instruction，不同於 Data_Memory 的是這個 module 沒有提供寫入的信號。這是一個 sequential module。

MUX32_2
這個 module 是 4-way 32bit 的 MUX，提供 EX 的 forwarding data 的選擇(從 MEM forward / 從 WB forward / 不 forward)。這是一個純 combinational 的

module。

MUX32
這個 module 提供一個 2-way 32bit MUX。這是一個純 combinational 的 module。

PC
這個 module 存放目前的 program counter，不同於上次的是加入了 PCWrite signal，以便在 pipeline stall 時 stall 一次 PC 的前進。這是一個 sequential module。

RegEXMEM, RegIDEX, RegIFID, RegMEMWB
這些 modules 基本上就是 pipeline register，只是把 input data 在下一個 cycle 送到 output 而已。比較不同的是 RegIFID 有 stall 跟 flush 信號，stall 時 pc 和 instruction field 均不動，flush 時 instruction registers 要存 0。這是一個 sequential module。

2. Difficulties Encountered and Solutions in This Lab

寫的過程中遇到要 debug 的問題也是各種東西漏加，但整體花的時間不久。主要是各個 select 訊號要選誰會選錯，例如 EX stage 要 flush 有兩種可能，一個是前面選擇 not taken 但其實是 taken，另一個是相反的狀況，一開始我好像有只寫了一個的情形，所以執行的結果就不對。

3. Development Environment

我使用了兩個 development 環境: 我的電腦的 Windows WSL (Ubuntu 20.04) + iverilog + gtkwave 以及電機系 IC lab 工作站 (CentOS 7) + ncverilog + nWave。我使用 diff -w 指令來比較我的輸出跟 sample output 的差異。