NTU CA Lab1 Report
B09901027 賀崇恩

1. Modules 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

一次寫完之後，我的邏輯錯誤並不多，邏輯的錯誤主要在於有些地方沒有加 default case、BEQ 的 opcode 沒有加入 switch case 等等。我花比較多時間解決的是關於 ImmGen 的語法，包含一開始 concatenate 起來根本沒有 32 bit、以及花最久的一個 bug 是不知道 verilog 的 shift operator 計算優先序比加法還後面，讓我以為 a + b >> 1 是 a + (b>>1)，這個點真的是瞪了好久才發現我的 Branching address 計算是錯在這裡。(instruction_4 的最後一個 beq offset 是-20，沒有錯誤之前 branching target 甚至是往前跳，我還想說減的變加的真的是見鬼了。)

3. Development Environment

我使用了兩個 development 環境: 我的電腦的 Windows WSL (Ubuntu 20.04) + iverilog 以及電機系 IC lab 工作站 (CentOS 7) + ncverilog + nWave。在使用 diff 指令比較我的 output 跟 golden 的差異的時候有遇到 golden 的換行字符好像是 windows 的的問題，因此我有先 dos2unix [golden output filename] 再使用 diff command。