Computer Architecture Project 1 Report

隊名：I'm Pasta 紅茶 🥤
謝獻沅 B07902049
黃昱翔 B07502159
陳正康 B07902143

Module 說明

1. Control

   Control負責CPU的signal分配，比起HW4新增了branch_o和NoOp_i，分別處理SB_type和nop instruction，以及負責lw/sw的MemRead_o和MemWrite_o，signal的output按照上課投影片的signal表及spec，依照不同type的instruction做分配。為了實現nop我們採用所有output設為0的作法，確保nop不會改變任何register或memory裡的資料。

2. ALU_Control

   ALU_Control依據Control傳來的ALUOp和function3或function7的值來決定ALU接下來要做什麼動作，首先會先判斷instruction type，再根據Funct_i來決定ALUCtl_o，原則上都是按照spec提供的表來做判斷依據。

3. Forwarding Unit

   Forward Unit 用來檢測是否有data hazard發生，其判斷依據為檢查由ID/EX stage傳來之register 號碼與EX/MEM, MEM/WB stage 傳回之register 號碼進行比較，若與EX/MEM相同則為EX hazard而若與MEM/WB相同則為MEM hazard。Forward Unit會根據不同的hazard傳送其對應的signal到位於EX stage的MUX從而選擇對應來源的register做為運算用之數值。其中需特別注意的是Forward Unit需要特別檢查目標是否為0號reg還有要判定EX hazard若與MEM hazard同時發生則須處理來自EX hazard之register (由於EX hazard位置較MEM hazard新)。

4. Hazard Detection Unit

   Hazard Detection Unit負責判斷load是否發生Hazard，判斷的依據為如果是load instruction(MemRead == 1)且當Rd == Rs1或Rd == Rs2時，就代表load

來不及把結果給接下來的instruction使用，此時需要stall pipeline來解決，值得一提的是，當下個instruction是I-Type的時候，由於Rs2並不在I-Type，所以需要特判來協助處理，並且也額外處理如果Rd = x0的狀況。stall的實現方面，當stall的時候要傳NoOp給Control，告訴Control是nop instruction，且讓IF/ID的pipeline reg停一個cycle，以及告訴PC要stall，如此一來確保整個pipeline會stall一個cycle。

5. CPU & Pipeline Registers

   CPU把各個module組起來，基本上都照著spec裡的圖接。Input有clk, start和rst。clk在posedge時，Pipeline registers、PC和register file會寫入新的值，讓cpu中的指令前進一個stage。加入pipeline registers是為了讓同一個訊號在不同stage隔離，實作中有例如rd_ID, rd_EX, rd_MEM, rd_WB等等宣告成reg的變數作為pipeline registers，並在stage之間用always block遞值，觸發條件是clk的posedge。如果clk posedge發生時rst是1，則初始化成0。以下是各個stage的register：

   i. IF/ID stage registers
      (a) pc
      (b) instruction
   ii. ID/EX stage registers
      (a) Control: RegWrite, MemToReg, MemRead, MemWrite, ALUOp, ALUSrc
      (b) Register data (for ALU input): RS1, RS2
      (c) Immediate Value (for ALU input)
      (d) Register number: RS1, RS2, RD
      (e) funct3, funct7 (for ALU Control)
   iii. EX/MEM stage registers
      (a) Control: RegWrite, MemToReg, MemRead, MemWrite
      (b) ALU Result
      (c) RS2 Data (for store)
      (d) RD number
   iv. MEM/WB stage registers

(a) Control: RegWrite, MemToReg

(b) ALU Result

(c) Memory Data (for load)

(d) RD number

6. ALU

   ALU 為EX stage中用來計算的主要元件，其中ALU會接受由兩個MUX傳來的值再根據ALU Control所傳來之signal決定這兩個值所需要進行之運算。在ALU中最主要要注意的是input和output要小心正負號的問題，這問題我們主要將兩個input數值指定為signed還有將output設定為reg signed從而解決。

7. ImmGen

   ImmGen的部分我們這組是直接將32個bits的instruction直接讀入ImmGen module在進行判斷要輸出的內容，主要判斷的依據是根據opcode決定他是哪一種type的instruction進而決定要輸出的32 bits imm值的擺法，其中我們將除了需要imm值的指令外輸出值都設為x，此部分由於如果不需要imm的instruction自然會在後續的MUX不選擇Imm的輸出故無需擔憂。

8. testbench

   把Stall和Flush的訊號接上，讓它可以加總stall和flush發生次數。另外，Instruction Memory的初始值改成nop(= addi x0, x0, 0 = 0x13)。

組員與工作分配

1. Memory load, store (改Control, ALU Control)
2. Hazard Detection (new module)
3. Forward (new module)
4. 整合+Pipeline+Branch (改CPU, testbench, +pipeline register)
5. Report
6. Immediate-Value Generator (把SignExtend改成ImmGen)
7. Debugging

{1, 2} 謝獻沅 (學號：B07902049)

{3, 6} 黃昱翔 (學號：B07502159)

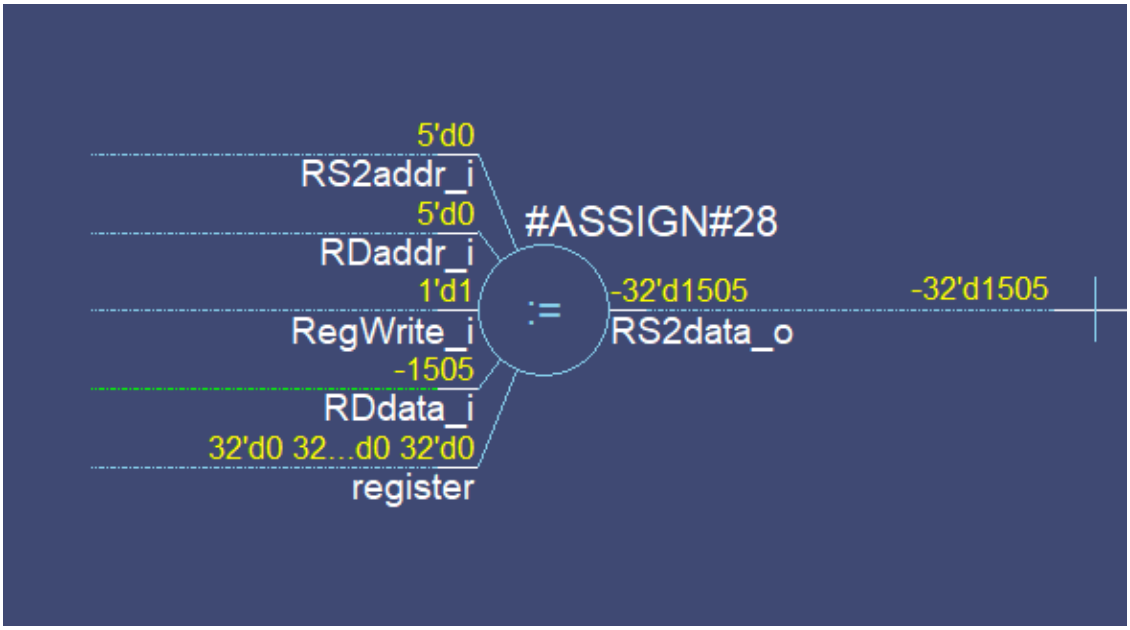
{4} 陳正康 (學號：B07902143)

{5, 7} 三人合作

遇到的問題

1. 每個pipeline register都要在reset時初始化成nop(addi x0,x0,0)，control都設成0，否則在剛開始時會有control訊號unknown
2. pc到某個cycle後開始變unknown，但指令還沒跑完
   原因：當instruction處厘到倒數4個時，因為pipeline的關係，前面的stage會讀到instruction memory全為0的地方
   解法：塞4個nop在所有instruction之後。我們為了方便，直接在testbench.v把instruction memory初始化成nop(= addi x0, x0, 0 = 0x13)
3. EX/MEM應該要接到Forward之後的Rs2Data而不是Forward之前，助教的圖有錯(12/5圖已修正)
4. ModelSIM會在pc還是unknown時就Print message
   解法：此問題在iverilog不會發生
5. lw/sw遇到未對齊的address時，助教給的data memory會捨棄除以4的餘數。如果要支援未對齊的address，Data memory可能需要做修改
   如：`lw x5, 1(x0)`在此設計中會讀出Mem[0:3]而非Mem[1:4]
6. Register file is forwarding x0
   助教給的Register.v有類似forwarding的東西，同時read/write同一個register時會直接forward出去。但當RS1addr=RDaddr=0且RegWrite=1時似乎會把不該寫入的值forward出去。我們嘗試修改了Register.v來解決這個問題(但沒有放進submission)。另一種可能解法是讓control另外吃rd來讓rd=0時disable RegWrite。但考慮x0理論上不會出現在rd，最後的submission裡並沒有處理這個問題。

圖示：



原本的Register.v:

```
assign RS1data_o = (RS1addr_i == RDaddr_i && RegWrite_i)? RDdata_i :
register[RS1addr_i];
assign RS2data_o = (RS2addr_i == RDaddr_i && RegWrite_i)? RDdata_i :
register[RS2addr_i];
```

修改後的Register.v:

```
assign  RS1data_o = RS1addr_i == 0 ? 0 :
                    RS1addr_i == RDaddr_i && RegWrite_i ? RDdata_i :
                    register[RS1addr_i];
assign  RS2data_o = RS2addr_i == 0 ? 0 :
                    RS2addr_i == RDaddr_i && RegWrite_i ? RDdata_i :
                    register[RS2addr_i];
```

造成錯誤的範例：

```
addi x0, x0, -1505 <-- WB, rd=0
mul x0, x0, x0
sub x0, x0, x0
add x1, x1, x0      <-- ID, rs2=0, got -1505
srai x1, x0, 13
```

7. Hazard Detection Unit中，出現load-use hazard時，若後一個指令是i-type，並沒有用到rs2，要特判。課本的解法並不完整，若用課本的寫法，會讓以下誤判成要stall：

```
lw x1, 0(x0)
addi x3, x2, -1503   <-- rs2 = instruction[24:20] = 1
```

執行環境

主要iverilog 10.3 on Ubuntu 20.04

有些debug是在ModelSIM 10.4a on Windows 10

Repository

https://github.com/soyccan/junk-cpu/tree/pipelined-11inst