lab5
0510008 藍挺毓
0510026 陳司瑋

Computer Organization

Source code and the note:

這次 code 有許多部份和 lab4 一樣。以下為這次修改的部分。

- MUX_4to1.v

```
    parameter size = 0;
        input [size-1:0]data0_i;
        input [size-1:0]data1_i;
        input [size-1:0]data2_i;
        input [size-1:0]data3_i;

        input  [1:0]select_i;
        output [size-1:0]data_o;
        reg    [size-1:0] data_o;
        always@(*)begin
        case(select_i)
        2'b00: data_o = data0_i;
        2'b01: data_o = data1_i;
        2'b10: data_o = data2_i;
        2'b11: data_o = data3_i;
        endcase
```

這是這次新增的 module，但寫法和 2 to 1 MUX 還有 3 to 1 MUX 相同。

- hazard_detection.v

```
if(id_ex_memread && ((if_id_regRs == id_ex_regRt)|(if_id_regRt == id_ex_regRt)))begin
    PCwrite = 1'b0;
    if_idWrite = 1'b0;
    if_flush = 1'b1;
    id_flush = 1'b0;
    ex_flush = 1'b0;
end
else if(PC_source)begin
    PCwrite = 1'b1;
    if_idWrite = 1'b1;
    if_flush = 1'b1;
    id_flush = 1'b1;
    ex_flush = 1'b1;
end
else begin
    PCwrite = 1'b1;
    if_idWrite = 1'b1;
    if_flush = 1'b0;
    id_flush = 1'b0;
    ex_flush = 1'b0;
end
```

依照 detection 的訊號，設定 flush 的 control 值，讓 control 給需要 flush 0 的值。

- forwarding_unit.v

lab5
0510008 藍挺毓
0510026 陳司瑋

```
always@(*)begin
 if( (ex_mem_RegWrite && (ex_mem_RegRd!=0)) && (ex_mem_RegRd == id_ex_RegRs) ) ForwardA=2'b01;
 else if ( mem_wb_RegWrite && (mem_wb_RegRd!=0) && (mem_wb_RegRd==id_ex_RegRs)) ForwardA=2'b10;
 else ForwardA=2'b00;

 if(ex_mem_RegWrite && (ex_mem_RegRd!=0) && (ex_mem_RegRd==id_ex_RegRt)) ForwardB = 2'b01;
 else if ( mem_wb_RegWrite && (mem_wb_RegRd!=0) && (mem_wb_RegRd==id_ex_RegRt) ) ForwardB=2'b10;
 else ForwardB = 2'b00;

end
```

寫法跟老師上課講義一樣，只是把 else if 部分，與 if 重複判斷的地方拿掉，因為程式必定會先判斷 if，所以這樣可以簡化程式。
另外講義提供的 forwarding 控制值和助教這次提供的 diagram 不一樣，我們選擇維持 forwarding_unit 的 0,1 值，更改接入 MUX 的 data 順序。

- decoder.v

```
R_format <= (~instr_op_i[5]) & (~instr_op_i[4]) & (~instr_op_i[3]) & (~instr_op_i[2]) & (~instr_op_i[1]) & (~instr_op_i[0]);
addi <= (~instr_op_i[5]) & (~instr_op_i[4]) & (instr_op_i[3]) & (~instr_op_i[2]) & (~instr_op_i[1]) & (~instr_op_i[0]);
slti <= (~instr_op_i[5]) & (~instr_op_i[4]) & (instr_op_i[3]) & (~instr_op_i[2]) & (instr_op_i[1]) & (~instr_op_i[0]);
beq <= (~instr_op_i[5]) & (~instr_op_i[4]) & (~instr_op_i[3]) & (instr_op_i[2]) & (~instr_op_i[1]) & (~instr_op_i[0]);
bne <= (~instr_op_i[5]) & (~instr_op_i[4]) & (~instr_op_i[3]) & (instr_op_i[2]) & (~instr_op_i[1]) & (instr_op_i[0]);
bge <= (~instr_op_i[5]) & (~instr_op_i[4]) & (~instr_op_i[3]) & (~instr_op_i[2]) & (~instr_op_i[1]) & (instr_op_i[0]);
bgt <=(~instr_op_i[5]) & (~instr_op_i[4]) & (~instr_op_i[3]) & (instr_op_i[2]) & (instr_op_i[1]) & (instr_op_i[0]);
lw <= (instr_op_i[5]) & (~instr_op_i[4]) & (~instr_op_i[3]) & (~instr_op_i[2]) & (instr_op_i[1]) & (instr_op_i[0]);
sw <= (instr_op_i[5]) & (~instr_op_i[4]) & (instr_op_i[3]) & (~instr_op_i[2]) & (instr_op_i[1]) & (instr_op_i[0]);
//jump <= (~instr_op_i[5]) & (~instr_op_i[4]) & (~instr_op_i[3]) & (~instr_op_i[2]) & (instr_op_i[1]) & (~instr_op_i[0]);
//jal <= (~instr_op_i[5]) & (~instr_op_i[4]) & (~instr_op_i[3]) & (~instr_op_i[2]) & (instr_op_i[1]) & (instr_op_i[0]);
//jr <= (~instr_op_i[5]) & (~instr_op_i[4]) & (~instr_op_i[3]) & (~instr_op_i[2]) & (~instr_op_i[1]) & (~instr_op_i[0]);

//RegDst_o[1] <= jal;
RegDst_o <= R_format;
ALUSrc_o <= lw|sw|addi|slti;
//MemtoReg_o[1] <= jal;
MemtoReg_o <= R_format|addi|slti;
RegWrite_o <= R_format|lw|addi|slti;
MemRead_o <= lw;
MemWrite_o <= sw;
Branch_o <= beq|bne|bgt|bge;
Branchtype_o[1] <=bne|bge;
Branchtype_o[0] <=bgt|bne;
ALU_op_o[2] <= addi|slti;
ALU_op_o[1] <= R_format;
ALU_op_o[0] <= bne|bgt|bge|beq|slti;
```

decoder 的部分增加了 advanced instructions 的判斷，及新的控制線 Branchtype_o。ALU_Ctrl.v 則因為 advanced instructions 都是 R-type 所以不需要修改。

- Pipe_reg_control.v

```
always@(posedge clk_i) begin
    if(~rst_i)
        data_o <= 0;
    else if(write)
        data_o <= data_i;
        else data_o<=data_o;
end
```

因為這次的 IF/ID 需要接 control 線，所以另外寫了一個有 control 線的 pipeline register 給 IF/ID。

- ProgramCounter.v

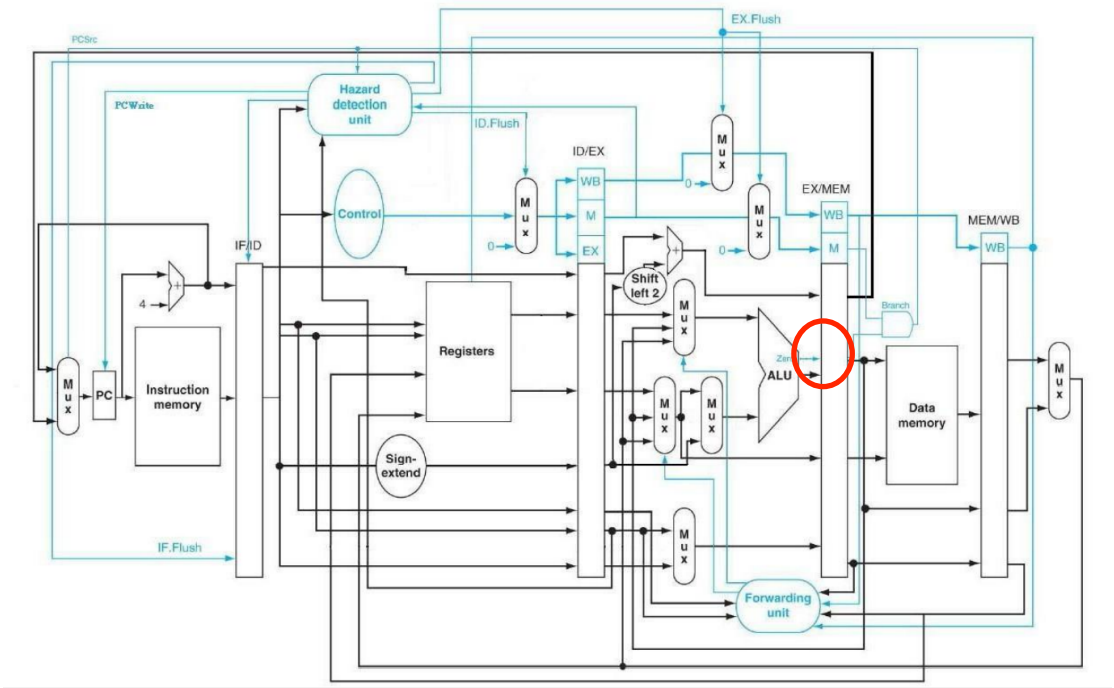
lab5
0510008 藍挺毓
0510026 陳司瑋

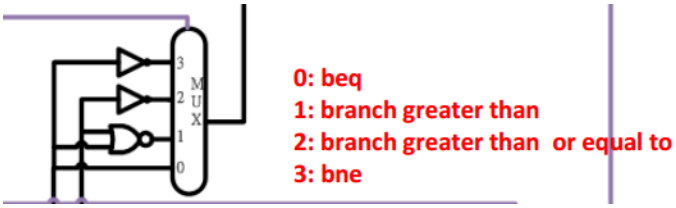
```
if(~rst_i)
    pc_out_o <= 0;

else if(pc_write)begin


       pc_out_o <= pc_in_i;
end
else pc_out_o <= pc_out_o;
```

和 pipeline register 一樣因為有 control 線，所以需要小小的修改。

- Pipe_CPU_1.v
  只是依照下面的 diagram 接線，而個別 module 也在上面說明了。

Architecture diagrams:





basic instruction set 的部分用助教提供的 architecture。
Advanced instruction set 的部分參考助教 lab3 提供的 diagram，將上圖的 MUX 放到這次助教提供的 diagram 中的 ALU 後面，EX/MEM pipe register 前面，如上圖紅色圓圈所框的地方。

lab5
0510008 藍挺毓
0510026 陳司瑋

Hardware module analysis:

這次 hardware 和上次明顯不同的是多了 data hazard 的 detection 以及因為 data hazard 的 forwarding 及 flush 機制。
以 forwarding unit 判斷發生 data hazard，則在 mux 選擇還未存回 reg_file 的值，也就是直接把 ex/mem mem/wb register 相對應的值拉給 ALU。
以 hazard detection unit 判斷 load-use hazard 或 branch hazard，這兩種 hazard 分別需要有 stall 或 flush 掉正在執行的 stage 的值。故由 hazrd detection unit 判斷後，在所需 mux 選擇 0 或讓 reg 等於零，就能 flush。
另外，是四種 branch 的判斷。需要再加 MUX 做是否符合條件需要 branch type。
大部分的截圖和說明都在上幾題解釋清楚了。

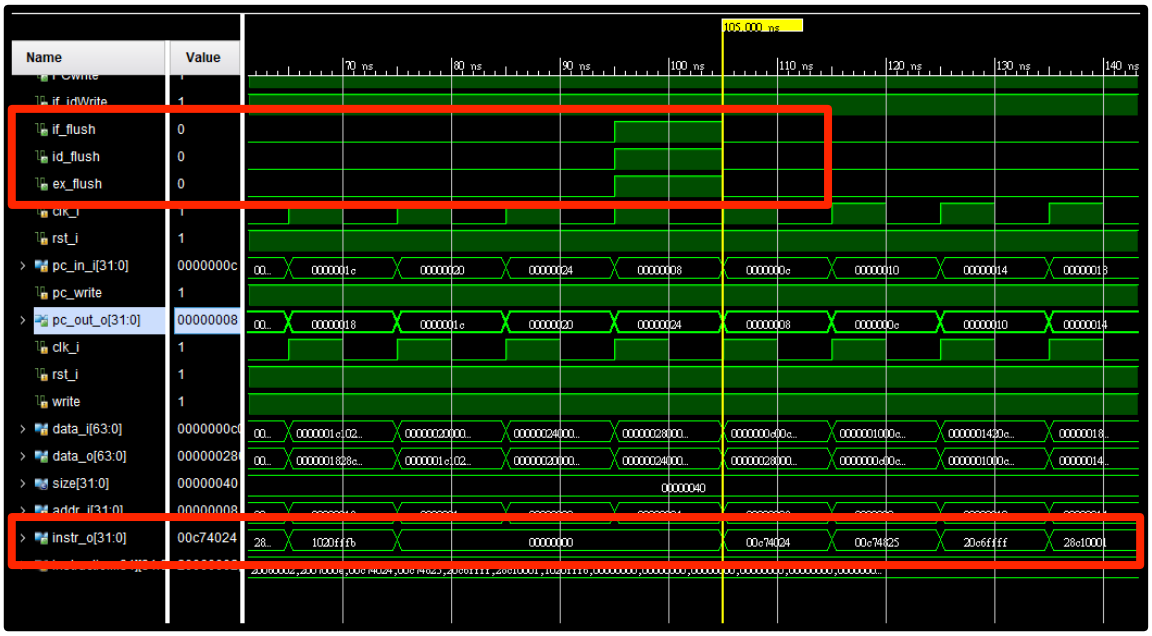
Finished part:

```
Register=============================================================

r0=          0, r1=         16, r2=        256, r3=          8, r4=         16, r5=          8, r6=         24, r7=         26

r8=          8, r9=          1, r10=          0, r11=          0, r12=          0, r13=          0, r14=          0, r15=          0

r16=          0, r17=          0, r18=          0, r19=          0, r20=          0, r21=          0, r22=          0, r23=          0

r24=          0, r25=          0, r26=          0, r27=          0, r28=          0, r29=          0, r30=          0, r31=          0


Memory=============================================================

m0=         0, m1=        16, m2=         0, m3=         0, m4=         0, m5=         0, m6=         0, m7=         0

m8=         0, m9=         0, m10=         0, m11=         0, m12=         0, m13=         0, m14=         0, m15=         0

r16=         0, m17=         0, m18=         0, m19=         0, m20=         0, m21=         0, m22=         0, m23=         0

m24=         0, m25=         0, m26=         0, m27=         0, m28=         0, m29=         0, m30=         0, m31=         0
INFO: [USF-XSim-96] XSim completed. Design snapshot 'TestBench_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

Problems you met and solutions:

因為沒有提供 advanced instruction set 的測資，所以寫測資且翻譯成 machine code 花了很多時間，不過之後想到如何在現有 code 中加簡單 branch 的測試，測試就很順利了。但因為只是簡單的測試，所以我們的 code 會進入無限迴圈，不過我們只是要判斷會正常運作，所以看 wave 確定程式跑對就 Ok 了，沒有寫出有意義的 test bench。
另外我們拿了 lab2 的 code 來測試 hazard 的運作。

lab5
0510008 藍挺毓
0510026 陳司瑋



此為用 lab2 testcase 截圖，在最後的 beq 指令會發生 branch hazard 的狀況，可以看到完成最後一行的指令後，instrustion 讀到 3 個 cycle 的 0 值，但在判斷 hazard 後 flush 等於 1 則讓 instruction 順利讀到該有的指令。

Division of work:

和之前一樣，我們約了幾次見面一起討論，一起打 code，所以分工不好分清楚。不過這次的 bug 比之前多，所以沒約見面的時候會自己想一下之前看到的錯誤的 wave 的原因，及試著修改 code，見面的時候再直接說自己改了甚麼。
比較算分開做的部分是一開始藍挺毓打 forwarding_unit.v，陳司瑋打 hazard_detection.v。不過 debug 的時候都有再一起看並做些微修改。

Summary:

這次的 lab 比之前花我們更多時間，debug 花了蠻多的時間，但都不是大錯，必較多是 pipeline register 的數值算錯。可能是因為這次的 pipeline register 要在之前的上面再多加一些空間，比較亂所導致。
經過一學期的 lab，發現 dlab 真的是堂偉大的課，經過一個學期學習 verilog，覺得能力真的增長許多，讓這學期的 lab 寫的都算順利。但也要謝謝這些 CPU 作業讓我們發現 verilog 其實還是蠻可愛的，且可以完成很多很實際的事情。
另外這堂課很開心的是作業可以一起寫，這樣 debug 才不會很痛苦，可以一起看 wave，比較不無聊，且抓錯不會一直陷入思考的盲點，快很多。