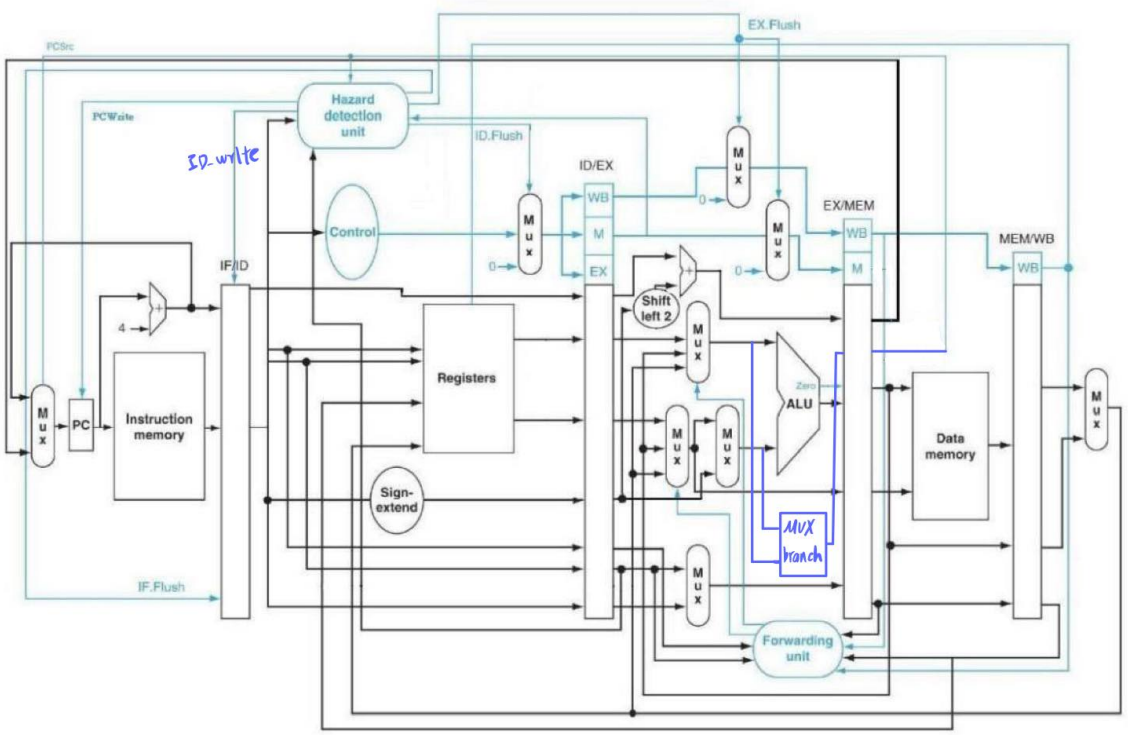
Computer Organization Lab5

Name: 郭昀 ID: 109550018

Architecture diagrams:

這次的 Lab 是結合 Lab4 的 pipeline CPU 加上 Lab3 的 branch 處理，所以我有在助教 Lab5 中提供的 Architecture diagrams 加入了 branch 相關的 module。另外，這次也要處理 Forwording 跟 load-use data hazard, branch hazard 的問題，所以我也加入了 Forwording Unit 及 Hazard Detection Unit。



Hardware module analysis:

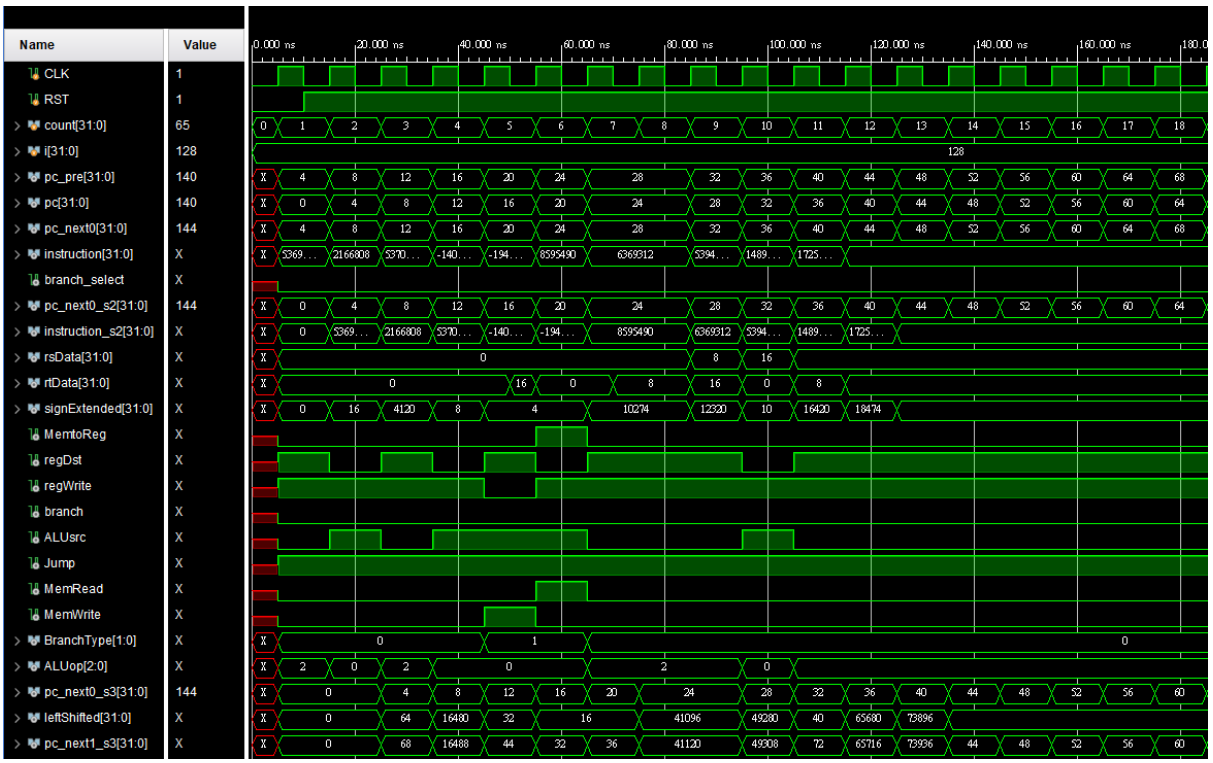
因為這次有多了 branch 相關的指令，所以需要額外處理 branch hazard 的問題，而我是透過 Decoder 產生每一個 branch 指令的類型 (branch type)，再根據他的類型來決定 branch 要不要跳的判斷式 (如下圖)。除此之外，我在 pipe_reg 另外加入 stall 跟 flush 兩個 input，透過 stall 跟 flush 的訊號來控制 pipe_reg 的 output。

```
MUX_4to1 #(.size(1)) Mux_branch(
    .data0_i(ALU_rs == MUX_ALUsrc),
    .data1_i(ALU_rs >= MUX_ALUsrc),
    .data2_i(ALU_rs > MUX_ALUsrc),
    .data3_i(ALU_rs != MUX_ALUsrc),
    .select_i(BranchType_s3),
    .data_o(Branch_new)
);
```

Finished part:

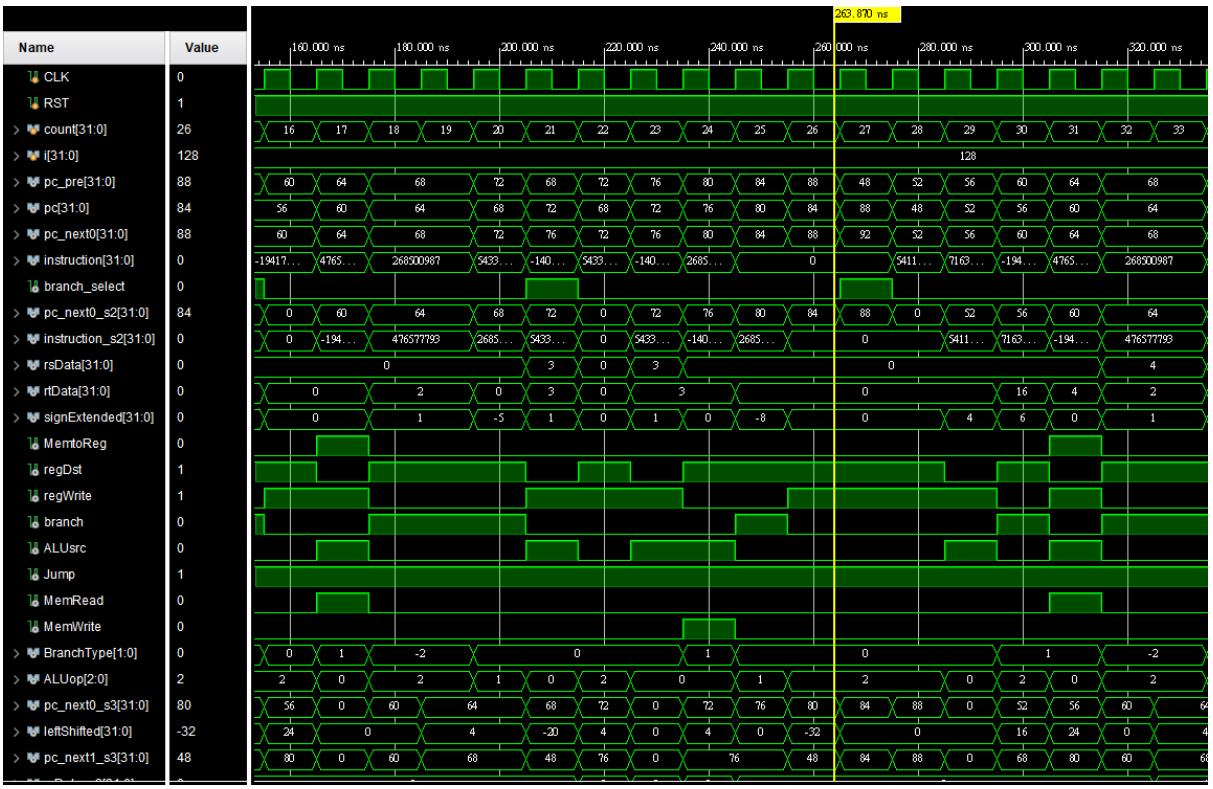
Test 1 :

```
################################ clk_count = 64####################################
=======================================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

m16=    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
```



Test 2 :

```
################################### clk_count = 64#####################################
=======================================Register=======================================
r0 =    0, r1 =    0, r2 =   16, r3 =    6, r4 =    0, r5 =   16, r6 =    0, r7 =    0

r8 =    2, r9 =    0, 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 =    4, m1 =    1, m2 =    0, m3 =    6, m4 =    0, m5 =    0, m6 =    0, m7 =    0

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

m16=    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
```



兩筆測資結果都與助教給的答案相同，也跟自己推導的結果相同！(我有把每一個訊號輸出到波型圖上方便 debug)

Problems you met and solutions:

我一開始在設計 hazard detection unit 中處理 branch hazard 的時候沒有分清楚每一個 stage 的 flush，導致我在 stall 的時候會把 EX stage 也 flush 掉了，後來把 load-use data hazard 跟 branch hazard 分開來以及加上分辨 branch type 的設計，就可以清楚區分 stall 以及 flush 各自要做的事情了。

```
{ pc_stall, ID_stall, IF_flush, ID_flush, EX_flush } = 5'b00000;

if (MemRead_s3 && ((RegisterRt_s3 == RegisterRs) || (RegisterRt_s3 == RegisterRt)))
    { pc_stall, ID_stall, ID_flush } = 3'b111;


if (branch)
    { IF_flush, ID_flush, EX_flush } = 3'b111;
```

Summary:

這次的 lab 雖然只是加入了 forwarding unit 以及 hazard detection unit 而已，但因為比上次多了 branch 相關的指令，所以遠比上次複雜許多，要非常清楚每一種 hazard 相對應需要做的事，並且要一步步推導指令的執行順序，才能用較高的效率 debug，這次真的花了很久在 debug QQ.