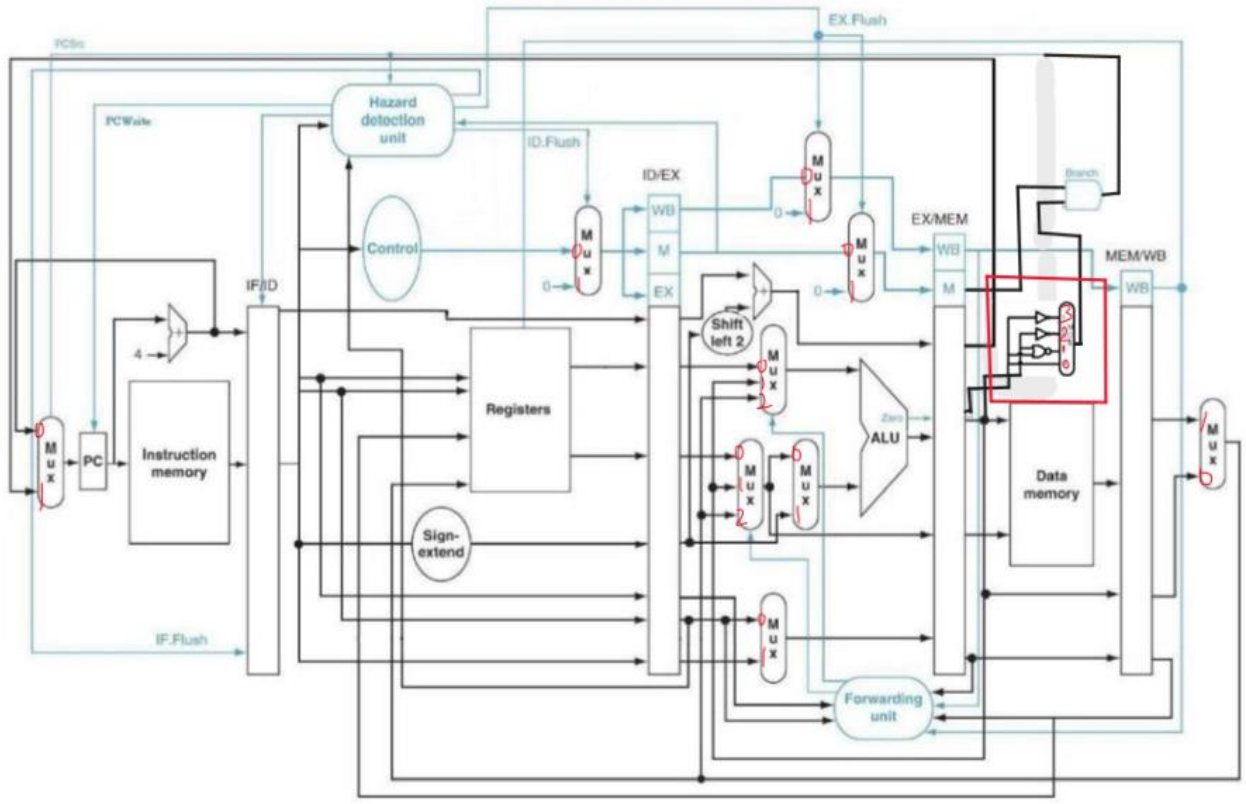
Computer Organization Lab5

Name: 施柏江

ID: 110550108

Architecture diagrams:



這次 Lab 要實作的 BEQ、BNE、BGE、BGT 指令，可以透過紅框框裡的 MUX 篩選。

Hardware module analysis:

Hazard_Detection: 負責偵測是否有 branch hazard or load-use hazard 發生。

Forwarding: 負責偵測是否需要 forward 資料以避免 data hazard。

Pipe_Reg: 負責儲存各個 stage 的 data。當遇到 data hazard 時，用 flush 清空資料。當遇到 load-use hazard 時，將 write 設為 0 以 stall and create bubble。此為 pipelined CPU 最重要的一部分。

Adder: 負責處理 immediate 指令及計算記憶體位置。

ALU_Ctrl：根據 opcode 和 ALU_op 來決定要讓 ALU 執行何種運算。

ALU：負責處理邏輯與加減乘運算。

Data_Memory：負責處理記憶體的讀寫。

Decoder：為電的核心，負責處理各種 control signal。

Instruction_Memory：將 address 轉成對應的 instruction。

ProgramCounter：指向要執行指令的 address。

Reg_File：輸出此 instruction 需要用到的 register 的資料。

Shift_Left_Two_32：將輸入的值左移 2 位。

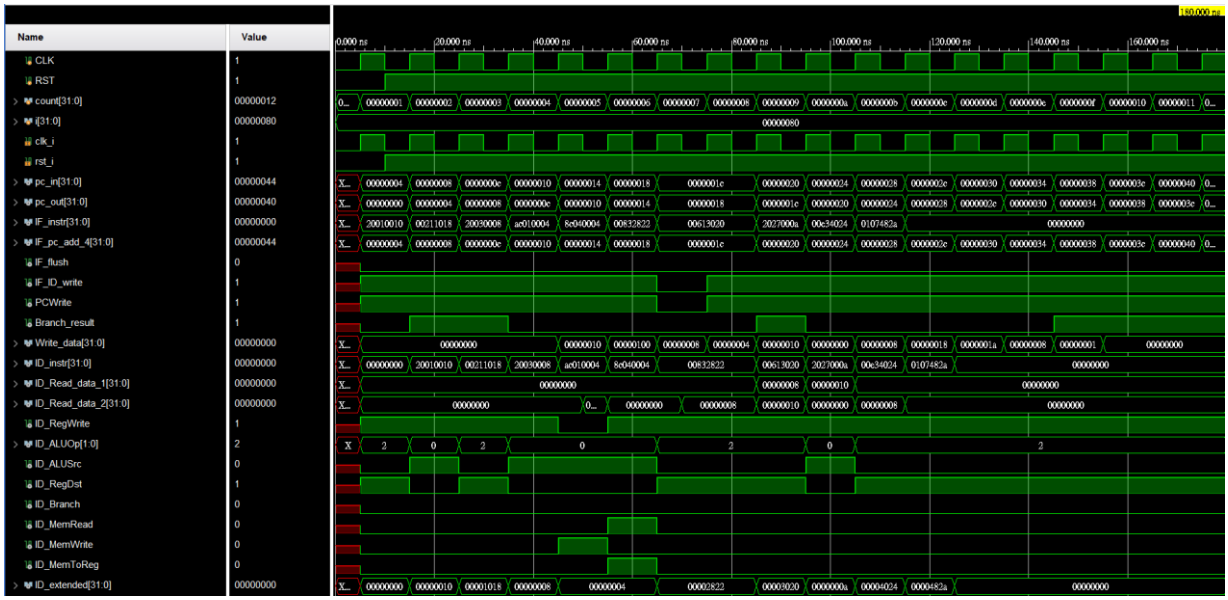
Sign_Extend：藉由把 sign bit 延伸到第 17~32 位，將 16bit 的數值延伸到 32bit。

MUX：根據 control signal 選擇要傳送的資料。

Pipe_CPU_1：負責將所有 module 統整再一起，組合成一個 pipelined CPU。

Finished part:

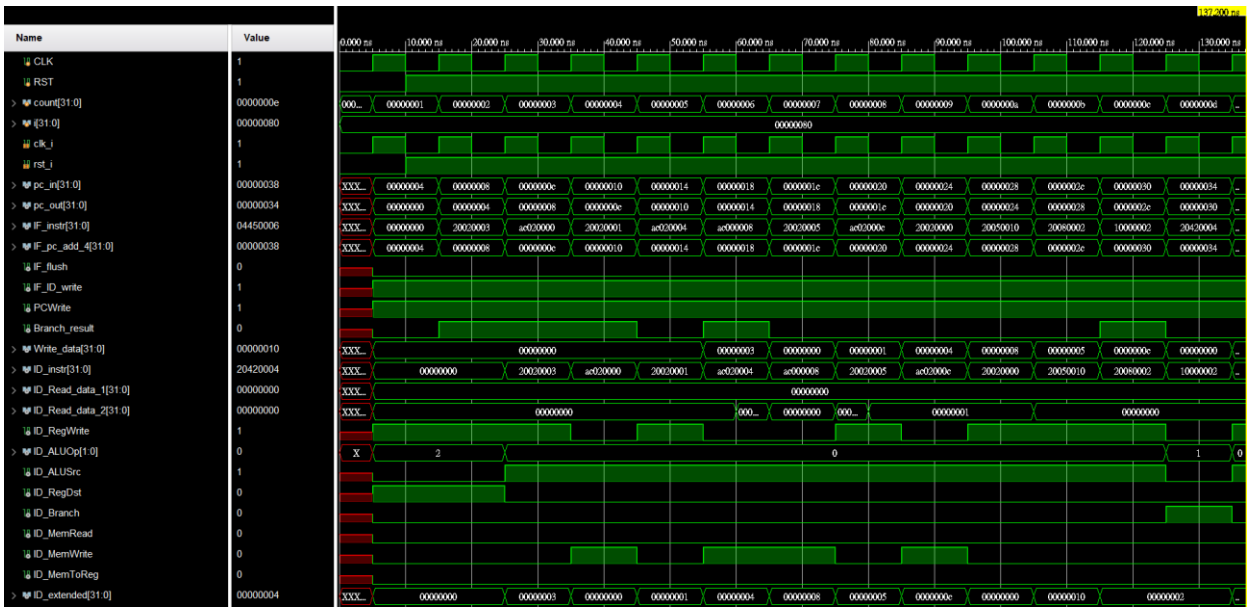
Test 1



```
############################## clk_count = 17##################################
=====================================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 = 70##################################
=====================================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
```

Test 1 和 Test 2 根據各自的 maximum clock count 跑完，結果皆與 MIPS code 的 result 一樣，且和 CO_P5_test_ans 的答案一致。

Problems you met and solutions:

這次 Lab 的 test bench 裡 top module 是叫 Pipe_CPU_1，與上次 Lab 的名稱不一樣 (Pipelined_CPU)，由於我是直接使用上次的檔案去做修改，導致一開始 simulation 的時候跑不出結果。而在 Pipe_CPU_1 中，vivado 的 warning 顯示了 port 的大小不一致，一開始我以為是在宣告 wire 大小時寫錯，可是在多次比對後都找不到錯誤，後來才發現是名稱大小寫不一致，修改完之後終於跑出了正確結果。

Summary:

這次的 Lab 寫起來還滿順利的，要如何處理 forwarding 和偵測 data hazard 講義裡都寫得很清楚，讓我對 pipeline 的運作機制相比 Lab4 有了更完整的理解，並體會到了 pipeline 的強大之處，不用重新排列指令就能避免掉很多 data hazard。