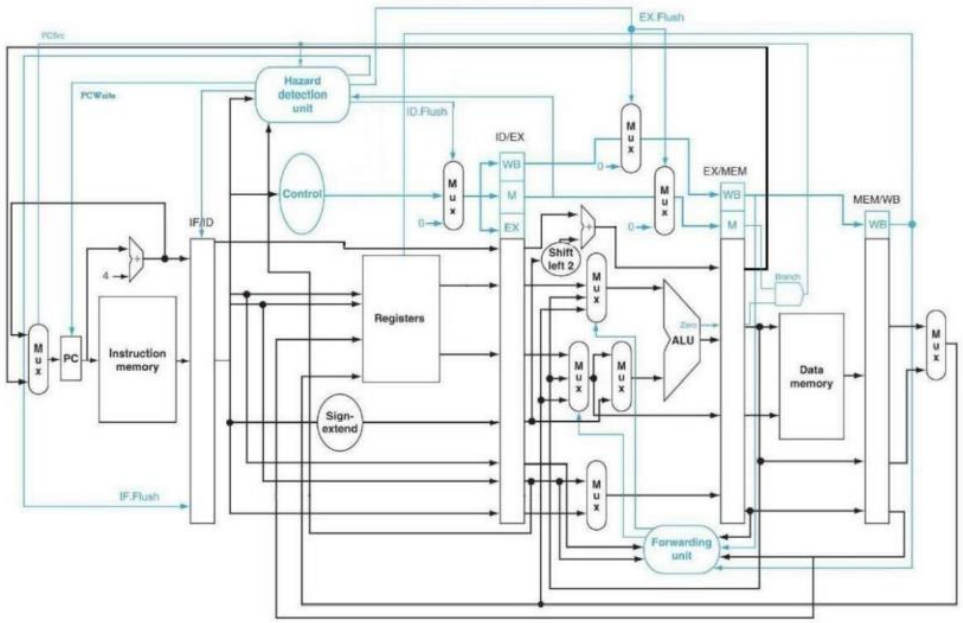
Computer Organization Lab5

Architecture diagrams:



Hardware module analysis:

Forwarding Unit:
判斷在 EX、MEM、WB stage 的指令，有沒有 data dependency
並提供正確的控制訊號，讓目前 EX stage 的指令不需要 stall CPU 太多次，就可以正確拿到需要的資料。

Hazard Unit : 分為 branch hazard 和 load-use hazard
Branch: 預設是 predict sequential execution，當 branch taken 時 (MEM stage)，flush IF/ID、ID/EX、EX/MEM pipe reg 就可以確保錯誤的指令不會被執行完
缺點是一次會 insert 3 個 bubble，降低 CPU 效能

Load-Use: 判斷 ID/EX 中的指令是否為 lw，並判斷 IF/ID 中的指令和 lw 是否有 data dependency，若上述皆成立，stall CPU，不讓 IF/ID、PC 改動，等於是只在 ID/EX insert bubble，而後方的指令還是會照順序執行。

Finished part:

兩組測資皆和助教提供的解答相同

Test1:

```
################################## 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
```

Test2:

```
################################## 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
```

Problems you met and solutions:

Hazard detection 一開始寫 先看有沒有 load-use 再看 branch，但發現 clk=64 時答案是對的，後面程式又繼續跑，導致最後的答案有誤，才發現應該要優先處理 branch hazard，不然 load-use 和 branch hazard 兩個同時發生的時候，就不會 flush 錯誤的指令。

Summary:

這次練習讓我更熟悉 verilog，也更了解 pipeline CPU 的架構。