Lab4

1.

IF/ID

IF/ID :

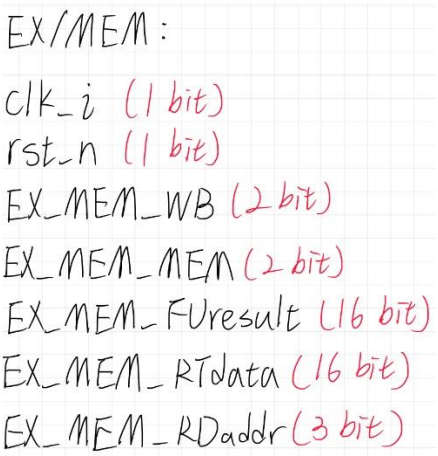
clk_i (1 bit)
rst_n (1 bit)
IF_ID_PCadder1_sum (16 bit)
IF_ID_Instruction (16 bit)
IF_ID_Hold (1 bit)
IF_ID_Flush (1 bit)

ID/EX

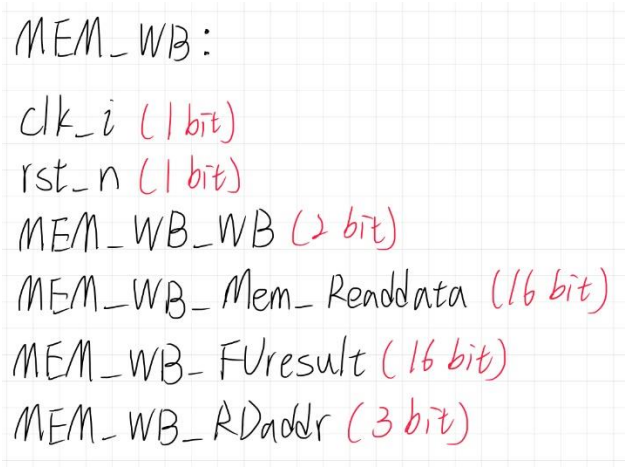
ID/EX :

Clk_i (1 bit)
rst_n (1 bit)
Data_IF_ID_Flush (1 bit)
Branch_IF_ID_Flush (1 bit)
ID_EX_EX (6 bit)
ID_EX_MEM (2 bit)
ID_EX_WB (2 bit)
ID_Ex_Jump_Dst (13 bit)
ID_Ex_PCadder1_sum (16 bit)
ID_Ex_RSdata (16 bit)
ID_Ex_RTdata (16 bit)
ID_Ex_SignExtend (16 bit)
ID_Ex_Zerofilled (16 bit)
ID_Ex_func (4 bit)
ID_Ex_RT_reg (3 bit)
ID_Ex_RS_reg (3 bit)

EX/MEM



MEM/WB



2.

(a)

加入四個 pipeline registers ，重新安排線路，把 RegDst 移到

EX Stage，並把 RDaddr 傳遞到 WB Stage。為了處理 Data 跟 Control

Hazard 加入 Data Hazard Detector 與 Control Hazard Detector。

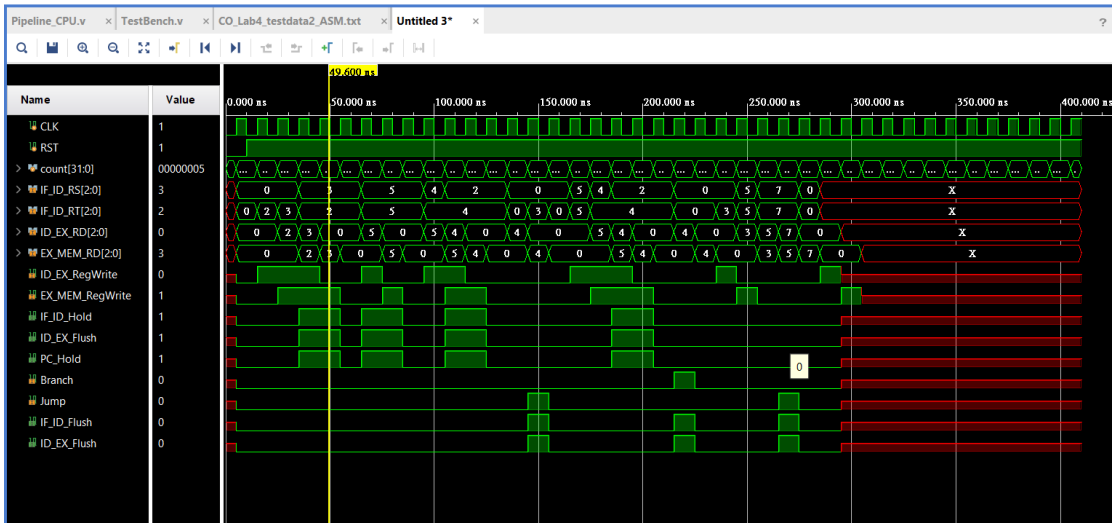
(b)

為了用 stall 處理 Data Hazard，Data Hazard Detector 必須能暫停 PC 與 IF/ID，並清除 ID/EX 產生 stall，因此加入 IF_ID_Hold , PC_Hold, DATA_ ID_EX_Flush 控制線，當偵測到 Data Hazard 發生時，三條線路同時變成 1，產生一個 stall，避開 data-dependency。PC_Hold 會讓 PC_in 從原本的加 2 變成加 0，藉此維持不變，IF_ID_Hold 為 1 時 IF/ID 則不寫入，DATA_ID_EX_Flush 為 1 時則全部寫入 0。

(c)

為了用 stall 處理 Control Hazard，Control Hazard Detector 在 EX Stage 計算出 Branch 結果後，如果結果是跳，或是 jump，則利用 IF_ID_Flush , Branch_ID_EX_Flush 把 IF/ID, ID/EX 全部寫入 0，把錯誤進入 Pipeline 的 instruction 洗掉。

3.



在執行中共產生 4 個 data hazard , 3 個 branch taken,

分別為 r2(instruction:1,3) , r3(instruction:2,3) , r5(instruction:3,4) , r4(instruction:5,6) , instruction 6 (beq) , instruction 7 (jump) , instruction 7(jump)。而透過 detector 偵測到 hazard 後，將 datahazard 訊號線 PC_Hold , IF_ID_Hold , Data_ID_EX_Flush 跟 controlhazard 訊號線 IF_ID_Flush , Branch_ID_EX_Flush 變為 1，產生 stall。

4.

在將 single cycle CPU 轉為 pipeline 的過程中，很常因為打錯線路名稱而產出錯誤訊號，為此偵錯了很久。還有改成 forward 的過程中，必須把原本的線路再次安排，也是歷經一番波折。但是看到最後的結果是正確時，感到相當有成就感。透過這堂課才第一次學習到 verilog，能完成各個 Lab，感到相當開心，收穫頗多。

5.

新增了 Load_Use_Hazard , EX_ForwardingUnit , MEM_ForwardingUnit

刪除原本用 stall 的 Data Hazard Detector，但保持 Control Hazard Detector 。為了 EX forwarding 加入 mux3to1 : forward_A 跟 forward_B，為了 MEM forwarding 加入 mux2to1 : mem_forwarding。而分別用 Forward_A , Forward_B , MEM_Forward 控制線來控制。

6.

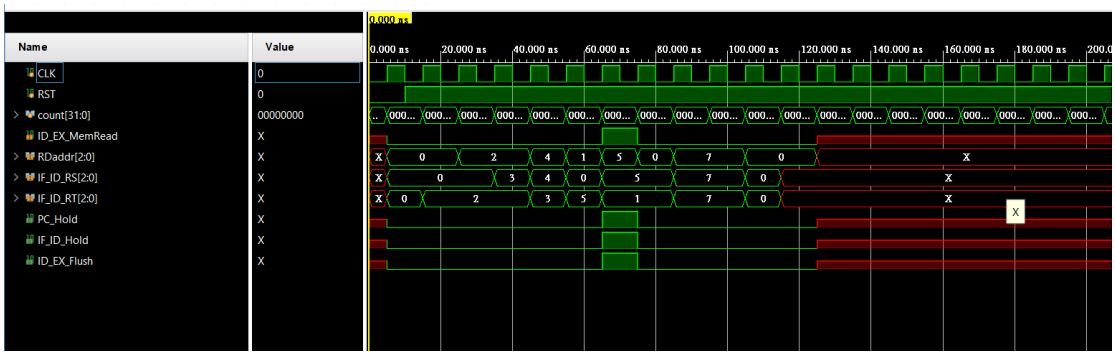
EX_forward:

```
If ( EX_MEM_RegWrite &
      (EX_MEM_RDaddr != 0) &
           (ID_EX_RS_reg == EX_MEM_RDaddr) ) ,
                then { Forward_A = 2 }

If ( ~( EX_MEM_RegWrite & (EX_MEM_RDaddr != 0) &
      (ID_EX_RS_reg == EX_MEM_RDaddr) ) &
         MEM_WB_RegWrite &
         (MEM_WB_RDaddr != 0) &
            (ID_EX_RS_reg == MEM_WB_RDaddr) ) ,
                then{ Forward_A = 1}

If ( EX_MEM_RegWrite &
       (EX_MEM_RDaddr != 0) &
          (ID_EX_RT_reg == EX_MEM_RDaddr) ) ,
                then { Forward_B = 2 }

If ( ~( EX_MEM_RegWrite & (EX_MEM_RDaddr != 0) &
        (ID_EX_RT_reg == EX_MEM_RDaddr) ) &
             MEM_WB_RegWrite &
                (MEM_WB_RDaddr != 0) &
                   (ID_EX_RT_reg == MEM_WB_RDaddr) ) ,
                         then{ Forward_B =1 }
```

MEM_forward:

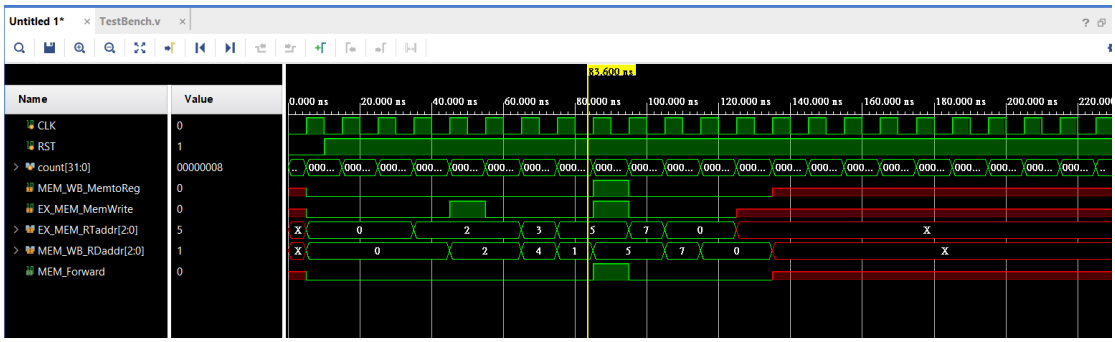
```
If ( MEM_WB_MemtoReg &
       EX_MEM_MemWrite &
        ( EX_MEM_RTaddr == MEM_WB_RDaddr)) ,
             then{ MEM_forward = 1 }
```

7.



結果正確，load_use_detetector 在 clock cycle 7 時，偵測到前一個 instruction 為 lw，且 r5 有 data dependency，故把三條訊號線 (PC_Hold , IF_ID_Flush , ID_EX_Flush)輸入 1，使其產生一個 stall。

8.



正確，在前面沒有被 load_use_detector 插入 stall 的情況下，MEM_forwardongUnit 偵測前一個指令為 lw，而當前指令為 sw，且 r5 有 data dependency，故把訊號線 MEM_Forward 輸入 1，將 lw 讀出的 data 傳入 MEM 的 WriteData 中。