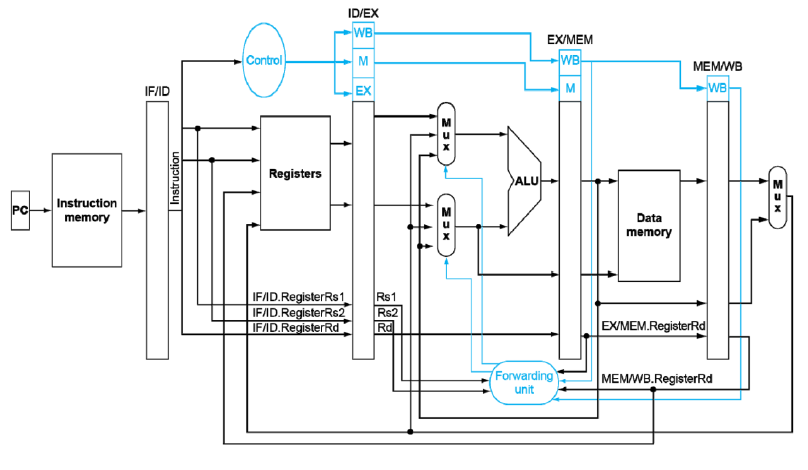
**Computer Organization**

**Architecture diagram：**

****

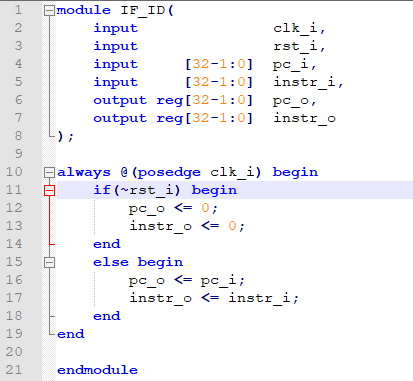
**Detailed description of the implementation：**

Lab 5 是利用 4 個 pipeline registers 把 Lab 4 時所做的Single-Cycle-CPU 切割成 5 stages (IF, ID, EX, MEM, WB)，然後再加上一個 Forwarding unit 來處理 Data Hazards (前面指令的Rd影響到後面指令的Rs1或Rs2) 的問題：

**1.Pipeline registers** (IF\_ID.v, ID\_EX.v, EX\_MEM.v, MEM\_WB.v):

資料在進入下一個 stage 前，需要先進入 pipeline register 進行緩衝，將這個stage的資料暫存起來，以等待下一個 stage 的資料成功離開。因此，我們只要把需要傳到下一個 stage 的資料 input 進 pipeline register ， 如果沒有reset的話，就在下一個 clock output 出來。

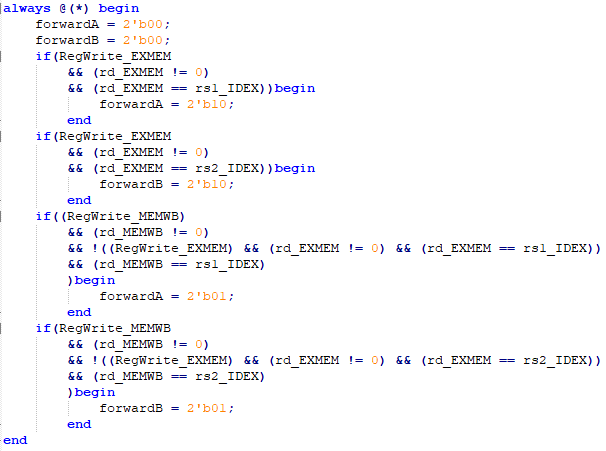
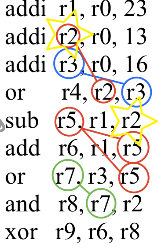
以IF\_ID舉例：



**2.Forwarding unit** (forwarding.v)：處理 data hazards 的問題

例如：在執行 or r4, r2, r3 時，在前面的兩行的 addi 都還沒把 r2, r3 從 WB 送回 register ，因此需要事先判斷 or 的 Rs1 與 Rs2 是否有與前面兩的指令的 Rd 衝突 。如果有衝突的話，就提前在MEM stage 把前面兩個指令的結果送給 or 的 Rs1 與 Rs2。

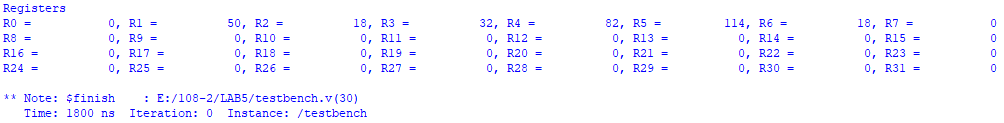
除此之外，黃色星號的部分：當 addi r2, r0, 13 的指令跑到 RB stage 的時候，sub r5, r1, r2 指令跑到了 ID stage ，此時這個 ID stage 需要同時把 addi 的結果存進 r2, 又要把 r2 的值丟給下一個 stage 進行 sub ，此時 sub 的 Src2 沒辦法使用到 addi 的結果(也就是 addi 的結果來不及跑回 r2 給 sub 使用)，解決方法在下面。



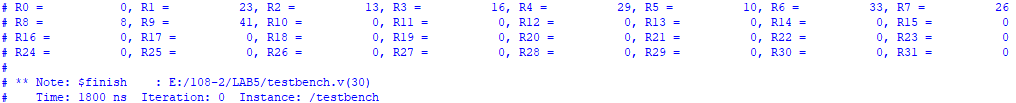
**3.Pipeline CPU** (Pipeline\_CPU.v)：我們把Lab4中原本的所有訊號&資料的接線，全部分成5個stage，所以verilog中的變數量會變得分長多，想變數命名的時候花費了不少功夫。

**Implementation results：**

**Test\_data\_1**

****

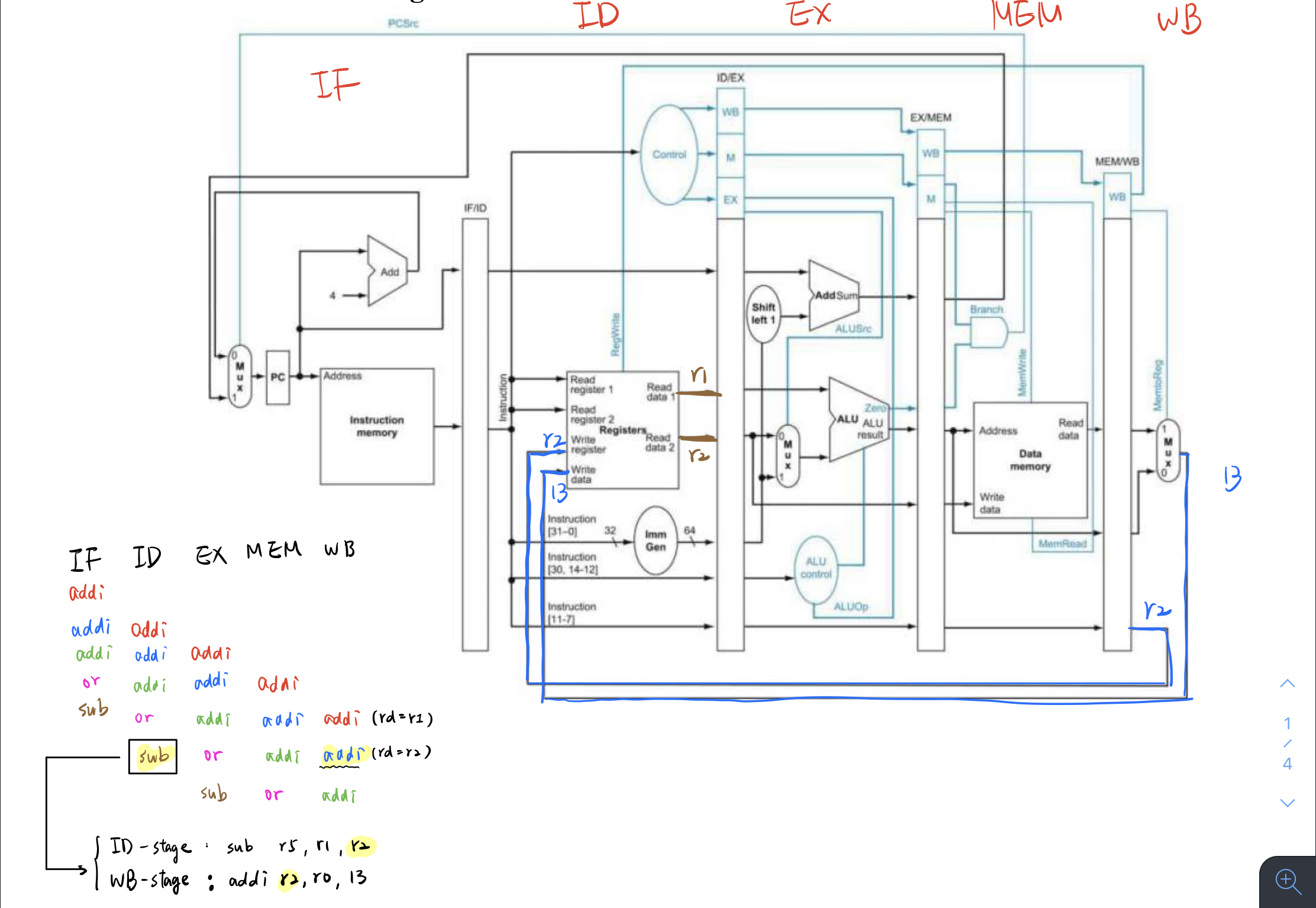
**Test\_data\_2**

****

**Problems encountered and solutions：**

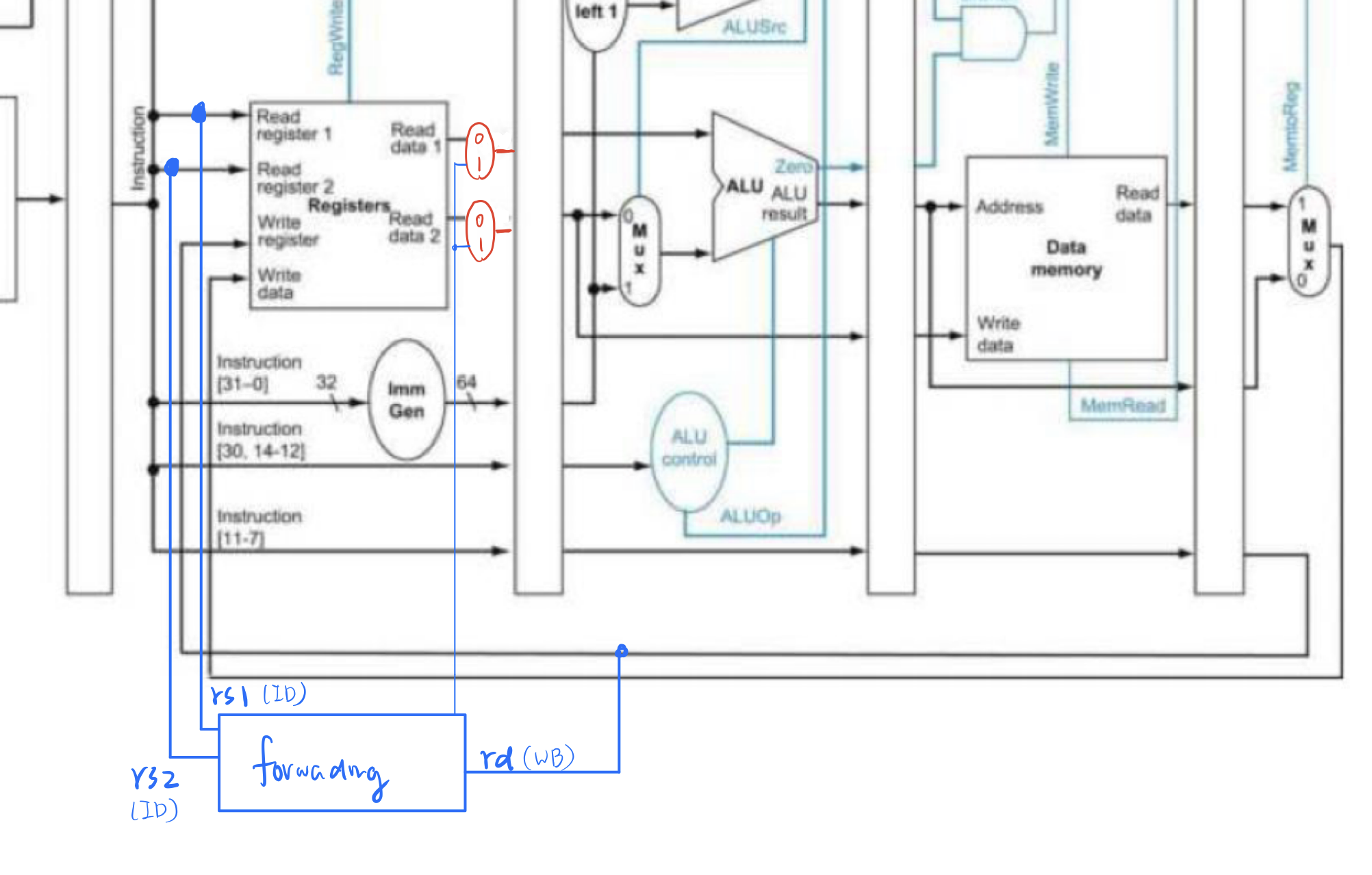
1. 當WB的rd == ID的rs1或rs2時，ID會讀到舊的值。

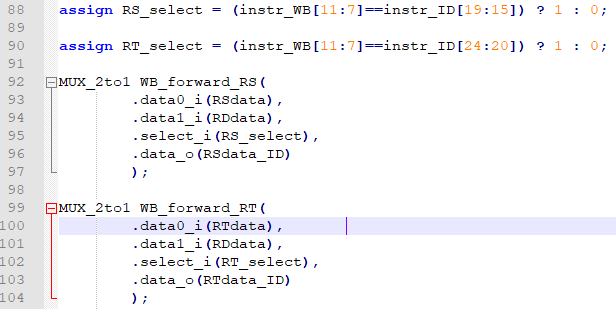
WB-stage寫回的rd=r2，且ID-stage的rs2=r2，因為Register File只有Positive Edge trigger，但是WB寫回r2會有延遲，所以ID-stage讀取r2的時候，r2還是舊的值所以這邊也要有一個forwarding unit。



解決方法：

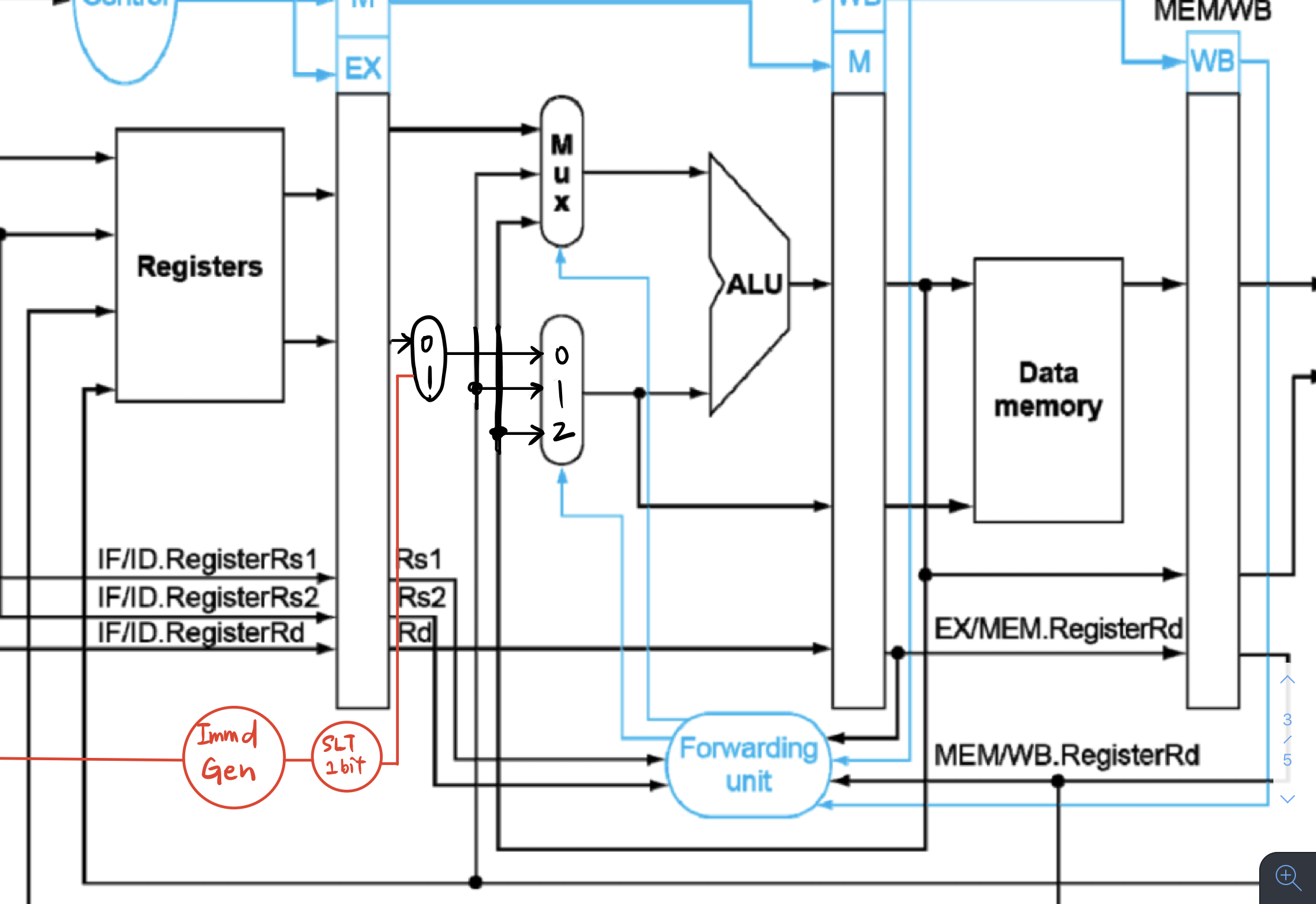
我們參考討論區同學提供的想法，製作一個類似WB的forwarding unit，當WB-stage的rd等於ID-stage的rs1或rs2時，rs1或rs2會取WB寫回的值。





2. ALU的Source2在其實會是4選1，我們原本想把immediate放入forwarding的Mux中，但這麼做讓forwarding unit的判斷更複雜。

所以我們決定把forwarding的Mux維持3個input，也把Lab4的ALU\_src2的Mux留著，這樣就不會讓forwarding unit變複雜。



**Comment:：**

這次 Lab 基本上都是沿用前次 Lab 的 CPU ，因此我們在這次 Lab 所花費的時間相對較少。而本次 Lab 的主要難度在於 data hazards 的處理，這方面我們從討論區上面得到了非常大的幫助，我認為討論區的建立大大的幫助我解決在這門課的 Lab 中所遇到的問題，因此希望這樣的機制可以延續下去。