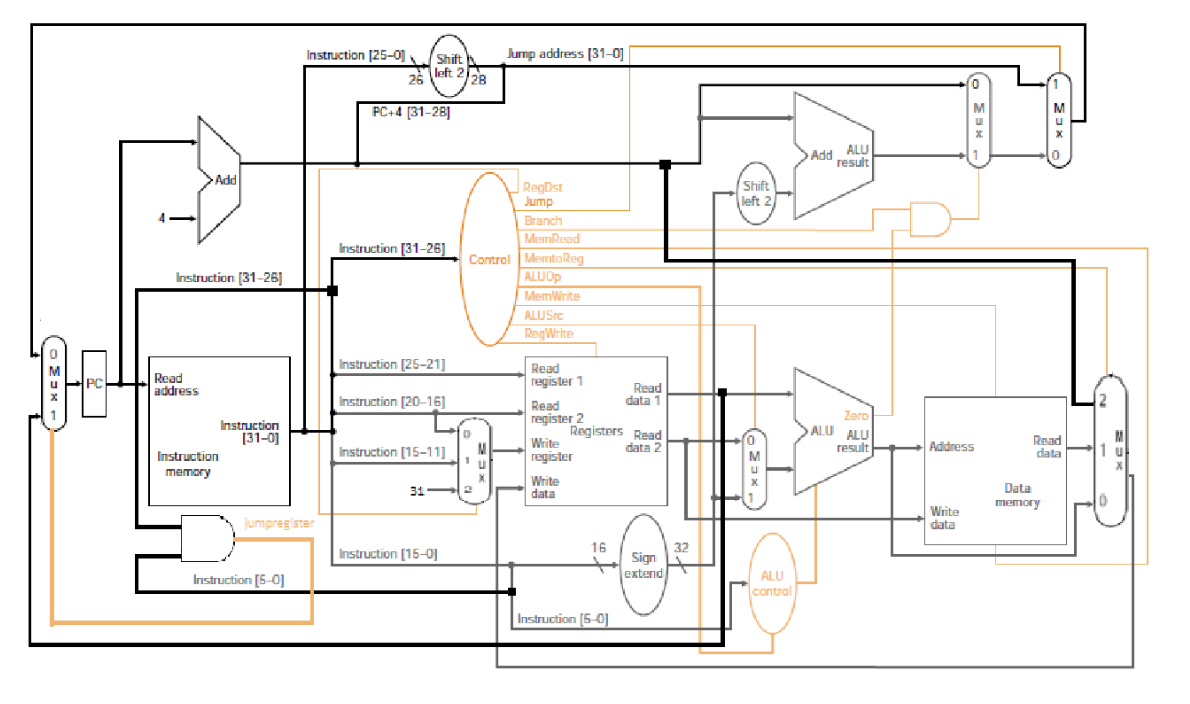
**Computer Organization Lab3**

**Name:單宇晟**

**ID:109550087**

**Architecture diagrams:**

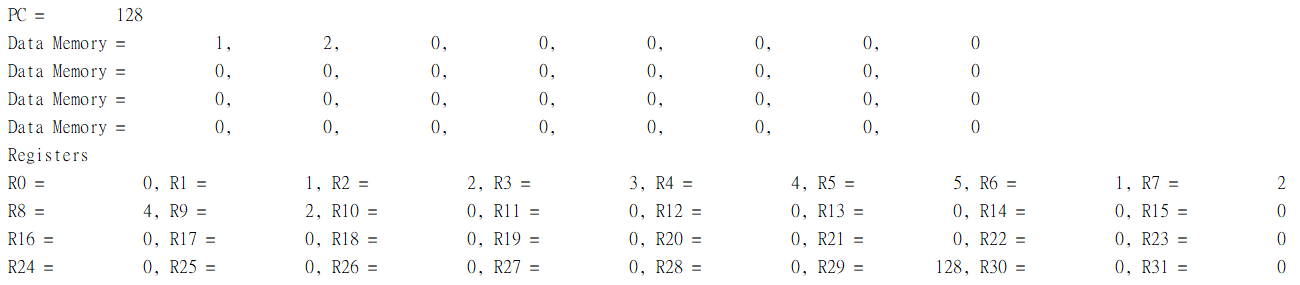
**Hardware module analysis:**

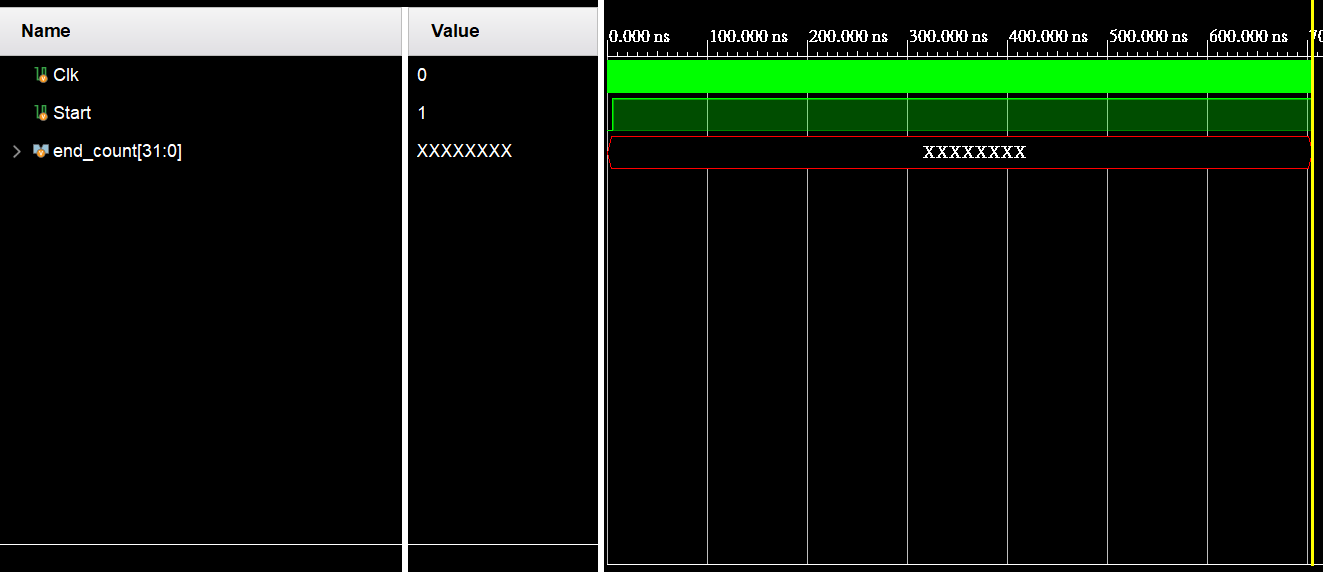
這次diagram的實作方法，主要可以參考simple\_single\_CPU.v。首先，會用PC來提供指令位址，之後會計算出下個指令的位址(PC Add 4)，同時instruction memory會給出指令，並且將各位元分別傳送到指定的地方做運算，並且decoder也會給出控制訊號，這次我用到了4個MUX2to1，分別判斷jr, R/I-format, jump, branch的情況，要注意的是因為這次並沒有實作其他的branchtype(PDF的圖上有)，所以右上方的MUX判斷的時候和上次一樣用branch & zero就好。

至於lw和sw，他們運作的方式其實有點像，只是lw是對記憶體做讀取、sw是儲存，此外，sw也不會把記憶體的值寫回暫存器。

雖然singlecycle的結構較為簡單，但缺點是相比multicycle，singlecycle的clk週期必須是相同長度，而此長度會由電路裡最長的路徑來決定，因此效率比較不好，且我們無法在每個指令中使用不同數目的clk。

**Finished part:**

****

****

**CO\_P3\_test\_data1:**

addi r1,r0,1 r1=1

addi r2,r0,2 r2=2

addi r3,r0,3 r3=3

addi r4,r0,4 r4=4

addi r5,r0,5 r5=5

這部分就是直觀的addi，因此r1, r2, r3, r4, r5的值會分別是1, 2, 3, 4, 5

jump j

addi r1,r0,31 r1=31

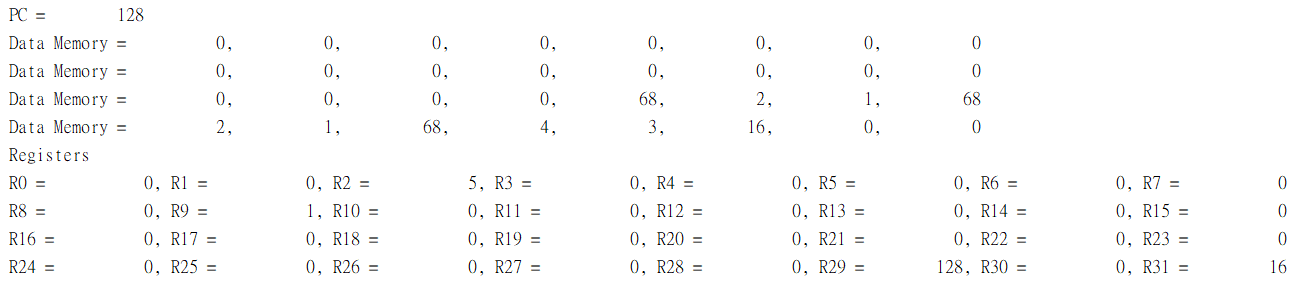
addi r2,r0,32 r2=32

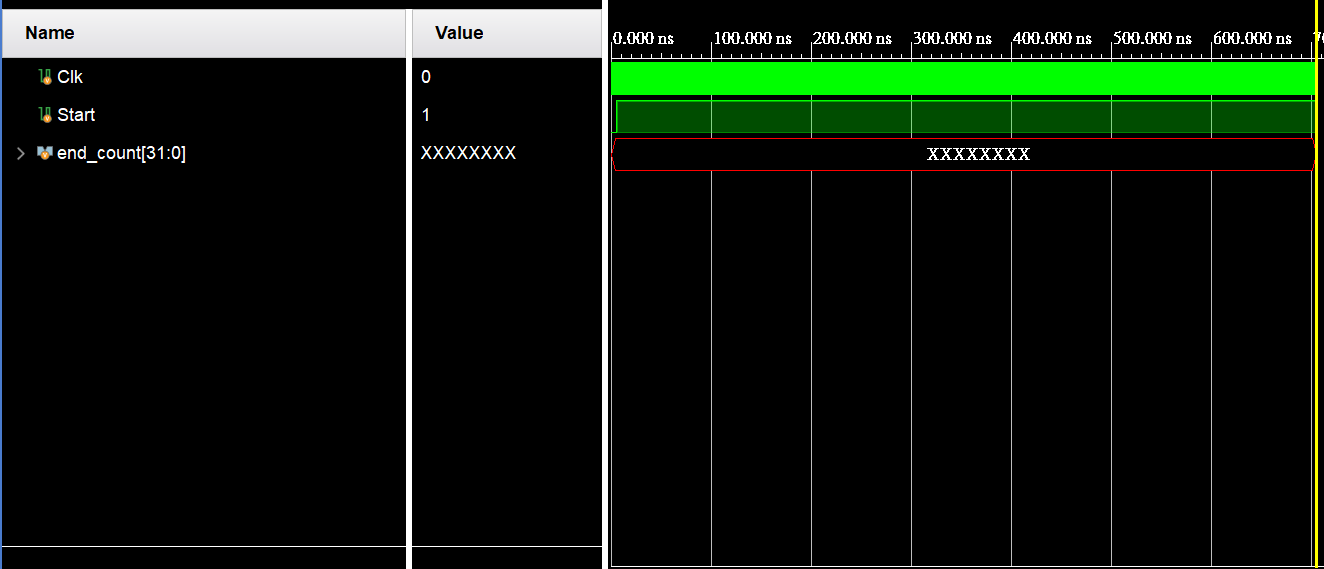
這裡因為jump執行了，所以jump下面的兩行會被跳過，直接去執行j的部分

j: …

這裡主要測試lw, sw的功能

**CO\_P3\_test\_data2:**

****

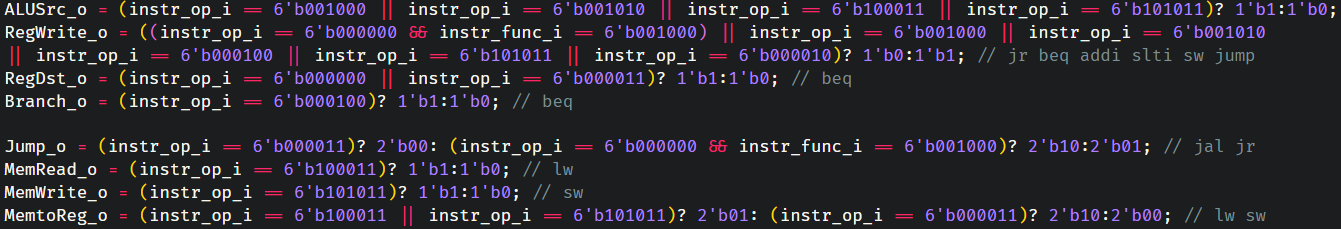
****

Test2的程式比較難按照步驟解釋，相比於test1他多測試了jal 跟jr的功能是否正常運行，而依照程式邏輯，最後出來的R2就會是5。

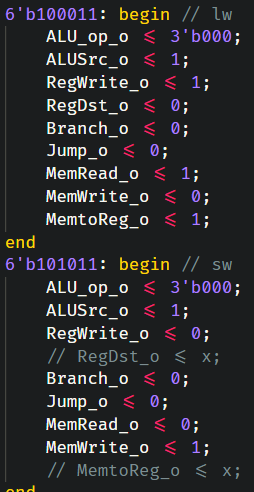
**Problems you met and solutions:**

這次lab相比於上次lab，又多了一些function，所以在decoder裡面又變得更複雜，也更多不同case要處理，我原本的寫法是用if else去寫，來assign

Jump\_o, MemRead\_o, MemWrite\_o, MemtoReg\_o這些值，像是這樣:



不過後來發現這樣如果不小心漏掉某個case或是打錯就會很難找到bug，所以就改成了現在這種寫法:



把每個情況都分開寫，雖然這樣會讓我的code變很冗長，但卻比較方便debug。另外一個問題倒是困擾我很久，就是pdf上給的diagram只是參考，並不像是上次lab那樣照著牽線就好，加上助教給的模板也沒有改過，一些要額外用到module、或是要用到mux3to1的地方都沒有直接標明出來，要我們自己判斷，因此在我花在simple\_single\_cpu上的時間非常多，幸好最後我並沒有卡住，還是有順利的把每條線、每個module弄好，按時的完成了這次的lab。

**Summary:**

這次lab主要是在上次的架構下加入一些新功能(lw, sw, jump, jal, jr)，所以大致上的架構其實都沒有變，實際上要改的檔案也比上次少很多。不過實際上還是花了我不少時間，正如上面提到的，這次的實作並不能完全照PDF上的圖做，要自己額外確認每條線的去向，同時還要知道那些額外的參數(Jump\_o, MemRead\_o, MemWrite\_o, MemtoReg\_o)在幹嘛以及他們的重要性，只要一個地方assign錯就可能全部都跑不出來，而這次更多了DataMemory這個module。經過了這兩次lab，我也終於完成了一個功能「稍微」完善的single cycle CPU，之後應該就要做pipeline CPU了，希望我也能夠順利地做出來。