國立臺灣科技大學

電機工程系



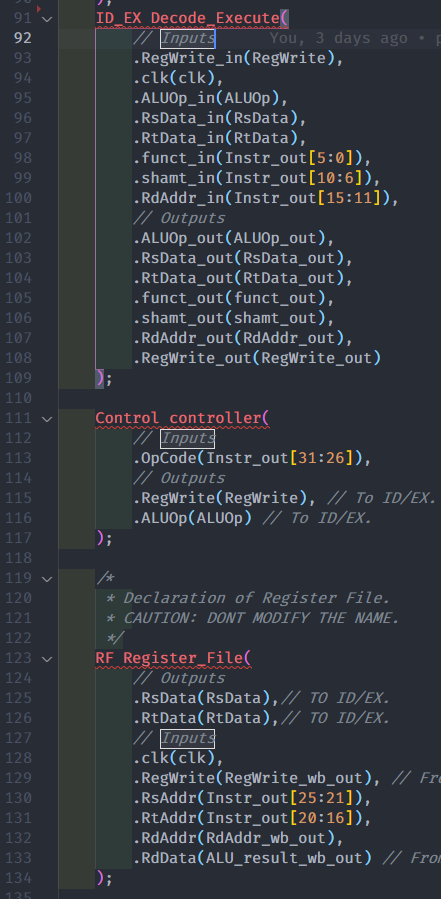
計算機組織

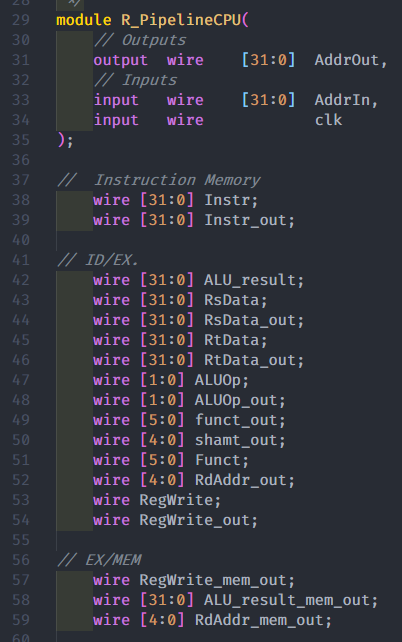
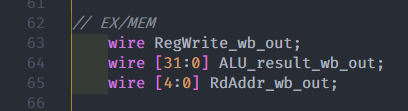
作業報告

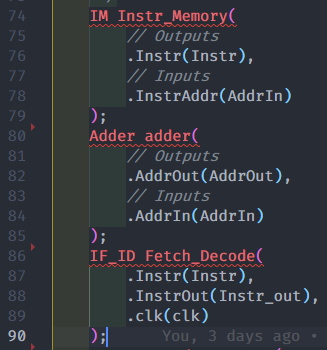
PA3

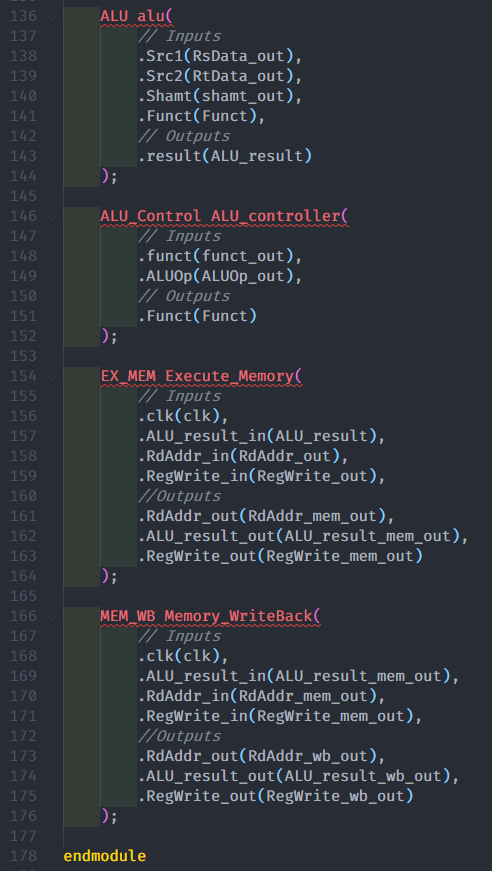
四電機三乙 | B10707128 | 劉杰閎

* **Part I**

1. **R\_PipelineCPU.v**







tb\_R\_formatCPU與題目所提供之檔案相同，因篇幅限制而不另行截圖。這個R\_PipelineCPU是將所有模組集大成之結果。我在裡面宣告了一些wire讓他去相互連接。而下圖為RF.out之輸出結果。右圖是經過執行後輸出出來的結果，利用notepad++之compare工具，而對比於題目提供之檔案(左圖)可以發現完全相同，因此R\_PipelineCPU到此順利做完。



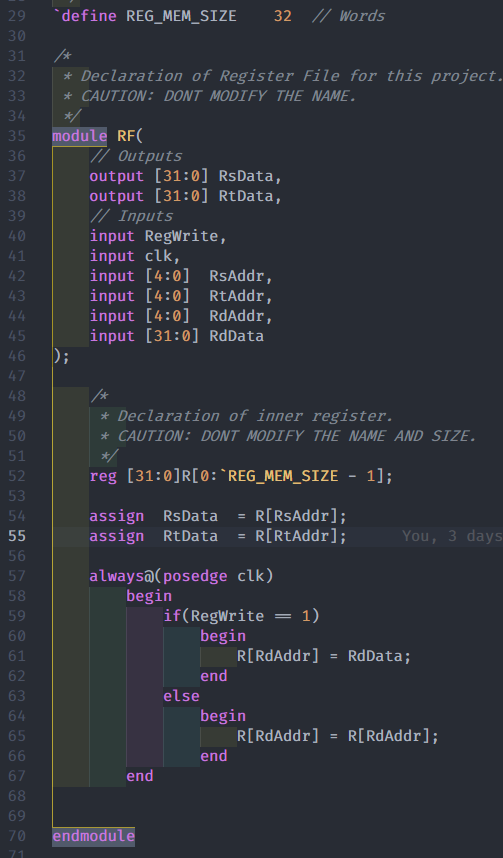
1. **Instruction Memory**

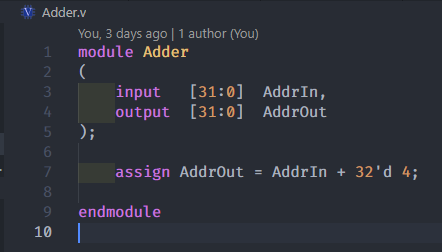
IM這個module其實很簡單，只要輸入Instruction Address，接著我就到該位置去抓Instruction，因為這個系統是BIG-ENDIAN，因此我抓的順序就會是如我程式碼的方式去抓{0,1,2,3}.



1. **Register File**

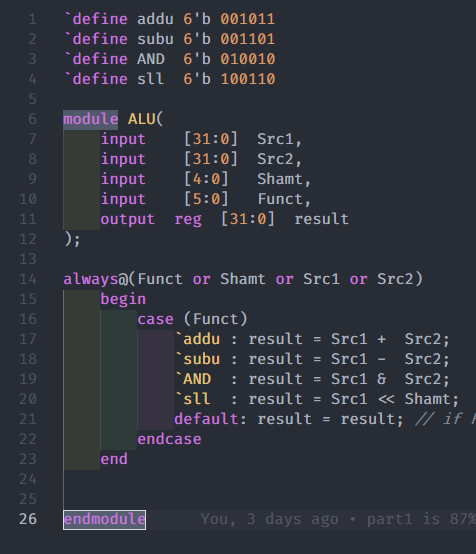
RM這個module其實不難，只要判斷RegWrite是否為1，來決定是否可以將值寫入Reg，那其他部分就是到Register的address去抓值去輸出。我將RsData與RtData使用assign而非放在always裡面是因為我發現放在裡面會等到正緣觸發才把值送入ALU，那這樣的話就無法達到我們想要的效果了。

1. **Adder**

Adder所做的事情非常簡單，因為在R-type的AdderOut僅僅需要能夠將AddrIn+4，因此我就只做了加4的動作，然後輸出。(我的加法是unsigned的加法，因為Address沒有負的。) 

1. **ALU**

ALU 主要是用來做Src1 and Src2的運算，由輸入的Funct來決定要做甚麼工作。



1. **ALU\_Control**

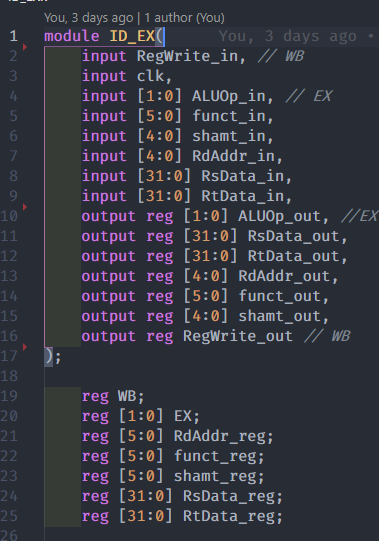
ALU\_Control主要是用來控制ALU工作與否，及將instr的指令轉為ALU懂得function code。比較特別的是在上一個作業的input\_addu與ALU所要求的funct code不同，而這次作業的code是相同的，導致我在debug時花了一些時間才發現。

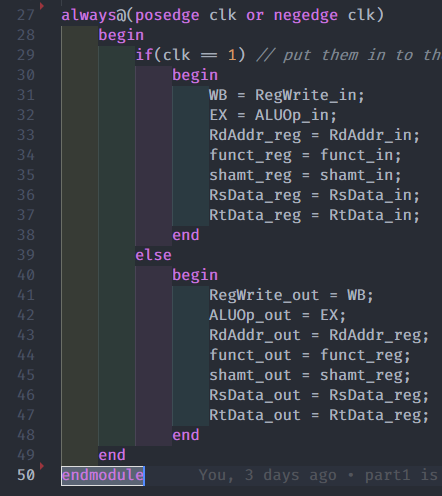
1. **Control**

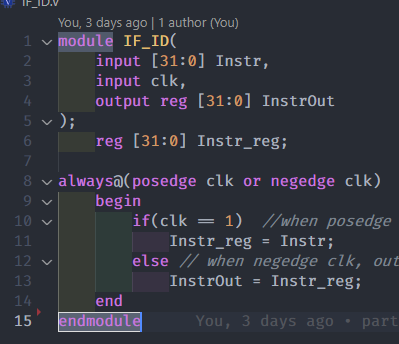
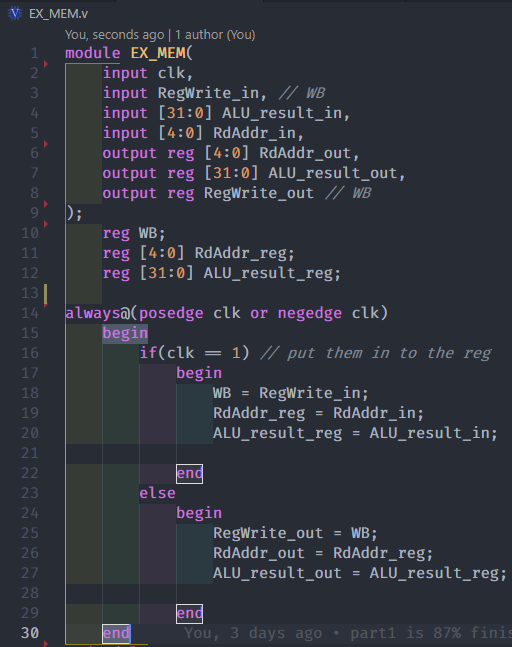
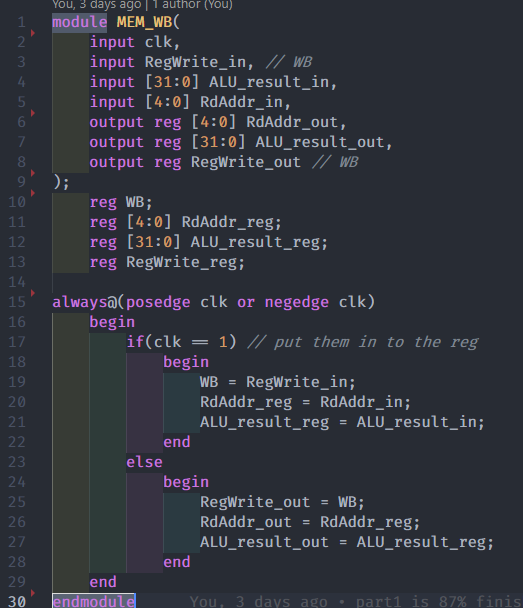
Control這個module在R-type沒什麼太複雜的工作，只要依照Opcode的要求，來確認會不會把值寫入Reg裡面，來決定是否要送RegWrite的訊號。而因為R-type的所有指令都會使用到ALU，因此我們ALUOp只要是對的Opcode，我們一律送2`b10.



1. **IF/ID, ID/EX, EX/MEM, MEM/WB**



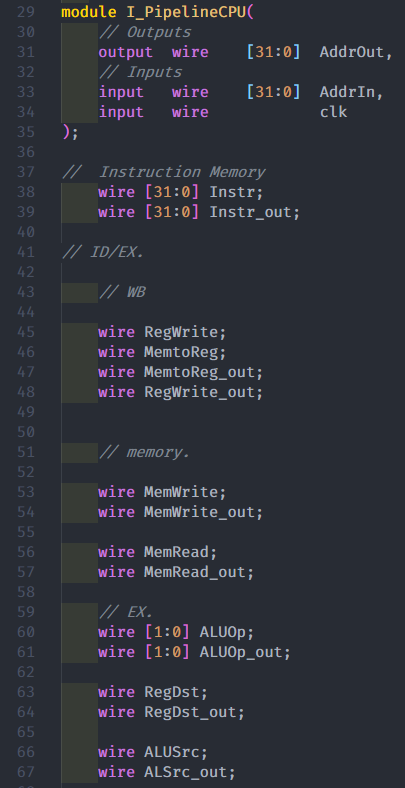
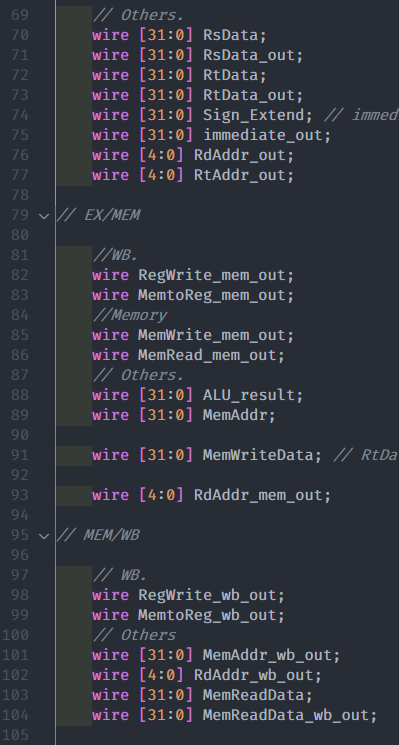
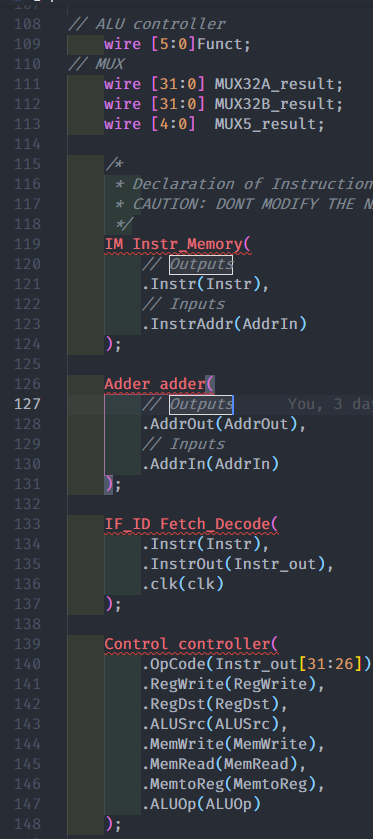


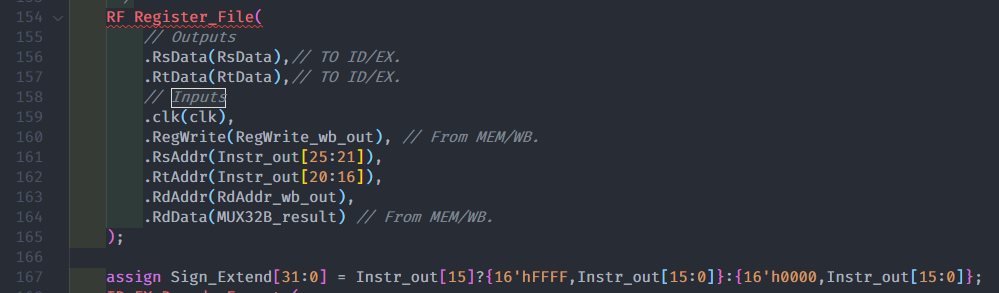
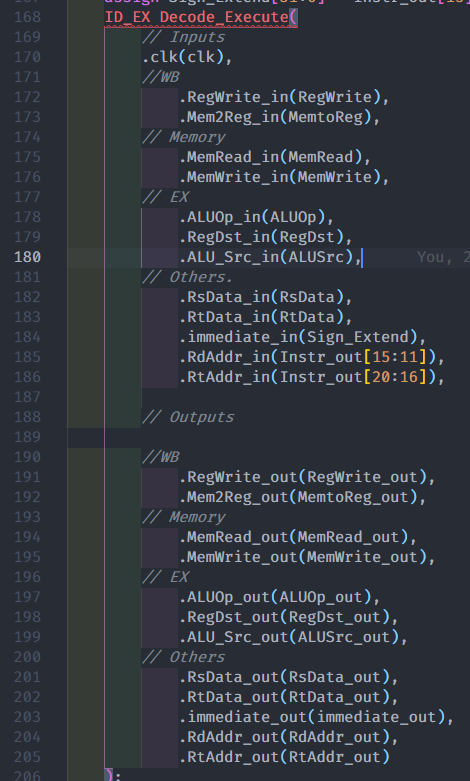
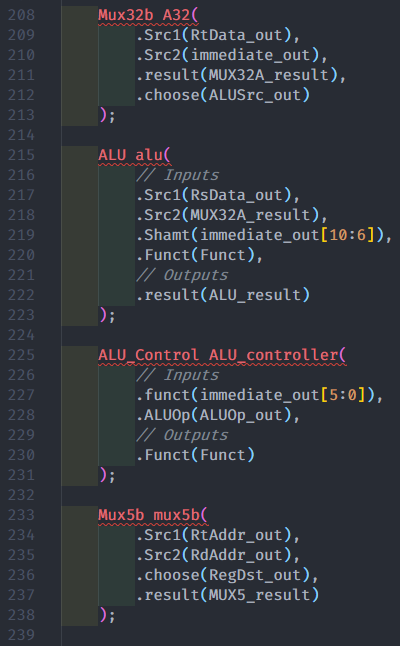
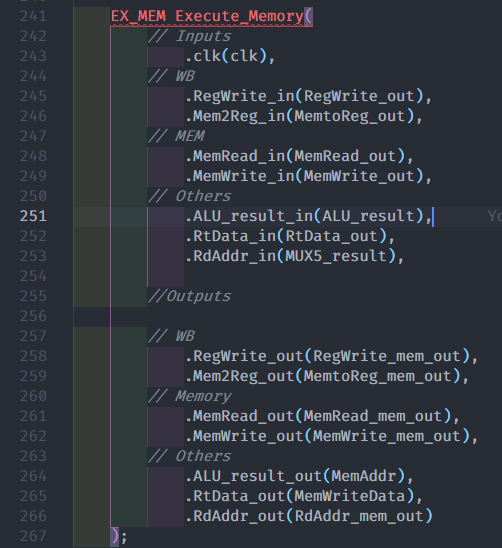


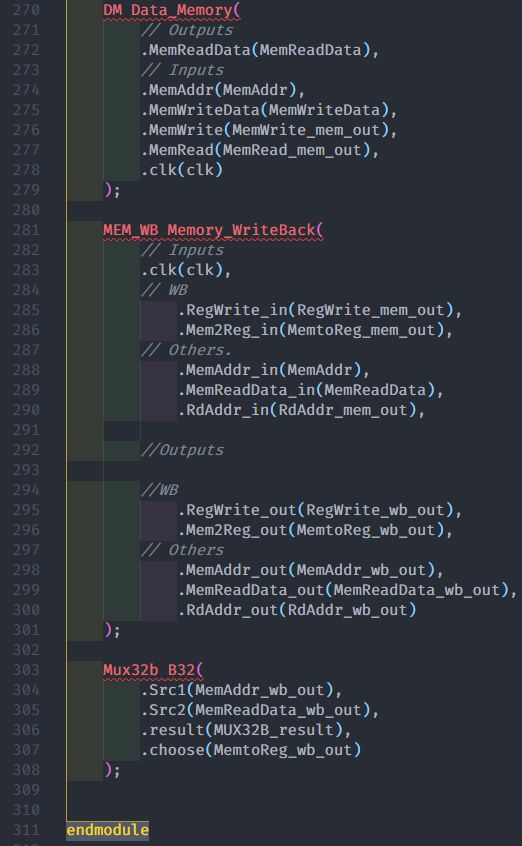
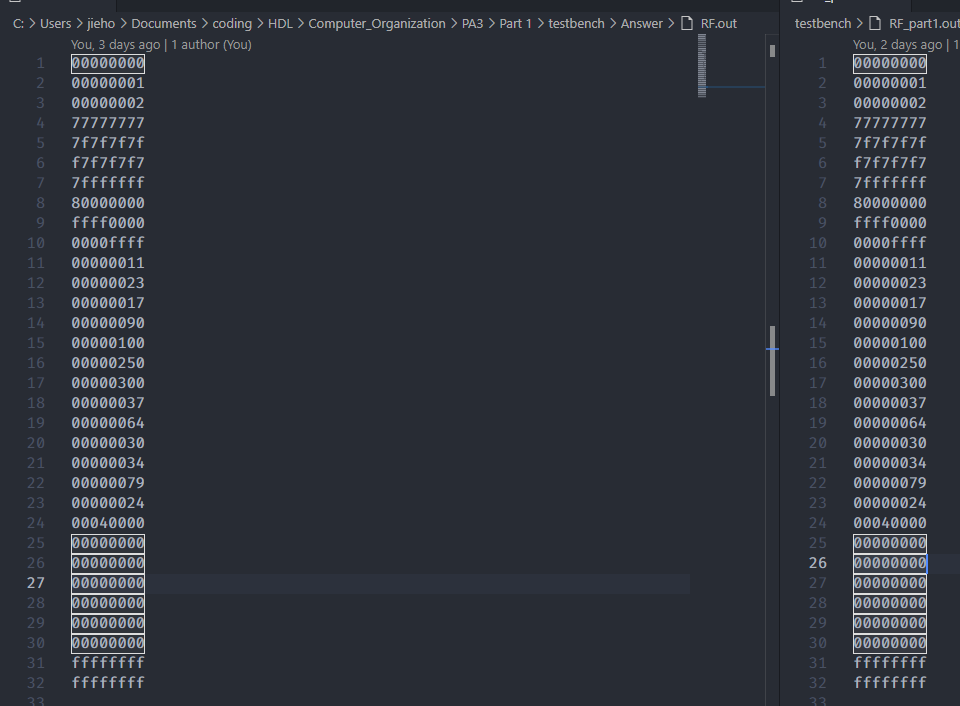
這幾個module都是Pipeline register.因為性質相同，且無任何特別的，因此在這個地方，就將他們放上來，所有的pipeline register都是正緣觸發，將值傳入register,當負緣的時候才更新輸出的值。確保不會影響到下一刻的output.

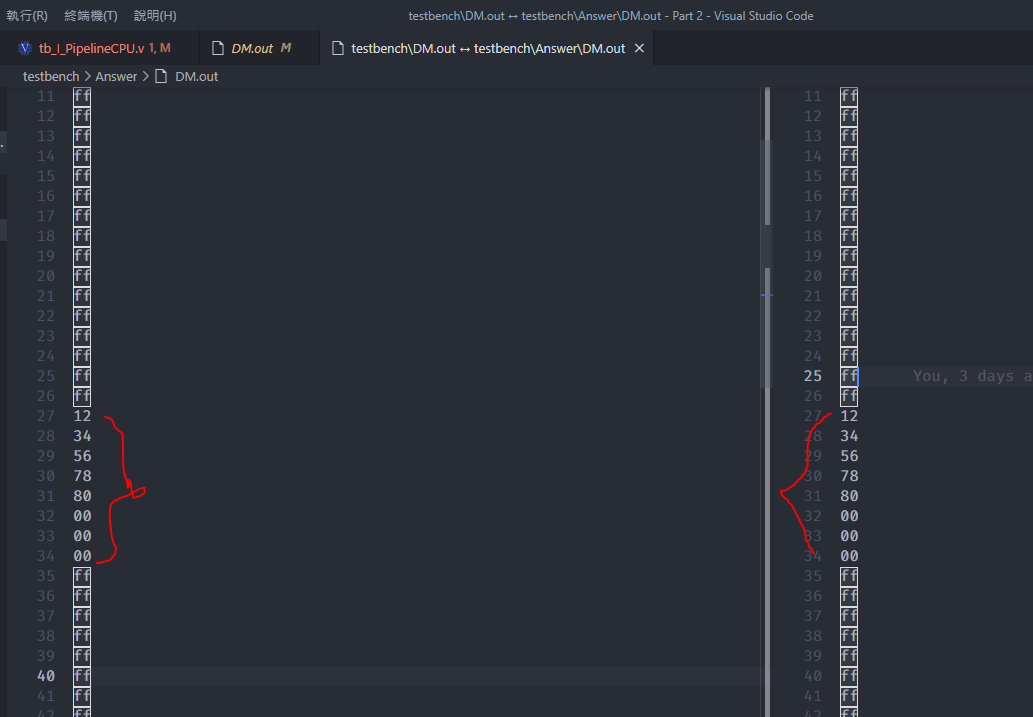
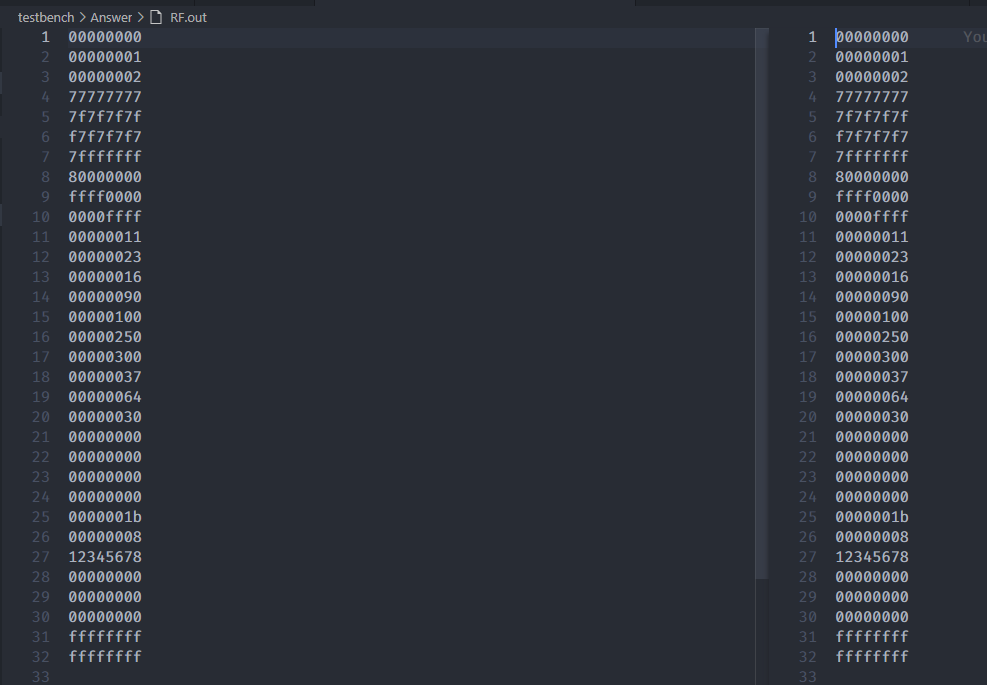
* **Part II.**

1. **I\_ PipelineCPU**





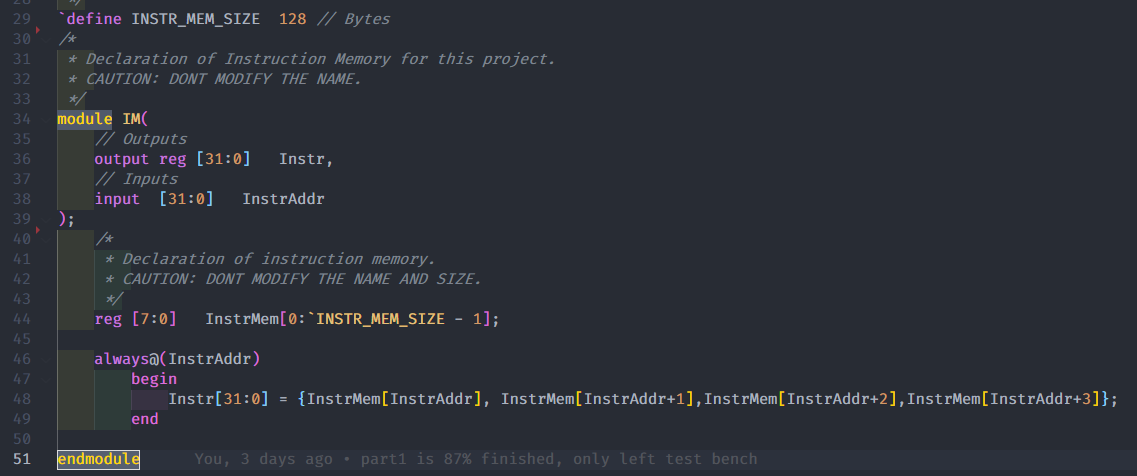


I\_PipelineCPU，testbench因篇幅限制而不特別截圖。因此在這個地方，只放上DM.out與RF.out來確認模擬結果是正確的。而下方左圖是DM.out經過執行後輸出出來的結果，而對比於題目提供之檔案(右圖)可以發現完全相同，除了27~34以外，其他結果皆為FF。上方右圖RF.out是經過執行後輸出出來的結果，而對比於題目提供之檔案(左圖)可以發現完全相同。

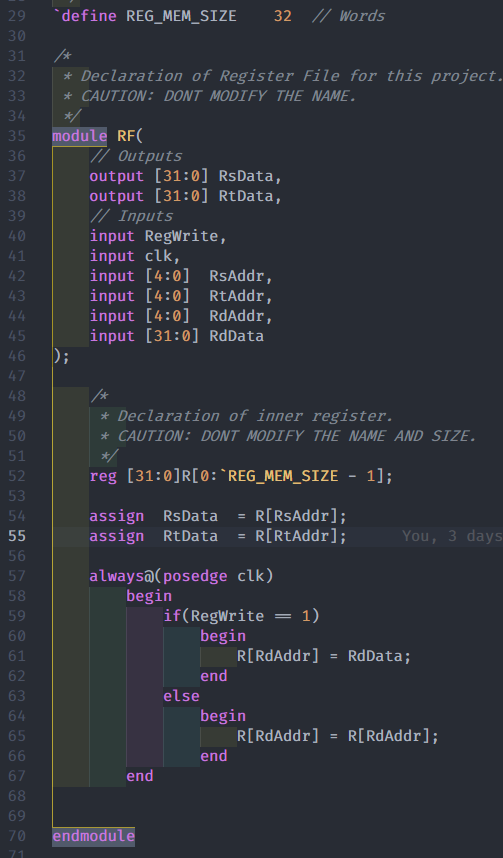
接著實測R\_format在I-type CPU上模擬之結果。可以看到完全相等，因此到這裡可以確認成功。

1. **Instruction Memory**

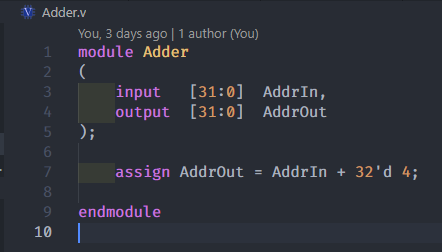
IM這個module其實很簡單，只要輸入Instruction Address，接著我就到該位置去抓Instruction，因為這個系統是BIG-ENDIAN，因此我抓的順序就會是如我程式碼的方式去抓{0,1,2,3}.



1. **Register File**

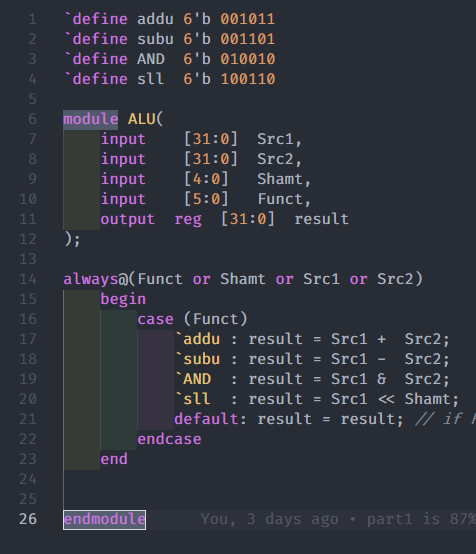
RM這個module其實不難，只要判斷RegWrite是否為1，來決定是否可以將值寫入Reg，那其他部分就是到Register的address去抓值去輸出。我將RsData與RtData使用assign而非放在always裡面是因為我發現放在裡面會等到正緣觸發才把值送入ALU，那這樣的話就無法達到我們想要的效果了。

1. **Adder**

Adder所做的事情非常簡單，因為在R-type的AdderOut僅僅需要能夠將AddrIn+4，因此我就只做了加4的動作，然後輸出。(我的加法是unsigned的加法，因為Address沒有負的。) 

1. **ALU**

ALU 主要是用來做Src1 and Src2的運算，由輸入的Funct來決定要做甚麼工作。



1. **ALU\_Control**

ALU\_Control主要是用來控制ALU工作與否，及將instr的指令轉為ALU懂得function code。比較特別的是在上一個作業的input\_addu與ALU所要求的funct code不同，而這次作業的code是相同的，導致我在debug時花了一些時間才發現。



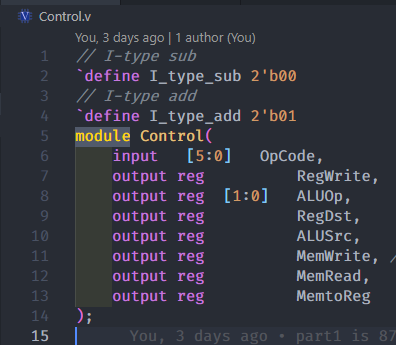
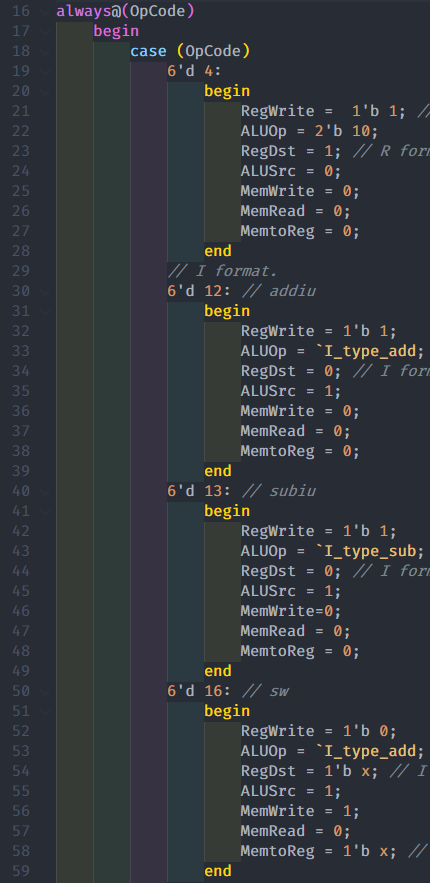
1. **Data Memory**

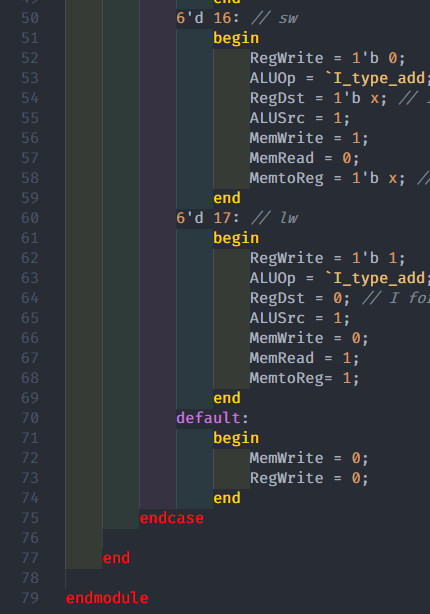
DM這個module其實也不難，只要判斷MemWrite跟MemRead是否為1，來決定是否可以將將值寫入Memory或是把Memory的值給讀出來。我將MemReadData使用assign而非放在always裡面，與RF的原因相同。我發現放在裡面會等到下個正緣觸發才把值送出，那這樣的話就無法達到我們想要的效果了。



1. **Control**

Control這個module在整個CPU裡面是一個非常重要的角色。他掌管整顆CPU現在要做甚麼，不要做甚麼。雖然他極其重要，但是其實沒有甚麼太複雜的工作，只要依照Opcode的要求，來決定是否要送各種訊號。而因為R-type的所有指令都會使用到ALU，因此我們ALUOp只要是對的Opcode，我們一律送2`b10.而對於I-type指令來說，只會有加法跟減法，因此除了subiu以外(2`b00)，其他的ALUOP都為2`b01，剩下的是J-type指令，因為branch用到的是減法，因此ALUOP為0。

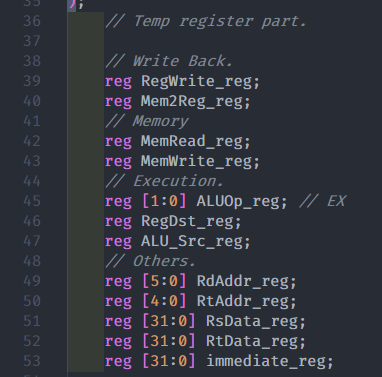


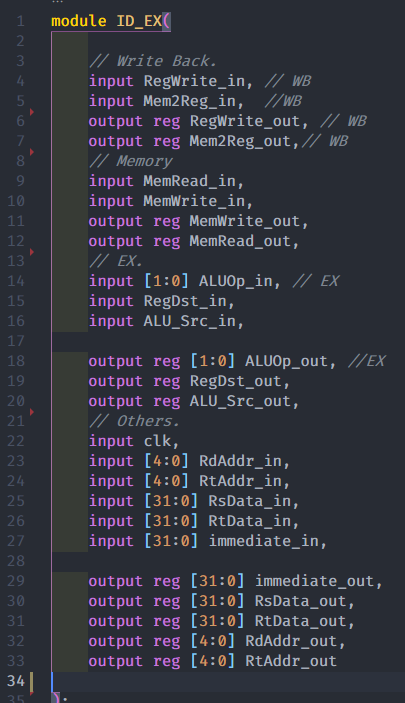


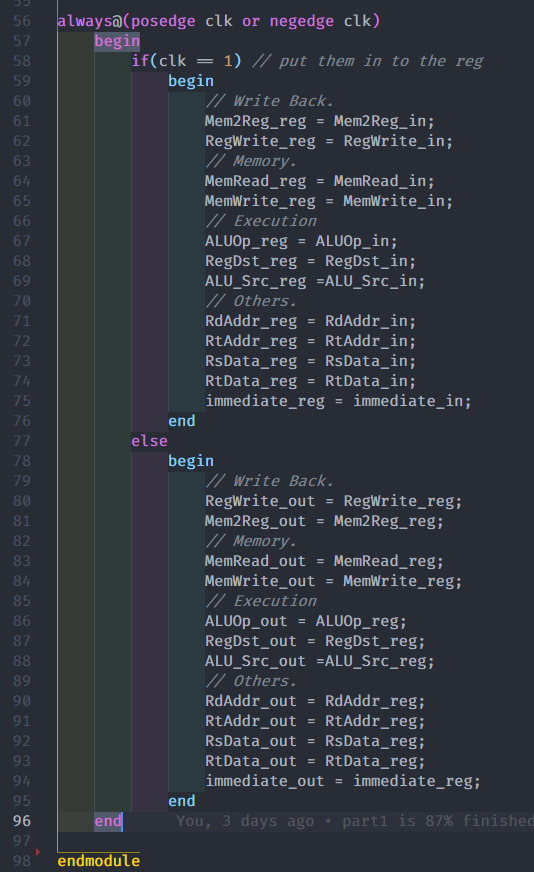
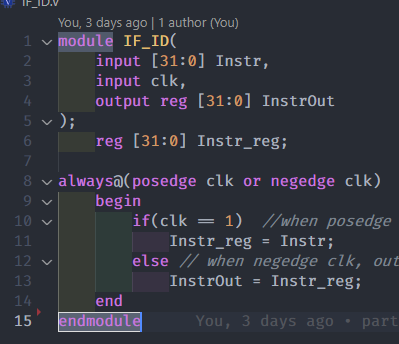
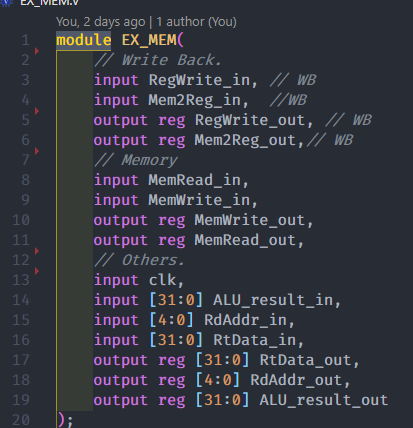
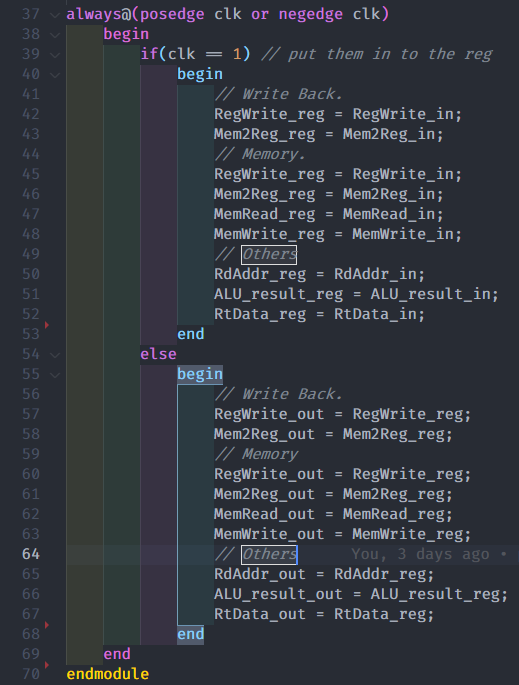
1. **MUX**

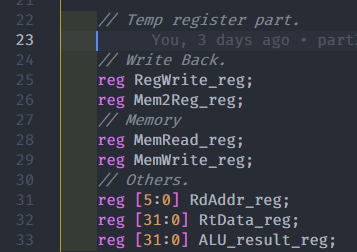
MUX這二個module其實非常簡單，一個是5bit,一個是32bit.只要判斷choose選的是多少，就決定輸出要送哪一個輸入出來。

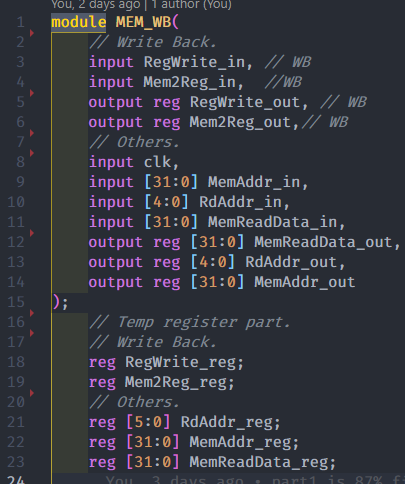
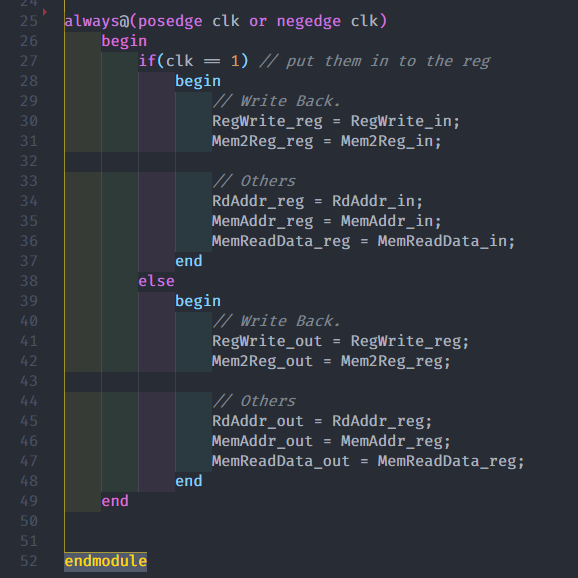


1. **IF/ID, ID/EX, EX/MEM, MEM/WB**







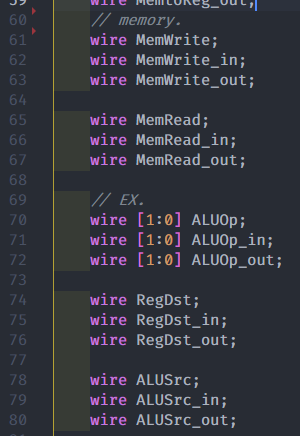
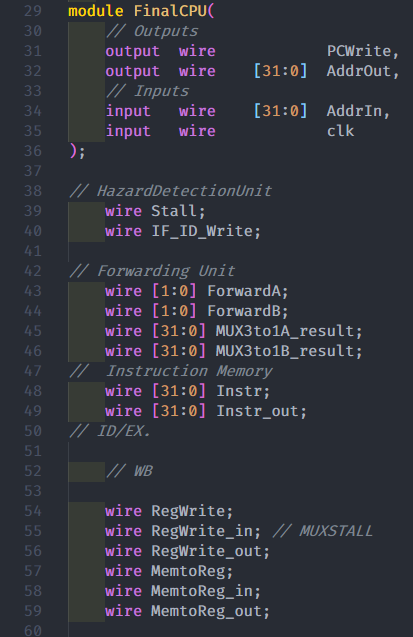


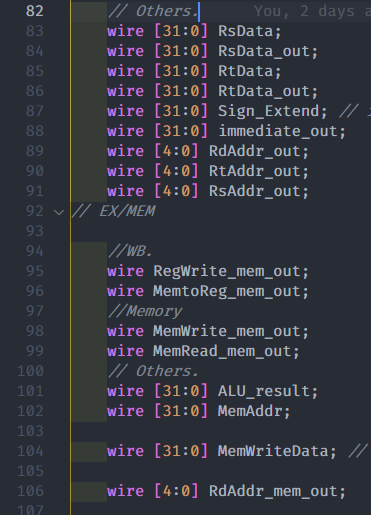
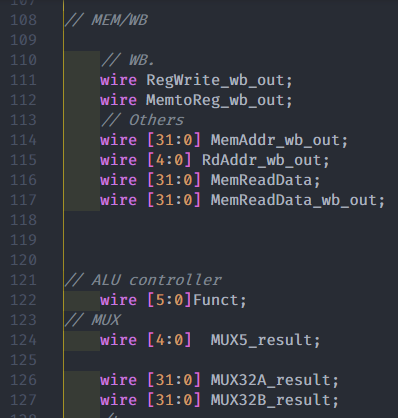
這幾個module都是Pipeline register.因為性質相同，且無任何特別的，因此在這個地方，就將他們放上來，所有的pipeline register都是正緣觸發，將值傳入register,當負緣的時候才更新輸出的值。確保不會影響到下一刻的output.

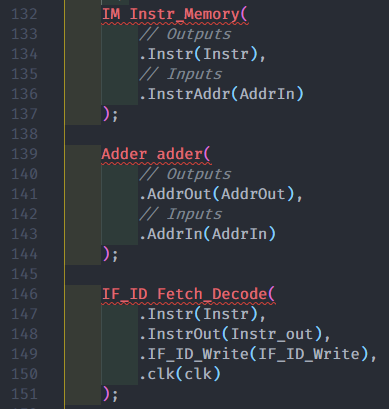
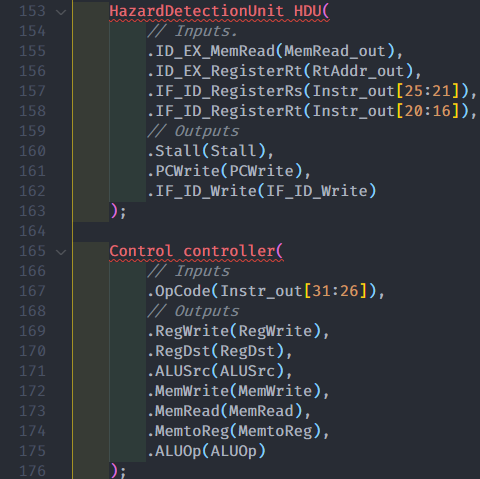
* **Part III.**

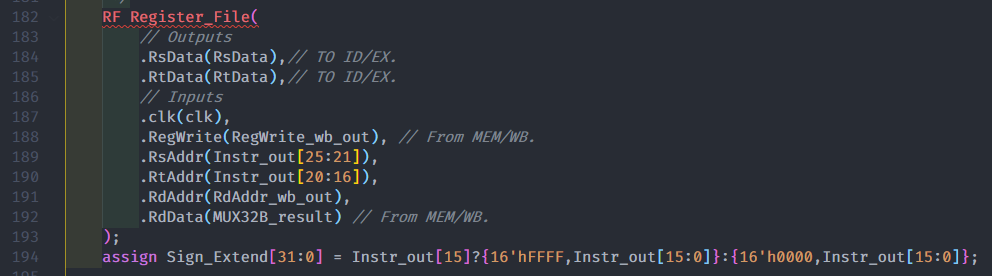
1. **FinalCPU**

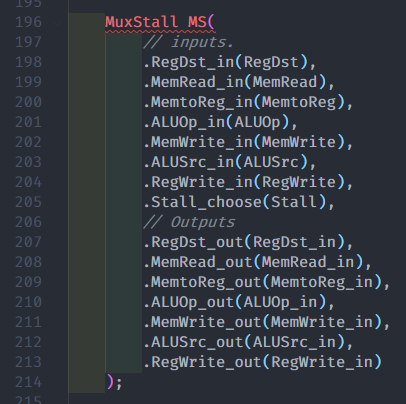
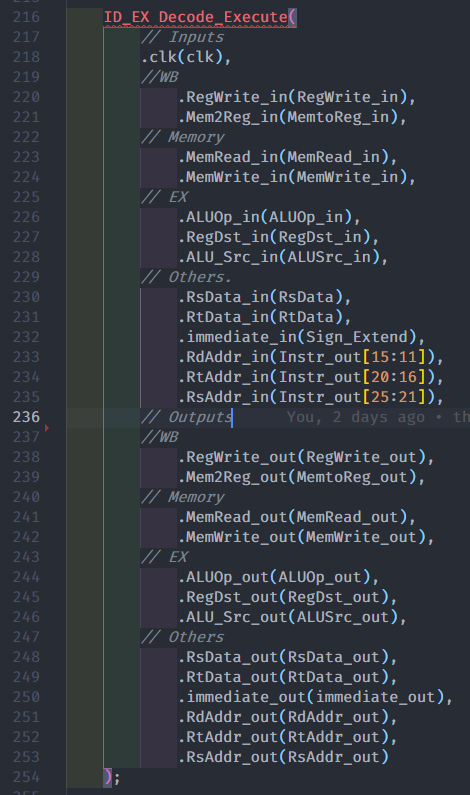
tb\_FinalCPU與題目所提供之檔案相同，因篇幅限制而不另行截圖。這個Final是將所有模組集大成之結果。我在裡面宣告了一些wire讓他去相互連接。這裡面比較特別的是有一個被我宣告的wire叫作SignExtend,是去把輸進來的16bit的Immediate值，延長到32bit。與part2不同的除了多了hazard detection unit, forwarding unit以外，在控制ALU的地方也有些許不同，則其他基本上都相同。

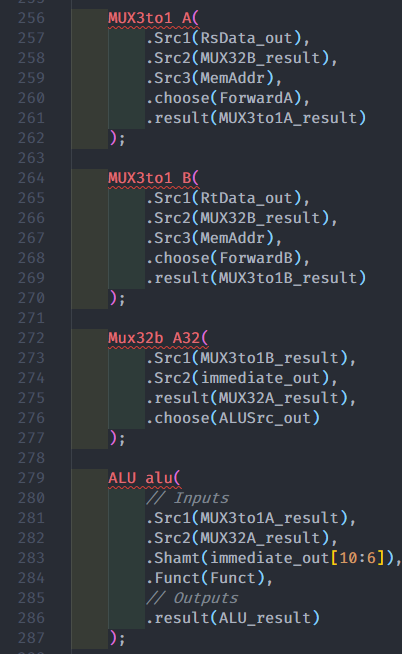
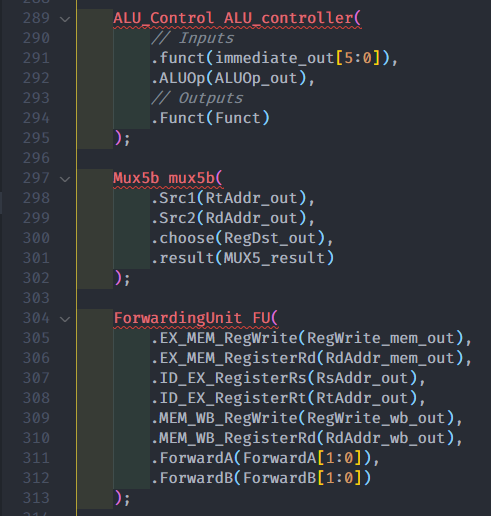
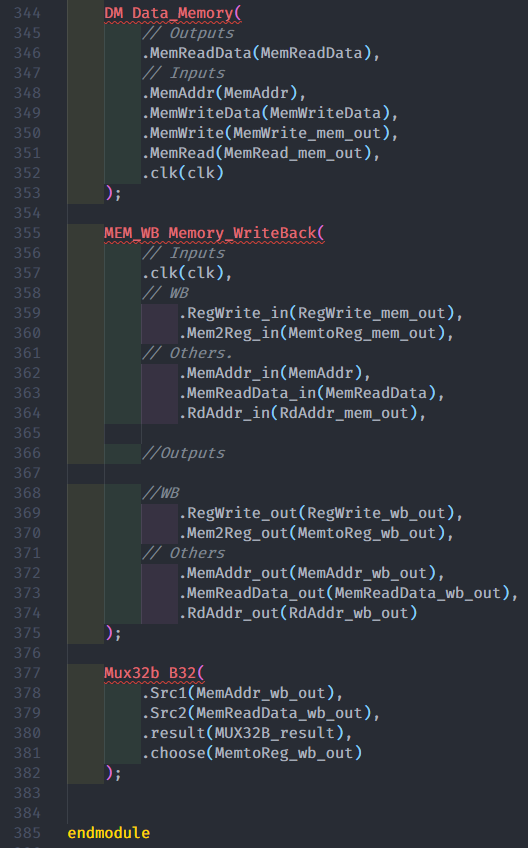
 

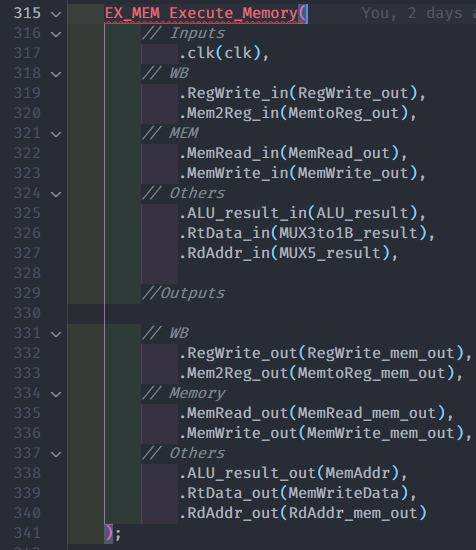








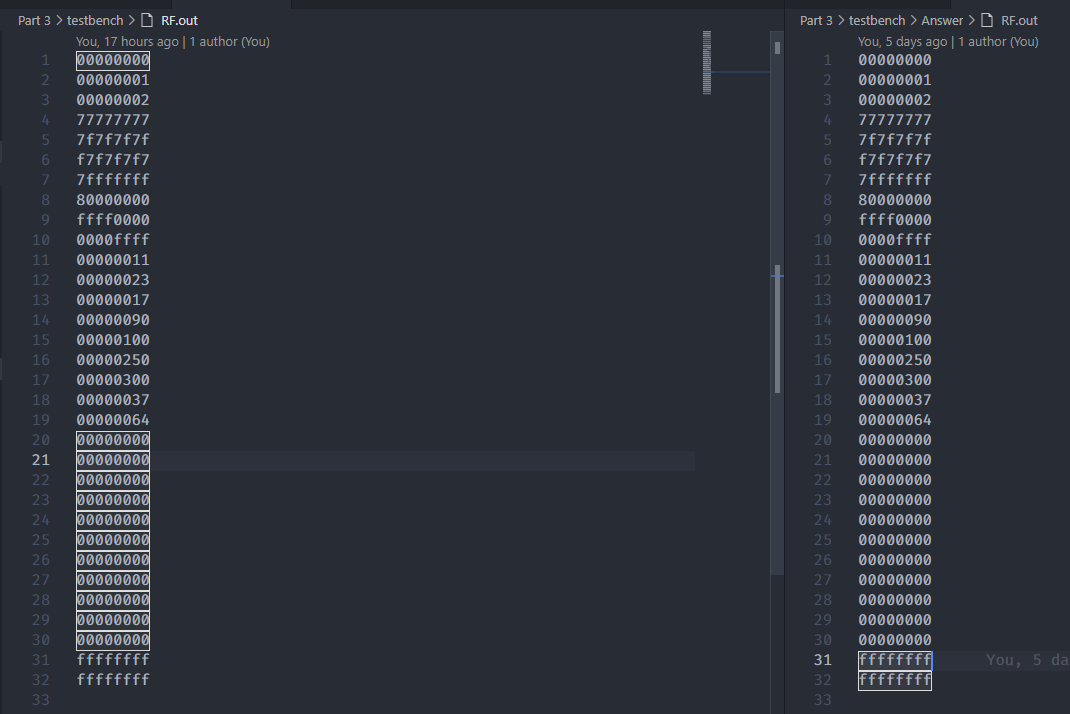




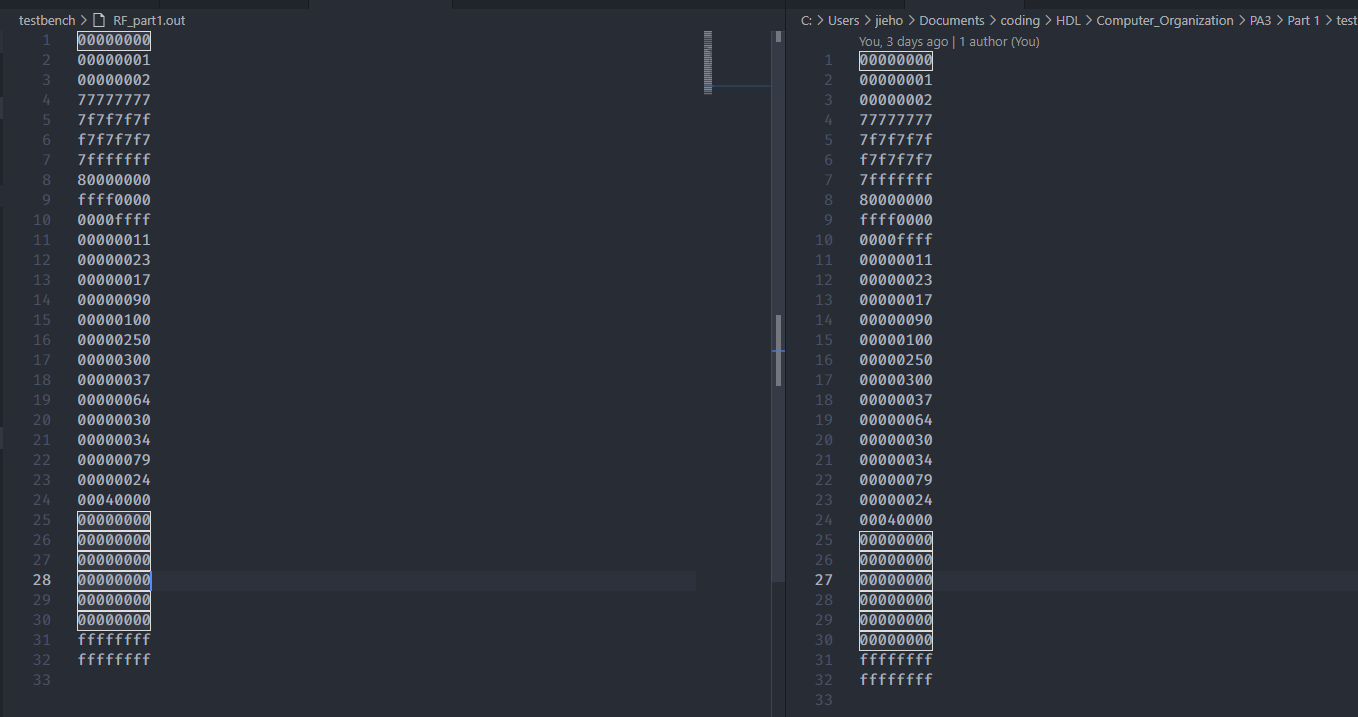
而下方右圖是DM.out經過執行後輸出出來的結果(全部都為FF) 而對比於題目提供之檔案(右圖)可以發現完全相同。



而下方右圖是RF.out經過執行後輸出出來的結果，而對比於題目提供之檔案(右圖)可以發現完全相同，因此Final到此順利做完。



下圖是Rtype指令在SimpleCPU上模擬後輸出的結果，我們可以對比part1的RF.out輸出，可以發現一模一樣，因此可以確認成功，

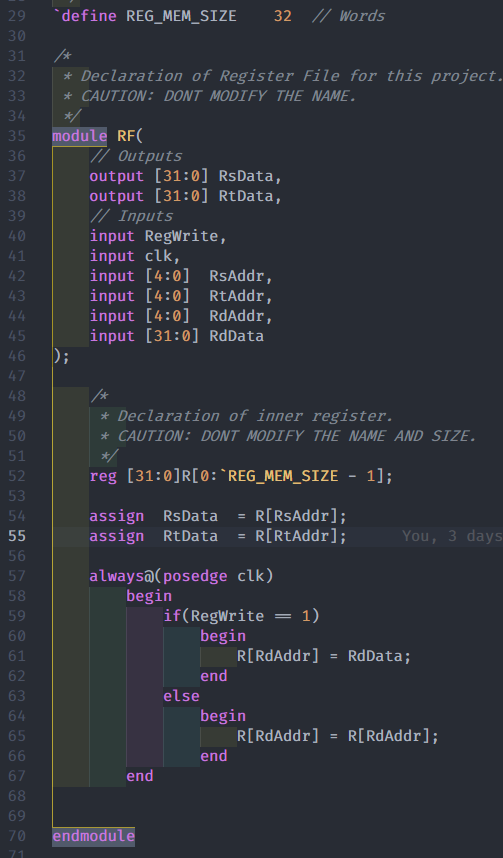


1. **Instruction Memory**

IM這個module其實很簡單，只要輸入Instruction Address，接著我就到該位置去抓Instruction，因為這個系統是BIG-ENDIAN，因此我抓的順序就會是如我程式碼的方式去抓{0,1,2,3}.



1. **Register File**

RM這個module其實不難，只要判斷RegWrite是否為1，來決定是否可以將值寫入Reg，那其他部分就是到Register的address去抓值去輸出。我將RsData與RtData使用assign而非放在always裡面是因為我發現放在裡面會等到正緣觸發才把值送入ALU，那這樣的話就無法達到我們想要的效果了。

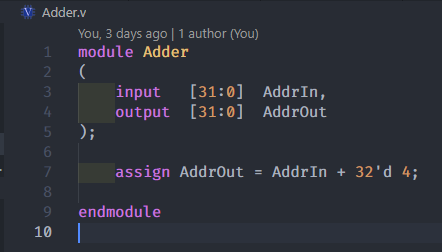
1. **Data Memory**

DM這個module其實也不難，只要判斷MemWrite跟MemRead是否為1，來決定是否可以將將值寫入Memory或是把Memory的值給讀出來。我將MemReadData使用assign而非放在always裡面，與RF的原因相同。我發現放在裡面會等到下個正緣觸發才把值送出，那這樣的話就無法達到我們想要的效果了。



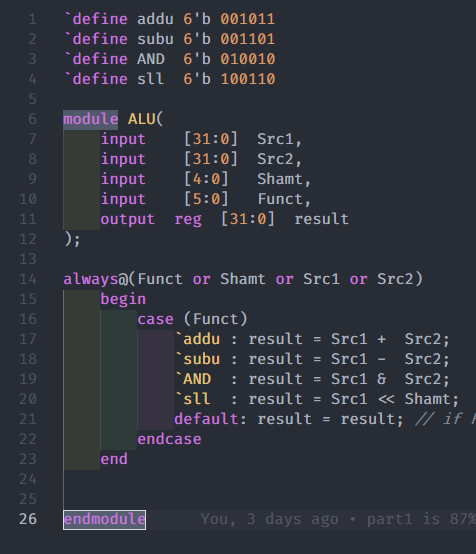
1. **Adder**

Adder所做的事情非常簡單，因為在R-type的AdderOut僅僅需要能夠將AddrIn+4，因此我就只做了加4的動作，然後輸出。(我的加法是unsigned的加法，因為Address沒有負的。)



1. **ALU**

ALU 主要是用來做Src1 and Src2的運算，由輸入的Funct來決定要做甚麼工作。



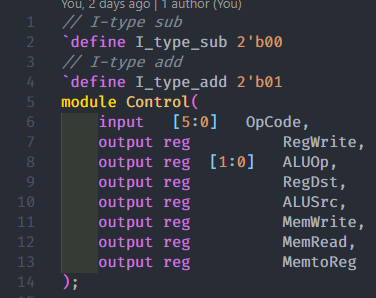
1. **ALU\_Control**

ALU\_Control主要是用來控制ALU工作與否，及將instr的指令轉為ALU懂得function code。比較特別的是在上一個作業的input\_addu與ALU所要求的funct code不同，而這次作業的code是相同的，導致我在debug時花了一些時間才發現。



1. **Control**

Control這個module在整個CPU裡面是一個非常重要的角色。他掌管整顆CPU現在要做甚麼，不要做甚麼。雖然他極其重要，但是其實沒有甚麼太複雜的工作，只要依照Opcode的要求，來決定是否要送各種訊號。而因為R-type的所有指令都會使用到ALU，因此我們ALUOp只要是對的Opcode，我們一律送2`b10.而對於I-type指令來說，只會有加法跟減法，因此除了subiu以外(2`b00)，其他的ALUOP都為2`b01。





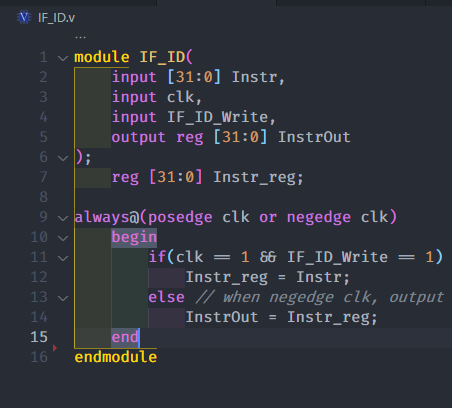
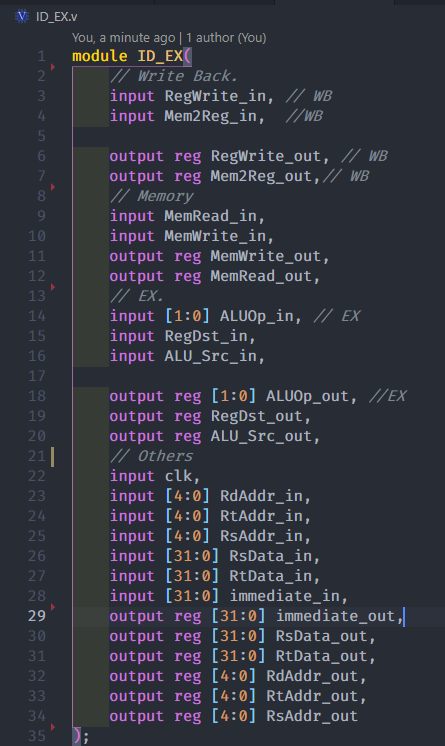
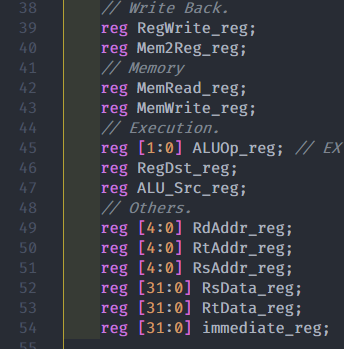
1. **MUX**

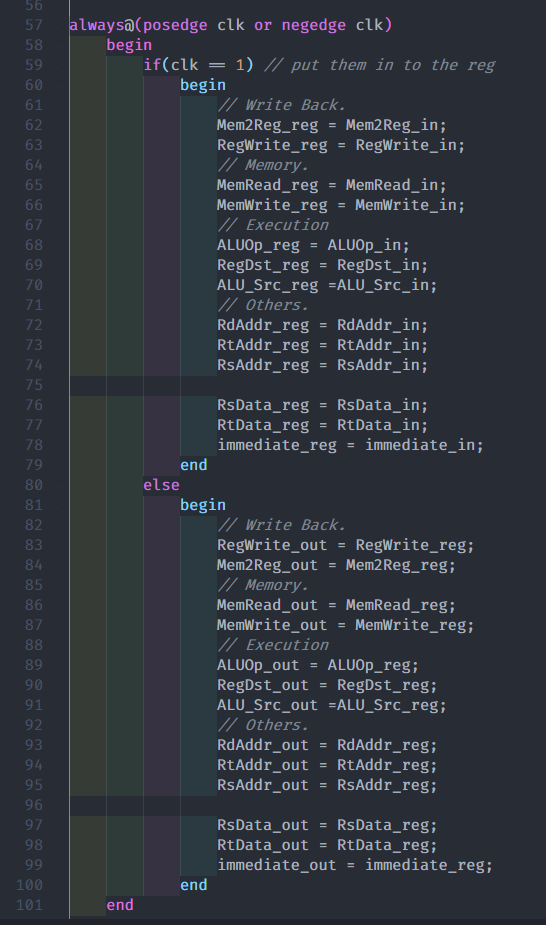
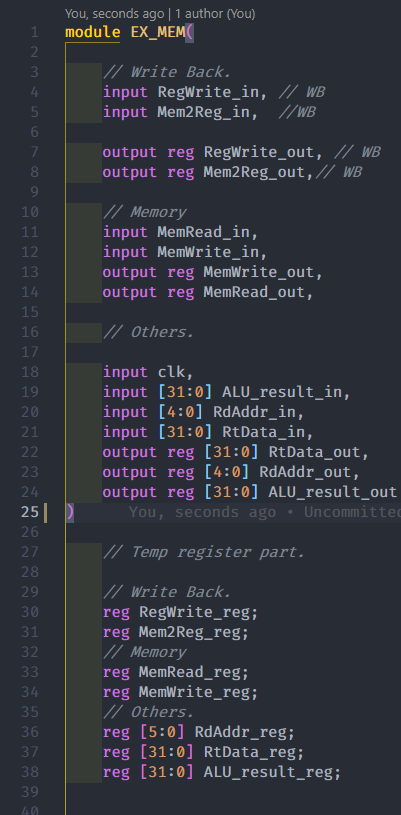
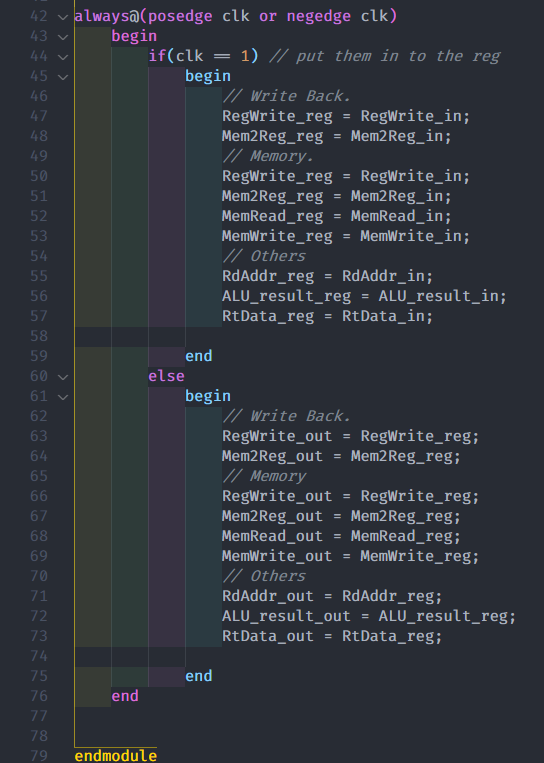
MUX這幾個module其實非常簡單，一個是5bit,一個是32bit.只要判斷choose選的是多少，就決定輸出要送哪一個輸入出來。比較特別的是MUX STALL與MUX 3 to 1.是左上角Hazard Detection Unit底下的mux.

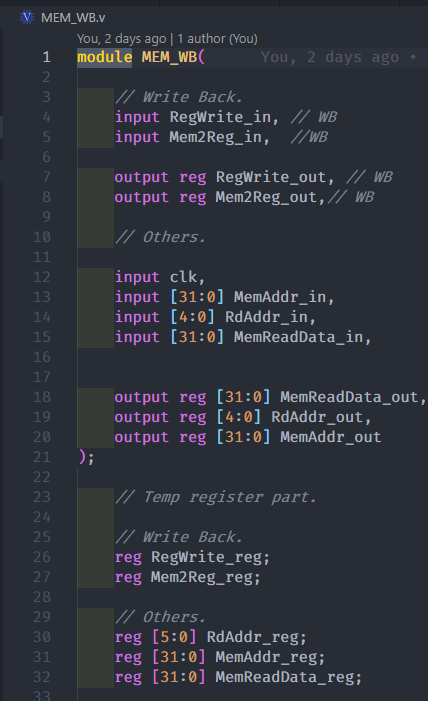
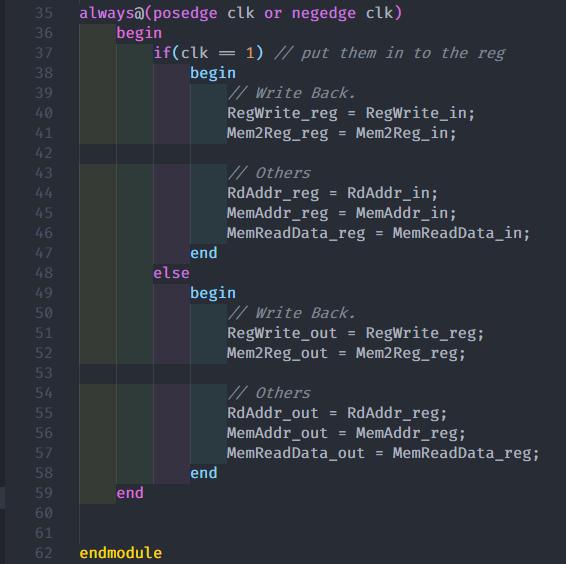




1. **IF/ID, ID/EX, EX/MEM, MEM/WB**







這幾個module都是Pipeline register.因為性質相同，且無任何特別的，因此在這個地方，就將他們放上來，所有的pipeline register都是正緣觸發，將值傳入register,當負緣的時候才更新輸出的值。確保不會影響到下一刻的output.

1. **4. 作業總結與心得**

這次的作業，從6/2星期三出的。因為疫情的關係，全台所有學校進行遠距教學，讓我原本排滿滿的生活，有了充裕的時間可以自行運用。因此這次的作業，我從出作業當天晚上開始思考如何做起，當天晚上把part1做完。而隔天早上把part2做完。下午開始處理Part3，相比part2與part3，真的是難度相差非常多，因為他不僅要處理pipeline, 又要可以偵測hazard，還要做forwarding.在那個晚上在兜完線以後，滿心期待地去跑模擬，結果出來有三個register，是錯誤的，於是就開始了整個晚上的debug時間。

一直忙到大概隔天的凌晨五點，我一直反覆追溯bug的來源，把所有波形放出來看，卻都解決不了問題。(這時候助教還沒有改線)，後來五點的時候我覺得實在是太累了，必須休息，但是我還是不死心的google了一下，結果躺在床上的時候居然找到答案了，這時候我就立刻跳起來打開電腦試試看新學到的方法，結果真的解決了1/3的bug. 但還是有兩個臭蟲仍然存活於我的code之中。不過我還是心滿意足地去睡覺了。到了隔天助教重新發了一張新的接線圖，我就開始依照這張圖重新施工。不過就算花了好幾小時重新施工，我仍然有相同的問題。後來我就有點進入放棄狀態，打算請求同學的幫助，並問他們是否接線圖有問題。結果同學回我說照著最新的圖是沒問題的。後來發現原來一開始我是照著上課投影片多工器的方法去接線，最後才發現原來課本的投影片多工器那邊出了問題。而更改過後，模擬結果還真的對了。

這次的題目主要是以PA2為基礎，增加pipeline register來減少clock數量。雖然因為part3的圖我沒有搞清楚就開始做所以多花了很多時間以外，其他部分都很順利的過五關斬六將了。我在做最後一個project才突然發現一些增進debug的技巧，可以從Model Sim那邊看到各條wire和reg的值，之前大部分的project我都會與同學討論，但這次我一人鑽研居然就成功了。這學期有修計算機組織真的很開心，希望這堂課的期末考試，也可以像這次一樣順利度過。