VLSI System Design (Graduate Level)

Fall 2021

HOMEWORK I

REPORT

Must do self-checking before submission:

Compress all files described in the problem into one tar

All SystemVerilog files can be compiled under SoC Lab environment

All port declarations comply with I/O port specifications

Organize files according to File Hierarchy Requirement

No any waveform files in deliverables

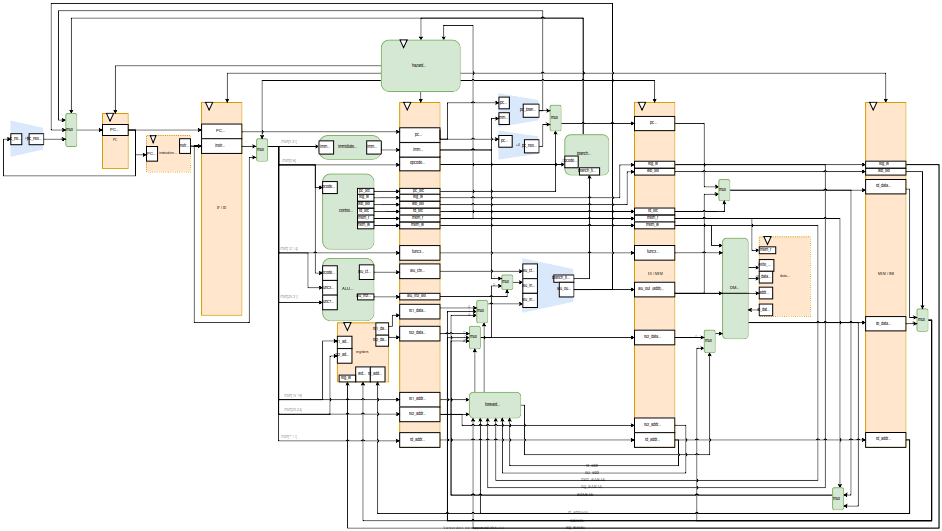
Student name: \_伍志忠\_

Student ID: \_P76104833\_

Summary

完成所有的指定的指令、測試程式，並且成功合成。

a , b. Proper explanation of your design,Block diagrams

[](https://raw.githubusercontent.com/cicero884/VLSI_system_design_2021/main/hw1/cpu.drawio.svg)如圖(<https://raw.githubusercontent.com/cicero884/VLSI_system_design_2021/main/hw1/cpu.drawio.svg>)

藍色主要是加法器或是ALU，綠色是控制單元或是mux，橘色是register或是memory，有加三角形的表示有接clock。

IF階段會看下個指令的PC是多少以及抓指令。  
ID階段會產生immidiate以及各種控制訊號線

EX階段負責算，如果要跳(branch or jump)的話也會在這階段執行

MEM階段負責把值整理後存進記憶體  
WB階段把值寫回去register

Forward unit如果遇到

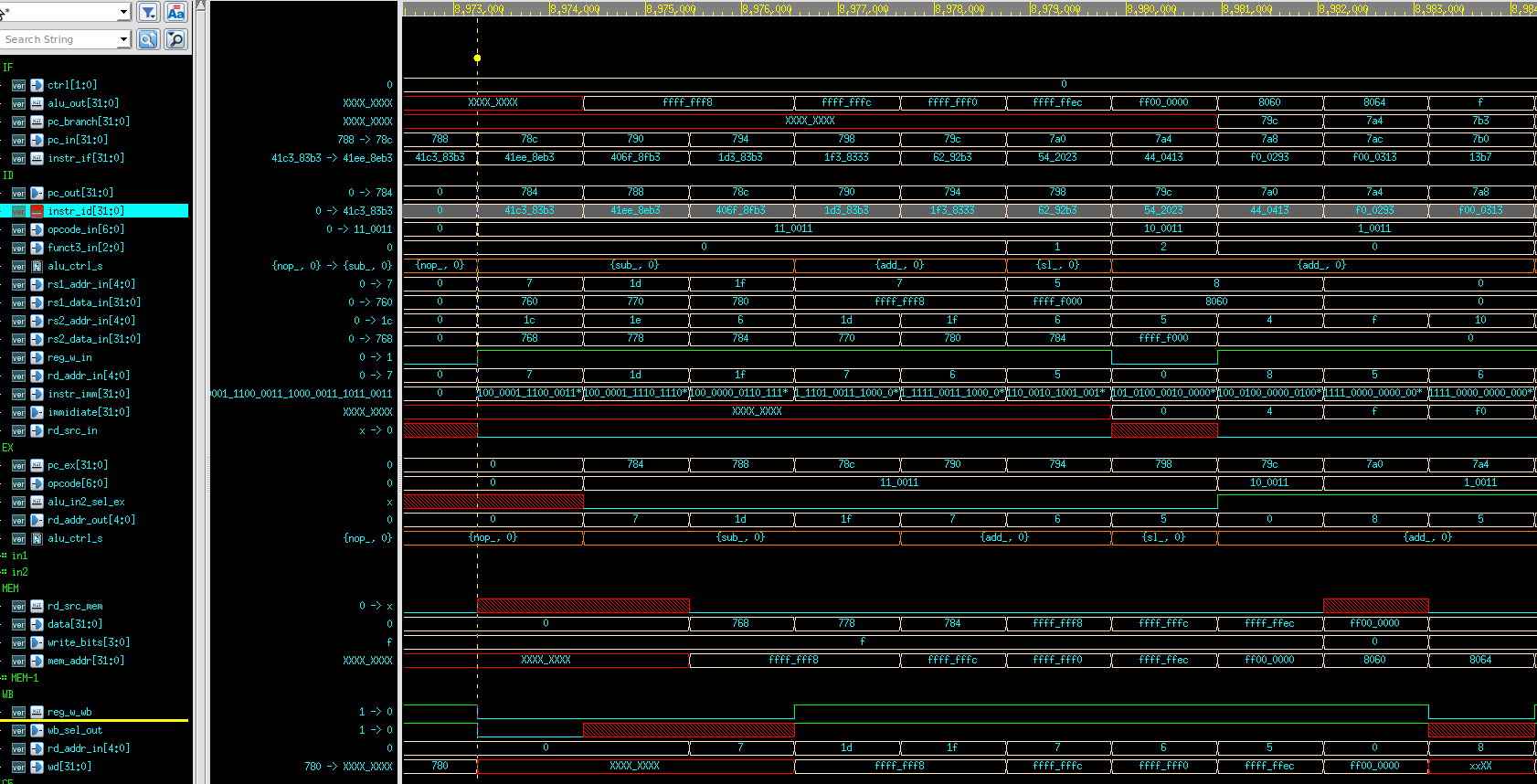
1.mem或是wb階段有ex階段需要的值就forward過去

2.wb階段有mem階段需要的值就forward過去

Hazard control unit如果看到memory read會stall pc,if\_id\_reg,id\_ex\_reg。如果遇到要跳指令的狀況就把if\_id\_reg和id\_ex\_reg裡的值(前一個指令)清空。

c. Show your snapshots of the waveforms and the simulation results on the terminal for the different test cases

R-type:



從指令78c(SUB),790(ADD),798(sll)

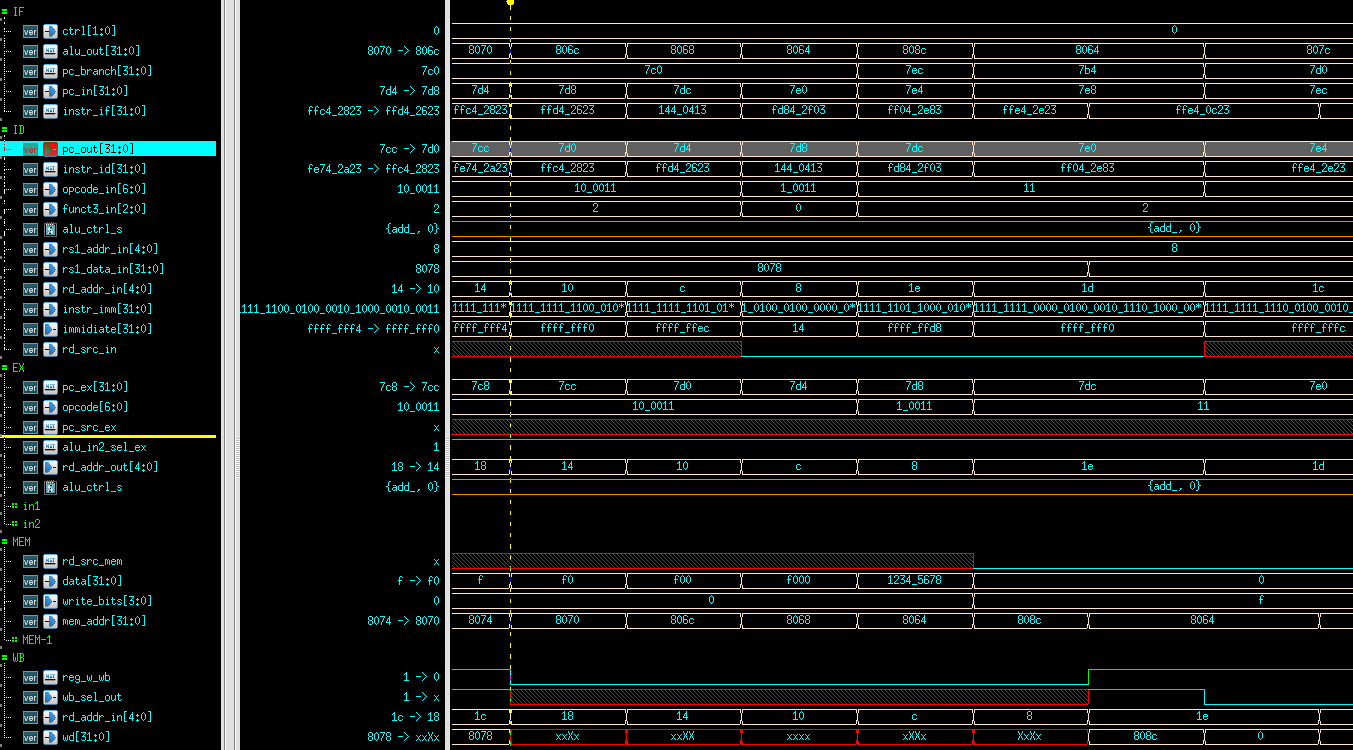
可以觀察到三者都是alu\_in2\_sel都被設成0（使in2信號從register而非immidiate到alu）。

wb\_sel設成1(寫回去計算的結果而非mem讀出來的資料)  
reg\_w設為1(要寫回去register)

從alu\_ctrl可以看到送給alu的信號為{sub\_,0}{add\_,0}{sl\_,0}

信號的前面部份表示執行的操作，後面(0)表示是有號還是無號

I-type

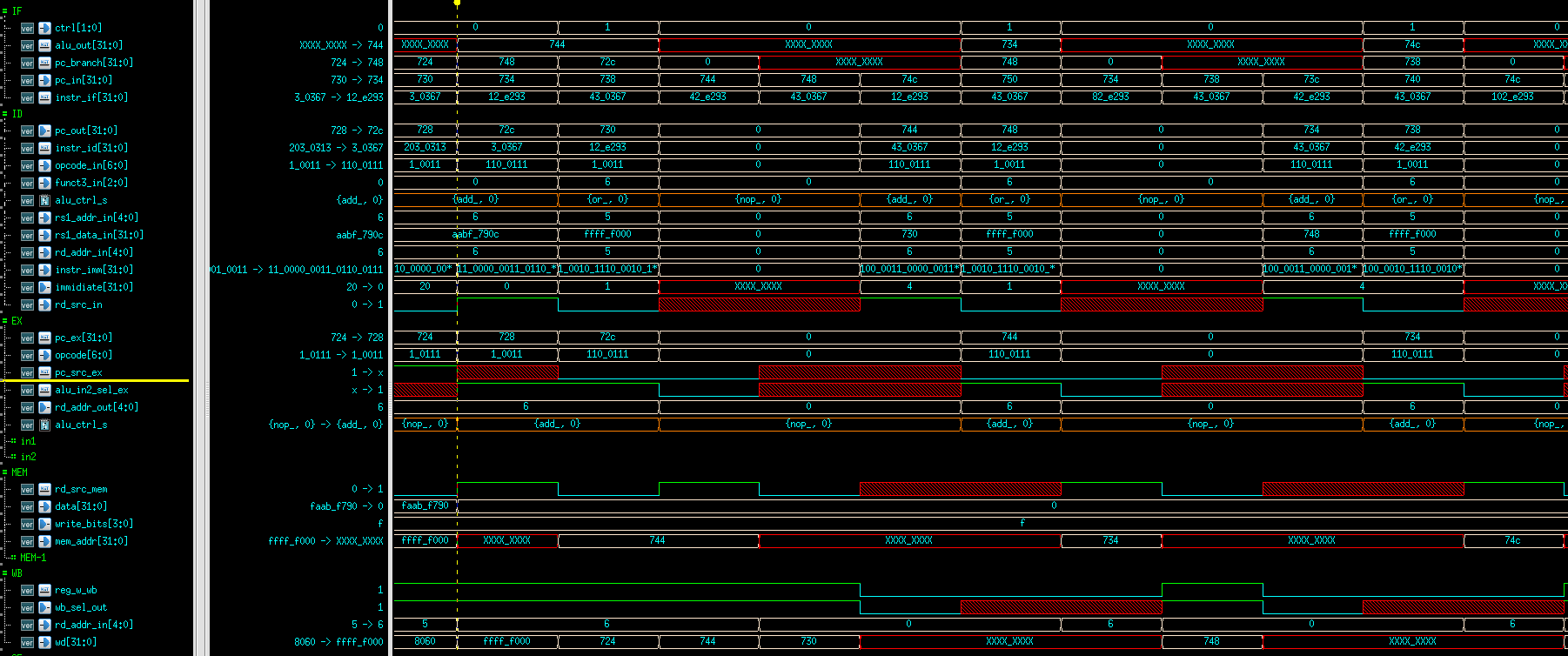


指令7d8(ADDI)

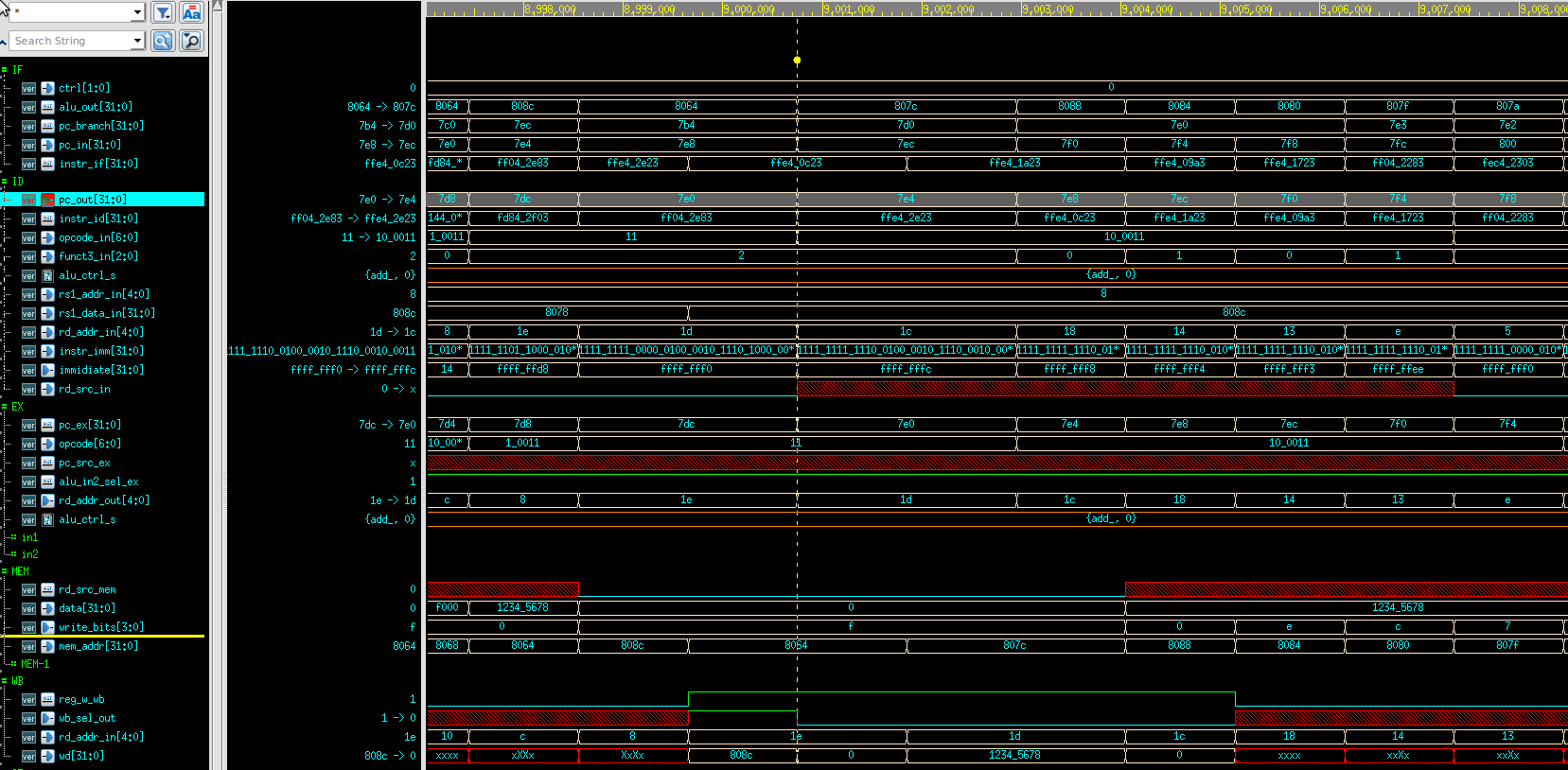
alu\_in2\_sel被設成1(使in2信號從immidiate而非register到alu)  
wb\_sel設成1(寫回去計算的結果而非mem讀出來的資料)  
reg\_w設為1(要寫回去register)  
從alu\_ctrl可以看到送給alu的信號為{add\_,0}。

指令7dc(LW)

alu\_in2\_sel被設成1(使in2信號從immidiate而非register到alu)  
wb\_sel設成0(mem讀出來的資料而非計算的結果寫回去)  
reg\_w設為1(要寫回去register)  
從alu\_ctrl可以看到送給alu的信號為{add\_,0}。  
控制信號到hazard controler把前面的(pc,IF\_ID\_reg,ID\_EX\_reg)stall，等待記憶體資料讀出來

指令72c(JALR)  
pc\_src設成0(把pc+4導出來，之後要存進register)  
wb\_sel設成1(寫回去計算的結果而非mem讀出來的資料)  
reg\_w設為1(要寫回去register)  
rd\_src設成1(選擇pc+4的資料而非alu的資料存)  
alu\_in2\_sel為1(使用immidiate的資料)  
alu的操作則是add，把immidiate跟register的資料相加送回IF，由branch controller控制要由alu輸出成下一個的PC，同時通知hazard controller把多餘已經進入pipline的指令清空

S-type

指令7e4(sw),7e8(sb),7ec(sh)

reg\_w設成0(無須存回register)

mem\_r設成0(沒有要讀記憶體)

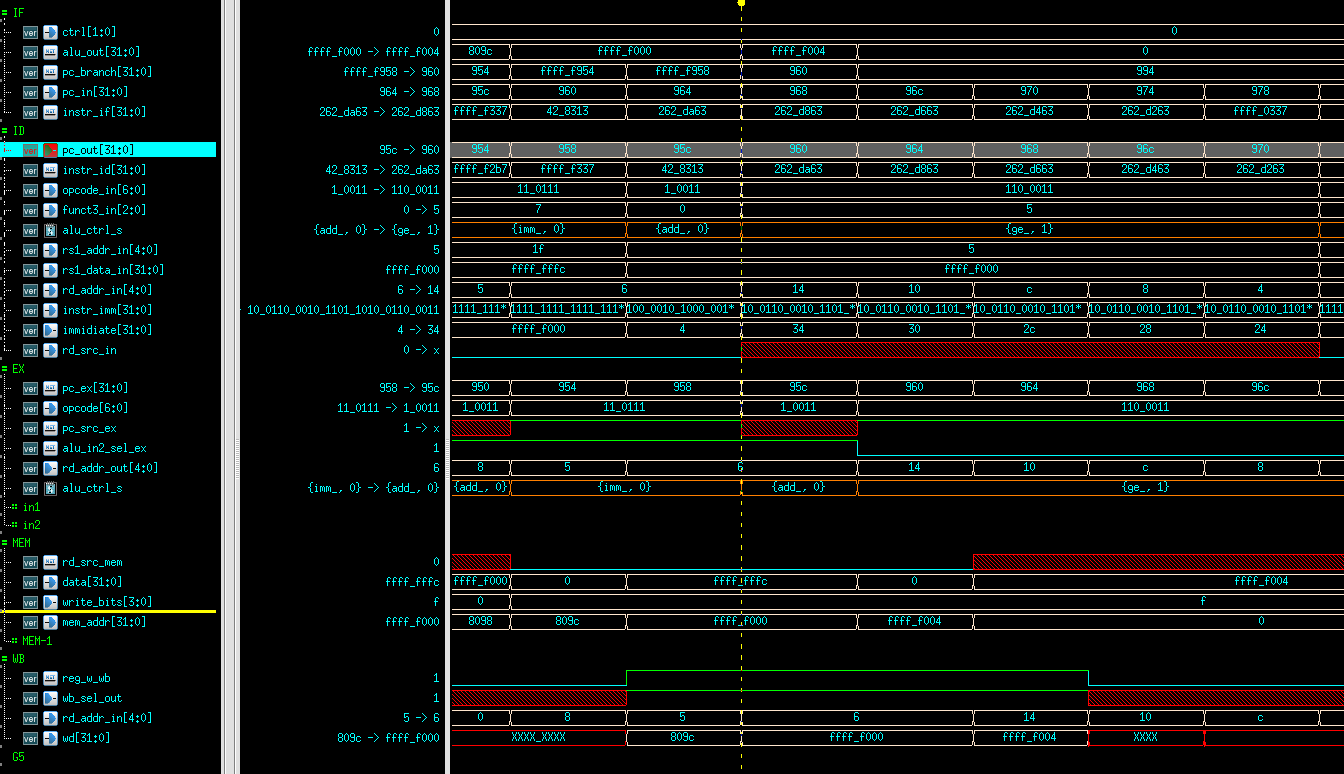
mem\_w設成1(要寫記憶體)

alu\_in2\_sel要設成1，因為mem\_address是由register存的值加上immidiate的值，alu\_sig也要設成add\_

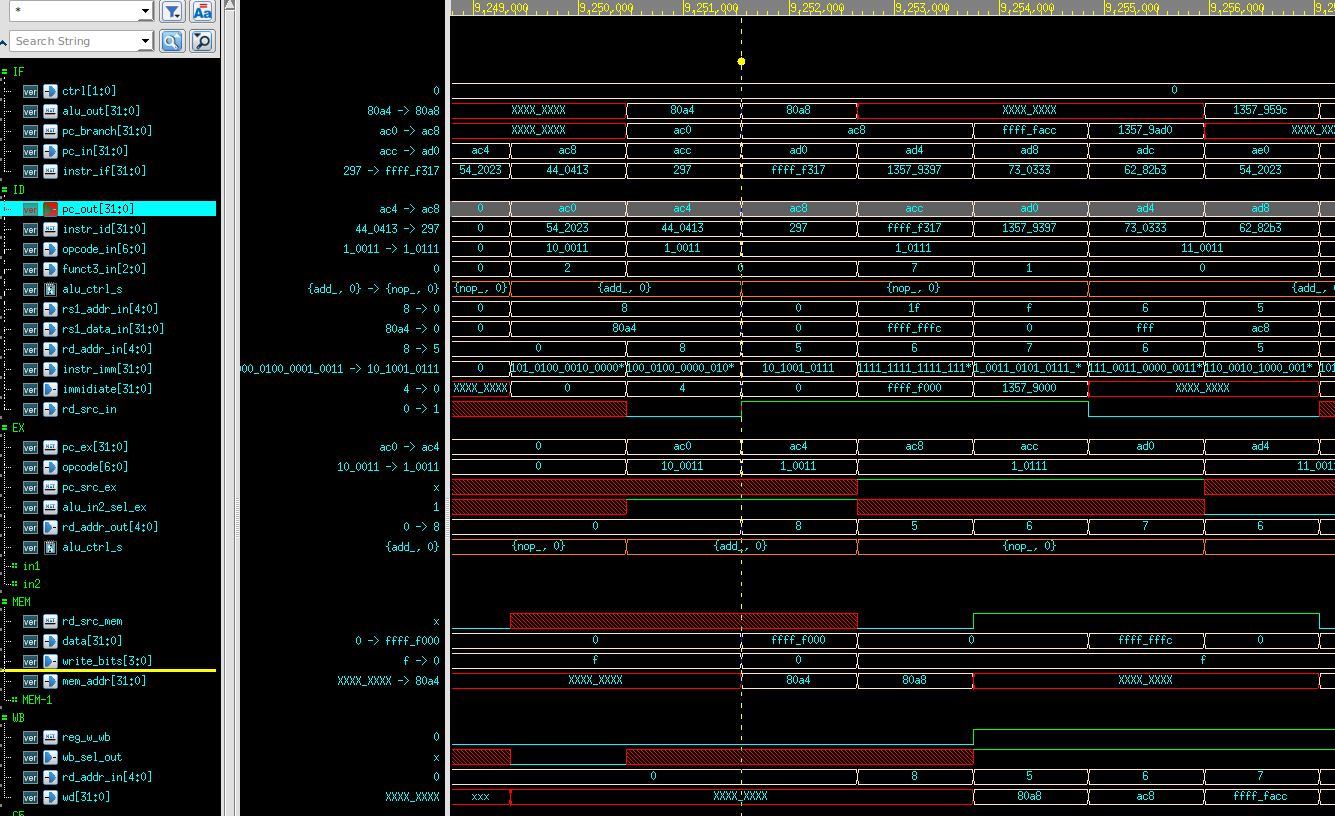
其他信號隨意

最後存資料時sb,sh要依據記憶體的位置製作相對應的位移以及設定WEB(DM\_controller)

B-type

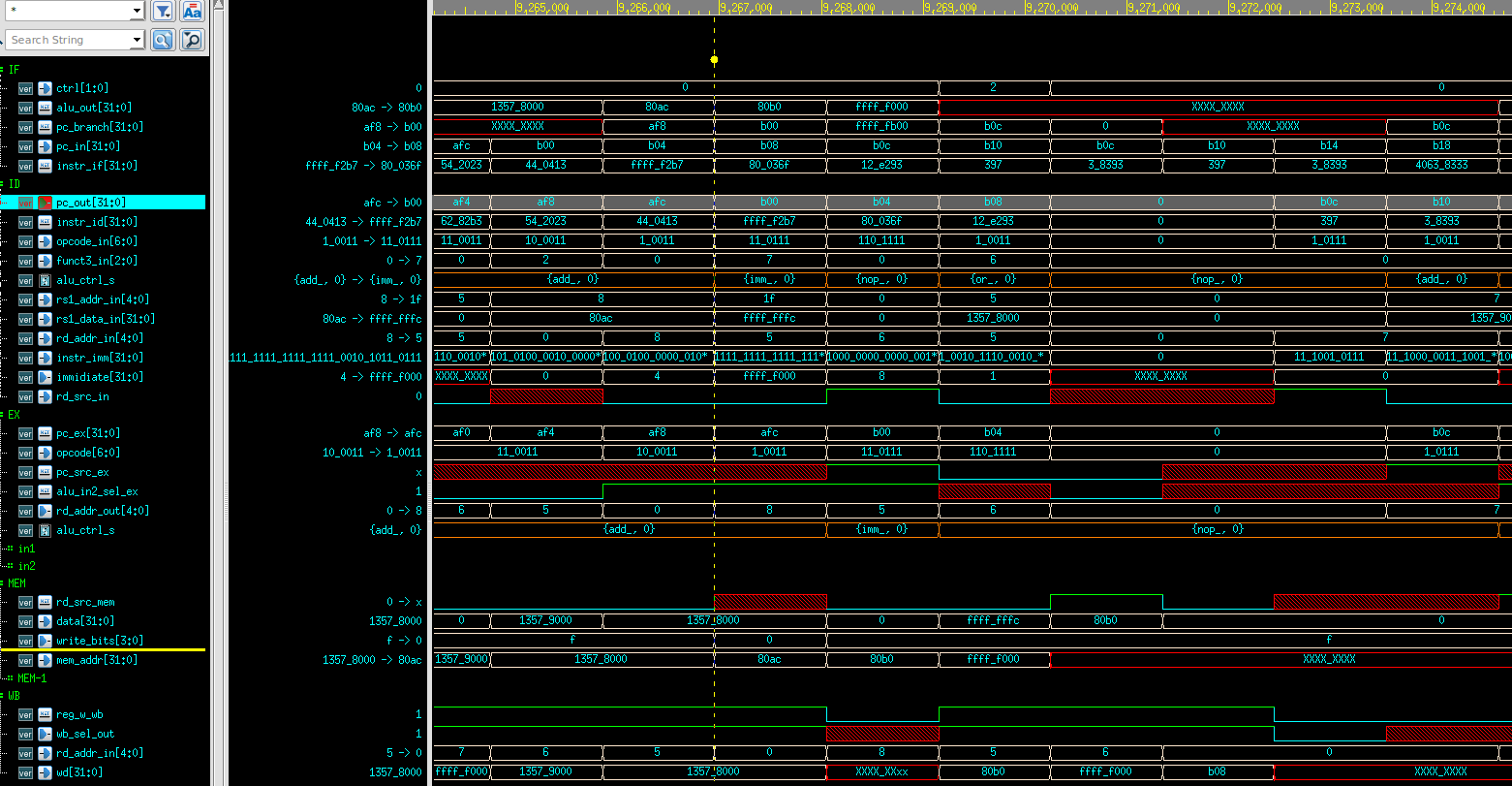
指令960(bge),beq,bne

B-type的指令都是類似的，只有給alu告訴要使用哪種運算的差別  
reg\_w,mem\_r,mem\_w設成0(沒有要寫或讀任何地方)  
如果ALU運算的結果要跳的話branch\_ctrl會設成2  
其他的信號值不用管，跳的時候同時通知hazard controller把多餘已經進入pipline的指令清空

U-type

指令ac8(auipc)  
pc\_src設成1(要存pc+imm)  
reg\_w設為1(要寫回去register)  
wb\_sel設成1(寫回去計算的結果而非mem讀出來的資料)  
rd\_src設成1(把pc+imm的資料導過去存)

mem\_r,mem\_w都設成0(跟mem無關)

指令b00(LUI)

reg\_w設為1(要寫回去register)  
wb\_sel設成1(寫回去計算的結果而非mem讀出來的資料)  
rd\_src設成0(把alu\_out的資料導過去存)  
雖然是alu出來的資料，但是不會加另一個值

指令b04(JAL)

pc\_src設成0(要存pc+4)  
reg\_w設為1(要寫回去register)  
wb\_sel設成1(寫回去計算(PC+4)的結果而非mem讀出來的資料)  
而ALU完全不會用到，行為是nop\_

d.

1.Report the number of lines of your RTL code:

`$wc -l src/\*.sv`

  11 src/Adder.sv   
  67 src/ALU\_control.sv   
  24 src/ALU\_signal\_def.sv   
  63 src/ALU.sv   
  27 src/Branch\_control.sv   
  92 src/Control\_unit.sv   
 285 src/CPU.sv   
  50 src/DM\_control.sv   
  47 src/EX\_MEM\_reg.sv   
  37 src/Forward\_control.sv   
  39 src/Hazard\_control.sv   
  71 src/ID\_EX\_reg.sv   
  55 src/IF\_ID\_reg.sv   
  21 src/Immidiate\_generator.sv   
  31 src/MEM\_WB\_reg.sv   
  13 src/Mux\_2in.sv   
  20 src/Mux\_3in.sv   
  20 src/PC\_reg.sv   
  20 src/Register.sv   
  99 src/SRAM\_wrapper.sv   
  56 src/top.sv   
1148 total

2.the final results of running Superlint

=============================================================

SUMMARY

=============================================================

Properties Considered : 19

assertions : 19

- proven : 2 (10.5263%)

- bounded\_proven (user) : 0 (0%)

- bounded\_proven (auto) : 0 (0%)

- marked\_proven : 0 (0%)

- cex : 17 (89.4737%)

- ar\_cex : 0 (0%)

- undetermined : 0 (0%)

- unknown : 0 (0%)

- error : 0 (0%)

covers : 0

- unreachable : 0

- bounded\_unreachable (user): 0

- covered : 0

- ar\_covered : 0

- undetermined : 0

- unknown : 0

- error : 0

3.3~5 most frequent warning/errors in your code

1.**"Constant '{imm\_,1'b0}' will be left-padded by 4 '0' bits"**

我在ALU\_signal\_def有定義了一個packed struct，是4bits的enum加上1bit的is\_signed，那個constent應該也是4bits enum加1bit的is\_signed，我不懂這錯誤出現的原因，因此沒改。

2.**"Synthesizing 'x'/'z' values in module 'Control\_unit'"**

我確定那些值在那些階段完全不會用到，因此我希望能在合成時讓他自己決定並優化。

3.**"Unequal length operand in bit/arithmetic operator addition in module/design-unit CPU1.dm\_control. LHS operand '{mem\_addr[1:0],3'd0}' is 5 bits, RHS operand '8' is 32 bits"**

我定義了

`define RANGED\_DATA(source,scale,shift) \

source[{shift,3'd0}+scale-1 -:scale]

那個值(8或16)只是表示scale，它最後應該會只是邏輯的一部分而不是線，因此我不想加上bit數。

4.**\*E, "A reachable x-assignment was found"**

同(2)，不管合成出來的值是多少都不會影響結果。

Encountered problem

在寫的過程中最煩的就是看波形debug，看到線接錯、邏輯寫錯、文件沒看仔細等等，發現data memory慢一個clock出來還改了一下架構。幸好main.S有一些註解有標注它應該出來的值是多少，省下了許多自己推演的時間。

這次作業直到合成出來，看到超出預設的clock。明白critical path後跑回去看計機組織他的forward怎設計的，才發現一般的forward沒有從mem讀出來forward。因此我的critical path是  
[mem control]->[forward mux]->[alu]->[branch control]->[hazard control]。我以為mem read的 stall會讓它多一個clock能來得及回去，但是實際上並沒有，幾乎是在一個clock做了2個cycle該做的事，因此預設跑出來的report timing slack=-0.2。這可以藉由修改設計來解決，但是在繳交時間壓力下我決定增加clock的值成11。

(以上是在自己實驗室的結果，在電機系的那邊跑好像優化較好6~7左右，可以改cycle改很低)

最後再打包的時候，因為我的系統sh並不是指向bash，但是makefile預設是bash，因此需要在makefile前面加上

`SHELL := /bin/bash`才能打包

learned

複習了verilog的相關知識以及計算機組織，並學習了system verilog的一些寫法、記憶體的delay、相關工具(nWave,superlint,synthesize等等)的使用方法。