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

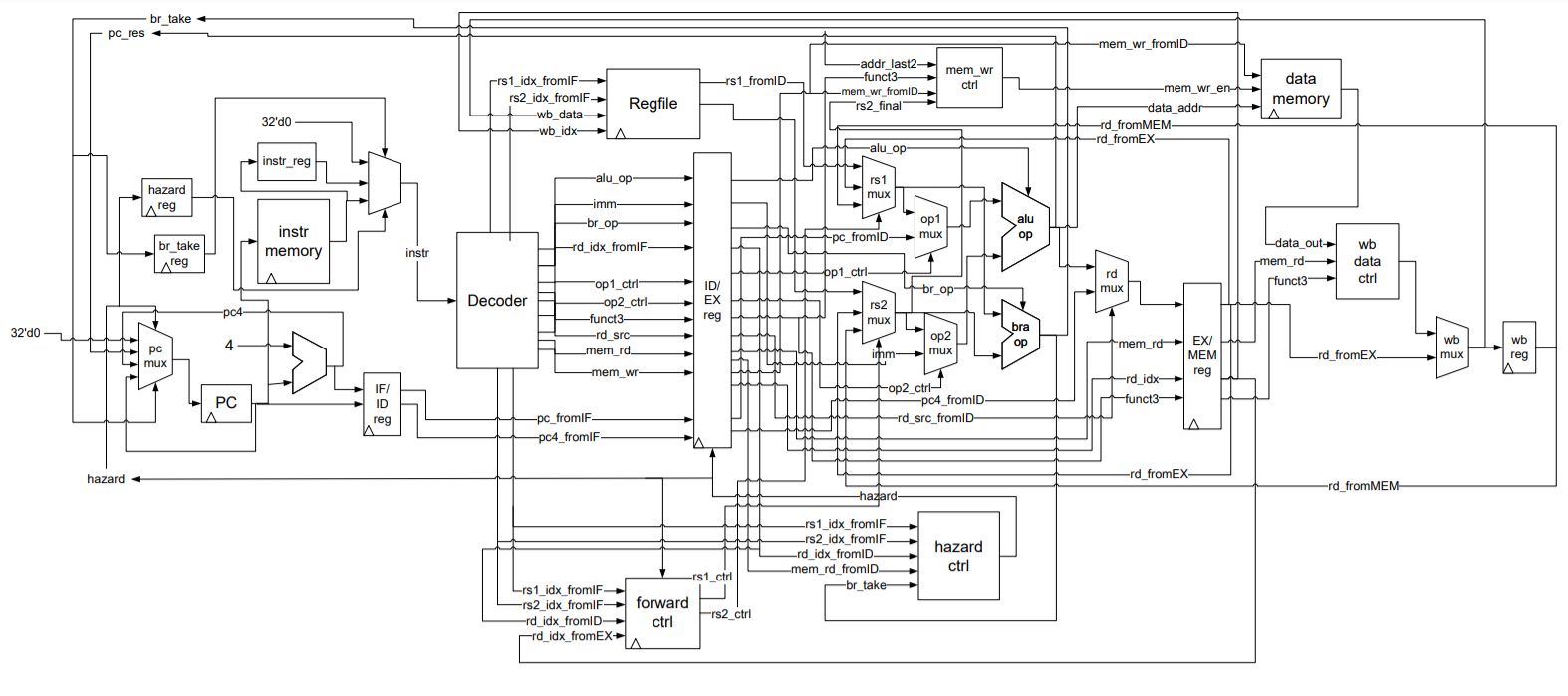
No any waveform files in deliverables

Student name: \_\_\_楊芸甄\_\_\_\_

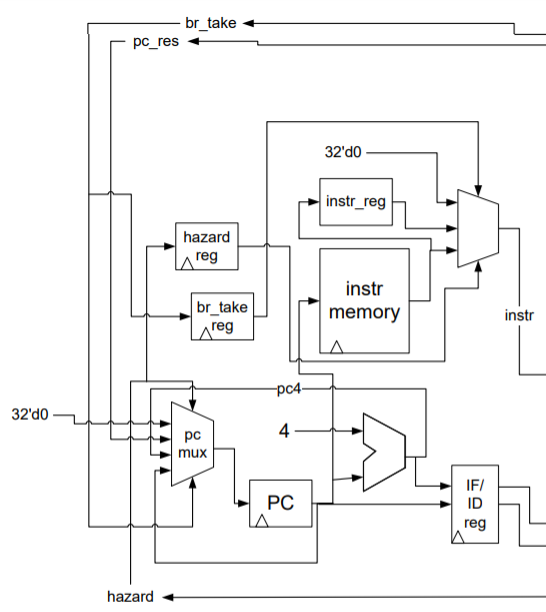
Student ID: \_\_H24071037\_\_

1. 架構設計

為5-stage pipeline CPU，搭配forward與hazard controller，總架構如下。



以下為各module說明。

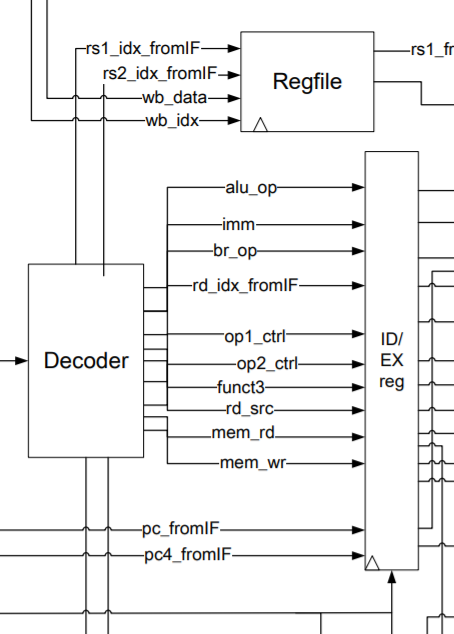
1. stage\_IF.sv

pc：在posedge時會改變值，普通情況下pc = pc+4；br\_take == 1 -> pc = pc\_res，其中pc\_res為EX中alu的運算結果；當hazard == 1(load的值在下一cycle就要用到時，hazard會拉成1)，pc = pc(產生1個bubble)

hazard\_reg、br\_take\_reg、instr\_reg：在posedge時，會將各自對應訊號的值儲存(存上一cycle的值)

instr mux：為comb電路，在普通情況下，instr = instr\_from\_mem；當hazard == 1，instr = 0(因為從instr mem拿到的指令是錯的)；當hazard\_reg == 1，instr = instr\_reg(再執行一次同樣指令，產生1個bubble)

IF/ID reg：pc和pc4的值在posedge傳到ID stage。

1. stage\_ID.sv

以opcode、funct3、funct7第5碼的值判斷為何種指令，決定各訊號如alu\_op, br\_op, op\_ctrl, mem\_wr, mem\_rd, imm，並連同其他後面stage需用到的訊號，在posedge時傳至EX stage。

其中alu\_op，在R與I type為funct7\_5與funct3組合(除sr)，其他type則為add。

br\_op的部分，若是jal或jalr會被設成Uncond，到EX時無論如何br\_take皆為1；B-type指令則把br\_op設為在EX時判斷是否需要take branch；其他指令則將br\_op設為都不會branch。

rs1與rs2的idx送往regfile，並在posedge時拿到值傳往EX stage。

當hazard == 1時，會flush掉各register的值(產生bubble)

1. stage\_EX.sv

(架構圖在下頁)

rs1, rs2 mux：以從forward controller傳來的rs\_ctrl訊號，決定rs值為由ID而來、或是需從前面指令的EX或MEM forward過來。

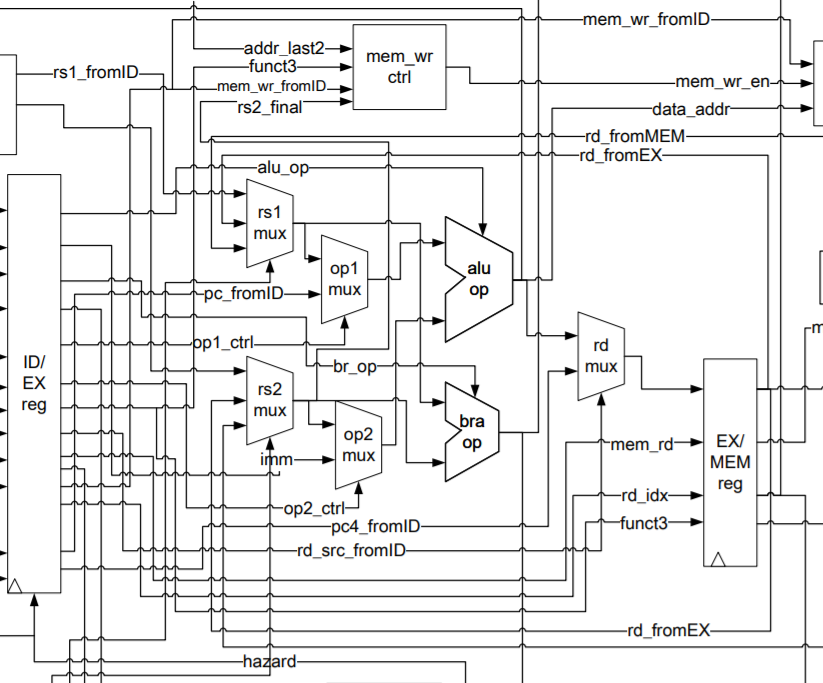
op1, op2 mux：由op\_ctrl訊號，決定op1為rs1或pc，op2為rs2或imm。

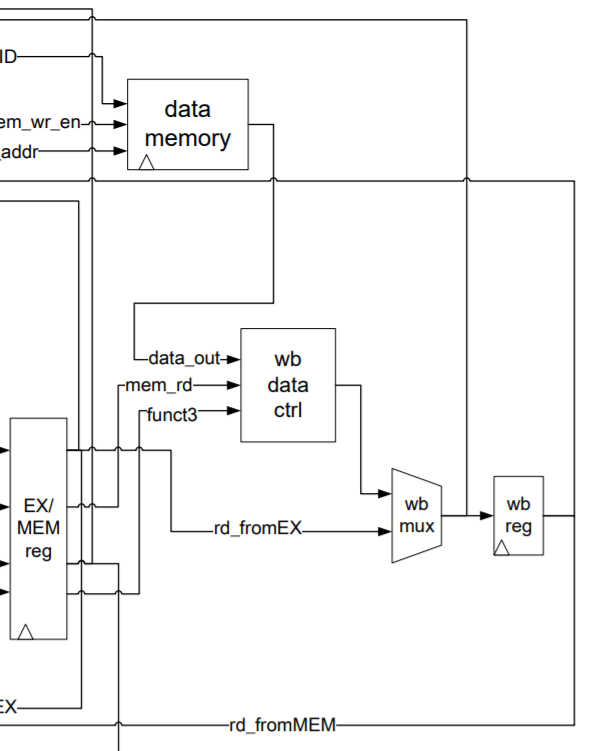
alu：由alu\_op決定為何種運算。

rd\_mux：由rd\_src訊號決定rd為alu運算結果還是pc4。

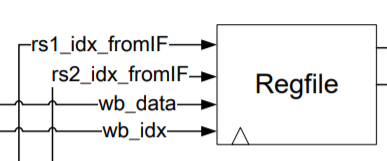
branch：根據br\_op訊號以及比較結果決定br\_take的值。

mem\_wr\_ctrl：控制store相關的訊號，根據funct3與alu\_res最後兩bit決定wr\_mem\_en的值，以及是否要針對rs2做shift。並將這些訊號以及mem\_addr送到data memory。



1. stage\_MEM\_and\_WB.sv

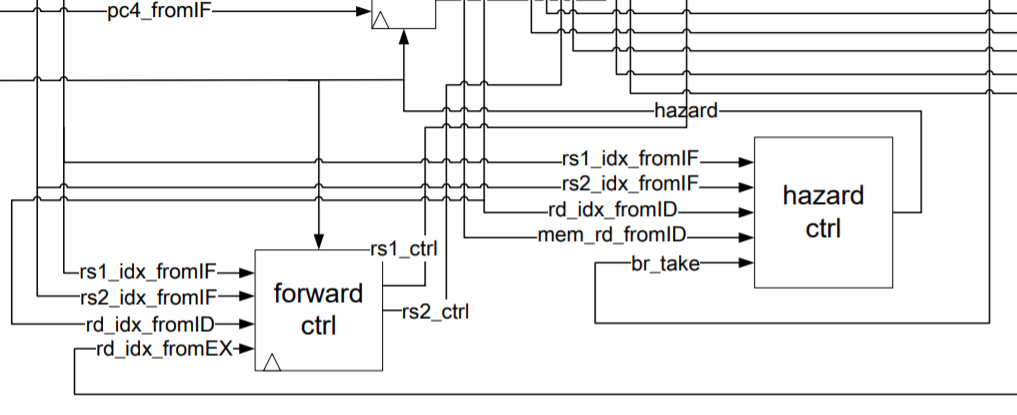
wb\_data\_ctrl：控制write back的值。此時load類指令已拿到從memory來的資料，根據funct3處理LH,LB的部分。若資料不是來自memory，則wb\_data為EX運算結果。

在下一個posedge，wb\_data會寫回regfile對應的wb\_idx，指令完成。

1. forward\_and\_hazard.sv

forward ctrl：判斷ID的rs1, rs2 idx是否有與EX或MEM的rd idx相同，並將rs\_ctrl訊號送至EX，決定rs1與rs2的資料來源。

hazard ctrl：判斷ID的rs1, rs2是否與EX的rd相同，且mem\_read為1；或是branch take若是，則hazard拉為1，flush IF和ID stage的值。



1. 波型圖

以下皆以prog0之main.S的指令做舉例

1. R-type
2. add

130: 006282b3 add t0,t0,t1

IF：pc = 130 ID：instr = 006282b3

EX： op1 = ffff\_ffff, op2 = ffff\_ffff, alu\_op = 0000

alu\_res= ffff\_ffff + ffff\_ffff = ffff\_fffe



1. slt

1e4: 0062a2b3 slt t0,t0,t1

IF：pc = 1e4 ID：instr = 0062a2b3

EX： op1 = ffff\_ffff, op2 = 1, alu\_op = 0010

alu\_res = (ffff\_ffff < 1) ? (signed) = 1

一張含有 文字, 時鐘, 時間, 已掛上 的圖片

自動產生的描述

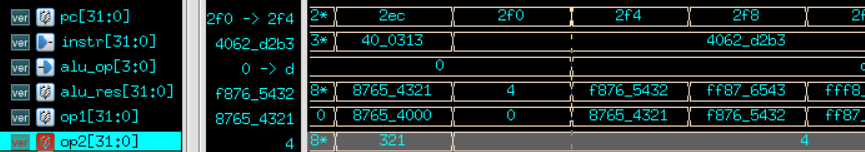
1. sra

2ec: 4062d2b3 sra t0,t0,t1

IF：pc = 2ec ID：instr = 4062d2b3

EX：op1 = 8765\_4321, op2 = 4, alu\_op = 1101

alu\_res = 8765\_4321 >> 4 (signed) = f876\_5432



1. I-type
2. addi

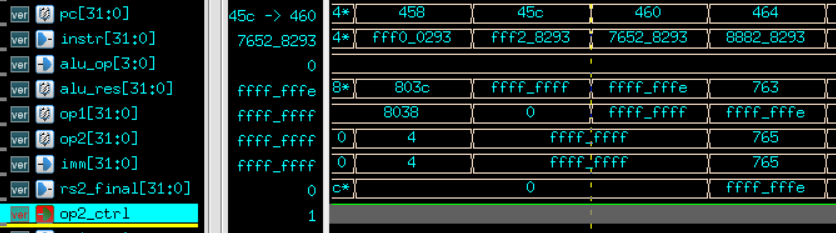
458: fff28293 addi t0,t0,-1

IF：pc = 458 ID：instr = fff28293

EX：op1 = ffff\_fffe, op2 = ffff\_ffff, alu\_op = 0000

其中op2\_ctrl為選擇來源為rs2或imm，在addi指令中選擇imm

alu\_res = ffff\_ffff + ffff\_ffff = ffff\_fffe



1. lw

3b8: 00032283 lw t0,0(t1)

IF：pc = 3b8 ID：instr = 00032283

EX：alu\_res = b30, mem\_addr = alu\_res[15:2] = 2cc

MEM：mem\_rd\_fromEX = 1 (表示對memory讀取), mem\_read\_out = 6666\_6666

wb\_data = 6666\_6666, wb\_idx = 5

WB： regfile [5] = 6666\_6666

一張含有 文字, 監視器, 陳列, 數個 的圖片

自動產生的描述

1. jal

72c: 00030367 jalr t1,t1

IF：pc = 72c ID：0080036f

EX： br\_take = 1, alu\_res = rs1 + imm = b04 + 8 = 744

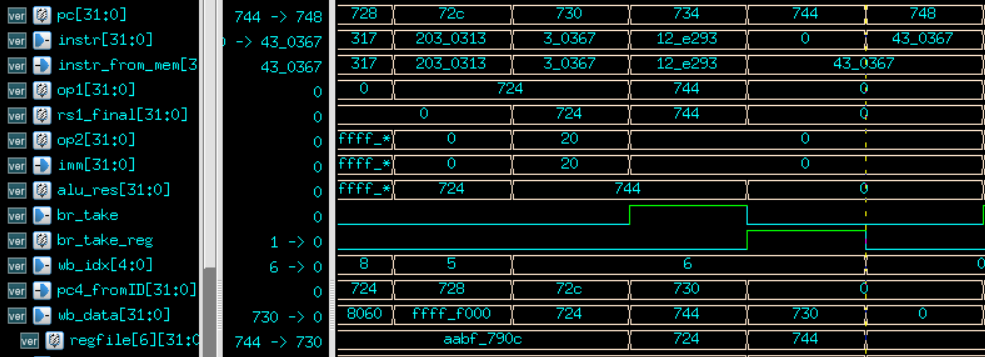
next cycle：pc = 744, instr\_from\_mem為pc=734對應的錯誤指令，由於br\_take\_reg = 1，instr\_mux會將instr設為0，產生bubble

此外ID stage register也會flush,共產生2個bubble

wb\_idx = 6, wb\_data = pc4\_fromID(last cycle) = 730

next cycle：instr為pc=744對應的正確指令，回歸正常執行。

regfile [6] = 730



1. S-type
2. sw

7c4: fe542e23 sw t0,-4(s0)

IF：pc = 7c4 ID：instr = fe542e23

EX：alu\_res = rs1 + imm = 8078 + ffff\_fffc = 8074

funct3 = 2, WEB = 0000, mem\_addr = alu\_res[15:2] = 201d = A,

rs2\_final = f = DI

在EX/MEM的posedge clk，資料寫入DM

一張含有 文字, 計分板 的圖片

自動產生的描述

1. sb

7f0: ffe409a3 sb t5,-13(s0)

IF：pc = 7f0 ID：instr = ffe409a3

EX：alu\_res = rs1 + imm = 808c + ffff\_fff3 = 807f

funct3 = 0, alu\_res[1:0] = 11, WEB = 0111 (store在高位btye)

mem\_addr = alu\_res[15:2] = 201f = A, rs2\_final = 1234\_5678 = DI

在EX/MEM的posedge clk，資料寫入DM

一張含有 文字, 計分板 的圖片

自動產生的描述

1. sh

7ec: ffe41a23 sh t5,-12(s0)

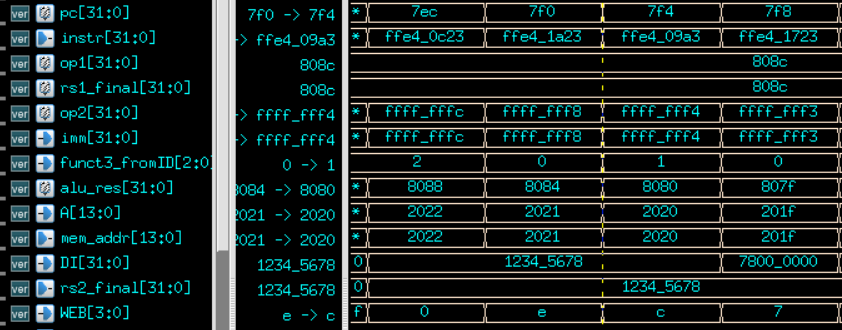
IF：pc = 7ec ID：instr = ffe41a23

EX：alu\_res = rs1 + imm = 808c + ffff\_fff4 = 8080

funct3 = 1, alu\_res[1:0] = 0, WEB = 1001 (store在中間half)

mem\_addr = 2020 = A, rs2\_final = 1234\_5678 = DI

在EX/MEM的posedge clk，資料寫入DM



1. B-type
2. beq

85c: 00628463 beq t0,t1,864 <beq+0x58>

IF：pc = 85c ID：instr = 00628463

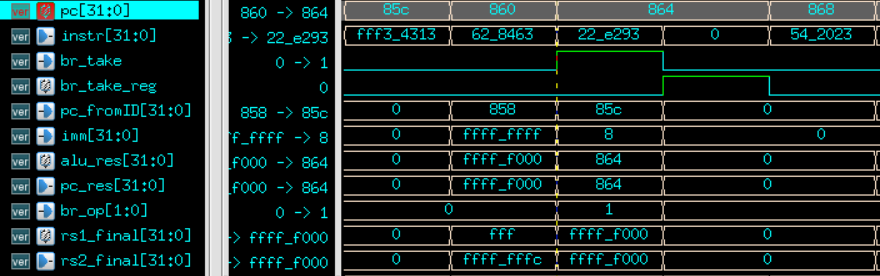
EX：br\_take = (rs1 == rs2)? = ffff\_f000 == ffff\_f000 ? = 1

pc\_res = alu\_res = pc + imm = 85c + 8 = 864

next cycle：pc = 864, instr 為錯誤指令，flush ID => bubble

br\_take\_reg = 1，instr mux將指令設為0 => bubble(共2個bubble)

next cycle：instr為正確地址之指令，回歸正常執行



1. blt

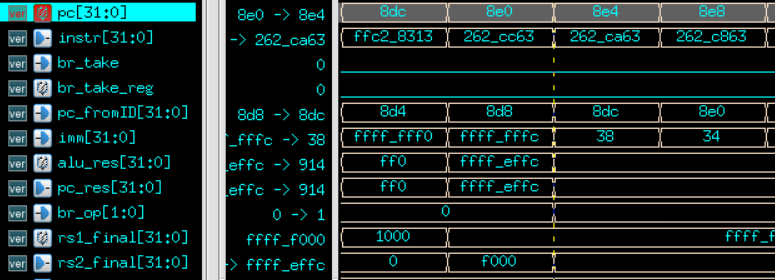
8dc: 0262cc63 blt t0,t1,914 <blt+0x48>

IF：pc = 8dc ID：instr = 0262cc63

EX：br\_take = (rs1s < rs2s)? = ffff\_f000 < ffff\_effc ? = 0

pc\_res = alu\_res = pc + imm = 8dc + 38 = 914

next cycle：pc沒有變為914，繼續原本的pc + 4



1. bltu

a18: 0062e463 bltu t0,t1,a20 <bltu+0x50>

IF：pc = a18 ID：instr = 0062e463

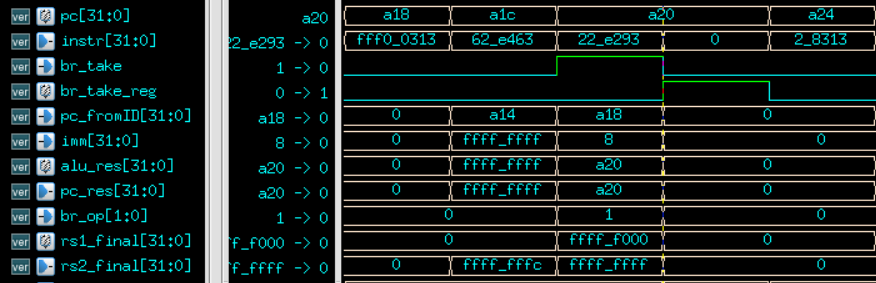
EX：br\_take = (rs1u < rs2u)? = ffff\_f000 < ffff\_ffff ? = 1

pc\_res = alu\_res = pc + imm = a18 + 8 = a20

next cycle：pc = a20(剛好與上一cycle一樣), flush ID => bubble

br\_take\_reg = 1，instr mux將指令設為 0 => bubble

next cycle：instr為正確地址之指令，回歸正常執行



1. U-type
2. auipc

7c: 00009197 auipc gp,0x9

IF：pc = 7c ID：instr = 00009197

EX：alu\_res = pc + imm = 7c + 9000 = 907c

MEM：wb\_data = 907c, wb\_idx = 3

WB：regfile[3] = 907c

一張含有 文字, 陳列 的圖片

自動產生的描述

1. lui

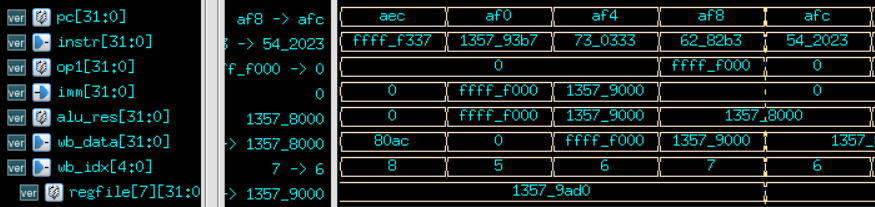
aec: 135793b7 lui t2,0x13579

IF：pc = aec ID：instr = 135793b7

EX：alu\_res = imm = 1357\_9000

MEM：wb\_data = 1357\_9000, wb\_idx = 7

WB：regfile[7] = 1357\_9000



1. J-type

98: 06c000ef jal ra,104 <fill\_block>

IF：pc = 98 ID：06c000ef

EX：br\_take = 1, alu\_res = pc + imm = 98 + 6c = 104

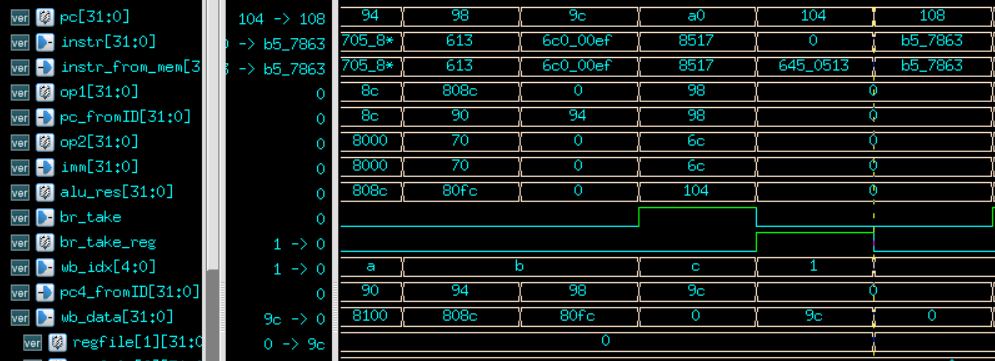
next cycle：pc = 104, instr 為錯誤指令，flush ID => bubble

wb\_idx = 1, wb\_data = pc4\_fromID(last cycle) = 9c

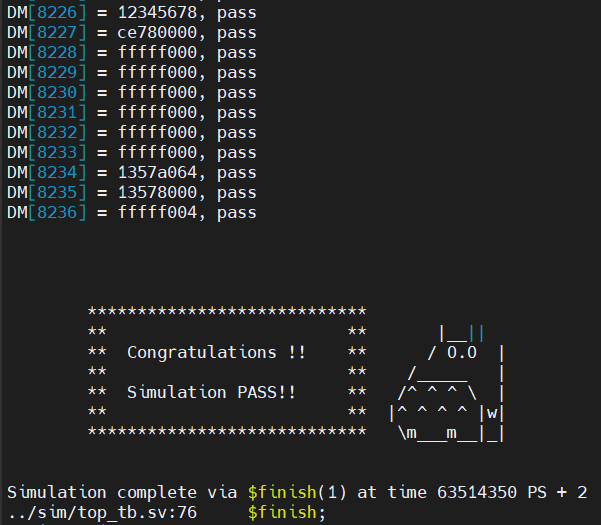
br\_take\_reg = 1，instr mux將指令設為0 => bubble

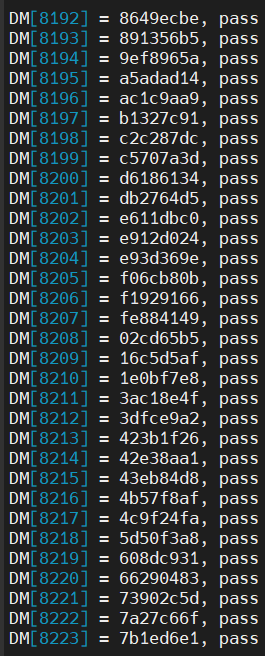
regfile [6] = 9c

next cycle：instr為正確地址之指令，回歸正常執行



1. test program
2. prog0



1. prog1

一開始我先嘗試用c實現insertion sort，但在邊譯成assembly後，我發現會多很多不必要的指令，因此我就改成用assembly實現。在演算法的部分，雖然insertion sort不是時間複雜度最小的，但是它運算簡單且不需要呼叫函式，可以減少函式呼叫的資料搬運，因此選擇該算法。

結果說明：資料由最小的8649ecbe(負數)，依序排到最大7b1ed6e1(最大)

一張含有 文字 的圖片

自動產生的描述

1. prog2

使用assembly實現，由於目前硬體沒有乘法指令，因此在軟體部分以Shift-and-Add實現乘法運算。

結果說明：lower part存在DM[8192]，higher part在DB[8193]

1a2b3c4d × 98765432 = f95c456cfb90b0a

一張含有 文字 的圖片

自動產生的描述

1. prog3

使用輾轉相除法找尋最大公因數，同樣是使用aseembly。

結果說明：0x07622814和0x00421923的最大公因數為3

一張含有 文字 的圖片

自動產生的描述

1. PA

cycle：合成與驗證皆為8.7

一張含有 文字 的圖片

自動產生的描述

area：5527032

一張含有 文字 的圖片

自動產生的描述

timing：

(已於前面有截圖)

prog0+prog1+prg2+prg3 = 63514350+77712750+64010250+60051750 = 265289100ps

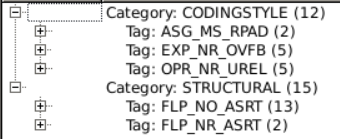
1. superlint

total line：845

superlint violation：27

27/845 = 3.2% < 15%

一張含有 文字 的圖片

自動產生的描述 

最多的warning：

1. no asynchronous reset：可是教授在上課說盡量使用同步reset，所以就還是使用synchronous reset

一張含有 文字 的圖片

自動產生的描述

1. shift overflow：不過overflow的bits也不影響運算結果，因此沒有修正

一張含有 文字 的圖片

自動產生的描述

1. unequal lengths operands：在部分地方判斷值是否大於0時，沒有specify 0的長度。



1. Summary and Lessons learned

在硬體部分，這次的作業讓我更熟悉pipeline CPU的電路實作。由於SRAM在下一個cycle才會拿到資料，因此各訊號在時序上需要多做考慮。此外最困難的部分是當有hazard或是take branch時要產生bubble，需如何flush相關訊號。一開始構思整體架構時也是困難重重，哪些要組合電路哪些要時序等等的問題都很需要思考。

在cycle的部分很幸運能夠合到8.7，其中critical path是指令從IM出來後經過decode，將rs\_idx送往regfile取出對應值的部分，似乎跟其他人卡在EX的狀況很不一樣。我想這是因為SRAM資料出來會有delay，而在對regfile取值時又要經過好幾層的mux，需要花較多時間。在下次作業中，我可能會嘗試將ID再切成2個stage，看能不能讓cycle下降更多。

而在軟體部分，我更加熟悉assembly的撰寫，雖然寫組語比C還要燒腦很多，但是在運算時間上，直接寫assembly真的可以比C還要快不少。此外，我也更熟悉了shift-add multiplication 的演算法。

總之，這份作業雖然花了很多時間，但過程中讓我受益良多，接下來的作業也要繼續努力。