VLSI System Design (Graduate Level)
Fall 2021

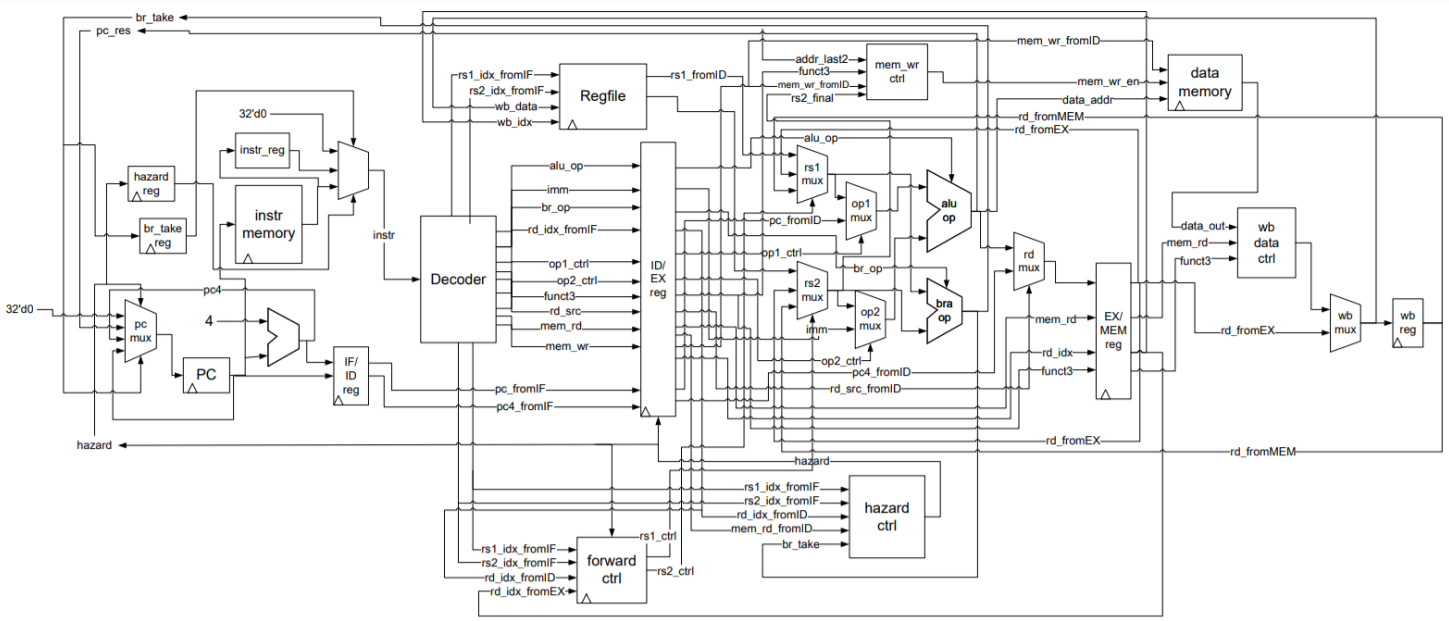
HOMEWORK I
REPORT

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__

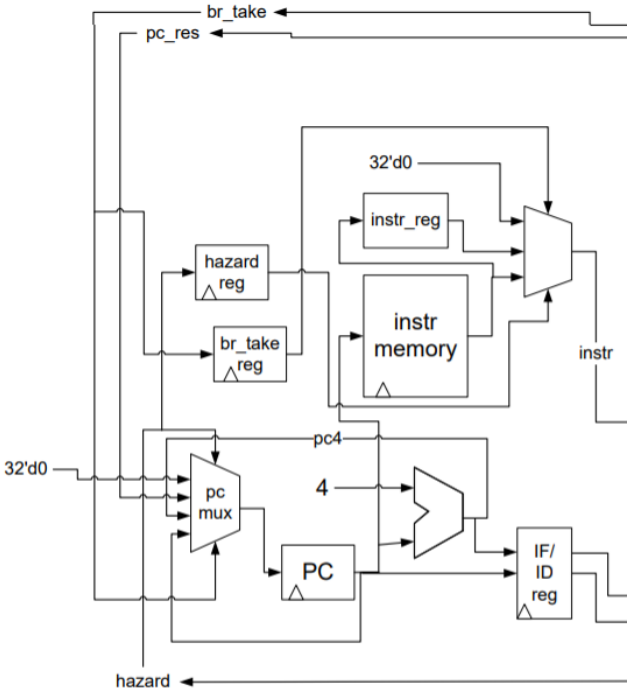
一、 架構設計

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



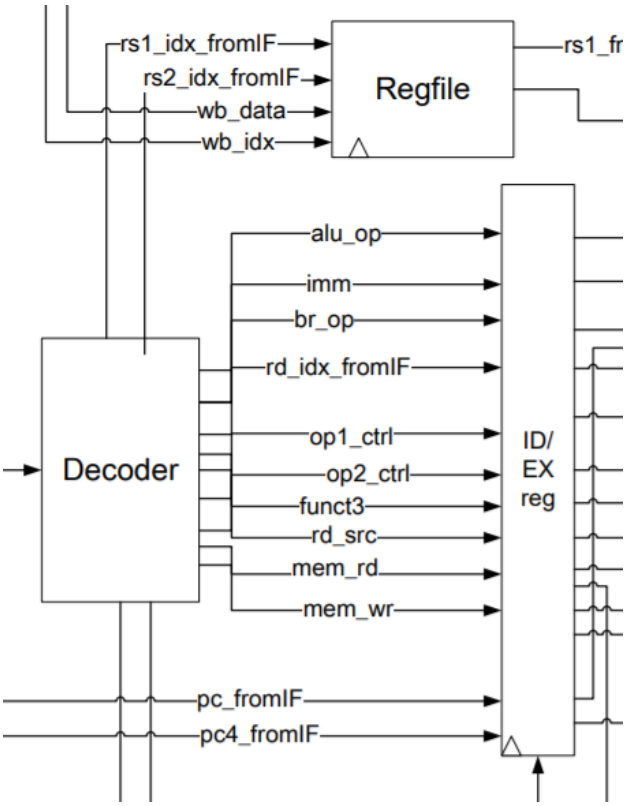
pc：在 posedge 時會改變值，普通情況下 pc = pc+4；br_take == 1 -> pc = pc_res，其中 pc_res 為 EX 中 alu 的運算結果；當 hazard == 1(load 的值在下一 cycle 就要用到時，hazard 會拉成 1)，pc = pc(產生 1 個 bubble)

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。

2. 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)

3. 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。

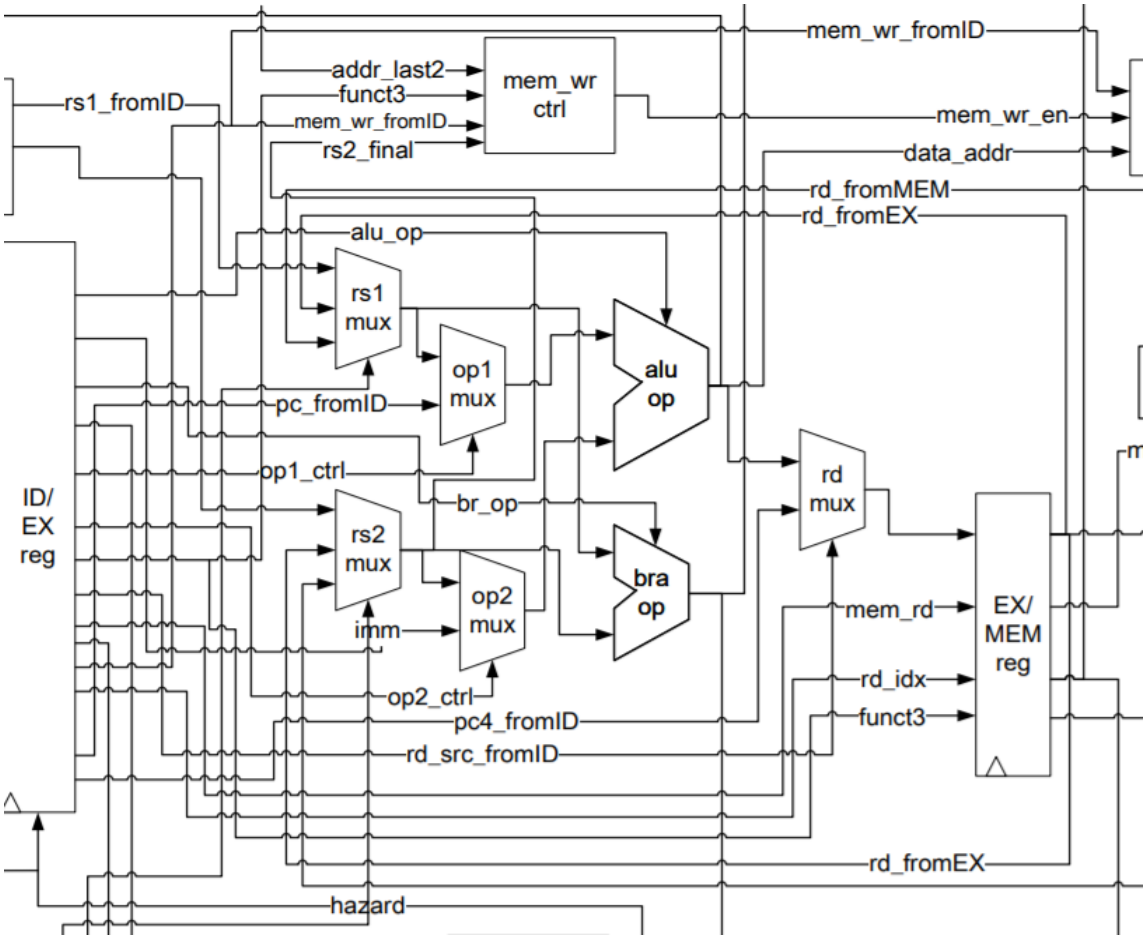
op1, op2 mux：由 op_ctrl 訊號，決定 op1 為 rs1 或 pc，op2 為 rs2 或 imm。

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。

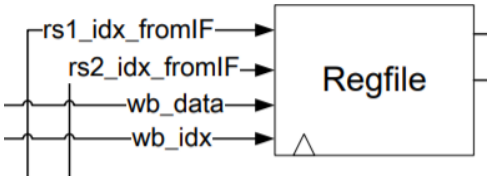
mem_wr_ctrl：控制 store 相關的訊號，根據 funct3 與 alu_res 最後兩 bit 決定 wr_mem_en 的值，以及是否要針對 rs2 做 shift。並將這些訊號以及 mem_addr 送到 data memory。

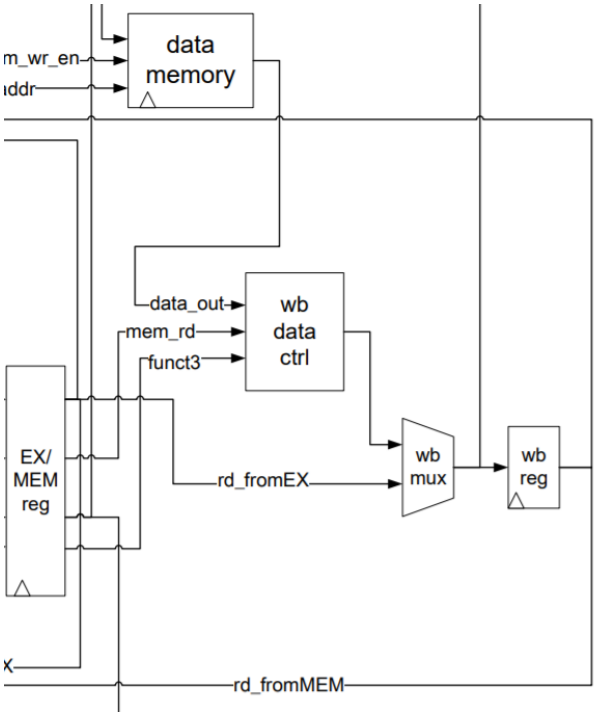


4. stage_MEM_and_WB.sv

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。

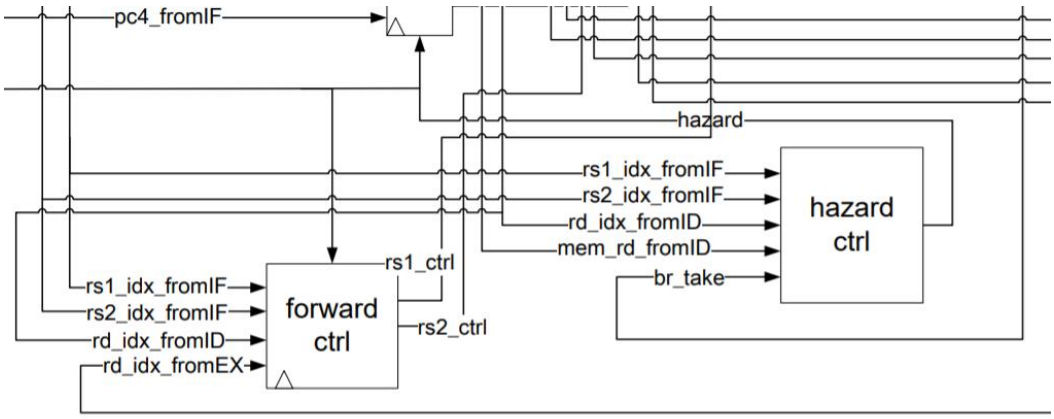




5. forward_and_hazard.sv

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。



二、 波型圖

以下皆以 prog0 之 main.S 的指令做舉例

1. R-type

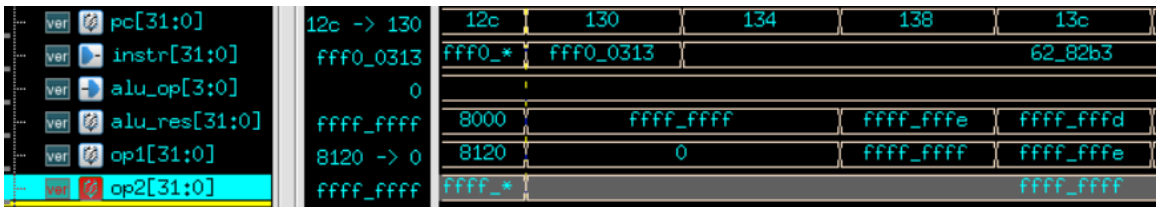
(a) add

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



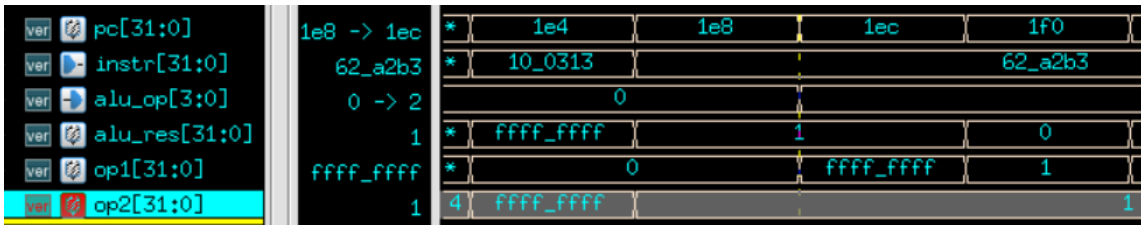
(b) slt

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



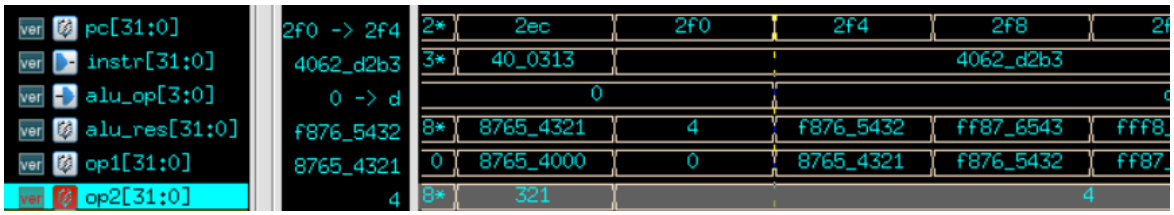
(c)　sra

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



2. I-type

(a)　addi

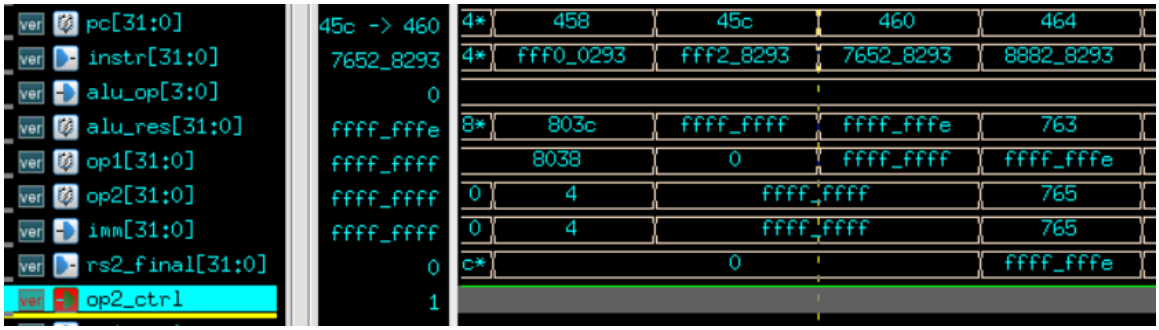
458: fff28293 addi t0,t0,-1

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



(b)　lw

3b8: 00032283 lw t0,0(t1)

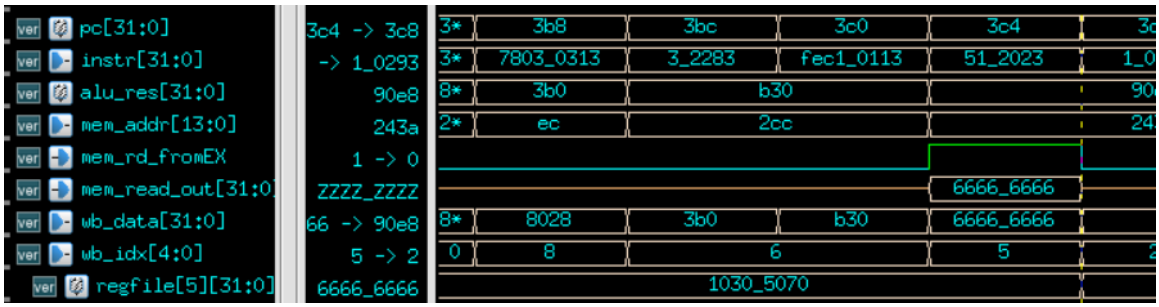
IF：pc = 3b8　　ID：instr = 00032283

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



(c)　jal

72c: 00030367 jalr t1,t1

IF：pc = 72c　　ID：0080036f

EX： br_take = 1, alu_res = rs1 + imm = b04 + 8 = 744

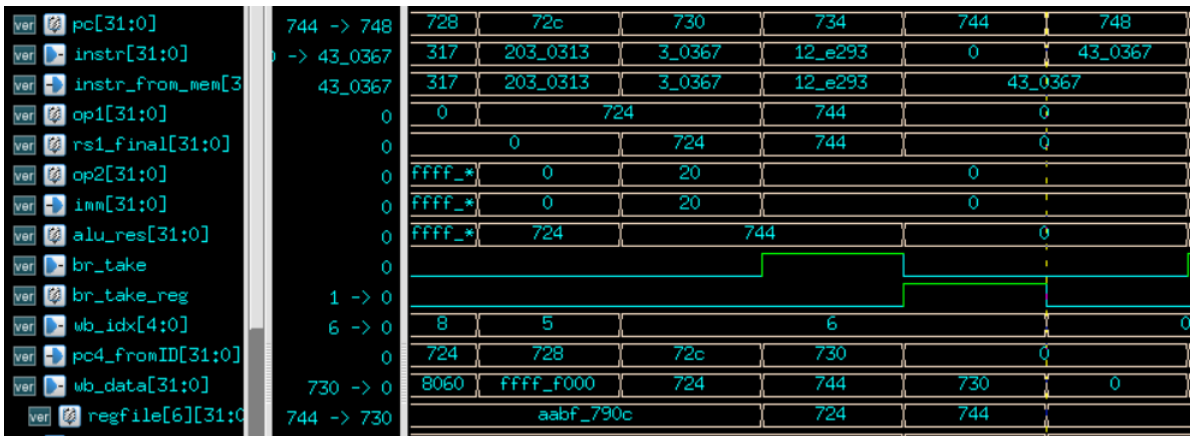
next cycle：pc = 744, instr_from_mem 為 pc=734 對應的錯誤指令，由於 br_take_reg = 1，instr_mux 會將 instr 設為 0，產生 bubble

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



3. S-type

(a)　sw

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

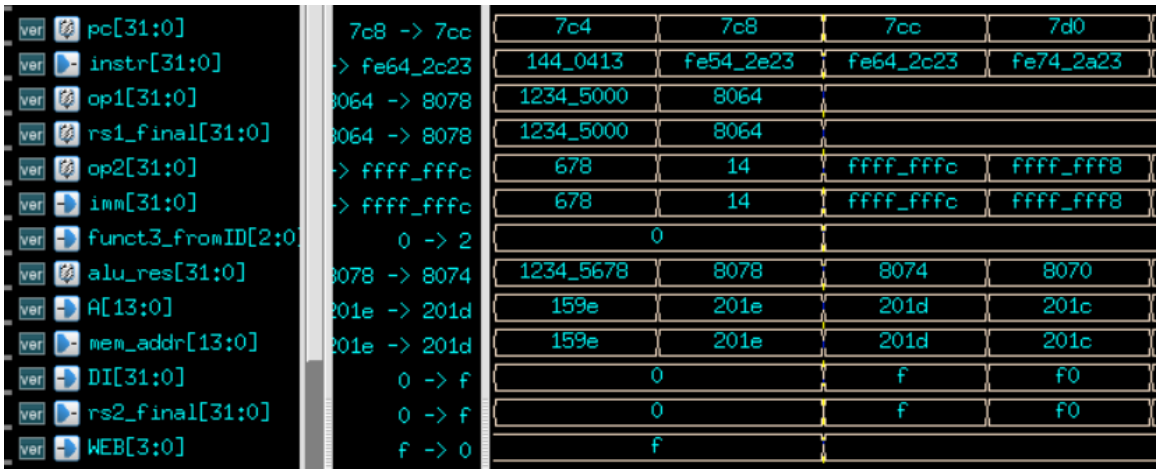
IF：pc = 7c4　　ID：instr = fe542e23

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



(b)　sb

7f0: ffe409a3　sb t5,-13(s0)

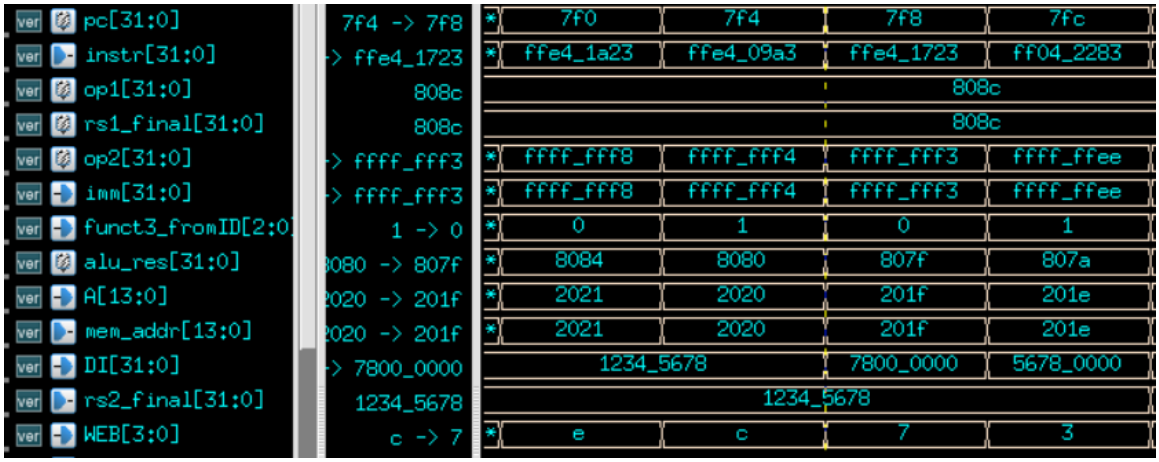
IF：pc = 7f0　　ID：instr = ffe409a3

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



(c) sh

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

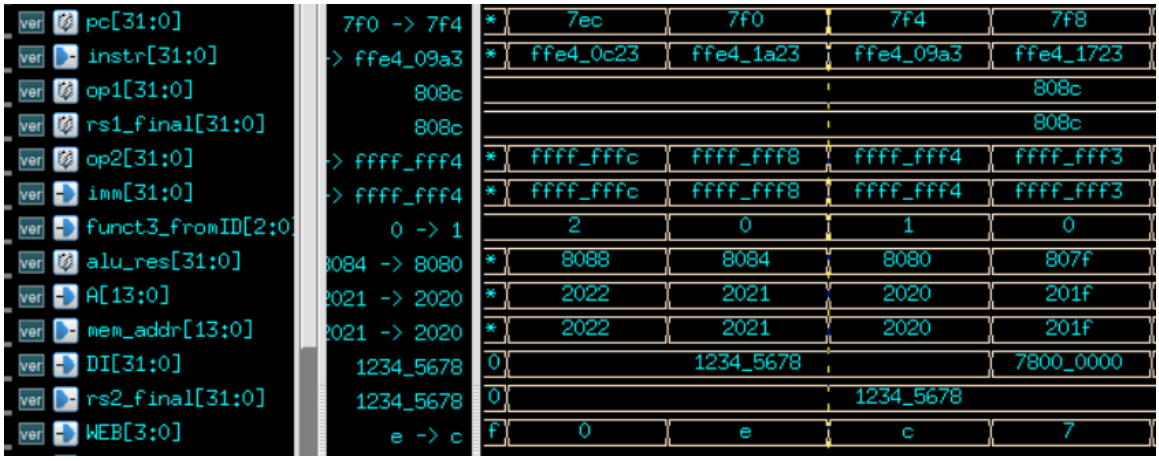
IF：pc = 7ec ID：instr = ffe41a23

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



4. B-type

(a) beq

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

IF：pc = 85c ID：instr = 00628463

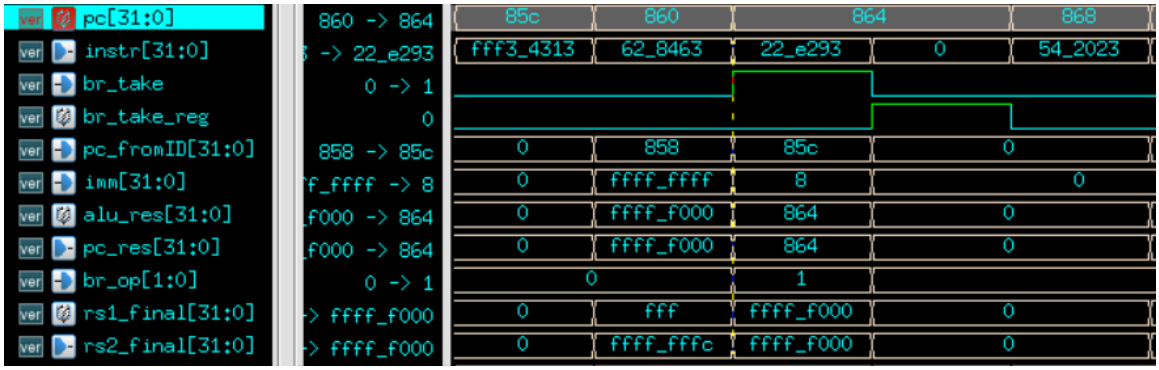
EX：br_take = (rs1 == rs2)? = ffff_f000 == ffff_f000 ? = 1

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



(b)　blt

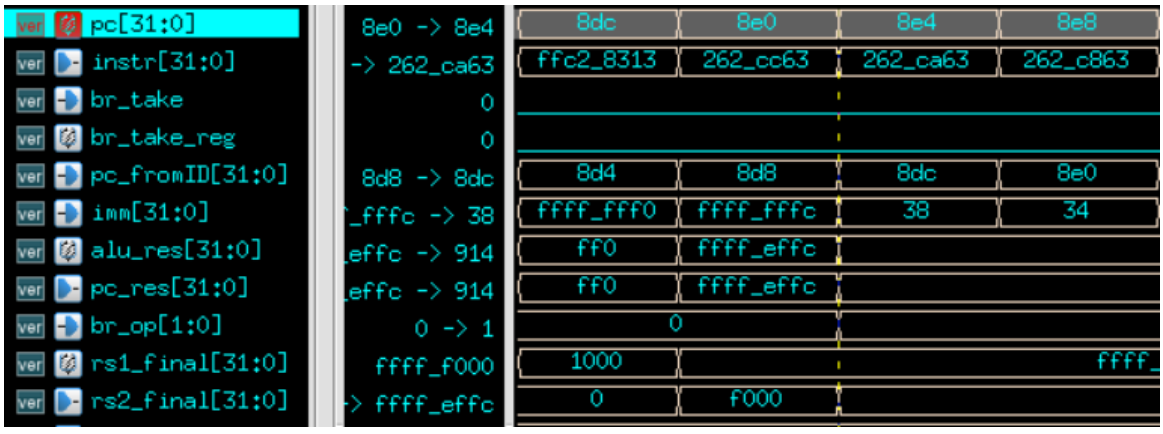
8dc: 0262cc63　blt t0,t1,914 <blt+0x48>

IF：pc = 8dc　　ID：instr = 0262cc63

EX：br_take = (rs1$_s$ < rs2$_s$)? = 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



(c)　bltu

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

IF：pc = a18　　ID：instr = 0062e463

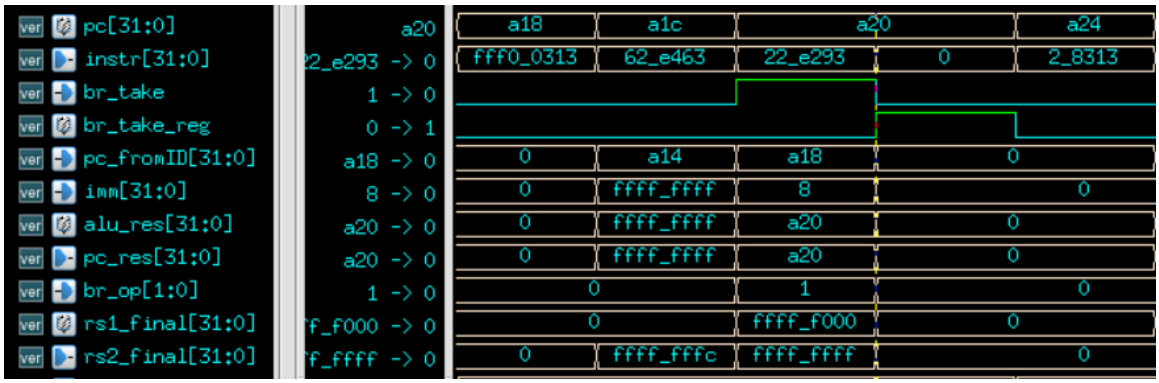
EX：br_take = (rs1$_u$ < rs2$_u$)? = ffff_f000 < ffff_ffff ? = 1

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



5. U-type

(a)　auipc

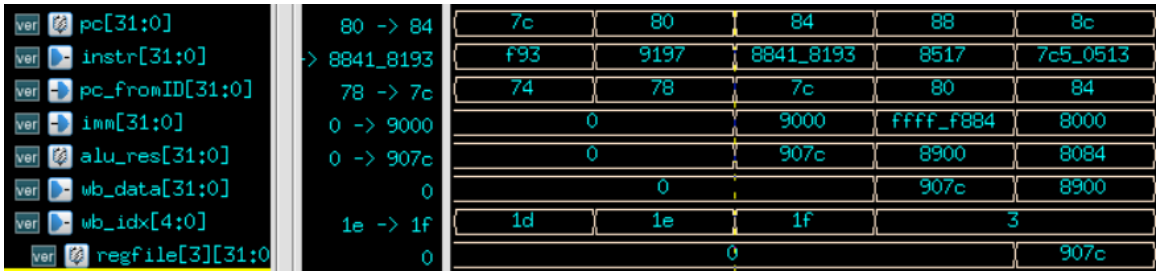
7c: 00009197　auipc gp,0x9

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



(b)　lui

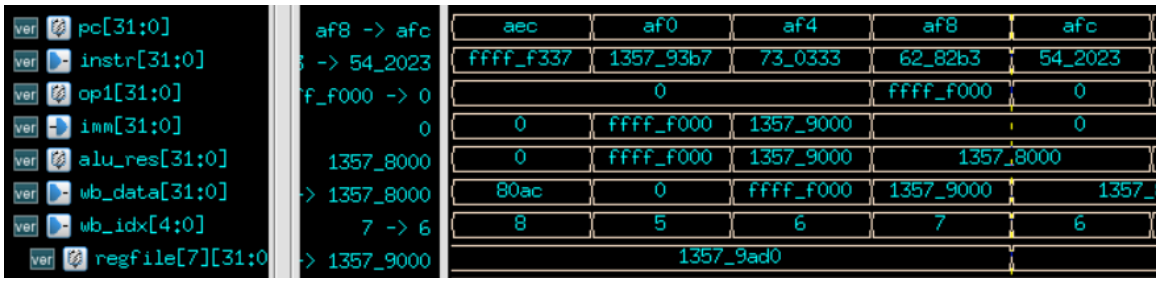
aec: 135793b7 lui t2,0x13579

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



6. 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

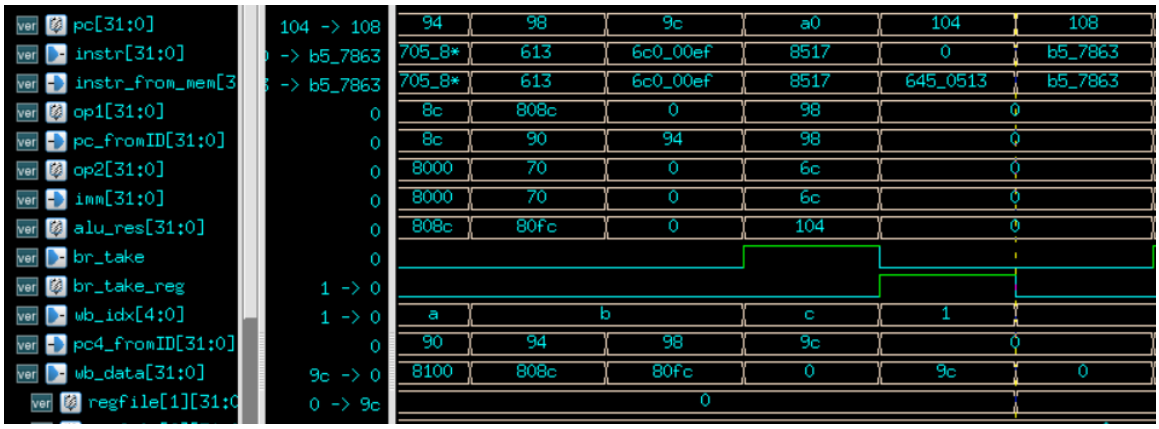
next cycle：pc = 104, instr 為錯誤指令，flush ID => bubble

wb_idx = 1, wb_data = pc4_fromID(last cycle) = 9c

br_take_reg = 1，instr mux 將指令設為 0 => bubble

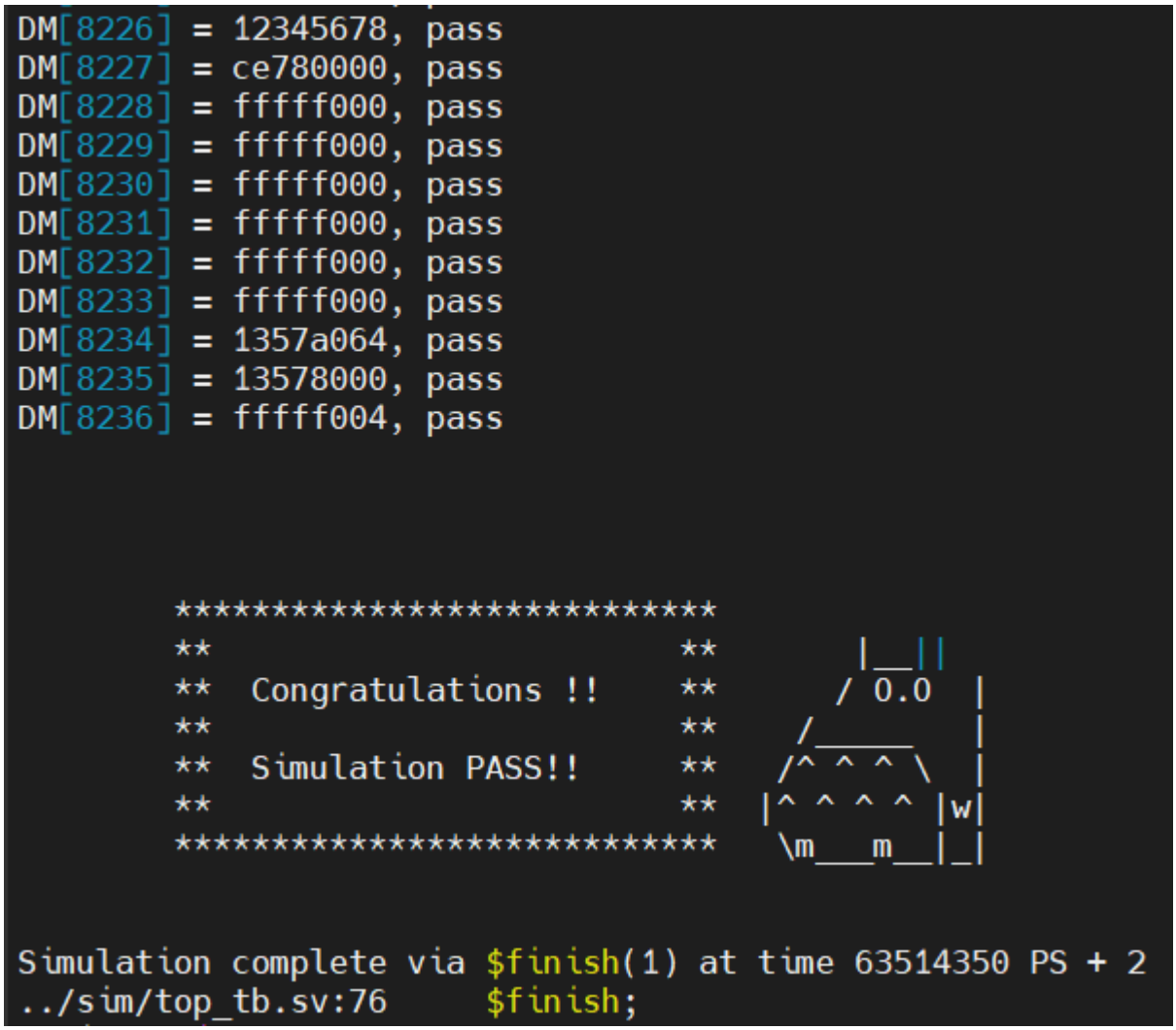
regfile [6] = 9c

next cycle：instr 為正確地址之指令，回歸正常執行



三、 test program

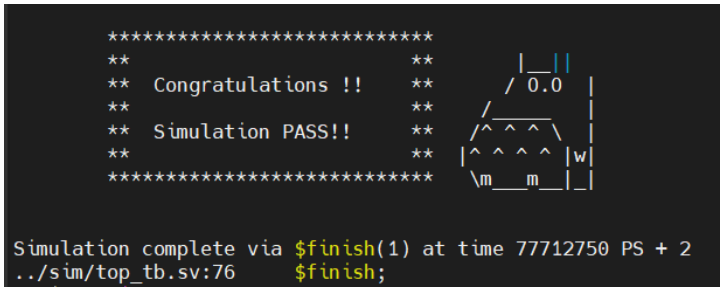
1. prog0



2. 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)




```
DM[8192] = 8649ecbe, pass
DM[8193] = 891356b5, pass
DM[8194] = 9ef8965a, pass
DM[8195] = a5adad14, pass
DM[8196] = ac1c9aa9, pass
DM[8197] = b1327c91, pass
DM[8198] = c2c287dc, pass
DM[8199] = c5707a3d, pass
DM[8200] = d6186134, pass
DM[8201] = db2764d5, pass
DM[8202] = e611dbc0, pass
DM[8203] = e912d024, pass
DM[8204] = e93d369e, pass
DM[8205] = f06cb80b, pass
DM[8206] = f1929166, pass
DM[8207] = fe884149, pass
DM[8208] = 02cd65b5, pass
DM[8209] = 16c5d5af, pass
DM[8210] = 1e0bf7e8, pass
DM[8211] = 3ac18e4f, pass
DM[8212] = 3dfce9a2, pass
DM[8213] = 423b1f26, pass
DM[8214] = 42e38aa1, pass
DM[8215] = 43eb84d8, pass
DM[8216] = 4b57f8af, pass
DM[8217] = 4c9f24fa, pass
DM[8218] = 5d50f3a8, pass
DM[8219] = 608dc931, pass
DM[8220] = 66290483, pass
DM[8221] = 73902c5d, pass
DM[8222] = 7a27c66f, pass
DM[8223] = 7b1ed6e1, pass
```

3. 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

```
DM[8192] = cfb90b0a, pass
DM[8193] = f56a8809, pass



        ****************************
        **                        **       |__||
        **  Congratulations !!    **      / 0.0  |
        **                        **    /_____   |
        **  Simulation PASS!!     **   /^ ^ ^ \  |
        **                        **  |^ ^ ^ ^ |w|
        ****************************   \m___m__|_|


Simulation complete via $finish(1) at time 64010250 PS + 2
../sim/top_tb.sv:76     $finish;
```

4. 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

```
DM[8192] = 00000003, pass



        ****************************
        **                        **       |__||
        **  Congratulations !!    **      / 0.0  |
        **                        **    /_____   |
        **  Simulation PASS!!     **   /^ ^ ^ \  |
        **                        **  |^ ^ ^ ^ |w|
        ****************************   \m___m__|_|


Simulation complete via $finish(1) at time 60051750 PS + 2
../sim/top_tb.sv:76     $finish;
```

四、 PA

cycle：合成與驗證皆為 8.7

```
#####CLK PERIOD CAN BE ADJUSTED UP TO 20.0 IF SYNTHESIS GOES
create_clock -name clk -period 8.7 [get_ports clk]
set_dont_touch_network      [all_clocks]
set_fix_hold                [all_clocks]
set_clock_uncertainty  0.1  [all_clocks]
set_clock_latency      1.0  [all_clocks]
set_ideal_network           [get_ports clk]

#####REMEMBER TO SET THIS MAX DELAY TO 1/2 CLK PERIOD#####
set_input_delay  -max 4.35   -clock clk [remove_from_collect
set_input_delay  -min 0.0    -clock clk [remove_from_collect
#set_output_delay  -max 5.0   -clock clk [all_outputs]
#set_output_delay  -min 0.0   -clock clk [all_outputs]
```

area：5527032

```
Number of ports:                         1862
Number of nets:                          9171
Number of cells:                         7289
Number of combinational cells:           5847
Number of sequential cells:              1426
Number of macros/black boxes:               2
Number of buf/inv:                       1333
Number of references:                      12

Combinational area:             105415.127974
Buf/Inv area:                    14911.545463
Noncombinational area:           77123.189804
Macro/Black Box area:          5344494.500000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:               5527032.817778
Total area:                 undefined
```

timing：

（已於前面有截圖）

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

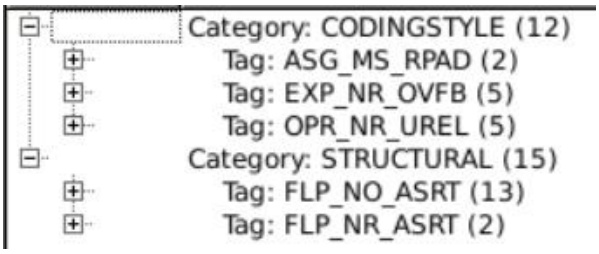
五、 superlint

total line：845

superlint violation：27

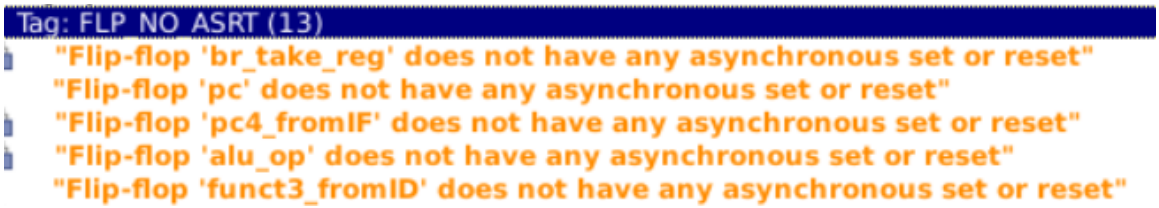
27/845 = 3.2% < 15%

```
 45 src/forward_and_hazard.sv
 99 src/SRAM_wrapper.sv
194 src/stage_EX.sv
162 src/stage_ID.sv
 63 src/stage_IF.sv
 44 src/stage_MEM_and_WB.sv
147 src/top.sv
 91 include/all_def.svh
845 total
```



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



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

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

六、 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。