

# VLSI System Design (Graduate Level)

## Fall 2021

### HOMEWORK I REPORT

Must do self-checking before submission:

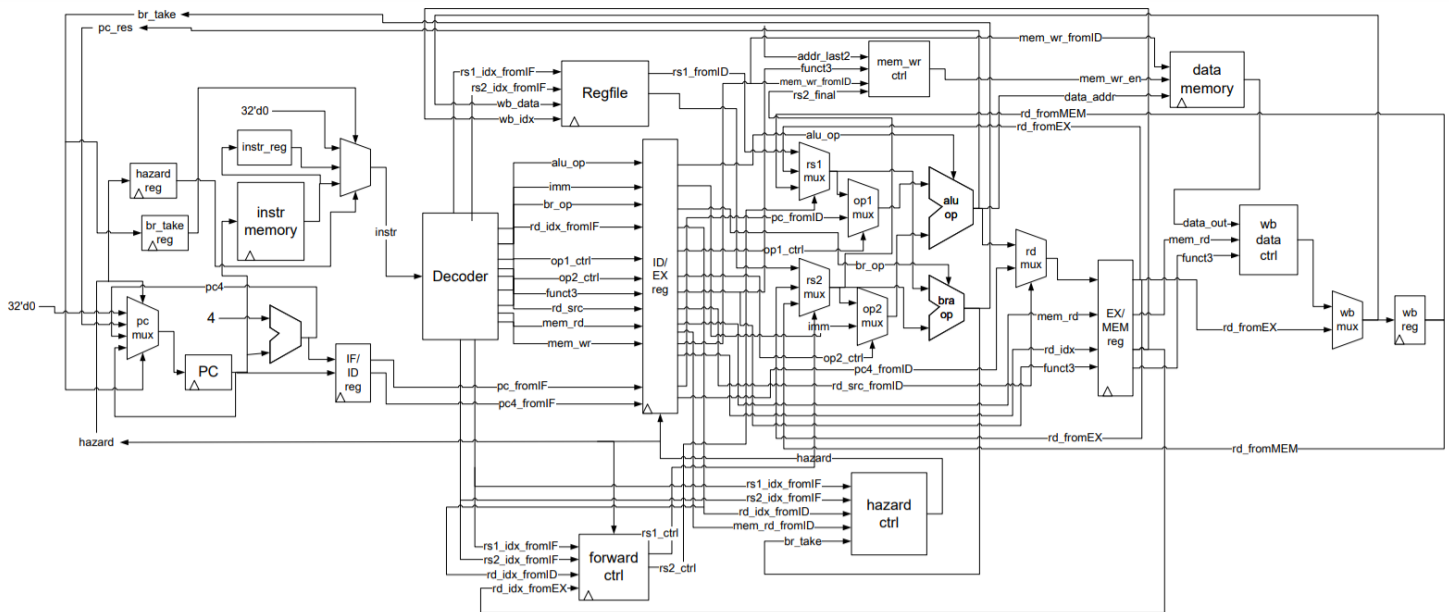
- ☒ Compress all files described in the problem into one tar
- ☒ All SystemVerilog files can be compiled under SoC Lab environment
- ☒ All port declarations comply with I/O port specifications
- ☒ Organize files according to File Hierarchy Requirement
- ☒ No any waveform files in deliverables

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

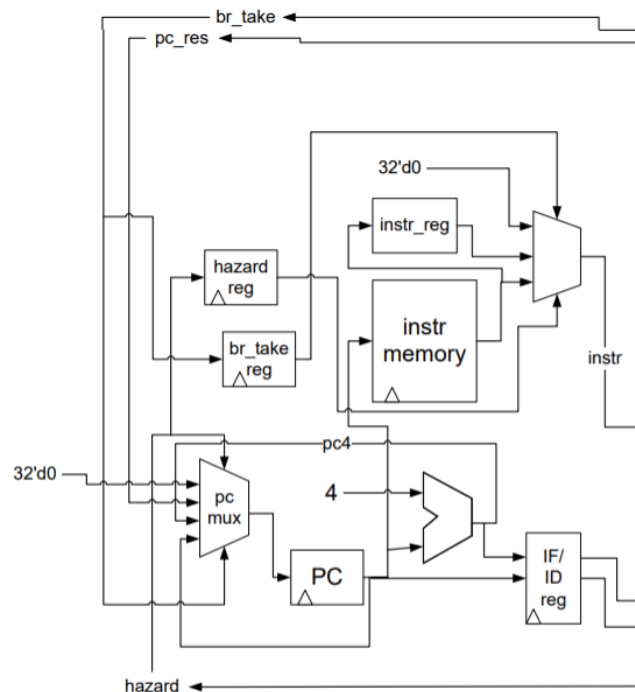
pc：在 posedge 時會改變值，普通情況下  $pc = pc + 4$ ； $br\_take == 1 \rightarrow pc = pc\_res$ ，其中  $pc\_res$  為 EX 中 alu 的運算結果；當  $hazard == 1$  (load 的值在下一 cycle 就要用到時，hazard 會拉成 1)， $pc = pc$  (產生 1 個 bubble)

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

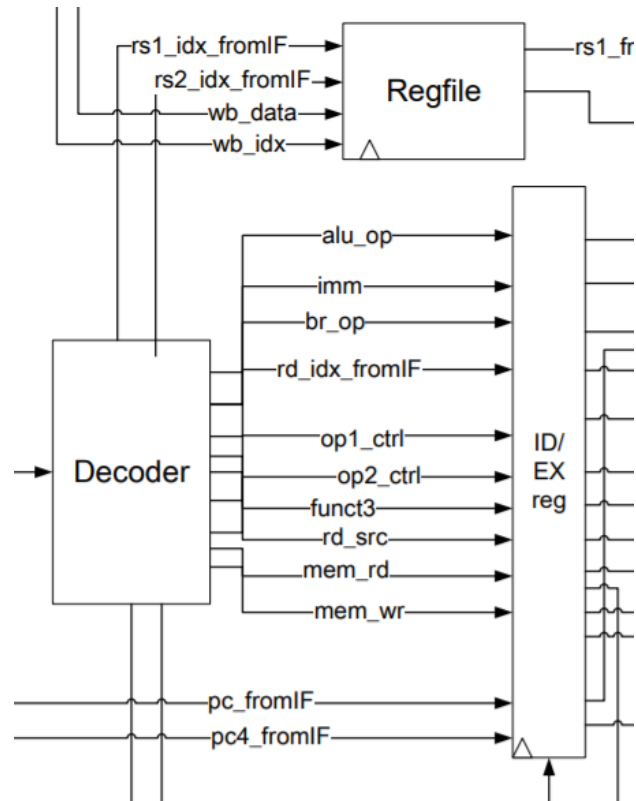
以 opcode、funct3、funct7 第 5 碼的值判斷為何種指令，決定各訊號如 alu\_op, br\_op, op\_ctrl, mem\_wr, mem\_rd, imm，並連同其他後面 stage 需用到的訊號，在 posedge 時傳至 EX stage。

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。

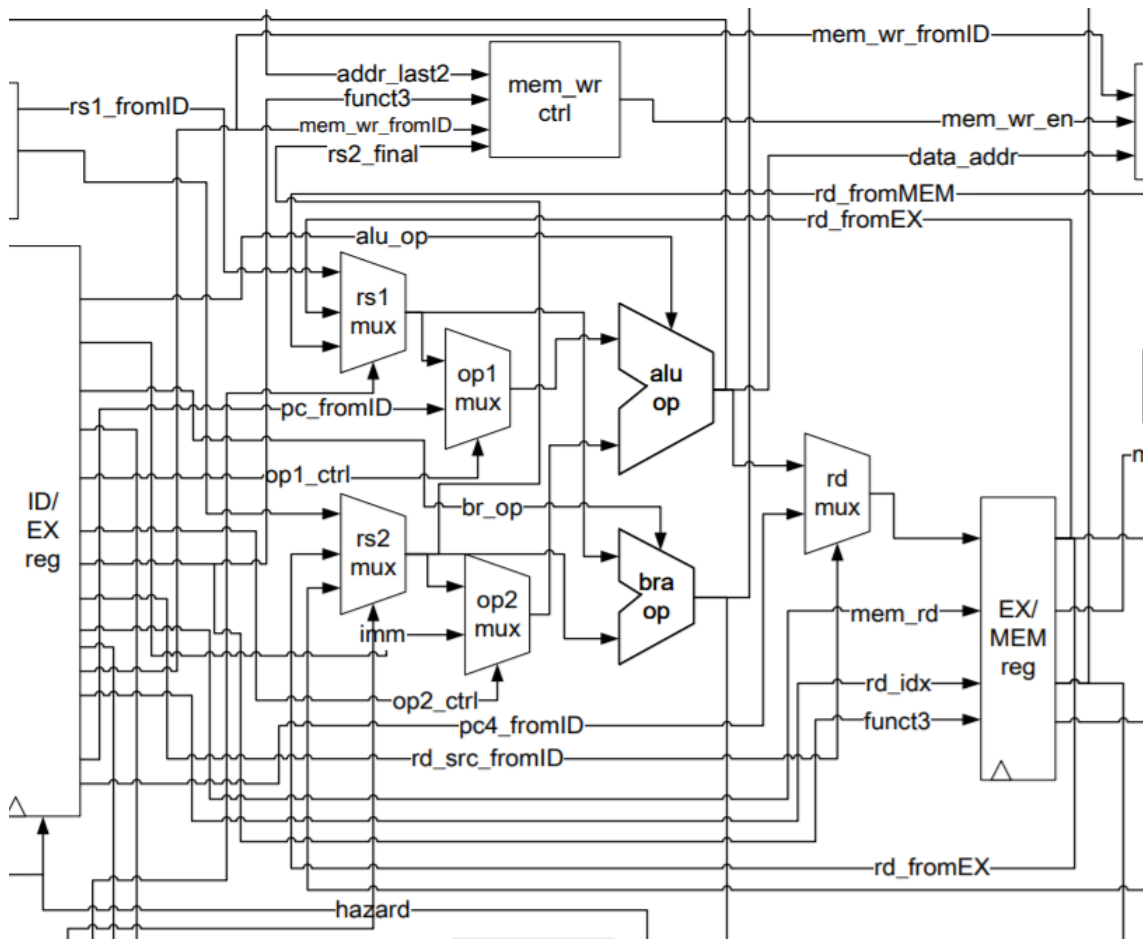
op1, op2 mux：由 op\_ctrl 訊號，決定 op1 為 rs1 或 pc，op2 為 rs2 或 imm。

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。

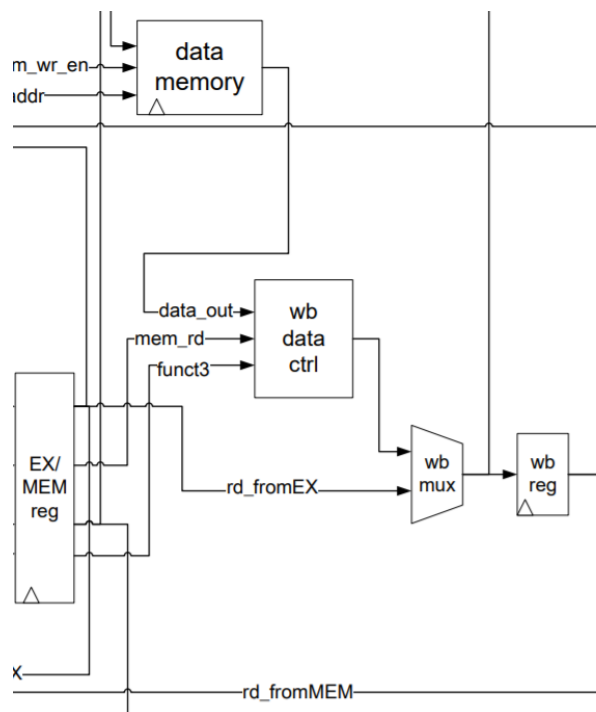
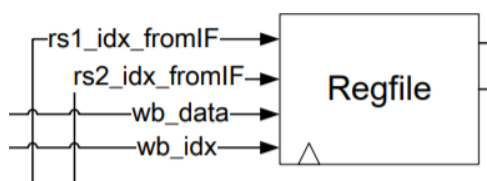
mem\_wr\_ctrl：控制 store 相關的訊號，根據 funct3 與 alu\_res 最後兩 bit 決定 wr\_mem\_en 的值，以及是否要針對 rs2 做 shift。並將這些訊號以及 mem\_addr 送到 data memory。



#### 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。

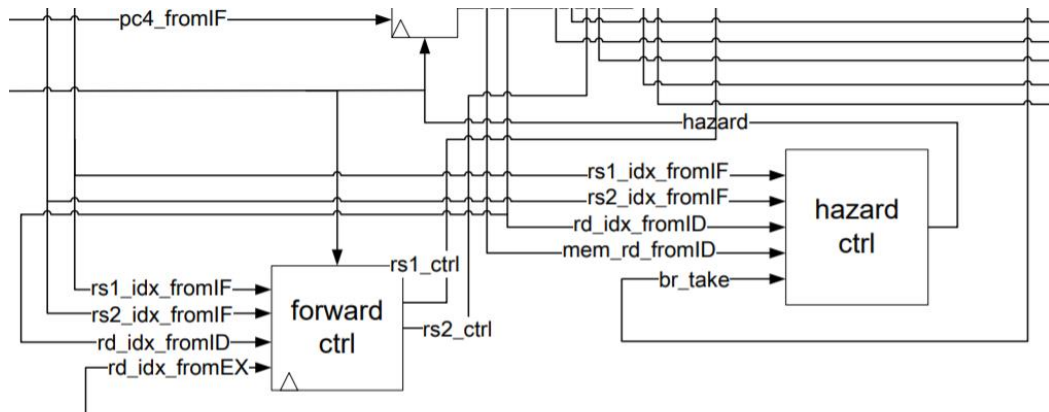
在下一個 posedge，wb\_data 會寫回 regfile 對應的 wb\_idx，指令完成。



## 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

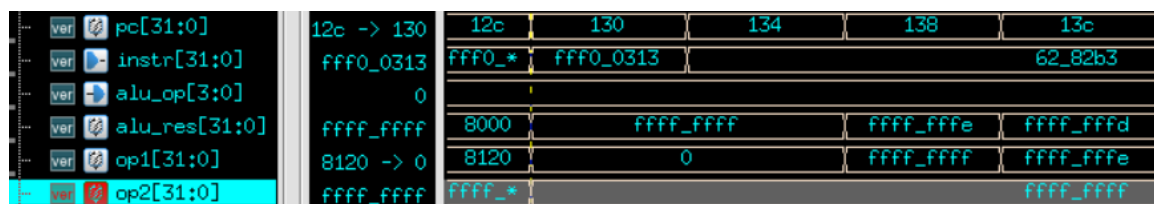
130: 006282b3 add t0,t0,t1

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

ver	pc[31:0]	1e8 -> 1ec	2*	1e4	1e8	1ec	1f0
ver	instr[31:0]	62_a2b3	3*	10_0313			62_a2b3
ver	alu_op[3:0]	0 -> 2		0			
ver	alu_res[31:0]	1	8*	ffff_ffff	1	0	
ver	op1[31:0]	ffff_ffff	0	0	ffff_ffff	1	
ver	op2[31:0]	1	4	ffff_ffff			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

ver	pc[31:0]	2f0 -> 2f4	2*	2ec	2f0	2f4	2f8	2fc
ver	instr[31:0]	4062_d2b3	3*	40_0313				4062_d2b3
ver	alu_op[3:0]	0 -> d		0				
ver	alu_res[31:0]	f876_5432	8*	8765_4321	4	f876_5432	ff87_6543	fff8_7654
ver	op1[31:0]	8765_4321	0	8765_4000	0	8765_4321	f876_5432	ff87_6543
ver	op2[31:0]	4	8*	321				4

## 2. I-type

(a) addi

458: fff28293 addi t0,t0,-1

IF : pc = 458

ID : instr = fff28293

EX : op1 = ffff\_ffff, 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\_ffff

ver	pc[31:0]	45c -> 460	4*	458	45c	460	464
ver	instr[31:0]	7652_8293	4*	fff0_0293	fff2_8293	7652_8293	8882_8293
ver	alu_op[3:0]	0					
ver	alu_res[31:0]	ffff_ffff	8*	803c	ffff_ffff	ffff_ffff	763
ver	op1[31:0]	ffff_ffff		8038	0	ffff_ffff	ffff_ffff
ver	op2[31:0]	ffff_ffff	0	4	ffff_ffff		765
ver	imm[31:0]	ffff_ffff	0	4	ffff_ffff		765
ver	rs2_final[31:0]	0	c*		0		ffff_ffff
ver	op2_ctrl	1					

(b) lw

3b8: 00032283 lw t0,0(t1)

IF: pc = 3b8

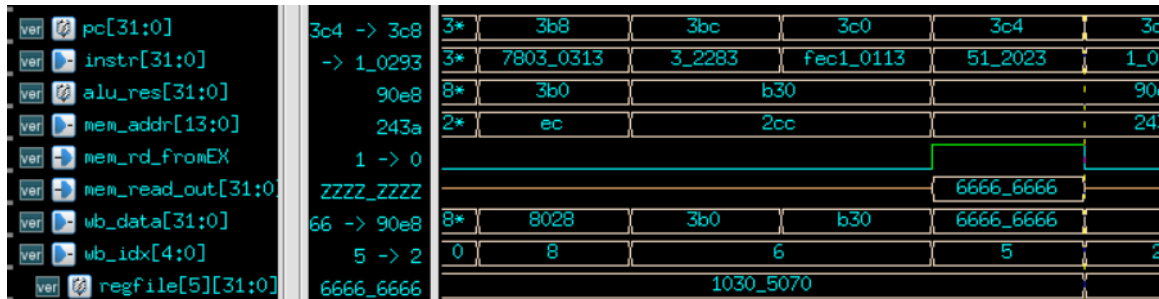
ID: instr = 00032283

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

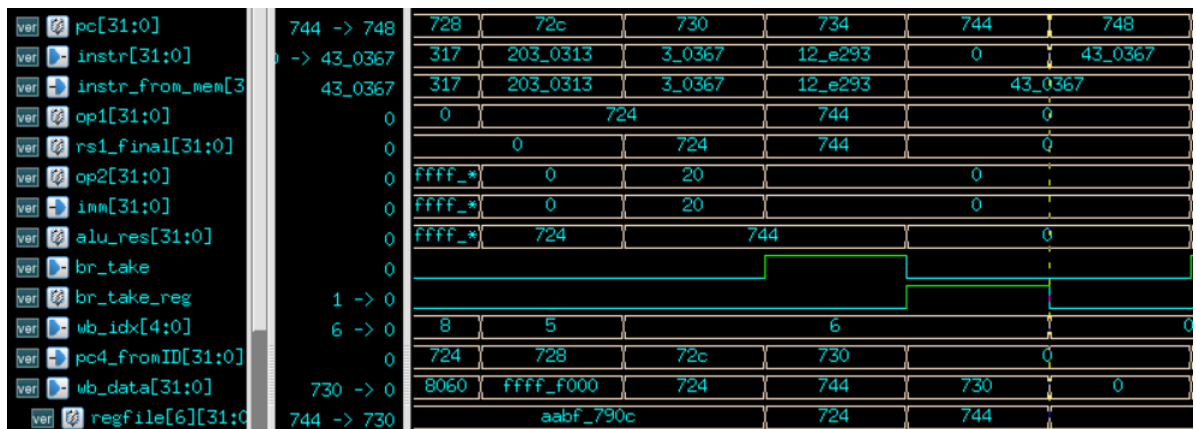
next cycle: pc = 744, instr\_from\_mem 為 pc=734 對應的錯誤指令，由於 br\_take\_reg = 1，instr\_mux 會將 instr 設為 0，產生 bubble

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

ver	pc[31:0]	7c8 -> 7cc	7c4	7c8	7cc	7d0
ver	instr[31:0]	-> fe64_2c23	144_0413	fe54_2e23	fe64_2c23	fe74_2a23
ver	op1[31:0]	8064 -> 8078	1234_5000	8064		
ver	rs1_final[31:0]	8064 -> 8078	1234_5000	8064		
ver	op2[31:0]	-> ffff_fffc	678	14	ffff_fffc	ffff_fff8
ver	imm[31:0]	-> ffff_fffc	678	14	ffff_fffc	ffff_fff8
ver	funct3_fromID[2:0]	0 -> 2		0		
ver	alu_res[31:0]	8078 -> 8074	1234_5678	8078	8074	8070
ver	A[13:0]	201e -> 201d	159e	201e	201d	201c
ver	mem_addr[13:0]	201e -> 201d	159e	201e	201d	201c
ver	DI[31:0]	0 -> f		0	f	f0
ver	rs2_final[31:0]	0 -> f		0	f	f0
ver	WEB[3:0]	f -> 0		f		

(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



ver	pc[31:0]	7f4 -> 7f8	*	7f0	7f4	7f8	7fc
ver	instr[31:0]	-> ffe4_1723	*	ffe4_1a23	ffe4_09a3	ffe4_1723	ff04_2283
ver	op1[31:0]	808c				808c	
ver	rs1_final[31:0]	808c				808c	
ver	op2[31:0]	-> ffff_fff3	*	ffff_fff8	ffff_fff4	ffff_fff3	ffff_ffee
ver	imm[31:0]	-> ffff_fff3	*	ffff_fff8	ffff_fff4	ffff_fff3	ffff_ffee
ver	funct3_fromID[2:0]	1 -> 0	*	0	1	0	1
ver	alu_res[31:0]	8080 -> 807f	*	8084	8080	807f	807a
ver	A[13:0]	2020 -> 201f	*	2021	2020	201f	201e
ver	mem_addr[13:0]	2020 -> 201f	*	2021	2020	201f	201e
ver	DI[31:0]	-> 7800_0000		1234_5678		7800_0000	5678_0000
ver	rs2_final[31:0]	1234_5678			1234_5678		
ver	WEB[3:0]	c -> 7	*	e	c	7	3

(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

ver	pc[31:0]	7f0 -> 7f4	*	7ec	7f0	7f4	7f8
ver	instr[31:0]	-> ffe4_09a3	*	ffe4_0c23	ffe4_1a23	ffe4_09a3	ffe4_1723
ver	op1[31:0]	808c				808c	
ver	rs1_final[31:0]	808c				808c	
ver	op2[31:0]	-> ffff_fff4	*	ffff_fff4	ffff_fff8	ffff_fff4	ffff_fff3
ver	imm[31:0]	-> ffff_fff4	*	ffff_fff4	ffff_fff8	ffff_fff4	ffff_fff3
ver	funct3_fromID[2:0]	0 -> 1		2	0	1	0
ver	alu_res[31:0]	8084 -> 8080	*	8088	8084	8080	807f
ver	A[13:0]	2021 -> 2020	*	2022	2021	2020	201f
ver	mem_addr[13:0]	2021 -> 2020	*	2022	2021	2020	201f
ver	DI[31:0]	1234_5678		0	1234_5678		7800_0000
ver	rs2_final[31:0]	1234_5678		0		1234_5678	
ver	WEB[3:0]	e -> c	f	0	e	c	7

#### 4. B-type

(a) beq

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

IF : pc = 85c ID : instr = 00628463

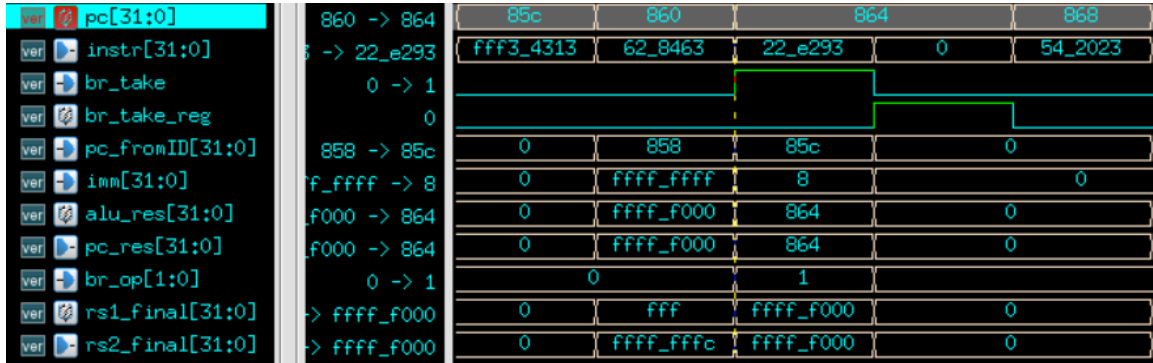
EX : br\_take = (rs1 == rs2)? = ffff\_f000 == ffff\_f000 ? = 1

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

next cycle:  $pc = 864$ , instr 為錯誤指令，flush ID  $\Rightarrow$  bubble

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

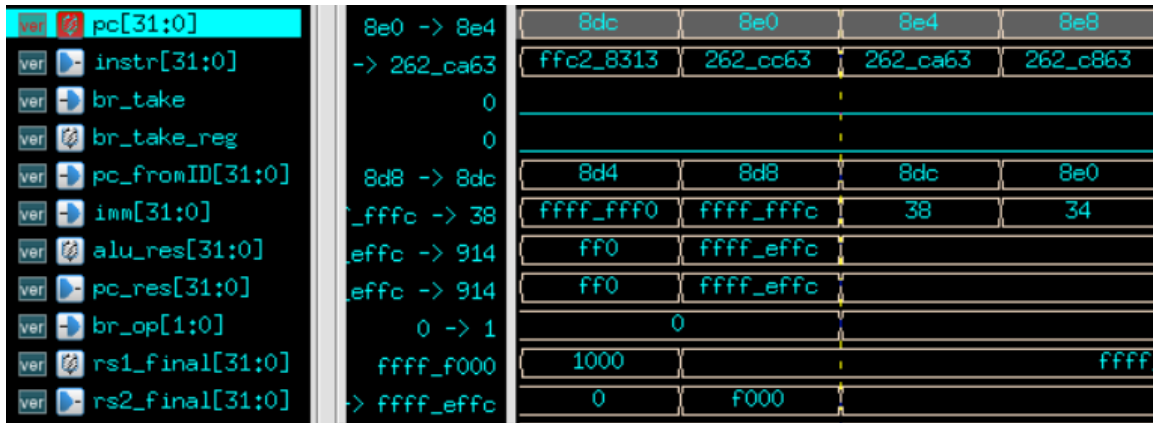
IF:  $pc = 8dc$

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

ver	pc[31:0]	a20	a18	a1c	a20	a24
ver	instr[31:0]	22_e293 -> 0	fff0_0313	62_e463	22_e293	0
ver	br_take	1 -> 0				
ver	br_take_reg	0 -> 1				
ver	pc_fromID[31:0]	a18 -> 0	0	a14	a18	0
ver	imm[31:0]	8 -> 0	0	ffff_ffff	8	0
ver	alu_res[31:0]	a20 -> 0	0	ffff_ffff	a20	0
ver	pc_res[31:0]	a20 -> 0	0	ffff_ffff	a20	0
ver	br_op[1:0]	1 -> 0	0		1	0
ver	rs1_final[31:0]	ff_f000 -> 0	0		ffff_f000	0
ver	rs2_final[31:0]	ff_ffff -> 0	0		ffff_ffff	0

## 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

ver	pc[31:0]	80 -> 84	7c	80	84	88	8c
ver	instr[31:0]	-> 8841_8193	f93	9197	8841_8193	8517	7c5_0513
ver	pc_fromID[31:0]	78 -> 7c	74	78	7c	80	84
ver	imm[31:0]	0 -> 9000	0		9000	ffff_f884	8000
ver	alu_res[31:0]	0 -> 907c	0		907c	8900	8084
ver	wb_data[31:0]	0			0	907c	8900
ver	wb_idx[4:0]	1e -> 1f	1d	1e	1f		3
ver	regfile[3][31:0]	0			0		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

ver	pc[31:0]	af8 -> afc	aec	af0	af4	af8	afc
ver	instr[31:0]	5 -> 54_2023	ffff_f337	1357_93b7	73_0333	62_82b3	54_2023
ver	op1[31:0]	f_f000 -> 0	0			ffff_f000	0
ver	imm[31:0]	0	0	ffff_f000	1357_9000	0	
ver	alu_res[31:0]	1357_8000	0	ffff_f000	1357_9000	1357_8000	
ver	wb_data[31:0]	-> 1357_8000	80ac	0	ffff_f000	1357_9000	1357_
ver	wb_idx[4:0]	7 -> 6	8	5	6	7	6
ver	regfile[7][31:0]	-> 1357_9000	1357_9ad0				

## 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

ver	pc[31:0]	104 -> 108	94	98	9c	a0	104	108
ver	instr[31:0]	0 -> b5_7863	705_8*	613	6c0_00ef	8517	0	b5_7863
ver	instr_from_mem[31:0]	0 -> b5_7863	705_8*	613	6c0_00ef	8517	645_0513	b5_7863
ver	op1[31:0]	0	8c	808c	0	98	0	
ver	pc_fromID[31:0]	0	8c	90	94	98	0	
ver	op2[31:0]	0	8000	70	0	6c	0	
ver	imm[31:0]	0	8000	70	0	6c	0	
ver	alu_res[31:0]	0	808c	80fc	0	104	0	
ver	br_take	0						
ver	br_take_reg	1 -> 0						
ver	wb_idx[4:0]	1 -> 0						
ver	pc4_fromID[31:0]	0	a	b	c	1		
ver	wb_data[31:0]	9c -> 0	90	94	98	9c	0	
ver	regfile[1][31:0]	0 -> 9c	8100	808c	80fc	0	9c	0
			0					

### 三、 test program

#### 1. prog0

```
DM[8226] = 12345678, pass
DM[8227] = ce780000, pass
DM[8228] = fffff000, pass
DM[8229] = fffff000, pass
DM[8230] = fffff000, pass
DM[8231] = fffff000, pass
DM[8232] = fffff000, pass
DM[8233] = fffff000, pass
DM[8234] = 1357a064, pass
DM[8235] = 13578000, pass
DM[8236] = fffff004, pass
```

```
*****
**                                     **
**  Congratulations !!              **
**                                     **
**  Simulation PASS!!               **
**                                     **
*****
```

```
      |__||
      / 0.0 |
      /_____|
     / ^ ^ ^ \
    | ^ ^ ^ ^ |w|
     \m__m__|_|
```

```
Simulation complete via $finish(1) at time 63514350 PS + 2
../sim/top_tb.sv:76      $finish;
```

## 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)

```
*****
**                                     **
** Congratulations !!               **
**                                     **
** Simulation PASS!!                **
**                                     **
*****
                                     |
                                     / 0.0 |
                                     / ^ ^ ^ \ |
                                     | ^ ^ ^ |w|
                                     \m   m _|_

Simulation complete via $finish(1) at time 77712750 PS + 2
../sim/top_tb.sv:76      $finish;
```

```
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]

$1a2b3c4d \times 98765432 = f95c456cfb90b0a$

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

*****
**                                     **
**  Congratulations !!              **
**                                     **
**  Simulation PASS!!               **
**                                     **
*****

          |__||
         / 0.0 |
        /_____|
       / ^ ^ ^ \
      | ^ ^ ^ ^ |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 !!              **
**                                     **
**  Simulation PASS!!               **
**                                     **
*****

          |__||
         / 0.0 |
        /_____|
       / ^ ^ ^ \
      | ^ ^ ^ ^ |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
```

Category: CODINGSTYLE (12)
Tag: ASG_MS_RPAD (2)
Tag: EXP_NR_OVFB (5)
Tag: OPR_NR_UREL (5)
Category: STRUCTURAL (15)
Tag: FLP_NO_ASRT (13)
Tag: FLP_NR_ASRT (2)

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

Tag: FLP\_NO\_ASRT (13)

- "Flip-flop 'br\_take\_reg' does not have any asynchronous set or reset"
- "Flip-flop 'pc' does not have any asynchronous set or reset"
- "Flip-flop 'pc4\_fromIF' does not have any asynchronous set or reset"
- "Flip-flop 'alu\_op' does not have any asynchronous set or reset"
- "Flip-flop 'funct3\_fromID' does not have any asynchronous set or reset"

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

Tag: EXP\_NR\_OVFB (5)

- "Shift overflow in module/design-unit EX, some bits will be lost"
- "Shift overflow in module/design-unit EX, some bits will be lost"
- "Shift overflow in module/design-unit EX, some bits will be lost"

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。

Tag: OPR\_NR\_UREL (5)

- "Unequal length operands in relational operator (padding produces incorrect result) in module..."
- "Unequal length operands in relational operator (padding produces incorrect result) in module..."
- "Unequal length operands in relational operator (padding produces incorrect result) in module..."
- "Unequal length operands in relational operator (padding produces incorrect result) in module..."
- "Unequal length operands in relational operator (padding produces incorrect result) in module..."

## 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。