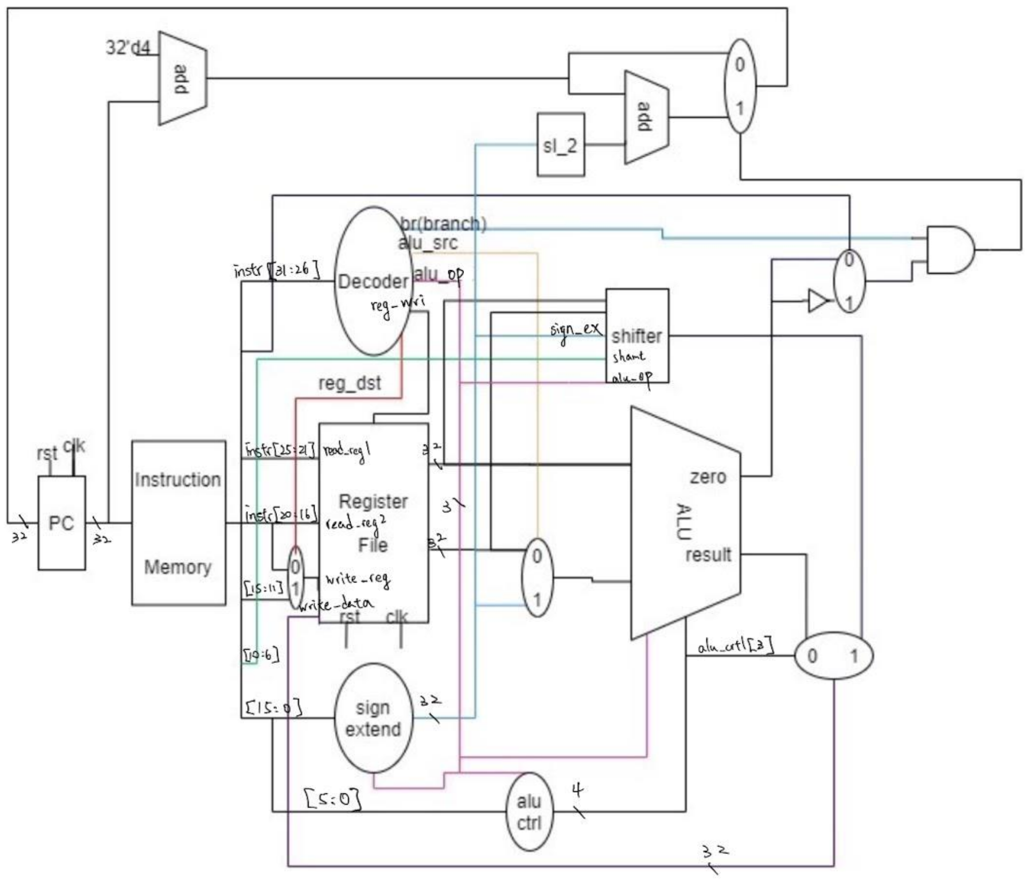
Lab2_Report

0710028 陳敬諺

● Architecture diagram:



● Module description:

Adder1:

執行 pc+=4

Mux_Write_Reg:

由 reg_dst 決定 write_reg 的位置

Decoder:

將 opcode 轉換至對應的{RegWrite , ALU_op , ALU_src , Reg_dst , Branch}

AC(ALU_Crtl):

由 ALU_op , function(instructions[5:0])決定 alu_ctrl

SE(Sign_Extend):

將 instructions[15:0]變更為 32bits，並用 alu_op 判斷是 sign ext./zero ext.

Mux_ALUsrc:

用 ALU_src 決定 ALU 的 src2，0 為 rt 輸入，1 為 sign_ex 輸入

ALU:

32bits ALU，原則上沿用 lab1 的架構，運算結果由 alu_ctrl 控制，

alu_ctrl[3]=>A_invert，alu_ctrl[2]=>B_invert，alu_ctrl[1:0]=00 為 and、01 為 or、10 為 add、11 為 slt。另外利用 alu_op 判斷 sltiu 的情形，當遇到 sltiu 運算，取第 32 個 bit 的 cout 跟 1 再做一次 add 運算才是真正的 less_set 結果。

shifter32:

以 shift_ctrl(alu_ctrl)決定 shift 的型態，shift_ctrl=1000 為 lui，1001 為 sra、1010 為 srav

Shifter_1_2:

執行（n << 2)

Adder2:

執行 pc += (4+ (n << 2))

equal_confirm:

比較 bne、beq 的 opcode，在 opcode[0](instruction[26])會有差異，所以用 opcode[0]作為 select，用 zero_result 作為 alu_src1 與 alu_src2 是否相同的依據。

and_gate to pointer:

以 branch 控制要不要輸出 equal_confirm 的結果

Mux_PC_Source:

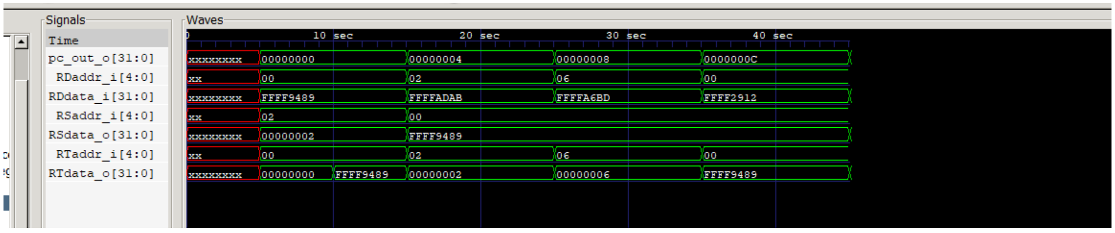
當 pointer 結果輸入會辨別要不要以 adder2 的結果回傳給 PC

result_confirm:

用 alu_ctrl[3]控制，1 為回傳 shifter_result 到 write_data、0 為回傳 alu_result 到 write_data

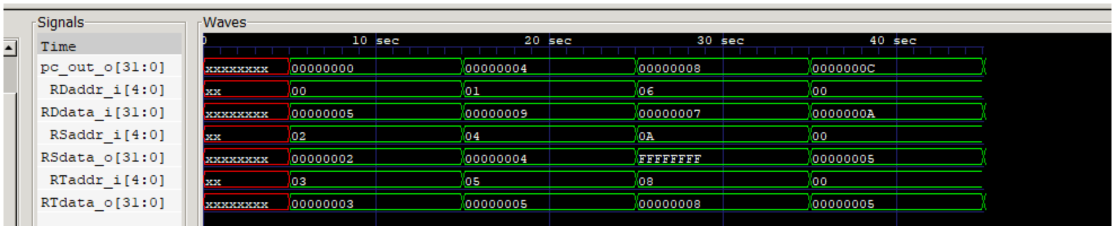
- ## Waveform:

addi



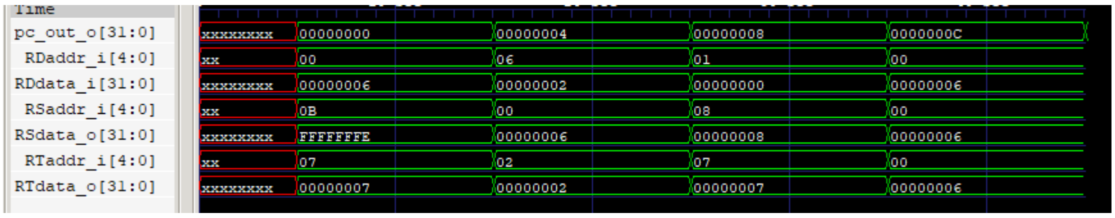
```
pc_out_o[31:0]   xxxxxxxx  00000000            00000004  00000008  0000000C
RDaddr_i[4:0]    xx        00                  02        06        00
RDdata_i[31:0]   xxxxxxxx  FFFF9489            FFFFADAB  FFFFA6BD  FFFF2912
RSaddr_i[4:0]    xx        02                  00
RSdata_o[31:0]   xxxxxxxx  00000002            FFFF9489
RTaddr_i[4:0]    xx        00                  02        06        00
RTdata_o[31:0]   xxxxxxxx  00000000  FFFF9489  00000002  00000006  FFFF9489
```

Addu



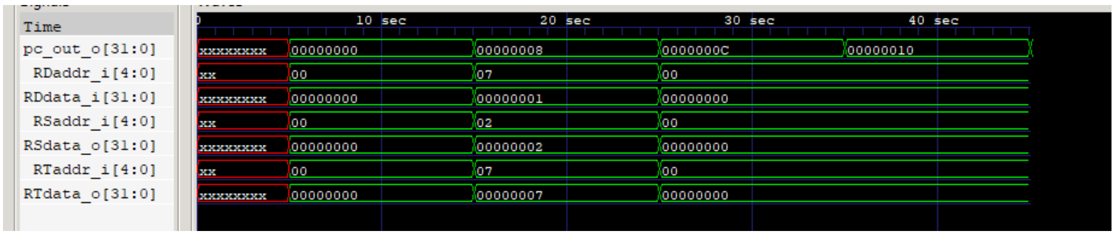
```
pc_out_o[31:0]   xxxxxxxx  00000000  00000004  00000008  0000000C
RDaddr_i[4:0]    xx        00        01        06        00
RDdata_i[31:0]   xxxxxxxx  00000005  00000009  00000007  0000000A
RSaddr_i[4:0]    xx        02        04        0A        00
RSdata_o[31:0]   xxxxxxxx  00000002  00000004  FFFFFFFF  00000005
RTaddr_i[4:0]    xx        03        05        08        00
RTdata_o[31:0]   xxxxxxxx  00000003  00000005  00000008  00000005
```

and



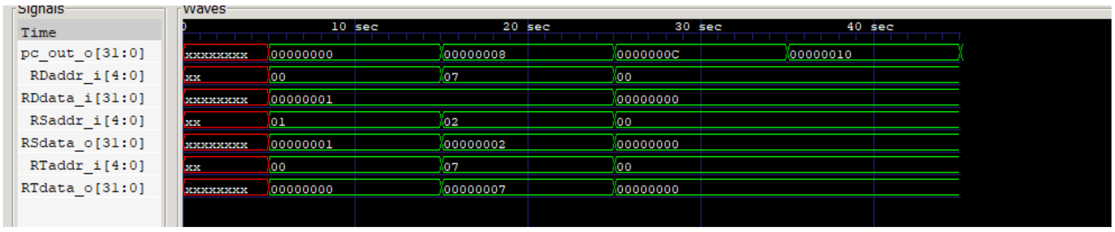
```
Time
pc_out_o[31:0]   xxxxxxxx  00000000  00000004  00000008  0000000C
RDaddr_i[4:0]    xx        00        06        01        00
RDdata_i[31:0]   xxxxxxxx  00000006  00000002  00000000  00000006
RSaddr_i[4:0]    xx        0B        00        08        00
RSdata_o[31:0]   xxxxxxxx  FFFFFFFE  00000006  00000008  00000006
RTaddr_i[4:0]    xx        07        02        07        00
RTdata_o[31:0]   xxxxxxxx  00000007  00000002  00000007  00000006
```

Beq



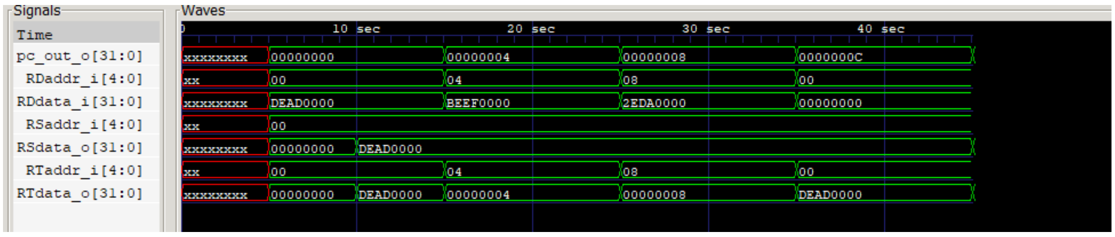
```
Time                       10 sec    20 sec    30 sec    40 sec
pc_out_o[31:0]   xxxxxxxx  00000000  00000008  0000000C  00000010
RDaddr_i[4:0]    xx        00        07        00
RDdata_i[31:0]   xxxxxxxx  00000000  00000001  00000000
RSaddr_i[4:0]    xx        00        02        00
RSdata_o[31:0]   xxxxxxxx  00000000  00000002  00000000
RTaddr_i[4:0]    xx        00        07        00
RTdata_o[31:0]   xxxxxxxx  00000000  00000007  00000000
```

Bne



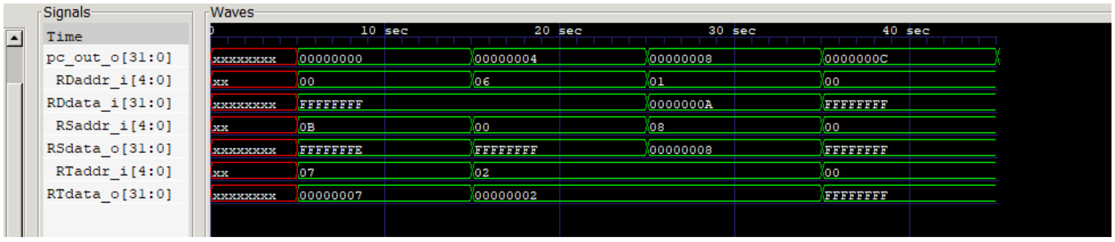
```
Signals
Time                       10 sec    20 sec    30 sec    40 sec
pc_out_o[31:0]   xxxxxxxx  00000000  00000008  0000000C  00000010
RDaddr_i[4:0]    xx        00        07        00
RDdata_i[31:0]   xxxxxxxx  00000001            00000000
RSaddr_i[4:0]    xx        01        02        00
RSdata_o[31:0]   xxxxxxxx  00000001  00000002  00000000
RTaddr_i[4:0]    xx        00        07        00
RTdata_o[31:0]   xxxxxxxx  00000000  00000007  00000000
```

Lui



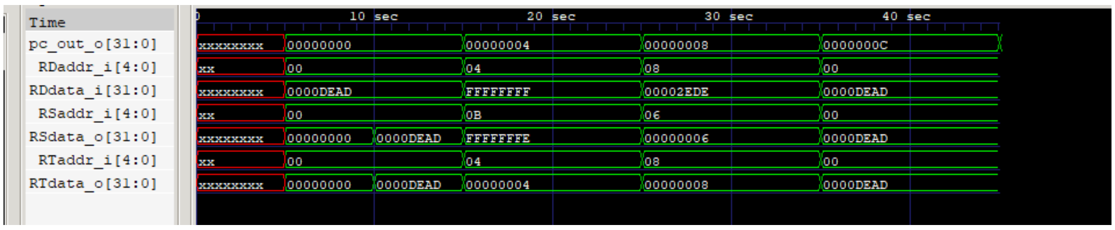
```
Signals
Time                       10 sec    20 sec    30 sec    40 sec
pc_out_o[31:0]   xxxxxxxx  00000000            00000004  00000008  0000000C
RDaddr_i[4:0]    xx        00                  04        08        00
RDdata_i[31:0]   xxxxxxxx  DEAD0000            BEEF0000  2EDA0000  00000000
RSaddr_i[4:0]    xx        00
RSdata_o[31:0]   xxxxxxxx  00000000  DEAD0000
RTaddr_i[4:0]    xx        00                  04        08        00
RTdata_o[31:0]   xxxxxxxx  00000000  DEAD0000  00000004  00000008  DEAD0000
```

or



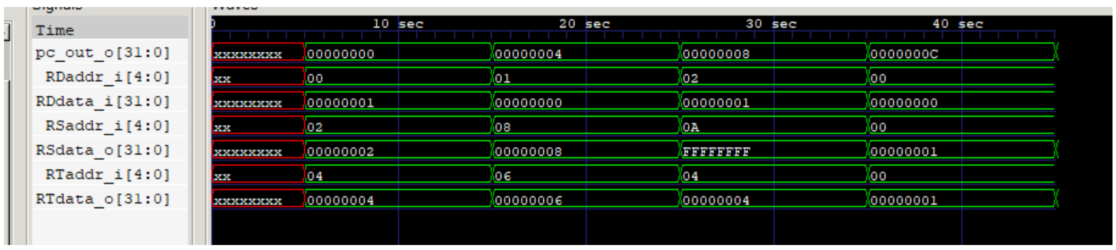
```
Signals
Time                       10 sec    20 sec    30 sec    40 sec
pc_out_o[31:0]   xxxxxxxx  00000000  00000004  00000008  0000000C
RDaddr_i[4:0]    xx        00        06        01        00
RDdata_i[31:0]   xxxxxxxx  FFFFFFFF            0000000A  FFFFFFFF
RSaddr_i[4:0]    xx        0B        00        08        00
RSdata_o[31:0]   xxxxxxxx  FFFFFFFE  FFFFFFFF  00000008  FFFFFFFF
RTaddr_i[4:0]    xx        07        02                  00
RTdata_o[31:0]   xxxxxxxx  00000007  00000002            FFFFFFFF
```

Ori

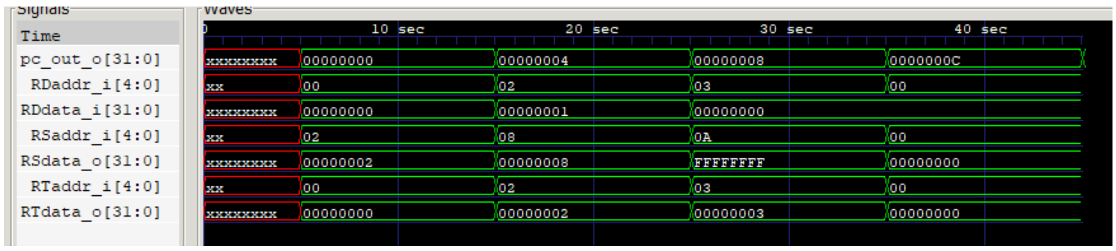


```
Time                       10 sec              20 sec    30 sec    40 sec
pc_out_o[31:0]   xxxxxxxx  00000000            00000004  00000008  0000000C
RDaddr_i[4:0]    xx        00                  04        08        00
RDdata_i[31:0]   xxxxxxxx  0000DEAD            FFFFFFFF  00002EDE  0000DEAD
RSaddr_i[4:0]    xx        00                  0B        06        00
RSdata_o[31:0]   xxxxxxxx  00000000  0000DEAD  FFFFFFFE  00000006  0000DEAD
RTaddr_i[4:0]    xx        00                  04        08        00
RTdata_o[31:0]   xxxxxxxx  00000000  0000DEAD  00000004  00000008  0000DEAD
```

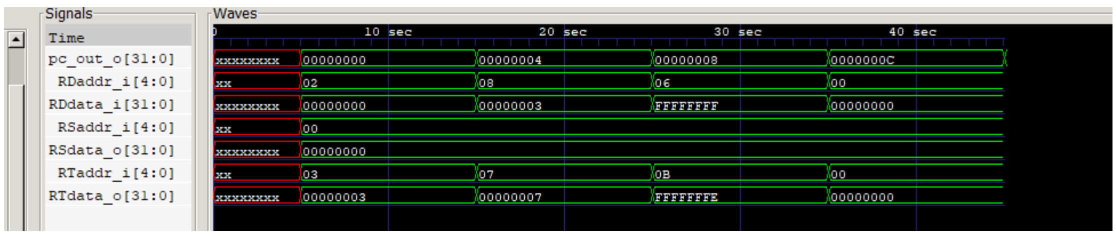
Slt


Signals
Waves
Time 10 sec 20 sec 30 sec 40 sec
pc_out_o[31:0] xxxxxxxx 00000000 00000004 00000008 0000000C
RDaddr_i[4:0] xx 00 01 02 00
RDdata_i[31:0] xxxxxxxx 00000001 00000000 00000001 00000000
RSaddr_i[4:0] xx 02 08 0A 00
RSdata_o[31:0] xxxxxxxx 00000002 00000008 FFFFFFFF 00000001
RTaddr_i[4:0] xx 04 06 04 00
RTdata_o[31:0] xxxxxxxx 00000004 00000006 00000004 00000001

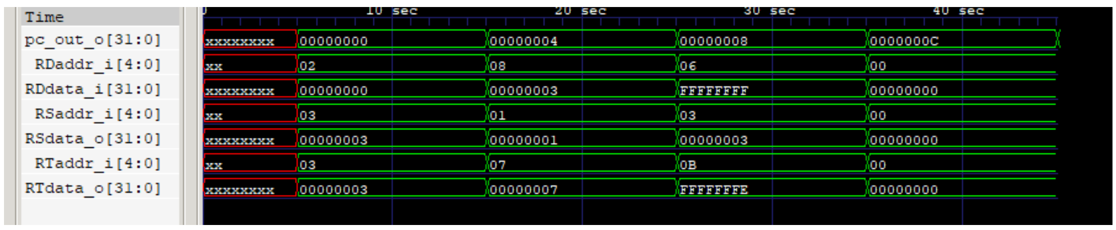
Sltiu


Signals
Waves
Time 10 sec 20 sec 30 sec 40 sec
pc_out_o[31:0] xxxxxxxx 00000000 00000004 00000008 0000000C
RDaddr_i[4:0] xx 00 02 03 00
RDdata_i[31:0] xxxxxxxx 00000000 00000001 00000000
RSaddr_i[4:0] xx 02 08 0A 00
RSdata_o[31:0] xxxxxxxx 00000002 00000008 FFFFFFFF 00000000
RTaddr_i[4:0] xx 00 02 03 00
RTdata_o[31:0] xxxxxxxx 00000000 00000002 00000003 00000000

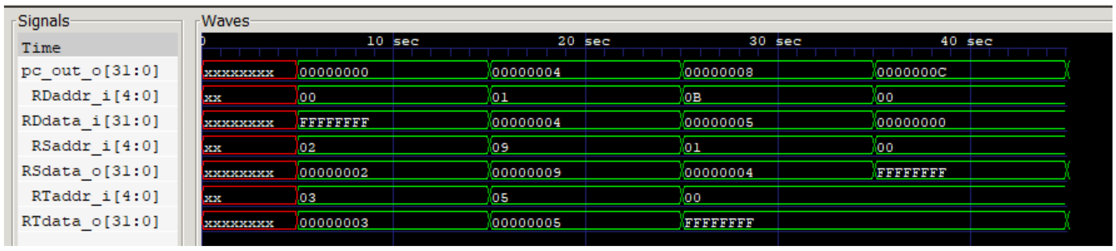
Sra


Signals
Waves
Time 10 sec 20 sec 30 sec 40 sec
pc_out_o[31:0] xxxxxxxx 00000000 00000004 00000008 0000000C
RDaddr_i[4:0] xx 02 08 06 00
RDdata_i[31:0] xxxxxxxx 00000000 00000003 FFFFFFFF 00000000
RSaddr_i[4:0] xx 00
RSdata_o[31:0] xxxxxxxx 00000000
RTaddr_i[4:0] xx 03 07 0B 00
RTdata_o[31:0] xxxxxxxx 00000003 00000007 FFFFFFFE 00000000

Srav


Time 10 sec 20 sec 30 sec 40 sec
pc_out_o[31:0] xxxxxxxx 00000000 00000004 00000008 0000000C
RDaddr_i[4:0] xx 02 08 06 00
RDdata_i[31:0] xxxxxxxx 00000000 00000003 FFFFFFFF 00000000
RSaddr_i[4:0] xx 03 01 03 00
RSdata_o[31:0] xxxxxxxx 00000003 00000001 00000003 00000000
RTaddr_i[4:0] xx 03 07 0B 00
RTdata_o[31:0] xxxxxxxx 00000003 00000007 FFFFFFFE 00000000

Subu


Signals
Waves
Time 10 sec 20 sec 30 sec 40 sec
pc_out_o[31:0] xxxxxxxx 00000000 00000004 00000008 0000000C
RDaddr_i[4:0] xx 00 01 0B 00
RDdata_i[31:0] xxxxxxxx FFFFFFFF 00000004 00000005 00000000
RSaddr_i[4:0] xx 02 09 01 00
RSdata_o[31:0] xxxxxxxx 00000002 00000009 00000004 FFFFFFFF
RTaddr_i[4:0] xx 03 05 00
RTdata_o[31:0] xxxxxxxx 00000003 00000005 FFFFFFFF

- Questions:

1. What is the difference between "input [15:0] input_0" and "input [0:15] input_0" inside the module?

   若是前者 input_0[0] 會是最右邊的位數，而後者 input_0[0]會最左邊的位數

2. What is the meaning of "always" block in Verilog?

   當指定參數條件符合或是設定任何在always block的參數改變，即執行一次always block裡的程序

3. What are the advantages and disadvantages of port connection by order and port connection by name in Verilog?

Port connection by order:

Advantage: 較簡潔，當module的參數與外部參數命名一樣時，可直接套用

Disadvantage: 容易混淆，debug時可能會造成不便

Port connection by name:

Advantage: 架構清楚，且不用對應原先module設定的順序，在trace和debug上較有優勢。

Disadvantage: 若module參數名稱與原先建立的不相符(錯字)，執行程序就會出錯。

- Contribution:

  Most of architecture、debug

- Anything to share:

  這次lab我發現自己有個壞毛病，在實行alu_ctrl之類的功能時，我會習慣直接輸入數字代碼，而非利用localparam之類的狀態機參數定義，導致後面在debug的時候，trace不易，要看數字去對應到功能花了不少時間。