VLSI System Design (Graduate Level)

Fall 2022

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

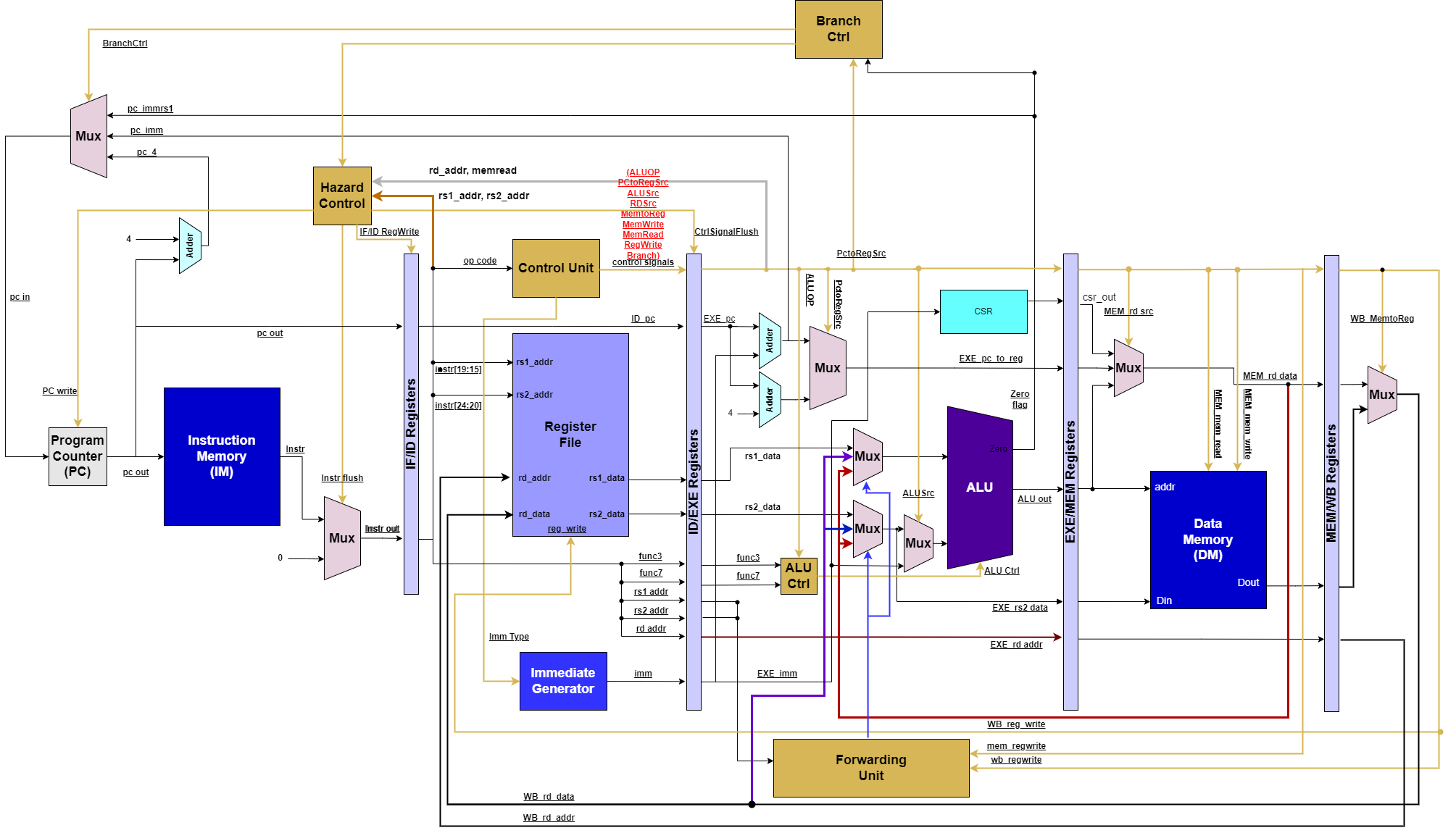
All port declarations comply with I/O port specifications

Organize files according to File Hierarchy Requirement

No any waveform files in deliverables

Student name: \_黃昱澄\_

Student ID: \_P76111238\_

一、架構:此架構根據pdf中的pipeline RISC-V所繪製

**IF級**:做PC的計算以及Instruction memory的讀取，由branch的output決定PC的值，如下。branchctrl=2 =>PC=aluout

branchctrl=1 =>PC=PC+imm

branchctrl=0 =>PC=PC+4

**ID級**:會讀出register的值，與rd的address，在Immediate Generator內依據指令的opcode產生imm值，並在Control unit決定指令的各個control signals，並輸出hazard control的input，hazard control會根據輸入決定是load-use data hazard/control hazard，做出1個stall/flush 2個指令。

**EXE級**:會根據指令來進行計算，而input根據Forwarding unit來決定，如下，並且輸出結果(至MEM跟branch)。

Forward=2 =>input=ID output

Forward=1 =>input=MEM前饋

Forward=0 =>input=WB前饋

有些指令會需要與pc有關的值，所以會有一個exe\_pctoreg的輸出，如下。

pctoregsrc=1 =>exe\_pctoreg=pc+imm

pctoregsrc=0 =>exe\_pctoreg=pc+4

**MEM級**:若有load或store指令，會根據輸入位址進行讀出/寫入的動作，其餘指令則是根據RDsrc決定要輸出至WB級的data，如下。

RDsrc=2 => mem\_data\_out = csr output

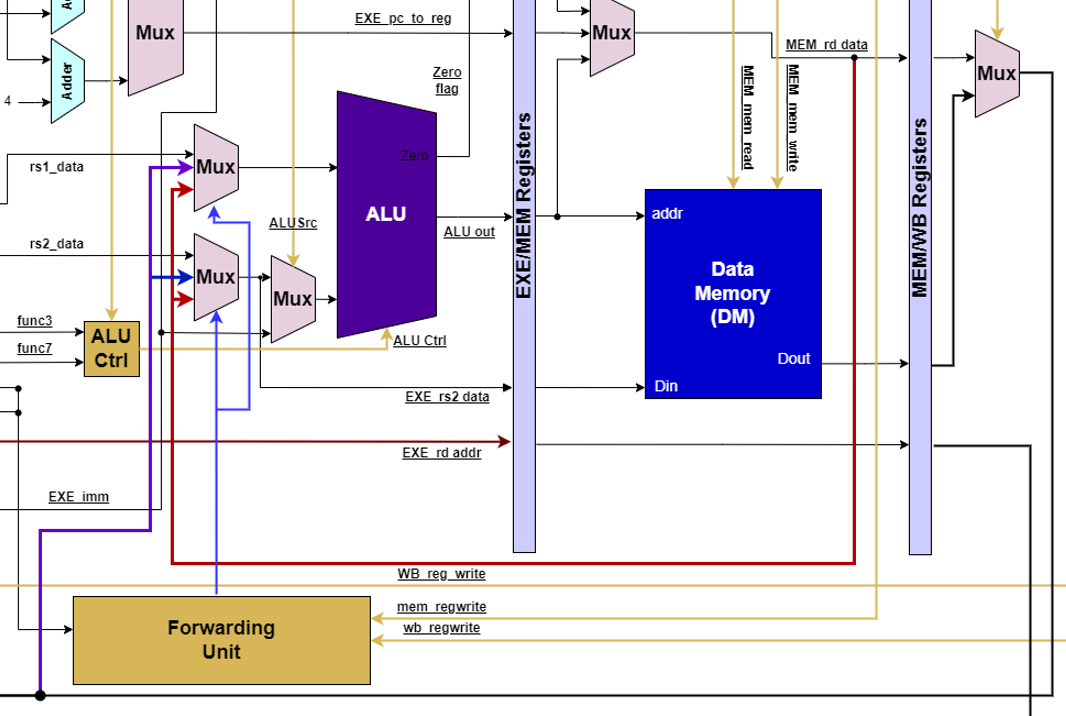
RDsrc=1 => mem\_data\_out = exe\_pctoreg

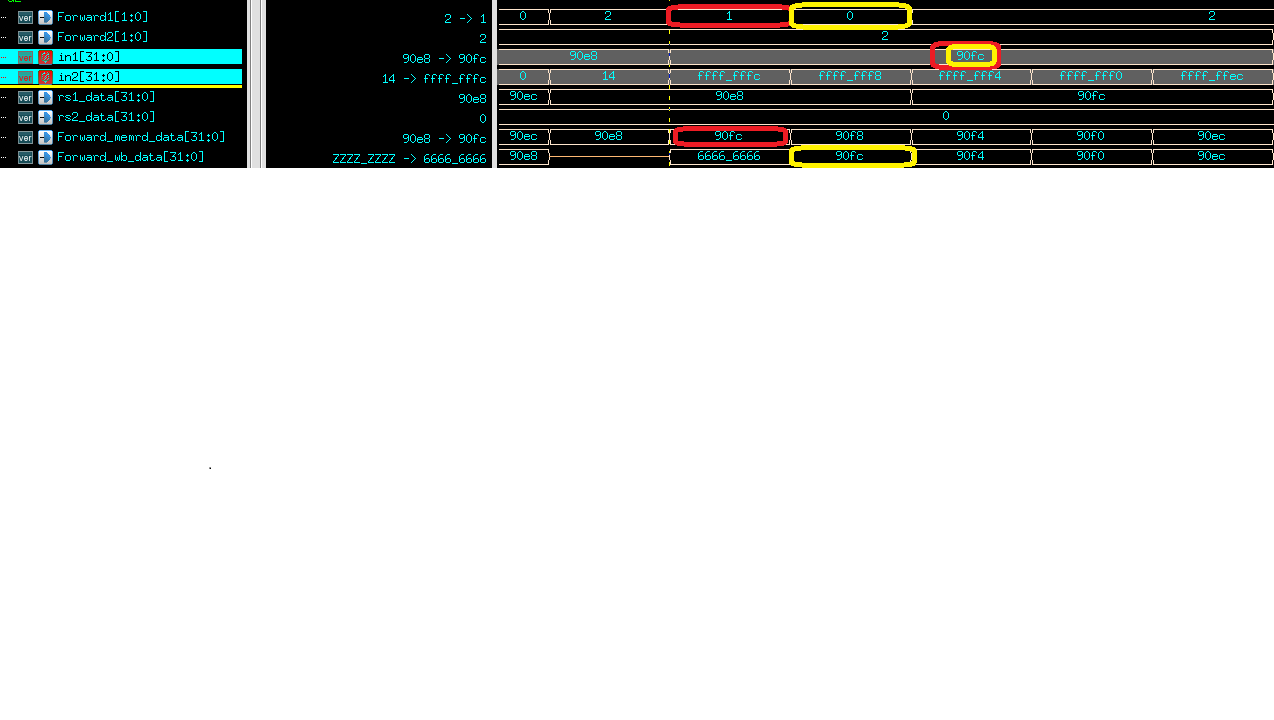
RDsrc=0 => mem\_data\_out = alu output

**WB級**:根據MEM的rd adress將輸出寫回register。

二、Lessons learned

1.練習使用system Verilog : 將上課的內容加入程式碼中，例如:logic、always\_comb、always\_ff，由於時間問題，這次作業還沒有使用interface接線，預計在下次作業中加入。

2.解決data hazard : 除了load-use data hazard，其餘的data hazard都可以用前饋解決，而load-use data hazard則須暫停一個clock再用前饋解決。



如上圖所示，當forward1=2時，會從ID傳data給EXE;當forward1=1時，會從MEM前饋data給EXE，見圖中紅色框框;而當forward1=0時，會從WB前饋data給EXE，見圖中黃色框框。(load-use data hazard的波形在下面有介紹)

3.解決control hazard : 針對branch採用predict always not taken，而branch的跳躍在EXE級執行，若branch predict錯誤則要洗掉在IF與ID級的指令。(關於control hazard的波型，在下面的Btype指令中有介紹)

4.使用superlist : 利用此軟體的worning，可以瞭解比較好的verilog寫法風格。

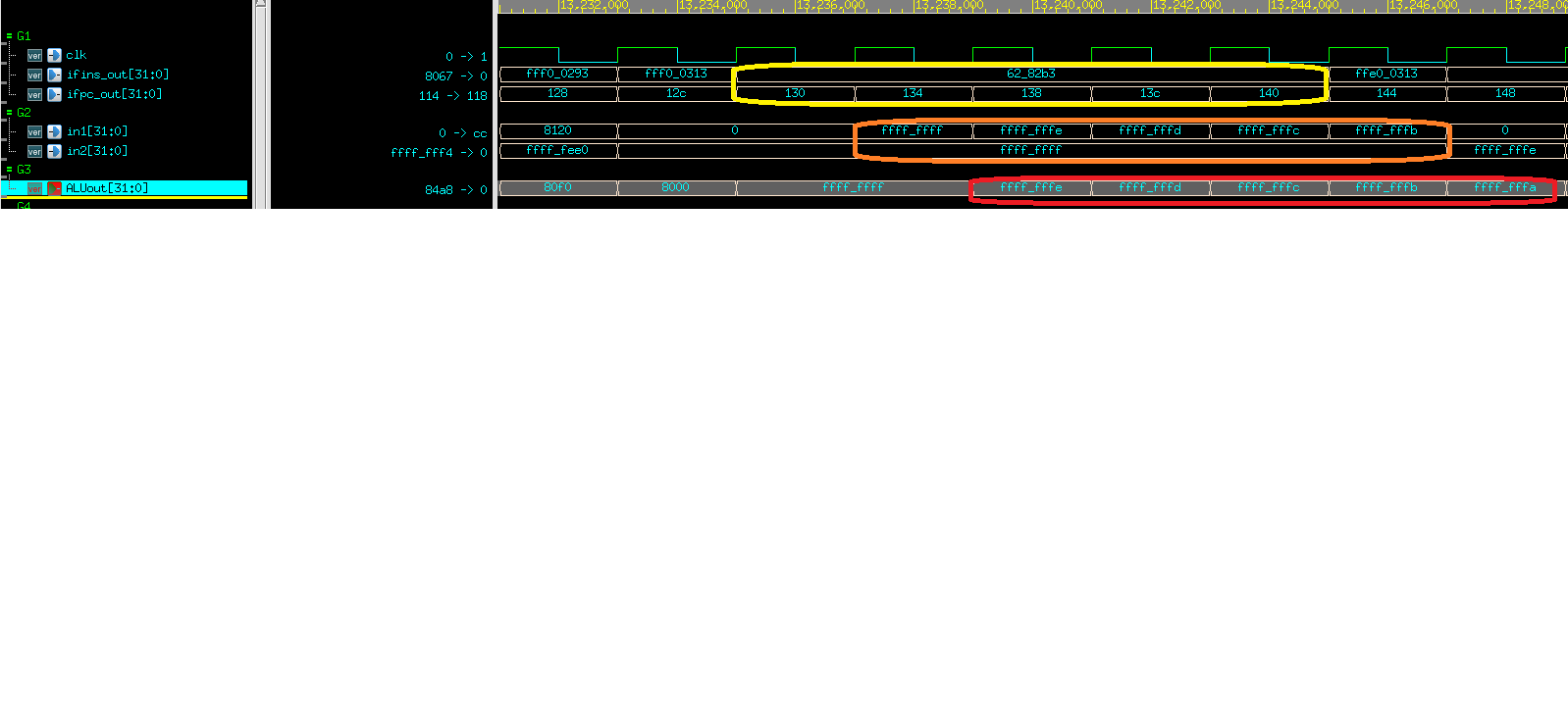
5.研究CSR :

\*RDCYCLE為一個64bit的register，計算經過的cycle數，所以有兩個指令，將64bit分為兩個32bit存入register，計算方法相對簡單，只須跟著clk加一即可。

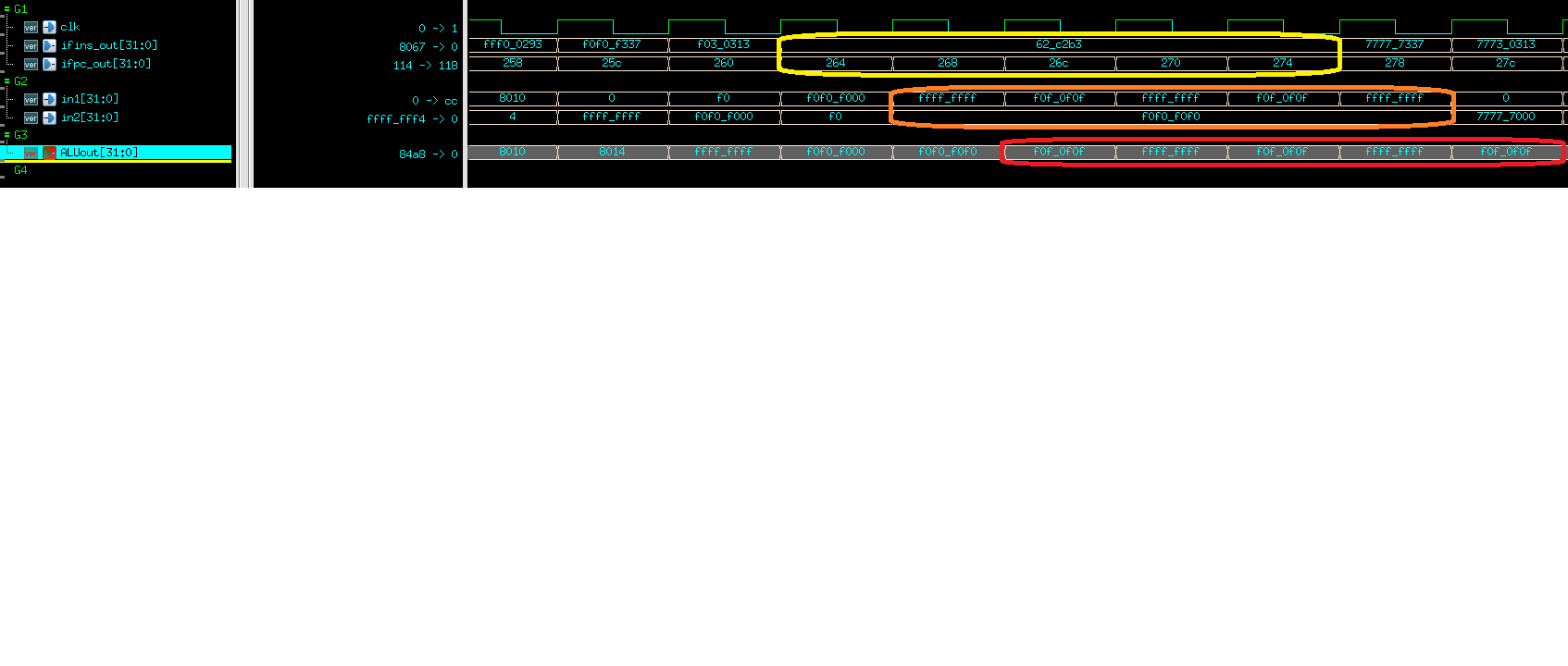
\*RDINSTRET為一個64bit的register，計算執行的instruct數，所以有兩個指令，將64bit分為兩個32bit存入register，計算時跟著clk加一，但由於是在EXE級計算，所以output需要扣2，以及要考慮load-use data hazard的stall與branch&jump會flush掉的instruct數。

三、驗證波形圖

1.R-type

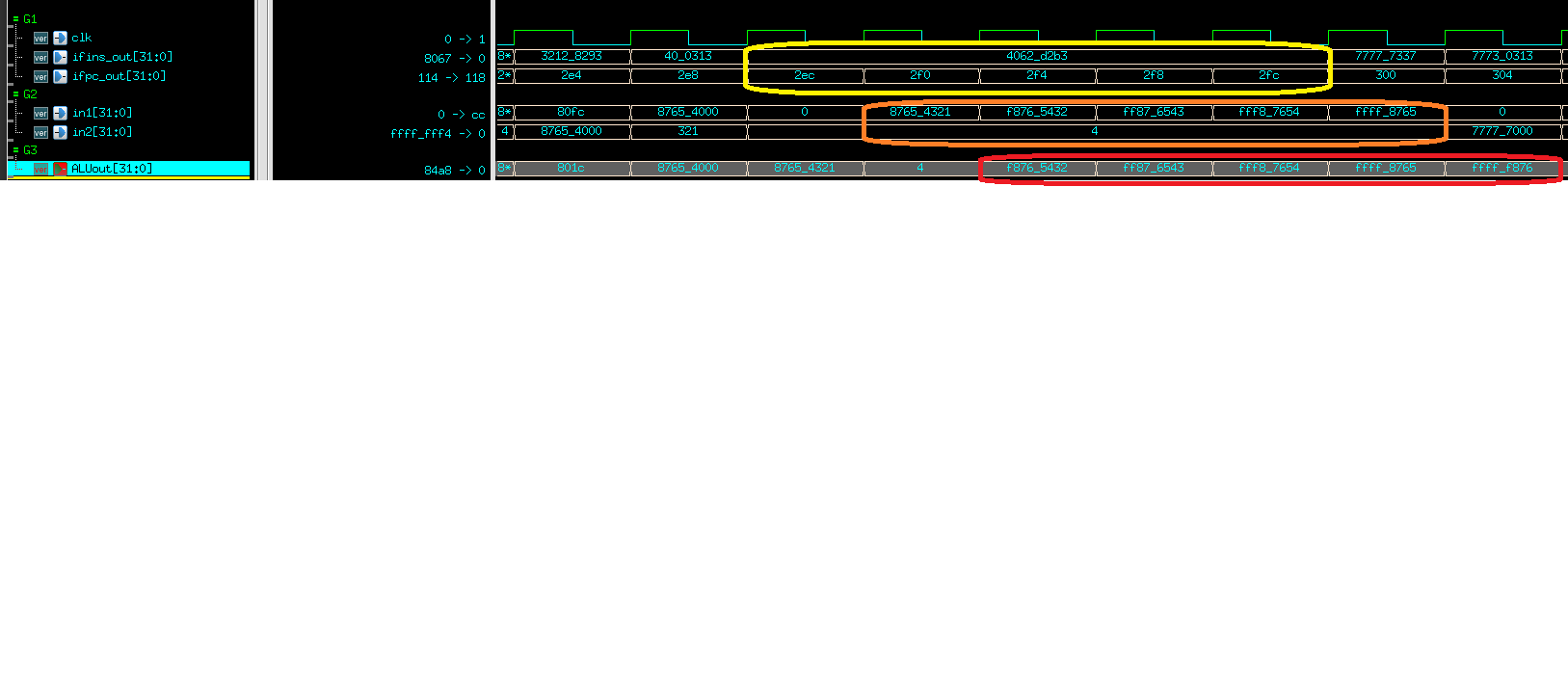
add

如上圖黃色表示IF的輸出，橘色表示ID的輸出，紅色表示EXE的輸出，此處為連續add運算，以第一筆資料為例，輸入為:ffff\_ffff與ffff\_ffff，輸出為ffff\_fffe。

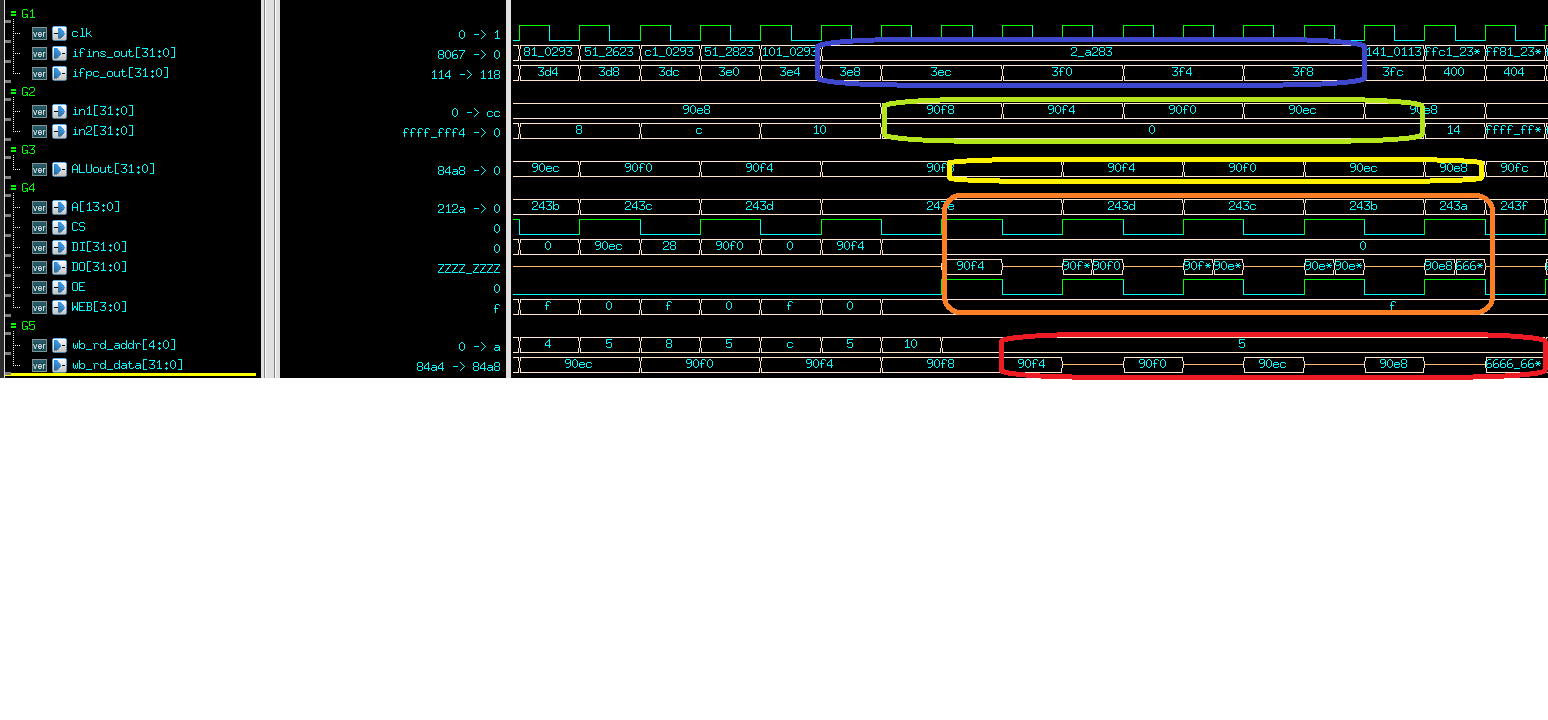
xor

如上圖黃色表示IF的輸出，橘色表示ID的輸出，紅色表示EXE的輸出，此處為連續xor運算，以第一筆資料為例，輸入為:ffff\_ffff與f0f0\_f0f0，輸出為0f0f\_0f0f。

sra

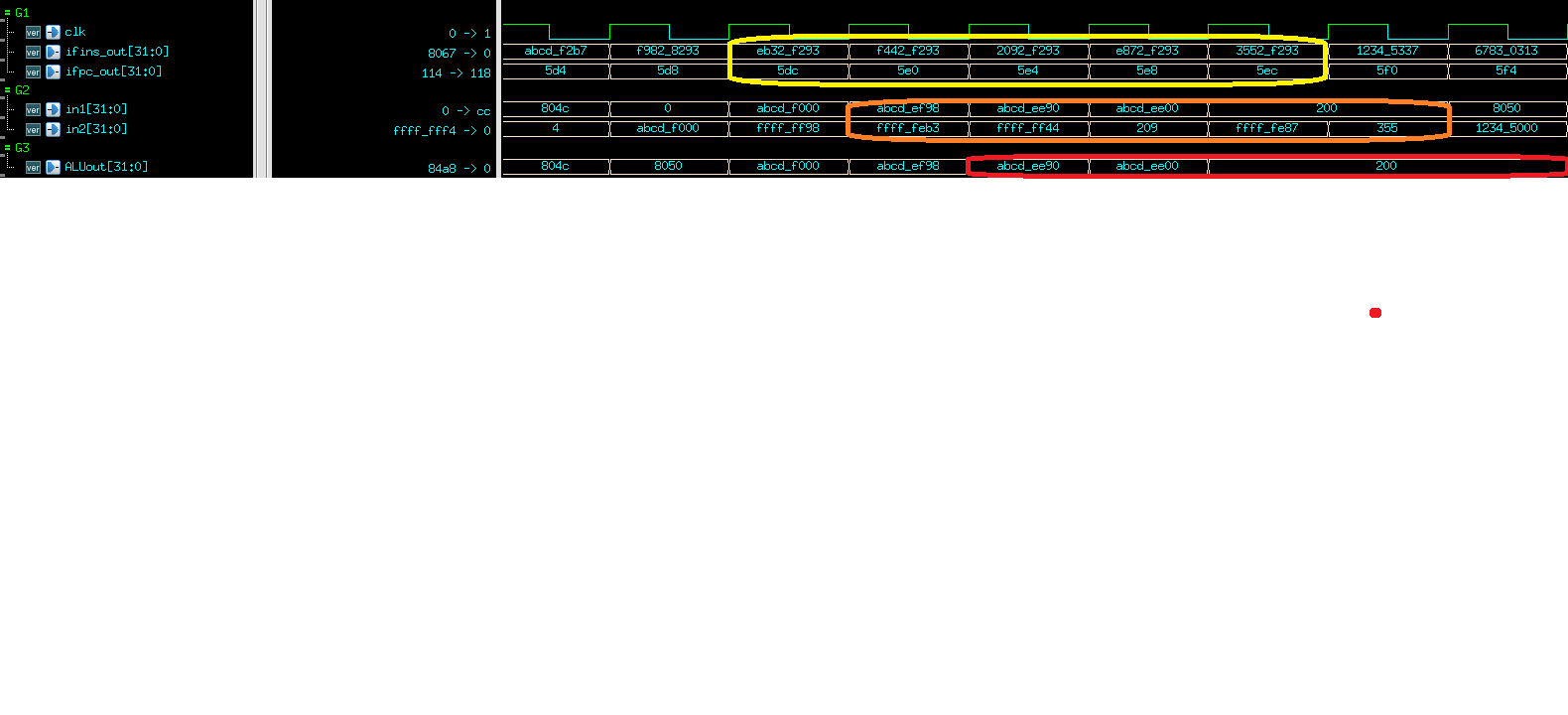
如上圖黃色表示IF的輸出，橘色表示ID的輸出，紅色表示EXE的輸出，此處為連續sra運算，以第一筆資料為例，輸入為:8765\_4321與4，輸出為f876\_5432。

2.I-type

lw

如上圖藍色表示IF的輸出，綠色表示ID的輸出，黃色表示EXE的輸出，橘色表示data memory的輸出(所以與EXE在同一個clock)，紅色表示WB的輸出，此處為連續lw運算以第一筆資料為例，要執行lw t0,0(t0)，所以ALUout為t0的值+0，取ALUout [15:2]bit，得到243e，送入memory的address，因為要讀取，所以CS=1、OE=1、WEB=1111，並讀出值90f4，在WB級存回t0 register，值得注意的是，因為是連續寫入同一個register，所以會出現load-use data hazard，所以後續的指令都會先暫停一個clock再執行。

Andi



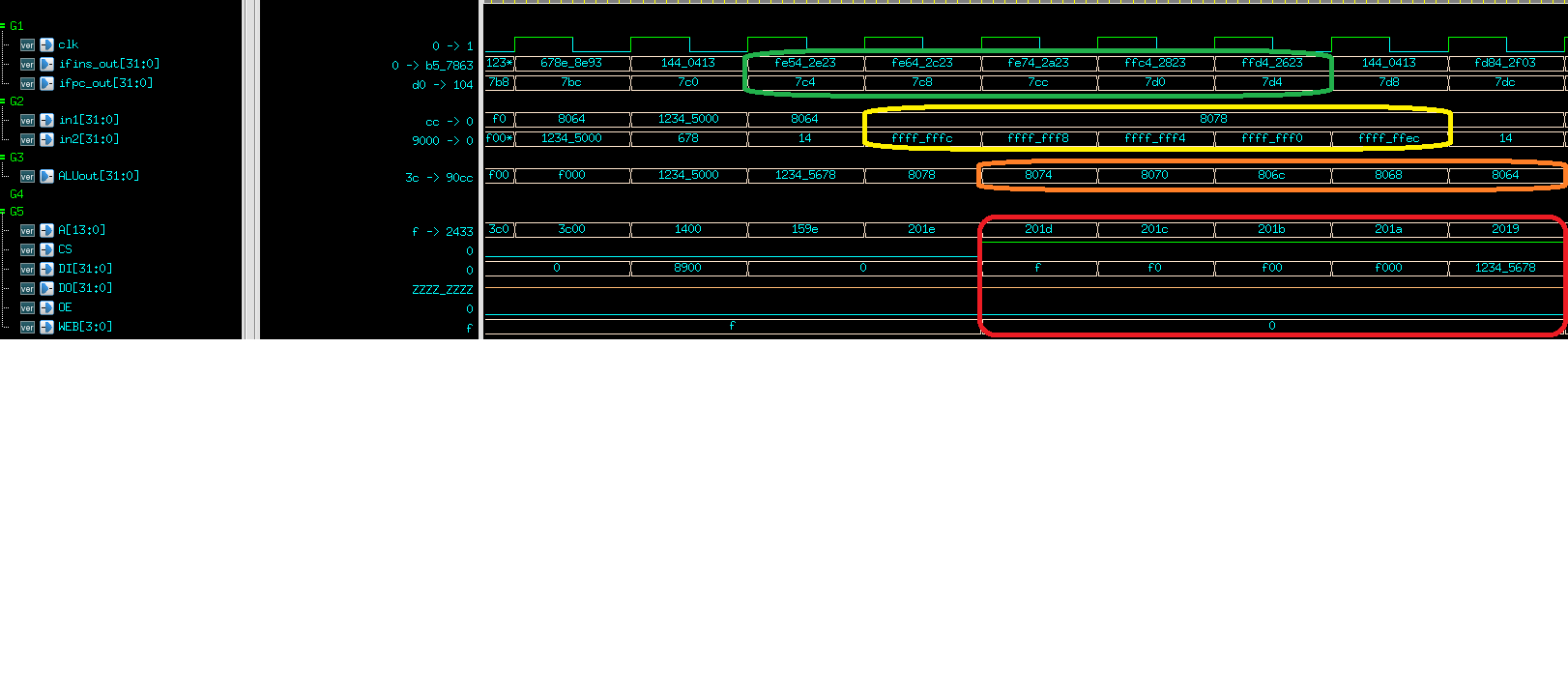
如上圖黃色表示IF的輸出，橘色表示ID的輸出，紅色表示EXE的輸出，此處為連續andi運算，以第一筆資料為例，輸入為:abcd\_ef98與ffff\_feb3，輸出為abcd\_ee90

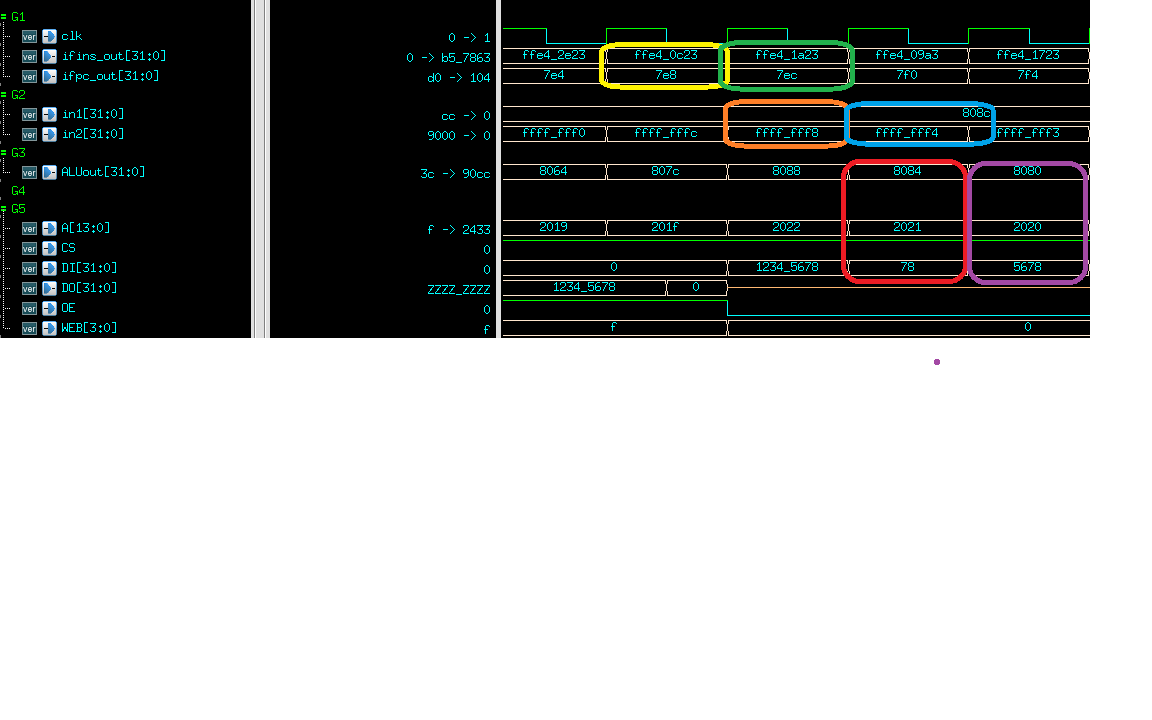
jalr

如上圖黃色表示IF的輸出，橘色表示ID的輸出，紅色表示EXE的輸出，此處為jalr運算，左邊藍色為branch的控制線，branch=2代表PC=ALUout，輸入為 744與0，輸出為744，此值會傳到PC，所以在EXE的下一個clock就跳到744，而RDsrc=代表rd的值為exe\_pctoreg，所以圖中可以看到rd=730(72c+4)。

3.S-type

sw

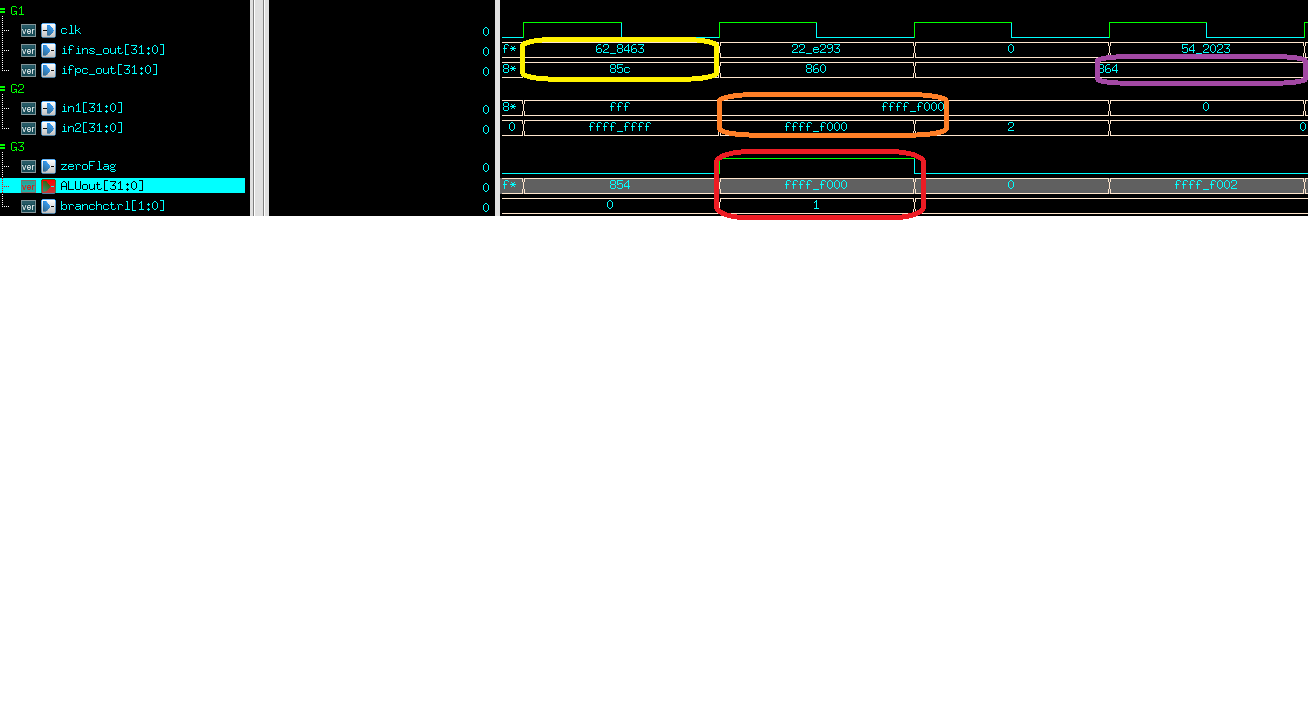
如上圖綠色表示IF的輸出，黃色表示ID的輸出，橘色表示EXE的輸出，紅色表示data memory的寫入(所以與EXE在同一個clock)，此處為連續sw運算，以第一筆資料為例，要執行sw t0,-4(s0)，所以ALUout為s0的值-4，取ALUout [15:2]bit，得到201d，送入memory的address，因為要寫入，所以CS=1、OE=0、WEB=0000，並將t0(f)寫入memory。

sb&sh

sb:如上圖，黃色表示IF的輸出，橘色表示ID的輸出，紅色表示EXE的輸出與data memory的寫入(所以與EXE在同一個clock)，要執行sb t5,-8(s0)，所以ALUout為s0的值-8，取ALUout [15:2]bit，得到2021，送入memory的address，因為要寫入，所以CS=1、OE=0、WEB=0000，並將t5(1234\_5678)的一bit 78寫入memory，此處的作法是。

sh:如上圖，綠色表示IF的輸出，藍色表示ID的輸出，紫色表示EXE的輸出與data memory的寫入(所以與EXE在同一個clock)，要執行sh t5,-12(s0)，所以ALUout為s0的值-12，取ALUout [15:2]bit，得到2020，送入memory的address，因為要寫入，所以CS=1、OE=0、WEB=0000，並將t5(1234\_5678)的一半5678寫入memory。

4.B-type

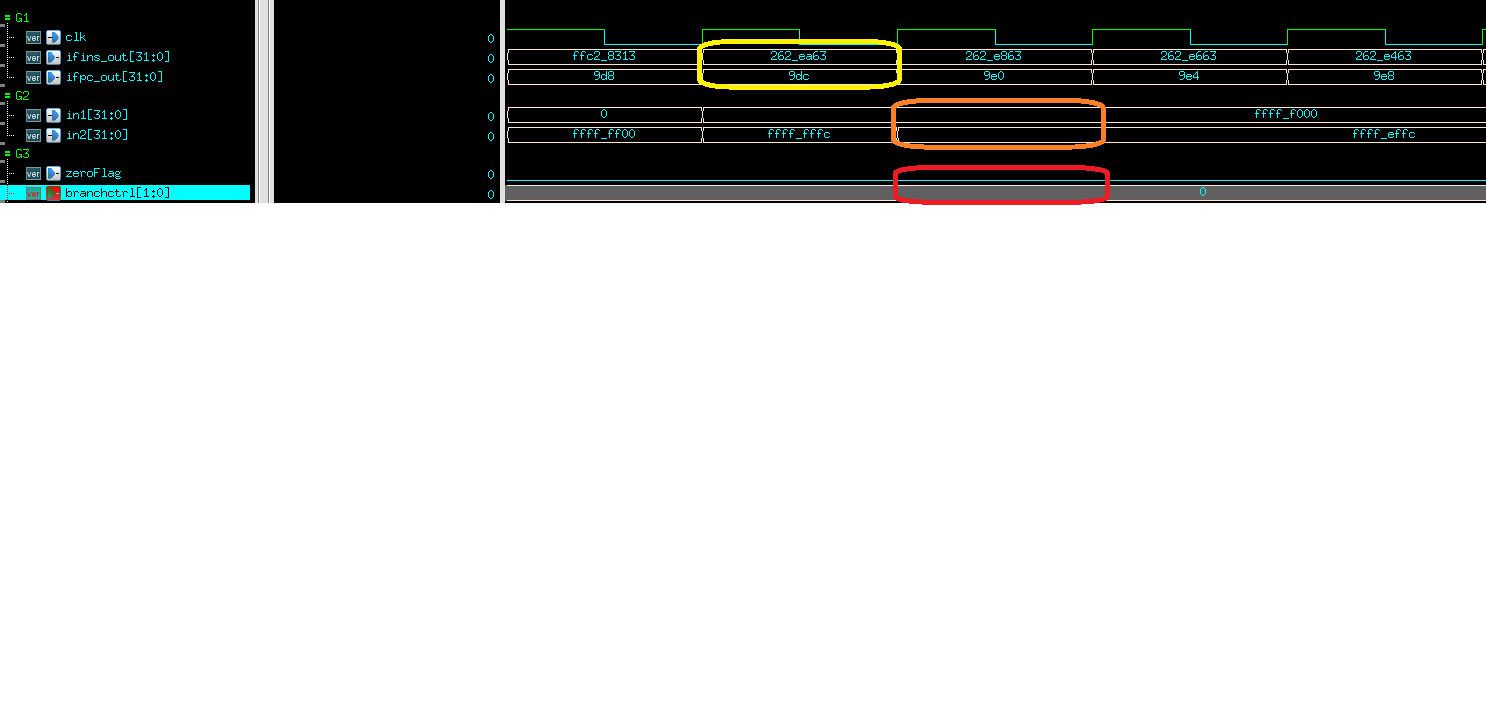
beq

如上圖，黃色表示IF的輸出，橘色表示ID的輸出，紅色表示zeroflag判斷與branch control(因為沒有設register所以與ID在同一個clock)，要執行beq t0,t1,864，alu的輸入為ffff\_f000與ffff\_f000，因為輸入相同，所以zeroflag=1要跳躍，因此branch control=1，在紫色框框中跳到864，會洗掉beq的下兩個指令，為pc=860與pc=864。

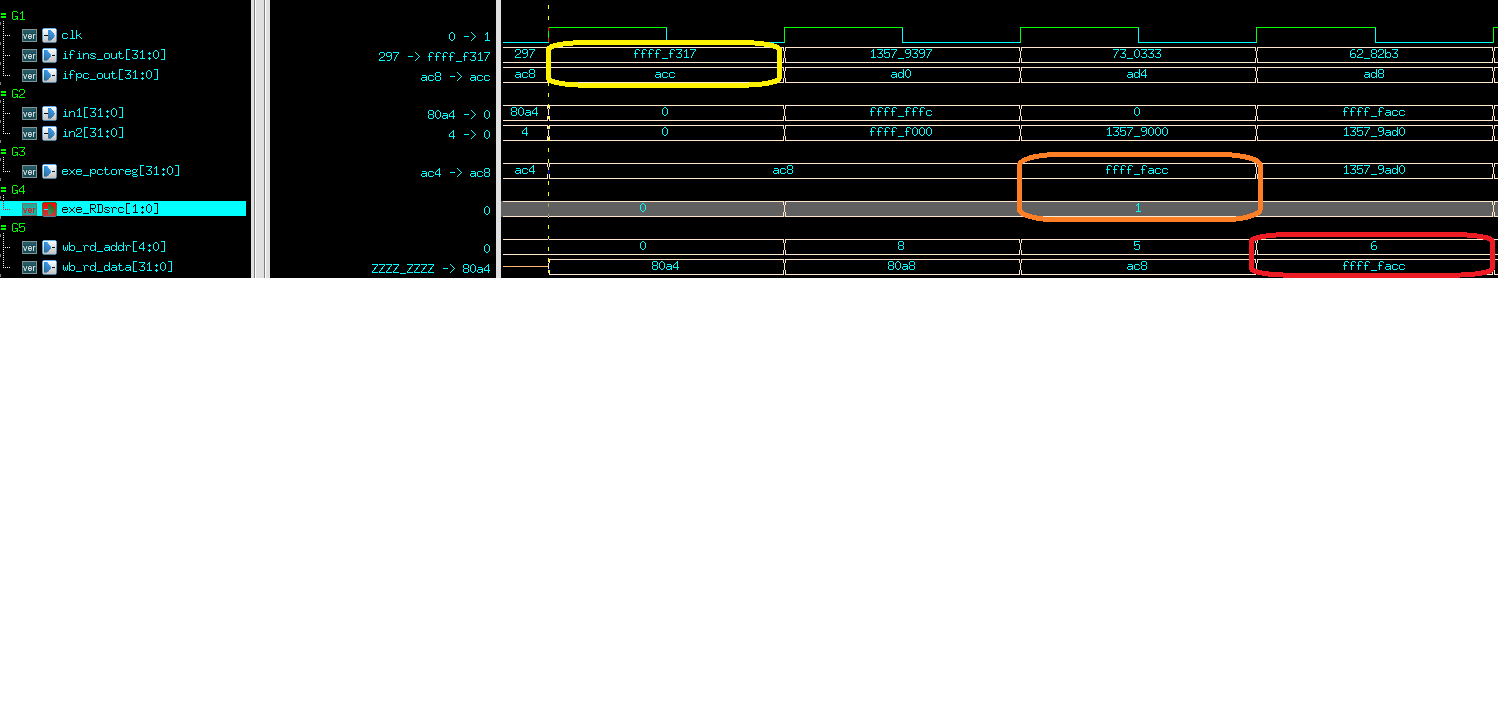
bne

如上圖，黃色表示IF的輸出，橘色表示ID的輸出，紅色表示zeroflag判斷與branch control(因為沒有設register所以與ID在同一個clock)，要執行bne t0,t1,8b4，alu的輸入為ffff\_f000與ffff\_f000，因為輸入相同，所以zeroflag=0不跳躍，因此branch control=0，pc=pc+4，繼續執行下個指令。

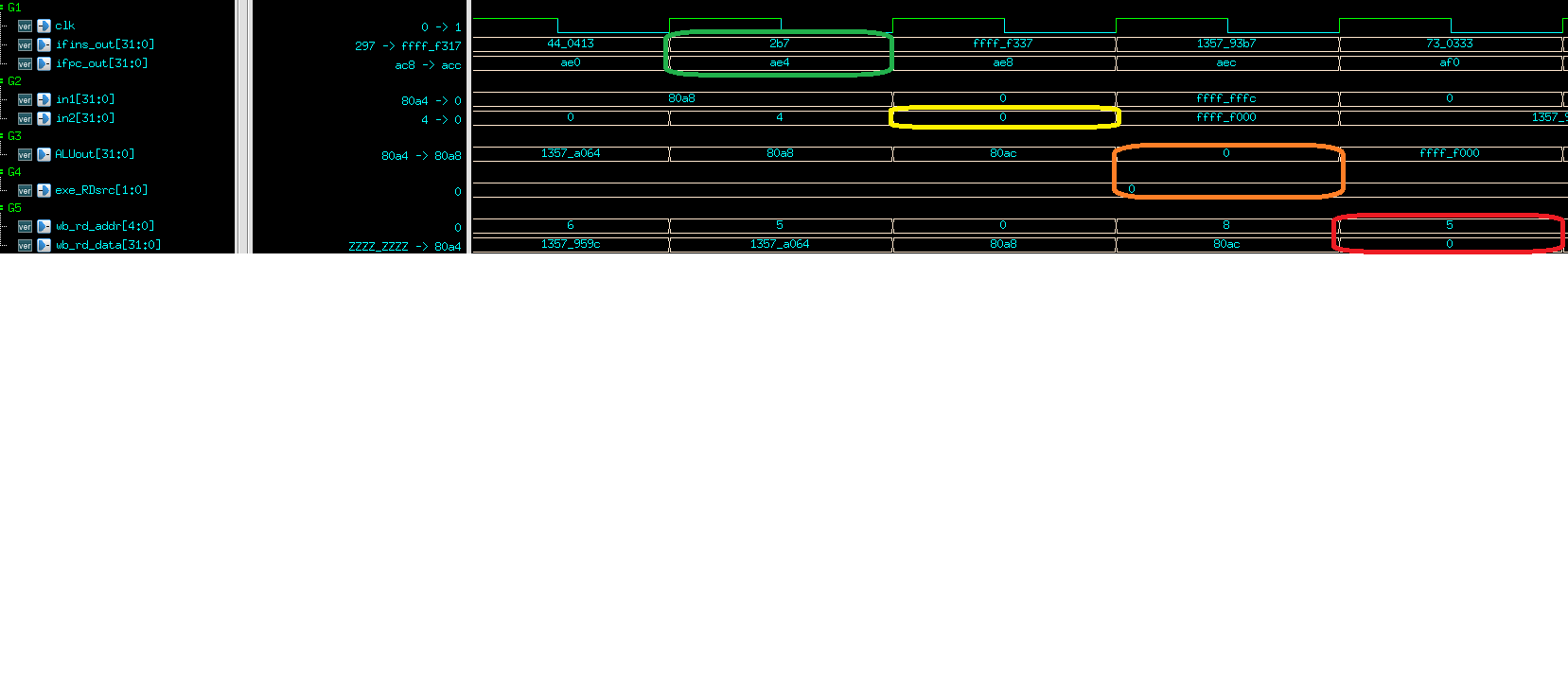
bltu

如上圖，黃色表示IF的輸出，橘色表示ID的輸出，紅色表示zeroflag判斷與branch control(因為沒有設register所以與ID在同一個clock)，要執行bltu t0,t1,a10，alu的輸入為ffff\_f000與ffff\_effc，因為in1<in2(此處為unsigned運算)，所以zeroflag=0不跳躍，因此branch control=0，pc=pc+4，繼續執行下個指令。

5.U-type

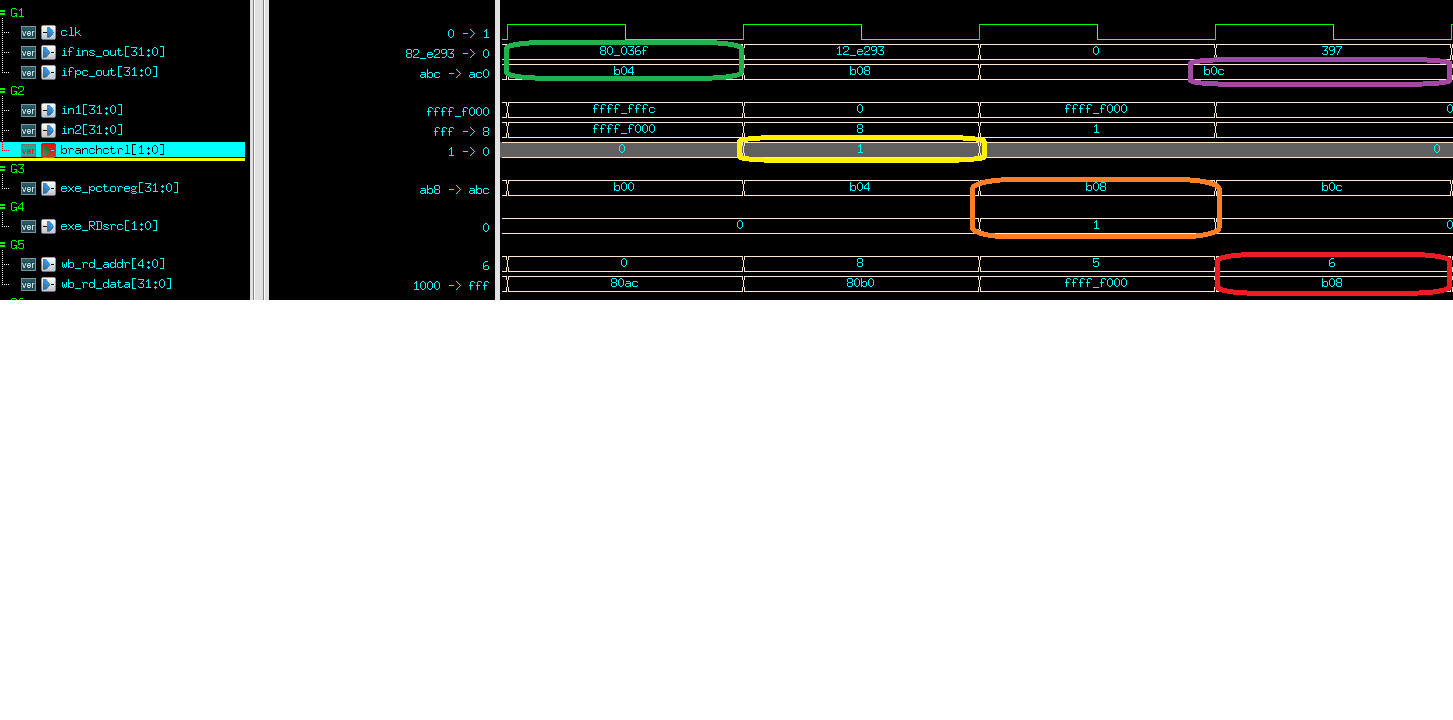
alupc

如上圖，黃色表示IF的輸出，橘色表示EXE的輸出，紅色表示WB的值，此處執行alupc t1,0xfffff，所以這邊exe輸出pc+0xfffff，RDsrc=1，並在wb級將ffff\_facc寫入t1 register。

lui

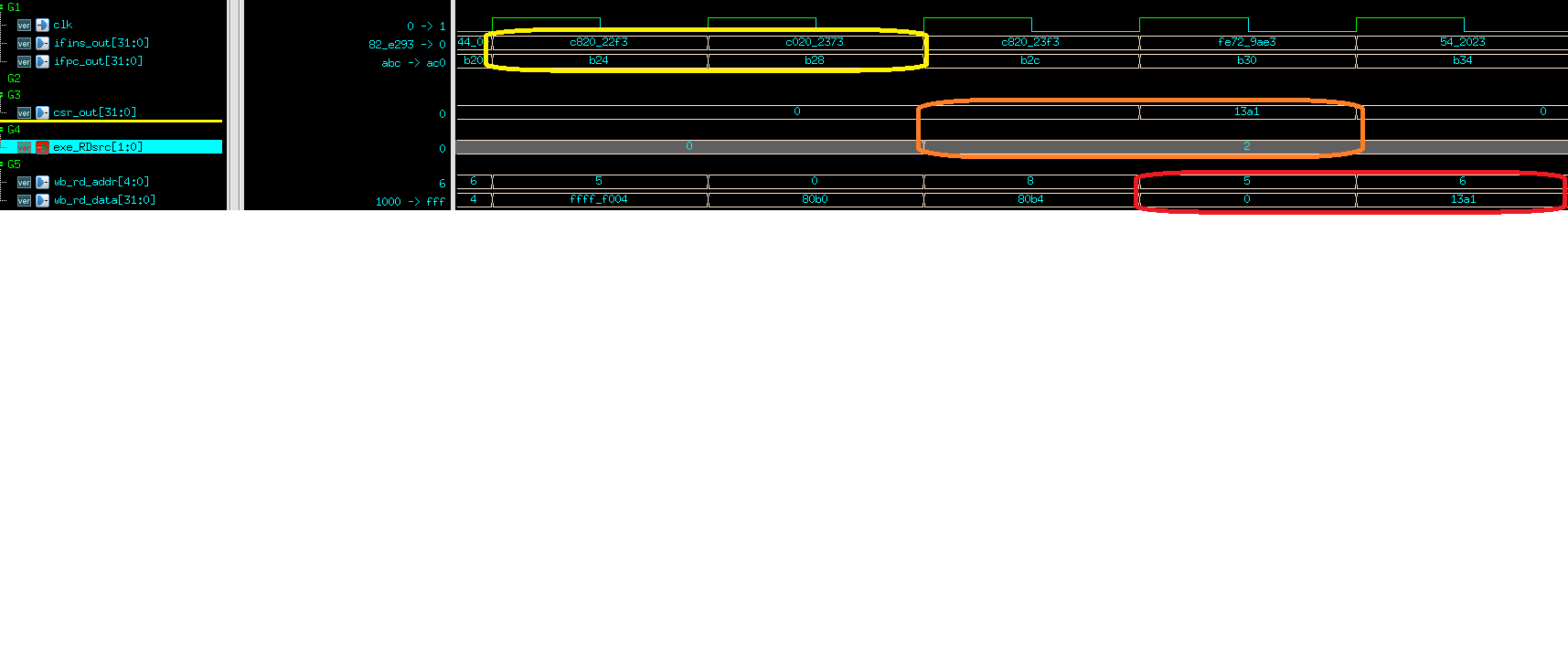
如上圖，綠色表示IF的輸出，黃色表示ID的輸出，橘色表示EXE的輸出，紅色表示WB的值，此處執行lui t0,0x0，所以這邊exe輸出等於imm，也就是黃色框框的值，RDsrc=0，並在wb級將0寫入t0 register。

6.J-type

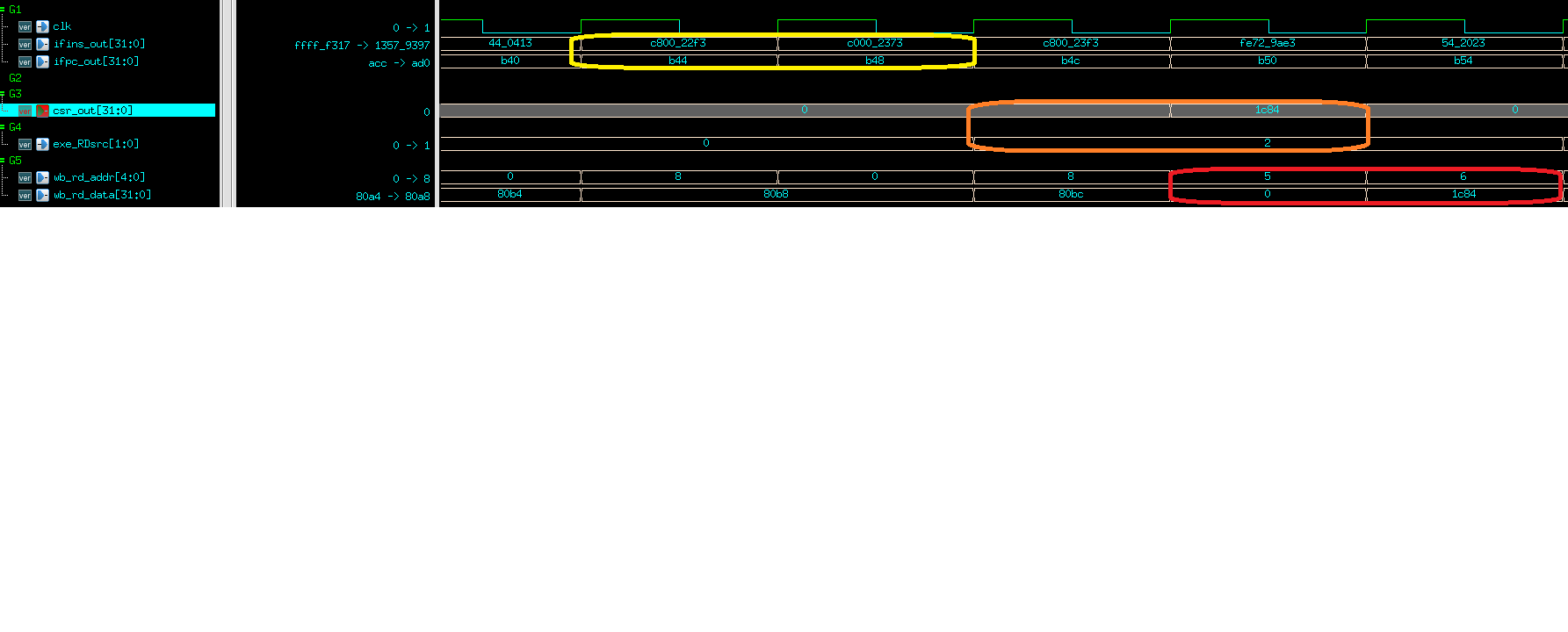
jal

如上圖，綠色表示IF的輸出，黃色表示branch control=1，所以pc=pc+imm，橘色表示EXE的輸出，而RDsrc=1所以MEM級會將PC+4的值，也就是b08傳給WB，紅色表示WB的值，此處執行jal t1,b0c，所以這在exe的下一級PC=b0c，也就是紫色框框，並在wb級將b08寫入t1 register。

7.CSR

rdinstreth&rdinstret

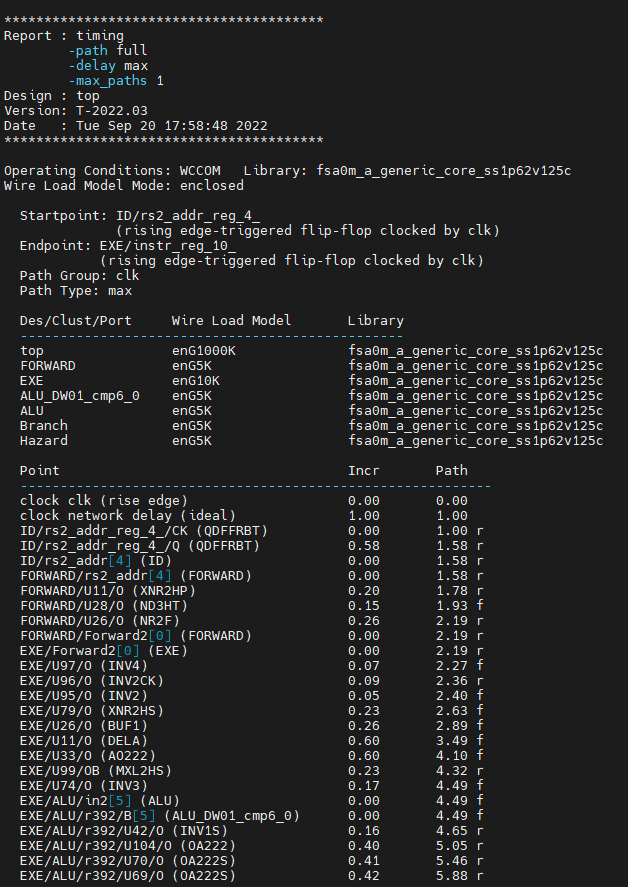
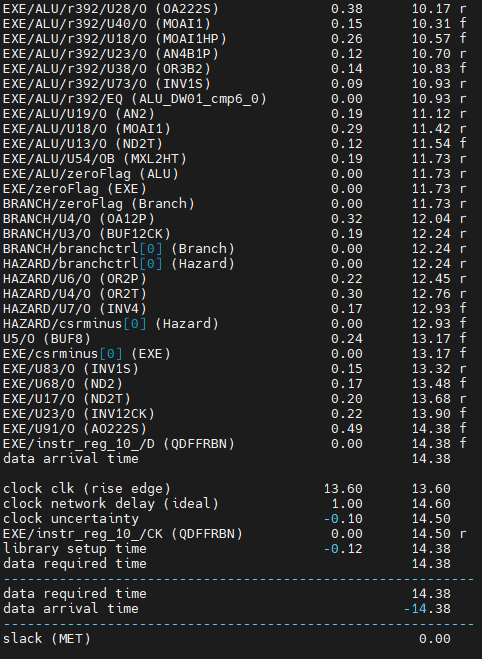
如上圖黃色表示IF的輸出，橘色表示EXE的輸出RDsrc=2，所以會把csrout的output傳給WB，紅色表示WB的寫入，b24為rdinstreth，b28為rdinstret，所以會將計算的instret的[63:32]bit寫入t0、[31:0]bit寫入t1，而csr的計算寫在lessons learned。

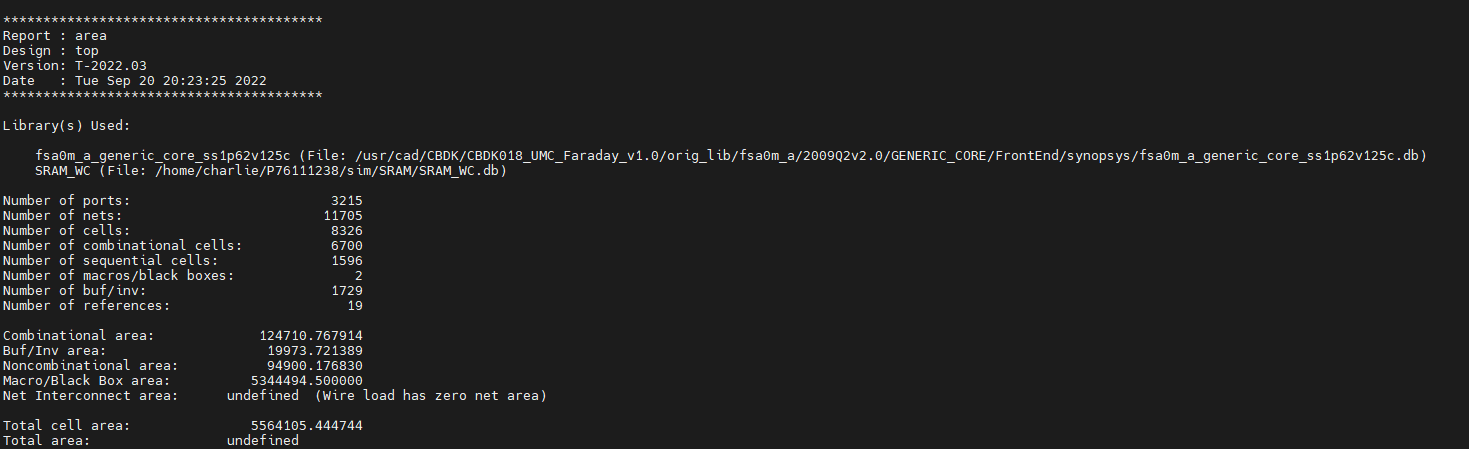
rdcycleh& rdcycle

如上圖黃色表示IF的輸出，橘色表示EXE的輸出RDsrc=2，所以會把csrout的output傳給WB，紅色表示WB的寫入，b44為rdcycleh，b48為rdcycle，所以會將計算的cycle的[63:32]bit寫入t0、[31:0]bit寫入t1，而csr的計算寫在lessons learned。

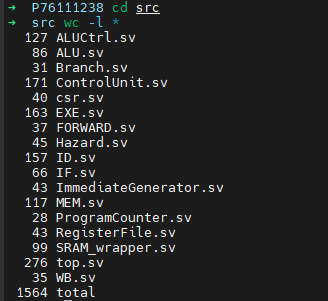
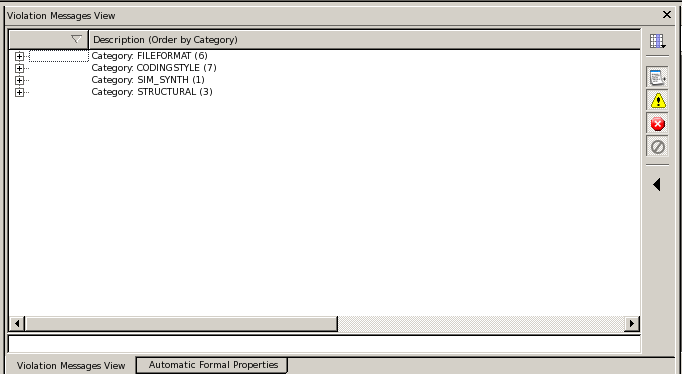
四、Report

1.timing report

2.area report  


3. Superlint

總warning數:17

總行數:1564

Warning所占比例:1%

修改過的warning:

\*檔名與module名不同

\*if條件沒寫滿，產生latch

4. Simulation results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Prog0 | Prog1 | Prog2 | Prog3 | Prog4 |
| RTL | Pass | Pass | Pass | Pass | Pass |
| SYN | Pass | Pass | Pass | Pass | Pass |

5.Performance

|  |  |
| --- | --- |
| Simulation time | 13.6ns |
| Total Cell Area | 5564105.444744 |

五、Summary

以前在計算機組織學過MIPS的5階pipeline CPU，學習到各個階段所負責的工作以及因為pipeline的設計會遇到的問題，像是各種Hazard，但這次實作才更了解CPU的運作，也並非就是照著記憶中的結構做就結束了，要自己設計ALU，自己寫各種情況的判斷，過程費時但是很有趣，而且這次的CSR指令是之前從未學習過的，在查資料過後，才知道原來CSR有4096個register，雖然這次沒用到，但也讓我眼界大開，完成這個作業後，除了疲倦以外也感受到踏實感，終於能把書上的內容變成一行一行的程式碼，令我很欣慰。