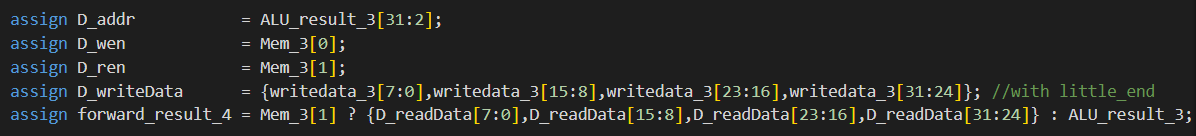
**2022.04.26**

------------------------------------------------------------------------------------------------

**Assertion Progress**

------------------------------------------------------------------------------------------------

* 這週進度先按照一個流水線標準的 hazard handling 寫 assertion
* 這部分完成後會依照正常時候的功能運作寫 assertion
  + now → if (signal) ? hazard\_handling : 1
    - 因為 always block 是 clk-depending 的, 每個 clk 會有不同值 assign 進來, 所以 if (!signal), 如果要寫"reg[address] == some\_data", 必須寫在 always block 裡面並用"assert (<expression>)", 而不是寫在 module 裡面, 這部分尚未實作
    - 要寫"reg[address] == some\_data"的困難處一是設計時由於 data 操作的方便性, designer 可能會把 data 做一些處理, 而不會單純該 reg 為甚麼值就是甚麼值, 所以可能如下圖, instruction 應該為 instru[31:0], 但因為一些操作, 讓內部的 immediate, rs, rd, func7, func3, opcode 等順序反過來 (e.g. @ stage 4) 
  + modify to → if (signal) ? hazard\_handling : 一般操作
  + 並加上其他一般功能
  + currently no assertion for cache operation
* 再來就是為避免被 src code 的判斷式影響, 寫更 hierarchy 高的 assertion 去包他
* 依舊用 yosys assertion, 因為方便 parse 完後不用轉格式, 轉變數, 直接寫在想要 assert 的地方即可
* 感覺到時候寫 assertion 不太像是一個太可依賴的辦法, 因為把重要的 handling 或判斷式 assert 下去, 加上變數統一配合 design 使用, 其實跟原本 design 的寫法跟判斷的東西基本上沒差多少, 所以感覺 design 寫好, assert 也寫完了 (?)

------------------------------------------------------------------------------------------------

**Assertion Note**

------------------------------------------------------------------------------------------------

* 因為粗略的 assertion 是按照正規步驟, 以及按照原始碼自訂的變數去 handle if hazard 等發生如何處理, 這些大致跟 Jeter 的 code 一樣, 所以如果要 insert bug, 可能要考慮 insert 繞多圈一點的 bug, 否則可能要嘛偵測不到 forwarding bug (assertion 除非寫到這麼細), 要嘛直接被抓到 (assertion 包含該判斷式), 或是我 assertion 再想怎麼讓他更 hierarchy 而不是照著正規步驟 handle
* ALU, jump 等細部 ALUop 操作假定都正確的 (?), i.e. 這些較低級的 bug 不會是用我們的方法偵測出來, 因為一般的 tb 就可以抓到了
* 如果 assertion 會用到 internal reg, wire 的話, assert 會寫在 pipeline\_stageX.v 的檔案中, 如果只會用到該 module 的 I/O, 則將他寫在 global 的 RISCV\_pipeline.v 中

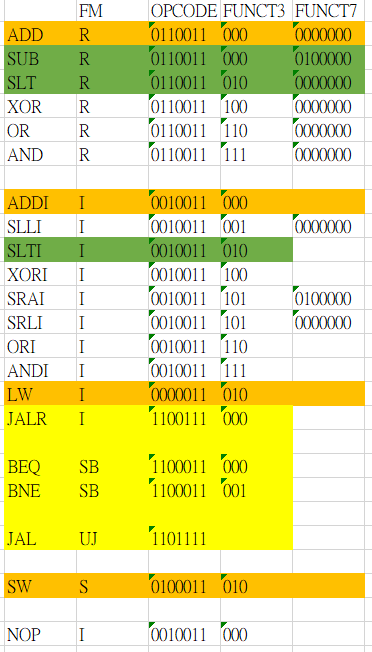
------------------------------------------------------------------------------------------------

**Assertion Implementation @ golden RISCV**

**(Consider CHIP.v > RISCV\_pipeline.v, no cache operation)**

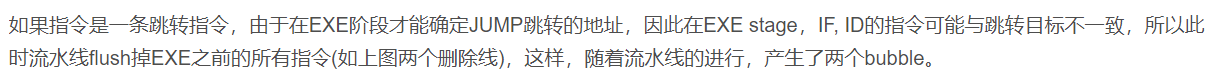
------------------------------------------------------------------------------------------------

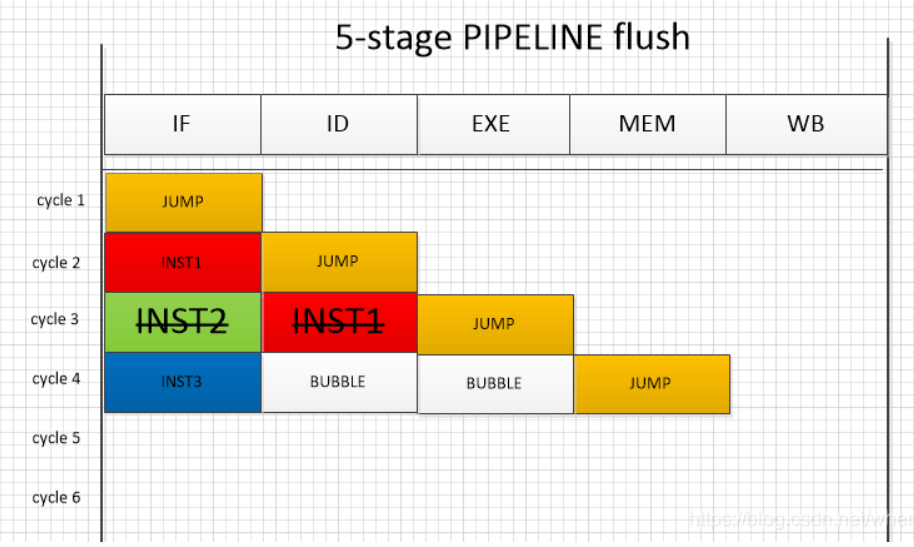
**Opcode Spec**

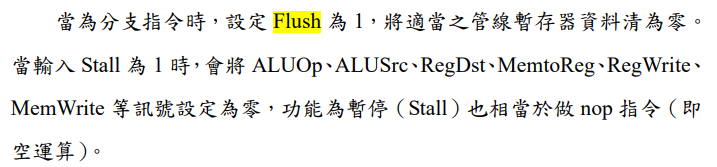


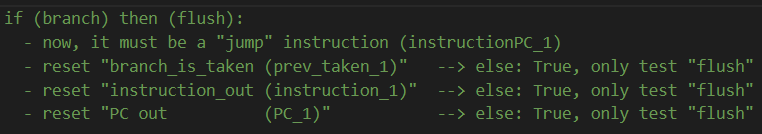
**Stage 1 (instruction fetch)**

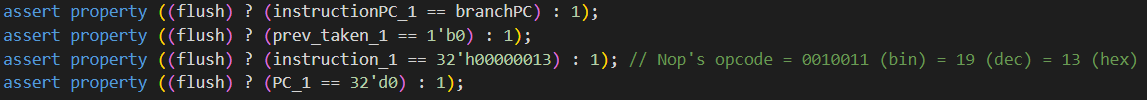
**[ branch & flush ]**





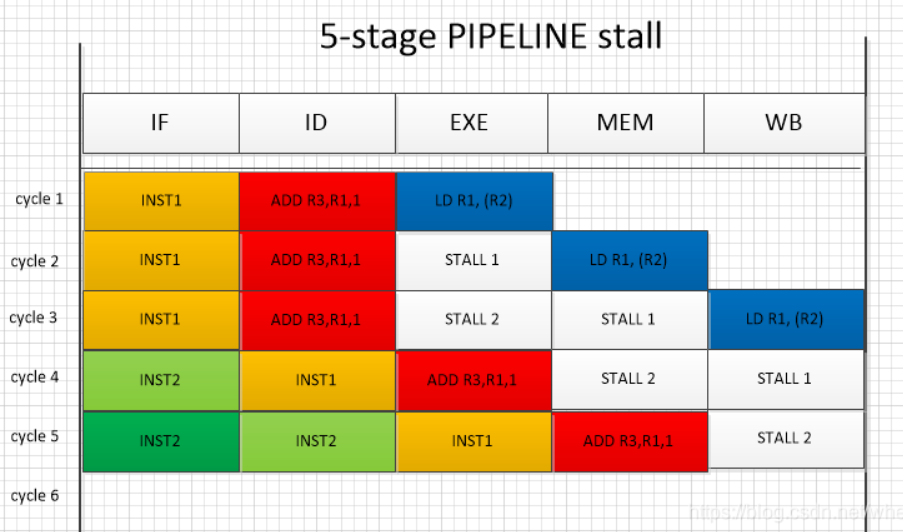




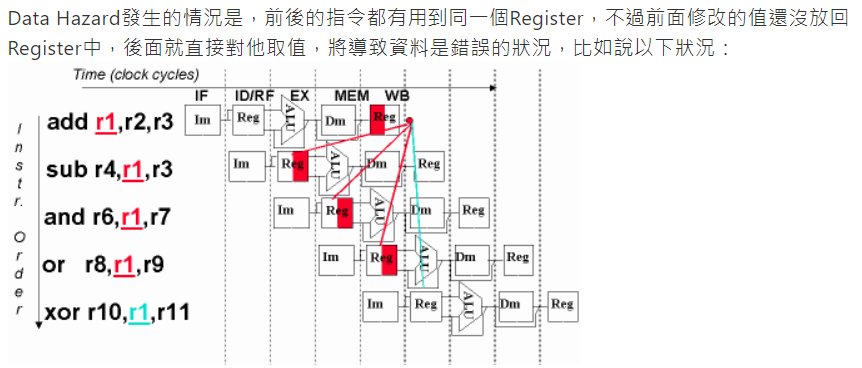


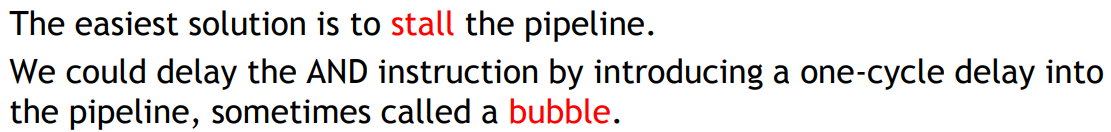
**[ memory stall ]**

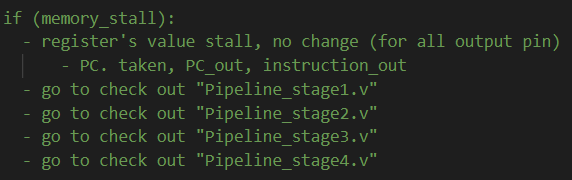
****

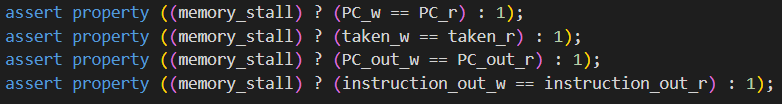
****

****



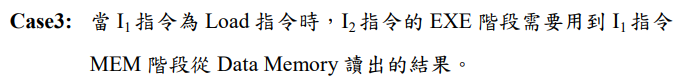
****

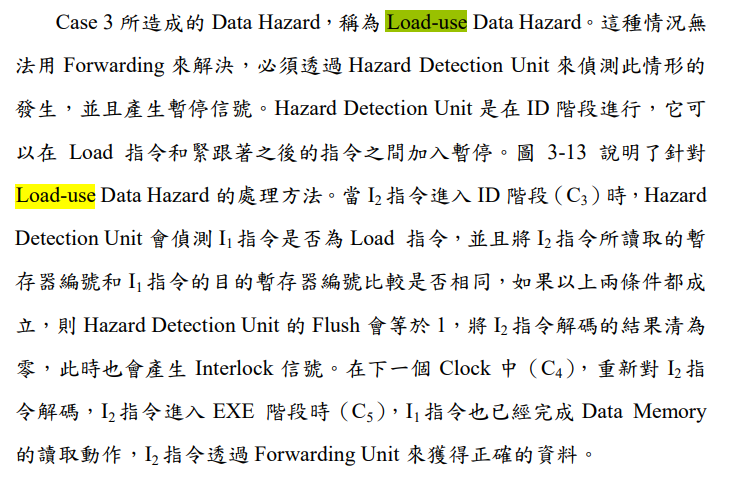


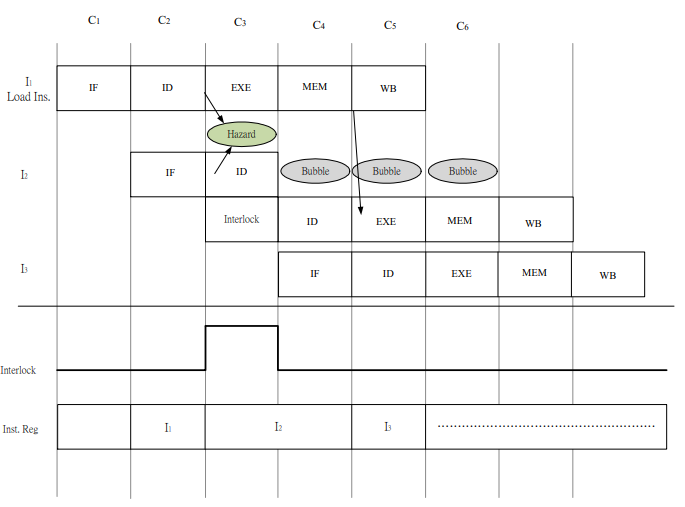


**[ load-use hazard ]**

* **Data Hazard, “Load” 指令需在 stage 4 才能從 memory access**

****

****

****

****

****

**[ if no special case ]**

* parameters will directly pass to stage 2, so no need assertion
  + PC\_1, instruction\_1, prev\_taken\_1

**[Stage 1 Simulation]**

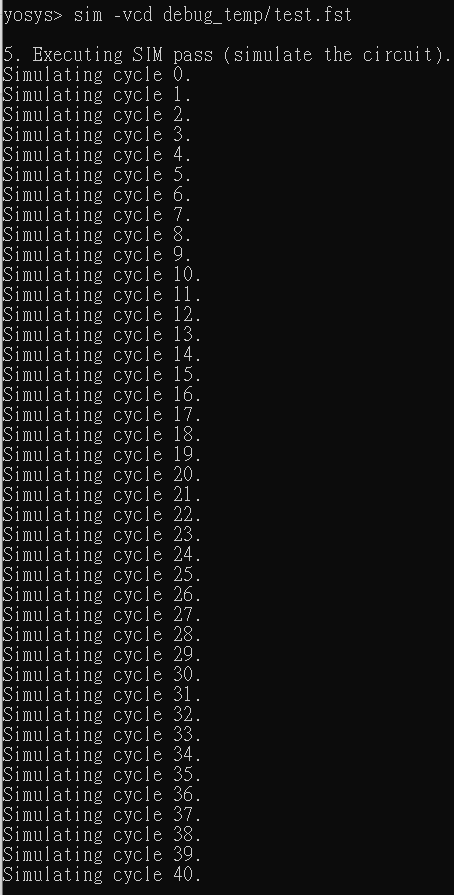
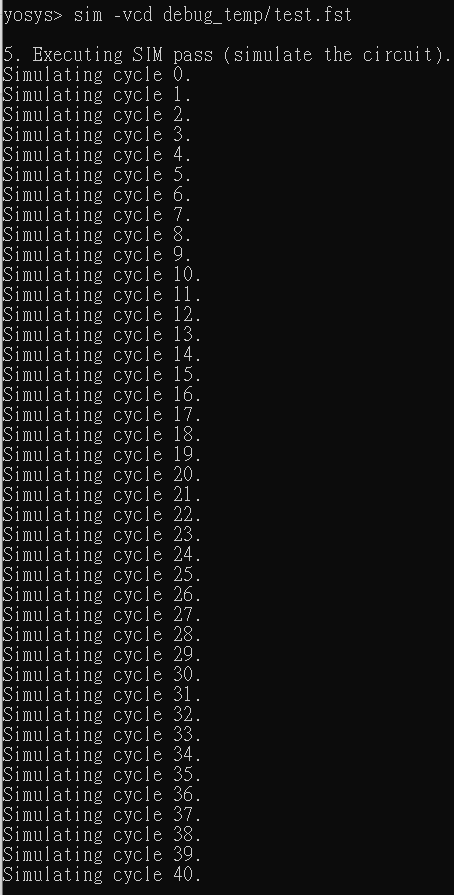
→ cd Baseline/src

→ read -sv ALUPipeline/RISCV\_pipeline.v

→ hierarchy -top RISCV\_Pipeline

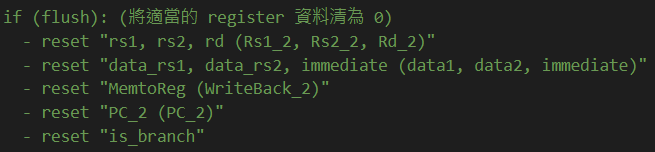
→ prep -top RISCV\_Pipeline

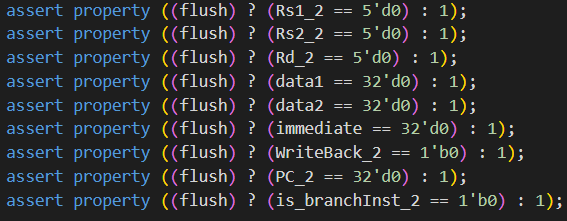
→ sim -vcd debug\_temp/test.fst (no bug is detected)

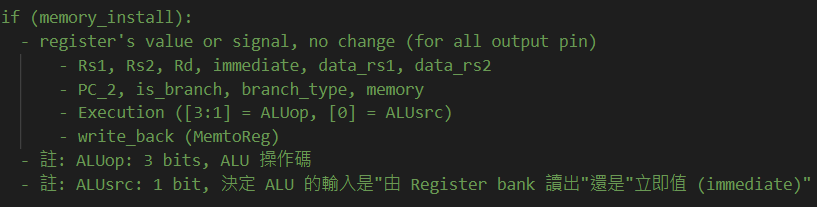
**Stage 2 (instruction decode)**

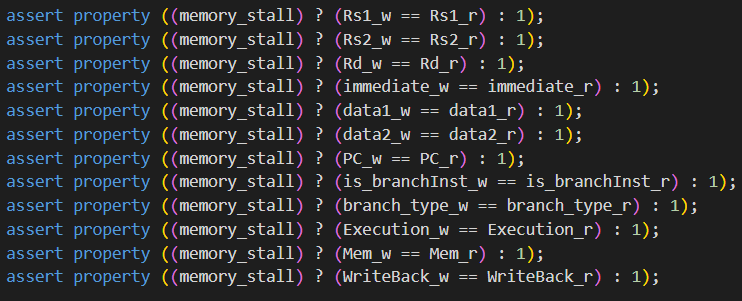
**[ flush ]**

****

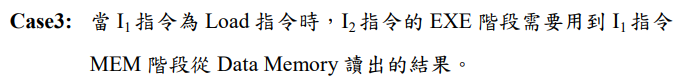
****

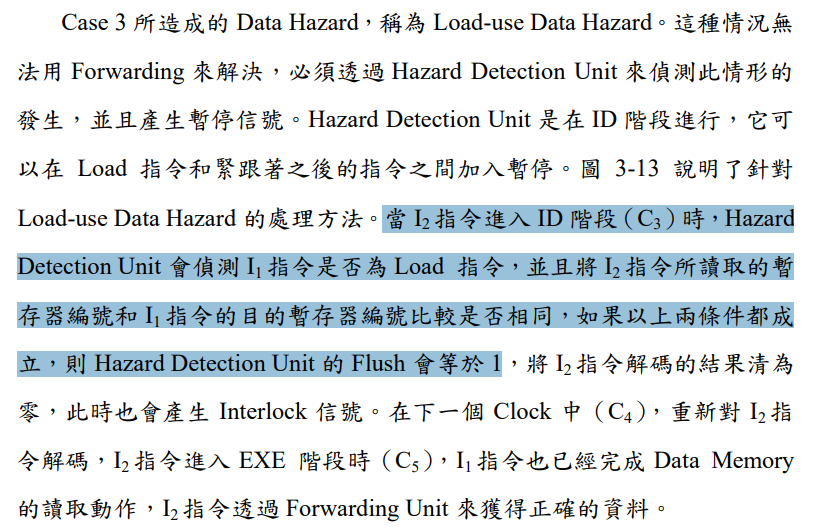
**[ memory stall ]**

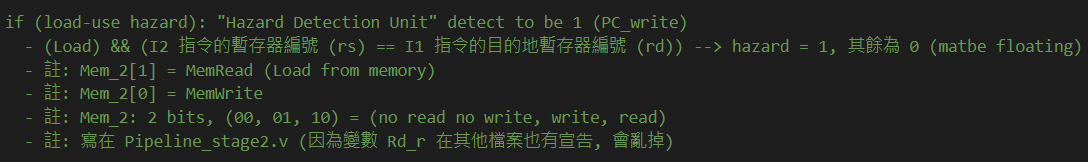
****

****

**[ data hazard → load-use hazard → handle by “interlock” ]**

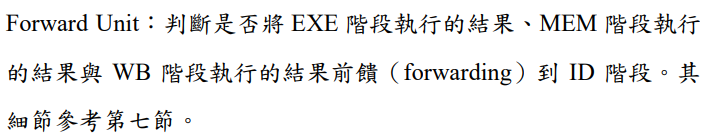
****

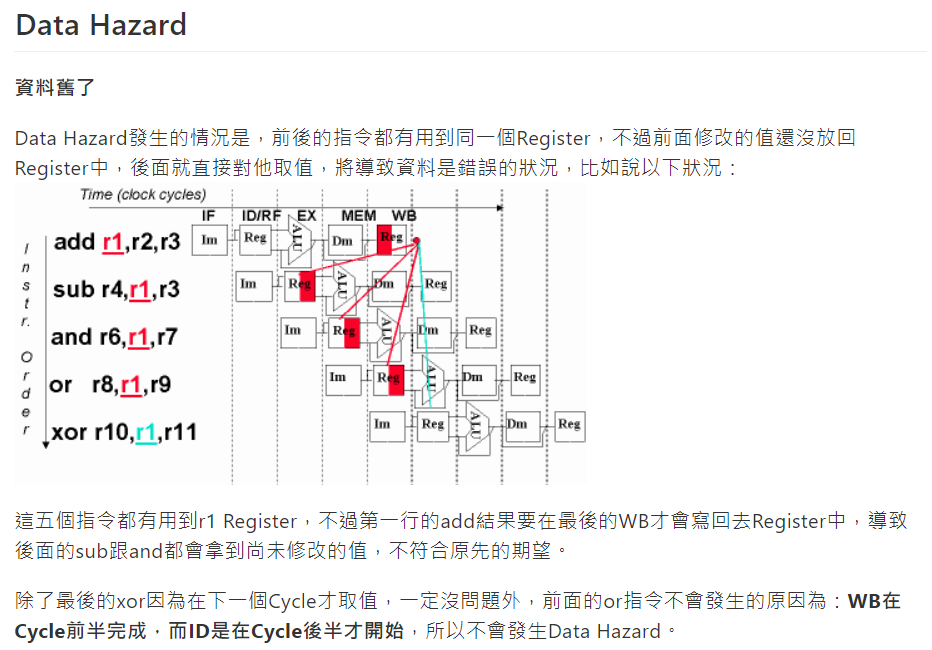
****

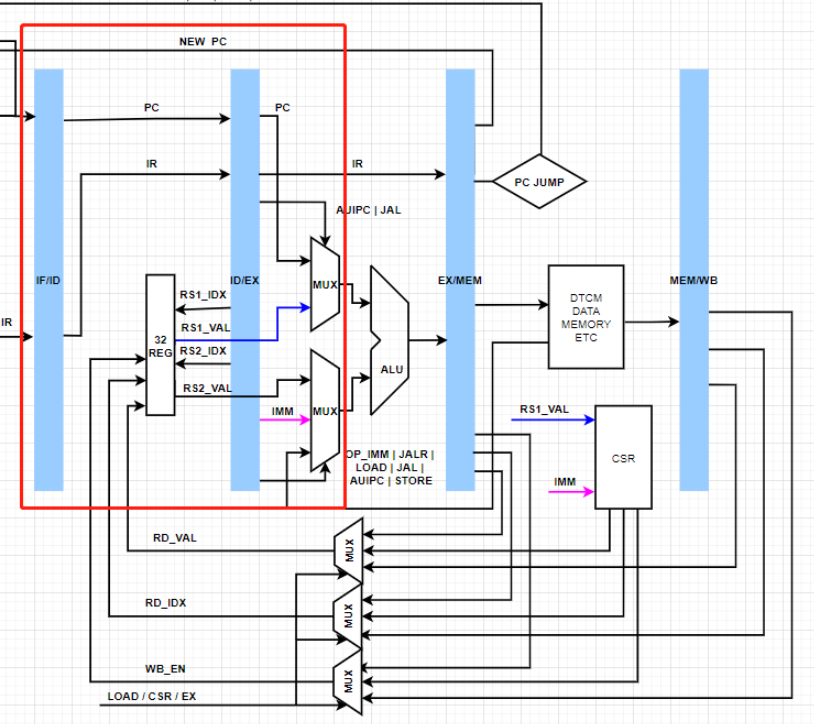
****

****

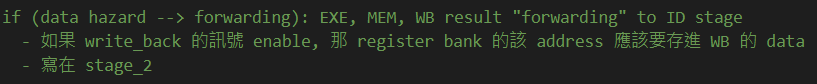
**[ data hazard → need undefined value → handle by “forwarding” ]**

****

****

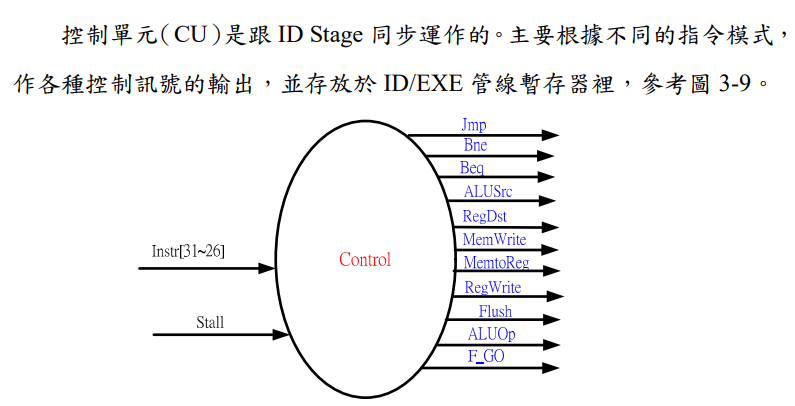
****

**(stage 5: drag wire to write\_back to register\_bank)**

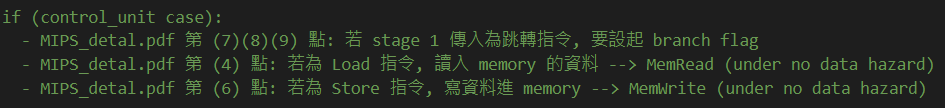
****

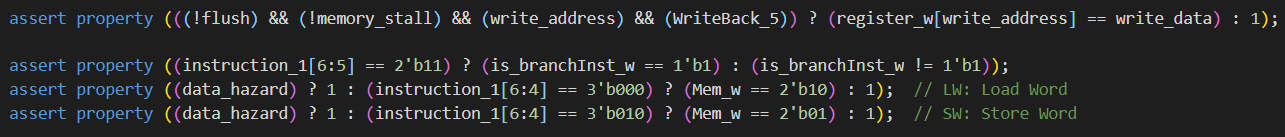
****

**[ control unit ]**

****

****

****

****

**[ if no special case ]**

* parameter will directly pass to stage 3, so no need assertion

**[Stage 2 Simulation]**

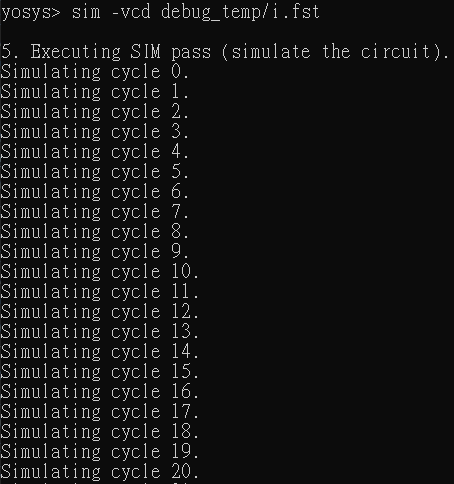
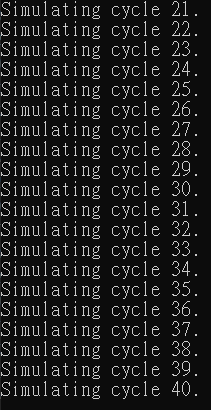
→ cd Baseline/src

→ read -sv ALUPipeline/RISCV\_pipeline.v

→ hierarchy -top RISCV\_Pipeline

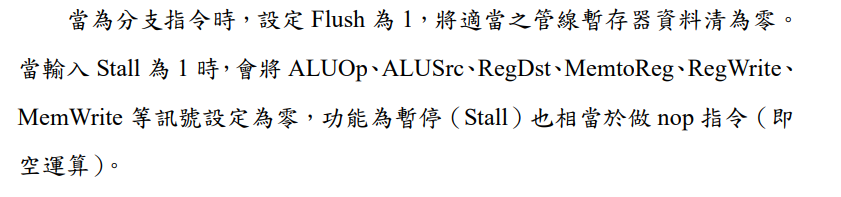
→ prep -top RISCV\_Pipeline

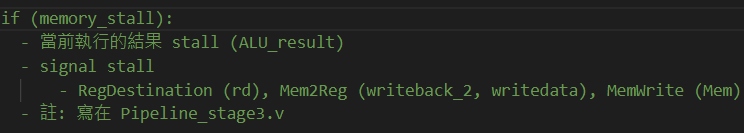
→ sim -vcd debug\_temp/test.fst (no bug is detected)

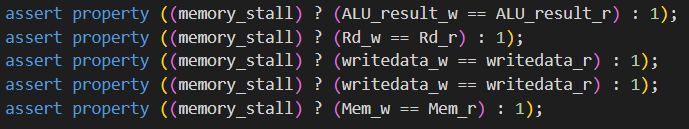
 

**Stage 3 (execute)**

**[ memory stall ]**

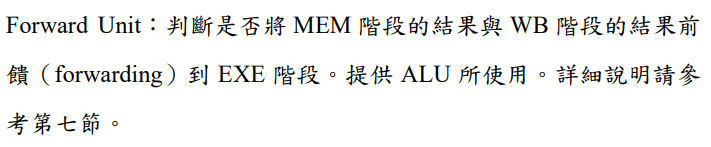
****

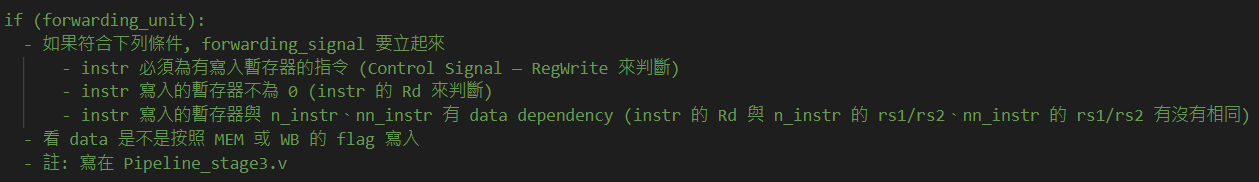
****

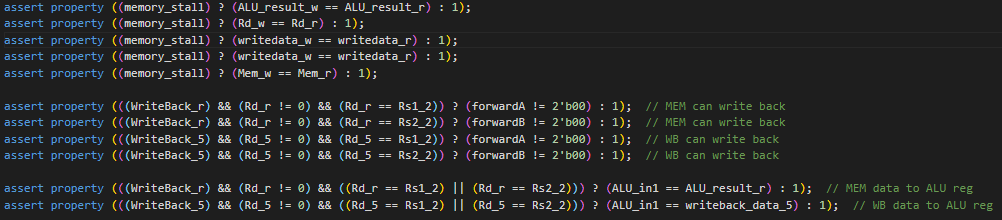
****

**[ forwarding unit ]**

****

****

****

****

**(consider data/reg/signal in MEM or in WB)**

**[ if no special case ]**

* 根據不同的 decode 指令結果做不同的運算

**[Stage 3 Simulation]**

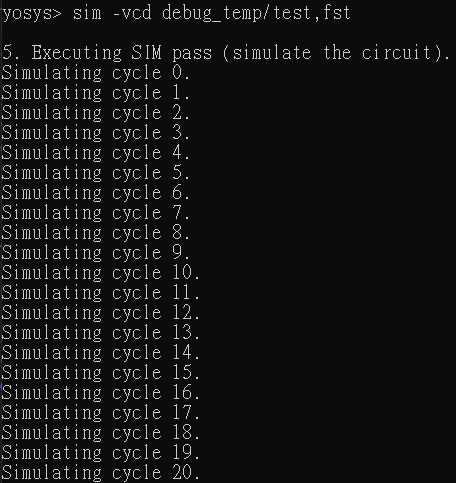
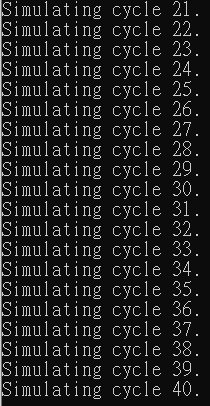
→ cd Baseline/src

→ read -sv ALUPipeline/RISCV\_pipeline.v

→ hierarchy -top RISCV\_Pipeline

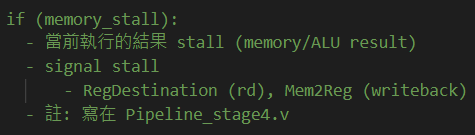
→ prep -top RISCV\_Pipeline

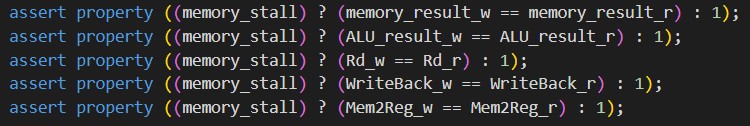
→ sim -vcd debug\_temp/test.fst (no bug is detected)

**Stage 4 (memory access)**

**[memory stall]**

****

****

**[ if nothing special ]**

* 按照 signal 指令賦值 (load, store)

**[Stage 4 Simulation]**

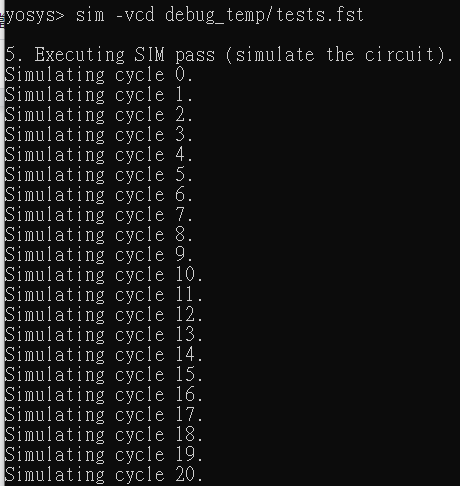
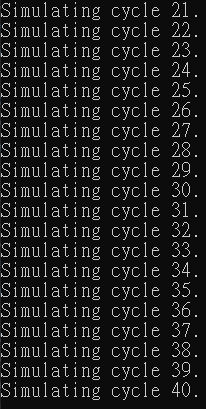
→ cd Baseline/src

→ read -sv ALUPipeline/RISCV\_pipeline.v

→ hierarchy -top RISCV\_Pipeline

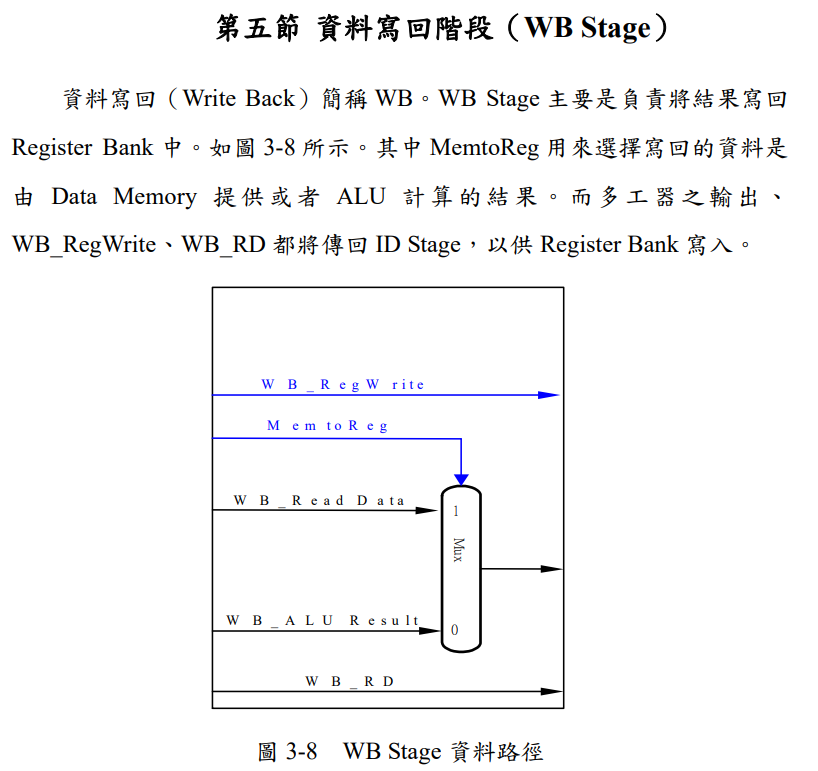
→ prep -top RISCV\_Pipeline

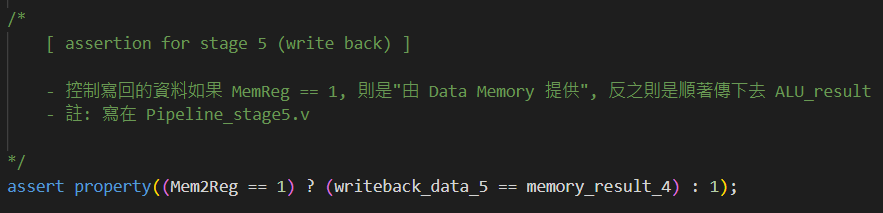
→ sim -vcd debug\_temp/test.fst (no bug is detected)

** **

**Stage 5 (writeback)**

**[ Mem2Reg signal to control write\_back\_data ]**

****

****

**(ALU\_result 的 data 有被處理過, 應該不是這麼簡單直接讓 assert (writeback\_data\_5 == ALU\_result)) → 因為這會噴 assertion error**

**[Stage 5 Simulation]**

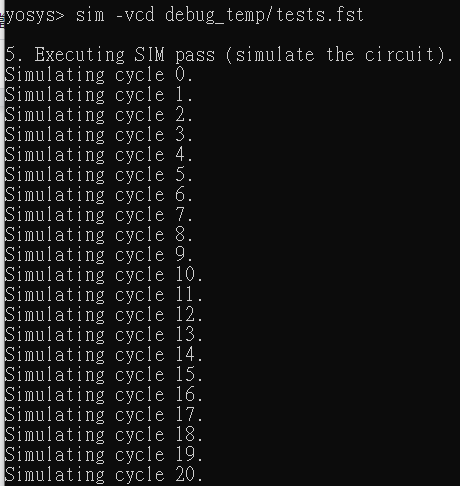
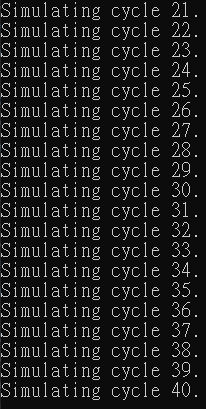
→ cd Baseline/src

→ read -sv ALUPipeline/RISCV\_pipeline.v

→ hierarchy -top RISCV\_Pipeline

→ prep -top RISCV\_Pipeline

→ sim -vcd debug\_temp/test.fst (no bug is detected)

** **

------------------------------------------------------------------------------------------------

**RISCV Reference**

------------------------------------------------------------------------------------------------

* RISC-V formal checking ref:
  + <https://github.com/SymbioticEDA/riscv-formal>
* ALU pipeline 5 stage to figure out some assertions:
  + <https://en.wikipedia.org/wiki/Classic_RISC_pipeline>
  + (\*) <https://blog.csdn.net/whenloce/article/details/87071341>
* Stage 3 (Execution) → forwarding unit
  + <https://reurl.cc/Er2bl1>

**2022.04.19**

------------------------------------------------------------------------------------------------

**Assertion 實驗目的**

------------------------------------------------------------------------------------------------

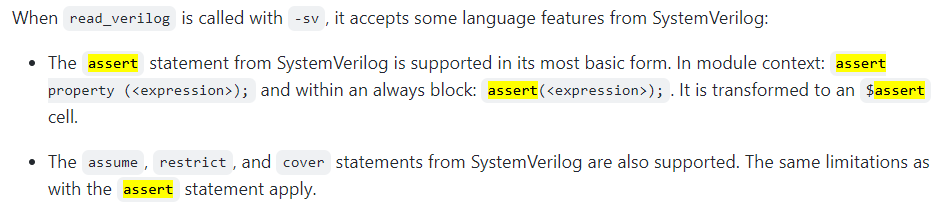
* 熟悉 assertion 語法如何在 yosys 中運作
* 針對我們的 CPU (golden) design, 寫一些必須的 assertion 去確定理論上沒有 bug (雖然跑過 tb 是沒有問題的)
* 針對我們的 CPU (insert bug) design, 看會不會被基本的 assertion 打到, 如果會, 那 bug 安插得太簡單了, 改掉 ; 如果不會, 那可以試著寫更深的 assertion, 看怎麼樣可以打到我們的 bug
  + 當然, 若是有 assertion 打到這個 bug, 除了看合不合理以外, 也要加入到 CPU (golden) design 的 assertion, 如果

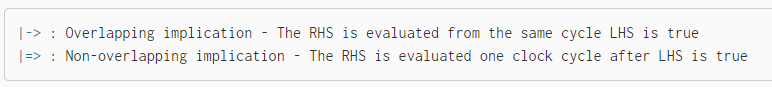
------------------------------------------------------------------------------------------------

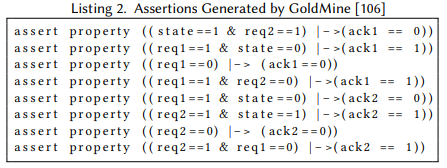
**yoysy 支援的 assertion format**

------------------------------------------------------------------------------------------------

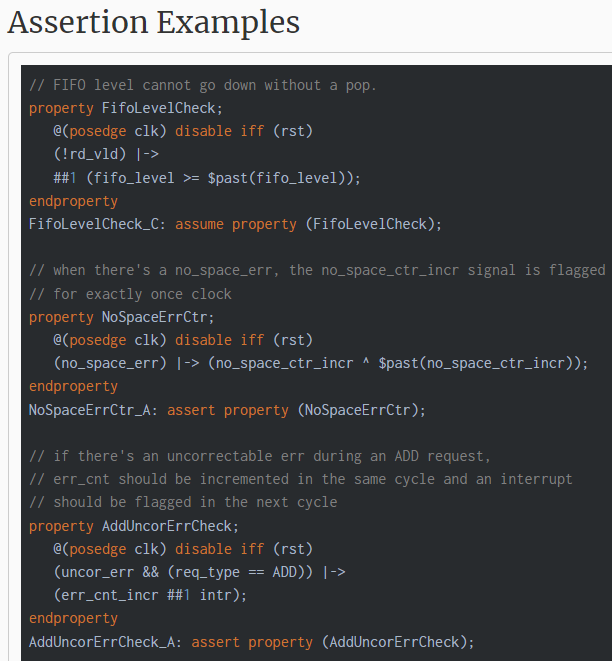
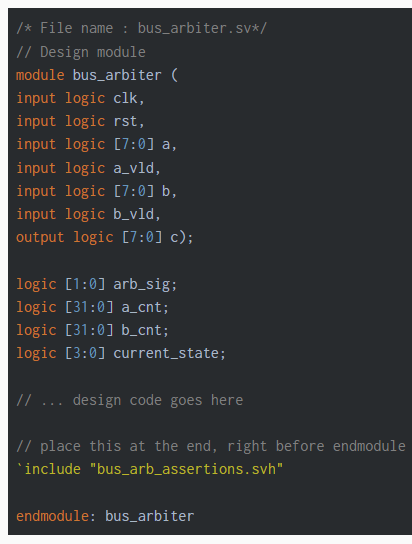
* yosys 支援的 SVA (SystemVerilog Assertion) 為"最基本的 format"
  + 在 module block 裡面, 沒有 $display, 單純 “assert property (event);”
  + 在 always block 裡面, 用 “assert (event);”
  + **(o) e.g.**







* 應該無法用額外新開的檔案 assertion.svh 寫入所有 assertion property, 然後再回到 module block 裡面直接 include 這個檔案 → **有空試試看**
  + tutorial: <https://www.systemverilog.io/sva-basics>
  + **(?)**

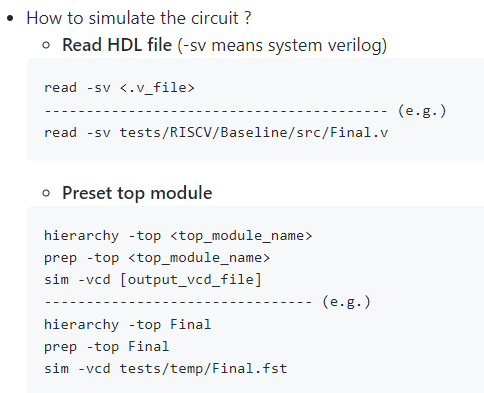
 

------------------------------------------------------------------------------------------------

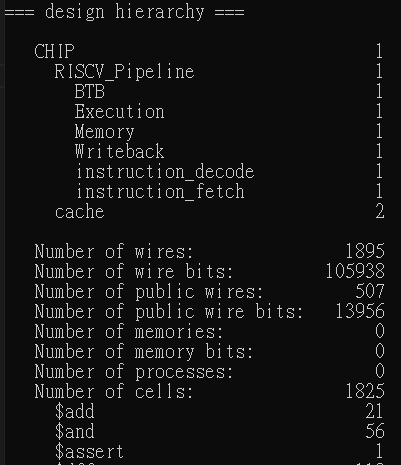
**Exp1: assert property 可以 work**

------------------------------------------------------------------------------------------------

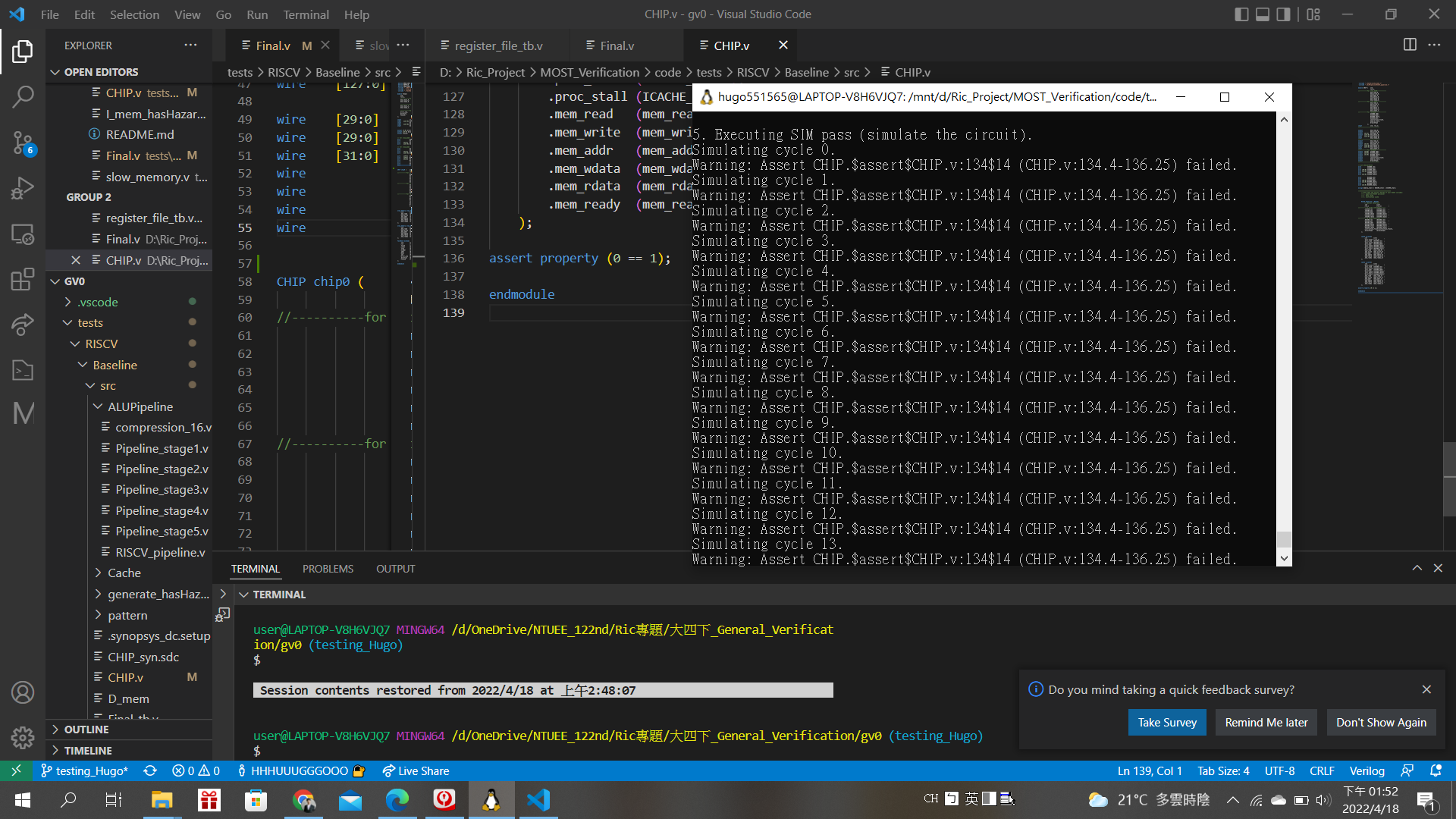
* 在 CHIP.v 的 CHIP module 中, assert 一個恆 false 的 property → (0 == 1), 然後執行下面步驟



* 在 design hierarchy 可以看到 assert 會被 yosys parsing 出來



* 當 sim 完後會發現每個 cycle 都能抓到這個 assertion 的 bug, 所以代表可以 work

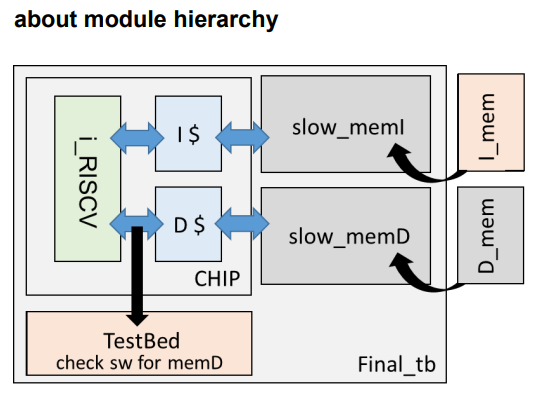


------------------------------------------------------------------------------------------------

**Exp2: 在必要 module 塞 basic assertion (w/o bug version)**

→ 確保在一般情況下不會有問題 (shallow assertion)

------------------------------------------------------------------------------------------------



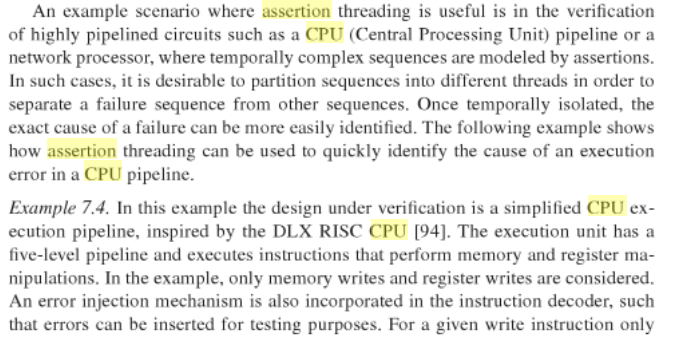
* **Final.v (Final\_tb.v 簡化版) → Final**
  + **CHIP.v → CHIP** **(file → module 架構)**
    - cache\_2way.v → cache (for I$ and D$)
    - RISCV\_pipeline.v → RISCV\_Pipeline
      * 2way\_BTB.v → BTB
      * stage1.v → instruction\_fetch
      * stage2.v → instruction\_decode
      * stage3.v → Execution
      * stage4.v → Memory
      * stage5.v → Writeback
      * 5 stage detail ref: <https://reurl.cc/anlZo3>
  + **TestBed\_hasHazard.v → TestBed**
  + **slow\_memory.v → slow\_memory**

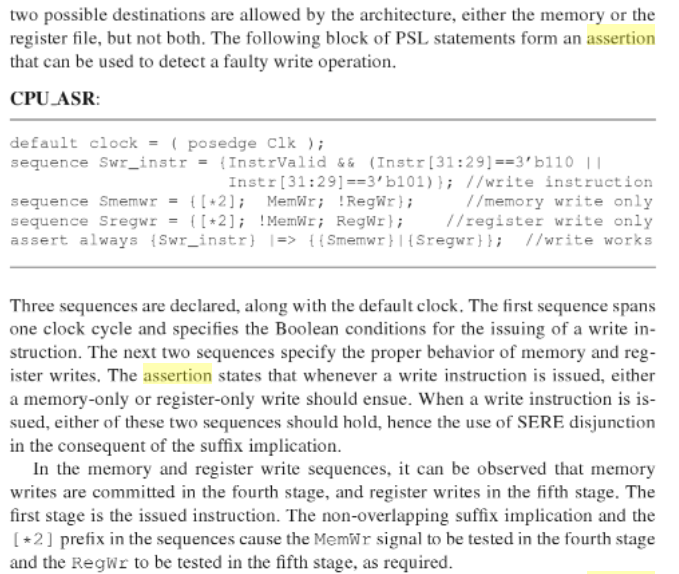
------------------------------------------------------------------------------------------------

**Exp3: 針對 write 寫 assertion (w/o vs. w/ bug version)**

------------------------------------------------------------------------------------------------

* ref: <https://reurl.cc/VDazYQ>





([\*2] → 連續兩個 clk 都成立 ? or sequence 內要測兩個 signal)