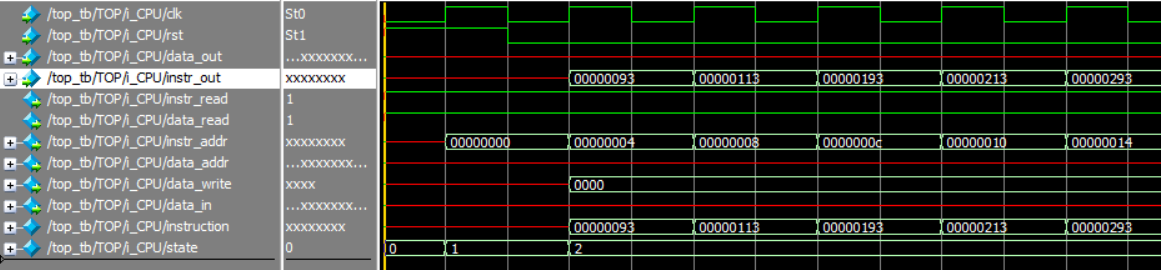
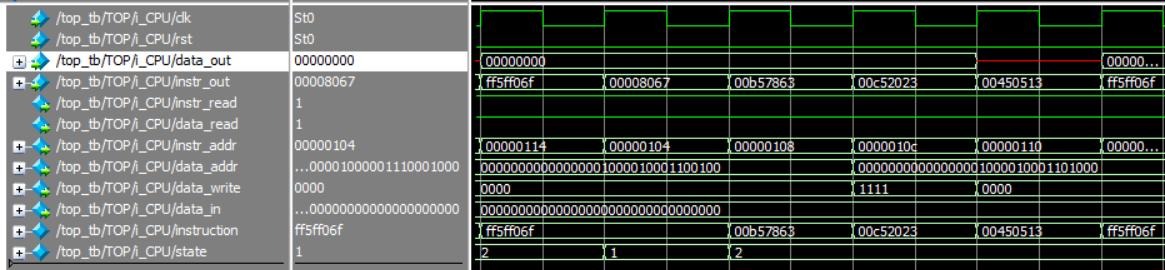
**Computer Organization 2019**

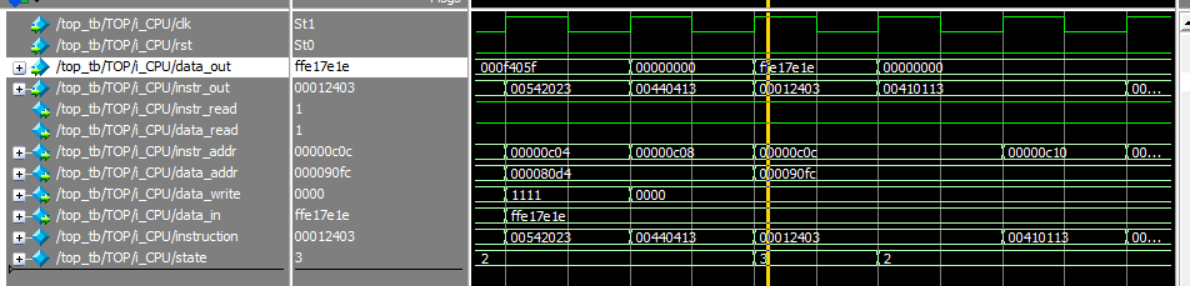
**HOMEWORK 4**

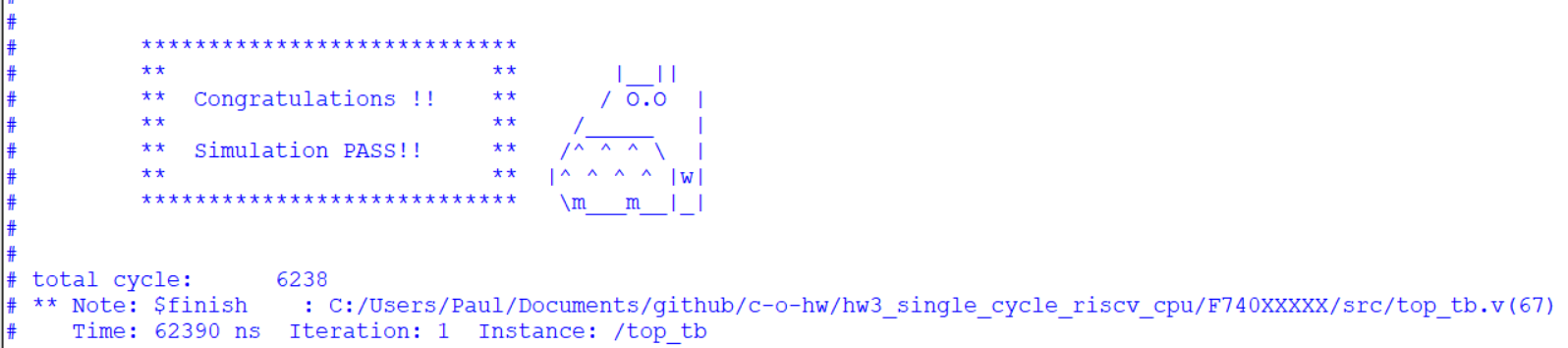
系級: 資訊 112 學號: F74081129 姓名: 吳信葆

**實驗結果圖:**







****

**程式運作流程:**

我發現這個程式，所有 module 都是 clk 正緣觸發，所以我 CPU 用到狀態機的概念，結合流水線。

狀態零，讀取對應 PC 指令，到狀態一。

狀態一，讀取的(instr\_out)指令放到工作區(instruction)執行，到狀態二。

狀態二，兩個 case

Case 1: 就是 狀態一加更新 register、pc，維持狀態。

Case 2: 如果發現要 branch 或 jump，pc 會更新到該位址，但執行狀態 零做的事，到狀態二。

狀態三，回狀態二，該狀態透過 opcode==load 發動 stage<=3，下次正緣觸 發。

指令 load、jump、branch 花了兩個 cycle，其他都只有一個。

我對每個 type 的指令都設計 alu 給它。

第一張波型圖代表一開始，可以看到我在第一個clk正緣就下我要第 0 行instr 的訊號，但是這是在本次正緣結束才更新，所以第二次正緣才收到這個訊號，給我第0行 instr。於是從 state=2 開始，我就要下一行指令，這樣我每條命令只要一個 cycle。

第二張波型圖代表，下一行程式碼不是我們要執行的，所以回狀態一，讀本行(jump 後程式碼)，不更新 register。

第三張波形圖代表我要 lw 但是資料會再clk正緣更新完 register 才剛好來，所以我必須給它 state=3 做緩衝(不做任何事等資料)。

**心得**

這個作業我打超久，花很多時間才發現所有 module都是正緣觸發，然後進度飛速提升，但中間有花一些時間用 display debug，再 store 實印出 pc 和存的值能夠很快從 main.log 找到錯在哪，大部分都是 sign 和 unsign 有錯。