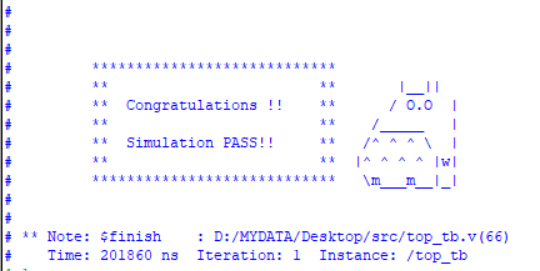
**Computer Organization 2019**

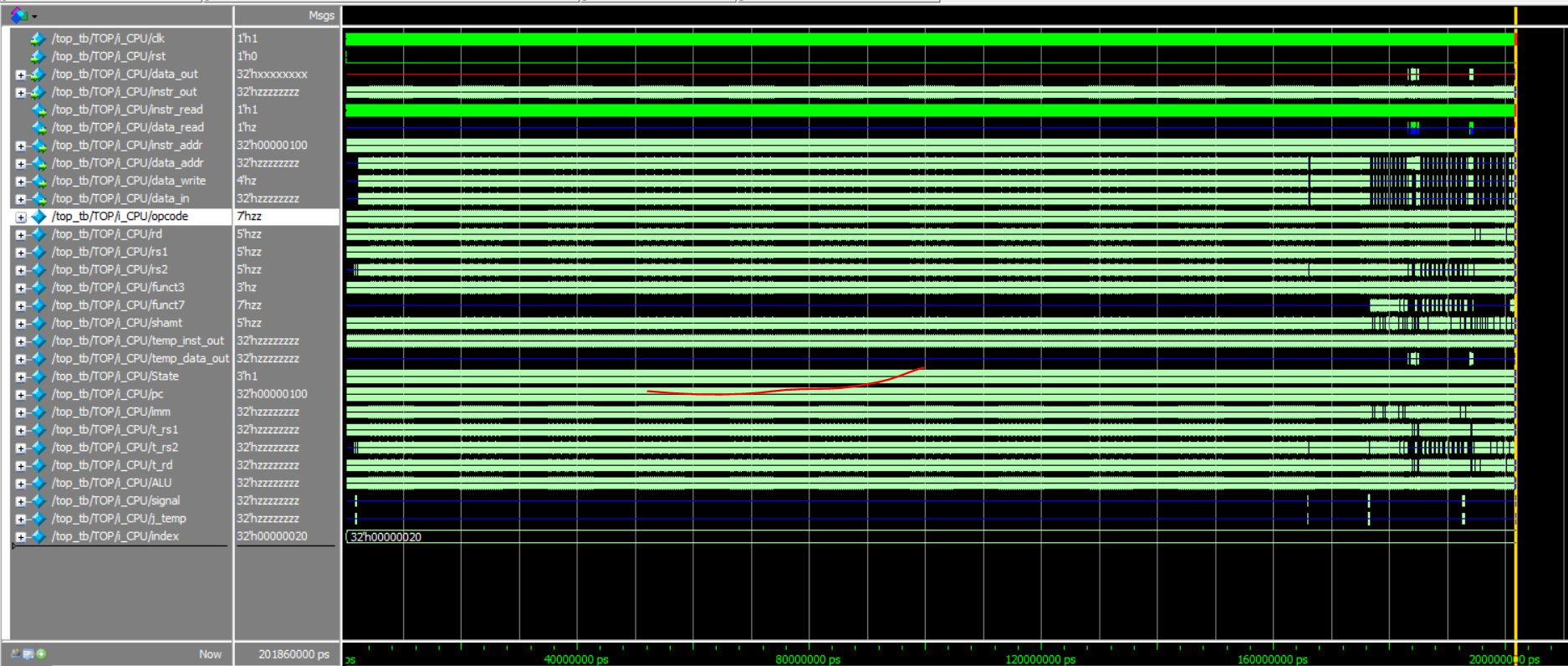
**HOMEWORK 4**

系級: 資訊系 學號: F74072099 姓名: 黃濬程

**實驗結果圖:**

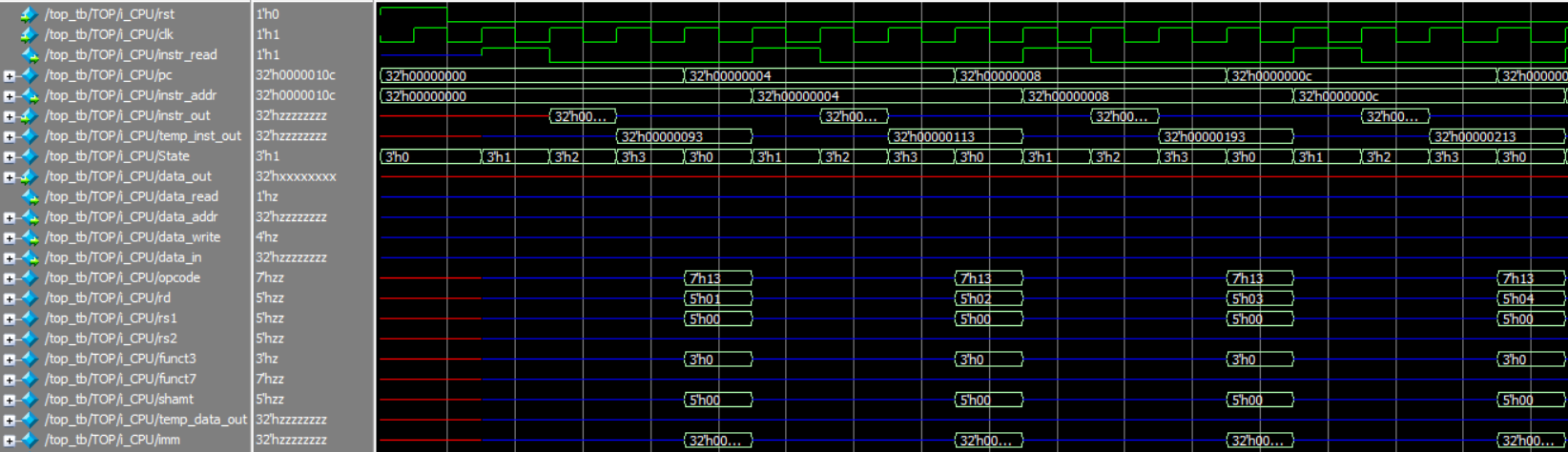
(波形圖及模擬完成截圖)





**程式運作流程:**

(簡單說明波形變化的意義)



透過上面完整的波形圖很難去分析wave中變化的意義。故我截了一開始rst後的一小部分，以說明此波形與程式間的關係。

此次作業我使用Sequential的寫法，將整個程式分成7個State如下圖

(圖(1))；

前4個State分別負責:

READ\_INS: 向IM 要資料，instr\_read=1;。(State=0)

GET\_INS: 在這個State，instr\_out已經有了值，但因為還未透過正緣傳入CPU，故此State只是將instr\_read設為0，以便確保instr\_out不會亂改變。 (State=1)

SAVE\_INST: 此State其實已經可以處理instr\_out(因為已經傳入cpu)，但我選擇不在這個State處理，因為在之後的state由於instr\_read已為0，我選擇先將instr\_out用temp\_inst\_out存起來；利用temp\_inst\_out，我能夠確保在處理這串指令時能夠在每個state都access到當前要處理的instruction，且能夠在不同state分割出我需要的片段，存入其他reg。(State=2)

DECODE: 在這個state我便開始處理所有instruction 需要的運算以及DM的處理；而像是LW這種需要用到DM內資料的指令，我便會在此階段將 Data\_addr跟Data\_read=1傳入DM，去作類似READ\_INS這個階段對IM做的事，也就是”申請資料”。

而剩下的State: GET\_DATA、SAVE\_DATA、LOAD\_DATA與前面對IM所做的事

類似；由於在DECODE時我已經提前做了READ\_DATA(申請資料)，我在GET\_DATA

就是將Data\_read設為0，與GET\_INS類似；SAVE\_DATA也是同理，利用temp

\_data\_out來存取此次的data\_out；最後的LOAD\_DATA即是將DATA寫回

REGISTER

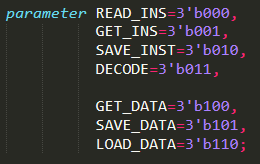
而我想特別說明上面波形圖中State值所代表的意義；在波形圖中，State=1

的時候其實是在State=0的狀況內；因為跑入了State0的case State的值才

會被改變成1；簡而言之，在wave中State所顯示的值代表的並不是當前的

State，而是下個要進入的State!

舉上圖中wave李第一次State=3的時候，可以看到State=3理應為DECODE，也就是處理資料；但其對應的rd rs1 opcode卻都沒跑出來；反而在下一個State=0的時候rd rs1 opcode才被賦值；而理由便如上述，在這個例子中，State=0時才意味著當前的State是”DECODE”。

(圖(1))；

**心得**

本次作業實在很難，但在充分了解題目以及詢問助教關於初始化失敗的問題後我才成功完成了這次作業。(未完成初始化前實在不知道如何完成此作業，因為modelism也無法看見當前register的值；且其實對於main.log和setup.s那些數字的意義在一開始是完全不能理解的，故無法確定程式執行的順序是否如main.log所示；不過後來認識以後，DEBUG上才有了依靠)

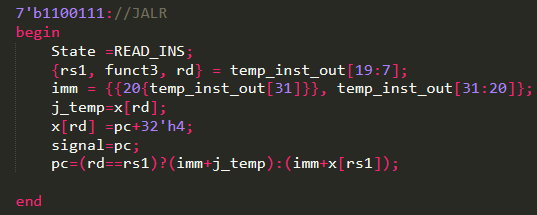
困難點其實很多，關於verilog以及modelism是我們比較少接觸的；光是blocking和non-blocking在還沒做作業前也是完全不認識的。

原本只知道說整個程式不能混用blocking和nonblocking，卻不知道其中的差異；而此次作業我使用的是blocking的策略，所以常常在賦值上會因為先改到pc的值再將pc assign給 register，這樣再JALR的指令便會有了問題；

且JALR 的rs1 跟 rd 是同一個register時也會因為rd的修改有了問題

。於是我利用reg j\_temp來儲存rd原本的值，以便解決blocking帶來的問題(如下圖)。

而這個問題是這次debug中最常遇到的問題，也很感謝林家葦助教讓我認識這樣的差異；再來是對於data\_write和imm對齊的事也非常細節，對於byte的操縱要有一定的認識才能夠完成。



不過這次作業確實讓我受益良多，對於machine language也有了更深一層的認識；看到101010101的時候比較不會排斥了；且對於single clock ， sequential 有了更深的理解。