CA Final Report

**CPU Architecture:**

CPU的架構基本上參考Final project slide上面的架構，有PC\_CAL、Control、ALU、ALU\_Control、Imm\_Gen、RegWrite\_mux還有multDiv這幾個架構。以下分各個架構介紹。此部分僅介紹基本功能，關於新加入的instructino例如jal, jalr等，請參考後面的部分有詳細說明。

1. PC\_CAL: 主要input是PC，output是PC\_nxt，是為了計算下一個clk的PC是多少。須由一個mux去控制，有jump就是jump到PC+immediate，沒有jump就是+4。
2. Control: 基本上就是圖片上的這7個signal，接線方法就不詳述。其中，我們將MemtoReg改為RegWrite\_src[1:0]，因為之後加上jal, jalr等instruction之後，寫進rd的data source不只兩種，所以需兩個bit控制。最後會拿RegWrite\_src去控制RegWrite\_mux。
3. ALU: 根據ALU control signal，來做對應的ALU operation。我們的CPU有支援+、-、&、|這4個opeartion。另外，為了要支援SLTI，必須另外加上compare的operation。
4. ALU\_Control: 利用control的ALUOp signal，還有func3, opcode以及func7的field判斷ALU會用到哪一種operation。要注意ADD和SUB的區別會用到func7，但是ADDI沒有func7 field，所以要先判斷是否是ADDI，再區別ADD和SUB。最後輸出一個4bit的control signal，用來控制ALU的運作。
5. Imm\_Gen: 根據不同instruction，去拿取對應的immediate field，生成相對應的immediate output。要注意此處output也是32bit而非64bit。
6. RegWrite\_mux: 決定要write到rd的data source要從哪裡拿取的mux，原先的CPU只會有兩個source(memory和alu\_out)，但後面支援其他instruction之後，就不只一個source，因此需要額外新增一個module來處理。

**Datapath of jal, jalr, auipc:**

因為三種情況都會有不同的做法，所以需要jal, jalr和auipc三個額外的control signal。以下分三種instruction的情況討論。

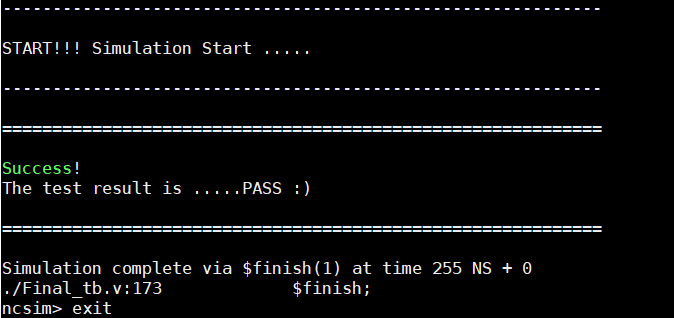
1. AUIPC: 在auipc當中，PC\_nxt = PC +4，但又需要將PC+immediate存到register，所以前者利用PC\_adder，後者利用ALU完成，所以alu\_in\_1 = PC，因此alu\_in\_1前面須加上mux，即alu\_in\_1 = auipc ? PC : rs1\_data。另外auipc的immediate比較不同，所以imm\_gen要另外判斷如果是auipc，imm\_gen = I\_in & {20'b1, 12'b0}。另外因為auipc是做加法，所以ALU control要判斷輸出相對應的control，且最後寫回regiser時是從ALU拿。(即regWrite\_mux要判斷)
2. JAL: 在jal當中，PC也要jump，所以PC會jump的條件變成 (branch & zero ) | jal。和auipc不同，jal不需要用到ALU，所以ALU相關的control和output都不須理會；只需要注意最後write register的時候，write的data是PC+4，所以regWrite\_mux要額外拿PC的數值，+4之後寫到rd。(不要不小心拿到ALU或memory的值)
3. JALR: 最後是jalr，它做的事情是要先把要jump的address從memory當中load出來，所以ALU做的是load的事情，只是這裡load的數值要寫到PC，而寫到rd的數值則是PC+4，所以在PC的mux要加上一個jalr的情況，是從mem\_rdata\_D去拿data；而regWrite\_mux和jal一樣，是存PC+4。

**How to handle multi-cycle operation - mul:**

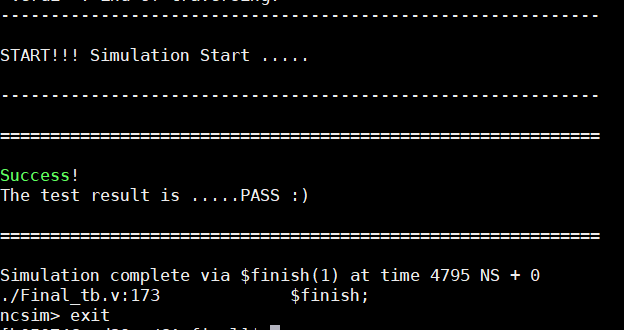
我們直接使用HW3的multDiv module，沒有改動內部的module(因此乘法器內部怎麼運作就不再贅述)。外面control的部分，我們為了讓整個CPU知道現在是在做mul，所以control signal多了一個mul。為了達到multi-cycle的功能，只要是在做mul的時候(mul=1’b1)，PC就不加4(PC\_nxt = PC)，這樣才不會fetch到下一個instruction。直到mul的ready拉起，代表mul做完之後，PC才會+4，fecth到下一個instruction。最後在要寫回register的時候，要額外判斷如果是mul，write的data要從multDiv的output拿。

**Total simulation time:**

* Leaf:



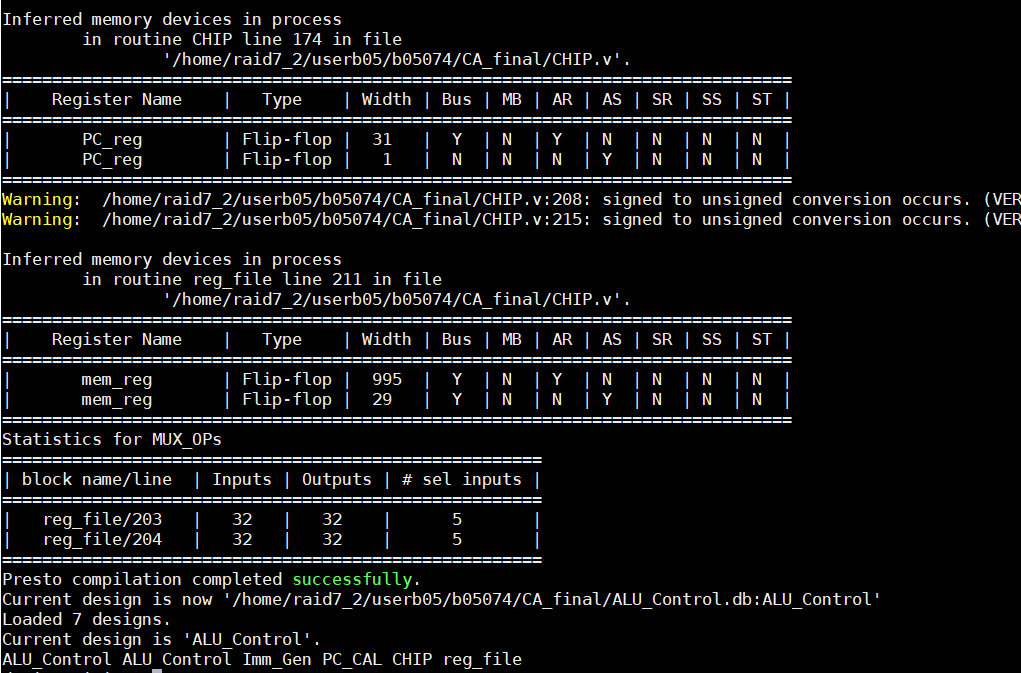
* Fact:



**Observation:**

// my observation

**Register Table:**



Work Distribution Table: