CA Final Report

B05901074 陳泓均

B05901158 林緯瑋

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 instruction 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 operation。另外，為了要支援 SLTI、SRAI 與 SLLI，必須另外加上 compare、logical left shift 和 arithmetic right shift 等 operation。
4. ALU_Control: 利用 control 的 ALUOp signal，還有 funct3, opcode 以及 funct7 的 field 判斷 ALU 會用到哪一種 operation。要注意 ADD 和 SUB 的區別會用到 funct7，但是 ADDI 沒有 funct7 field，所以要先判斷是否是 ADDI，再區別 ADD 和 SUB。最後輸出一個 4-bit 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 32 bit 而非 64 bit。
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。

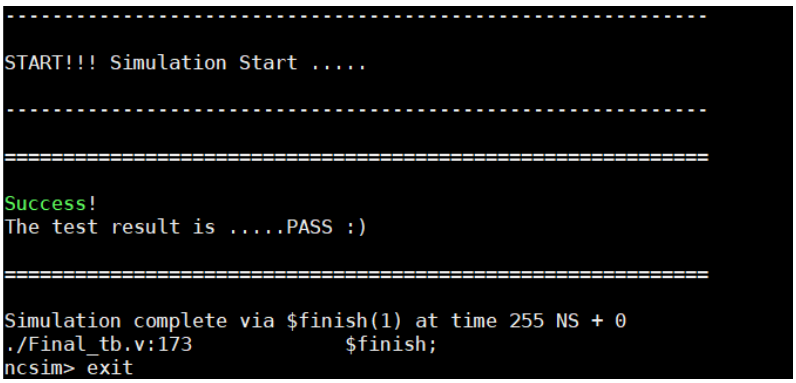
1. AUIPC: 在 auipc 當中，PC_nxt = PC +4，但又需要將 PC+immediate 存到 register，所以前者利用 PC_adder，後者利用 ALU 完成，所以 alu_in_1 = PC，因此 alu_in_1 前面須加上 mux，即 alu_in_1 = auipc ? PC : rs1_data。另外 auipc 的 immediate 比較不同，所以 imm_gen 要另外判斷如果是 auipc，imm_gen = I_in & {20'b1, 12'b0}。另外因為 auipc 是做加法，所以 ALU control 要判斷輸出相對應的 control，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:

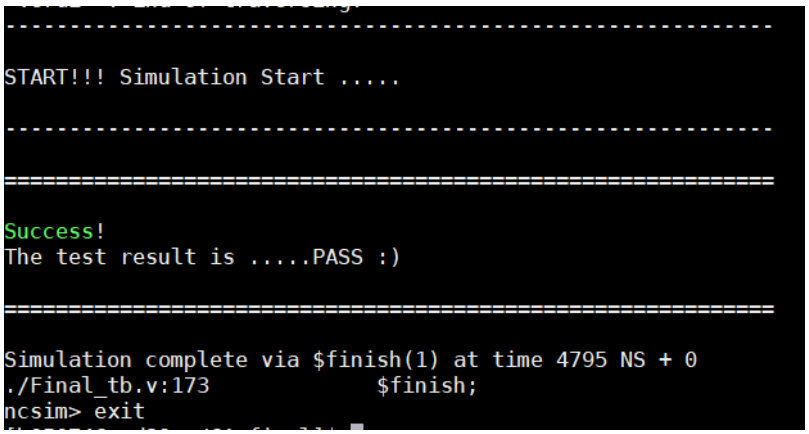
我們直接使用 HW3 的 multDiv module，沒有改動內部的 module(因此乘法器內部怎麼運作就不再贅述)。外面 control 的部分，我們為了讓整個 CPU 知道現在是在做 mul，所以 control signal 多了一個 mul。為了達到 multi-cycle 的功能，只要是在做 mul 的時候 (mul=1'b1)，PC 就不加 4 (PC_nxt = PC)，這樣才不會 fetch 到下一個 instruction。直到 mul 的 ready 拉起，代表 mul 做完之後，PC 才會 +4，fetch 到下一個 instruction。最後在要寫回 register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

Total simulation time:

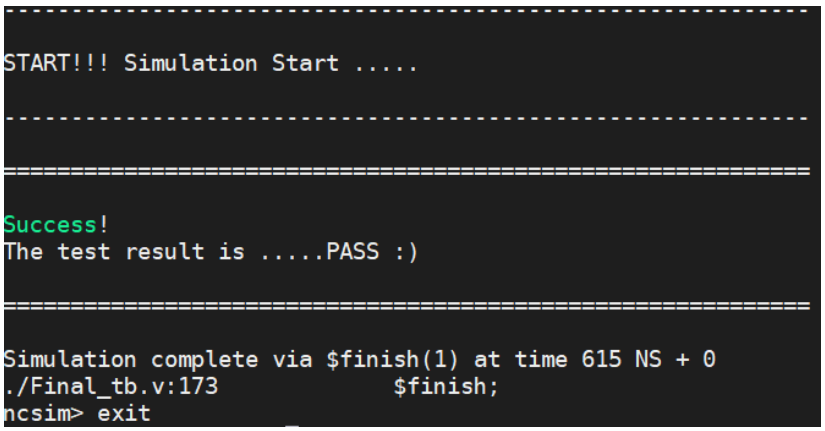
- Leaf:



- Fact:



- HW1:



Observation:

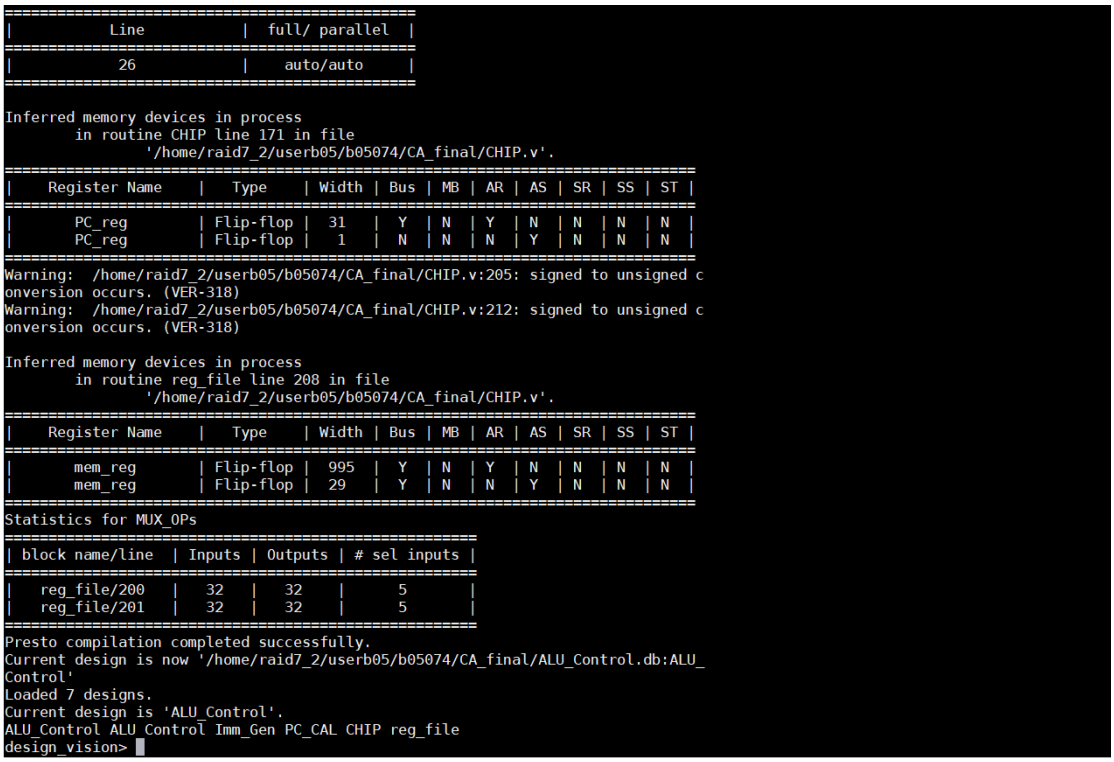
Non-pipelined 的 CPU 實作起來相對簡單，因為幾乎使用的都是 combinational circuit，且 memory 部分都由助教完成，因此實作上沒有太大的困難；只是需要額外處理 multDiv 的 multi-cycle 問題。原本以為必須像作業三一樣寫一個 FSM、定義兩種 state、加上 state 以及 state_nxt 等參數，後來發現只需一個 mul signal 紀錄 state 即可。

一開始遇到的困難是在 debug 的時候，因為只要有一個部分有錯，例如 PC，instruction 的順序就會亂掉，所以要一個一個去 trace 回去找到源頭；只要一個指令有錯，整個程式執行的情形就會完全亂掉，所以 debug 上不是非常容易。後來就發現，debug 的時候一定要看著要執行的 assembly，一行一行去看有沒有執行正確，找出開始錯誤的地方下去修正。

實作了 processor 後體會到 risc-v 指令集的分類邏輯非常清楚好懂，依照不同類型的指令分為大類別再往下細分，也盡量將相同的東西放在 instruction 中相同的位置，因此 parse instruction 方面並沒有很困難。像是 SRAI、SLLI 等就是在 I-type 下的 special case，因此在處理上不需要額外判斷，而是按照 I-type 的方式判斷、計算 immediate，最後在 ALU 中取出 immediate 需要的部分實現功能即可。

另外在實作 auipc, jalr 等指令的時候發現，如果設計得宜，其實可以有效節省硬體，例如 auipc 就不用再多加一個 adder，直接用 ALU 的 adder 就可以(因為 PC 的 adder 是+4，和運算需要用到的不同)，只是需要另外用 control 訊號處理接線的問題。

Register Table:



```
=========================================
|          Line          |  full/ parallel  |
=========================================
|           26           |    auto/auto     |
=========================================

Inferred memory devices in process
        in routine CHIP line 171 in file
                '/home/raid7_2/userb05/b05074/CA_final/CHIP.v'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|       PC_reg        | Flip-flop |  31   |  Y  | N  | Y  | N  | N  | N  | N  |
|       PC_reg        | Flip-flop |   1   |  N  | N  | N  | Y  | N  | N  | N  |
===============================================================================
Warning:  /home/raid7_2/userb05/b05074/CA_final/CHIP.v:205: signed to unsigned c
onversion occurs. (VER-318)
Warning:  /home/raid7_2/userb05/b05074/CA_final/CHIP.v:212: signed to unsigned c
onversion occurs. (VER-318)

Inferred memory devices in process
        in routine reg_file line 208 in file
                '/home/raid7_2/userb05/b05074/CA_final/CHIP.v'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|       mem_reg       | Flip-flop |  995  |  Y  | N  | Y  | N  | N  | N  | N  |
|       mem_reg       | Flip-flop |  29   |  Y  | N  | N  | Y  | N  | N  | N  |
===============================================================================
Statistics for MUX_OPs
=====================================================
| block name/line  | Inputs | Outputs | # sel inputs |
=====================================================
|   reg_file/200   |   32   |   32    |      5       |
|   reg_file/201   |   32   |   32    |      5       |
=====================================================
Presto compilation completed successfully.
Current design is now '/home/raid7_2/userb05/b05074/CA_final/ALU_Control.db:ALU_
Control'
Loaded 7 designs.
Current design is 'ALU_Control'.
ALU_Control ALU_Control Imm_Gen PC_CAL CHIP reg_file
design_vision>
```

Work Distribution Table:

	Work
陳泓均	架構：PC_CAL、Control、Imm_Gen、RegWrite_mux、multDiv 額外指令：auipc、jal、jalr、mul、slti
林緯瑋	架構：ALU、ALU_Control 額外指令：slli、srai