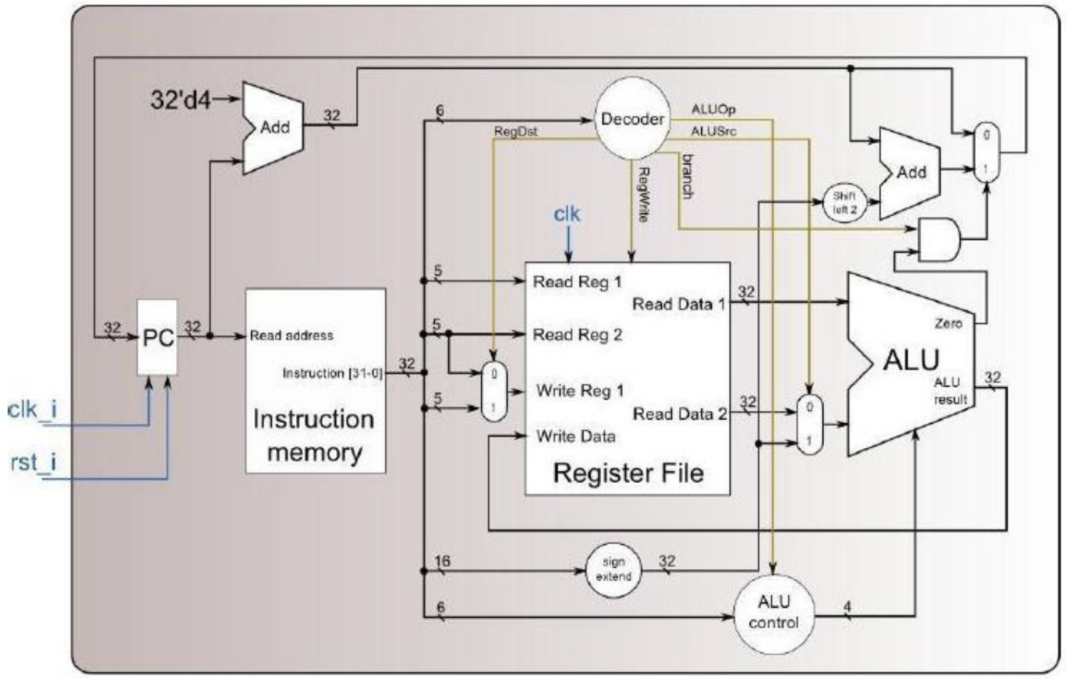
Computer Organization Lab2

Name: 莊婕妤　ID: 109550182

Architecture Diagrams



Hardware Module Analysis

- **Program Counter**：指向程式要執行的指令位置
  經過一個 cycle 後，PC 會自動加 4，指向下個指令。但若該 instruction 為 beq 也需要 jump to branch 的話，需要加上 offset

- **2-1 Multiplexer**：依據條件 0 或 1，輸出相對應的值，此設計有用到 3 個 MUX
  Mux_Write_Reg 用來決定寫入的 register 為 rt 或 rd，I-Type 寫入 rt，R-Type 寫入 rd
  Mux_ALUSrc 用來決定輸入 ALU 的 source 2 為 constant 還是 rt
  Mux_PC_Source 用來決定下一個 cycle 的 PC 為加 4 的還是加 offset 的

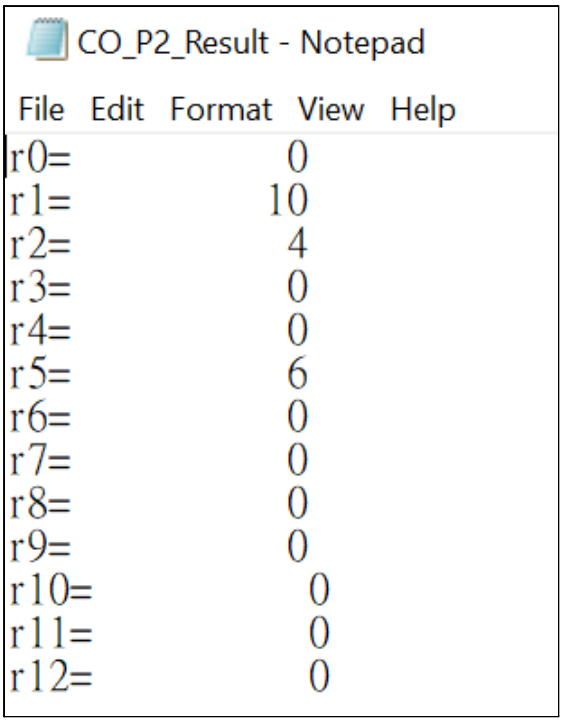
- **Adder**：把兩個 32-bit 的 source 加起來，Adder1 用於將 PC 加 4（Sequential），Adder2 用於將 PC 加上處理過後的 offset（Branch）

- **Sign Extend**：把不足 32-bit 的值，用 sign bit 將其 extend成 32 bit，用來 extend 只有 16 bit 的 offset（beq instruction）或 constant（immediate instruction）

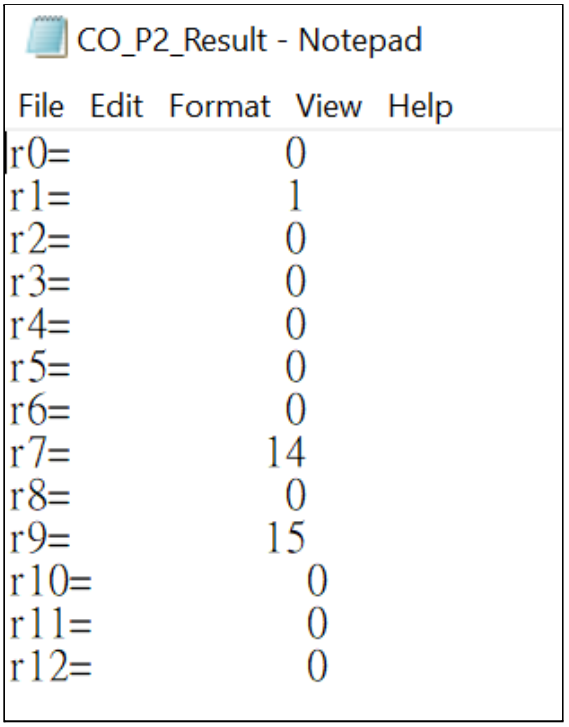
- **Shift Left Two**：將輸入的值向左平移兩位，用於將 beq 會用到的 offset 乘以 4
- **Decoder**：根據輸入的 Op Code 輸出對應的 contro，分別輸出 RegDst 來控制 Mux_Write_Reg、RegWrite 來決定 **Register File** 是否做寫入的動作、branch 來看此指令是否為 beq、ALUSrc 來控制 Mux_ALUSrc、ALUOp 來控制 **ALU Control**
- **ALU Control**：根據 **Decoder** 傳出的 ALU_Op 跟 instruction 中的 function field（for R-Type），來決定 **ALU** 要執行哪個指令（跟著 spec 給的 instruction set 來設定）
- **ALU**：將兩個輸入的值根據 **ALU Control** 輸出的 operation code 做運算
- **Instruction Memory**：輸入 PC，輸出該位置的 instruction
- **Register File**：儲存 register，做值的讀入以及寫出
- **Simple Single CPU**：把所有的 module 接在一起，形成一個簡易的 Simple Single CPU

Finished Part

Test Data 1



Test Data 2



Problems You Met and Solutions

在實作的過程中，我遇到的問題主要是不太熟悉 Decoder 以及 ALU Control 中 immediate operations 的 control code 要如何設定，畢竟老師在課堂中教的沒有 addi、slti 等 operations。後來，我才找到 spec 上的 Instruction Set 中有明確的指示出他們對應到的 Op Field 才解決了我的問題。

P.S. 但不確定我有沒有理解錯誤 Op Field 的用法，因為 ALU_Op 只能用三個 bit 去表示，但 Immediate Instructions 的 Op Field 卻需要 4 個 bit 才足以表示，但我觀察出最後一個 bit 皆為 0，因此將最後一個 bit 去掉，並用其他三個 bit 去做對應。

Summary

這次的 Lab 讓我更了解老師在課堂上教授的那些電路圖，透過各個 Module 的設計，再整個接起來合成一個 Simple Single CPU，幫助我更熟悉他們背後運作的原理，像是如何控制電路讓其可以執行許多不一樣的 Instructions，同時我也更加熟悉 verilog 的語法了。