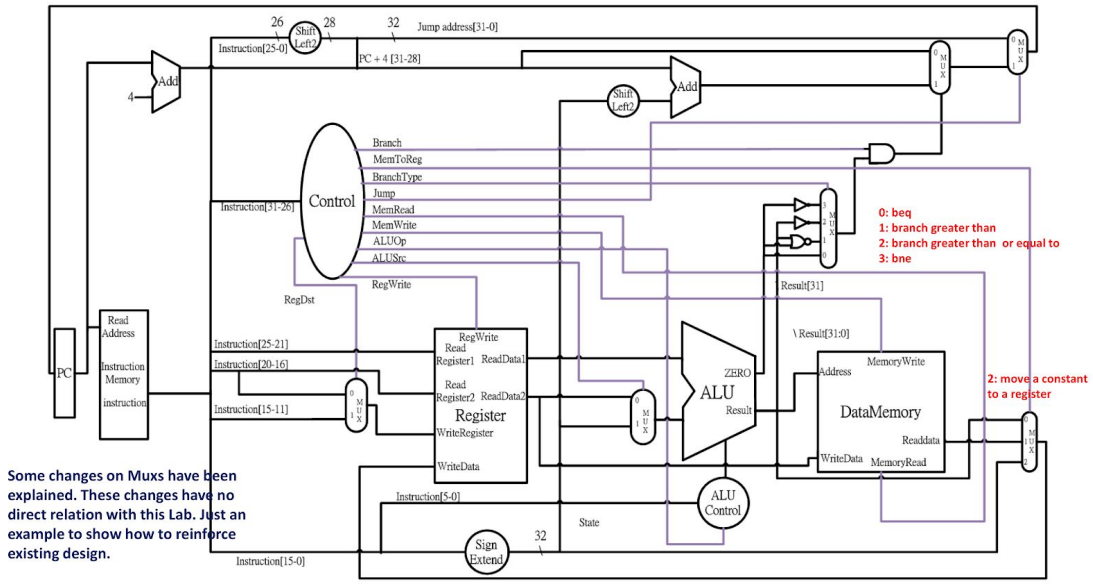
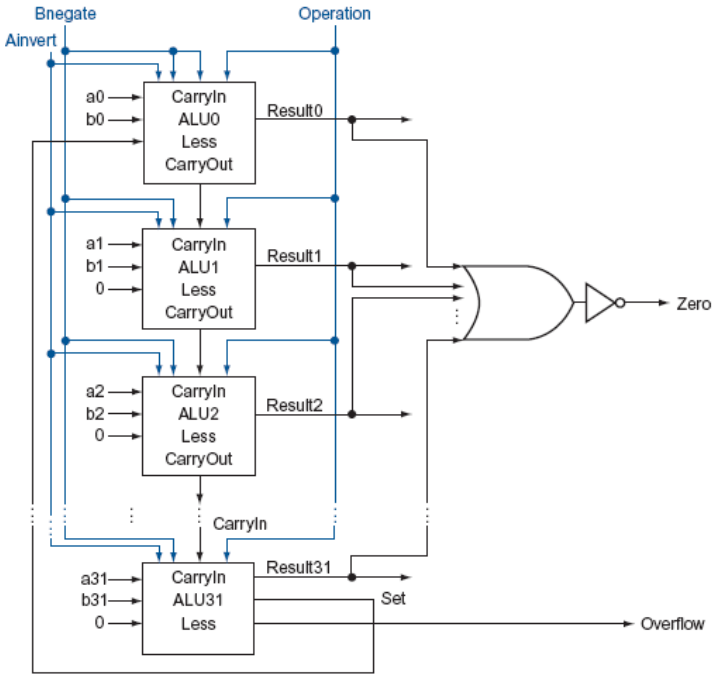
Computer Organization

Architecture diagrams:

Simple_Single_CPU.v



ALU.v



Hardware module analysis:

硬體分析

- module Simple_Single_CPU (clk_i, rst_i)
  - module ProgramCounter (clk_i, rst_i, pc_in_i, pc_out_o)
  - module Adder (src1_i, src2_i, sum_o)
    有兩個，一個是 PC 的 Adder (PC <- PC + 4)，另一個是算要 Branch 到哪裡 (位址) 的 Adder
  - module Instr_Memory (pc_addr_i, instr_o)
  - module MUX_2to1 (data0_i, data1_i, select_i, data_o)
    有三個，一個是選擇指令的哪個部分作為 Write Register，另一個是選擇 ALU 的 Source (從 Register 的值還是從 Instruction 的值)，最後一個是決定新的 PC 是否要採用 jump 的那個
  - module Reg_File RF (clk_i, rst_i , RSaddr_i, RTaddr_i, RDaddr_i, RDdata_i, RegWrite_i, RSdata_o, RTdata_o)
  - module Decoder (instr_op_i, RegWrite_o, ALU_op_o, ALUSrc_o, RegDst_o, Branch_o, Jump_o, MemRead_o, MemWrite_o, MemtoReg_o)
    根據指令的 OP Field (高 6 bit) 決定要送往各個單元的控制訊號，相較於上次多了控制 jump、記憶體讀寫、以及 Memory 是否要寫回 Register 的訊號
  - module ALU_Ctrl (funct_i, ALUOp_i, ALUCtrl_o)
    根據 Decoder 送來的控制訊號 (aluop) 以及 Instruction 的 Function Field (如有需要的話) 來決定 ALU 實際要執行的動作
  - module Sign_Extend (data_i, data_o)
    把十六位元的數 extend 成三十二位元，並保持正負號不變
  - module ALU (src1_i, src2_i, ctrl_i, result_o, zero_o)
    用 Behavioral 的方式實現，由 aluctrl 控制訊號來決定要執行什麼動作
  - module Shift_Left_Two_32 (data_i, data_o)
    Shift Left 兩次等同於乘以 4，因為一個 word 是四個 byte

訊號分析

- Simple_Single_CPU.v：Single Cycle CPU 的主要架構
  - clk_i：clock 訊號
  - rst_i：reset 訊號
- ProgramCounter.v：每過一個 clock 更新一次 PC
  - clk_i：clock 訊號
  - rst_i：reset 訊號

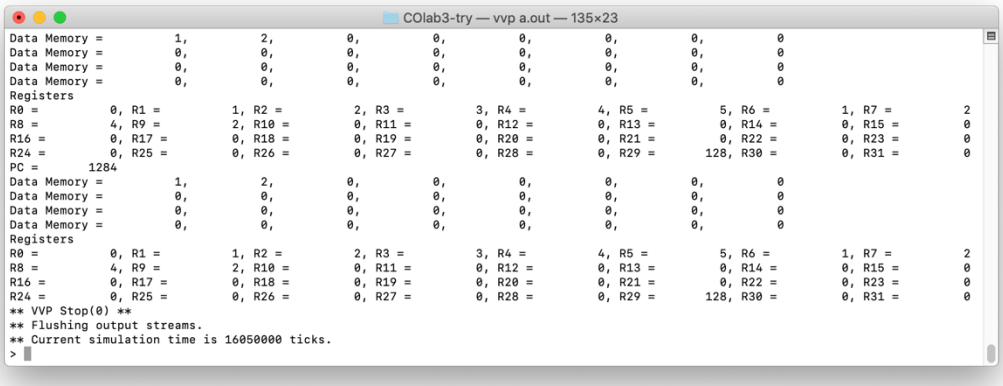
- pc_in_i：新計算出來的 PC
- pc_out_o：每過一個 clock 給 Instruction Memory 一個新的 PC

- Adder.v：把兩個 source 訊號相加後輸出
  - src1_i：32-bit 輸入
  - src2_i：32-bit 輸入
  - sum_o：相加後的 32-bit 輸出
- Instr_Memory.v：由 PC 指向的位址去 Fetch Instruction
  - pc_addr_i：輸入 PC 指向的位址
  - instr_o：輸出 32-bit Instruction
- MUX_2to1.v：由控制訊號決定要輸出哪一個輸入訊號
  - data0_i：1-bit 輸入
  - data1_i：1-bit 輸入
  - select_i：控制訊號
  - data_o：1-bit 輸出
- Reg_File.v：
  - clk_i：clock 訊號
  - rst_i：reset 訊號
  - RSaddr_i：選擇 RS 暫存器
  - RTaddr_i：選擇 RT 暫存器
  - RDaddr_i：選擇 RD 暫存器
  - RDdata_i：要寫入 RD 暫存器的資料
  - RegWrite_i：控制是否要寫回暫存器
  - RSdata_o：讀取到的 RS 暫存器的值
  - RTdata_o：讀取到的 RT 暫存器的值
- Decoder.v：由 Instruction 來決定要發送往各個單元的控制訊號
  - instr_op_i：Instruction 的高 6 位元
  - RegWrite_o：Register File 是否要寫回
  - ALU_op_o：送往 ALU Control 的控制訊號
  - ALUSrc_o：控制 ALU 的第二個 Source 要採用哪一個
  - RegDst_o：控制要寫回的 Register 是哪一個
  - Branch_o：決定是否為 Branch 指令
  - Jump_o：決定是否為 Jump 或其他跳躍指令
  - MemRead_o：決定是否要寫入記憶體（通常是 sw 指令）
  - MemWrite_o：決定是否要讀取記憶體（通常是 lw 指令）
  - MemtoReg_o：決定是否要寫回暫存器
- ALU_Ctrl.v：由 Instruction 及 Decoder 的控制訊號決定要對 ALU 進行什麼操作
  - funct_i：Instruction 的低 6 位元

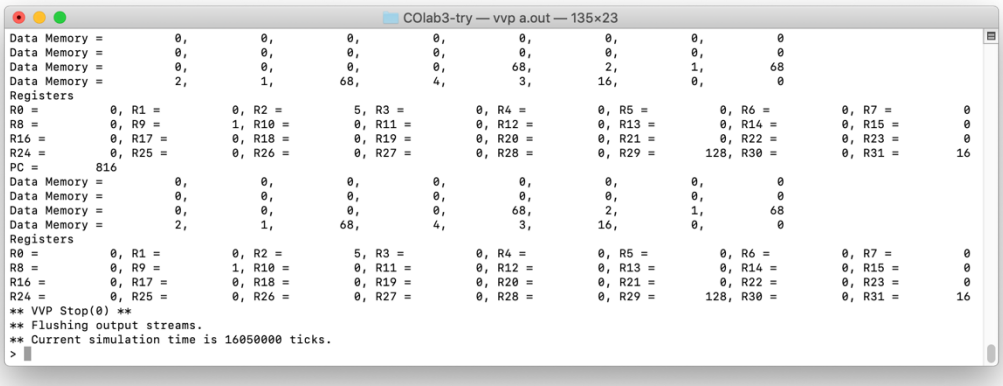
- ALUOp_i：Decoder 送來的 2-bit 控制訊號
- ALUCtrl_o：由上面兩個訊號決定實際要送往 ALU 的控制訊號

- Sign_Extend.v：把 16-bit 的訊號 extend 成 32-bit，並維持正負號
  - data_i：16-bit 輸入訊號
  - data_o：32-bit 輸出訊號
- ALU.v：運算邏輯單元
  - src1_i：32-bit 輸入訊號
  - src2_i：32-bit 輸入訊號
  - ctrl_i：4-bit 控制訊號
  - result_o：32-bit 計算的結果
  - zero_o：判斷計算出來的結果是否為 0
- Shift_Left_Two_32.v：把輸入進來的位址乘以 4
  - data_i：32-bit 輸入進來的訊號（事實上是位址）
  - data_o：32-bit 輸出訊號

Finished part:



CO_P3_test_data1.txt 的實驗結果



CO_P3_test_data2.txt 的實驗結果

兩題做出來的結果都跟助教提供的解答相同。

Problems you met and solutions:

這次實驗我遇到的問題大部分都在第二題，經過研究發現是跟 jal 指令有關（藉由每個步驟會印出每個 Register 跟 Memory 的值），因為我一開始沒有正確地把要返回的位址存在 Reg[31]中，也就是沒有把 link 的部分處理好，所以後來的結果會整個大亂。另外一個遇到的問題是因為提供的 **diagram** 跟實際上要完成的架構有些微不同，為了要釐清要添加跟刪減哪些控制訊號與線路，也花了蠻多時間。

Summary:

這次的實驗相較於上次，多了存取記憶體的 lw、sw 指令跟 jump、jal 等跳躍指令。所以 Decoder 跟 ALU Ctrl 的設計都變得複雜了許多，要添加一些額外的控制訊號去處理。這些部分花費了我許多時間，包括要先回憶上次的內容（都忘光光了），再來是歸納與整理每個指令對應的到控制訊號等等。透過這次實驗，算是瞭解了一個完整的 Single Cycle CPU 應該如何設計與實踐。