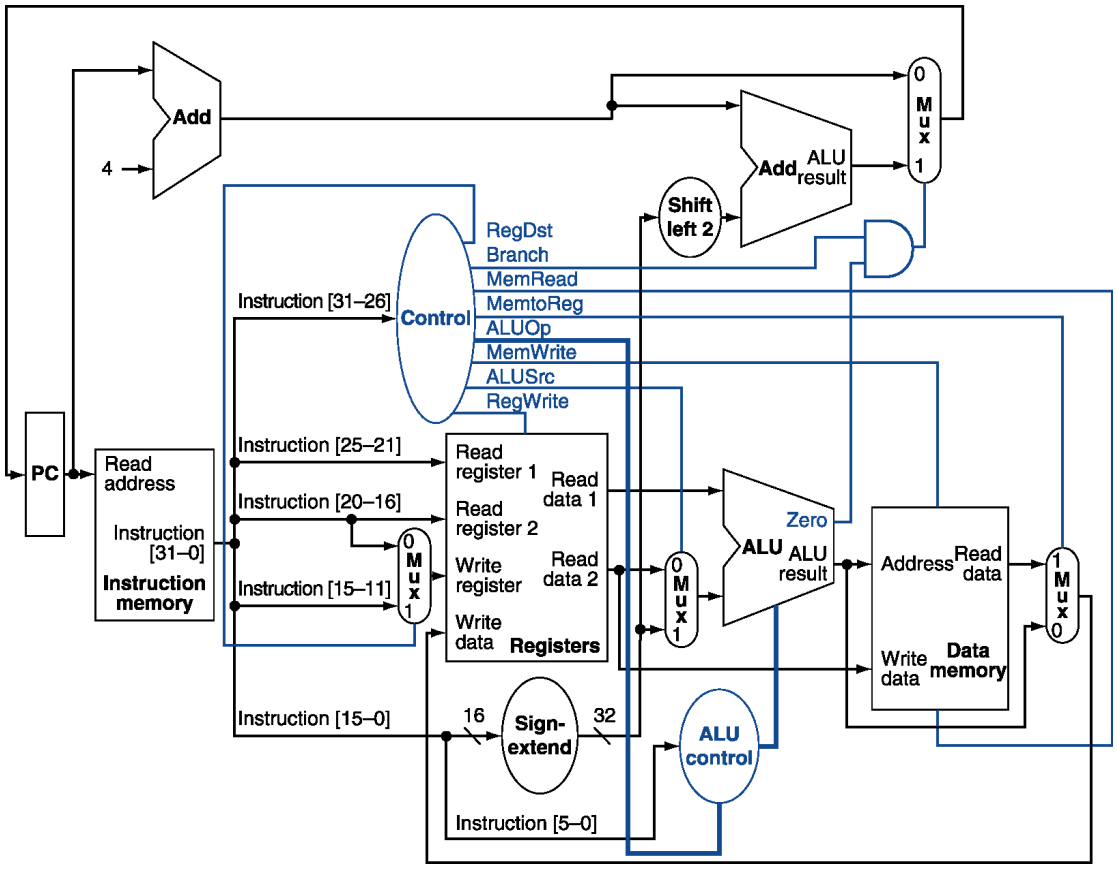
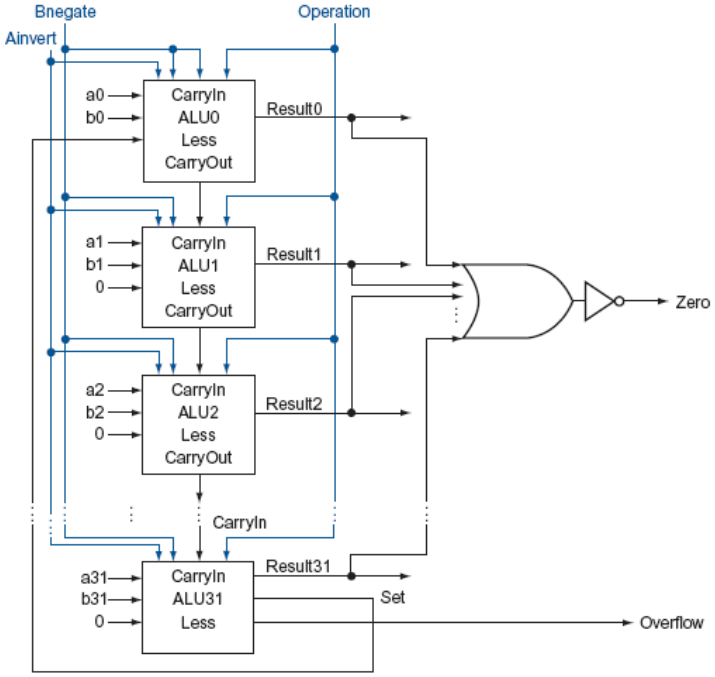
Computer Organization

Architecture diagrams:

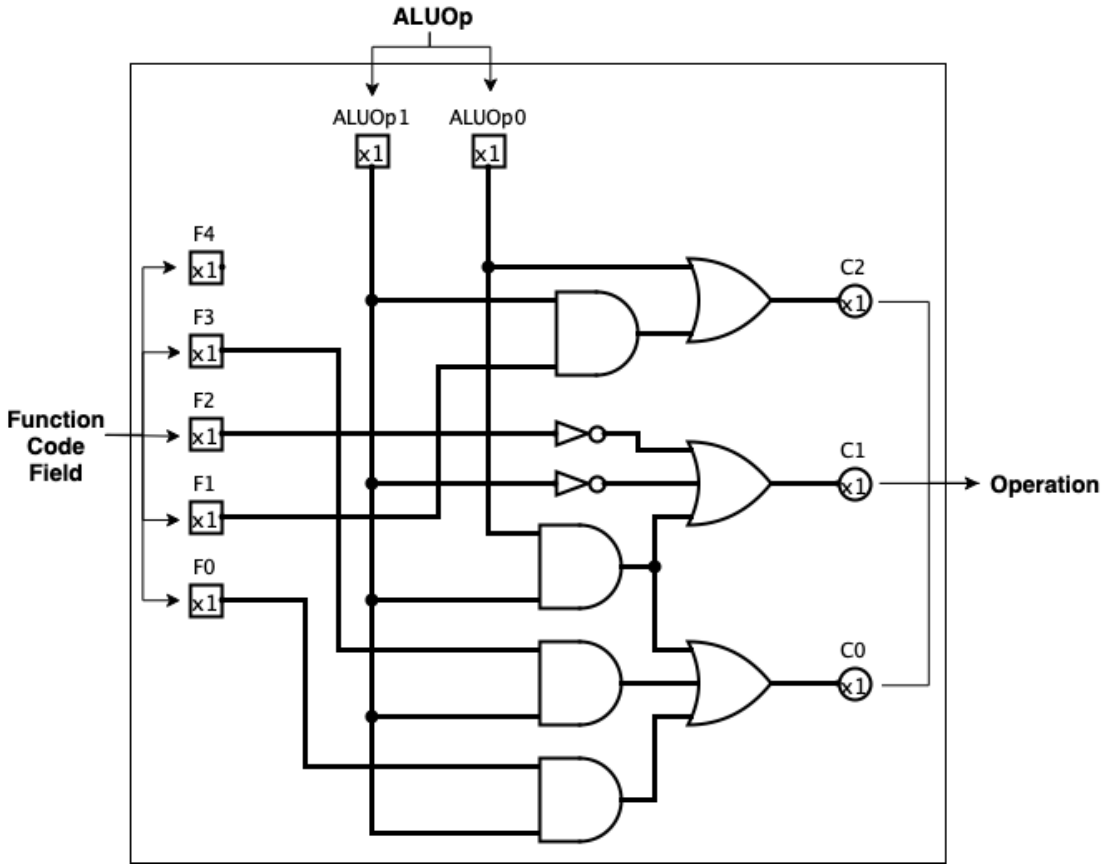
Simple_Single_CPU.v



ALU.v



Decoder.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)
    有兩個，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)
  - 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)
    根據指令的 OP Field 決定要送往各個單元的控制訊號

- module ALU_Ctrl（funct_i, ALUOp_i, ALUCtrl_o）
  根據 Decoder 送來的控制訊號以及 Instruction 的 Function Field（如果需要的話）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）
  跟上個作業的 ALU 很接近，但這次用 Behavioral 的方式實現，相對來說簡單一些
- 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

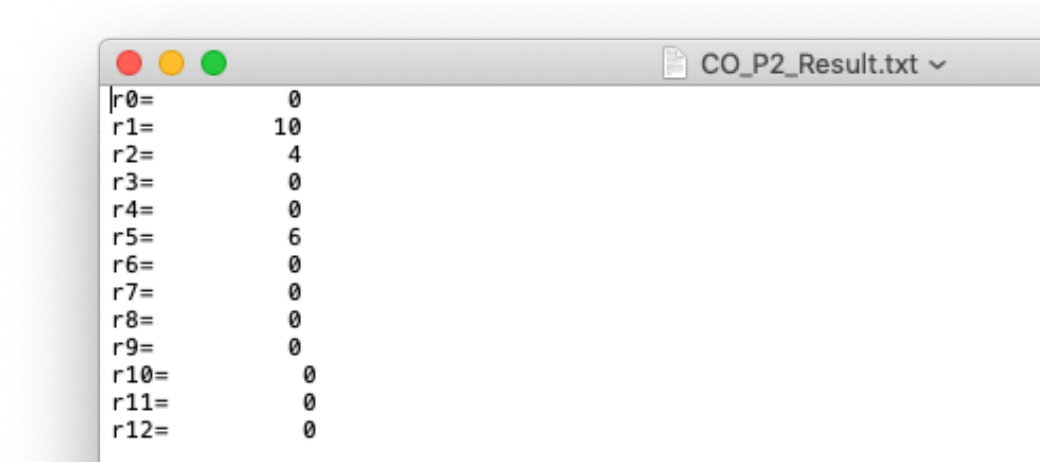
訊號分析

- 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

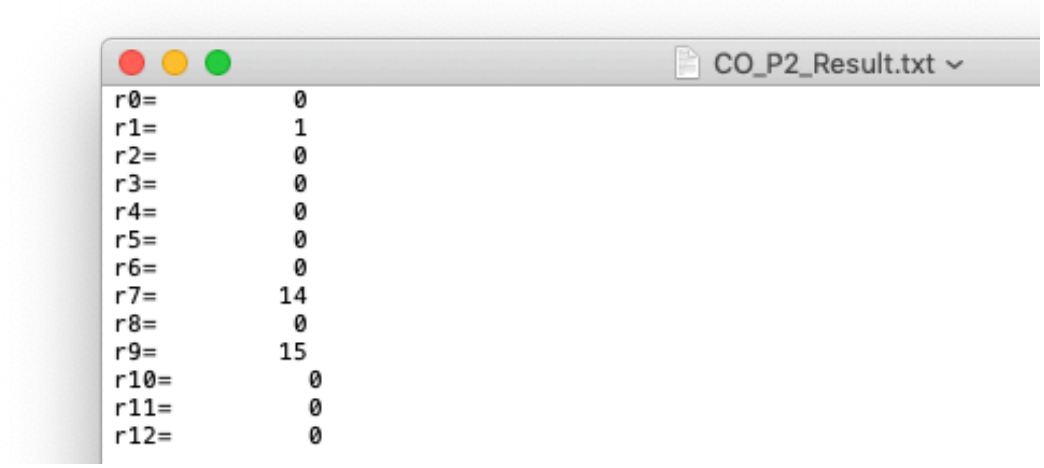
- RegWrite_i：控制是否要寫回暫存器
- RSdata_o：讀取到的 RS 暫存器的值
- RTdata_o：讀取到的 RT 暫存器的值

- 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
- 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_P2_test_data1.txt 的實驗結果



CO_P2_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:

跟上次實驗遇到差不多的問題，一開始我在各個小單元裡面都是寫

always@(輸入訊號)

但跑出來的結果都不太對，後來把全部都改成

always@(*)

就出現正確的結果了。應該是因為我沒有考慮到某些訊號的改變也應該被納入，但實際是哪些訊號，目前還在尋找。

另外有遇到一個問題是 Decoder 的設計，要把八個輸入訊號的邏輯化簡，我覺得這個實驗最有技巧性的地方，一開始很抓不到竅門，用了之前邏輯設計教的方法發現根本行不通，後來在重看了一次講義和課本才回想起老師教的方法，順利解出這個問題。

Summary:

這次的實驗是做一個 Single Cycle CPU，主要的工作是把各個小單元互相連接好，並完成裡面的功能，大部分的小單元功能都很簡單，再加上複雜的 Program Counter、Instruction Memory 和 Register File 都由助教提供，所以我想這次最有挑戰性的就屬 Decoder 的設計了。雖然因為目前還沒有納入 pipeline 的設計，控制訊號的設計相對來說還算簡單，即使如此，Decoder 還是花費了我相當多的時間，主要是在化簡各個輸入訊號。這次的實驗讓我好好地理解了 CPU 最基礎的設計，真是獲益良多。