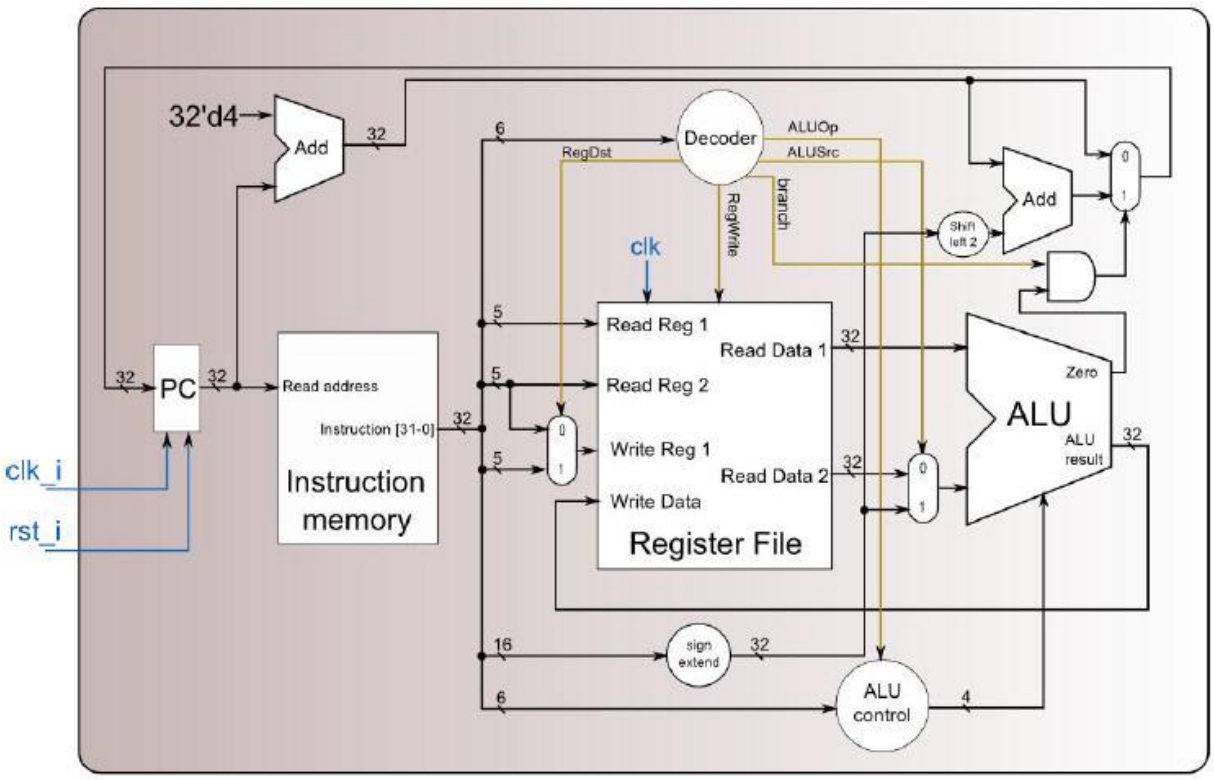
0416025 呂翊愷
0416081 趙賀笙

Computer Organization

Architecture diagram:



Top module: Simple_Single_CPU

Detailed description of the implementation:

1. **Decoder.v**
   將指令的 opfield 傳入其中一個 input Instr_op_i，依據每個不同的 opcode，控制所有其他的 module，例如: RegWrite_o 控制要不要把運算結果寫入 register、ALU_op_o 告訴 ALU Ctrl 是哪種 Opcode、RegDst_o 告訴 Reg_File 要存到哪個 register、Branch_o 判斷 opcode 是不是 j-type 的指令、ALUSrc_o 控制 ALU 的第二個運算元是來自 Register 或 Instruction Memory。
2. **ALU_Ctrl.v**
   透過 Input funct_i 和 ALUOp_i 判斷 ALU 要做哪種計算
3. **ALU.v**
   根據 ctrl_i 對 src1_i 和 src2_i 做出不同的運算，種類有 and, or, add, sub, nor,

nand, slt, sra, srav, lui, ori, bne。
如果要做的運算是 bne 且相減結果不等於零，則 zero 等於 1。
如果要做的運算是 beq 且相減結果等於零，則 zero 也等於 1。
為了做出 sra、srav 的運算，把 src2_i 改成 signed 的型態，並且增加一個 input shamt_i 將 instruction 的第 6~10 位傳入 module，判斷 sra 指令要右移幾位。
要做出 lui 的運算，直接將第 0~15 位的數字換到第 16~31 位，原本的第 0~15 位則都為 0。

4. **Adder.v**
   把兩數相加並輸出。
5. **MUX_2to1.v**
   根據 select_i 判斷要輸出 data0_i 還是 data1_i。
6. **Shift_Left_Two_32.v**
   把 data_i 左移兩位後輸出。
7. **Sign_Extend.v**
   把原本只有 16bit 的 data_i 增加為 32bit，作法是將 sign extend 過後的數字第 16~31 位等於原本的第 15 位。
8. **Simple_Single_CPU.v**
   主要是將各個 module 呼叫並把 module 之間的線接起來。以下說明各個 module 之間的關係：

   PC, Adder1, Adder2, Mux_PC_Source：
   將 PC 的 output 結果接到 Adder1 讓 address 加 4，然後再傳入 Adder2，若運算指令為 j-type，則會把 address 再相加到應該要 jump 到的位址，最後透過 Mux_PC_Source 判斷要不要跳到 j-type 指令要跳的位址。

   PC, IM：
   將 PC 的 output 結果接到 IM，IM 再根據傳入的 address 讀出 register 裡面的 instruction。

   Decoder, Mux_Write_Reg, RF, Mux_PC_Source, Mux_ALUSrc, AC：
   Decoder 將 IM 輸出的指令第 31~26bit 接進來，然後根據指令種類分別控制 Mux_Write_Reg, RF, Mux_PC_Source, Mux_ALUSrc, AC。

   IM, RF, Mux_Write_Reg, SE, AC：
   將 IM 輸出的指令第 25~21, 20~16 位傳入 RF，輸出兩個 register 的兩筆資料；第 20~16, 15~11 位傳入 Mux_Write_Reg，判斷要把運算結果存到哪個 register；第 15~0 位傳入 SE，若 instruction 為 I-type，則第 15~0 位為一個常數，將該常數做 sign extension 然後輸出，若 instruction 為 J-type，則第

15~0 位為一個 address，將該常數做 sign extension 然後輸出；第 6~0 位傳入 AC，AC 判斷做出甚麼運算後再傳入 ALU。

RF, Mux_ALUSrc, AC, ALU, Mux_PC_Source：
ALU 根據 AC 的結果判斷要做甚麼運算，然後將 RE 和 Mux_ALUSrc 的 output 做運算，並將運算結果傳回 RF，讓 RF 將結果寫入 register。若 instruction 為 J-type 指令，則將 output zero_o 傳入 Mux_PC_Source

Problems encountered and solutions:

在寫 decoder 的 module 的時候，需根據每個 operation 判斷每個輸出是多少，在做到這個部分時想了很久，因為需要去了解每個 operation 的特性才有辦法寫出正確的 output。在寫 ALU 中做 sra, srav 運算的時候，一開始在使用>>>這個運算子的時候，發現結果並沒有把 sign bit 做 signed extension，導致答案錯誤，後來研究>>>運算子的特性，發現如果要使用該運算子的話，前面運算元必須使用 signed 型態，於是把 src2_i 改為 signed 型態終於正確。

Lesson learnt (if any):

透過這次的作業，我終於深刻了解 CPU 的運作模式，也了解到原來電腦在做運算的時候，是按照這樣的邏輯去執行。在 coding 方面，我們也不知不覺增進自己寫 verilog 的能力，尤其是學會了>>>這個運算子。做完這個作業，實在是收穫滿滿，以後 CPU 的運作我大概是忘不了了。