Modules Explanation

1. Adder

   這個 circuit 會讀兩個 32 bit 的整數，然後輸出兩者的和，不考慮溢位。

2. ALU_Control

   這個 circuit 會讀一個 10 bit 的`funct`和 2 bit 的 `ALUOp`，`funct`由 instruction 的`funct7`和`funct3`接在一起而成，`ALUOp`則是 Control 的輸出之一。ALU_Control 會先根據`ALUOp`是 10 還是 00 判斷 instruction 是 R-Type 還是 I-Type，然後再根據`funct`決定要輸出什麼 3 bit `AluCtrl`給 ALU。

3. ALU

   這個 circuit 會讀入兩個 32 bit 的有號整數，及 ALU_Control 輸出的 3bit `AluCtrl`，然後根據`AluCtrl`決定要輸出以輸入的兩個整數的哪種運算結果，不考慮溢位。有 ADD、SUB、MUL、AND、XOR、SLL、SRA 這 7 種運算。另外還會輸出 Zero，因為在這個作業不會用到，所以我就把它設為 0 了。在實作`srai`的時候，因為 immediate 只有 5 個 bit，所以要把第二個整數 input 和 11111 做 bit and ，不然會錯。

4. Control

   這個 circuit 會從 instruction 中讀 7 bit 的`opcode`，然後根據`opcode`代表的 instruction 來輸出要給 ALU_Control、ALU、register 的訊號。如果`opcode`代表的是 R-Type instruction，就把`ALUOp`設為 10 並把 `ALUSrc`恆設為 0。如果是 I-Type instruction，則把 `ALUOp`設為 00 並把`ALUSrc`設為 1。因為這次的作業只有這兩個 type 的 instruction，而且都需要寫入 register，所以我把 `RegWrite`恆設為 1。

5. CPU

   在這裡根據 spec 的 datapath (Figure 1) 把所有的 module 接在一起。根據每條在 datapath 上的線 (有分叉的算同一條) 都宣告一條 wire，然後在`CPU.v`裡面的括號填入對應的 wire。我在這裡還加了一個 wire four 恆等於 4，專門送到 Adder 裡面幫 PC 做加法。

6. MUX32

   這個 circuit 的 input 有 2 個 32 bit 的整數和 1 bit 的`select`。如果`select`是 0 的話就輸出第一個整數，否則輸出第二個整數。

7. Sign_Extend

   這個 circuit 會從 instruction 的最左邊 12 bit 讀入整數，然後根據它的正負號決定要在 most significant bit 的地方補 20 個 0 或 1，輸入是正數就補 0，負就補 1。

Develop Environment

OS: WSL 2 (Arch Linux 5.10.60.1-microsoft-standard-WSL2) @ windows 11 21H2 (build 22000.318)
Compiler: iverilog 11.0 (stable)