Computer Architecture Lab1 Report

Name: 何明洋
Student ID: r11922208

Modules Explanation

- Note: `Instruction Memory`, `PC`, `Register`, and `testbench` 皆無改動，以下針對增加的module進行解釋

`Adder`

這個 `Adder` 主要是拿來執行 `PC=PC+4` ，故簡單實作了一個接受兩個32 bits source並回傳相加值的module，而加法本身即有 + operation支援。

`Control`

根據題目要求的data path， `Control` module接受 `instrucion[6:0]` ，回傳 `ALUOp`, `ALUSrc`, and `RegWrite` ，而根據下表：

opcode	function
0110011	and, xor, sll, add, sub, mul
0010011	addi, srai

可以發現能夠用是否為 `0110011` 區分兩者，故在實作中定義了 `RTYPE` 為 `0110011` ，比較 `instrucion[6:0]` 是否為 `RTYPE` ，若是的話 `ALUOp` 設定為 `10` ，若不是的話設定為 `11` 。
考慮 `ALUSrc` ，可以發現也只有 `addi` 與 `srai` 需要來自 `immediate` 的source，所以
當 `instrucion[6:0]` 為 `RTYPE` 時， `ALUSrc` 設定為 `0` ，否則為 `1` 。
最後是 `RegWrite` ，在所需實作的所有operation都要把算出的值寫回register file，所以直接設定為 `1` 。

`MUX32`

這個module input `src0` 與 `src1` ，用 `select` signal選擇其一output，故實作上相當容易，當 `select` 為 `0` 就output `src0` ，否則為 `src1` 。

Sign_Extend

這個module要input一個12bits的 `immediate` ，回傳一個sign extended 32 bits output，所以實作上就是把最高位的bits複製20次。對於這樣的操作，參考了課堂上建議的HDLBits教程，用 `{20{data_i[11]}}` 這樣的寫法達成複製的操作。

ALU_Control

這個module較為複雜，要以 `ALUOp` , `func7` , and `func3` 來回傳ALU要使用的操作 `ALUCtr` ，為方便起見，
把 `func7` 與 `func3` 拼在一起考慮，且剛好有8種操作，
$8 = 2^3$，可以用3個bits來表示每一種操作：

ALUOp	func7+func3	function	Self-defined ALUCtr
10	0000000111	and	000
10	0000000100	xor	001
10	0000000001	sll	010
10	0000000000	add	011
10	0100000000	sub	100
10	0000001000	mul	101
11	xxxxxxx000	addi	110
11	0100000101	srai	111

依據上表，可以先用 `ALUOp` 判斷，若 `ALUOp` 是 `10` ，那就逐一比較 `func7+func3` 來找出對應的 `ALUCtr` ；若 `ALUOp` 不是 `10` ，那就判斷 `func7+func3` 是否為 `0100000101` ，若是的話即為 `srai` ，不是的話為 `addi` 。實作上用了 `define` 來為每個 `ALUCtr` 命名相對應的function name方便閱讀。

ALU

根據data path， `ALU` module要input `src1` , `src2` , and `ALUCtr` ，回傳計算結果 `res` 與 `zero` ，雖然本次 `zero` 用不到，但我也有實作。
根據前一個 `ALU_Control` module，每一種function都已經有對應的 `ALUCtr` signal，所以只要簡單判斷即可：

function	Self-defined ALUCtr	operation
and	000	&
xor	001	^
sll	010	<< (unsigned)
add	011	+
sub	100	-
mul	101	*
addi	110	+
srai	111	>>> (signed)

因verilog已經提供了這些operation，所以根據對應結果對 `src1` 與 `src2` 操作就完成了。唯一要注意的是 `srai` 只拿 `src2` 5 bits做操作。

CPU

最後就是把所有module串起來。串接上需要額外加入wire，32 bits的部分如 `instr`，`immed`，`pc`，`pc_next`，`RS1data`，`RS2data` 。Control signal含 `zero`，`ALUSrc`，`RegWrite` ，以及2 bits `ALUOp` 、3 bits `ALUCtr` 。最後是串接 `ALU` 額外需要的I/O `ALUres`，`ALUdata2_i` 。
接著根據題目提供的data path圖，以及sample code中每個module，把I/O全部接上就能完成，其中 `ALU control` 中所需的combined func7 func3可以用 `{instr[31:25], instr[14:12]}` 來達成。

Development Environment

這個作業主要是在 `MacOS 12.3` 搭載 `Apple M1` 晶片上開發的，並也在 `Ubuntu 20.04` 環境中起 `Ubuntu 22.04` Docker container測試過