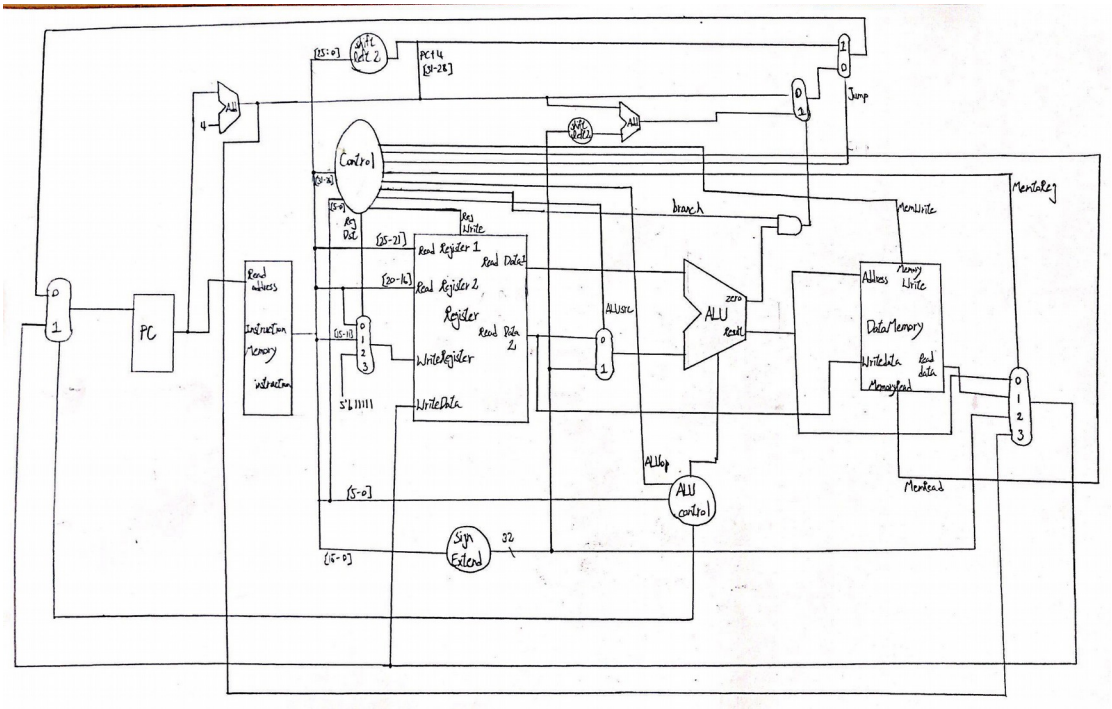
Computer Organization

Architecture diagram:



Detailed description of the implementation:

Lw : 從 memory 中取 rs+imm 位置的值存到 register(rt)中, 需要 ALU 計算並 read memory , 再 write back 回去 register 中。

Sw : 把 register(rt)的值存到 memory 中 rs+imm 的位置, 需要 control signal 去 write memory (stack pointer 存在 reg[29])。

J : 讀 opcode 跟 address, 讓 PC = 該 address 即可 (address 須先做 shift left 2, 並跟當前 PC 的前四位組合)。

mul : 輸出兩數相乘, 用 ALU 做。

NOP : 什麼事都不做, decoder 會視為 R-type 發出訊號, 但因為都是 0, 所以不影響結果。

Jal : Jump 到 address (同上 jump), 但需要把 reg[31]存 PC+4, 也就是下一條指令的位置, 讓之後可以把它存到 memory, 以便之後可以跳回來。

Jr : R-type, 我們用 ALU_Ctrl 去判斷這條指令, 並發出一個訊號讓 PC = Reg[rs] (

那個訊號在其他情況都是 0, 只有 jr 才會是 1, 因為他需要 memory 的結果卻沒有辦法讀到)。
Ble : 用 ALU 判斷有沒有 <= , 有的話讓 result = 0, zero 就會等於 1, 跟 branch and 後, PC 就會跳過幾行指令了。
Bnez : 跟 bne 一樣, 只是會讀到 0, 若不等於 0 就讓 result = 0,zero = 1, branch=1, 就會 branch 到後幾行指令了, 基本上不用更改。
Bltz : 判斷 rs 的首 bit 是不是 1, 是的話就是小於 0, 就會讓 result = 0,zero = 1, branch=1, 就會 branch 到後幾行指令了。
Li : decoder 要判斷這種指令, 發出去的訊號都跟做 addi 一樣的即可(做的事也一樣, 就是+0 的意思)。

Problems encountered and solutions:

一開始並不懂為什麼 jal 要存 reg[31], 也不懂 jal 存那個到底要做什麼, 做完後實際在 debug 一行一行看知道原理。

還有就是一開始在想 ble, bltz 的時候, 都想不到要怎麼讓 PC branch, 後來只好用發訊號的方式決定, 但我們總覺得這方法好像有點糟糕, 不曉得有沒有更好的辦法。

最後就是 mips 轉 machine code 真的好累…, 希望之後都有 machine code ~。

Lesson learnt (if any):

了解 reg 跟 mem 的交互作用, 更熟悉指令的運作原理, 也更熟悉 verilog。