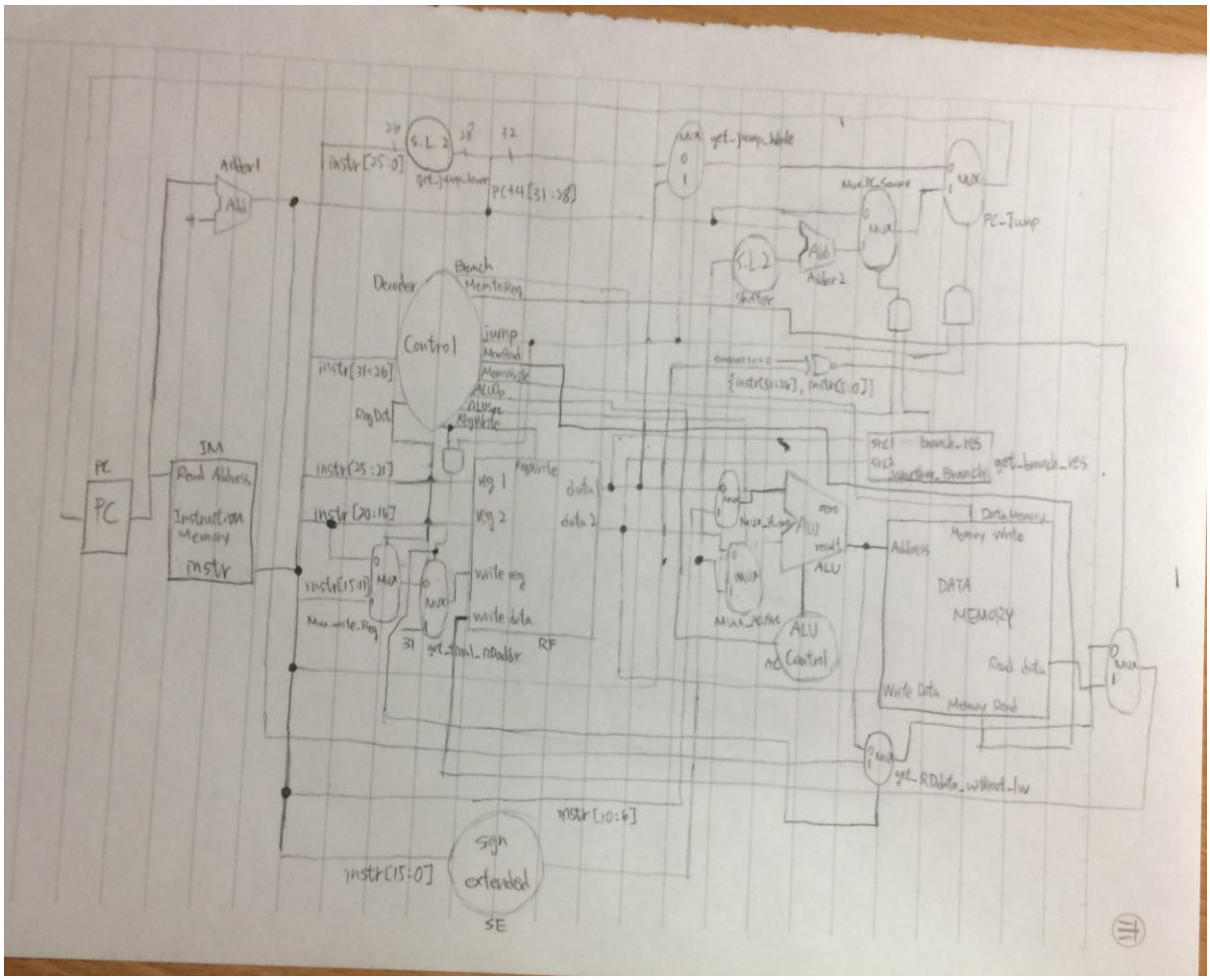
Computer Organization

Architecture diagram:



Detailed description of the implementation:

接續 Lab2

lw:

首先在 decoder，ALUSrc, RegWrite, MemoryRead, MemtoReg 都設為 1，ALU_op 則設為和 addi 一樣的 000，因為最後需要取用 memory 中 result 的值。

接著要對 memory 做存取的動作，這部分直接拿 ALU 的 result 當作 address。

最後把剛才存取的資料放到 register 中，這部分需要一個 Mux 來判定要使用之前的 data 當作存進去的值還是剛才得到的資料，MemtoReg 為 select。

Sw:

首先在 decoder，ALU_src, MemWrite 為 1，ALU_op 則設為和 addi 一樣的 000，因為最後需要讀寫 memory 中 result 的值。。

接著要對 memory 做讀寫的動作，寫入的資料為暫存器 rt 的值，位址則是剛剛 ALU 算出來的 result。

Jump:

首先要在 Decoder 中多一個判斷，如果判定 1~5 bit 是 00001，就讓 jump=1。但是在判定裡面要再判定如果是指令 j，就要讓 reg_write=0，jal 則為 1。jr 的話會先被當成 R-type，只是會再跟其他 R-type 做判定區別，一樣 jump=1，reg_write=0。

如果是 j, jal，PC 要變成後 26bits 的 shift left 2 並補上原本 PC+4 的前 4bits。jr 則是取用 rs 暫存器的值做位址。接著還需要一個 Mux 來選擇要使用 j,jal 的位址還是 jr 的位址，這部分就直接用 opcode+functioncode 來當作 select。

還有關於 jal 存取 r31 的部分要再用兩個 Mux 分別決定最後的 RDaddr 和 RDdata。

最後還要跟之前的 PC+4 或 branch 用 Mux 來決定 PC 最終要到哪個位址，這部分的 select 則是指接用 jump 即可。

Branch:

首先在 decoder 中把 branch 設為 1，接著要看是哪種 branch，這裡我們沒有照著教科書多設一個 branchtype，而是直接拿 opcode 來判定，並新增一個 whether_branch.v 來判定 branch 的結果是否為真，並把結果存在 branch_res 中。

得到判定結果之後，就放到 Lab2 的 Mux，以 Branch & branch_res 為 select，得到下一個 PC。

Problems encountered and solutions:

Branch 的部分一開始想寫在 ALU 裡面，但是後來發現 ALU_op 只有 3bits 不夠用，又不想改太多以前的東西，所以就另外寫個 whether_branch.v 來得到 branch_res。

Debug 超麻煩的，我們最後還一步一步拆解，最後才成功 debug。QAQ

Lesson learnt (if any):

學到如何寫一個完整的 CPU