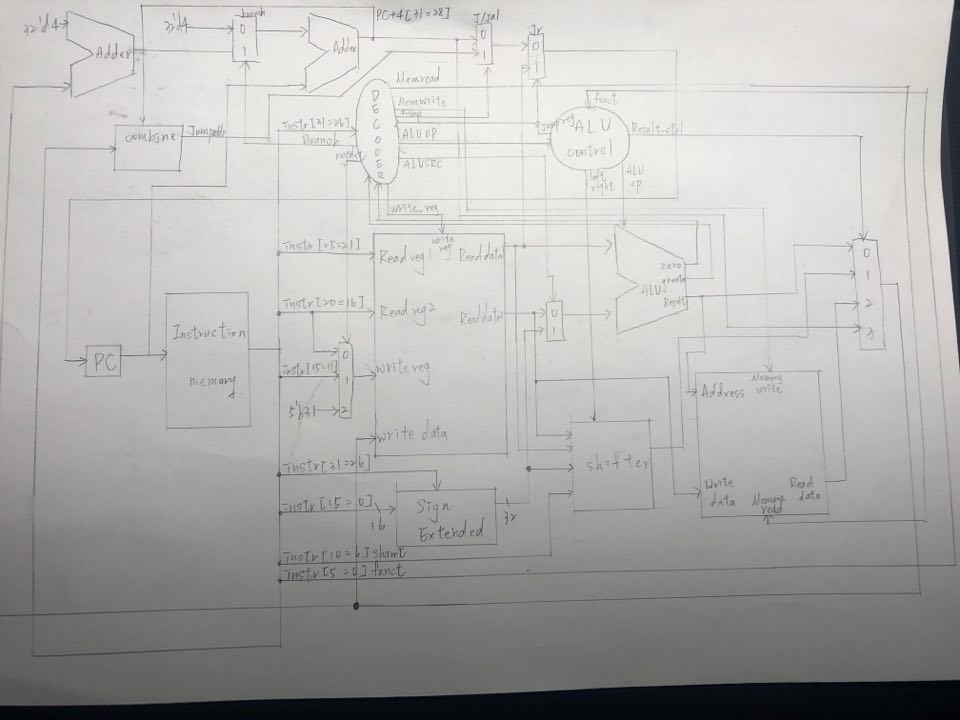
**Computer Organization Lab 3**

Single-cycle CPU With Memory Access And Jump

* Architecture diagram:



* Implement details:

這次多加了jump的功能以及branch greater than zero還有multiplication load word save word sll增加的有:

Decoder: 增加data\_memory 要不要做讀取及寫入 還有判斷jr不需要寫入register\_file的write reg.

ALU\_control: 增加ALUop的選項這次從Decoder那邊拿了全部的 op針對funct去做判斷ALU要做哪個動作，多了leftrightshift的選項來應付leftshift, 多了jump reg 去做jr的判斷接到上面的2to1MUX，result\_ctrl變成兩個bits, 來應付各種要求.

ALU: 多了multiplication 做乘法, 多了greater\_zero rs大於0就輸出，反則輸出0，zero 跟greater\_zero都接到decoder.

左上角有針對branch j jal jr去做，最後的結果接到PC.

最後要寫入register file的 data有去做了MUX 4to1的判斷，多了2 for data memory的data，3 for jal所要寫入r31的PC+4.

Register file 的write reg 前的MUX多了一個選項是jr 的r31

* Questions:

1. //j Jump

00001000000000000000000000011001

//bgtz $at, no\_swap

00011100001000000000000000000011

//bgtz $at, next\_turn

00011100001000000000000000000001

//j inner

00001000000000000000000000001011

//bne $t1, $0, outer

00010101001000001111111111110000

//j End

00001000000000000000000000011101

//beq $t1, $t2, Loop

00010001001010101111111111111110

//j bubble

00001000000000000000000000000101

2. func:

.

.

jr $ra

main:

.

jal func

mul $3, $2

上面事先寫一個function是叫func 裡面的指令是jr $ra 跳到$ra所存的值的值, main 裡面有一個jal func意思就是跳到func的位址然後儲存$ra到下一條指令的位置，意思是說在執行main的jal會跳到func並且記住jal下一條指令(mul)位置，跳到func執行到jr再跳回原本所儲存在$ra裡的位址跳到mul

3. slt $rd, $rs, $rt bne $zero, $rd, offset 運用這兩個instruction就可以先判斷rs有沒有小於rt 得到的結果再跟zero比較有沒有不相等不想等代表rs>=rt 就branch

4.我認為lab3的blez, bgtz, jal都可以改掉換成其他instruction，換掉的好處是可以減少cpu的某一些元件的輸入輸出，缺點是要打兩行以上的code來達成，無法只用一行表示，刪掉三個後還有16個instruction.

* Contribution

這次都是由兩個人所完成bug 也是兩個人一起找到的，只有小分工一下誰改simple\_single\_cpu, decoder, alu\_ctrl, ALU, 多種MUX之類的，但還是兩個人都有改，基本上都是兩人合力完成的。

* Discussions, problem encountered and miscellaneous

感覺上這次作業因為是建立在上一個作業，所以我覺得坐起來比上次還要簡單一點。對於CPU也越來越了解。感覺越來越完整。