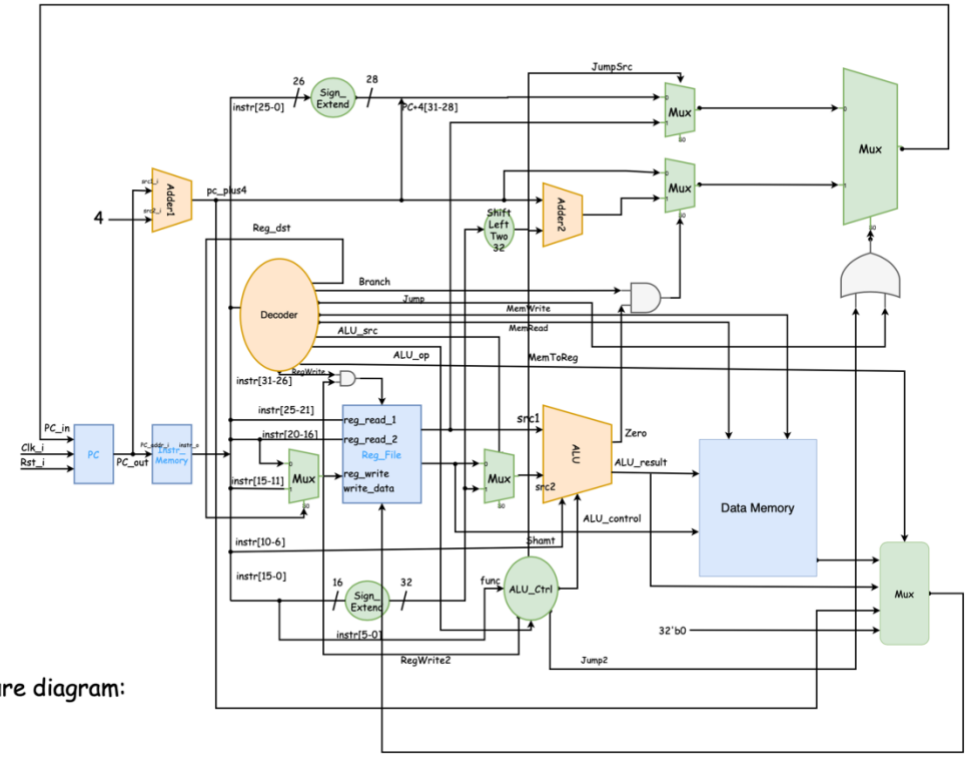
CO_LAB3

0716055 王耀德

1. Architecture diagram :



Architecture diagram:

2. Implementation details:

這次多的部分包含：

1. RegWrite: 因為多了 jump register 的關係，要和 ALU_Ctrl 確認是 jr 還是一般 R-type，所以用一個 And gate 做 Reg_write。
2. Data Memory: 把 ALU 的結果傳進 Data Memory，再決定要 read 還是 write。

3. Write_data: 透過最右邊的 4 to 1 Mux，依照 control 選擇寫入 Data Memory、ALU_result、PC+4 或是 32’b0。

4. Jump: 新增的第一個 Mux 決定要依照後 26 bits 還是 register1 來 jump，後一個 Mux 則是決定是 branch 還是 jump。

3. 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. jal 跟 jr 可以實現 call function 的操作，用 jal 跳進 label 並把 PC+4 記錄在 register，在 function 結束時用 jr 跳回 PC+4。

Ex: 把$a0, $a1, $a2 加起來的 function:

```
    li $a0, 3
    li $a1, 5
    li $a2, 7
    jal func
    … (next instructions)
func: add $t0, a0, a1
```

add $t0, t0, a2

addi $a0, $t0, 0

jr $ra

3. It can be replaced with :

slt $t0, $rt, $rs

bne $t0, $zero, offset

4. 可以被取代的指令：

sra	用 addi 把值存進 register，再用 srav 完成
sltiu	用 addi 把值存進 register，再用 slt 完成
lui	用 addi 把值存進 register，再 sll 左移 16 bits。
ori	用 addi 把值存進 register，再用 or 完成
bne	用 beq 後接 jump 替代。
mul	持續用 add 跟 sll 做乘法。
blez	用 slt $zero 把值存進 register，再用 beq $zero 跟 beq $register。
bgtz	可用 blez 後接 jump 替代，blez 又可用 slt, beq 替代。

Minimum instruction set:

addu	srav	jr
subu	addi	lw
and	beq	sw
or	sll	jump
slt	mul	jal

優點：指令集更精簡，CPU 的設計可以更簡單。

缺點：複雜的指令需要更多 clock cycle 完成，時間會花比較久。

4. Contribution:

   我做設計 CPU, decoder 跟 test, bug fixing。

5. Discussions, problem encountered and miscellaneous

   這次的 Lab 跟上次的流程差不多，但是指令要做的事複雜很多，讓整張圖變的滿複雜的，像是為了 jr 跟 jal 要多拉很多條線特別去做這些功能，雖然可以運行但整體的設計就變得有點亂，尤其是 decoder、Alu_Src 跟 ALU 裡面。然後 bubble sort 的部分實際測試出來時覺得很神奇，雖然檢查 instruction 有沒有寫錯很累。大概就是這樣，感謝老師跟助教的指導！