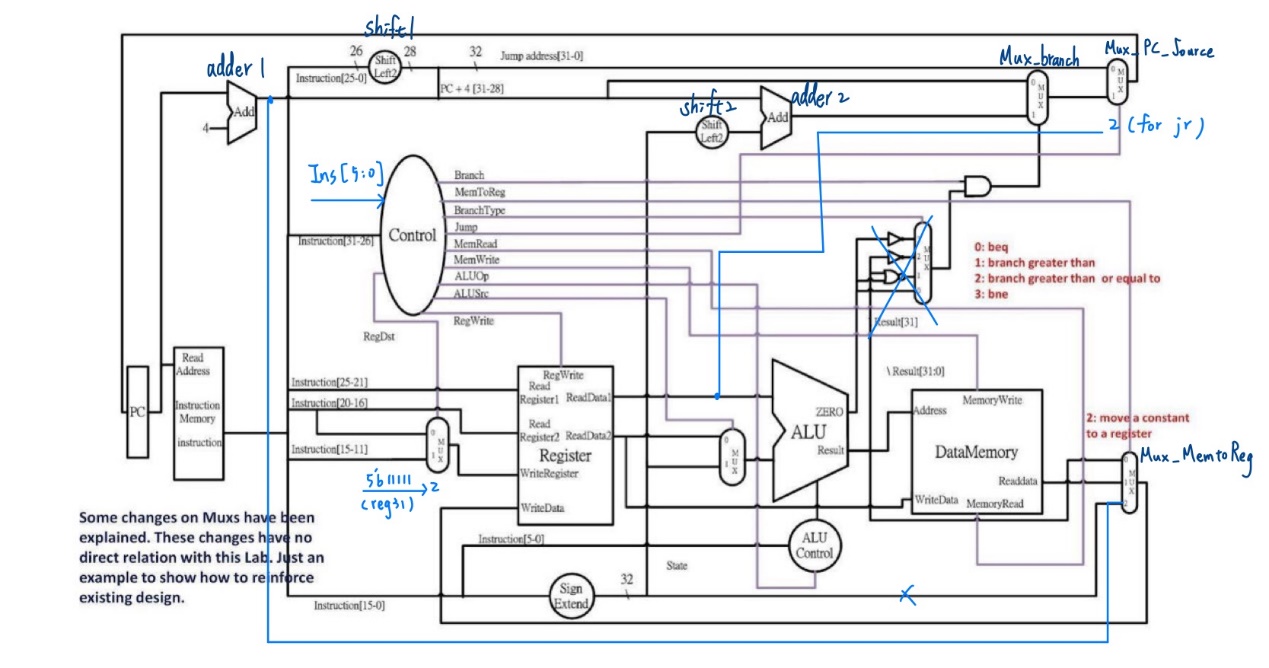
**Computer Organization Lab3**

**Name:**尤茂為

**ID:**110550065

**Architecture diagrams:**

****

**Hardware module analysis:**

****

上圖為根據不同的op field透過decoder出來的內容，在原本的decoder中加入了

ＭemRead:判斷是否需要從DataMemory讀取data

MemWrite:判斷是否需要將data寫入DataMemory

MemToReg:判斷需要將何種data再寫入Register

Jump:選擇要回傳的PC形式

function:用來進一步判斷R-type中的jr, 將其與他種R-type運算區別

這次lab所提供的MUX為2\*1，為了方便Register中WriteReg、WriteData與PC的選擇,，我將其架構改為3\*1，MUX中的select\_i也要改為2bits，所以就在decoder中稍做修改，而MUX\_branch的select需要與1 bit的zero做and，於是就將branch[0]取出與zero做and，而少數幾個MUX並不會使用到第三個輸入，這邊一致填入0;而這次的lab並沒有使用到>,>=, bne，於是就將此4\*1MUX移除。

ADDER1輸出的值為PC+4，ADDER2則為PC+4+beq經過shift後的值

***MUX\_PC\_Source*** *(jump)*

Data1:{PC[31:28],address<<2}//j jal

data2:PC+4 or PC+4+beq…//default

Data3:Reg[RS]// jr

***MUX\_MemToReg*** *(MemToReg)*

Data1:Result of ALU// R-type I-type

Data2:Data from DataMemory// lw

Data3:PC+4// jal

***MUX\_Write\_Reg*** *(RegDst)*

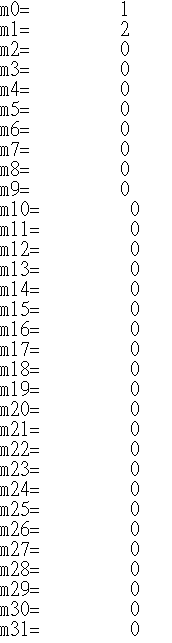
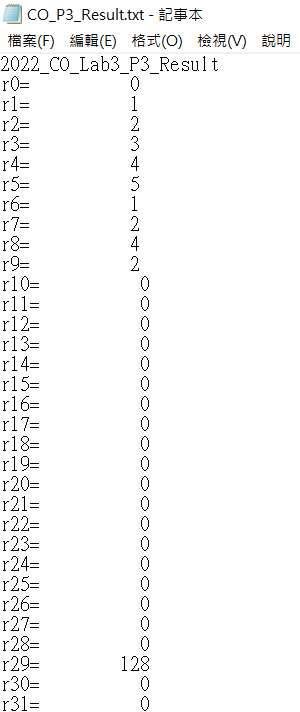
Data1:instruction[20:16]//I-type lw

Data2:instruction[15:11]// R-type

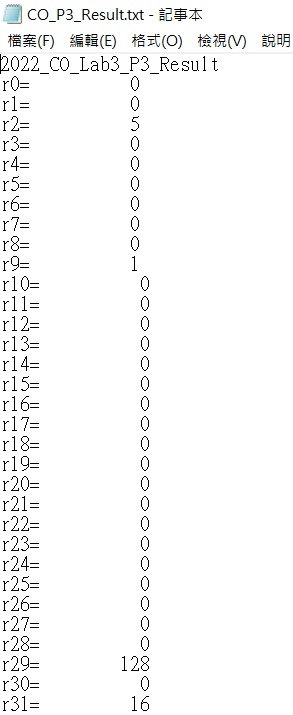
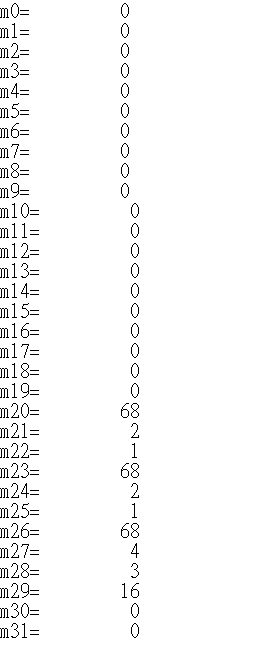
Data3:31// jal

**Finished part:**

For Part 1:



For Part 2:



**Problems you met and solutions:**

只單純用template code所提供的MUX 2 to 1在實作上需花費更多心思操作，於是稍加更改變成3 to 1。且求一致性，所有會用到MUX的都改成3 to 1，若input3不需使用到且不可能輸出，就填入相對應bit數的0。

在使用到jal時須將return address存進reg[31]，原本不太了解這邊的31要從哪輸入，上網查了一些資料後才發現原來在MUX\_Write\_Reg的其中一個input直接丟31進去即可。

**Summary:**

這次的作業相較於lab2難度又上升許多，須對MIPS與CPU的架構與操作相當熟悉才有辦法完成。在寫這次的作業前，特地回去複習了之前在CH2提到的一些相關知識才勉強地寫出這次的作業。