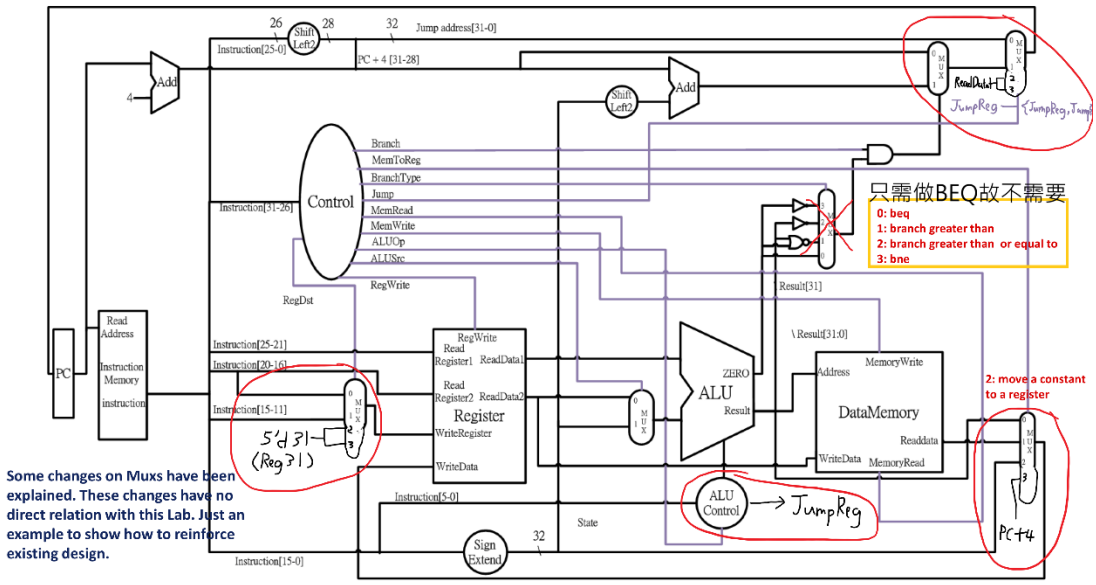
Computer Organization

Architecture diagrams:



修改內容：

1. 對提供 PC 輸入來源的多工器新增了 ReadData1 的輸入以及在選擇線上多了一條以 ALU Control 提供的 JumpReg，以支援 jr 指令可以將 ra(31th register)寫入 PC 中。
2. 由於本次 Branch 類型只有 beq，所以把 4 合 1 多工器移除。
3. 為了支援 jal，在到 WriteData 的多工器中新增一個輸入， 來自 Adder 1 的 PC+4 的輸出。
4. 為了支援 jal，在到 WriteReg 的多工器中新增了 31th Reg 的位址輸入。
5. 由於 jr 指令的分類為 R-format，在 Decoder 中無法輕易判定，為了 jr 指令，在 ALU Control 中新增了 JumpReg 的輸出。

Hardware module analysis: ModelSim

Adder.v ： 實作加法就好，將兩個數值相加
ALU.v ： 實作 ALU，可以直接參考 PDF 的 Appendix，基本的運算。
ALU_Ctrl.v ： 決定在 ALU.v 的 operation，用 switch…case…實作
Decorder.v ： 將 instruction 轉變成實作用的 code，以及對應的 ALU_Ctrl，用 switch…case…實作
MUX_2to1.v ： 兩個可能的多功器，套 if…else or .. ? .. : .. 即可
Shift_Left_Two_32.v ： 將所有位數左移 2 即可，利用”<< ”
Sign_Extend.v ： 重複 16 次 MSB 再 concatenate input value

Simple_Single_CPU.v : 將所有小程式串起來，宣告及填完相應的參數即可。

Instruction Memory. v : 紀錄所有 Instruction 資訊的記憶體，只在 Positive Clock Edge 時才可輸出值的改寫。

Register File. v : 全 32 個 32 位元的暫存器的儲存值資料 f，其中編號第二十九的暫存器 儲存 Stack Pointer 的值，初始為 128；編號第三十一的暫存器做為儲存 Return Address 使用。除了第二十九號暫存器，其餘暫存器初始值為 0。循 序電路的一部份，只有在 Positive Clock Edge 時才可輸出值的改寫。

MUX_4to1.v : 四個可能的多功器，用 switch…case…實作

Program Counter(PC). v : 計數器，只在 Positive Clock Edge 時才可輸出值的改寫，決定要取得的 Instruction Memory 的 Address。

Data Memory. v : 模擬外部記憶體。需要允許寫入的訊號才可以在 Positive Clock Edge 時將想寫入的數值寫到指定的記憶體位址。一樣也需要允許讀取的訊號才能讀取。只要允許讀取，便會輸出記憶體位置所儲存的值。

Decorder:

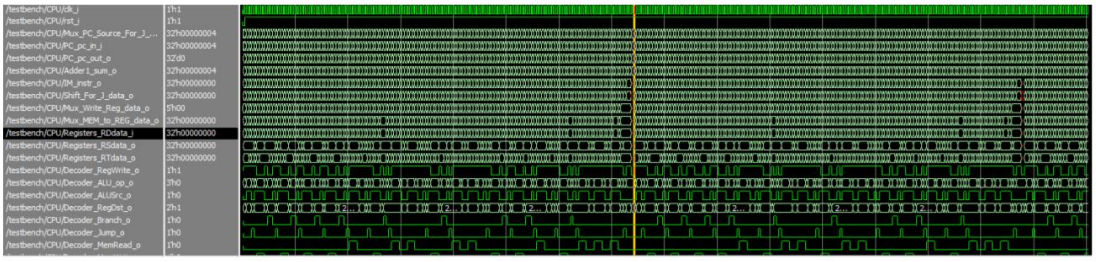
Op	ALUOp	ALUSrc	RegWrite	RegDst	Branch	Jump	MemRead	MemWrite	MemtoReg
R	000	0	1	01	0	0	0	0	00
ADDI	001	1	1	00	0	0	0	0	00
SLTi	010	1	1	00	0	0	0	0	00
BEQ	011	0	0	00	1	0	0	0	00
LW	100	1	1	00	0	0	1	0	01
SW	100	1	0	00	0	0	0	1	00
jump	100	0	0	00	0	1	0	0	00
jal	100	0	1	10	0	1	0	0	11

Finished part:

CO_P3_test_data1.txt :

```
# Data Memory =          1,          2,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Registers
# R0 =          0, R1 =          1, R2 =          2, R3 =          3, R4 =          4, R5 =          5, R6 =          1, R7 =          2
# R8 =          4, R9 =          2, R10 =          0, R11 =          0, R12 =          0, R13 =          0, R14 =          0, R15 =          0
# R16 =          0, R17 =          0, R18 =          0, R19 =          0, R20 =          0, R21 =          0, R22 =          0, R23 =          0
# R24 =          0, R25 =          0, R26 =          0, R27 =          0, R28 =          0, R29 =          0, R30 =        128, R31 =          0
```

CO_P3_test_data2.txt :



```
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,         68,          2,          1,         68
# Data Memory =          2,          1,         56,          4,          0,         16,          0,          0
# Registers
# R0 =          0, R1 =          0, R2 =          5, R3 =          0, R4 =          2, R5 =          0, R6 =          0, R7 =          0
# R8 =          0, R9 =          1, R10 =          0, R11 =          0, R12 =          0, R13 =          0, R14 =          0, R15 =          0
# R16 =          3, R17 =          0, R18 =          0, R19 =          0, R20 =          0, R21 =          0, R22 =          0, R23 =          0
# R24 =          0, R25 =          0, R26 =          0, R27 =          0, R28 =          0, R29 =          0, R30 =         92, R31 =         56
```

Problems you met and solutions:

先按照原本的 Architecture diagrams 完成後，有一些小錯誤要去想，也要為了一些更貼近真實的 CPU 去做思考，後來才想出那些地方可能有點小瑕疵及如何做調整，尤其是為了 jal 和 jr 必須改很多地方。

Decorder.v 的部分弄了很久，因為又增加了許多參數，而且畢竟它是 CPU 的核心部分，解碼錯會導致它全盤錯掉，靠試試跟想想後弄好。

同 Lab2，最困難的依然是把所有的小程式併在一起，也就是 Simple_Single_CPU.v，一個沒弄好，輸出都有問題，要花很多時間思考研究和整理每條電路的連接，然後宣告跟填上相對應的參數，參數的 bits number 弄錯害我一直沒過，而且有時候想錯參數就會擺錯位置，完全弄懂 CPU 後參數才放對，也才能看到輸出，命名也很重要，因為這樣才比較不會亂掉。

Summary:

這次新增了 J-type 的指令(jump、jal)、jr、lw、sw 等，並且在硬體上還新增了模擬外部記憶體。

這次一樣是只要把每個小部分完成，project 就大致做完了，每一小份不難，但把他們合在一起就很有難度，整併很棘手尤其是這次程式又大了，對每個細節必須更用心。先弄懂 CPU 的實作後，比較困難的只有接線填參數(Simple_Single_CPU.v)和解碼(Decorder.v)，還有要記得讀寫檔的路徑問題。

先把原圖完成後再完成 jal 和 jr ，但這兩個指令很難去實作，而且跟原圖最有衝突的就在這，因此要完全了解才比較能實作出來，這次最大的問題大概就在這了。

在 Lab 中，因為有記憶體的加入，更貼近完善的 CPU，我更加理解 CPU，也更了解 CPU 的實作方法，能用軟體模擬硬體，這真的提供不小的便利性。