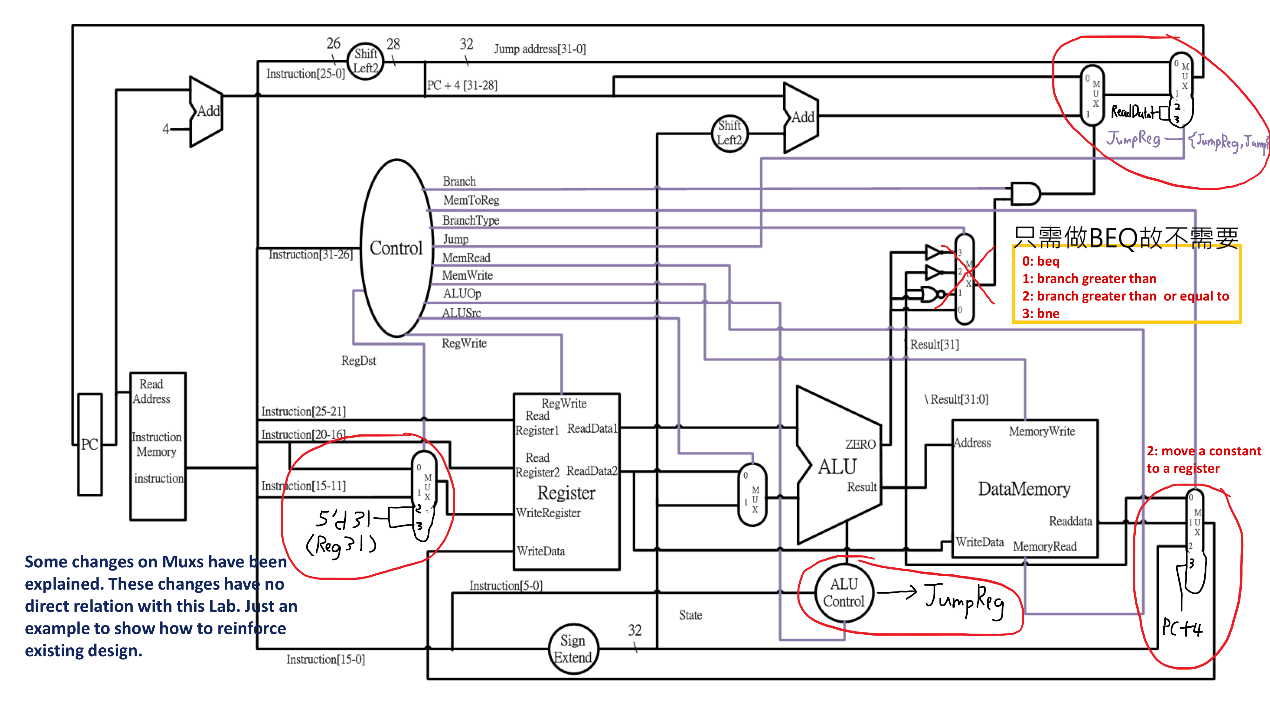
**Computer Organization**

**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

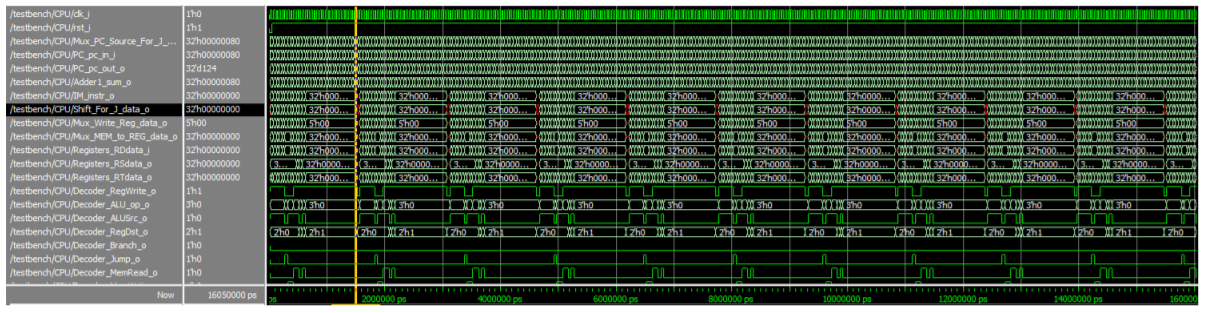
置所儲存的值。

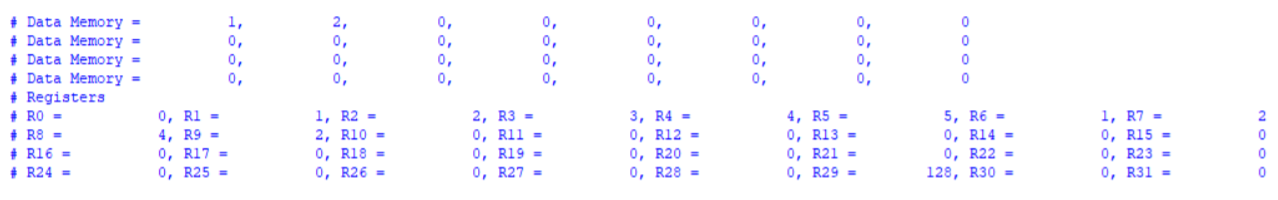
**Decorder:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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** |

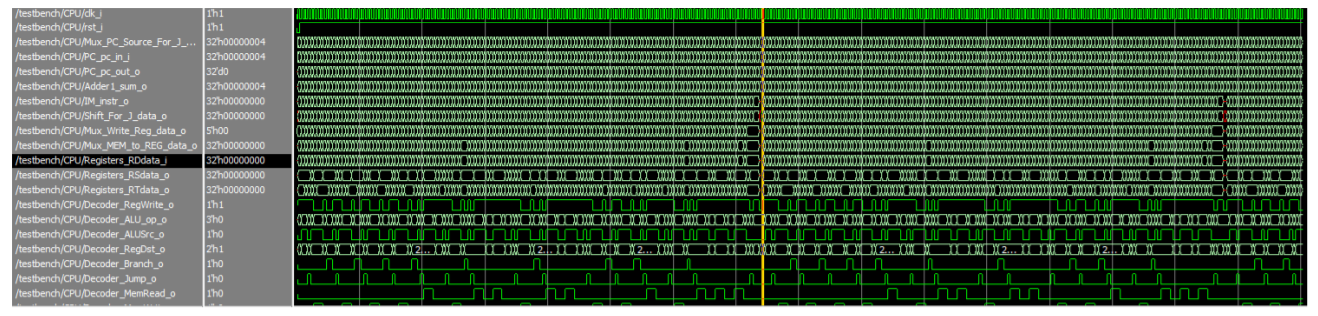
**Finished part:**

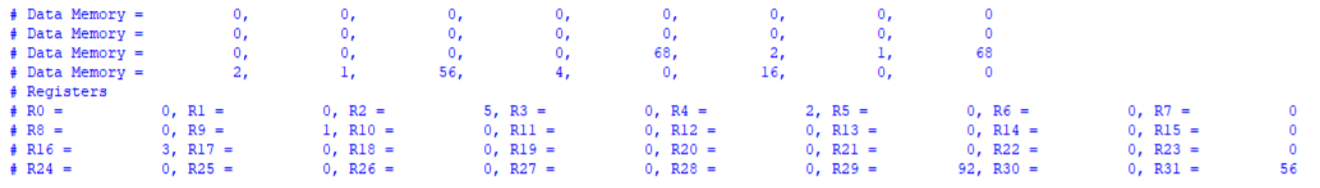
CO\_P3\_test\_data1.txt :

****

****

CO\_P3\_test\_data2.txt :

****

****

**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。