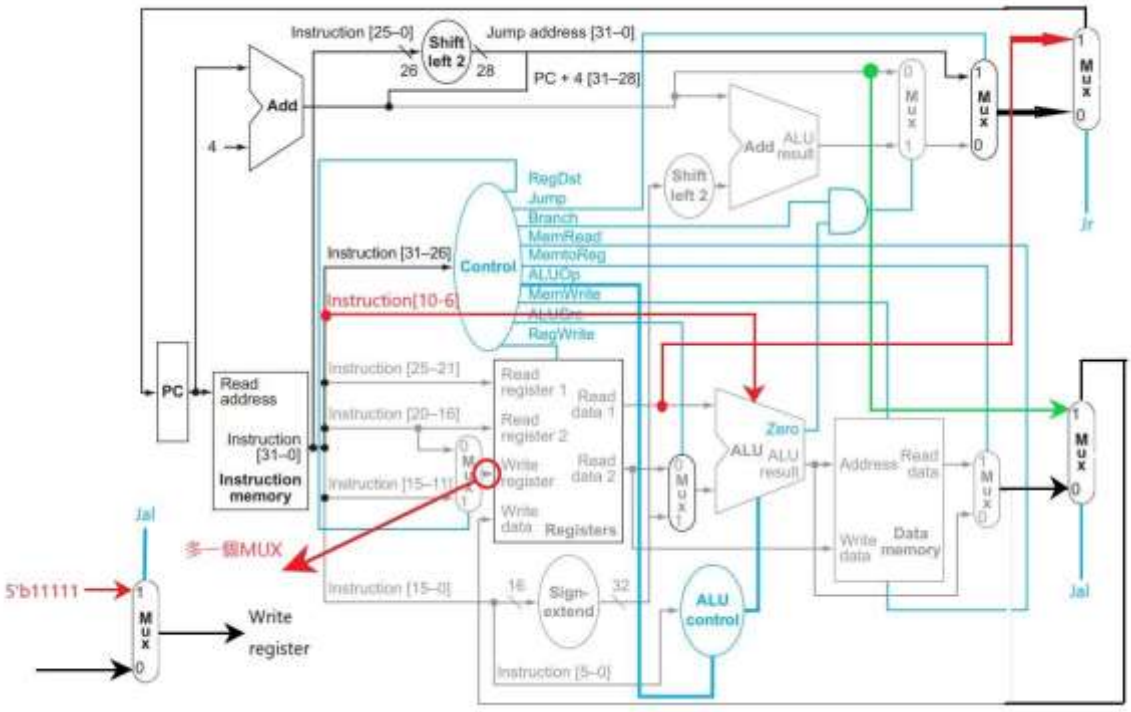
Computer Organization

Architecture diagrams:



Hardware module analysis:

本次實驗重新沿用上次的簡易線路圖，不過為了應付新的運算元，所以在 alu 以及 decorder 還有 alucontrol 對於新的 opcode 有更多的控制線。

為了應付 lw、sw 所牽涉的 data memory，我們需要一個控制線(memwrite、memread)來應付資料的存取還有提取，還有 register 的存取也由 memtoreg 來控制。

Branch 則是運用 alu 計算兩者相減是否為 0 來判斷分支是否發生，不過我沒有用 branchtype，而是直接在 alu 裡加上一個反向器來讓 bne 的 0 輸出為相反，省略了一些麻煩的控制線。

Jump 這部分較簡單，只要在 pc 計數器的部分根據 jump 的位址計算方式(offset 左移兩個單位再結合 pc 值)多計算一個位址的運算元，再用運多工器來選擇下一步指令的位址值，就可以得到新的指令。

接下來是麻煩的 jr jal，要解決這兩項指令必須在原本的線路圖多增加不少額外的指令。Jal 的部分是在 register 的 write register 前增加一個多工器讓位置值儲存進 register 的第 31 號位元裡，還有 datamemory 後多加一個多工

器來決定寫入的內容。Jr 的部分則是在 alucontrol 判斷 jr 和一般 r_type 指令的差別，再增加一個多工器再 pc 位置值的部分，讓下一個 pc 指令值讀取到 register 的 read datal，就可以讓指令回到原本 jal 的下一道指令的位址值。

Finished part:

完成 Basic instruction（Lab 2 instruction + lw、sw、beq、bne、j）、Advance set 1（ Jal、Jr ）部分

Problems you met and solutions:

一開始在 bne、beq 的部分沒有什麼問題，發生的問題頂多就是 opcode 打錯造成出來的指令錯誤。Jr 的部分一開始忘了要在新的 bit[31:28]放入原本的 pc 值+4，造成指令跳錯。

這次問題最多的地方在於 jal 和 jr 這部分，由於 jr 和 r_type 的格式類似，但走的線路卻不一樣，造成我在 regwrite 的部分出現錯誤，導致 registers 寫入的答案被後面錯誤的答案給覆蓋。後面在 alucontrol 的部分根據 function code 來判斷是否是 jr 來改變控制線的值。後面遇到一個 bug 是 reg 的值正確但是 data memory 的值卻是錯誤的，原因是 alucontrol 裡的 always 啟動值錯誤，沒有偵測到輸入運算元的改變所以輸出值沒有變化，導致 memtowrite 的值發生錯誤，沒有將答案寫入記憶體裡。為了應付這個問題幾乎花費了我 3 個小時。

Summary:

本次實驗時間較緊迫，沒有時間完成助教規定的全部指令。不過對於 jr jal 這兩項指令有許多深刻的了解，畢竟為了應付那兩項 bug 幾乎將整個程式翻了好多遍，幾乎成為了我惡夢的來源。除此之外之前的程式幾乎都是照線路圖就可以把程式給完成，而今天我們必須自己額外增加許多控制線路來應付各種狀況的發生，是目前最有挑戰的題目。