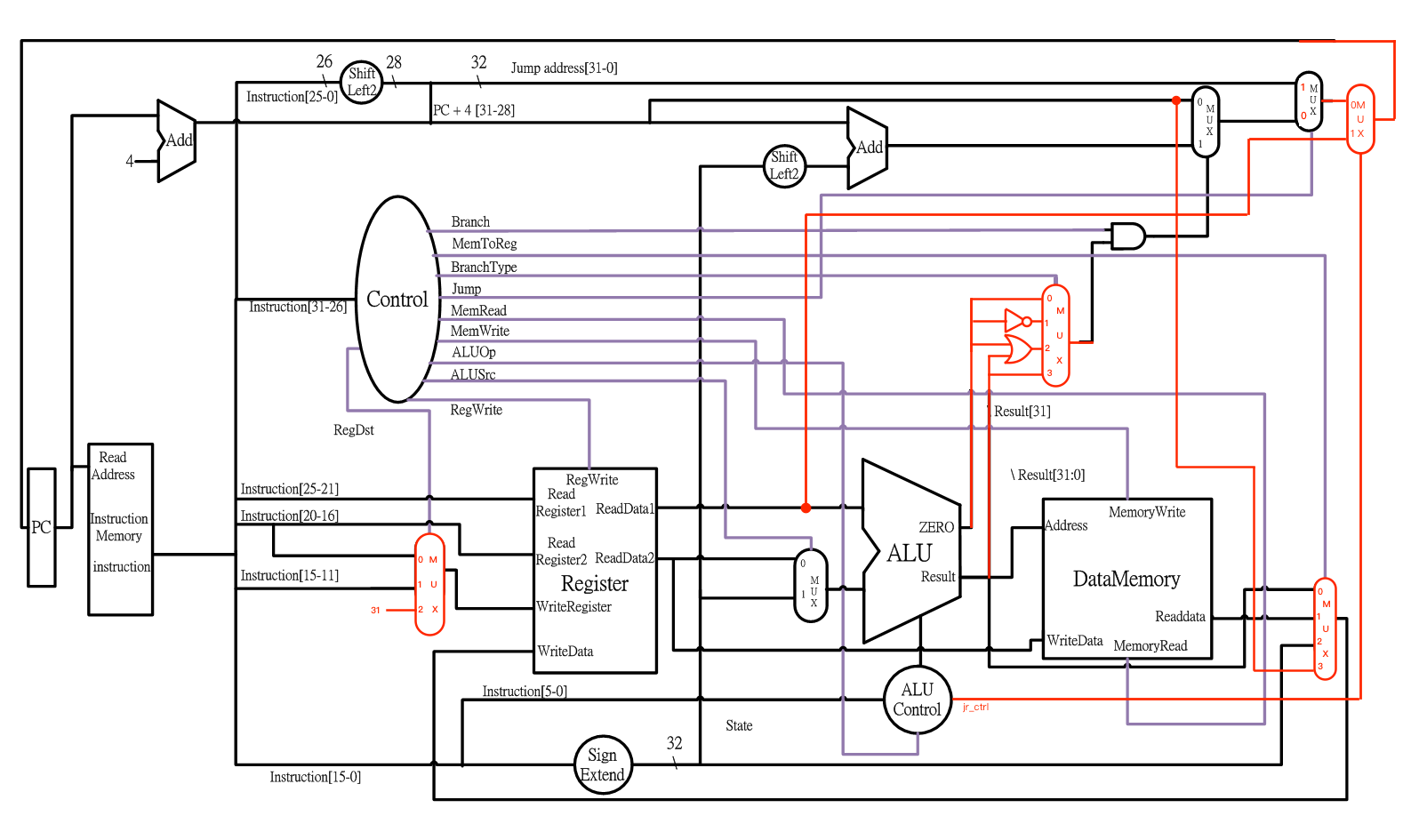
**Computer Organization**

0316055許庭嫣 / 0316313張逸群

**Architecture diagram:**

**Detailed description of the implementation:**

1. Load/Store instructions：

多了一個DataMemory的元件進行操作，要注意的便是Control衍生的MemoryWrite和MemoryRead，還有RegisterWrite的調控。先讓rs的data和immediate做ALU的加法確定位址，若做store把rd的data存入DataMemory中的那個位址即可；若做load要先確定好mux回傳Register的WriteData和另外一個mux傳出的WriteRegister，並存入Register當中。

li(load immediate)更容易施行，讓Control的MemToReg設為2，即可調控mux使得回傳的WriteData為經過sign extend的immediate了。

1. Branch instructions：

branch有五個種類：beq、bne、ble、bnez、bltz，必須確認好Control輸出的BranchType使得用ALU做減法的結果做mux可以得到需要的結果並確定是否要做branch。beq希望Zero是1；bne希望Zero是0；ble結果要小於等於，所以希望Zero是1或Result[31]是1(結果為負)；bnez可視為和bne完全相同，尤其它的rt位址直接設為0，因此Register的data即為0($zero)，與直接讓rs和rt做bne結果相同；bltz結果要小於，希望Result[31]是1。

若是否要做BranchType和ALU的結果合上Branch是1，就會讓mux做pc加offset<<2，使得下一條執行的是branch指向的label。

1. Jump instructions：

單純做jump其實很容易，讓instruction的後26bits左移兩位，前面加上(pc+4) [31:28]，並設定好Control的Jump為1使得mux可以存取需要的位址。做jal時除了做jump還要將pc+4存入reg[31] ($ra)，因此要控制Control的MemToReg為3，讓WriteData為下一個instruction的位址，並讓RegDst為2，直接讓WriteRegister為第31個位址進行存取。做jr就稍嫌麻煩，instruction中他的op field為0，會被視為R type，因此無法在Control就確定他要做jump，要先經過ALU Control讀取function field才知道要做jr，因此多設了一條jr\_ctrl控制mux，使得pc為reg[rs]存放的下一個instruction的位址。

**Problems encountered and solutions:**

branch有很多種類型，雖然都是做ALU的減法，但對結果的判斷(原圖中的問號處)會依不同的branch有不同的需求，需要稍微思考。

jump指令的文字敘述要求和附圖不相同，後來商訂好決定統一依直觀想法要做jump時Control給1。

覺得jr算是比較特別的指令，需要讀取Register中的值，並在ALU control才知道要做jump，因此要多接一些線和mux；或是判斷時直接拿出function field做比對確定是jr。必須先思考和畫出diagram才比較知道該如何做，分組作業也能更好地進行。

**Lesson learnt (if any):**

對branch、jal、jr有更深刻的認識，畫了diagram接過一次線也更了解他的運作方式了。寫過mips code轉machine code後，也了解到兩者之間的關係了。