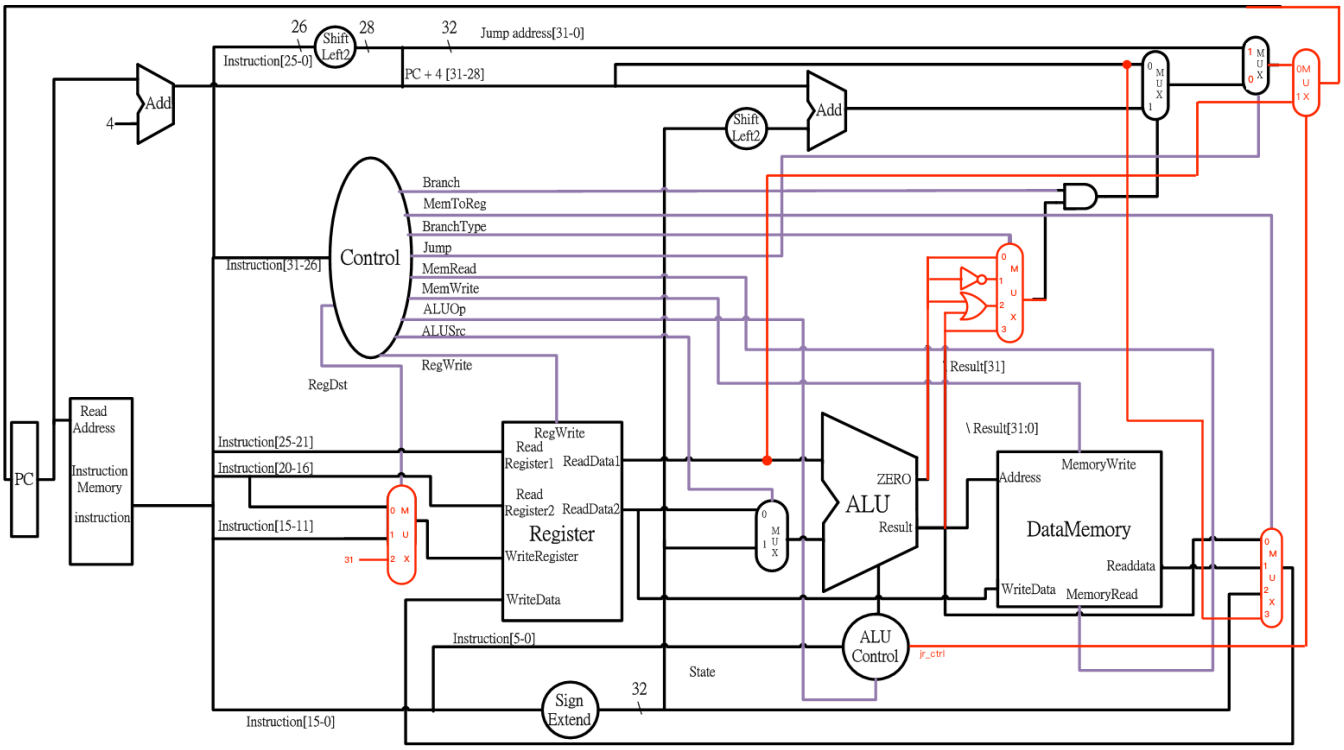
Computer Organization

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。

2. 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。

3. 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。