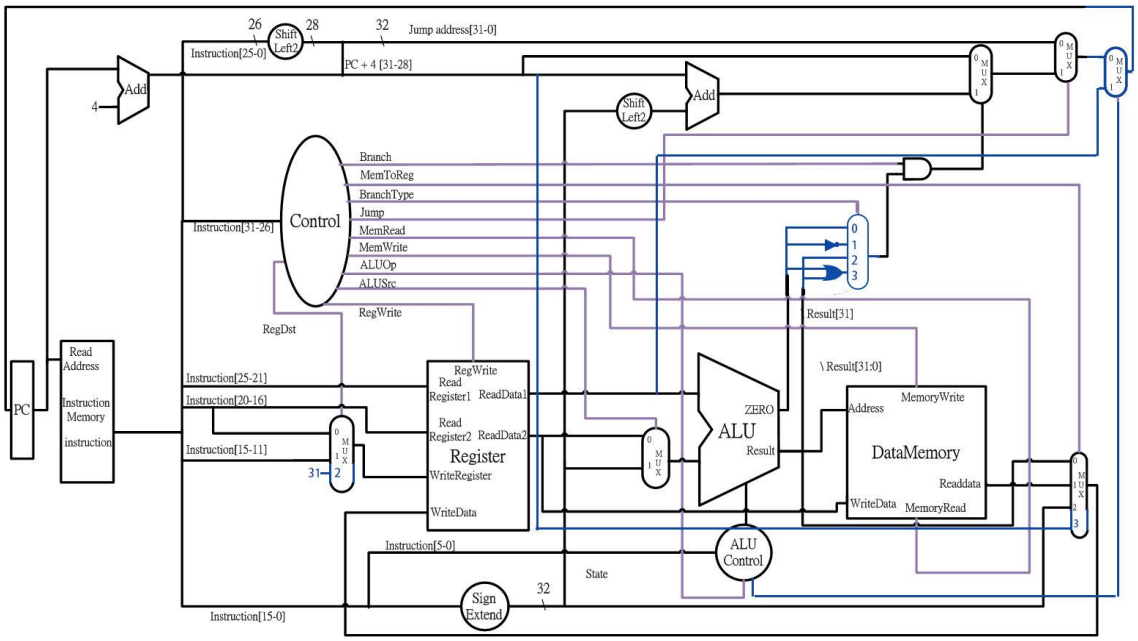
Computer Organization

Architecture diagram:



Detailed description of the implementation:

1. MUL
   與其他 R-type 操作相同，多增加一個 ALUCtrl，並且在 ALU 實作乘法運算。
   `(result_o <= src1_i * src2_i)`
2. LW
   ALU 設定為 addi 運算，啟動 MemRead，最後將 MemToReg 設為 1 透過 MUX 選擇 DataMemory 讀出的資料寫回 Register。
3. SW
   ALU 設定為 addi 運算，啟動 MemWrite，將 Register 讀出的資料寫入 DataMemory。
4. J
   線路圖上方將指令最後 26bit 左移 2bit 之後與 PC 前 4bit 結合。將 Jump 設定為 0 來選擇結合後的 PC 寫回。
5. JAL
   與 J 不同的是將 PC+4 連到 DataMemory 旁的 MUX，並將 MemToReg 設為 3、Regdst 設為 2，讓 PC+4 寫回$r31。
6. JR
   在兩個 PC MUX 後再加一個 MUX，選擇原來的 PC 或是從 Register 讀出的資料。ALU Control 讀到 JR 的 funct 後，將 MUX 設為 1 選擇 Register 資料。

7. BEQ、BLE、BNEZ、BLTZ
   將 ALU 結果的 MSB 及 Zero 按照上電路圖接入一個 4 to 1 MUX。Decoder 將 ALU 設定為 Sub，依照指令選擇對應的 BranchType，若 MUX 選擇出來的結果為 1 則將 PC +/- 對應的數字。
   BEQ：相減後等於 0（zero）
   BLE：相減後為負(MSB 為 1)或是等於 0（zero | alu_result[31]）
   BNEZ；減 0 之後不等於零(!zero)
   BLTZ：減 0 後為負(MSB 為 1)的(alu_result[31])
8. LI
   直接使用 ADDI 指令

Problems encountered and solutions:

這次整體上的線路比上次複雜許多，JR 花了比較多時間，因為他的 opcode 屬於 R-type，Decoder 沒辦法給特殊的訊號，最後讓 ALUControl 設定特殊的控制訊號。另外這次的測試資料都長很多，除錯的時候比較麻煩，後來自己先血衣些簡單的指令做測試比較方便。

Lesson learnt (if any):

因為這次的線路圖故意把一部分蓋上國防布，再加上線路本身就比較複雜，因此寫 Verilog 的時候常常弄亂，像是變數名稱弄錯之類的，下次如果也有這樣的作業，我想應該是邊寫 Verilog 邊在圖上筆記會比較好一些。