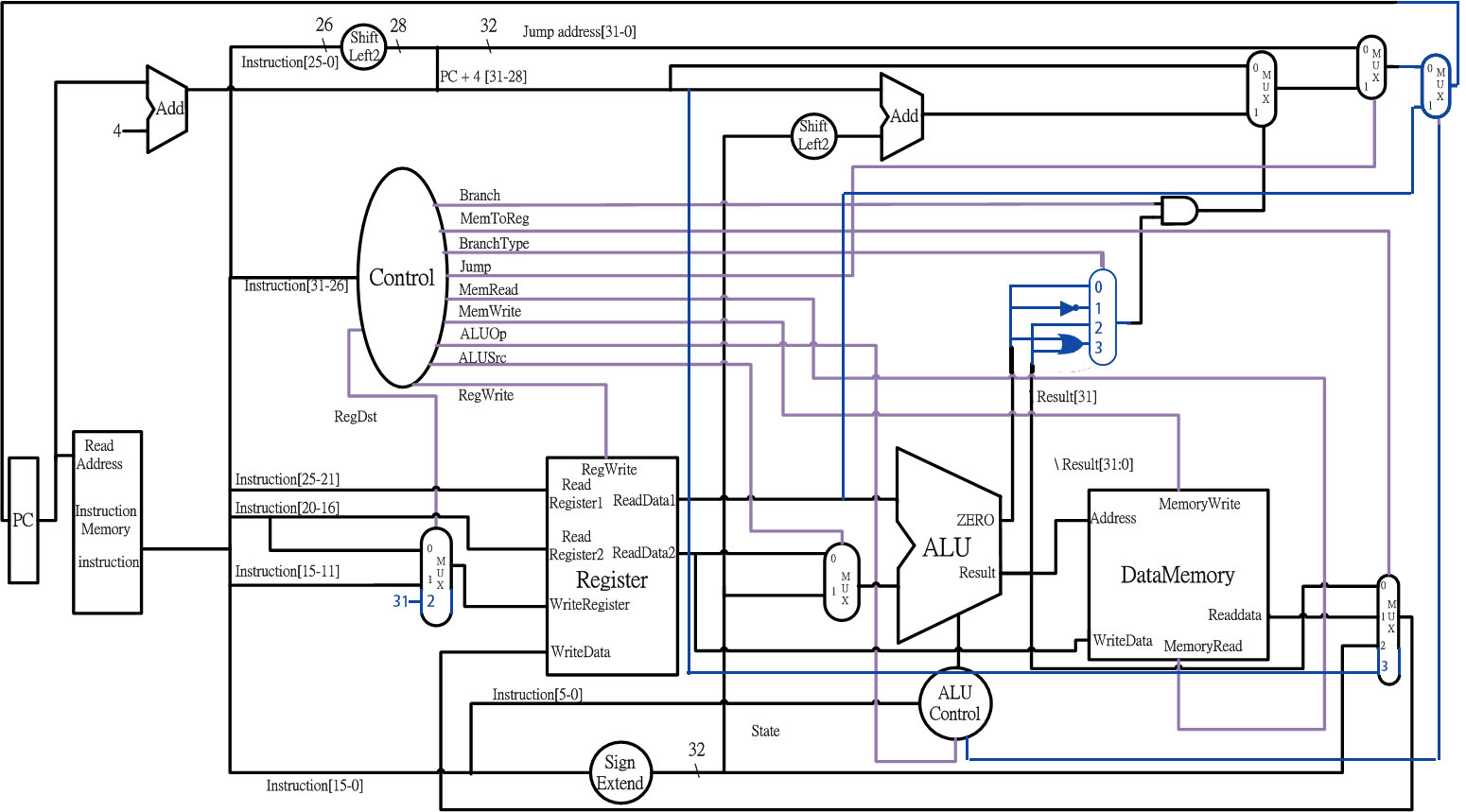
**Computer Organization**

**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)

1. 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。

1. SW

ALU設定為addi運算，啟動MemWrite，將Register讀出的資料寫入DataMemory。

1. 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。

1. 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。

1. 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。

1. 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])

1. 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。