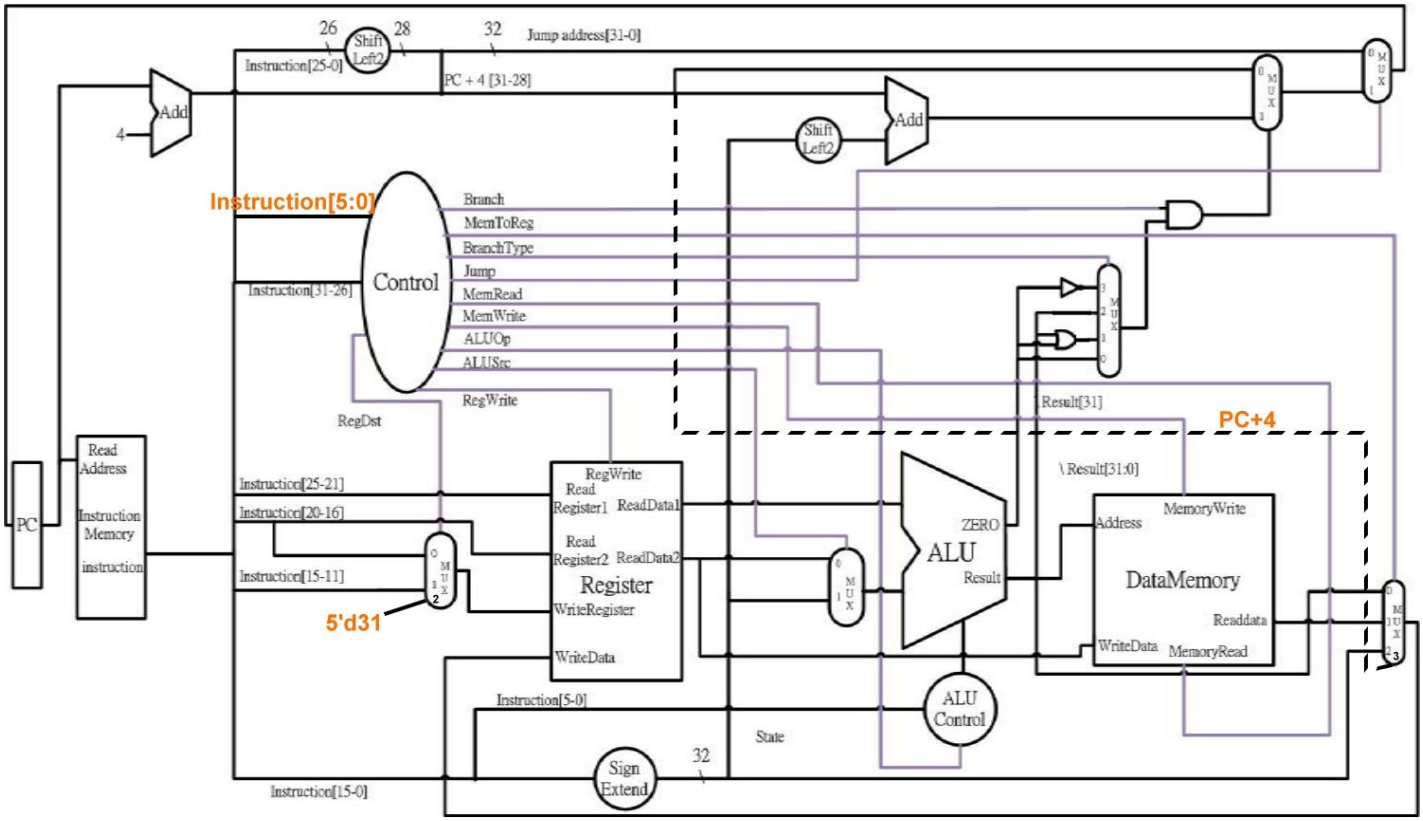
0416081 趙賀笙
0416025 呂翊愷

Computer Organization

Architecture diagram:



Detailed description of the implementation:

1. Decoder.v

由於相較於上次的架構，這次多了DataMemory，為了實作DataMemory，在Decoder裡面又加了Branch_o, Jump_o, MemRead_o, MemWrite_o, MemtoReg_o, BranchType_o這幾個output，讓decoder能夠控制DataMemory，以做出lw, sw。

而由於jr的opCode也是0，為了與其他運算指令區分，不得不把func_i也引進來Decoder，雖然想了很久，最後好像還是沒有其他方法。

2. Simple_Single_CPU.v

為了作出jal和jr，我們在寫入RegeisterFile的MUX加入PC+4，還有WriteRegister的MUX加入31(for jal寫入地址到$ra)，還有PC_source的

MUX加上Read_data_1。
為了新增ble和bltz，在branch的mux加上兩個來源~alu_result[31]和 alu_result[31] | alu_zero，用來判斷有沒有符合branch的條件。

Problems encountered and solutions:

一開始不太理解 DataMemory 的功能還有 Decoder 新增很多條線的用途，我們為了作出 lw, sw 去研究他們怎麼運作，然後發現要加入這些功能的話有些 MUX 要加入幾個 input，所以花了不少時間。之前的作業都已經習慣照著圖的結構寫出 verilog 的架構，但這次好幾個指令都要修改圖的架構才可以做出來，由於這次作業有 jump 指令，害我們 Debug 的過程相當辛苦，花了很多時間在一一比對指令跟執行結果。

Lesson learnt (if any):

我學到在波形圖中，有如茫茫大海的 0 和 1 之間，找出我想要的 0 和 1，這過程有如重度近視的人在地板上找一根針，像個迷途羔羊常常來回比對迷失了方向，我想這就是這作業迷人的地方吧。這次的測資不像上次有每個瞬間 register 的值可以對照，只能看到最後執行的結果是錯的，所以花了非常多時間在 trace 到底是哪一行 code 開始錯的，雖然很費工夫，但也著實更會使用 ISim 也更了解整個 Simple CPU 的架構，算是有收穫了。