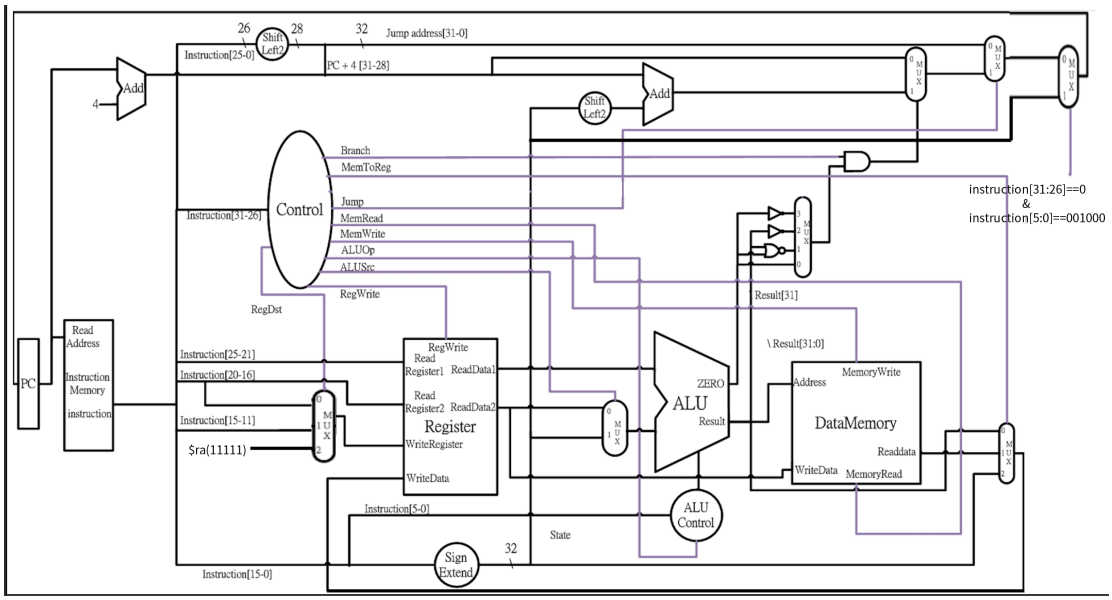
Computer Organization

0516013 吳泓寬

Architecture diagrams:



(改自講義附圖 有改動部分 MUX)

Hardware module analysis:

一開始時會把 instruction 全部放入 instruction memory 中，然後取出 PC 的值去 instruction memory 拿指令，接下來再由 decoder 去做第一層的解析算出 ALU_op 和一些作為 MUX 的 select 的值，而 R-format(ALU_op=3'b000)則須由 funct 的部分進一步求出，解析完後便可以由 ALU 計算出結果，基本上就是按照那些 control 的值(由 MUX 去選擇)去執行。

PC 的部分基本上變為三層 mux，第一層看是要走 branch 還是 pc+4，第二層看要不要 jump，最後一層則是 jr，決定完 pc_next 後再將 pc_next 賦值至 pc_now，然後重複執行至全部指令執行完。

而跟上個 lab 相異之處在於多出了一個 data memory，跟其相關的指令有 sw / lw，基本上都是由 ALU 算出 memory address 再去取值，然後這次寫回去 reg 的部分也因為增加了 lw 跟 jal 而有所調整

Finished part:

跟上次相異的地方在於使用 MUX_4to1 讓結構變簡潔些，還有 decoder 增加一些 control 去決定 pc 的選擇，而 MemToReg 也變為 2bits(配合 MUX_4to1)分別對應 alu_result(0)/mem_data(1)/pc_add4(2)[for jal] ，而 jump/jal 所需要的 address 使用

{pc[31:28],instruct[25:0],2'b0}達成。
而寫回 reg 的 address 則是用 MemToReg 和 RegDst 共同決定，如果是 jal 的話就固定寫回$ra(31)

Problems you met and solutions:

1. jr 是 R-format 真的蠻麻煩的，比起其他 R-format 的指令，他不需要 RegWrite 但卻要 jump，因此在 MUX select 上又要再做一次判斷。
2. 都是 0 的 instruction 一開始沒把 RegWrite 擋掉，讓一開始 R0 都是 x，可能是還沒 reset 前就做運算!?

Division of work: **(one-member teams don't need to write)**

Summary:

這次的作業雖然 code 不用打很多，但處理 jal / jr 這兩個指令真的算蠻複雜的，要去設計一下寫回 reg 跟 pc 的部分，不過寫完後感覺 CPU 漸漸成形了，蠻有成就感的。