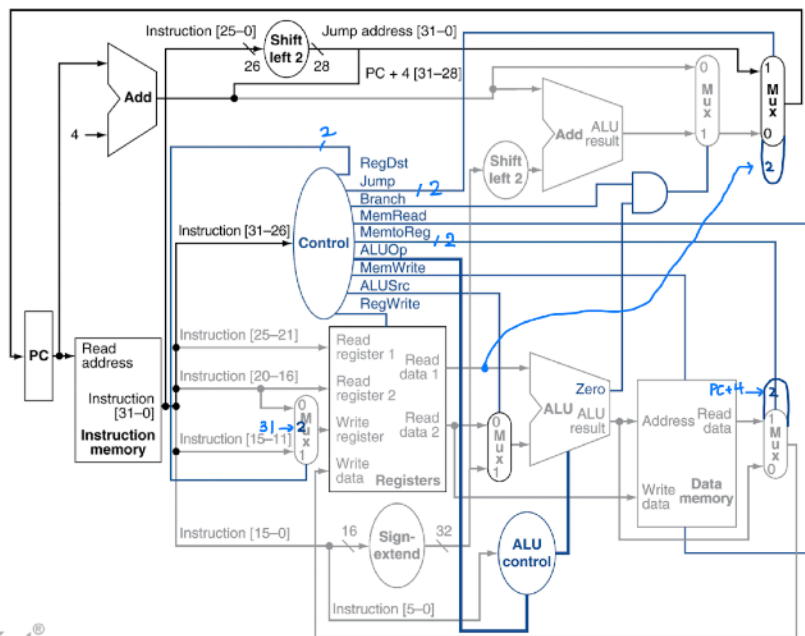


Computer Organization - Lab3

Single Cycle CPU Complete Edition

Architecture diagrams



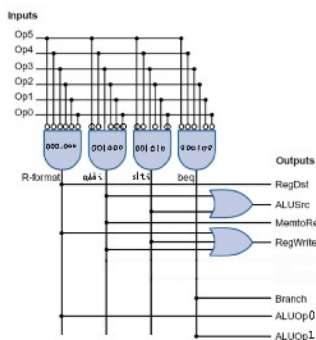
本次設計延續 Lab2 的電路，以及用課本上的擴增方式，做出如左圖。因為指令與課本上多了 jal 以及 jr，所以做了一些改變：

- Jump 1 -> 2 bits
- MemtoReg 1 -> 2 bits
- RegDst 1 -> 2 bits

針對上面控制訊號的 MUX，用了 MUX_4to1 module 作四選一（實際上只需要三選一），對於增加的 source 如圖。

Hardware Module Analysis

• Decoder



這次的 **Decoder** 我採用與上次相同的方法，不同的 **Instruction AND** 起來做區別，再將不同的控制訊號 **OR** 起來，達到正確的 **control** 訊號。

這次的 **jr** 為 **r-format** 的一種，所以我把 **Instra[5-0]** 也傳進來做 AND，來拿到指令訊號是不是 **jr**。

```
always @(*) begin
    RegDst_o[0] <= rfmt;
    RegDst_o[1] <= jpal;

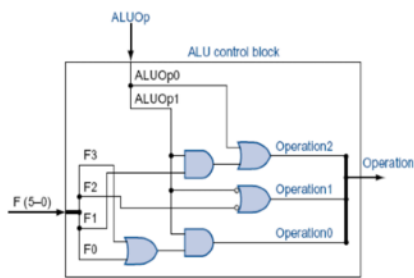
    RegWrite_o <= rfmt | addi | slti | lowd | jpal;
    Branch_o <= bieq;
    ALUSrc_o <= addi | slti | lowd | stwd;

    MemtoReg_o[0] <= lowd;
    MemtoReg_o[1] <= jpal;
    MemRead_o <= lowd;
    MemWrite_o <= stwd;

    Jump_o[0] <= jump | jpal;
    Jump_o[1] <= (jprt & rfmt);
end
```

Input or output	Signal name	R-format	lw	sw	beq	jal	jr
Inputs	Op5	0	1	1	0	0	0
	Op4	0	0	0	0	0	0
	Op3	0	0	1	0	0	0
	Op2	0	0	0	1	0	0
	Op1	0	1	1	0	1	0
	Op0	0	1	1	0	1	0
Outputs	RegDst	1	0	X	X	2	0
	ALUSrc	0	1	1	0	X	X
	MemtoReg	0	1	X	X	2	X
	RegWrite	1	1	0	0	1	X
	MemRead	0	1	0	0	0	1
	MemWrite	0	0	1	0	0	X
	Branch	0	0	0	1	X	X
	ALUOp1	1	0	0	0	X	X
	ALUOp2	0	0	0	1	X	X
Jump 0						1	1
Jump 1						0	1

• ALU control



ALU_Control 在這次的 Lab 沒有與 Lab2 的一模一樣，本次新增的 **lw sw** 都在上次所使用的 ALU control 一樣，不需做其他更動。

而 **j, jal, jr** 這三個指令不會用到 ALU control，所以不需更動，與 Lab2 相同。

• Shift_Left_Two_with_prv

這的 module 做的事情為算出 jump address，要將原本 PC + 4 的前 4 個位元，加上指令的 [25-0] shift left 2，所以這些事情都在這個 module 裡做完。

Result

Testcase 1

```
PC = x
Data Memory = 1, 2, 0, 0, 0, 0, 0, 0
Data Memory = 0, 0, 0, 0, 0, 0, 0, 0
Data Memory = 0, 0, 0, 0, 0, 0, 0, 0
Data Memory = 0, 0, 0, 0, 0, 0, 0, 0
Registers
R0 = 0, R1 = 1, R2 = 2, R3 = 3, R4 = 4, R5 = 5, R6 = 1, R7 = 2
R8 = 4, R9 = 2, R10 = 0, R11 = 0, R12 = 0, R13 = 0, R14 = 0, R15 = 0
R16 = 0, R17 = 0, R18 = 0, R19 = 0, R20 = 0, R21 = 0, R22 = 0, R23 = 0
R24 = 0, R25 = 0, R26 = 0, R27 = 0, R28 = 0, R29 = 128, R30 = 0, R31 = 0
```

Testcase 2

```
PC = x
Data Memory = 0, 0, 0, 0, 0, 0, 0, 0
Data Memory = 0, 0, 0, 0, 0, 0, 0, 0
Data Memory = 0, 0, 0, 0, 68, 2, 1, 68
Data Memory = 2, 1, 68, 4, 3, 16, 0, 0
Registers
R0 = 0, R1 = 0, R2 = 5, R3 = 0, R4 = 0, R5 = 0, R6 = 0, R7 = 0
R8 = 0, R9 = 1, R10 = 0, R11 = 0, R12 = 0, R13 = 0, R14 = 0, R15 = 0
R16 = 0, R17 = 0, R18 = 0, R19 = 0, R20 = 0, R21 = 0, R22 = 0, R23 = 0
R24 = 0, R25 = 0, R26 = 0, R27 = 0, R28 = 0, R29 = 128, R30 = 0, R31 = 16
```

PC 為 x 是因為 clock cycle 大於實際要執行的 clock cycle，所以會變成 X。

Summary

這是 Lab 我覺得有點難度，因為不是全部講義裡面找得到，要自己想要怎麼接線，還有要怎麼控制它。但是想到且做完後會非常有成就感，也會覺得自己比之前還要更了解 verilog 要如何寫，還有 Single Cycle CPU 的運作方式。

另外，我這次才發先有這條規定：

- Please **attach student IDs as comments** at the top of each file.

希望前面沒有被扣分～