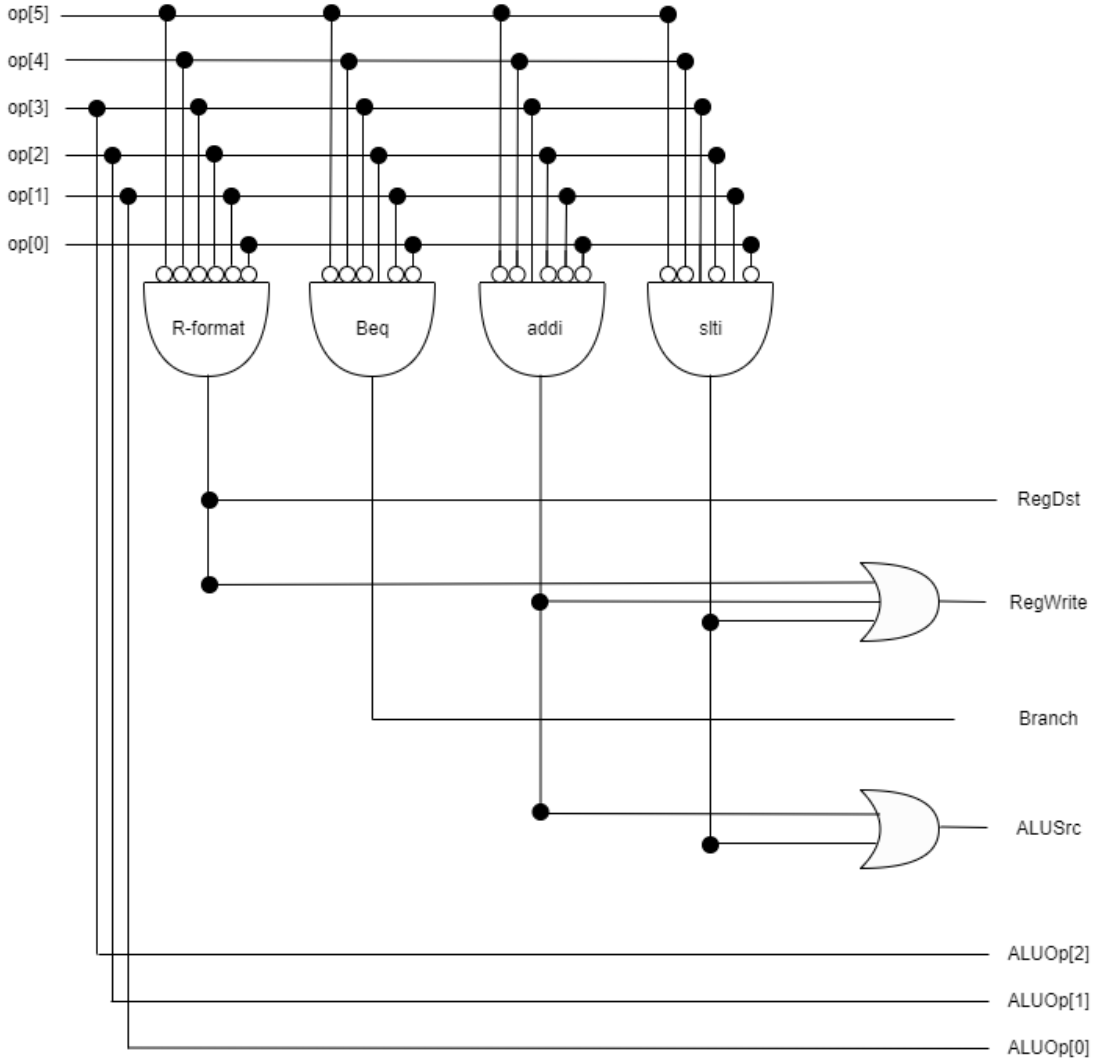
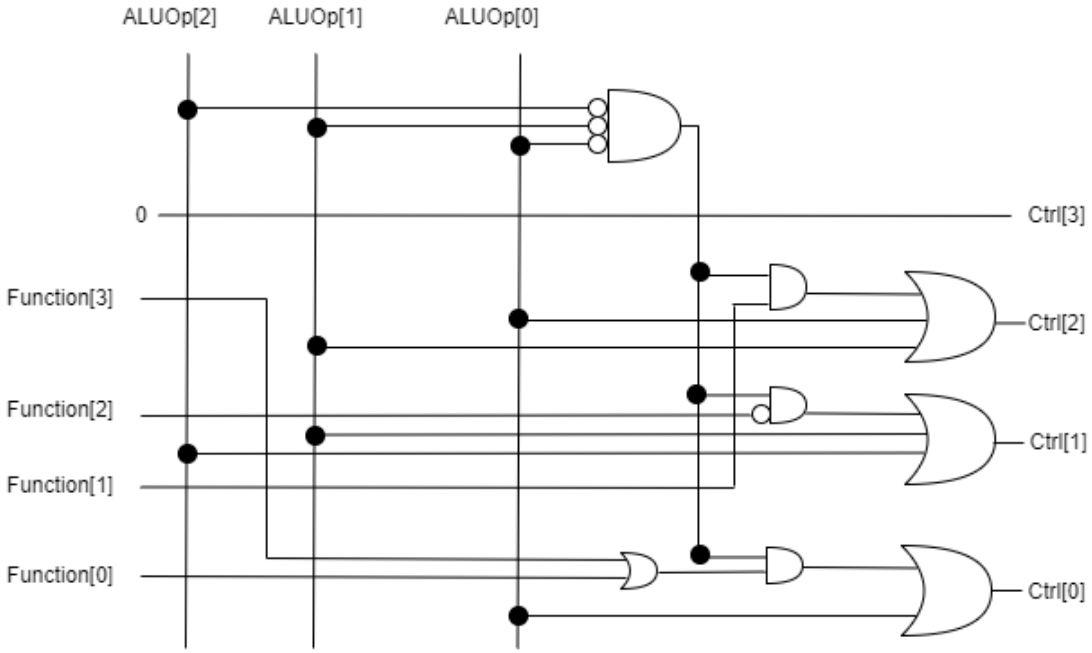
Computer Organization Lab2

Architecture diagrams:

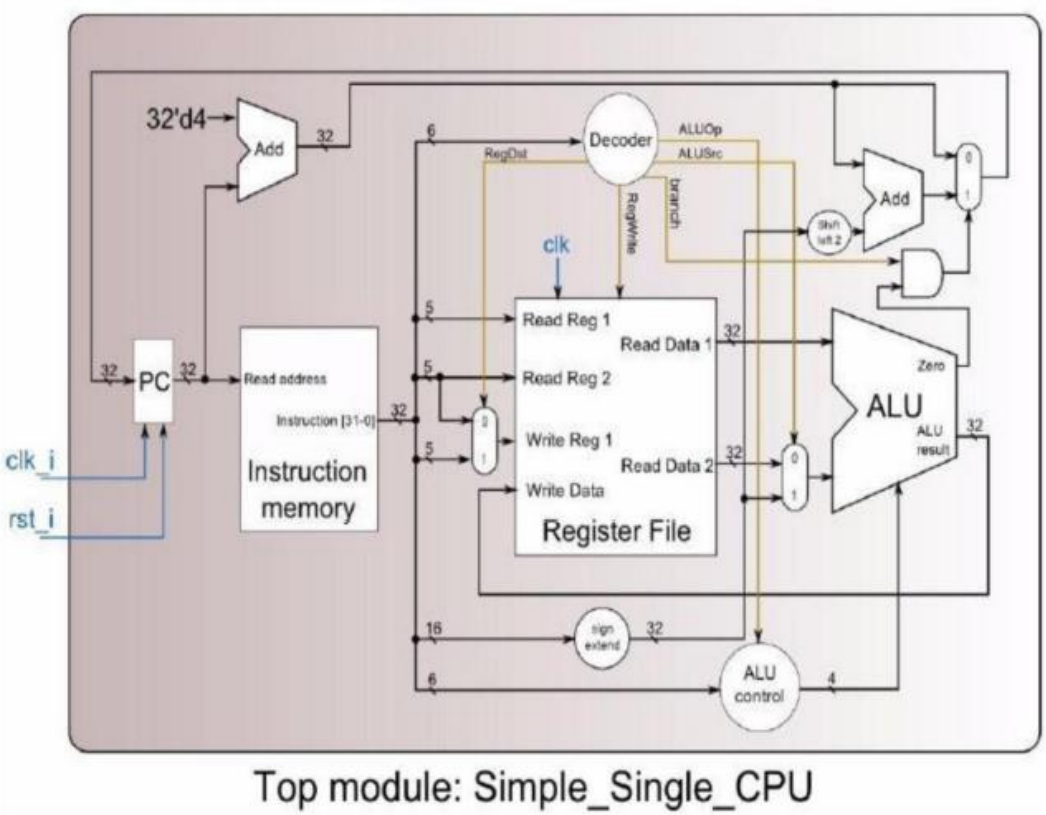
Decoder:



ALU Control:



CPU:



Hardware module analysis:

Decode: 直接用 OPcode 將不同種類的 Instruction 分開，這樣的設計會比較繁瑣，但當種類多的時候，設計起來簡單，不容易出錯。
將 ALUOp 設為 OP[3:1]，也是因為設計非常簡便，可以直接拉出原本的線路，不用額外設計邏輯電路。

ALUControl: 設計的原則是減少需要的線路，根據 truth table 可以將原本的電路一一簡化。

Finished part:

Testcase1:

r0=	0
r1=	10
r2=	4
r3=	0
r4=	0
r5=	6
r6=	0
r7=	0
r8=	0
r9=	0
r10=	0
r11=	0
r12=	0

Testcase2:

r0=	0
r1=	1
r2=	0
r3=	0
r4=	0
r5=	0
r6=	0
r7=	14
r8=	0
r9=	15
r10=	0
r11=	0
r12=	0

和老師所提供的 simulation 結果是相同的

Problems you met and solutions:

因為對於硬體語言比較不熟，有些寫法是錯誤的，找了很久才知道
在 sing extended 的 code 中，發現

```
Data_o[15:0] <= Data_i[15:0];
For(i=16; i<32; i=i+1)
Begin
    Data_o[i] <= Data_i[15];
End
```

這樣的方式，會使 data_o = X，
也連帶地影響到，在每一次的 instruction，ALU 的 src1=0、src2=X
後來改成

```
Assign Data_o = {{16{Data_i[15]}}, Data_i[15:0] };
```

Data_o 才變回正確的，ALU 也正常讀取 src1、src2

不過雖然解決了這個問題，卻還是沒有理解為什麼前者的寫法是錯誤的

Summary:

在這次的 lab 中發現了很多自己理解不夠透徹的地方，也比較熟悉 verilog 了