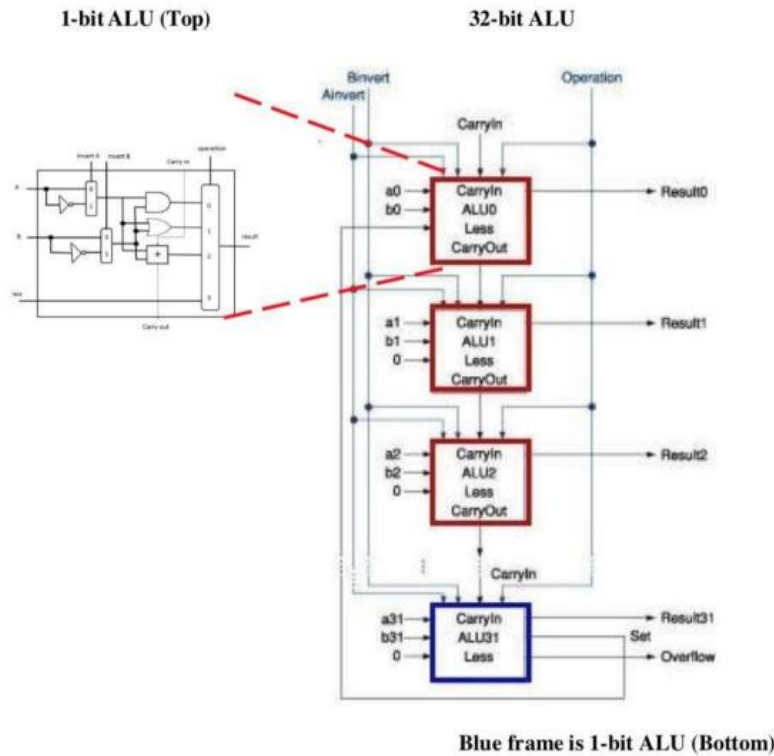
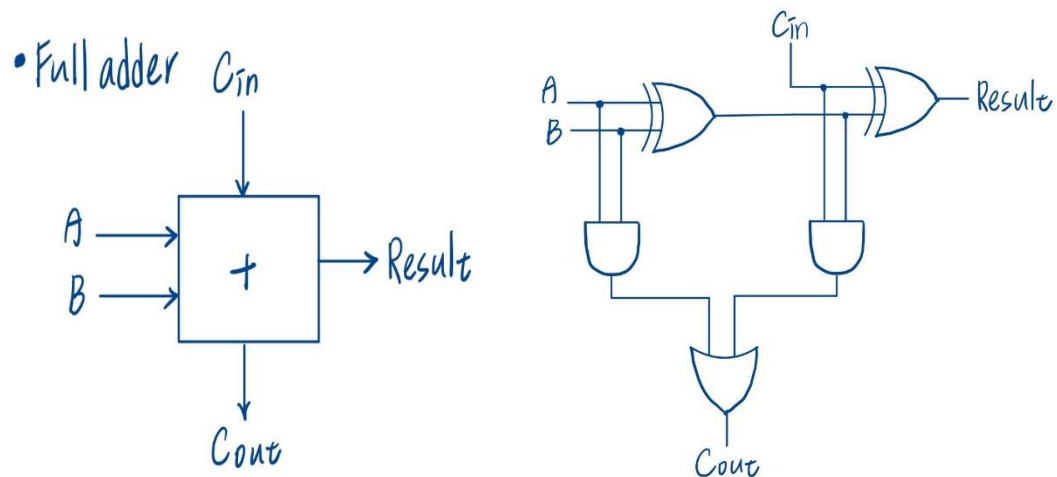


# Computer Organization

Architecture diagrams:



紅色部分使用 alu\_top module  
藍色部分使用 alu\_base module



實作此 32-bit ALU，將每一 bit 分別使用 1-bit ALU 處理。  
1-bit ALU 包含 AND、OR、ADD、SLT operation，整合成 32-bit ALU 後運用 ADD  
延伸出 SUB operation，AND 延伸出 NOR operation。  
總共使用了 31 個 alu\_top module、1 個 alu\_base module。

Module alu\_top 用於處理除了 MSB 的其餘 31 個 bit、  
Module alu\_base 用於處理 MSB、  
Module alu 用於將參數運算結果傳入 alu\_top 和 alu\_base 模組。

```

*****
|
| Congratulation! All data are correct!
|
| *****

```

| Name                    | Value                   | 0.000 ns  | 20.000 ns | 40.000 ns | 60.000 ns | 80.000 ns | 100.000 ns | 120.000 ns | 140.000 ns | 160.000 ns | 180.000 ns | 200.000 ns | 220.000 ns |  |
|-------------------------|-------------------------|---|-----------|-----------|-----------|-----------|------------|------------|------------|------------|------------|------------|------------|--|
| clk                     | 0                       |   |           |           |           |           |            |            |            |            |            |            |            |  |
| rst_n                   | 0                       |   |           |           |           |           |            |            |            |            |            |            |            |  |
| > src1_in[31:0]         | 00000000                | f... 3... f... 7... f... 0... X...                                      |           |           |           |           |            |            |            |            |            |            |            |  |
| > src2_in[31:0]         | 00000000                | 0... 0... 2... 0... 0... X...   |           |           |           |           |            |            |            |            |            |            |            |  |
| > operation_in[3:0]     | 0                       | 1 2 6 7 c X   |           |           |           |           |            |            |            |            |            |            |            |  |
| > mem_src1[0:23][7:0]   | 00,00,ff,ff,98,c3,13,31 | ff,ff,ff,ff,23,50,da,7e,ff,ff,ff,ff,00,00,00,00                         |           |           |           |           |            |            |            |            |            |            |            |  |
| > mem_src2[0:23][7:0]   | ff,ff,00,00,54,49,8e,08 | 01,00,00,00,e5,6a,c3,2e,01,00,00,00,00,00,00,00                         |           |           |           |           |            |            |            |            |            |            |            |  |
| > mem_opcode[0:5][7:0]  | 00,01,02,06,07,0c       | 00,01,02,06,07,0c   |           |           |           |           |            |            |            |            |            |            |            |  |
| > mem_result[0:23][7:0] | 00,00,00,00,dc,cb,9f    | 00,00,00,00,dc,cb,9f,39,00,00,00,00,3e,c5,16,50,01,00,00,00,ff,ff,ff,ff |           |           |           |           |            |            |            |            |            |            |            |  |
| > mem_zcv[0:5][7:0]     | 04,00,06,02,00,00       | 04,00,06,02,00,00   |           |           |           |           |            |            |            |            |            |            |            |  |
| > pattern_count[5:0]    | 00                      | 00 01 02 03 04 05 06 07   |           |           |           |           |            |            |            |            |            |            |            |  |
| start_check             | 0                       |   |           |           |           |           |            |            |            |            |            |            |            |  |
| > error_count[5:0]      | 00                      | 00  |           |           |           |           |            |            |            |            |            |            |            |  |
| > error_count_tmp[5:0]  | 00                      | 00  |           |           |           |           |            |            |            |            |            |            |            |  |
| > result_out[31:0]      | XXXXXXXX                | XXXXXXXX 00000000 3... 0... 5... 0... f...                              |           |           |           |           |            |            |            |            |            |            |            |  |

1.  
在 code 方面，首先是 reg 和 wire 的設置，在使用 wire 時才可以用 assign，因為忽略了這點我在 run simulation 時一直接收到 " concurrent assignment to a non-net 'result' is not permitted" 這樣的報錯。
2.  
在觀念理解上面，carryout 和 overflow 使我搞混，最後理解 carryout 是使用於 unsigned，而 overflow 是使用於 signed。
3.  
在使用 vivado 執行程式上面我遇到我最大的困難，常常在重啟程式之後要點開 source 的.v 檔，又或者只是在 add source file，vivado 會直接跑很久然後就卡在那邊。關於解決辦法，我透過網路也無從得知如何從根本解決這個問題，而我只能不斷嘗試將電腦重開機又或者不斷重新 create project。  
(ps.請問助教知道這類問題如何解決嗎?)

## Summary:

經過這次 lab 我更了解 ALU 的細節運作是如何進行的，verilog 硬體語言對我而言屬於較少接觸的語言，透過實作部分需要上網搜尋相關語法，我對 verilog 也更熟悉了。