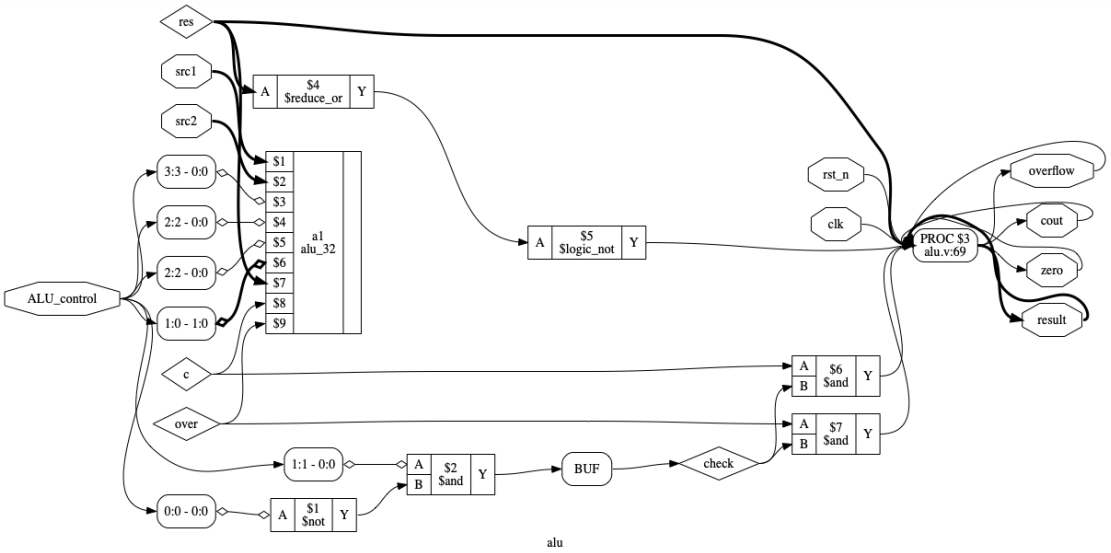
Computer Organization

Architecture diagrams:

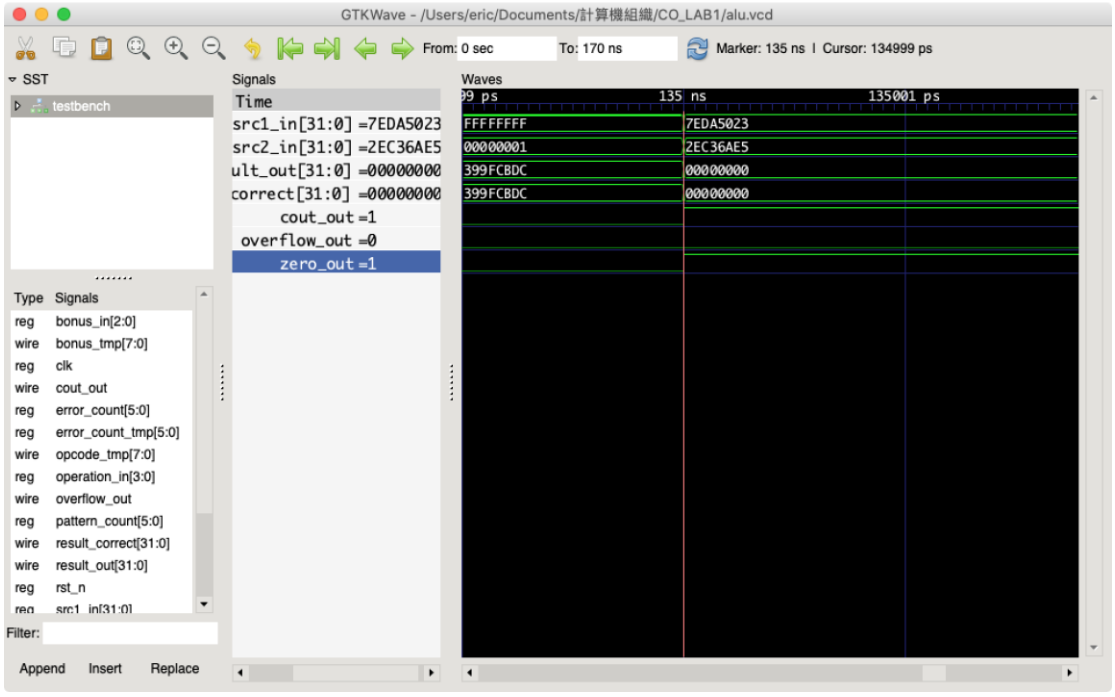


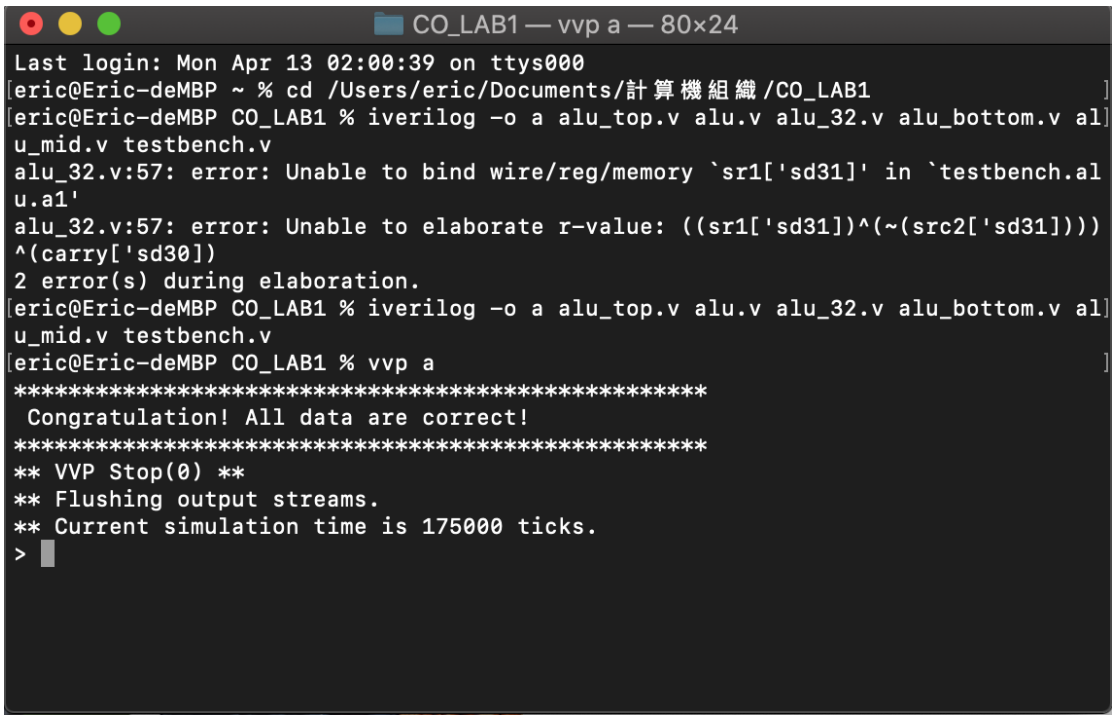
Hardware module analysis:

第1個bit是alu_top.v 之後中間2~31個bit 是alu_mid.v 最後第32個bit是alu_bottom.v，最後用alu_32.v宣告上述這三個module，其中alu_mid.v會重複宣告30次，最後才是用alu.v宣告alu_32.v。然後add和sub的部分適用ripple adder的方式去實作的，而sub就是a+(b的complement)+1，而or和and就是直接寫｜和&，而nor就是a和b的complement and，nand是a和b的complement or，最後slt則是讓第2~32的bit的輸出都為0，並做a-b的結果看正的還是負的，這個部分看第32個bit的result是0或1就好，是1就代表是負的也代表slt的結果是1，是0

就代表是正的也代表 slt 的結果是 0，因此只需要把 a-b 的結果接到第 1 個 bit 的 result 就好，並稱為 set。

Experiment result:





Problems you met and solutions:

在最後結果的時候我的加減法的 result 一直是亂碼，後來發現在 alu_top 等等一個 bit 的 module 裡的 always block 的觸發條件少了 cin，之後我將觸發條件改成＊就順利的解決這個問題了。還有就是在 slt 的判斷上，一開始很苦惱要如何在直接 slt 這個 operation 的時候執行減法，後來發現我的每個 bit 的 cin cout 都是一直在運作的，所以我只要將第 32 個 bit 的兩個運算 bit 和 cin 做運算便可得知減法的結果，之後再將這個結果用成 set 的判斷就可以了。

Summary:

總結來說這次的作業一開始有點困難，但只要有了一個起頭之後就不會這麼的困難，主要是 debug 不像 c++等語言如此方便，因此讓我有點苦惱，還有其實有很多種寫法，而我使用的是滿直觀的一種寫法，就是去實作每一個 bit 最後再用一個 module 全部宣告出來，這樣的方法雖然耗時但我覺得比較好 debug 一點點，還有因為很久沒打 verilog 了，在開始打之前也上網複習了很多 verilog 的語法，因此這次的作業也讓我找回了打 verilog 的感覺，十分的有幫助。