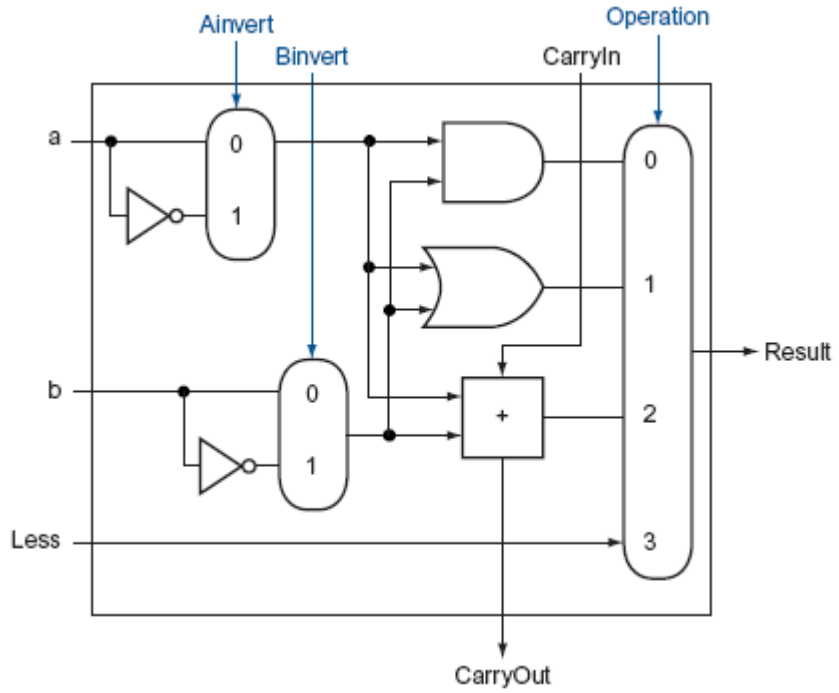
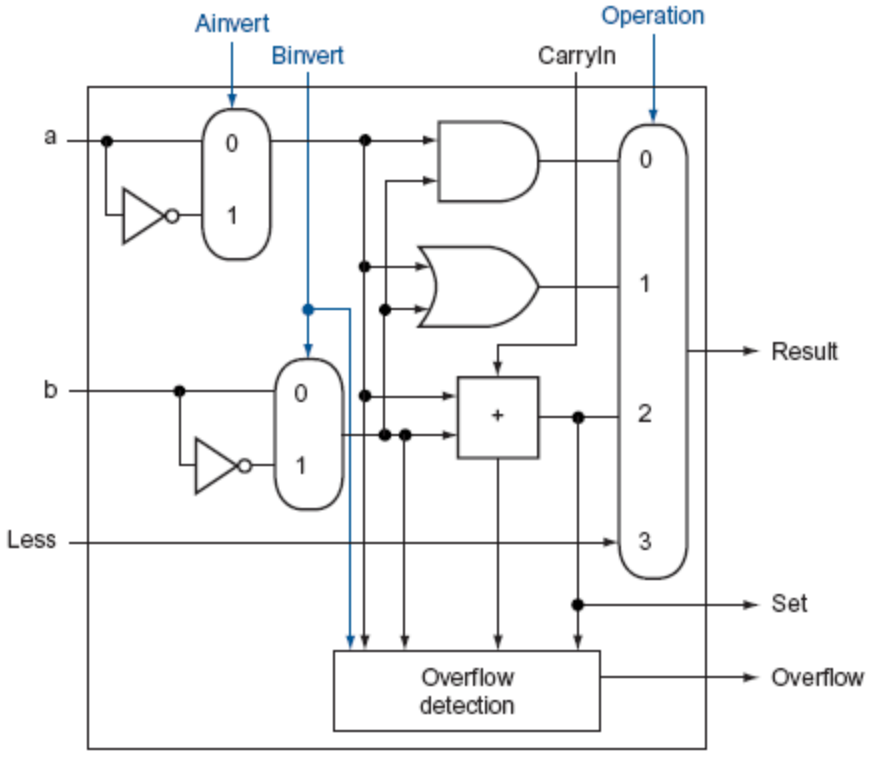
Computer Organization

Architecture diagrams:

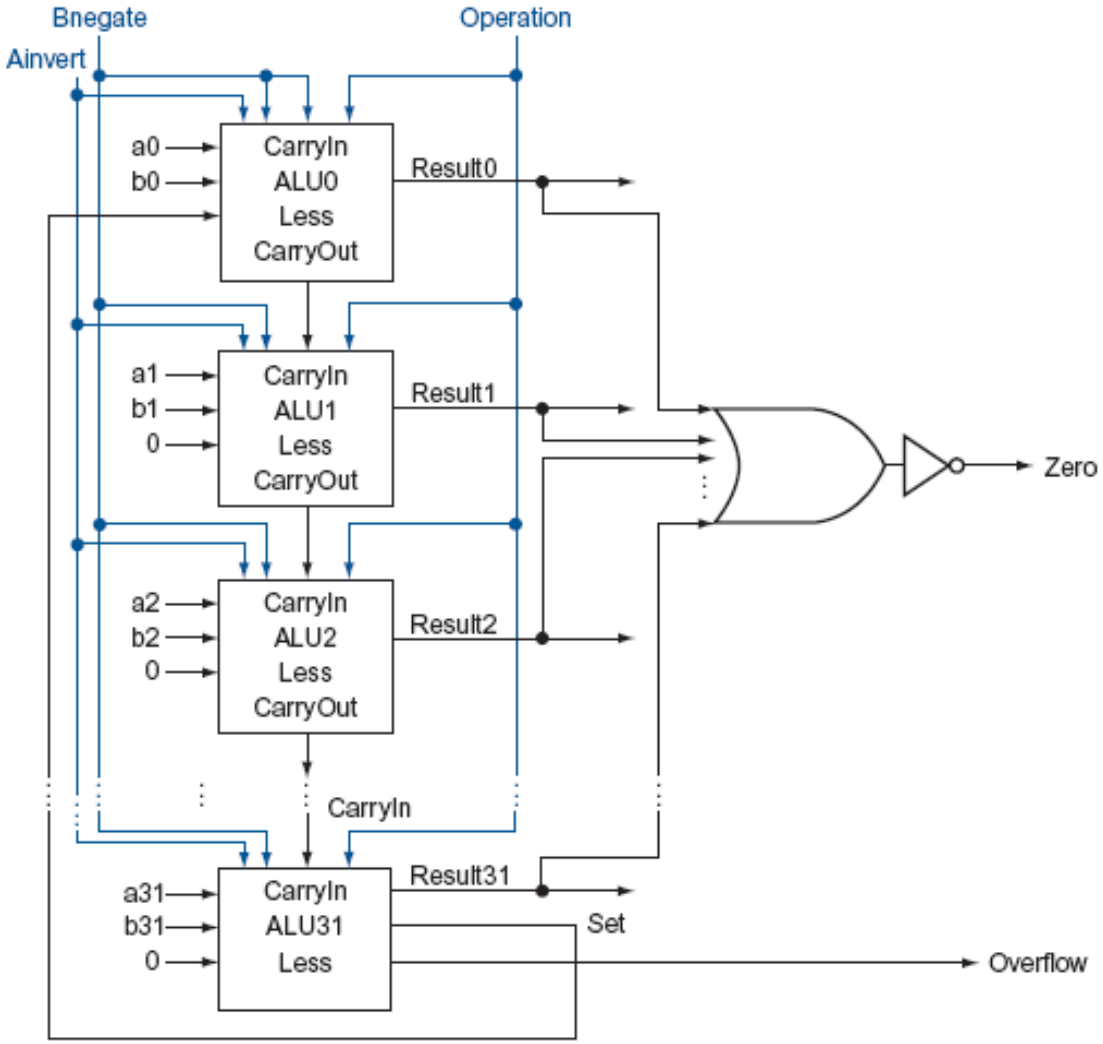
alu_top.v



alu_msb.v



alu.v



Hardware module analysis:

硬體架構：

- module alu(clk, rst_n, src1, src2, ALU_control, result, zero, cout, overflow)
  - module alu_top(src1, src2, less, A_invert, B_invert, cin, operation, result, cout)
  - module alu_msb(src1, src2, less, A_invert, B_invert, cin, operation, result, cout, set, overflow)

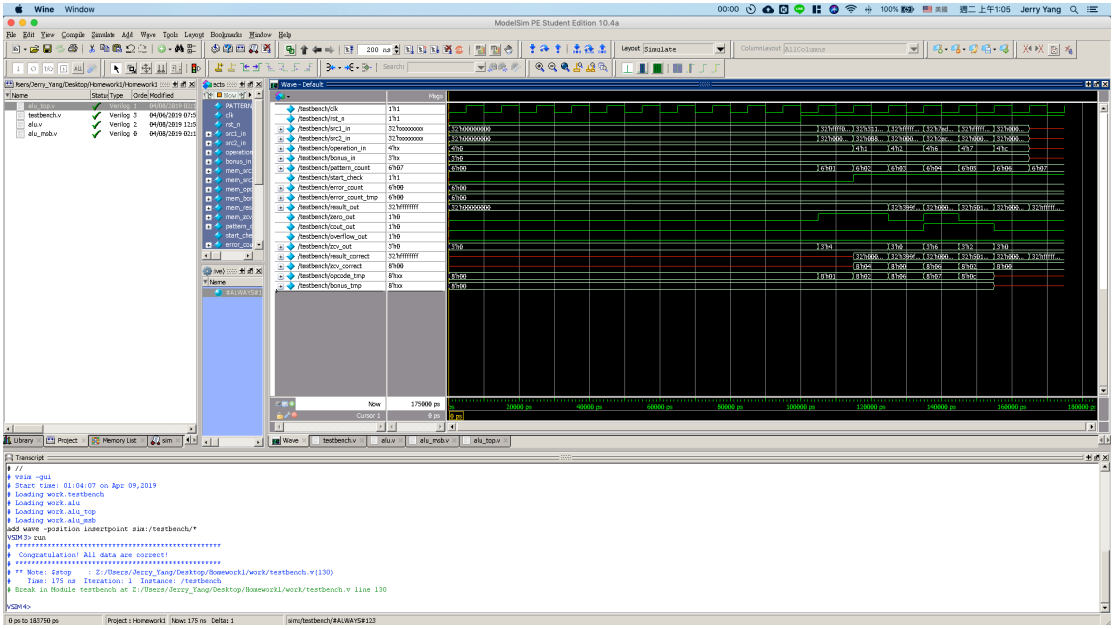
訊號分析：

- alu.v： alu **主要的架構**
  - clk: system clock(input)
  - rst_n: negative reset(input)

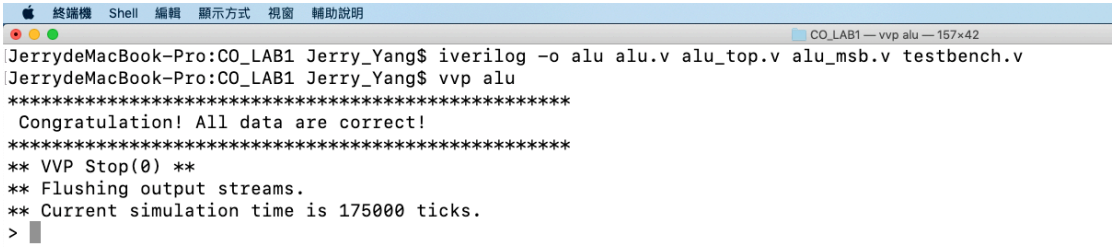
- src1: 32 bits source 1(input)
- src2: 32 bits source 2(input)
- ALU_control: 4 bits ALU control input(input)
- result: 32 bits result(output)
- zero: 1 bit when the output is 0, zero must be set(output)
- cout: 1 bit carry out(output)
- overflow: 1 bit overflow(output)
- set: 1 bit set for less(wire)
- carry: 31 bit carry in and carry out between alu_top and alu_msb(wire)

- alu_top.v： alu 前 31 個 bit 的基本運算單位
  - src1: 1 bit source 1(input)
  - src2: 1 bit source 2(input)
  - less: 1 bit less (input)
  - A_invert: 1 bit A_invert(input)
  - B_invert: 1 bit B_invert(input)
  - cin: 1 bit carry in(input)
  - operation: operation(input)
  - result: 1 bit result(output)
  - cout: 1 bit carry out(output)
  - a_out: src1 經過 A_invert 後的訊號(wire)
  - b_out: src2 經過 B_invert 後的訊號(wire)
- alu_msb.v： alu MSB(第 32 bit)的基本運算單位
  - 包含 alu_top 含有的全部東西，以及兩個額外的 output
  - set: 1 bit set(output)
  - overflow: 1 bit overflow(output)

Experiment result:



使用 ModelSim 的結果



使用 iverilog 和 vvp 的結果

Problems you met and solutions:

到快完成之前，我有遇到一個問題是：我寫的全加器在遇到 a = 1, b = 1, cin = 1 這種情況時，會輸出 s = 0, cout = 1 的錯誤結果（但其他 7 種情況都是正確的）。我實在百思不得其解，後來才發現只要把原本 alu_top.v 中的 always@(src1_temp or src2_temp or operation)改成 always@(*)就解決了。我想是因為在運算過程中，有時候 src1_temp、src2_temp、operation 都沒有被更新，always 底下的東西也就不會被執行到了，才造成了這個錯誤。

另一個我遇到的問題是我一開始很不會用 ModelSim，後來無意間發現了有 iverilog 和 vvp 這兩個工具可以使用，雖然功能沒有 ModelSim 這麼強大，但執行起來速度很快，一下就可以看到結果，也許未來想測試某個小東西時，也不失是一個好選擇。

Summary:

alu.v 是由 31 個 alu_top.v 以及 1 個 alu_msb.v 所組成，其中 ALU_control 的最

高位跟第二高位分別對應到要傳進每個小模組的 A_invert 以及 B_invert，而剩下的兩位則是對應 operation，我想這應該是設計好的。
之所以多設計了一個不一樣的 alu_msb.v，有兩個原因：一是因為最高位需要多兩個輸出，set 以及 overflow，二是因為整個 aiu 的 cout 以及 overflow 是由 alu_msb 提供，需要額外的設定。
這個作業是我第一次寫 verilog，因為我之前是修電機系的數位邏輯，從來沒有寫過這類東西，所以有很多不會的地方，花了超多時間才完成，要感謝我室友耐心教我 verilog。這次的作業讓我更加瞭解 ALU 運作的機制、內部各類訊號的傳遞及用途，以及怎麼從組合語言中看到的 and、or、add、sub、nor、slt 轉換成 4 位元的 ALU_control。此外，ALU_control 的前兩位對應到控制 A_invert 跟 B_invert 的巧思，實在是讓我覺得太聰明了。