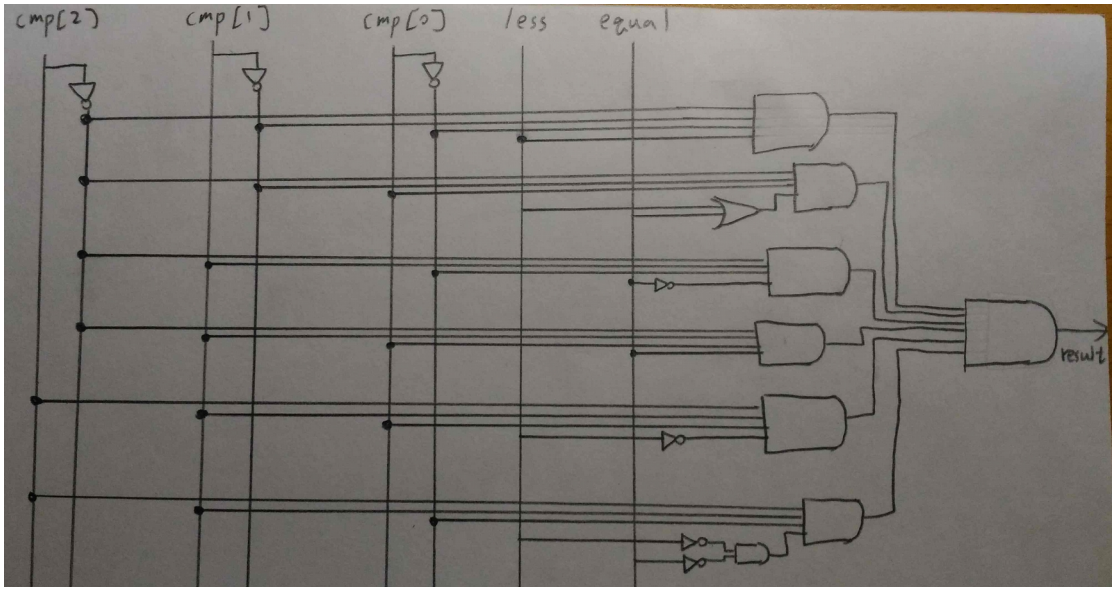
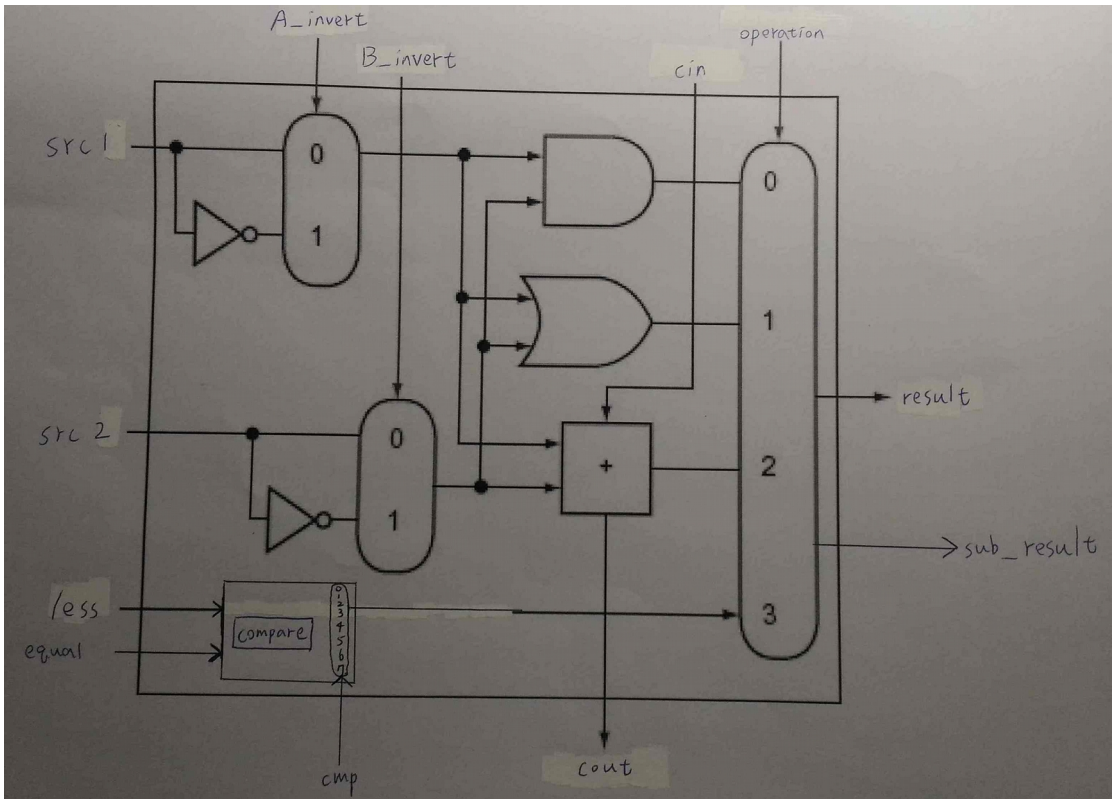
Computer Organization

Architecture diagram:

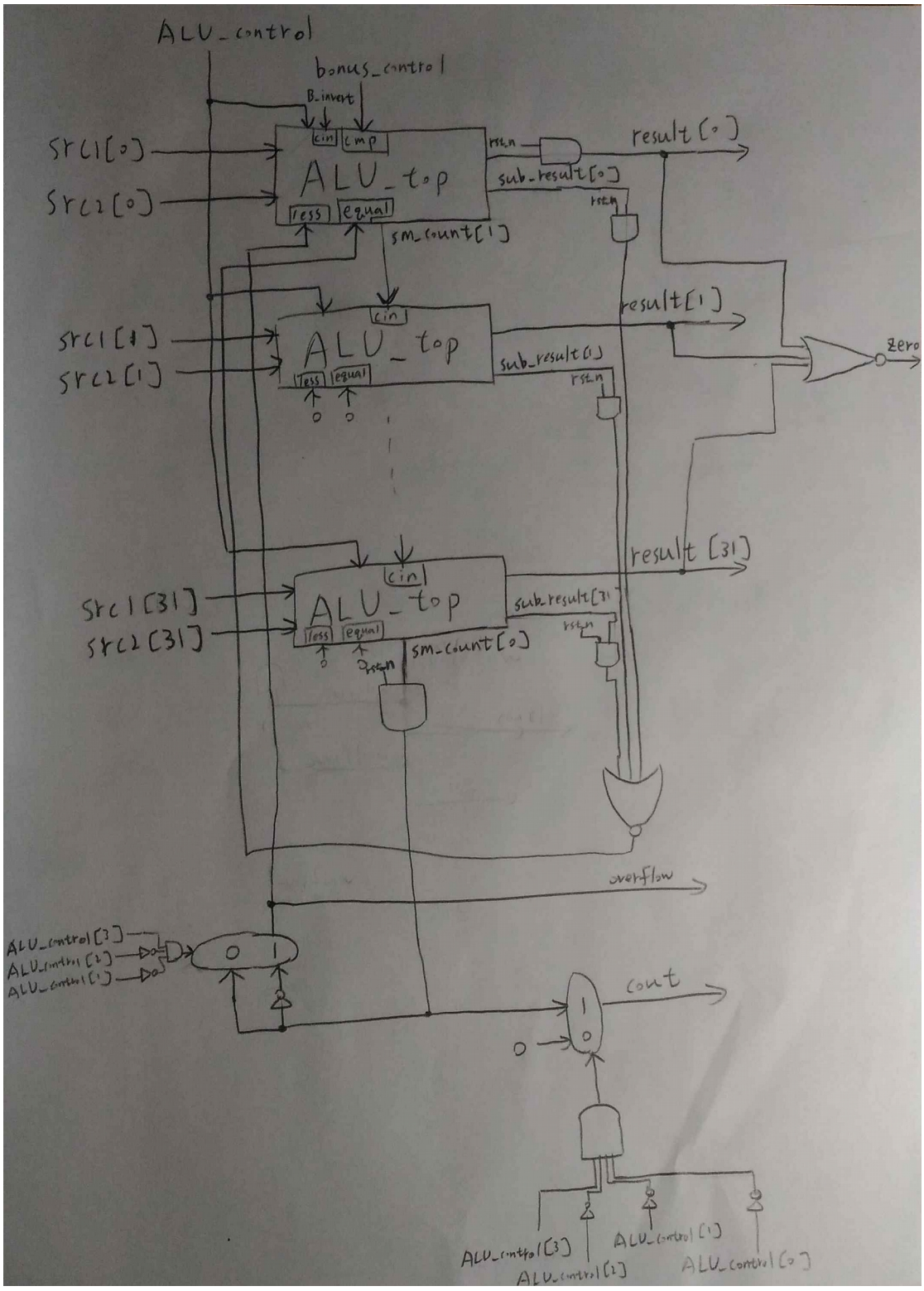
compare



ALU_top



ALU


ALU_control
bonus_control
B_invert
cin
cmp
ALU_top
less
equal
src1[0]
src2[0]
rst_n
result[0]
sub_result[0]
sm_count[1]
result[1]
src1[1]
src2[1]
sub_result[1]
zero
result[31]
src1[31]
src2[31]
sub_result[31]
sm_count[0]
overflow
ALU_control[3]
ALU_control[2]
ALU_control[1]
cout
ALU_control[0]

Detailed description of the implementation:

沒有 bonus 的部份就是依照課堂中教的東西實踐出來，比較特別的是電路圖中，因為多加了 rst_n 的判斷，因此在電路圖中幾乎所有的輸出都 and 了 rst_n，讓只有在 rst_n=1 時輸出才有機會不等於 0

而 bonus 的部份，首先是先多加了 compare 這個 module，可以根據不同的 bonus_code 來利用 less 和 equal 決定 output，而 less 就是接上原本的 less，equal 則是在 ALU_top 裡多輸出了 sub_result，代表相減時的結果輸出，最後在 ALU 裡判斷這個 sub_result 全部位元是否皆是 0 來作為 equal 的輸入

Problems encountered and solutions:

第一個遇到的問題當然就是 verilog 不熟悉，這個在看了網路資源後解決了

第二個是在判斷 unsigned int overflow 的部份，想了一段時間，才想到負數的 overflow 跟正數的 overflow 判斷方式剛好相反（一個是最後一個 ALU_top cout 為 1 時 overflow，另一個反而就是要 cout=1 才不是 overflow）

第三個問題是在判定兩數是不是 equal 的情況，一開始是直接將 zero 接到 equal 那邊，但這樣的結果會不停止的振盪。因為如果一開始的 result=0，zero 就會是 1，接回來 equal 之後，因為符合了 equal 的情況，result 就會變成 1，而 zero 就變成 0 了，這樣 result 又會不符合，因此一直來回振盪。最後因為想不到其他方法，只好再接一個 sub_result 代表相減的結果，直接利用此來判斷是否相等當作 equal 輸入才解決了問題

Lesson learnt (if any):

一、對於 ALU 與其內部的結構實際了解了一遍
二、overflow 的判斷方法
三、學會寫 verilog