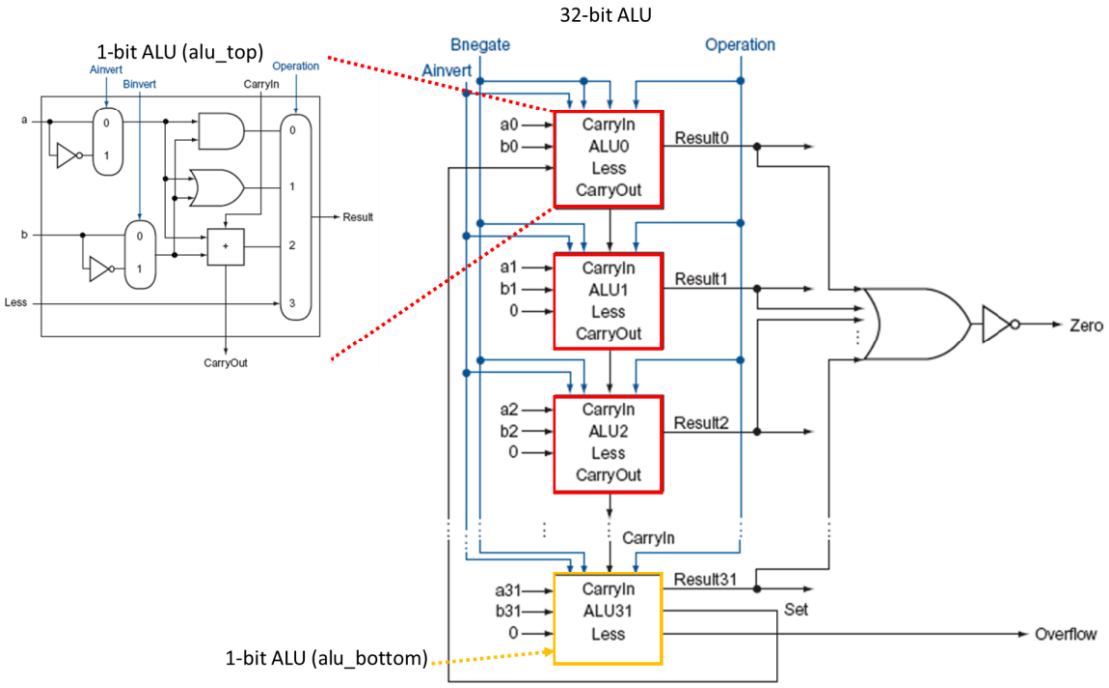
計算機組織 HW1

0416081 趙賀笙
0416025 呂翊愷

Architecture diagram:



Description of Implementation:

1. alu_top.v

   首先，透過兩個 Mux 決定輸入的 source1, 2 要不要 invert，

   接著再從輸入的 operation 判斷要執行甚麼運算。

   如果 operation 是 00，表示 A 和 B 要做 AND，所以 result 是 A&B，cout 是 0。

   如果 operation 是 01，表示 A 和 B 要做 OR，所以 result 是 A|B，cout 是 0。

   如果 operation 是 10，表示 A 和 B 要相加，所以 result 是 A 和 B 和 Cin 做 XOR，cout 是 A, B, Cin 兩兩做 AND 再 OR 起來。

   如果 operation 是 11，表示 A 和 B 的大小判斷，若 A>B 輸出 1，否則輸出 0。大小比較的方法是將兩數相減，若減完的結果是負的，則可知 A 比 B 小。判斷的方法是檢查兩數相減後的 sign bit，如果是 1 則代表是 A less than B，故輸出 1。

2. alu.v

首先，透過 ALU_control 判斷要執行甚麼運算。

如果 ALU_control 是 0000，表示 src_1, src_2 要做 AND，因此讓傳入 alu_1bit 的 operation 是 2’ b00，做 AND 運算，A 和 B 不 invert。

如果 ALU_control 是 0001，表示 src_1, src_2 要做 OR，因此讓傳入 alu_1bit 的 operation 是 2’ b01，做 OR 運算，A 和 B 不 invert。

如果 ALU_control 是 0010，表示 src_1, src_2 要相加，因此讓傳入 alu_1bit 的 operation 是 2’ b10，A 和 B 不 invert，傳入 Full_Adder，加上 CarryIn 一起計算。

如果 ALU_control 是 0110，表示 src_1, src_2 要相減，因此讓傳入 alu_1bit 的 operation 是 2’b10，但是 B 要 invert，利用 A-B=A+B 的 2 complement+1，傳入 Full_Adder，利用最低位 CarryIn 傳入 1 當作+1，一起計算，最後得到相減結果。

如果 ALU_control 是 1100，表示 src_1, src_2 要做 NOR，因為~(src1 | src2) = ~src1 & ~src2，因此讓傳入 alu_1bit 的 operation 是 2’ b00(AND)，做 AND 運算，A 和 B 皆 invert。

如果 ALU_control 是 1101，表示 src_1, src_2 要做 NAND，因為~(src1 & src2) = ~src1 | ~src2，因此讓傳入 alu_1bit 的 operation 是 2’ b01(OR)，做 OR 運算，A 和 B 皆 invert。

如果 ALU_control 是 0111，表示 src_1, src_2 要比大小，若 src1<src2 輸出 1，否則輸出 0。因為必須一起考慮 32 個 bit，做法是把所有最低位以外的 less 設成 0 直接傳出，接著把最後用來判斷相減結果的最高位 carry out，傳回最低位的 less，若減完的結果是負的，則因為最低位的 less 也變成 1，所以整體輸出也是 1。

zero: 如果 result==0，zero 就等於 1，否則 zero 是 0。

overflow: 把最高兩位的 cout 做 XOR，如果是 1 則 overflow 是 1，否則就是 0。

cout: 如果最高位的 cout 是 1，則 cout 是 1，否則就是 0。

心得：

一開始我們花了一些時間才看懂 Architecture diagram，因為一直不懂

A_invert 和 B_invert 的用途是甚麼。好不容易將 alu_top 實作出來後，又不知道如何將 set less than 實作出來，後來得知將 cout 接到第一個 bit 的 less 就能判斷大小。最後遇到的問題是不知道如何判斷 overflow，後來我們上網搜尋才發現原來將最高兩位的 cout 做 XOR 就好。此份作業最大的收穫大概是除了因此比較了解 ALU 的運作原理之外，還讓我們重新複習了 verilog 的語法，重新找回 verilog 的邏輯思維。