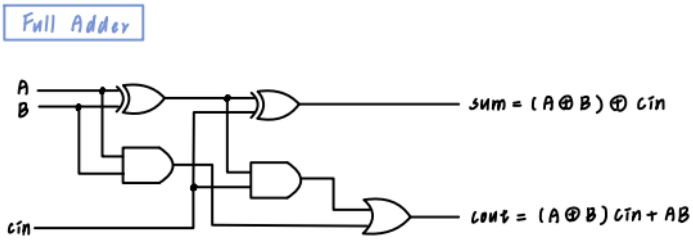
Computer Organization Lab1

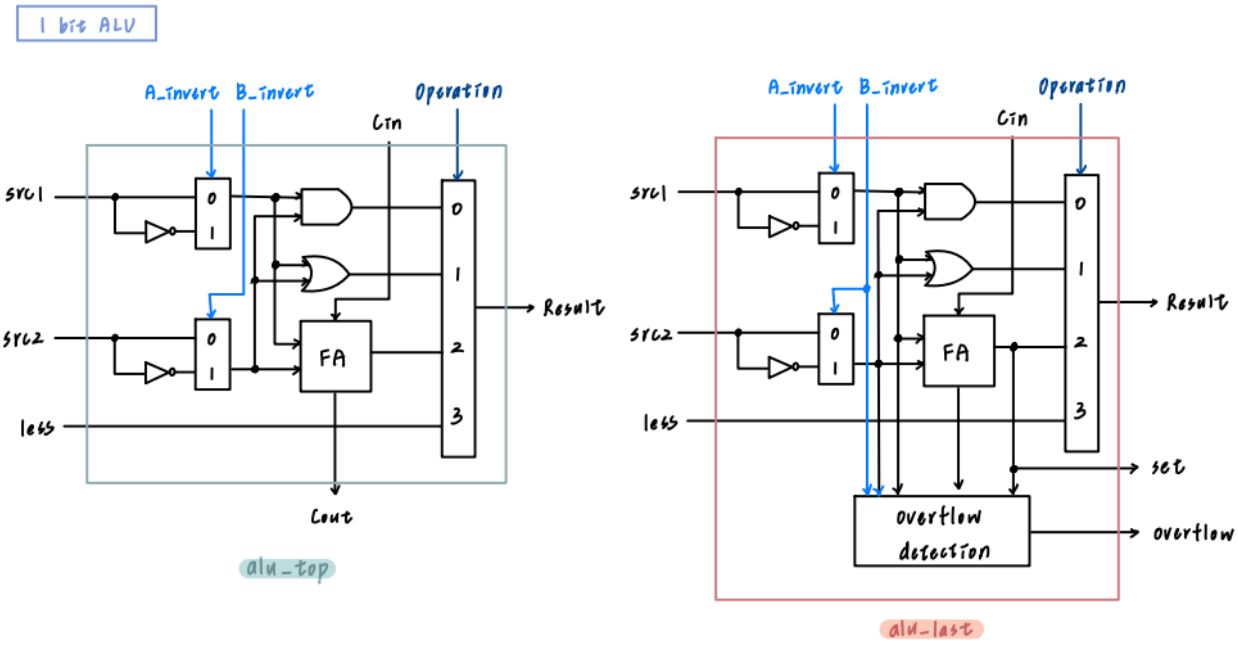
運管11 0713216 龔祐萱

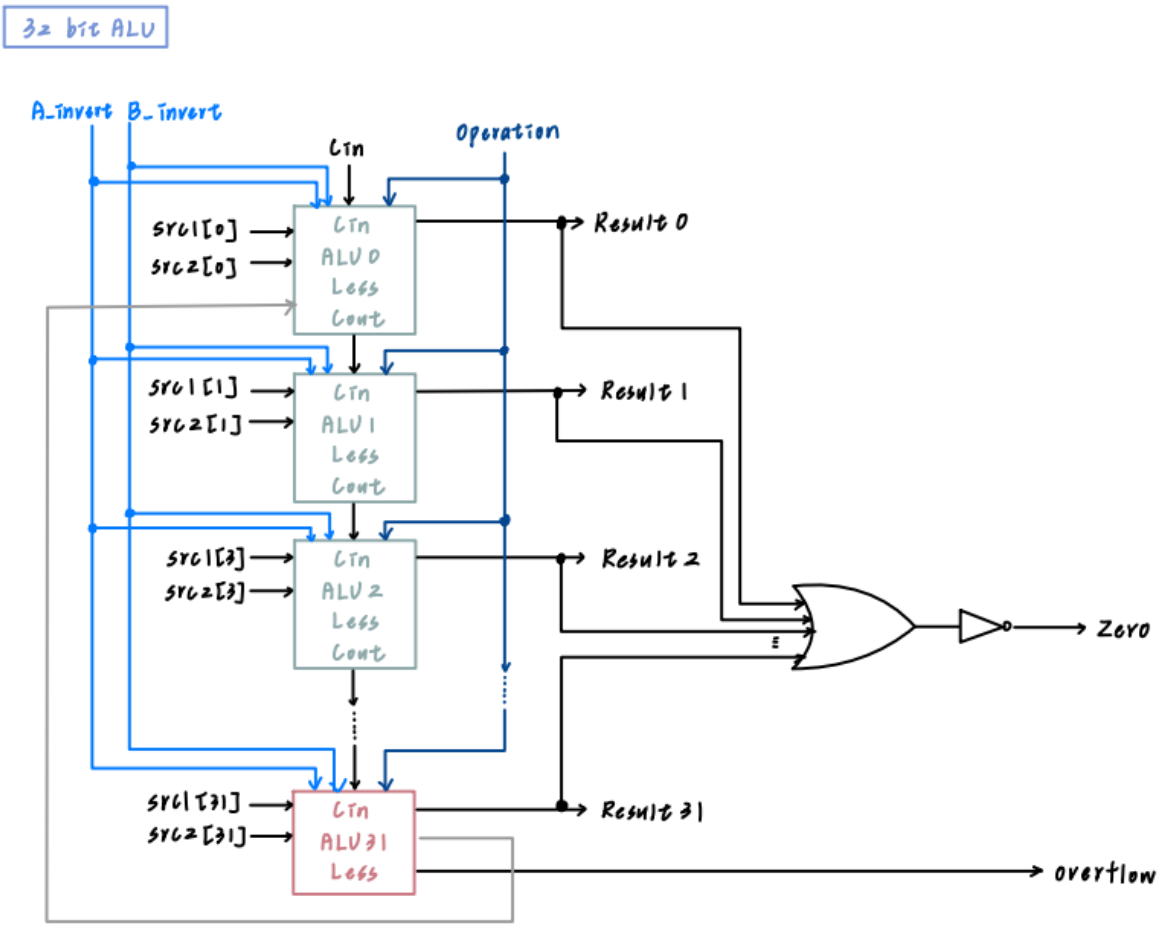
1. Architecture diagrams:

下圖為組成 32-bit ALU 的元件，包括 full adder 以及 1-bit ALU。其中 1-bit ALU 分成兩種，一種為針對非 MSB 的 ALU (alu_top)，另一種為針對 MSB 的 ALU (alu_last)，兩種 ALU 皆包含 AND gate、OR gate 以及 full adder 並搭配 operation 以及 A_invert 和 B_invert 來進行不同的運算處理。

而 32-bit ALU 則是使用 31 個針對非 MSB 的 ALU (alu_top)以及 1 個針對 MSB 的 ALU (alu_last)組合而成的。







2. Hardware module analysis:

32-bit 的 ALU 是由 32 的 1-bit ALU 所組成，每個 1-bit ALU 中皆包含一個 AND gate、一個 OR gate 以及一個 full adder，並使用 operation 來決定要做什麼運算。如果要實作 AND、OR 以及 NOR 的話，就可以使用 AND gate 和 OR gate 完成，若是要實作 Add、Sub 或 Slt 的話可以藉由 full adder 來進行實作。

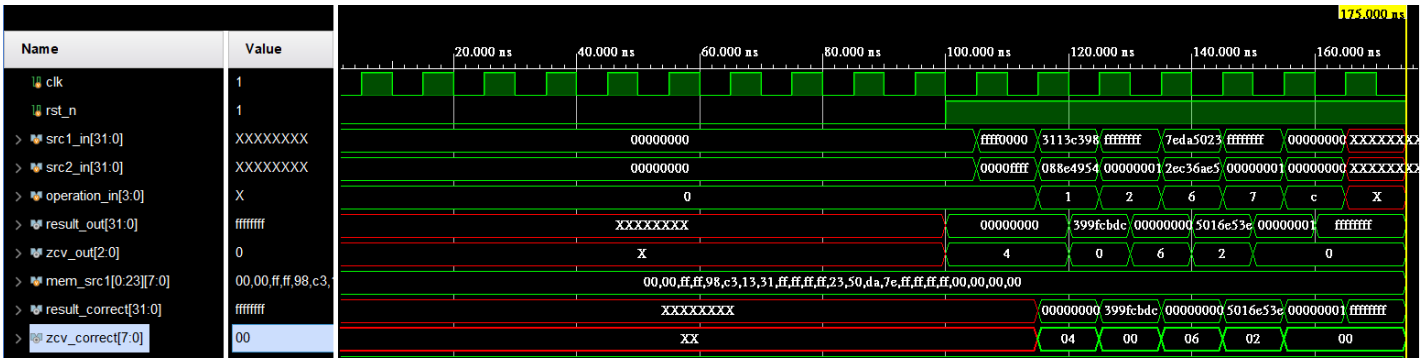
因每個小單位的 ALU 的架構大致相同，因此在實作上可以使用多個小單位的 ALU 去組成大單位的 ALU，使得在設計硬體電路以及生產操作上會較簡單以及具有規律性，可以降低生產成本並提升效能，或者去組合成更複雜的電路。

反之，因每個 1-bit ALU 的 carry out 都會是下一個 1-bit ALU 的 carry in，因此會造成 ALU 有 ripple 的問題，下一個 ALU 需要等待前一個 ALU 完成運算後才能進行運算，在實作上會使得整體的 ALU 運算較沒有效率。

3. Experiment result:

下圖為執行測資的波形圖，其中測資以及輸出結果如下:

operation		src1	src2	result	zcv
0	AND	ffff0000	0000ffff	00000000	4
1	OR	3113c398	088e4954	399fcbdc	0
2	ADD	ffffffff	00000001	00000000	6
6	SUB	7eda5023	2ec36ae5	5016e53e	2
7	SLT	ffffffff	00000001	00000001	0
C	NOR	00000000	00000000	ffffffff	0

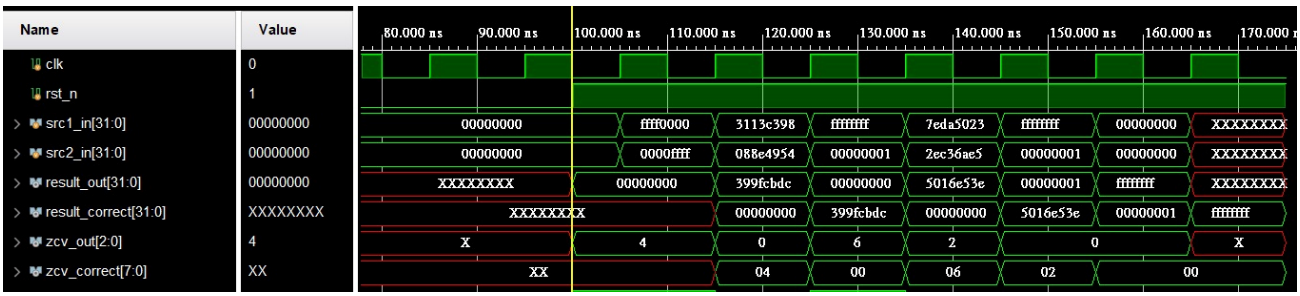


從上述測資以及波形圖輸出結果(result_out 以及 zcv_out)可知皆符合與測資結果(result_correct 以及 zcv_correct)相同，因此符合 ALU 各運算應輸出的結果。

4. Problems you met and solutions:

在實作 ALU slt 運算時比較不清楚要怎麼設定第 32bit 的 ALU，因此就多寫了一個 alu_last 的 module 來實作，裡面的結構大致與 alu_top 相同，但在實作 slt 時 alu_last 的 carry out 會設定為 0，並且會輸出 set 拉到 bit 0 ALU 的 less 中。起初，我對於怎麼設定 set 有點沒頭緒，但參考了老師的講義以及上網找了一些關於 ALU 設計的資料後，發現 set 就是兩個數值相加後結果的最高位 bit，因此我就使用加法的邏輯表達式來得到 set (set = src1[31] ^ ~src2[31] ^ carry[31]) ，並將其輸入至 bit 0 的 ALU 中來完成 32-bit ALU 的實作。

另一個問題就是一開始在設計 ALU 時，我並沒有考慮到 clock 的和 ALU 之間運作上的關係，所以當 output 結果皆為正確數值時無法得到” Congratulation! All data are correct!”的字樣，查看波形圖後發現 ALU 輸出的結果都會比 result_correct 輸出的結果早(如下圖)，因此重新將 ALU 進行調整，並設定 ALU 在 clock 0->1 時才會將 input 輸入至 ALU 中。



這樣的調整雖然解決了 ALU 的 output 和 result_correct 不同步的情況，但卻影響了加法器的判讀以及輸出，倒置輸出的結果錯誤，後來進行不斷地修改以及嘗試，發現是因為在 alu_top 以及 alu_last 的 always 指令中只考慮了當 operation、A_invert 以及 B_invert 發生改變時才進入 always，卻沒考慮到 carry in，導致在 carry in 發生改變時無法進入 always 執行程式，使得輸出結果發生錯誤。

5. Summary:

透過這次 lab1 的實作，使我更加了解 ALU 的架構以及其運作的方式，雖然在設計電路的過程遇到很多問題，但也透過網路上的參考資料以及和同學討論過後找到自己問題所在，並解決問題，同時也可以參考其他同學所遇到的問題以及想法，來找出更好的解決或實作方案。