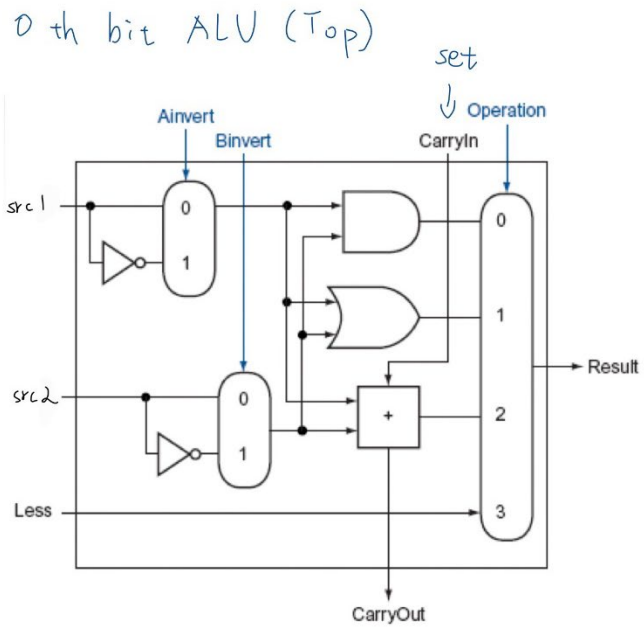
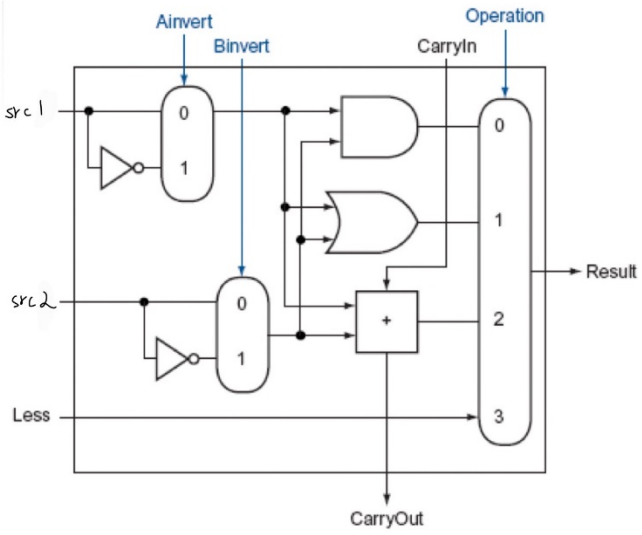
Computer Organization
0819823陳子祈 LAB1

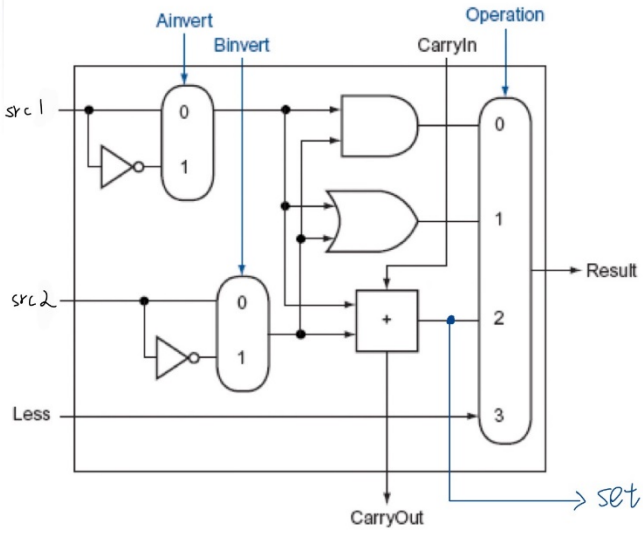
Architecture diagrams:



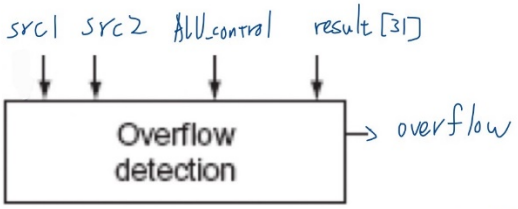
1 th bit ~ 30 th bit ALU (Top)



31 th bit ALU



overflow detection

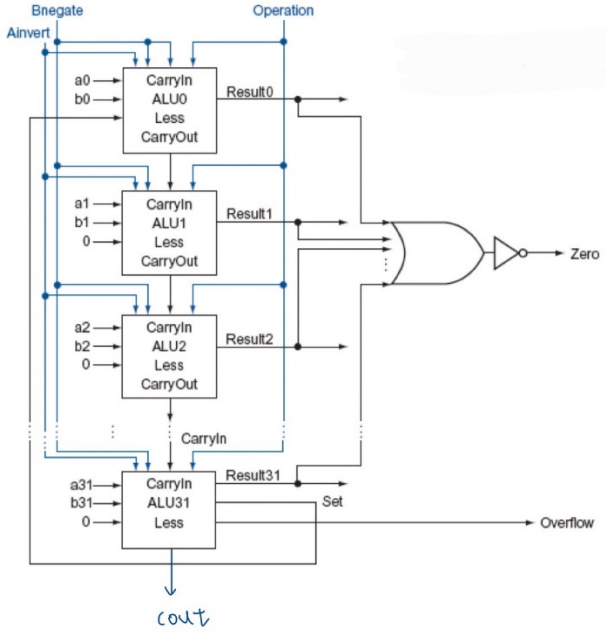


↳加法 ↗ 負+負的結果為正
正+正的結果為負

```
assign overflow
= ( (ALU_control_tmp==4'b0000) & src1_tmp[31] & src2_tmp[31] & ~result_tmp[31]) ? 1
  :( (ALU_control_tmp==4'b0000) & ~src1_tmp[31] & ~src2_tmp[31] & result_tmp[31]) ? 1
  :( (ALU_control_tmp==4'b0110) & src1_tmp[31] & ~src2_tmp[31] & ~(result_tmp[31])) ? 1
  :( (ALU_control_tmp==4'b0110) & ~src1_tmp[31] & src2_tmp[31] & result_tmp[31]) ? 1
  : 0;
```

↳減法 ↳ 負-正的結果為正
正-負的結果為負

Overall ALU Architecture



Hardware module analysis:

1. ALU

   32-bit ALU 可用 devide and conquer 來解決，也就是將問題拆成 1 個 bit 的 ALU(devide)，解決 1 個 bit 的 ALU(conquer)，並串連所有 1-bit 的 ALU module(combine)，其詳細作法如下:根據 ALU_control 的輸入來決定呼叫 1-bit 的 ALU 需要用到甚麼 operation 及 src1、src2 需不需要反向的 A_invert、B_invert，再將 src1、src2、less、operation、A_invert、B_invert、carry_in 等參數輸入到 1-bit 的 ALU，然後將 1-bit 的 ALU 的輸出 carry_out 接到下一級的 ALU，1-bit 的 ALU 也會輸出 result，依此類推直到倒數第二個 bit 的 ALU。因為 set 要接到第 0 個 bit 的 ALU，所以最後一個 bit 的 ALU 長得有點不一樣，會多輸出 set。這邊可以使用 generate for 的方式呼叫 module，這樣就不用一個一個 bit 呼叫 module，不用怕接錯也比較容易讀。

   把所有的 result 都算完之後，再來輸出 zero、cout、overflow。Zero 就是看 result 所有 bit 是不是都是 0，cout 就是最後一個 bit 的 ALU 的 cout。Overflow 的設計可以參考上一頁的說明。

2. ALU_top

   ALU_top 就是除了最後一個 bit 之外其他所有的 1-bit ALU，負責將 src1、src2、less、operation、A_invert、B_invert、carry_in 等參數算出 result 與 cout。先根據 A_invert、B_invert 看要不要將 src1、src2 變號，將結果放到 s1、s2，再根據 operation 的不同，result 與 cout 有各自的算法:

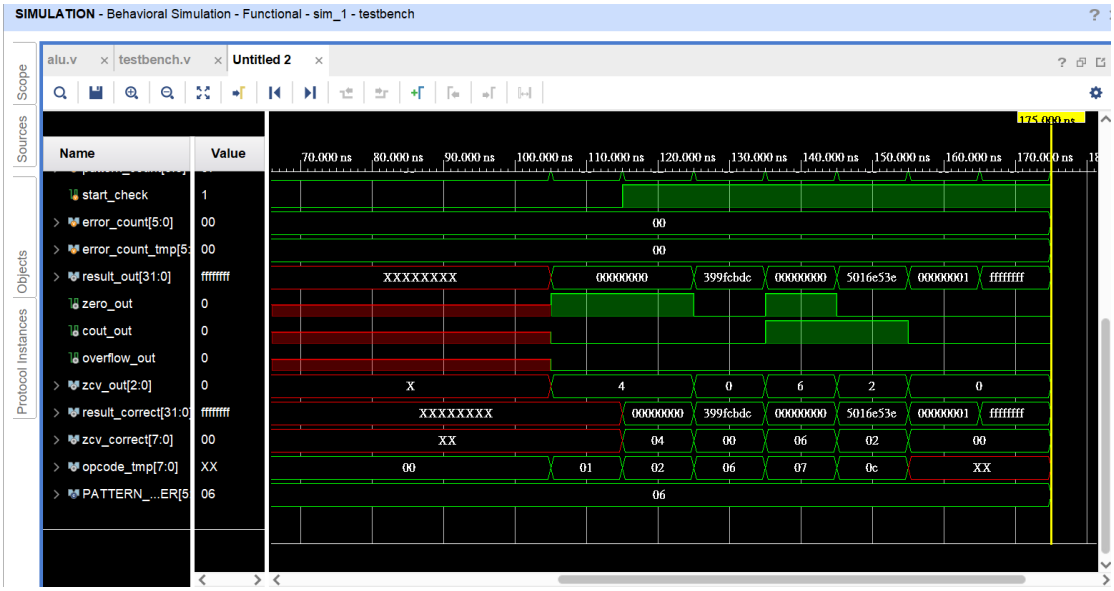
Operation	result	cout
And	s1 & s2	0
Or	s1 | s2	0
ADD	s1 ^ s2 ^ cin	(s1&s2) | (s1&cin) | (s2&cin)
Set less than	less	(s1&s2) | (s1&cin) | (s2&cin)

3. ALU_last

   如同前述所說，ALU_last 會比 ALU_top 多一個 set 的輸出，set 其實就是 s1 ^ s2 ^ cin。Result 與 cout 算法如下:

Operation	result	cout
And	s1 & s2	0
Or	s1 | s2	0
ADD	s1 ^ s2 ^ cin	(s1&s2) | (s1&cin) | (s2&cin)
Set less than	less	0

Experiment result:



result_out 與 result_correct 結果相同，也就是我的輸出與預期輸出結果相同。另外雖然我的 zcv 早一點出現，但 check correct 的時間是 zcv_correct 有訊號的時候才會開始，所以我的結果測出來是對的。

Problems you met and solutions:

問題 1: Verilog 語法極不熟悉

解決: 透過不斷上網查資料，慢慢了解一些 verilog 的語法，雖然還不成熟，但已經盡力了。遇到比如呼叫 module 應寫在 always 前面還是後面、output 要寫在 always 前面還是後面、wire 跟 reg 的運用等問題，後來終於解決。

問題 2: 結果總會早一個週期出現

解決: 本來我是直接把 ouput 接每個 bit 的 module，每次 result 跟 zcv 都會早一個週期出現，後來改成進入 always 才讀取 src1、src2，才終於解決這個問題。

Summary:

本次 LAB 是我第一次真正寫 verilog，所以過程中遇到很多大大小小的問題，花了好多時間上網查、問學長才終於解決，希望可以將這次寫 32 bits ALU 的過程當成一次經驗，以後不會這麼害怕碰硬體描述的東西了。