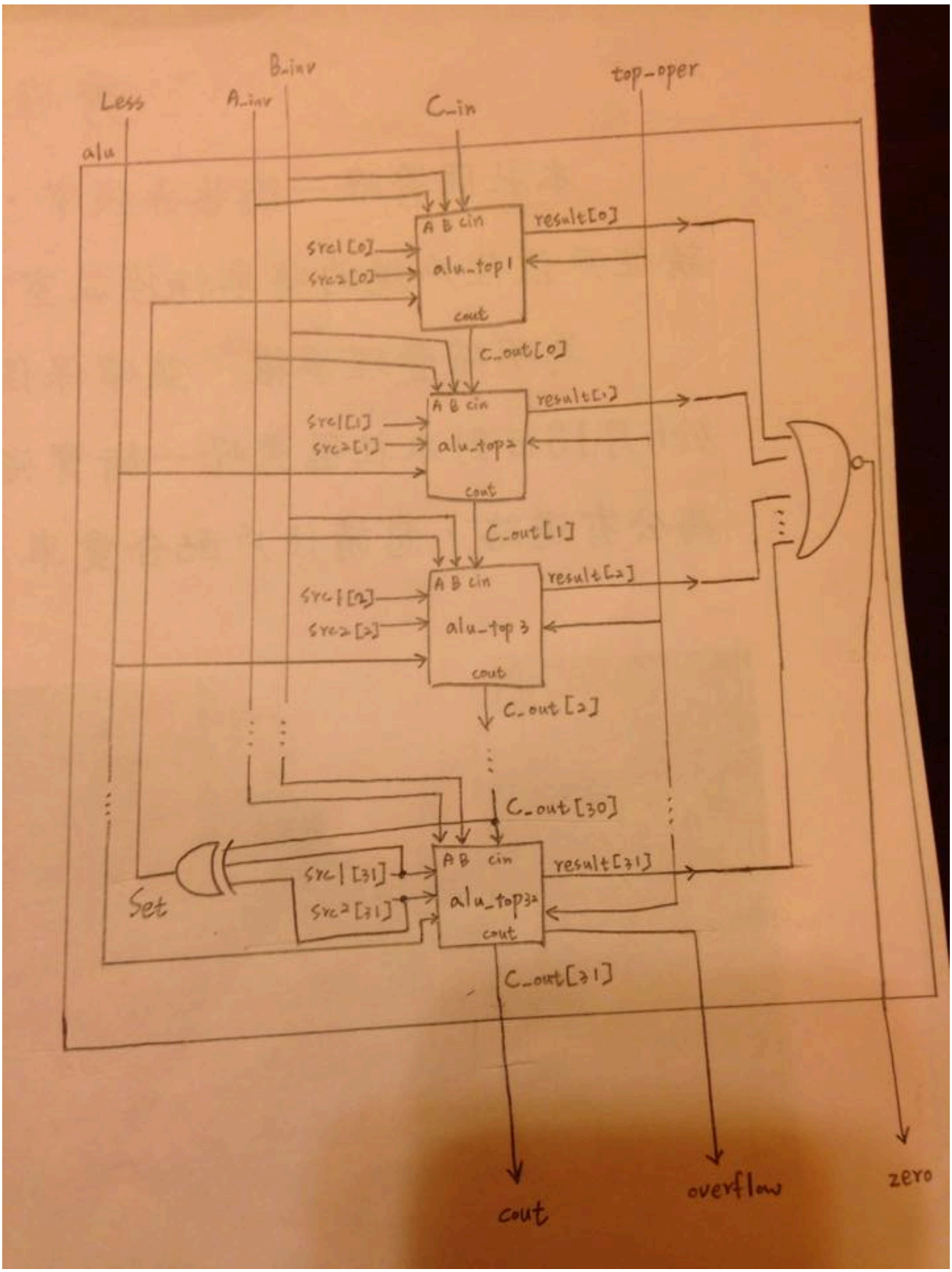
Computer Organization

Architecture diagram:



Detailed description of the implementation:

alu.v：

[1]

always @ (*)內我首先以 if (rst_n)來判斷是否可以開始執行動作(catch data from src1 and src2)；接著在 if (rst_n) begin … end 之間放入 case(ALU_control) … endcase，藉此判斷當下的 ALU action 是 And、Or、Add、Sub、Nor、Nand、Slt 中的哪一種，然後針對不同的 ALU_action 給予 Less、Ainv、Binv、C_in、top_oper 等變數(這些變數分別對應到 mpdule alu_top 中的 less、A_invert、B_invert、cin、operation)不同的值，如此以來，instantiate module alu_top 的時候就可以將它們當作參數傳入。

ALU action	ALU control input	Less	Ainv	Binv	C_in	Top_oper
And	0000	0	0	0	0	00
Or	0001	0	0	0	0	01
Add	0010	0	0	0	0	10
Sub	0110	0	0	1	1	10
Nor	1100	0	1	1	0	00
Nand	1101	0	1	1	0	01
Slt	0111	0	0	1	1	11

[2]

alu.v 的第二個部分是 instance alu_top 三十二次，如以下 code：

```
alu_top head(src1[0],src2[0],Set,Ainv,Binv,C_in,top_oper,result[0],C_out[0]);
generate for(i=1;i<=30;i=i+1)begin alu_top
body(src1[i],src2[i],Less,Ainv,Binv,C_out[i-1],top_oper,result[i],C_out[i]); end endgenerate
alu_top bottom(src1[31],src2[31],Less,Ainv,Binv,C_out[30],top_oper,result[31],C_out[31]);
```

第一次及最後一次的 intance 分別稱為 head 及 bottom，而中間的 30 次 instance 則稱為 body。值得注意的是，head 中本該傳 Less 的位置改成傳 Set，其目的在 Slt 的時候用來判別是否 a-b < 0 ([3] 將會詳細說明])。

補充：多次 instance 需要額外加上 generate … endgenerate(如紅字)，且 for loop 中的 i 其 type 要是 genvar。

[3]

此部分是專門 assign 四個變數，分別是：Set、zero、cout、overflow。

1. **assign Set=(src1[31])^(~src2[31])^(C_out[30]);**

Set 其實就是第 32 個 alu_top 所產生的 Sum，唯其 src2[31]必須改成 ~src[31]。原因是，在做 Slt 首先就是要判斷 a-b 是否小於 0，必須將 b (相當於 src2) 先 complement 才能做減法。

2. **assign zero=(result==32'h00000000)? 1:0;**
   當 result 為 0 的時候，zero 就要被設為 0。
3. **assign cout=(ALU_control==4'b0010 || ALU_control==4'b0110 || ALU_control==4'b0111)? C_out[31]: 1'b0;**
   當有 C_in 的時候，必然也會有 C_out。所以在 ALU_control 為 0010 或 0110 或 0111 的時候(C_in 為 1)，alu 的 cout 就要等於 C_out[31])(第 32 個 alu_top 的 cout)。
4. **assign overflow=(ALU_control==4'b0010 && src1[31]==1'b0 && src2[31]==1'b0 && result[31]==1'b1)? 1:**
   (ALU_control==4'b0010 && src1[31]==1'b1 && src2[31]==1'b1 && result[31]==1'b0)? 1:
   (ALU_control==4'b0110 && src1[31]==1'b0 && src2[31]==1'b1 && result[31]==1'b0)? 1:
   (ALU_control==4'b0110 && src1[31]==1'b1 && src2[31]==1'b0 && result[31]==1'b1)? 1: 0;
   做加減法的時候才會有 overflow：加法的 overflow 出現在「正加正」及「負加負」；減法的 overflow 則是出現在「正減負」及「負減正」。而正負數則是由第 32 個 bit 來決定，就分別是 src1[31]、src2[31]、result[31]，為 0 則該數為正，反之，為 1 則該數為負。

alu_top：

1-bit alu_top module 中就只使用了一個 always @ (*)，在其內部用 if、else if 及 else 將四種 operation(00、01、10、11)做分類。不同的 operation 再進階以 A_invert 或(和)B_invert 是否為 1 進行更進一步的分類。如下：

```
always@( * )begin
    if(operation==2'b00)begin
        if(A_invert==1'b1 && B_invert==1'b1)result=(~src1)&(~src2);   //Nor
        else result=(src1)&(src2);   //And
    end
    else if(operation==2'b01)begin
        if(A_invert==1'b1 && B_invert==1'b1)result=(~src1)|(~src2);   //Nand
        else result=(src1)|(src2);   //Or
    end
    else if(operation==2'b10)begin
        if(B_invert==1'b1)begin result=(src1)^(~src2)^(cin); cout=(((src1^(~src2)) & cin) | (src1&(~src2))); end   //Sub，src2 要 complement，result 就是 sum
        else begin result=src1^src2^cin; cout=(((src1^src2) & cin) | (src1&src2)); end //Add
    end
    else if(operation==2'b11)begin
        //result=(src1)^(~src2)^(cin);
        cout=(((src1^(~src2)) & cin) | (src1&(~src2)));   //Slt 要判別 a-b<0，所以 src2 要
```

```
complement
            result=less;    //Slt 輸出的 result 是 less
        end
end
```

Problems encountered and solutions:

寒假過完，其實幾乎把 Verilog 的語法給忘光光了。所以第一天開始寫的時候，甚至還把 instance 寫在 always block 裡面，或是使用 if、else 的時候不知道必須寫在 always 的裡面；或是 reg、wire 分不清楚，導致第一次 simulation 的時候，連波型圖都跑不出來，點腦甚至還顯示我的程式裡面有 fatal mistake。只好索性找了已經寫完的人來教我語法還有這次作業的大綱，才終於把 alu.v 及 alu_top.v 給寫出來。

不過 debug 的部分就全部都是靠自己的力量了。遇到的問題包括 Add、Slt 的時候出問題：

Add problem：

result_correct 是顯示 0000 …0000(32 個 0)，但我的 result_out 卻是顯示 0000…000X(31 個 0，最後一位是 X)。這個錯誤我找了很久都找不到為甚麼，最後是因為想說 X 是在 LSB，所以往 alu_top 的 head 找，然後卻在下一行的 for loop 發現寫成 **for(i=0;i<=30;i=i+1)**，真是一個不該出現的錯誤，所以當我改成正確的 **for(i=1;i<=30;i=i+1)**，問題也就此解決。

Slt problem：

由於 Slt 要做 a-b<0(即 src1-src2<0)，所以要將 src2 做 complement。然後我在 alu.v 中 instance 的第一個 alu_top(head)的 less 位置是傳入 Set，Set 就是第 32 個 alu_top 的 Sum，如果它是 1 代表 a-b<0，若為 0 則表示 a-b>0。本來我是這樣寫：

assign Set=(src1[31])^(src2[31])^(C_out[30]); //src2沒有 做 complement

改正之後寫成：

assign Set=(src1[31])^(~src2[31])^(C_out[30]); // src2 有做 complement

原因是，如果 Slt 本身就有做 src2 的 complement，那麼用來判斷是否 src1-src2<0 的 Set 也必須跟著做 src2 complement 的動作。