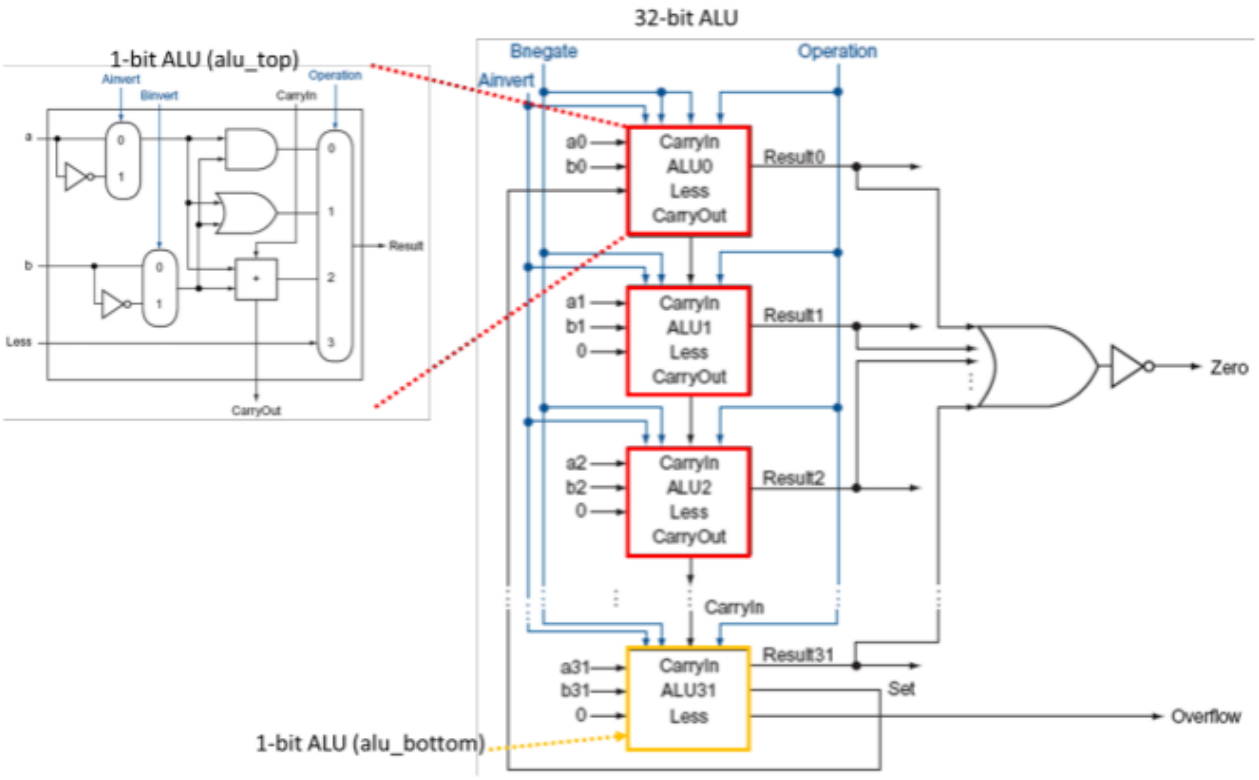
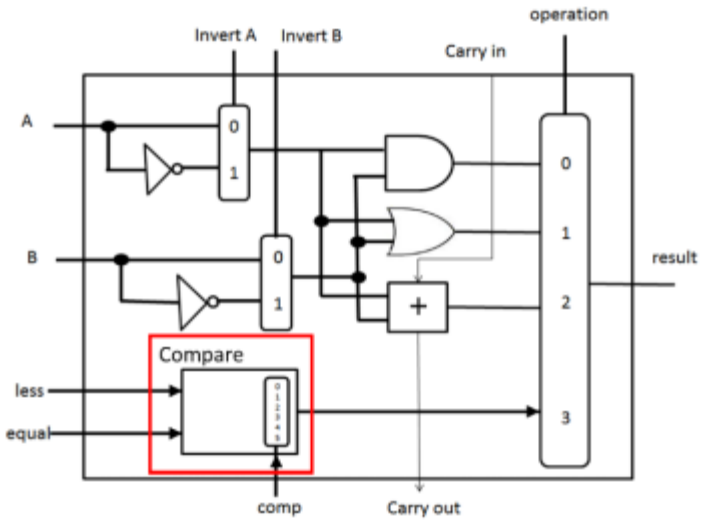
Computer Organization LAB1 Report

1.architecture diagram

基本上都照著講義與 PDF 的架構,詳細文字資料寫在第二點





module:

alu.v	修改自附檔,目的是利用 32 個 1 bit ALU 來達到 32 bits ALU 效果	傳入的參數與一開始助教提供相同 這裡便不解釋
Onebit_Alu.v	onebit_ALU	**input: a,b,carry_in,** **Alu_ctrol,bonus_input** **output:** **result,carry_out,bonus_output** **a**: 接收來自 src1 1bit 的資料 **b**: 接收來自 src2 1bit 的資料 **carry_in**:上一個 bit 的進位結果 **carry_out:** 進位結果 **bonus_input:** 上筆判斷大小結果 **bonus_output:** 判斷大小的結果 **eq_input**:上筆比較相等結果 **eq_output**:此次比較相等之結果 **result:** 儲存結果
Onebit_Adder.v	onebit_ALU 之加法器	input: **a,b,carry_in** output:**carry_out,result** **a**: 接收來自 src1 1bit 的資料 **b**: 接收來自 src2 1bit 的資料 **carry_in**:上一個 bit 的進位結果 **carry_out:** 進位結果 **result**:儲存結果

2.Detailed description of the implementation

Onebit_Adder:

```
1 module Onebit_Adder(a,b,carry_in,carry_out,result);
2
3     input a,b,carry_in;
4     output carry_out,result;
5
6     assign {carry_out,result} = a+b+carry_in;
7
8 endmodule
```

```
// onebit_alu:(a,b,carryin,alucontrol,result.carryout)
Onebit_Alu positone_0(src1[0],src2[0],ALU_control[2],ALU_control,temp_result[0],temp_carry_out[0],1'b1,bonus_temp[0],1'b1,equal_temp[0]);
Onebit_Alu positone_1(src1[1],src2[1],temp_carry_out[0],ALU_control,temp_result[1],temp_carry_out[1],bonus_temp[0],bonus_temp[1],equal_temp[0],equal_temp[1]);
Onebit_Alu positone_2(src1[2],src2[2],temp_carry_out[1],ALU_control,temp_result[2],temp_carry_out[2],bonus_temp[1],bonus_temp[2],equal_temp[1],equal_temp[2]);
Onebit_Alu positone_3(src1[3],src2[3],temp_carry_out[2],ALU_control,temp_result[3],temp_carry_out[3],bonus_temp[2],bonus_temp[3],equal_temp[2],equal_temp[3]);
Onebit_Alu positone_4(src1[4],src2[4],temp_carry_out[3],ALU_control,temp_result[4],temp_carry_out[4],bonus_temp[3],bonus_temp[4],equal_temp[3],equal_temp[4]);
Onebit_Alu positone_5(src1[5],src2[5],temp_carry_out[4],ALU_control,temp_result[5],temp_carry_out[5],bonus_temp[4],bonus_temp[5],equal_temp[4],equal_temp[5]);
Onebit_Alu positone_6(src1[6],src2[6],temp_carry_out[5],ALU_control,temp_result[6],temp_carry_out[6],bonus_temp[5],bonus_temp[6],equal_temp[5],equal_temp[6]);
Onebit_Alu positone_7(src1[7],src2[7],temp_carry_out[6],ALU_control,temp_result[7],temp_carry_out[7],bonus_temp[6],bonus_temp[7],equal_temp[6],equal_temp[7]);
Onebit_Alu positone_8(src1[8],src2[8],temp_carry_out[7],ALU_control,temp_result[8],temp_carry_out[8],bonus_temp[7],bonus_temp[8],equal_temp[7],equal_temp[8]);
Onebit_Alu positone_9(src1[9],src2[9],temp_carry_out[8],ALU_control,temp_result[9],temp_carry_out[9],bonus_temp[8],bonus_temp[9],equal_temp[8],equal_temp[9]);
Onebit_Alu positone_10(src1[10],src2[10],temp_carry_out[9],ALU_control,temp_result[10],temp_carry_out[10],bonus_temp[9],bonus_temp[10],equal_temp[9],equal_temp[10]);
Onebit_Alu positone_11(src1[11],src2[11],temp_carry_out[10],ALU_control,temp_result[11],temp_carry_out[11],bonus_temp[10],bonus_temp[11],equal_temp[10],equal_temp[11]);
Onebit_Alu positone_12(src1[12],src2[12],temp_carry_out[11],ALU_control,temp_result[12],temp_carry_out[12],bonus_temp[11],bonus_temp[12],equal_temp[11],equal_temp[12]);
Onebit_Alu positone_13(src1[13],src2[13],temp_carry_out[12],ALU_control,temp_result[13],temp_carry_out[13],bonus_temp[12],bonus_temp[13],equal_temp[12],equal_temp[13]);
Onebit_Alu positone_14(src1[14],src2[14],temp_carry_out[13],ALU_control,temp_result[14],temp_carry_out[14],bonus_temp[13],bonus_temp[14],equal_temp[13],equal_temp[14]);
Onebit_Alu positone_15(src1[15],src2[15],temp_carry_out[14],ALU_control,temp_result[15],temp_carry_out[15],bonus_temp[14],bonus_temp[15],equal_temp[14],equal_temp[15]);
Onebit_Alu positone_16(src1[16],src2[16],temp_carry_out[15],ALU_control,temp_result[16],temp_carry_out[16],bonus_temp[15],bonus_temp[16],equal_temp[15],equal_temp[16]);
Onebit_Alu positone_17(src1[17],src2[17],temp_carry_out[16],ALU_control,temp_result[17],temp_carry_out[17],bonus_temp[16],bonus_temp[17],equal_temp[16],equal_temp[17]);
Onebit_Alu positone_18(src1[18],src2[18],temp_carry_out[17],ALU_control,temp_result[18],temp_carry_out[18],bonus_temp[17],bonus_temp[18],equal_temp[17],equal_temp[18]);
Onebit_Alu positone_19(src1[19],src2[19],temp_carry_out[18],ALU_control,temp_result[19],temp_carry_out[19],bonus_temp[18],bonus_temp[19],equal_temp[18],equal_temp[19]);
Onebit_Alu positone_20(src1[20],src2[20],temp_carry_out[19],ALU_control,temp_result[20],temp_carry_out[20],bonus_temp[19],bonus_temp[20],equal_temp[19],equal_temp[20]);
Onebit_Alu positone_21(src1[21],src2[21],temp_carry_out[20],ALU_control,temp_result[21],temp_carry_out[21],bonus_temp[20],bonus_temp[21],equal_temp[20],equal_temp[21]);
Onebit_Alu positone_22(src1[22],src2[22],temp_carry_out[21],ALU_control,temp_result[22],temp_carry_out[22],bonus_temp[21],bonus_temp[22],equal_temp[21],equal_temp[22]);
Onebit_Alu positone_23(src1[23],src2[23],temp_carry_out[22],ALU_control,temp_result[23],temp_carry_out[23],bonus_temp[22],bonus_temp[23],equal_temp[22],equal_temp[23]);
Onebit_Alu positone_24(src1[24],src2[24],temp_carry_out[23],ALU_control,temp_result[24],temp_carry_out[24],bonus_temp[23],bonus_temp[24],equal_temp[23],equal_temp[24]);
Onebit_Alu positone_25(src1[25],src2[25],temp_carry_out[24],ALU_control,temp_result[25],temp_carry_out[25],bonus_temp[24],bonus_temp[25],equal_temp[24],equal_temp[25]);
Onebit_Alu positone_26(src1[26],src2[26],temp_carry_out[25],ALU_control,temp_result[26],temp_carry_out[26],bonus_temp[25],bonus_temp[26],equal_temp[25],equal_temp[26]);
Onebit_Alu positone_27(src1[27],src2[27],temp_carry_out[26],ALU_control,temp_result[27],temp_carry_out[27],bonus_temp[26],bonus_temp[27],equal_temp[26],equal_temp[27]);
Onebit_Alu positone_28(src1[28],src2[28],temp_carry_out[27],ALU_control,temp_result[28],temp_carry_out[28],bonus_temp[27],bonus_temp[28],equal_temp[27],equal_temp[28]);
Onebit_Alu positone_29(src1[29],src2[29],temp_carry_out[28],ALU_control,temp_result[29],temp_carry_out[29],bonus_temp[28],bonus_temp[29],equal_temp[28],equal_temp[29]);
Onebit_Alu positone_30(src1[30],src2[30],temp_carry_out[29],ALU_control,temp_result[30],temp_carry_out[30],bonus_temp[29],bonus_temp[30],equal_temp[29],equal_temp[30]);
Onebit_Alu positone_31(src1[31],src2[31],temp_carry_out[30],ALU_control,temp_result[31],temp_carry_out[31],bonus_temp[30],bonus_temp[31],equal_temp[30],equal_temp[31]);
```

alu.v 傳入 32 個 1bit 的資料給 Onebit_Alu.v，須注意的是第一個 carry_in 傳入了 ALU_control[2], 目的是區別加法與減法, 若爲加法 ALU_control[2]爲 0,減法則爲 ALU_control[2]爲 1,作爲補數轉換後需要加一的部份, 並且 carry_out 之結果將傳入下一位 carry_in, 而最後一位 carry_out[31]則作爲判斷 cout 的部份

equal_temp 則是判定兩數是否相等的部份,一開始給予 1,若遇到不相等的值則給予 0 並傳給下一位,若最後 equal_temp[31]爲 1 表兩數相等,爲 0 表兩數不相等

Onebit_ALU:

在 Onebit_ALU 中會先將所有 basic 計算都先做過一次並將結果存起來

```
assign temp[0] = (Alu_ctrol==0)? (a&b) :0 ;                 // and
assign temp[1] = (Alu_ctrol==1)? (a|b) :0 ;                 // or
assign temp[2] = (Alu_ctrol==13)? (!(a&b)) :0;              // nand
assign temp[3] = (Alu_ctrol==12)? (!(a|b)) :0;              // nor
Onebit_Adder add_mod(a,b,carry_in,temp[6],temp[4]);         // add
Onebit_Adder sub_mod(a,inverse_b,carry_in,temp[7],temp[5]); // sub

assign result = (temp[0] || temp[1] || temp[2] || temp[3] || (Alu_ctrol==2 && temp[4]) || (Alu_ctrol==6 && temp[5]));
assign carry_out = (Alu_ctrol==6 && temp[7]) || (temp[6] && Alu_ctrol==2);
```

result 則利用 ALU_control 的值取相對應的值

ALU_control	result 應該取自
0	AND 結果
1	OR 結果
13	NAND 結果
12	NOR 結果
2	ADD 結果
6	SUB 結果

carry_out 結果則看是加法或減法

ALU_control==6	取減法之 carry_out
ALU_control==2	取加法之 carry_out

這邊是做 compare 的部份,bonus_input[0]給予常數 1,這邊最主要目的爲若 **src1>=src2**

```
assign temp_bonus = (a==b)?bonus_input:0;   // for bonus compare
assign bonus_output = (a>b)? 1:temp_bonus;  // for bonus compare
```

則 **alu.v 中的 bonus_temp[31]爲 1,a<b 爲 0**,利用此結果再去做 bonus 的部份

判斷兩數是否相等:

```
assign eq_output= (a==b)?eq_input:0;
```

一開始初始化爲 1,遇到兩數不相等情況則變更爲 0 並傳給下一位

```
assign temp_zero      = (temp_result==0)?1:0;
```

```
assign overflow = (temp_carry_out[31]==temp_carry_out[30]) ? 0 : 1 ; // XOR for determine overflow or not
assign cout     = (ALU_control==2 || ALU_control==6)?temp_carry_out[31]:0; // add or sub need to determine carryout
assign result= (ALU_control==7)?temp2_answer:temp_result;
assign zero = (ALU_control==7)?(result==0):temp_zero;
```

最後會將 BONUS 與非 BONUS 的結果都先算出來再給 result,carry_out,zero 相對應之值

OVERFLOW: 判斷最高位與次高位的 cout 是否相同值,相同的話爲 1 不同爲 0 (即 XOR)
ZERO :則是必須考慮是否爲 BONUS 的 operation, 若是的話則等於(result 是否等於 0 的值); 否的話等於 temp_zero((非 BONUS 的運算結果)是否等於 0)
RESULT :則是判斷是否爲 BONUS 的 operation,是的話其值等於 temp2_answer(Bonus 的運算結果); 否的話等於 temp_result(非 BONUS 的運算結果)

Operation	ALU_control	bonus_control
Slt	7	0
Sgt	7	1
Sle	7	2
Sge	7	3
Seq	7	6
Sne	7	4

```
assign temp2_answer   =   ((src1[31]==0 && src2[31]==0 && bonus_control==0 && bonus_temp[31]==0 && equal_temp[31]==0) || (bonus_control==0 && src1[31]>src2[31]
&& equal_temp[31]==0) || (src1[31]==1 && src2[31]==1 && bonus_control==0 && bonus_temp[31]==0 && equal_temp[31]==0)) || ((src1[31]==0 && src2[31]==0 && bonus_co
ntrol==1 && bonus_temp[31]==1 && equal_temp[31]==0) || (bonus_control==1 && src1[31]<src2[31] && equal_temp[31]==0) || (src1[31]==1 && src2[31]==1 && bonus_cont
rol==1 && bonus_temp[31]==1 && equal_temp[31]==0)) || ( (bonus_control==2 && bonus_temp[31]==0)||(bonus_control==2 && src1==src2)||(src1[31]==0 && src2[31]==0 &
& bonus_control==2 && src1[31]>src2[31])||(src1[31]==1 && src2[31]==1 && bonus_control==2 && src1[31]>src2[31]) ) || ((src1[31]==0 && src2[31]==0 && bonus_contr
ol==3 && bonus_temp[31]==1)||(src1[31]==1 && src2[31]==1 && bonus_control==3 && bonus_temp[31]==1)||(bonus_control==3 && src1[31]<src2[31]) ) || (bonus_control=
=6 && equal_temp[31]==1) || (bonus_control==4 && equal_temp[31]==0); // see detail in report
```

最後則是temp2_answer的部份
利用上述有提到到 **src1>=src2則alu.v中的bonus_temp[31](先取無號情況再深入各做判斷)爲1,a<b爲0**
以及 若兩數相等的話 equal_temp[31]==1的性質

slt: 在bonus_control==0的情況下
　　若sign bit同號 且 bonus_temp[31]爲0且兩數不相等 的話爲1 其餘爲0
　　若sign bit不同號則比較sign bit
sgt: 在bonus_control==1的情況下
　　若sign bit同號 且 bonus_temp[31]爲1且兩數不相等的話爲1 其餘爲0
　　若sign bit不同號則比較sign bit
sle:在bonus_control==2的情況下
　　若sign bit同號 且 bonus_temp[31]爲0的話爲1 其餘爲0
　　若sign bit不同號則比較sign bit
　　若兩數相等的話也爲1
sge:在bonus_control==3的情況下
　　若sign bit同號 且 bonus_temp[31]爲1的話爲1 其餘爲0
　　若sign bit不同號則比較sign bit
　　若兩數相等的話也爲1
seq: 若bonus_control==6且equal_temp[31]==1的話爲1
sne: 若bonus_control==4且equal_temp[31]==0的話爲1

3.Command for compiling your source codes

iverilog -o bonus.vvp testbench.v alu.v Onebit_Adder.v Onebit_Alu.v

4.Problems encountered and solutions

(1)不會寫 Verilog 以及 沒有先修過數位電路設計看不太懂

solution: 上網找資料或找書看, 雖然花很多時間不過因此對其有更深入的了解

(2)Verilog 不像平時寫的軟體:

solution: Verilog 爲硬體描述語言非程式語言, 一開始很不適應, 後來慢慢大概了解怎麼運作, 也知道不能用寫程式語言的角度去寫 Verilog

(3) 看不懂助教給的 data 還有文件:

solution: 有一份 ppt 跟安裝 verilog 的 pdf 不知道要按照哪個做, 後來自己照著 pdf 成功了

(4)PDF 文件說明要我們不要動 testbench,但 Bonus 要測試卻要修改後才能測試：

solution: 寄信問助教(寄信問的問題有點多 QQ 助教辛苦了！)

(5) Bonus 不清楚爲有號還是無號數:

solution: 寄信問助教

5.Lesson learnt

(1) 了解如何寫 Verilog

(2) 對此次的架構有更深入的了解以及知道如何實現他們