实验名称：实验三 加法器与 ALU

姓名：张涵之

学号：191220154

班级：周一 5-6

邮箱：191220154@smail.nju.edu.cn

实验时间：2020/9/22

3.3.1 简单加减法运算器的设计

实验目的：完成一个进行补码加减运算 4 位加减运算器，能够根据控制端完成加、减运算，能判断结果是否为 0，是否溢出，是否有进位等。输入的操作数 A 和 B 都是补码。

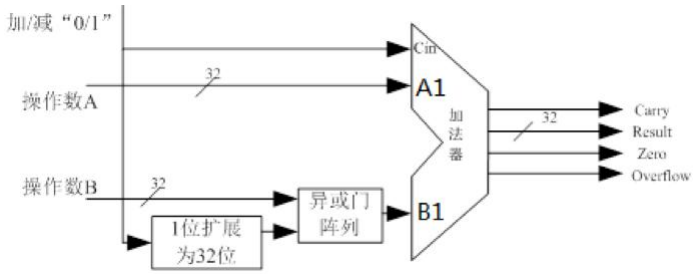
输入信号有：两个 4 位的参与运算的数据的补码操作数 A 和操作数 B，

一个控制做加法还是做减法的加/减控制端 Cin。

输出信号有：一个 4 位的结果 Result、一位进位位 Carry，一位溢出位 Overflow，

一位判断结果是否为零的输出位 Zero。

实验原理：此加减运算器的核心部件是一个 4 位加法器。



进行减法运算时，对操作数 B 取负，已知一个 4 位二进制数的表示范围是-8~7，4 位二进制数和它的相反数的补码代数和为二进制数 10000（0 除外，0000 的补码仍为 0000，即结果仅保留低 4 位），则取 B1=(16–B)%16 得到-B 的补码表示。

计算结果：运算结果为补码相加和的低 4 位。

判断进位：最高位产生进位，即{out_c,out_s} = in_x + in_y;

判断溢出：负数相加得到正数/非负数相加得到负数即溢出，即

Overflow = (in_xn 1 == in_yn 1)&&(out_sn 1! = in_xn 1);

判断是否为零：out_s == 0 则结果为零。

边界条件问题：受到二进制补码表示的局限，减数为-8 时，对 B1=16-B=1000，取值仍为-8，-(-8)转化为+(-8)而不是+8，取 B1 最高位与 A1 比较无法得到正确的溢出位，故 Cin 为 1 时应该直接对 B 的最高位取反与 A1 进行比较，从而得到正确的溢出位。

实验环境/器材：实验箱一个，笔记本电脑一台。

程序代码或流程图：

```
module exp3_1(
    ////////////// SW //////////
    input            [8:0]      SW,

    ////////////// LED //////////
    output           [6:0]      LEDR
);


//=======================================================
//  REG/WIRE declarations
//=======================================================

adder code(SW[0],SW[4:1],SW[8:5],LEDR[0],LEDR[4:1],LEDR[5],LEDR[6]);
```

```
module adder(cin,a1,b1,carry,result,zero,overflow);
   input cin;
   input [3:0]a1;
   input [3:0]b1;
   output reg carry;
   output reg [3:0]result;
   output reg zero;
   output reg overflow;
   integer i;
   integer b;

   always @(cin or a1 or b1) begin

      if (cin) begin
         b = (16 - b1) % 16;
         i = !b1[3];
      end
      else begin
         b = b1;
         i = b1[3];
      end
      {carry, result} = a1 + b;
      if (result == 0)
         zero = 1;
      else
         zero = 0;
      overflow = (a1[3]==i) && (result[3]!=a1[3]);

   end
endmodule
```

A1=A，B1 根据 Cin 判断是否取负 —> 运算和设置标志位。

实验步骤/过程：

根据 Cin 判断操作数 B 是否需要取负，若为减法则求 B1 为-B 的补码。

运算 A+B1，得出结果并设置 Carry，Overflow 和 Zero 标志位。

测试方法：

```
   SW[0]=1'b0; SW[4:1]=4'b0000; SW[8:5]=4'b0011; #20;
// 0 + 3 = 3 = 4'b0011; cf = 0; of = 0; zf = 0;
   SW[0]=1'b1;                                   #20;
// 0 - 3 = -3 = 4'b1101; cf = 0; of = 0; zf = 0;
   SW[0]=1'b0; SW[4:1]=4'b0111;                  #20;
// 7 + 3 = 10 = 4'b1010; cf = 0; of = 1; zf = 0;
   SW[0]=1'b1;                                   #20;
// 7 - 3 = 4 = 4'b0100; cf = 1; of = 0; zf = 0;
   SW[0]=1'b0;                   SW[8:5]=4'b1111; #20;
// 7 + (-1) = 6 = 4'b0110; cf = 1; of = 0; zf = 0;
   SW[0]=1'b1;                                   #20;
// 7 - (-1) = 8 = 4'b1000; cf = 0; of = 1; zf = 0;
   SW[0]=1'b0; SW[4:1]=4'b1000;                  #20;
// (-8) + (-1) = -9 = 4'b0111; cf = 1; of = 1; zf = 0;
   SW[0]=1'b1;                                   #20;
// (-8) - (-1) = -7 = 4'b1001; cf = 0; of = 0; zf = 0;
                                 SW[8:5]=4'b0000; #20;
// (-8) - 0 = -8 = 4'b1000; cf = 0; of = 0; zf = 0;
   SW[0]=1'b0;                   SW[8:5]=4'b1000; #20;
// (-8) + (-8) = -16 = 4'b0000; cf = 1; of = 1; zf = 1;
                 SW[4:1]=4'b0111;                #20;
// 7 + (-8) = -1 = 4'b1111; cf = 0; of = 0; zf = 0;
   SW[0]=1'b1;                                   #20;
// 7 - (-8) = 15 = 4'b1111; cf = 0; of = 1; zf = 0;
                                 SW[8:5]=4'b0111; #20;
// 7 - 7 = 0 = 4'b0000; cf = 1; of = 0; zf = 1;
```

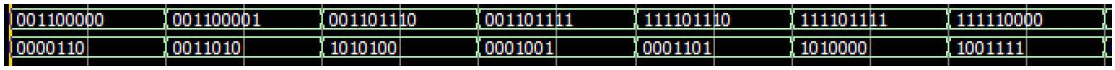
其中 SW[0]为 Cin，SW[4:1]和 SW[8:5]分别为操作数 A 和 B 的补码。

LEDR[0]为 Carry，LEDR[4:1]为 Result，LEDR[5]为 Zero，LEDR[6]为 Overflow。

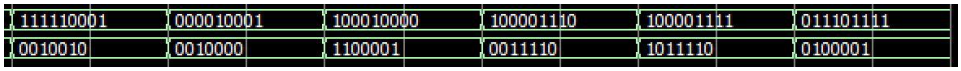
根据输入的测试数据手动计算结果和判断标志位，用于仿真时对照判断。

样例设置进位/不进位，溢出/不溢出，零/非零，并对 0 和(-8)的加减进行额外测试。

实验结果：



001100000	001100001	001101110	001101111	111011110	111101111	111110000
0000110	0011010	1010100	0001001	0001101	1010000	1001111



111110001	000010001	100010000	100001110	100001111	011101111
0010010	0010000	1100001	0011110	1011110	0100001

通过观察对比，运算结果与符号位均与手动计算所得相符。
经接入实验箱检验，显示也符合预期。

实验中遇到的问题及解决办法：
对减 0 和减-8 时边界条件的处理：
1）0 的补码仍为 0，用二进制数 10000 做减法需舍去最高位，模 16 可以解决；
2）对-8 取补码仍为-8 而非 8，造成溢出位的判断发生错误，结合计算机系统基础相关知识得出运算时不应先取负再取相反数的最高位，而应直接对原操作数最高位取反。

实验得到的启示：应该关注特殊值和边界条件的处理，从具体例子发现和解决问题。

意见和建议：无。

思考题 1：应该比较 A、B 和运算结果的符号位，其中若 Cin 为 1 则 B 的符号位取反。若比较 A1、B1 和运算结果的符号位，则当 Cin 为 1，B1 为-8 时出现问题。

思考题 2：两种方法的产生的运算结果一样，进位位和溢出位不完全一样。
方法一在做减法且 B 为 0 时出现问题，0000 的相反数补码为 0000，计算 B1 为 10000，则进位位为 1，而任何数减 0 都不应该有借位，是第二步中相反数补码计算错误造成的。
方法二在做减法且 B 为-8 时出现问题，1000 的相反数补码为 1000，计算 B1 为 1000，该补码表示的是-8 而非 8，符号位为 1 而非 0，则根据 B1 符号位判断的溢出位错误，是第四步中使用相反数符号位而非对原操作数符号位取反造成的。

思考题 3：assign zero = ~(| Result)，其中一元约简运算的操作过程为对操作数各位依次执行同一操作直到最后一位，此处 Result 为 4 位二进制数，相当于 zero = ~(Result[0] | Result[1] | Result[2] | Result3)，即 Result 各位全部为 0 时 zero = 1，否则 zero = 0。

3.3.2 实现一个带有逻辑运算的简单 ALU

实验目的：实现对 4 位有符号数操作的 ALU，SW 作为数据输入，button 为选择端。
其中选择端共可以实现八种功能，输入的操作数均为有符号数的补码表示。

表 3-1: ALU 功能列表

功能选择	功能	操作
000	加法	A+B
001	减法	A-B
010	取反	Not A
011	与	A and B
100	或	A or B
101	异或	A xor B
110	比较大小	If A>B then out=1; else out=0;
111	判断相等	If A==B then out=1; else out=0;

实验原理：用 case 语句选择需要实现的功能：
加法与减法：原理同 3-3-1，根据加减法确定操作数 B 是否需要取负，对 A1 和 B1 进行补码的加法运算得到计算结果，并设置相应的进位和溢出标志位；
取反、与、或、异或：对操作数进行简单逻辑运算即可。
比较大小：分类讨论，若 A 和 B 一正一负可直接得出大小关系；
若 A 和 B 均为正数，则补码和原码相同，可以通过 A>B 直接判断；
若 A 和 B 均为负数，则它们与各自相反数的补码之和为二进制数 10000，补码的无符号值越大，其绝对值越小，真值越大，也可以通过 A>B 直接判断；
判断相等：直接使用==判断相等。

实验环境/器材：实验箱一个，笔记本电脑一台。

程序代码或流程图：
输入 S —case 语句—> 功能选择 —输入 A 和 B—> 进行运算 —> 输出结果。

```
module exp3_2(
    ////////////// KEY //////////
    input             [2:0]     KEY,

    ////////////// SW //////////
    input             [7:0]     SW,

    ////////////// LED //////////
    output            [5:0]     LEDR
);


//=======================================================
//  REG/WIRE declarations
//=======================================================

alu code(KEY[2:0],SW[3:0],SW[7:4],LEDR[3:0],LEDR[4],LEDR[5]);
```

```
module alu(s,a1,b1,result,carry,overflow);
   input [2:0]s;
   input [3:0]a1;
   input [3:0]b1;
   output reg [3:0]result;
   output reg carry;
   output reg overflow;
   integer i;
   integer b;

   always @(s or a1 or b1) begin

      case (s)
         3'b000 : begin
            b = b1;
            i = b1[3];
            {carry, result} = a1 + b;
            overflow = (a1[3]==i) && (result[3]!=a1[3]);
         end
         3'b001 : begin
            b = (16 - b1) % 16;
            i = !b1[3];
            {carry, result} = a1 + b;
            overflow = (a1[3]==i) && (result[3]!=a1[3]);
         end
         3'b010 : begin
            result = ~a1;
            carry = 0;
            overflow = 0;
         end
         3'b011 : begin
            result = a1 & b1;
            carry = 0;
            overflow = 0;
         end
         3'b100 : begin
            result = a1 | b1;
            carry = 0;
            overflow = 0;
         end
         3'b101 : begin
            result = a1 ^ b1;
            carry = 0;
            overflow = 0;
         end
         3'b110 : begin
            result = 4'b0000;
            if (a1[3]==1 && b1[3]==0) carry = 0;
            else if (a1[3]==0 && b1[3]==1) carry = 1;
            else carry = (a1>b1) ? 1:0;
            overflow = 0;
         end
         3'b111 : begin
            result = 4'b0000;
            if (a1 == b1) carry = 1;
            else carry = 0;
            overflow = 0;
         end
         default : begin
            result = 4'b0000;
            carry = 0;
            overflow = 0;
         end
      endcase

   end
endmodule
```

实验步骤/过程：

根据 Cin 判断操作数 B 是否需要取负，若为减法则求 B1 为-B 的补码。

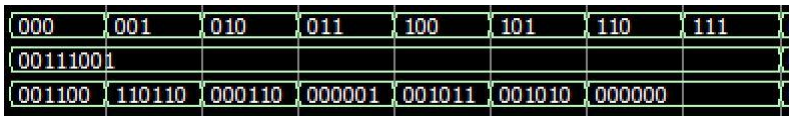
运算 A+B1，得出结果并设置 Carry，Overflow 和 Zero 标志位。

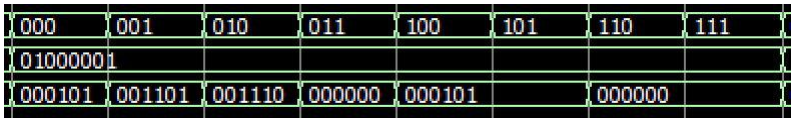
测试方法：

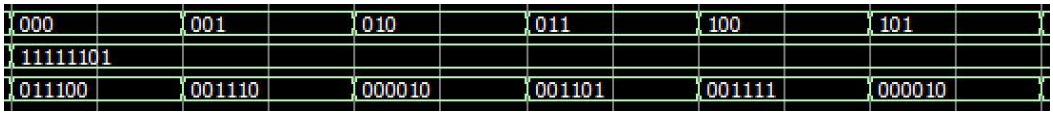
```
// code that executes only once
// insert code here --> begin
    KEY[2:0]=3'b000; SW[3:0]=4'b1001; SW[7:4]=4'b0011; #20;  //-7+3=-4     1100 0 0
    KEY[2:0]=3'b001;                                   #20;  //-7-3=-10    0110 1 1
    KEY[2:0]=3'b010;                                   #20;  //0110
    KEY[2:0]=3'b011;                                   #20;  //0001
    KEY[2:0]=3'b100;                                   #20;  //1011
    KEY[2:0]=3'b101;                                   #20;  //1010
    KEY[2:0]=3'b110;                                   #20;  //0
    KEY[2:0]=3'b111;                                   #20;  //0
    KEY[2:0]=3'b000; SW[3:0]=4'b0001; SW[7:4]=4'b0100; #20;  //1+4=5       0101 0 0
    KEY[2:0]=3'b001;                                   #20;  //1-4=-3      1101 0 0
    KEY[2:0]=3'b010;                                   #20;  //1110
    KEY[2:0]=3'b011;                                   #20;  //0000
    KEY[2:0]=3'b100;                                   #20;  //0101
    KEY[2:0]=3'b101;                                   #20;  //0101
    KEY[2:0]=3'b110;                                   #20;  //0
    KEY[2:0]=3'b111;                                   #20;  //0
    KEY[2:0]=3'b000; SW[3:0]=4'b1101; SW[7:4]=4'b1111; #20;  //-3+(-1)=-4 1100 1 0
    KEY[2:0]=3'b001;                                   #20;  //-3-(-1)=-2 1110 0 0
    KEY[2:0]=3'b010;                                   #20;  //0010
    KEY[2:0]=3'b011;                                   #20;  //1101
    KEY[2:0]=3'b100;                                   #20;  //1111
    KEY[2:0]=3'b101;                                   #20;  //0010
    KEY[2:0]=3'b110;                                   #20;  //0
                     SW[3:0]=4'b0000;                  #20;  //1
    KEY[2:0]=3'b111;                                   #20;  //0
                                     SW[7:4]=4'b0000; #20;  //1
// --> end
//$display("Running testbench");
```

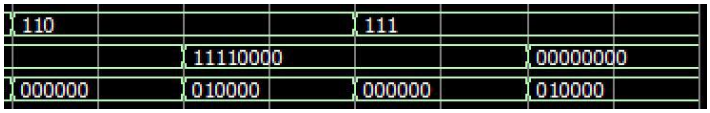
其中 KEY[2:0]为功能选择端，SW[3:0]和 SW[7:4]分别为操作数 A 和 B 的补码。
LEDR[3:0]为 Result，LEDR[4]为 Carry，LEDR[5]为 Overflow。
根据输入的测试数据手动计算结果和判断标志位，用于仿真时对照判断。

实验结果：









通过观察对比，运算结果与符号位均与手动计算所得相符。
经接入实验箱检验，显示也符合预期。

实验中遇到的问题及解决办法：问题都在 3-3-1 解决了，3-3-2 没有出现问题。

实验得到的启示：好玩。

意见和建议：无。