中国科学院大学计算机组成原理（研讨课）

实 验 报 告

学号：2021K8009929034 姓名：侯汝垚 专业：计算机科学与技术

实验序号：1 实验名称：基本功能部件——寄存器堆和算术逻辑单元

注 1：撰写此 Word 格式实验报告后以 PDF 格式保存 SERVE CloudIDE 的 /home/serve-ide/cod-lab/reports 目录下（注意：reports 全部小写）。文件命名规则：prjN.pdf，其中“prj”和后缀名“pdf”为小写，“N”为 1 至 4 的阿拉伯数字。例如：prj1.pdf。PDF 文件大小应控制在 5MB 以内。此外，实验项目 5 包含多个选做内容，每个选做实验应提交各自的实验报告文件，文件命名规则：prj5-projectname.pdf，其中“-”为英文标点符号的短横线。文件命名举例：prj5-dma.pdf。具体要求详见实验项目 5 讲义。

注 2：使用 git add 及 git commit 命令将实验报告 PDF 文件添加到本地仓库 master 分支，并通过 git push 推送到 SERVE GitLab 远程仓库 master 分支（具体命令详见实验报告）。

注 3：实验报告模板下列条目仅供参考，可包含但不限定如下内容。实验报告中无需重复描述讲义中的实验流程。

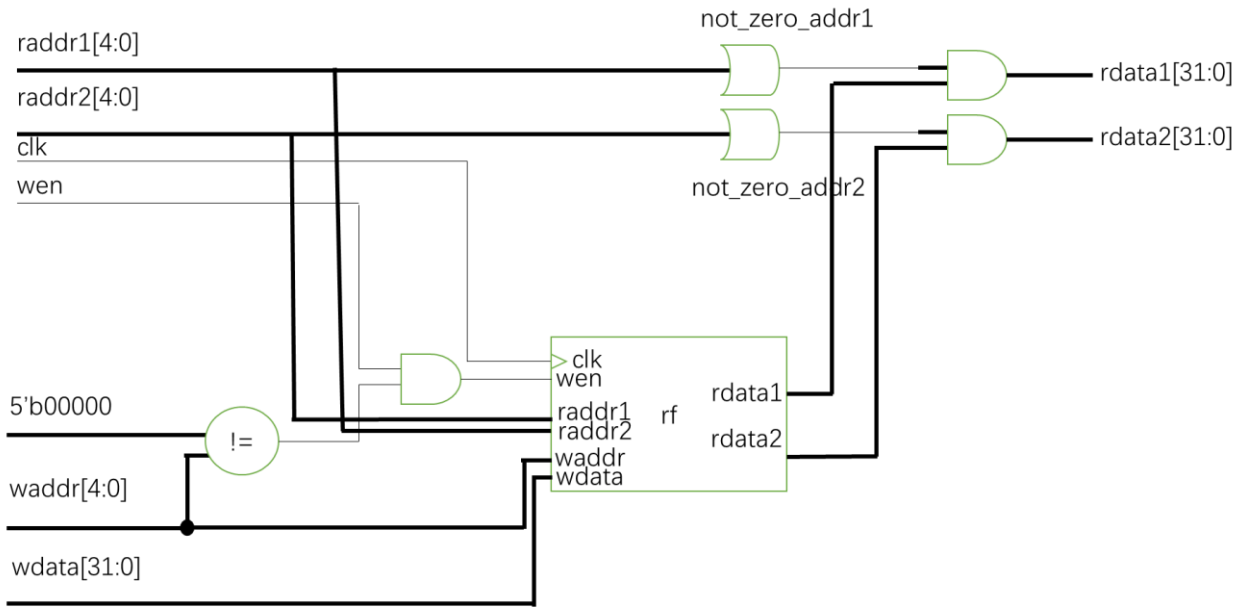
一、 逻辑电路结构与仿真波形的截图及说明

1. 寄存器堆

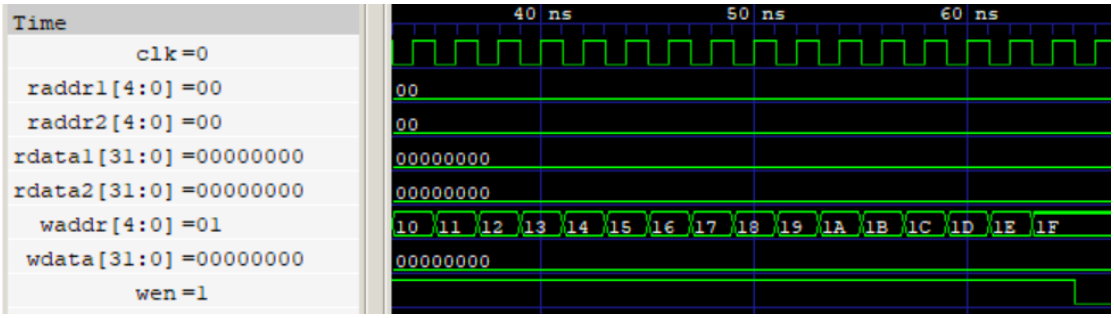
a) RTL 代码

```
reg [`DATA_WIDTH - 1:0] rf [`REG_NUM - 1:0];

always @(posedge clk) begin
        if (wen == 1'b1 && waddr != `ADDR_WIDTH'b0) begin
                rf[waddr] <= wdata;
        end
end

// Judgment signal for whether the address is 0
wire not_zero_adder1 = | raddr1;
wire not_zero_adder2 = | raddr2;

assign rdata1 = {`DATA_WIDTH{not_zero_adder1}} & rf[raddr1];
assign rdata2 = {`DATA_WIDTH{not_zero_adder2}} & rf[raddr2];
```

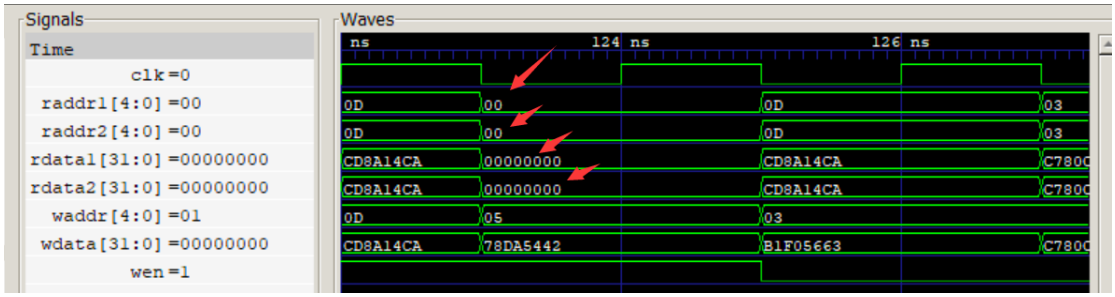
b) 逻辑电路图



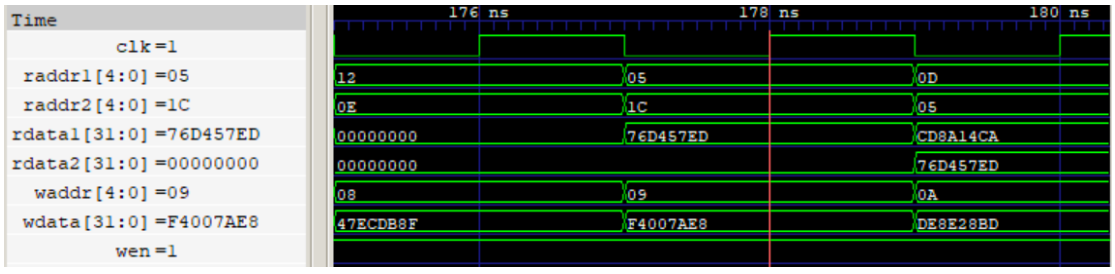
c) 仿真波形图



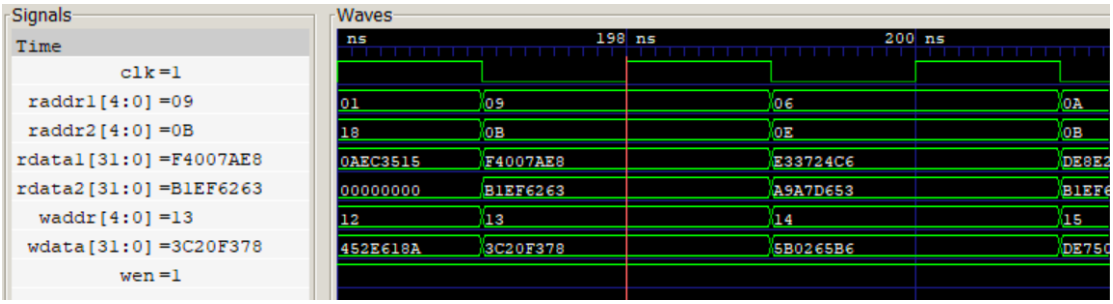
图表 1 初始化时向每个寄存器写入 0



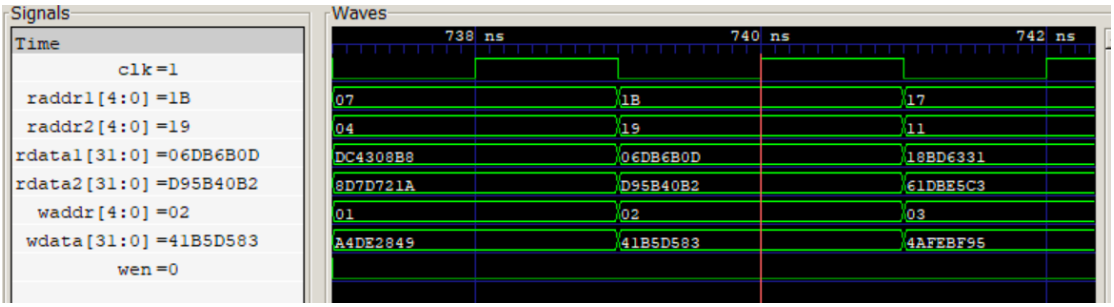
图表 2 读取 0 号寄存器时输出一定位 0



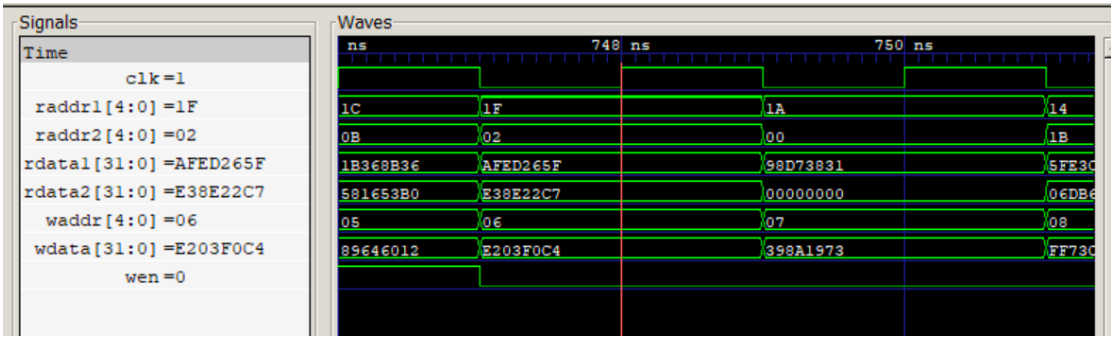
图表 3 向 9 号寄存器写入 F4007AE8



图表 4 从 9 号寄存器中读取写入的 F4007AE8



图表 5 wen = 0 时向 2 号寄存器写入 41B5D583



图表 6 wen = 0 时无法写入，故 2 号寄存器读取中的值不是 41B5D583

2. 算术逻辑单元

a) RTL 代码

采用独热码方式标记输入的 ALUop，方便判断需要进行的操作。

```
parameter AND = 3'b000;
parameter OR  = 3'b001;
parameter ADD = 3'b010;
parameter SUB = 3'b110;
parameter SLT = 3'b111;
parameter MSB = `DATA_WIDTH - 1;

wire op_and = (ALUop == AND);
wire op_or  = (ALUop == OR );
wire op_add = (ALUop == ADD);
wire op_sub = (ALUop == SUB);
wire op_slt = (ALUop == SLT);
```

定义多条 wire 类型变量来完成电路，其中 add_A 和 add_B 是进入加法器的两个数，add_A 就是输入的 A，但是 add_B 会根据所做操作连接 B 或者 B 的反码。带“result”的 wire 变量连接各种操作得到的结果，cin 和 cout 则是加法器的进位输入和进位输出，含“sign”的 wire 变量连接加法器两个输入数字和结果的符号位，注意 sign_B 是 add_B 的符号而不是 B 的符号。

```
    wire [MSB:0] add_A, add_B; // two operands in adder
    wire [MSB:0] add_result;   // get the results of
    wire [MSB:0] slt_result;   // add/sub, slt, and, or
    wire [MSB:0] and_result;
    wire [MSB:0] or_result;
    wire cin, cout;            // cin is used to select add or sub
    wire sign_A, sign_B;       // the signs of add_A, "add_B" and
                               //result
    wire sign_result;

    assign cin = ALUop[2];          // do subtraction if ALUop is
                                    //SUB //or SLT, that is ALUop[2]
                                    //= 1
    assign add_B = (B ^ {32{cin}}); // If the cin is 1, add_B will
                                    //be the inverse of B.
    assign add_A = A;
    assign sign_A = add_A[MSB];     // get the signs of add_A,
                                    //"add_B" and result
    assign sign_B = add_B[MSB];
    assign sign_result = add_result[MSB];
    assign and_result = A & B;
    assign or_result  = A | B;
```

创建一个带有 cin 和 cout 的加法器。

```
    assign {cout,  add_result} = add_A + add_B + cin;
```

判断减法结果是否为负数，如果加法器的两个运算数都为负数，则结果一定为负数，即 sign_A && sign_B，如果操作数符号不同，可以直接判断结果符号，因为这个时候结果不会发生溢出，符号一定正确。

```
   assign slt_result = {31'b0,  (sign_A && sign_B) || ((sign_A ^
                                                sign_B) && sign_result)};
```

根据需要做的操作给 Result 赋值。

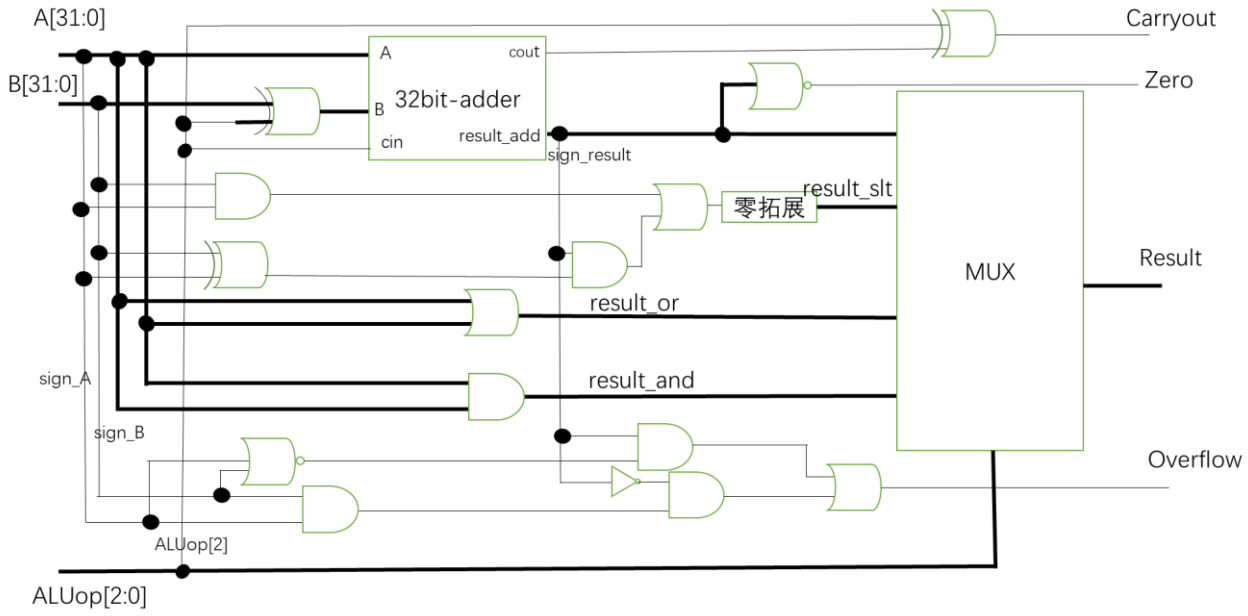
```
   assign Result = (and_result & {`DATA_WIDTH{op_and}})
                 | (or_result  & {`DATA_WIDTH{op_or }})
                 | (add_result & ({`DATA_WIDTH{op_add}} |
                                  {`DATA_WIDTH{op_sub}}))
                 | (slt_result & {`DATA_WIDTH{op_slt}});
```

进位/借位和溢出的判断，进位借位判断并不直观，如果是加法时进位则 cout 为 1，如果是减法时借位 cout 为 0，后面给出证明。而溢出判断则是采用两个操作数和结果符号不同来判断，后面给出证明。Zero 信号判断直接将结果所有位取或非即可。

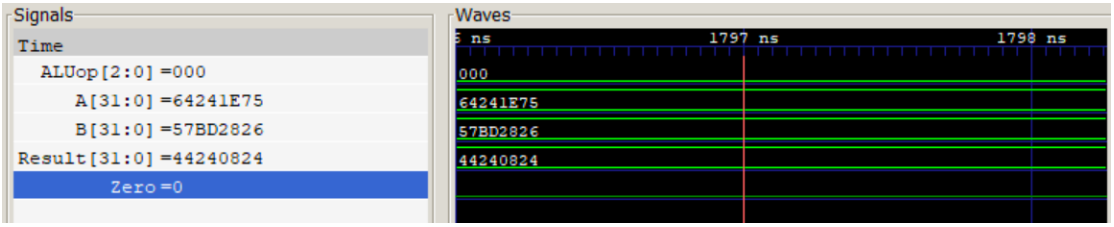
```
    assign CarryOut = cout ^ cin;
```

```
    assign Overflow = (sign_A && sign_B && !sign_result) || (!sign_A
                                        && !sign_B && sign_result);
    assign Zero = ~(| Result)
```

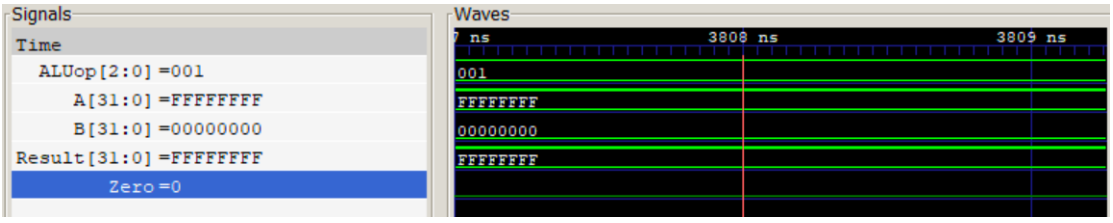
b) 逻辑电路图



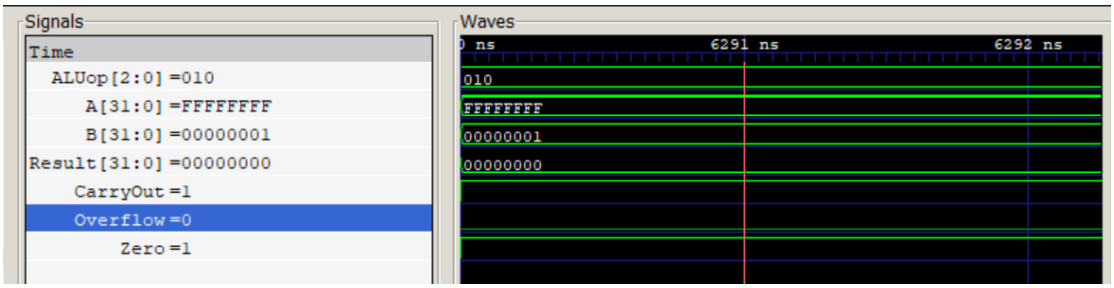
c) 仿真波形图



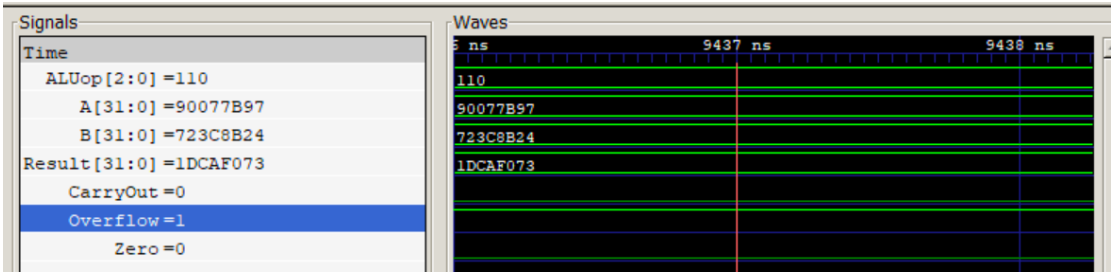
图表 7 按位与操作



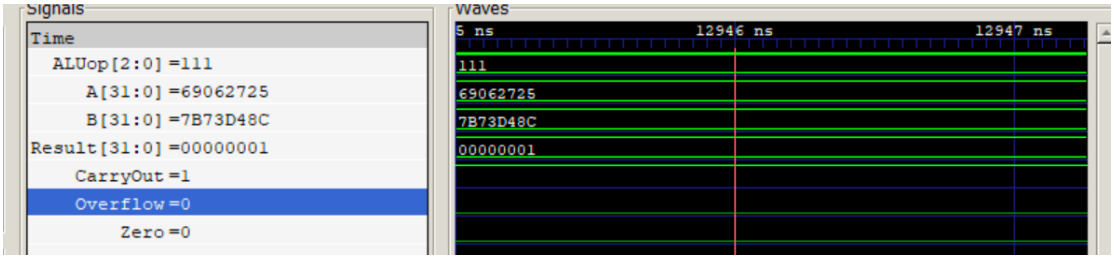
图表 8 按位或操作



图表 9 加法操作，对于无符号数来说发生进位，对于有符号数来说结果正确，且结果为 0



图表 10 减法操作，对于无符号数来说结果正确，对于有符号数来说结果溢出



图表 11 A 小于 B，结果置为 1

二、 实验过程中遇到的问题、对问题的思考过程及解决方法

1. Reg_file 中如何对 0 地址进行处理?

讲义中提到不要用 initial 语句进行处理，因为通常情况下 initial 块中的语句不可综合，也不要用 generate-for 语句对寄存器初始化，然而假如不做任何判断，在 bhv_sim 阶段则会出错：

```
ERROR: Read at 00, should get 00000000, but get xxxxxxxx.
```

图表 12 读取 0 时获得不定态

最先想到的办法是在每个 clk 周期都给 rf[0]赋值 0，也成功通过了测试，讲义上提到“需要通过 assign 语句，确保读 0 号寄存器时输出 32’d0”，然而 rf[0]是 reg 类型，不能通过 assign 赋值，所以我想到了使用三目运算符做判断，假如地址为 0 则输出 0，否则读取相应数据。然后又改成通过逻辑门进行选择。这么写在每次读取寄存器的时候都需要进行判断，在实际操作中读取零寄存器应该是少数情况，因此可能效率较低。我想最高效的办法是直接初始化零寄存器值为 0，然后就无须再进行其他判断，但是我猜测这样会因为一些原因更改了零号寄存器导致 bug。

2. Reg_file 提交后在 fpga_eval 阶段出现报错。

我的 reg_file 文件测试时在 fpga_eval 阶段出现报错，在助教的帮助下的排查，原因在于我在定义寄存器堆时在索引中使用了 parameter 类型变量。

```
parameter NUM_OF_REG = 1 << `ADDR_WIDTH;
reg [`DATA_WIDTH - 1:0] rf [NUM_OF_REG - 1:0];
```

图表 13 排查出的错误代码

```
Error: r2 does not have written data=643c9869 (rs1=00000000, rs2=0000
0000)
Error: r1's data has been overwritten
```

图表 14 报错内容

根据报错内容，在 r2 写入内容时并没有写入，而是 r1 被 overwritten，两者相差 1，这可能和索引中的 `[NUM_OF_REG - 1:0];` 有关。假如我将 NUM_OF_REG 定义为 31 不减 1，那么可以通过测试。在网上搜索并未发现相关内容，网上也有部分人采用 parameter 变量经过计算后定义存储器，我猜测 bug 的原因是我对 parameter 的使用有问题或者研讨课的硬件设备不支持这种用法。

3. Alu 中为什么有符号数和无符号数可以采用同一套加法电路?

对于有符号数和无符号数的加法采用同一套加法电路比较容易理解，毕竟加法电路和人工计算的逻辑一致，都是对于位相加然后向上传递进位，然而对于减法来说却不是那么显然。对于有符号数计算的方法是 $A_{补}-B_{补}=A_{补}+(-B)_{补}$，但是补码是有符号数的概念，对于无符号数来说为什么也可以这么计算呢?

该问题可以写为：设A，B是k位二进制无符号数（相当于非负整数），证明在模2^k的条件下有$A-B\ =\ A+\sim B\ +1$（~表示按位取反）

显然有

$$B+\sim B=2^k-1$$

故

$$-B\equiv(\sim B+1)\ mod\ 2^k$$

因此

$$A-B\equiv A+\sim B+1\ mod\ 2^k$$

因为无符号数表示的范围为$0\sim 2^k-1$，输入的A和B默认在范围内，只要结果不为负数，则一定在范围内，即电路上的二进制结果即为结果的真实值。

因为加法器在计算过程中自动抹除 k 位以上的位数，等价于对2^k取模，所以电路计算的结果即使出现借位，模2^k后也是正确的。同样对于无符号数加法来说，只要结果小于2^k，就一定是结果的真实值，即使发生了进位，在取模的条件下结果依然正确。因为默认输入在$0\sim 2^k-1$范围内，故加的结果范围为$0-2^{k+1}\sim 2$，减的结果范围为$-2^k+1\sim 2^k-1$，故当真实结果超过2^k-1时为真实结果为电路获得的结果加上2^k，当真实结果小于 0 时真实结果为电路获得的结果减去2^k，所以可以设定一个$Carryout$标志，理解为向高位的进位或借位。

4. 进位/借位和溢出有什么区别? 为什么不用同一个信号而要区分开?

进位/借位和溢出都是因为有限位能表示的数字有限，所以结果超出能表示的边界值时就会出错。进位/借位针对无符号数，溢出针对无符号数，这两个信号很有可能是

不一样的，比如计算$0-1$时如果将其看作无符号数则发生借位，但是看作有符号数则不发生溢出。

5. 进位/借位公式$Carryout = cin \wedge cout$的证明.

对于无符号数加法比较显然，发生进位时加法器中的$cout$即为向高位的进位，也就是 Carryout.一个比较直观的猜想是发生借位是也有$cout = 1$，然而经过检验发现并非如此，甚至恰恰相反，发生借位时$cout = 0$，否则$cout = 1$。证明如下:

假设A，B是k位二进制数，根据 3.中证明有

$$A - B \equiv A + \sim B + 1 \; mod \; 2^k$$

因此加法器电路中实际执行的是$A + \sim B + 1$.显然有

$$B + \sim B = 2^k - 1$$

当$A - B$为发生借位时，即$A - B$为负数时有$A < B$.故

$$A + \sim B + 1 < 2^k$$

因此有$cout = 0$

当$A - B$不发生借位时，即$A - B$为非负数时有$A \geq B$.故

$$A + \sim B + 1 \geq 2^k$$

因此有$cout = 1$

6. 溢出公式$c_n \wedge c_{n-1}$(表示第 n 位和第 n-1 位进位)的证明。

对于一位符号位判断溢出，其原理是“参加操作的两个数符号相同，其结果的符号与原操作数的符号不同”。

设$A_n = B_n = m$， $m \in \{0, 1\}$， 即两数符号位相同，则$c_n = m$， 设两数相加结果为D，则其符号位为$D_n = A_n \wedge B_n \wedge c_{n-1} = c_{n-1}$， 故$A_n \wedge D_n = c_n \wedge c_{n-1}$.

可以通过符号判断溢出的原因是，对于正数相加来说，相加的结果一定小于$2^k - 1$，故溢出时最多只向第 k 位上一位进 1，这个 1“污染”了正确的符号位 0 使得符号位变成了 1；对于负数相加来说结果不是那么直观，发生溢出的原因是“两个数值太小导致不足以产生进位到符号位”。假设有两个负数A和B，则它们的补码表示为$2^k + A$和$2^k + B$，其和为$D = 2^{k+1} + A + B$，假如发生了溢出，那么$A + B < -2^{k-1}$，则$D < 2^{k+1} - 2^{k-1} = 3 * 2^{k-1}$，模$2^k$后得$D < 2^{k-1}$，因此此时$D$的符号位为 0。

想要连接第$n-1$位的进位在电路上比较容易，但是因为代码中直接用加号生成 32 位加法器，不能直接获得第$n-1$位的进位，所以采用符号判断而不是进位判断，稍微麻烦了一点。如果自己手写加法器模块就可以做到这直接用进位判断。

7. Alu 中对最大负数取补码时对符号判断错误产生的 bug 以及减 0 的借位判断 bug。

在做减法运算时最开始的想法是对减数取补码然后再把它送入加法器。根据公式$(-A)_{补} = \sim A + 1$，可以算出减数的补码，虽然这样会多创建一个加法器。有一个特例是，假设包括符号位总共有k位，那么2^{k-1}，也就是 1 后面跟上$k-1$个 0，这个数是最大的负数，这个数的相反数在k位条件下是没有补码表示的，按照公式算出来的数字是它本身。计算的结果并不会出现问题，因为$2^{k-1} + 2^{k-1} \equiv 0 \; mod \; 2^k$，但是在判断符

号的时候会出现 bug。假如计算$1-(-2^{k-1})$，取反加一后计算$1+2^{k-1}$，那么-2^{k-1}的“补码”符号仍然为 1，根据溢出判断会认为这是异号数字相加，没有 Overflow。

另一个特例是 0，同样对 0 取反加一后仍为 0，假如计算1 - 0，相当于$1+0$，那么根据$Carryout = cin \wedge cout$，会得到$Carryout = 1$，但是减 0 不发生借位的。

第一个 bug 的根本原因在于 0 的“逆元”和2^{k-1}的“逆元”都是它们本身，根据维基百科，“For a given number of bits k there is an even number of binary numbers 2k, taking negatives is a group action (of the group of order 2) on binary numbers, and since the orbit of zero has order 1, at least one other number must have an orbit of order 1 for the orders of the orbits to add up to the order of the set. Thus some other number must be invariant under taking negatives (formally, by the orbit-stabilizer theorem).”也就是说因为这个群中 0 的“逆元”是它本身，那么必定存在另一个数，其逆元是它本身。第二个 bug 的根本原因在于，根据 3.中推导我们有$A-B \equiv A+\sim B+1 \ mod\ 2^k$，发生借位时有$A+\sim B+1<2^k$，当$B=0$时$A+\sim B+1=A+2^k \geq 2^k$，没有借位，但是假如先对$B$取反加 1，则算式为$A+(\sim B+1)\ mod\ 2^k = A < 2^k$，判断为有借位。所以这个结果的原因是因为先计算了$(\sim B+1)\ mod\ 2^k$，当$B \neq 0$时$(\sim B+1)\ mod\ 2^k = \sim B+1$，不出错，当$B=0$时就会出错，因为$cout$位上的 1 在对 0 取反加 1 时候被忽略了。

修复这两种特殊情况最先想到的方法是加一个判断逻辑，对这两种情况进行特殊处理，但是这样效率很低。一个两全其美的办法是将做减法的判断信号cin作为加法器输入，而对减数只是按位取反，然后将减法判断信号cin作为加法器进位输入，这样每个正数（零视为正数）和负数存在一一对应关系，当2^{k-1}作为减数时取他的反码为$2^{k-1}-1$，保证了符号判断不出错，另外加法器计算的确实是$A+\sim B+1$而不是$A+(\sim B+1)\ mod\ 2^k$，当$A+\sim B+1 \geq 2^k$时会正确输出$cout=1$。同时这样设计还可以少用一个加法器，非常巧妙。

8. 二进制补码表示的想法

补码表示是电路上的值和我们理解的值之间的一个双射。

模2^k剩余类加群$(G_1,+)$也就是电路高低电平代表的无符号数，设$G_2 = -2^{k-1},-2^{k-1}+1,\dots,0,\dots,2^{k-1}-1$，也就是我们理解的有符号数，定义$G_2$上的运算$\cdot$，$\forall A,B \in G_2$

$$A \cdot B = \begin{cases} A+B, & A+B \in G_2 \\ A+B-2^k, & A+B \geq 2^{k-1} \\ A+B+2^k, & A+B < -2^{k-1} \end{cases}$$

则G_2也是一个循环群，单位元为 0，除了最小负数外逆元为其相反数，最小负数逆元是它本身，G_1和G_2取补码定义的映射是一个群同构。$\forall A \in G_2$

$$f(A) = A_{补} = \begin{cases} A, & A \geq 0 \\ 2^n + A, & A < 0 \end{cases}$$

而G_1中元素的逆元为其二进制表示取反加 1，根据同构保持运算的性质，$\forall A,B \in G_2$，有

$$f(A+B^{-1}) = f(A)+f(B^{-1}) = f(A)+f(B)^{-1} = f(A)+\sim f(B)+1$$

即$(A-B)_{补} = A_{补}+B^{-1}{}_{补} = A_{补}+\sim B_{补}+1$

三、 在课后，你花费了大约 7 小时完成此次实验

课后花 1 小时复习 Verilog 相关语法，4 小时编写 RTL 代码，2 小时完成实验报告。

四、 对于此次实验的心得、感受和建议

虽然我在上学期的数字电路实验课上已经使用过 VerilogHDL 编写一些模块，但是只停留在了仿真阶段，并没有真正在硬件上检验。在本次实验中我深刻体会到了仿真正确并不代表硬件实现也能正确，在前四个阶段的验证都正确的情况下，我遇到了“正常情况下不会出错的”fpga_evl 错误，这还只是一个简单的寄存器堆，对于更大的电路，在各种不确定因素的叠加下更有可能出现仿真正确但是实践时错误的情况，从中可见上板验证是非常重要的过程，仅停留在仿真模拟阶段是不够的。

在验收阶段和陈卓勋助教交流过程中，我也有很多收获。在助教的指导下我意识到即使通过了实验的所有测试案例也不代表我的设计是正确的，对于设计的原理必须要有严格的证明而不是仅凭直觉或者猜想，某种设计正确可能只是因为碰巧正确，bug 发现得越晚代价越大。因此回去后在实验报告中我对于一些电路设计进行了证明。

本次实验难度适中，reg_file 较为简单，alu 需要对二进制运算有一定理解，不过因为忘记的语法内容较多经常出现一些语法错误。实验平台非常省心非常人性化。希望一些设计上的指导建议可以在实验发布阶段就提出，例如使用组合逻辑门描述选择电路，这样就不用再重新修改写好的部分代码。