中国科学院大学计算机组成原理实验课

实 验 报 告

学号：2020K8009929017　姓名：侯昱帆　专业：计算机科学与技术

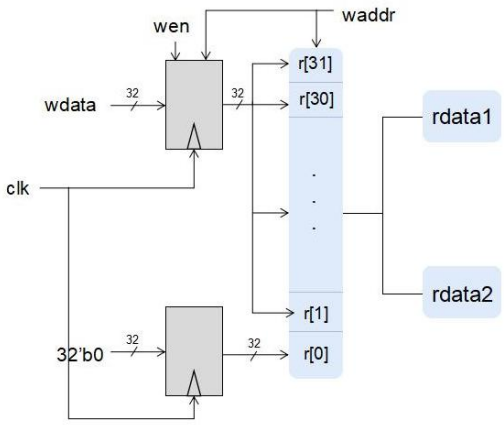
实验序号：1　实验名称：基本功能部件——寄存器堆和算术逻辑单元

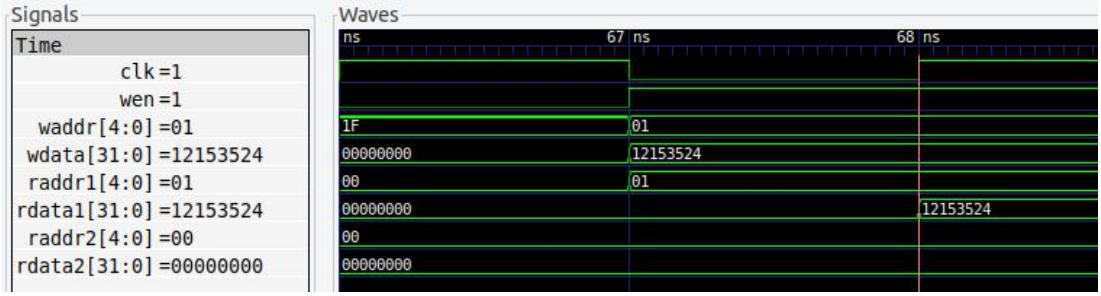
注 1：撰写此 Word 格式实验报告后以 PDF 格式保存在~/COD-Lab/reports 目录下。文件命名规则：prjN.pdf，其中“prj”和后缀名“pdf”为小写，“N”为 1 至 4 的阿拉伯数字。例如：prj1.pdf。PDF 文件大小应控制在 5MB 以内。此外，实验项目 5 包含多个选做内容，每个选做实验应提交各自的实验报告文件，文件命名规则：prj5-projectname.pdf，其中“-”为英文标点符号的短横线。文件命名举例：prj5-dma.pdf。具体要求详见实验项目 5 讲义。
注 2：使用 git add 及 git commit 命令将实验报告 PDF 文件添加到本地仓库 master 分支，并通过 git push 推送到 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（比如关键 RTL 代码段{包含注释}及其对应的逻辑电路结构图、相应信号的仿真波形和信号变化的说明等）

1.Register

```
// 32*32-bit register
reg    [`DATA_WIDTH - 1:0]  r [`DATA_WIDTH - 1:0];

// synchronous write
always @(posedge clk) begin
    r[0] <= 32'b0;
    if (wen == 1 && waddr != 0)
        r[waddr] <= wdata;
end
// asynchronous read
assign rdata1 = r[raddr1];
assign rdata2 = r[raddr2];
```



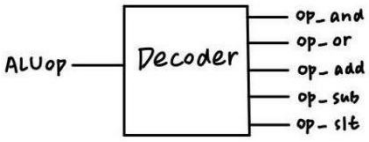


写操作：当 clk 上升沿到来，wen=1，此时 waddr=1，故将向 r[1]写入数据 12153524。

读操作：raddr1 为 1，从 r[1]读出的数据是 12153524。

2. ALU

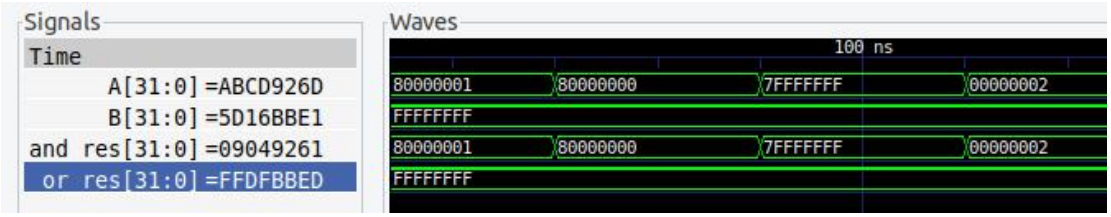
```
// decoding
wire op_and = ALUop == `ALUop_AND;
wire op_or  = ALUop == `ALUop_OR;
wire op_add = ALUop == `ALUop_ADD;
wire op_sub = ALUop == `ALUop_SUB;
wire op_slt = ALUop == `ALUop_SLT;
```



由波形图，可见控制信号译码成功。

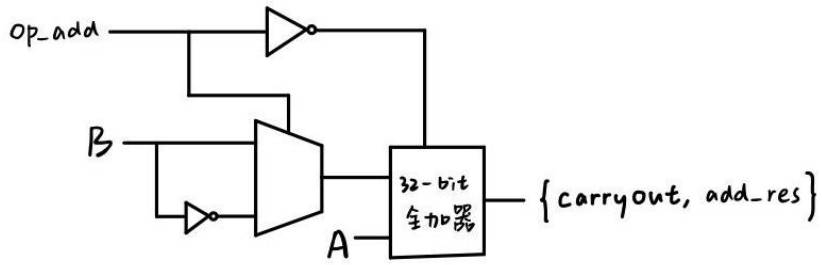
如：ALUop=000 时，op_and=1；ALUop=001 时，op_or=1；ALUop=010 时，op_add=1。

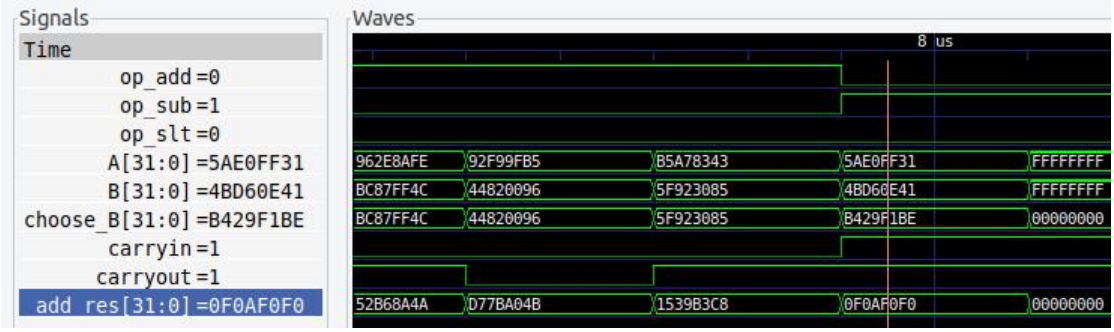
```
    // and & or
    wire [`DATA_WIDTH - 1:0] and_res, or_res;

    assign and_res = A & B;
    assign or_res  = A | B;
```



由波形图可得，ALU 实现了与或功能，and_res 和 or_res 分别为与操作和或操作的结果。

```
    // add & sub
    wire [`DATA_WIDTH - 1:0] add_res;
    wire [`DATA_WIDTH - 1:0] choose_B;
    wire carryin, carryout;

    assign carryin  = ~op_add;      // if add, carryin = 1; else if add or slt, carryin = 0
    assign choose_B = op_add? B:~B; // if add, choose B; else if add or slt, choose ~B
    assign {carryout, add_res} = A + choose_B + carryin; // if add, add_res = A + B; otherwise,
add_res = A + ~B + 1
```

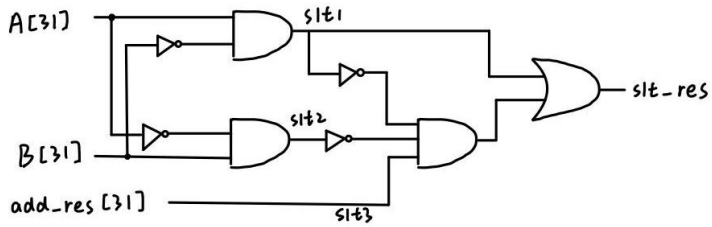


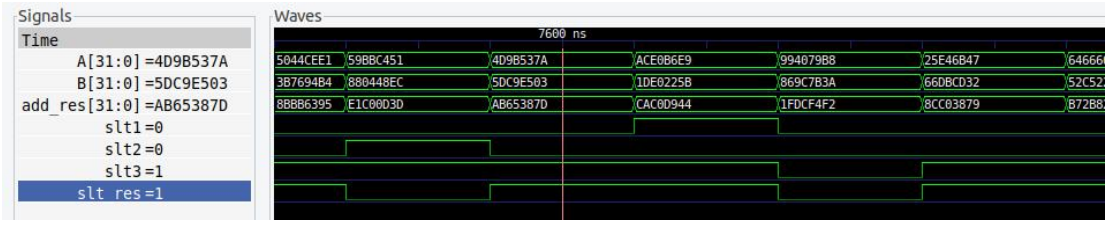


当 op_add=1,choose_B 与 B 相等，carryin=0，add_res 为 A+B 的结果。

当 op_add=0,choose_B 为~B，carryin=1，add_res 为 A-B 的结果。

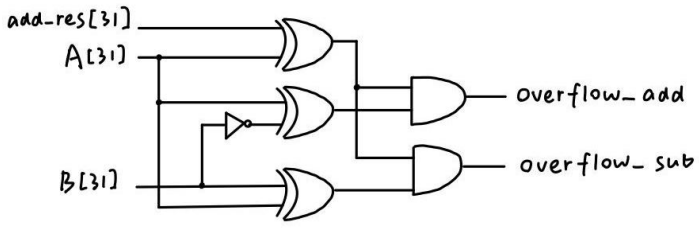
```
    // slt
    wire slt_res;
    wire slt1,slt2,slt3;

    assign slt1 =  A[`DATA_WIDTH-1] & ~B[`DATA_WIDTH-1];  // if A<0,B>0 slt1=1
    assign slt2 = ~A[`DATA_WIDTH-1] &  B[`DATA_WIDTH-1];  // if A>0,B<0 slt2=1
    assign slt3 = add_res[`DATA_WIDTH-1];                 // if A and B have the same sign, check
the sign of the add_res
    assign slt_res = slt1 | (~slt1 & ~slt2 & slt3);
```

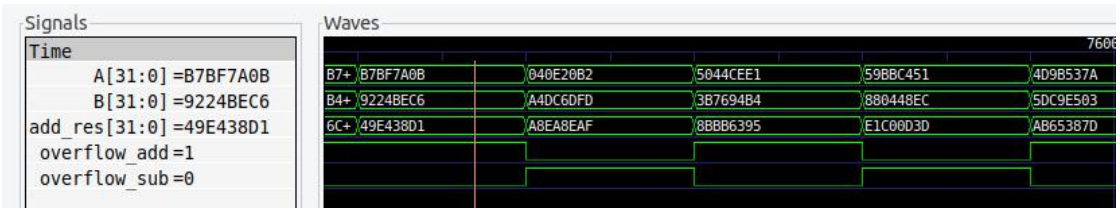




当 A=4D9B537A，B=5DC9E503，A 与 B 符号相同，故 slt1 和 slt2 均为 0。
而 A-B=AB65387D，为负数，故 slt3=1。所以 A<B，slt_res=1。

```
    // overflow
    wire overflow_add, overflow_sub;

    assign overflow_add = (A[`DATA_WIDTH-1] ^ ~B[`DATA_WIDTH-1]) & (A[`DATA_WIDTH-1] ^
add_res[`DATA_WIDTH-1]); // pos + pos = neg (0 0 1) or neg + neg = pos (1 1 0)
    assign overflow_sub = (A[`DATA_WIDTH-1] ^  B[`DATA_WIDTH-1]) & (A[`DATA_WIDTH-1] ^
add_res[`DATA_WIDTH-1]); // pos - neg = neg (0 1 1) or neg - pos = pos (1 0 0)
```

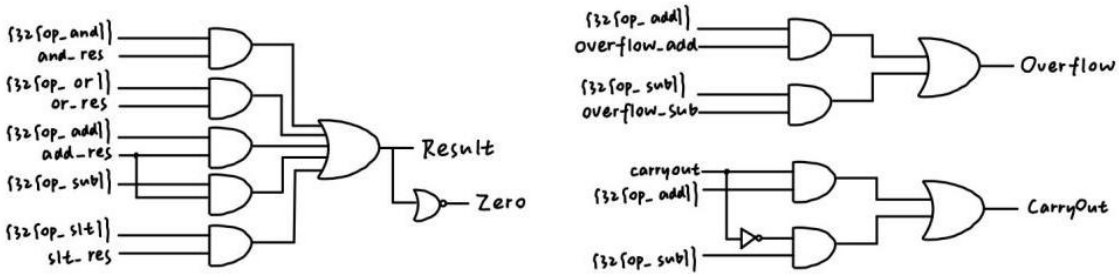


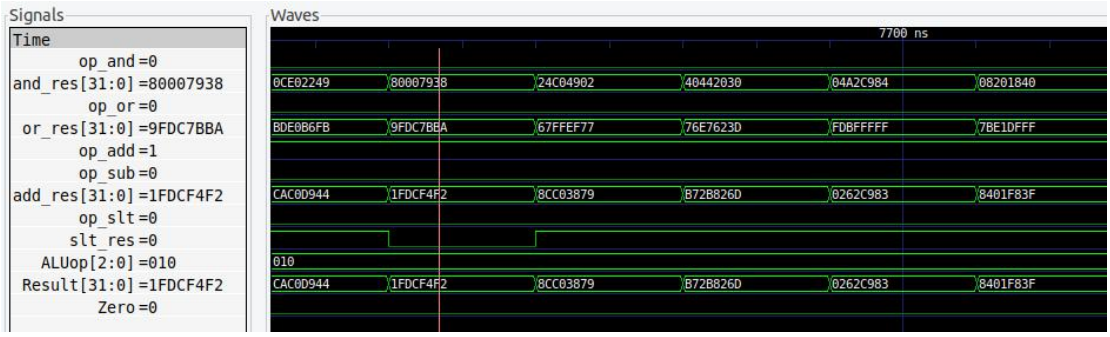


A=B7BF7A0B，B=9224BEC6。

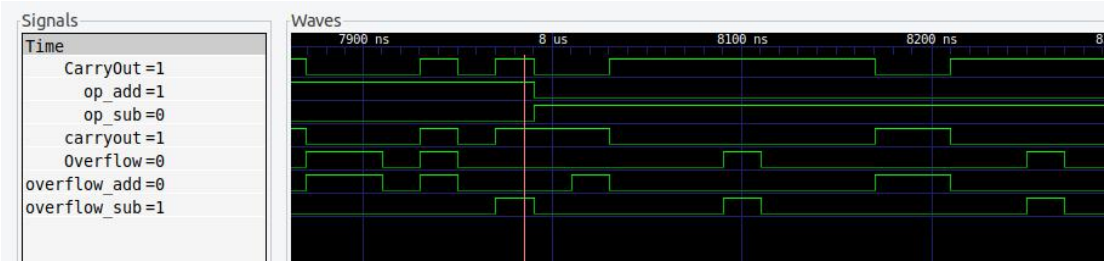
A 符号位为 1，B 符号位为 1，add_res 符号位为 0。做加法时会溢出，overflow_add=1。

```
    // choose the result
    assign Result   = {32{op_and}} & and_res | {32{op_or}} & or_res | {32{op_add}} & add_res |
{32{op_sub}} & add_res | {32{op_slt}} & slt_res;
    assign Overflow = ({32{op_add}} & overflow_add) | ({32{op_sub}} & overflow_sub);
    assign CarryOut = {32{op_add}} & carryout | {32{op_sub}} & ~carryout;
    assign Zero     = ~|Result;
```





根据控制信号 op 来选择 Result 的值。比如 op_add=1，则 Result=add_res=1FDCF4F2。



CarryOut 和 Overflow 的值也会根据 op_add 和 op_sub 确定。比如：

op_add=1，做加法，carryout=1，则 CarryOut=1。overflow_add=0，则 Overflow=0。

op_sub=1，做减法，carryout=1，则 CarryOut=0。overflow_sub=0，则 Overflow=0。

二、 实验过程中遇到的问题、对问题的思考过程及解决方法（比如 RTL 代码中出现的逻辑 bug，逻辑仿真和 FPGA 调试过程中的难点等）

1.Register

主要遇到的问题是对 0 号寄存器的处理。

0 号地址读出的值总是常量 32’b0，且不能向 0 号地址写入 wdata。

起初我使用 always 块对除 0 号地址外的其他地址写入数据，然后又单独使用 assign 语句对 0 号地址赋值，结果出现报错。因为一个寄存器类型变量的所有的赋值逻辑仅能出现在一个 always 块里。

```
// synchronous write
always @(posedge clk) begin
    r[0] <= 32'b0;
    if (wen == 1 && waddr != 0)
        r[waddr] <= wdata;
end

assign r[0] = 32'b0;
```

于是我将对 0 号寄存器的赋值也写到了 always 块里，但是使用了 if-else 语句。在验收时老师指出：如果 waddr 永远不是 0，那么将无法从 0 号地址读出 32’b0。

```
// synchronous write
always @(posedge clk) begin
    if (wen == 1 && waddr != 0)
        r[waddr] <= wdata;
    else if (waddr == 0)
        r[waddr] <= 32'b0;
end
```

所以 0 号寄存器的赋值不可以使用 if，而是直接赋值。最终代码如下：

```
// synchronous write
always @(posedge clk) begin
    r[0] <= 32'b0;
    if (wen == 1 && waddr != 0)
        r[waddr] <= wdata;
end
```

2. ALU 实验

① 第一个问题是 add，sub 和 slt 的逻辑。起初我没有理解使用同一套加法器逻辑的含义，将 add 和 sub 分开写了，得到了加法的结果 add_res 和减法的结果 sub_res，并且在 slt 中使用了 sub_res，最后再根据控制信号选择结果。

```
    assign {carryout1, add_res} = A + B;
    assign {carryout2, sub_res} = A + ~B + 1;
```

后来我才理解，使用同一套加法器逻辑是指只有一个加法器，也就是只有一条使用了加法的语句。如果有多条语句都使用了加号，就会生成多个加法器了，不符合要求。我们需要根据控制信号对加数 B 进行选择，再使用同一个加法器运算。

```
    assign carryin  = ~op_add;      // if add, carryin = 1; else if add or slt, carryin = 0
    assign choose_B = op_add? B:~B; // if add, choose B; else if add or slt, choose ~B
    assign {carryout, add_res} = A + choose_B + carryin; // if add, add_res = A + B; otherwise,
add_res = A + ~B + 1
```

② 第二个问题是化简。

起初，我使用了一些非必要的二路选择器，比如：

```
    assign carryin = (~op_add)?  1:0;
    assign Zero    = (~|Result)? 1:0;
```

验收时经老师提醒，已经做了如下修改。

```
    assign carryin  = ~op_add;
    assign Zero     = ~|Result;
```

起初，在写 overflow 时，我将所有会溢出的情况都进行了列举，

```
    // pos + pos = neg 0 0 1
    assign overflow1 = ~A[`DATA_WIDTH-1] & ~B[`DATA_WIDTH-1] &  add_res[`DATA_WIDTH-1];
    // neg + neg = pos 1 1 0
    assign overflow2 =  A[`DATA_WIDTH-1] &  B[`DATA_WIDTH-1] & ~add_res[`DATA_WIDTH-1];
    // pos - neg = neg 0 1 1
    assign overflow3 = ~A[`DATA_WIDTH-1] &  B[`DATA_WIDTH-1] &  add_res[`DATA_WIDTH-1];
    // neg - pos = pos 1 0 0
    assign overflow4 =  A[`DATA_WIDTH-1] & ~B[`DATA_WIDTH-1] & ~add_res[`DATA_WIDTH-1];
```

验收时经老师指导，也进行了化简。

```
    // pos + pos = neg (0 0 1) or neg + neg = pos (1 1 0)
    assign overflow_add = (A[`DATA_WIDTH-1] ^ ~B[`DATA_WIDTH-1]) & (A[`DATA_WIDTH-1] ^
add_res[`DATA_WIDTH-1]);
    // pos - neg = neg (0 1 1) or neg - pos = pos (1 0 0)
    assign overflow_sub = (A[`DATA_WIDTH-1] ^  B[`DATA_WIDTH-1]) & (A[`DATA_WIDTH-1] ^
add_res[`DATA_WIDTH-1]);
```

三、 在课后，你花费了大约 5 小时完成此次实验。

四、 对于此次实验的心得、感受和建议（比如实验是否过于简单或复杂，是否缺少了某些你认为重要的信息或参考资料，对实验项目的建议，对提供帮助的同学的感谢，以及其他想与任课老师交流的内容等）

我认为这次实验难度适中。ALU 的设计比较有挑战性，特别是对加减法和比大小使用同一套加法器逻辑的理解。经过反复思考和调试代码，最终通过测试，能获得满满的成就感。上课时除了对寄存器和 ALU 的讲解外，还额外补充了一些 verilog 组合逻辑的知识,对实验帮助很大。这次实验也加深了我对寄存器和 alu 工作原理的理解，和理论课相辅相成，收获很大。