# 中国科学院大学计算机组成原理（研讨课）

# 实 验 报 告

学号：2021K8009929010 姓名：贾城昊 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。<br>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）。<br>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{自行画图，推荐用 PPT 画逻辑结构框图，复制到 word 中}、相应信号的仿真波形和信号变化的说明等）

1. reg_file:

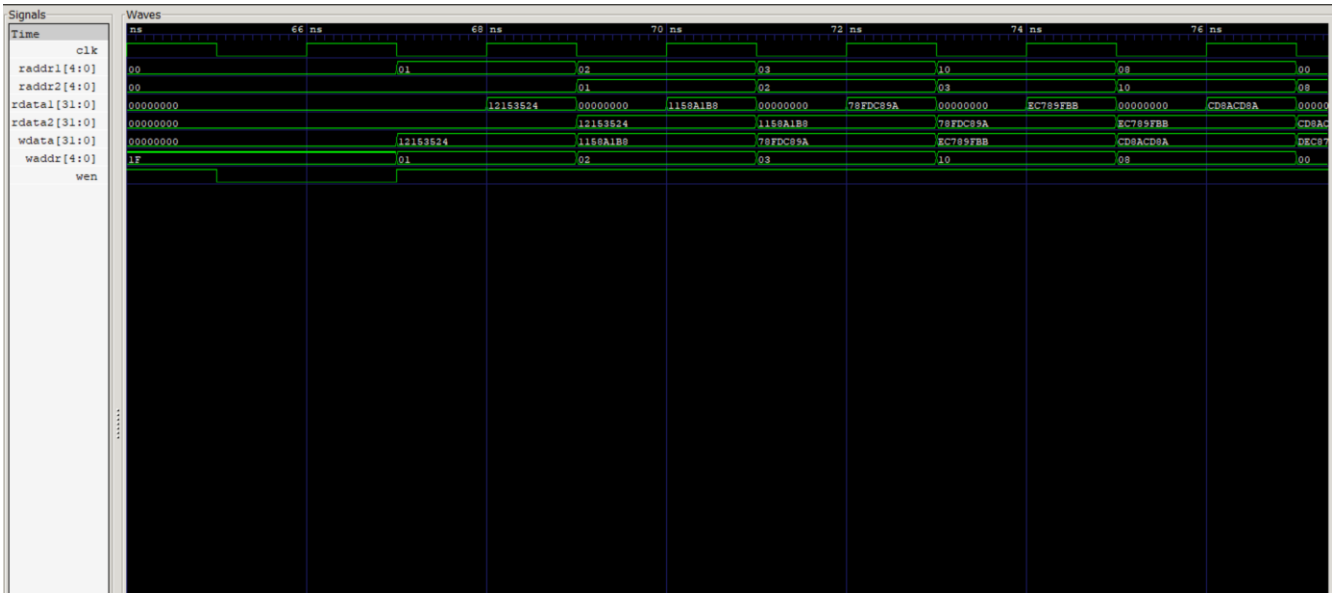
1） 关键 RTL 代码段：

```
reg [`DATA_WIDTH-1:0] rf [`DATA_WIDTH-1:0];
always @(posedge clk) begin
    if(wen && waddr) begin
        rf[waddr] <= wdata;
    end
end

assign rdata1= (raddr1)? rf[raddr1] : 32'b0;
assign rdata2= (raddr2)? rf[raddr2] : 32'b0;
```

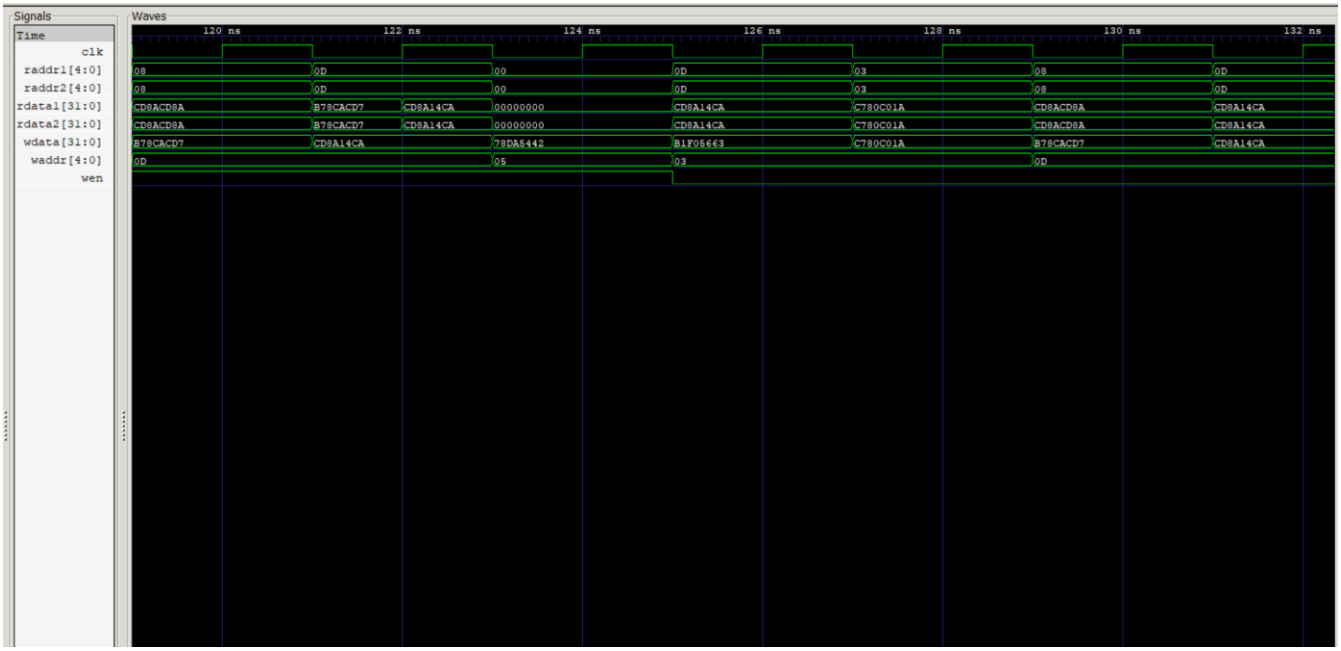
这里通过 always 语句块，使用时序逻辑对寄存器堆进行写入；通

过 assign 语句,使用组合逻辑对寄存器堆进行读取。当读地址不为 0 且 wen 有效时，对 waddr 对应的寄存器写入 wdata；同时，对 raddr1 和 raddr2 进行判断，如果不为 0，rdata1 和 rdata2 赋值为 raddr1 和 raddr2 对应的寄存器的值;如果为 raddr 为 0,对应的 rdata 赋值为 0;

2） 仿真波形图

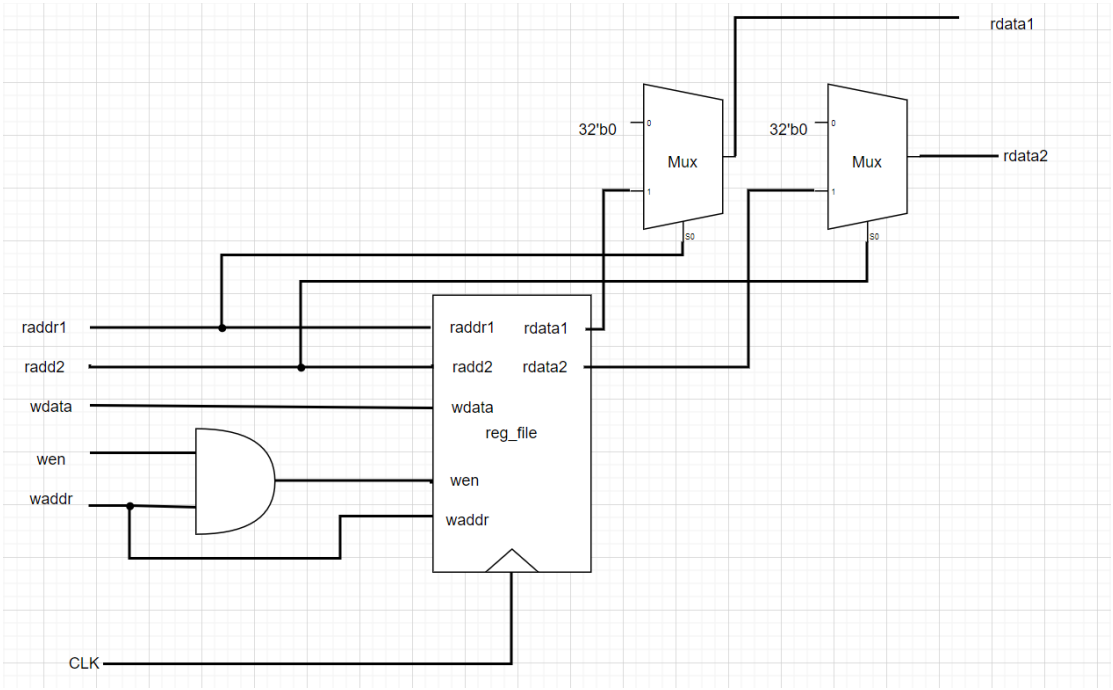



例如在 70ns 时刻,读头 1 读取了位于 01 地址的数据,此时为 00000000;与此同时,虽然wen为高电平,但此时 clk并未处于上升沿,受时序逻辑控制的写头直到clk位于上升沿的68ns时刻才会在01地址写入 12153524 的数据;而在 68ns 时刻,可以观察到由组合逻辑控制的读头 1 的读数由 00000000 立刻转为刚写入的数据 12153524;在 69ns 时刻,同样受组合逻辑控制的读头 2 也读取了更新后的地址 2 的数据。说明寄存器堆工作正常。




例如在 124ns 时候，raddr1 和 raddr2 均为 0，而此时读出的 rdata1 和 rdata2 也是 00000000，满足要求。

3） 逻辑结构框图




（注：Mux 为多路选择器）

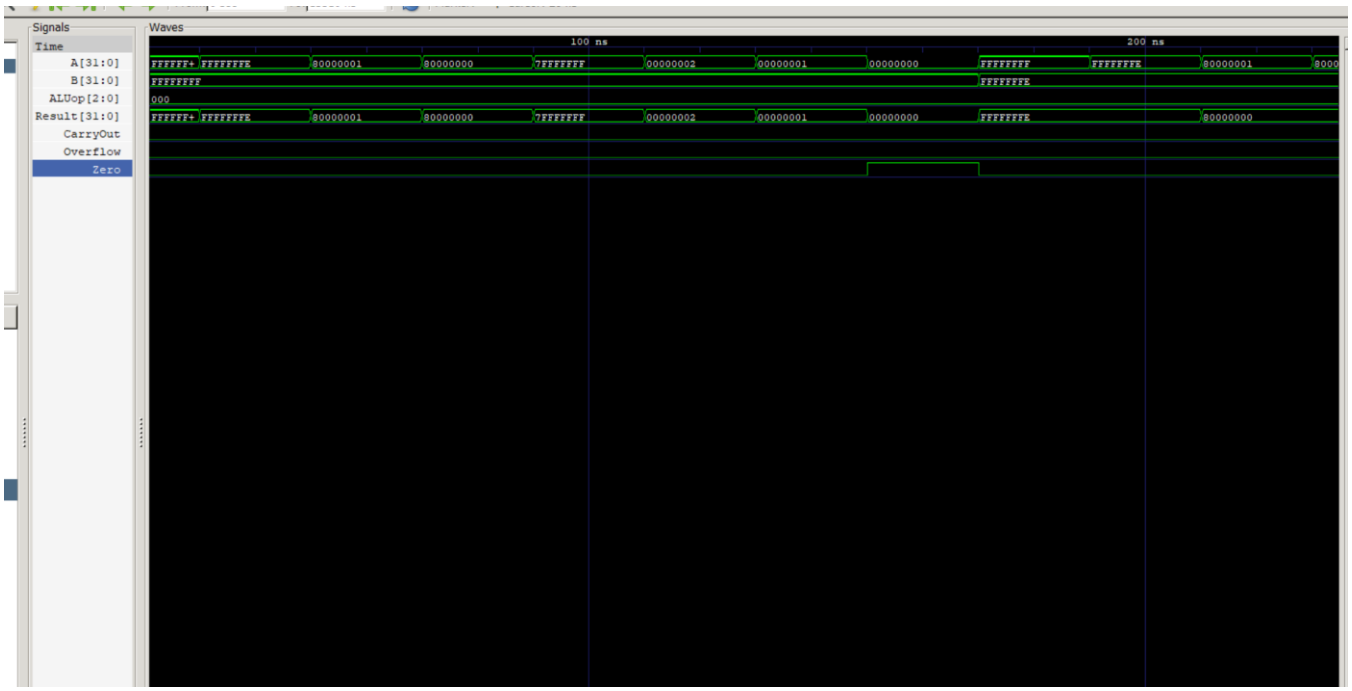
2. alu:

1） 关键 RTL 代码段：

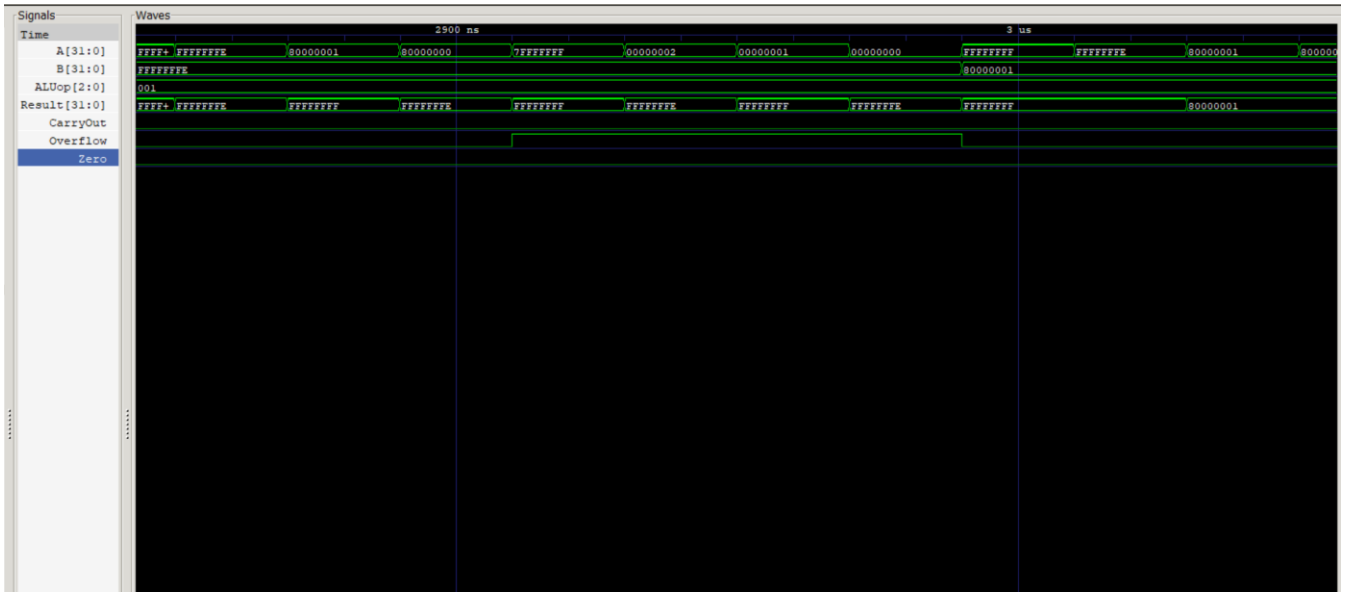
```
//控制信号译码
wire op_and = ALUop == `ALUOP_AND;
wire op_or = ALUop == `ALUOP_OR;
wire op_add = ALUop == `ALUOP_ADD;
wire op_sub = ALUop == `ALUOP_SUB;
wire op_slt = ALUop == `ALUOP_SLT;

//临时变量
wire [`DATA_WIDTH - 1:0] temp = ({`DATA_WIDTH{op_add}} & B) | ({`DATA_WIDTH{op_sub | op_slt}} & (~B));
wire cin = op_sub | op_slt;
wire [`DATA_WIDTH - 1:0] result_temp;
wire CarryOut_temp;

//计算
wire [`DATA_WIDTH - 1:0] and_res = A & B;
wire [`DATA_WIDTH - 1:0] or_res = A | B;
wire [`DATA_WIDTH - 1:0] add_res;
assign {CarryOut_temp,add_res} = A + temp + cin;

//选择结果
assign result_temp = {`DATA_WIDTH{op_and}} & and_res |
        {`DATA_WIDTH{op_or}} & or_res |
        {`DATA_WIDTH{op_add | op_sub | op_slt}} & add_res;

assign Overflow = (~A[`DATA_WIDTH - 1] & ~temp[`DATA_WIDTH - 1] & result_temp[`DATA_WIDTH - 1] |
                A[`DATA_WIDTH - 1] & temp[`DATA_WIDTH - 1] & ~result_temp[`DATA_WIDTH - 1]);

assign Result = {`DATA_WIDTH{~op_slt}} & result_temp |
        {`DATA_WIDTH{op_slt}} &  {{`DATA_WIDTH - 1{1'b0}}, Overflow ^ (op_slt & add_res[`DATA_WIDTH - 1])};

assign CarryOut = (op_add & CarryOut_temp) | (op_sub & ~CarryOut_temp);

assign Zero = (Result == 32'b0);
```

这里先对 ALUop 进行译码,然后同时生成各个操作所对应的结果,并利用译码后生成的独热码对结果进行数据选择。Carryout 和 Overflow 的具体实现逻辑以及各个临时变量的代表含义将会在第二部分会具体描述。（Zero 就是判断结果是否为 0 就行）

2） 仿真波形图

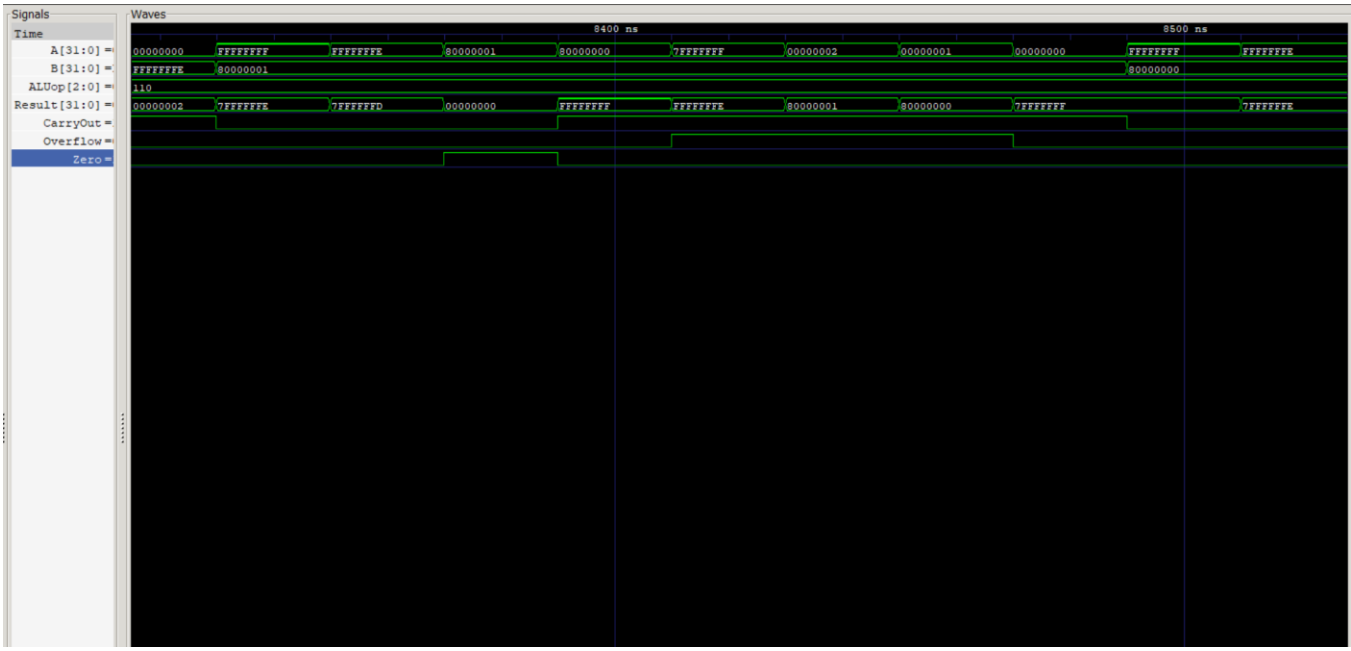



例如该图中，当 A 为 7FFFFFFF，B 为 FFFFFFFF，ALUop 为 000 时，然后 Result 为 7FFFFFFF，符合 A 与 B 的值。且当 Result 为 00000000 时，Zero 为 0。

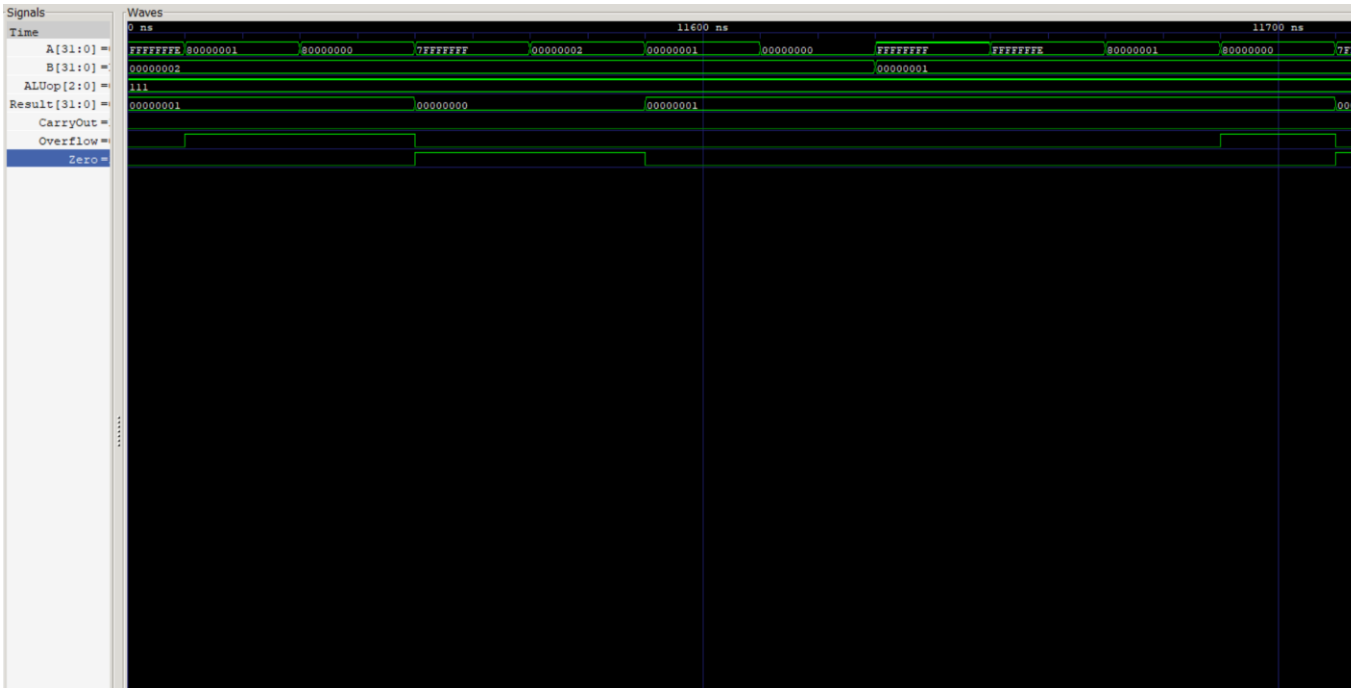



例如该图中，当 A 为 00000001，B 为 FFFFFFFE，ALUop 为 001 时，然后 Result 为 FFFFFFFF，符合 A 或 B 的值。


例如该图中，当 A 为 00000001，B 为 FFFFFFFE，ALUop 为 010 时，然后 Result 为 FFFFFFFF，符合 A+B 的值，且此时 Carryout 和 Overflow 为 0；而当当 A 为 00000002，B 为 FFFFFFFE，ALUop 为 010 时，然后 Result 为 00000000，符合 A+B 的值，且此时 Carryout 为 1，Overflow 为 0（有符号数没有溢出，但无符号数有进位）

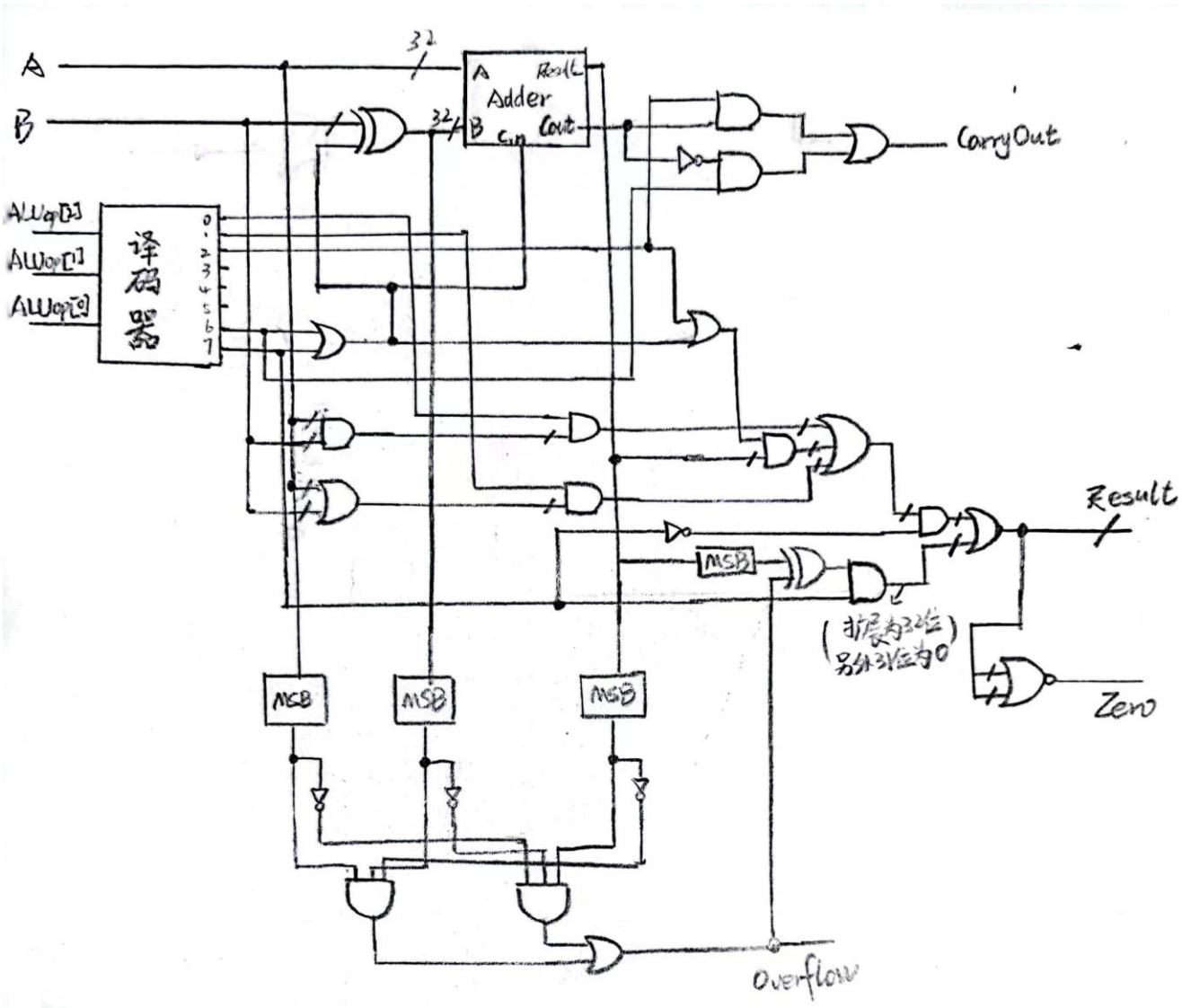


例如该图中，当 A 为 00000000，B 为 FFFFFFFE，ALUop 为 110 时，然后 Result 为 00000002，符合 A-B 的值，且此时 Carryout 为 1，Overflow 为 0（有符号数没有溢出，但无符号数有借位）；而当当 A 为 FFFFFFFE，B 为 80000001，ALUop 为 110 时，然后 Result 为 7FFFFFFD，符合 A-B 的值，且此时 Carryout 为 0，Overflow 为 0。




例如该图中，当 A 为 80000000，B 为 00000002，ALUop 为 111 时，然后 Result 为 00000001，对应 A 大于 B；而当当 A 为 7FFFFFFF，B 为 00000002，ALUop 为 111 时，然后 Result 为 00000000，对应 A 小于等于 B。

3） 逻辑结构框图




（注：上述电路图中，MSB 代表该数据的最高位，对于与或门中加了斜杠的线代表是 32 位数据线）

## 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）

本次实验的难点即出现的 bug 主要在 alu 的设计上，具体如下：

难点：

1. 如何尽可能少的使用加法器
2. 如何确定 Carryout 与 Overflow

3．如何在设计过程中避免组合逻辑环

4．对于大量的选择(如输出结果)，如何降低延迟

对难点的思考：

1．如何减少加法器的使用

首先，与和或不需要加法器，问题是怎么把加法，减法，比较都只用一个加法器。然后我想到，比较可以等价与减法，然后看符号位（虽然这一步在后面证明没有这么简单），而减法很明显可以用加上它对应的相反数的补码进行操作，所以最开始我是用一个变量存了 B 的相反数的补码。但由于求相反数补码需要逐位取反再加 1，这一步就会多出一个加法器。所以后来我就想能不能把这个求相反数的补码的加 1 用作加法器的 cin，最后统一起来。

2．如何确定 Carryout 和 Overflow

对于 Carryout，首先想到它肯定跟加法操作的时候的进位赋值有关(不如设为 Carryout_temp)，接着继续思考它的关系。在加法时候，如果溢出，那么 Carryout_temp 为 1，如果是减法，Carryout_temp 为 0，这样就可以得到 Carryout 的表达式了；而对于 Overflow，溢出的情况只有以下几种，即 A 为负，且减去一个正数或加上一个负数，但输出结果为正数；或者 A 为正数，加上一个正数或减去一个负数，但输出结果为负数。在这个基础上，就可以得到一个很简单的思路就是，如果加法器中是

两个正数相加，但结果为负数，或者两个负数相加，但结果为正数，就为溢出，这样就得到了 Overflow 的表达式。

3. 如何避免组合逻辑环

对于如何避免组合逻辑环，也是个很麻烦的事情，最简单的思路就是多定义临时变量。比如在描述 Result 时，我定义了临时变量 result_temp 用于处理不是比较的情况(因为如果比较的时候，有溢出情况，那么结果就得取反，而 Overflow 跟 Result 又有关，所以如果直接在 Result 中引用 Overflow 会出现组合逻辑环)，定义 result_temp 后，Result 就与 Overflow 和 result_temp 有关，而 Overflow 就只与 result_temp 有关了，这样就可以避免组合逻辑环了。

4. 对于大量的选择(如输出结果)，如何降低延迟

最初的设计中使用了大量三目运算符，因为不知道该如何使用单比特的数据来选择多比特的数据。在老师的启发下，发现可以通过利用 Verilog 语法特性扩展单比特数据到多比特,因此选择数据可以简化为多比特与或非运算。最终的 alu 解决方案中避免了所有三目运算和==运算。原本多层嵌套的三目运算得出结果需要经过多级延迟,使用“与或非”代替“三目”可以将串行的链式结构转换为并行的树式结构,从而降低延迟。

遇到的 bug 以及解决办法:

1. 对 0x80000000 的处理不当(它相反数的补码不存在)

解决办法：对于减法操作，先求反码，补码相较于反码的加1操作利用加法器的cin来实现，0x80000000相反数的反码是存在的，就解决了这个问题。

2. 对比较操作的输出结果没有考虑溢出的情况

解决办法：对溢出进行考虑，有溢出情况，那么结果就得取反，且为了防止生成组合逻辑环，用一个临时变量存不考虑溢出的Result。

## 三、 对讲义中思考题（如有）的理解和回答

本实验中没有思考题。

## 四、 在课后，你花费了大约___3__小时完成此次实验。

寄存器堆所花的时间较少，主要在alu设计上花的时间较长，特别如何去避免组合逻辑环，只用1个加法器的以及确定Carryout和Overflow的表达式上。

## 五、 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）

这次实验的任务初看不是特别复杂，实现一个能通过测试的电路设计并不困难,但是要设计一个高效、严谨、规范的电路仍然需深入每一个细节,经过仔细的思考（如对组合逻辑环的考虑，对延迟的考虑，对加法器

数量的限制)。

通过这次实验,我受益匪浅：一方面，对寄存器堆和 alu 有了更深的理解。另一方面，通过这次实验，我进一步熟悉了很多 Verilog 语法上的特性，例如逐位逻辑运算(用于判断一个多位数据是否为零)， 如何对位数进行扩展(用于 1 位选择多位数据)，如何把加法器的进位（即最高位）赋值给一个变量。在以往的数字电路课上,往往只会注意电路的结构上的实现,而没有特别关注 Verilog 的一些行为上的描述。然而在实际设计过程中,这些语法是不可或缺的，而且十分方便、高效。所以说通过这次实验，我大大加深了硬件描述语言特性的理解。另外，在撰写实验报告的时候，我了解了如何用 Vscode 的 draw.io 的插件画电路图（但 ALU 的电路图有点复杂，所以选择了手画)。

另外,感谢舍友李金明在 alu 设计上的启发与帮助。在本次实验中，我和他讨论了很多关于细节的优化的方式以及输出的更简单的表达方式。