中国科学院大学
《计算机组成原理(研讨课)》实验报告

姓名 薛翼舟 学号 2023K8009929044 专业 计算机科学与技术

实验项目编号 1 实验名称 寄存器堆 REGFILE 与运算器 ALU

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` 推送到实验课 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(比如 Verilog HDL 关键代码段 {包含注释} 及其对应的逻辑电路结构图 {自行画图, 推荐用 PPT 画逻辑结构框图后保存为 PDF, 再插入到 LaTeX. 中}、相应信号的仿真波形和信号变化的说明等)

1 寄存器设计

1.1 Verilog 关键代码段

```
`define DATA_WIDTH 32
`define ADDR_WIDTH 5

module reg_file(
  input                   clk,
  input [`ADDR_WIDTH - 1:0] waddr, //写地址
  input [`ADDR_WIDTH - 1:0] raddr1, //读地址1
  input [`ADDR_WIDTH - 1:0] raddr2, //读地址2,有两个读地址信号,因为读2写1
  input                   wen, //写使能信号
  input [`DATA_WIDTH - 1:0] wdata, //写数据
  output [`DATA_WIDTH - 1:0] rdata1, //读数据1
  output [`DATA_WIDTH - 1:0] rdata2 //读数据2
);
  reg [`DATA_WIDTH - 1:0] myreg [`DATA_WIDTH - 1:0]; //用于定义一个32*32的寄存器堆

  //同步写,采用时序逻辑
  always @(posedge clk) begin
    //check wen and waddr
    if(wen && waddr)begin
      myreg[waddr] <= wdata;
```

```
    end
  end

  //异步读,采用组合逻辑
  //read1, set 0 if address = 0
  assign rdata1 = (raddr1 == 0)? 32'h00000000 : myreg[raddr1];
  //read2, set 0 if address = 0
  assign rdata2 = (raddr2 == 0)? 32'h00000000 : myreg[raddr2];
endmodule
```

在本实验项目的代码中, 主要有如下要点

1. 要注意当读、写地址不为 0 时, 才能进行读、写操作, 否则读出（写入）的数据为 0，
2. 在本实验中，采用同步写、异步读，于是在读逻辑的代码块中要采用组合逻辑，在写逻辑的代码块中要采用时序逻辑。

1.2 逻辑电路结构图

1.3 部分仿真波形

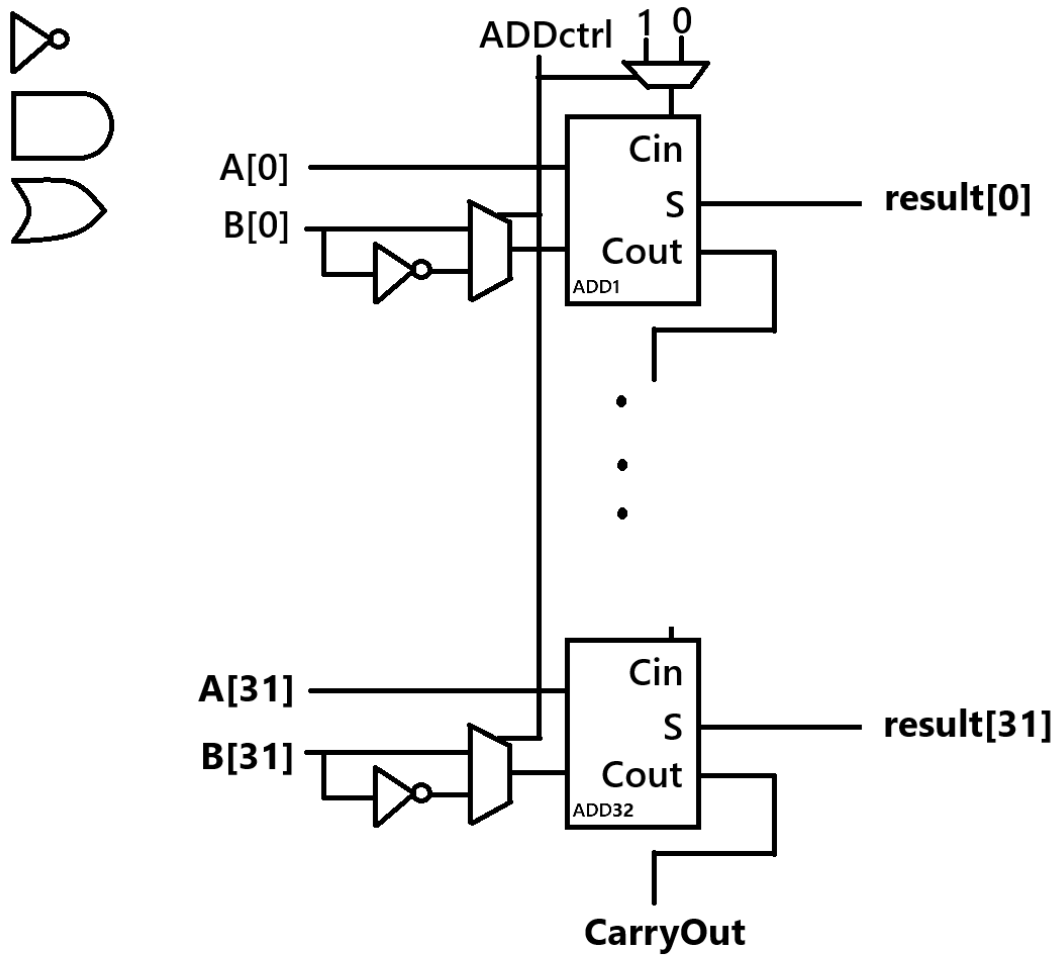

图 1: 寄存器堆设计的仿真波形

观察并分析波形可以发现，只有当 clk 上升沿中，wen 为 1 时，waddr 对应的寄存器才会被写入数据 wdata。而读数据时与 clk 无关，属于组合逻辑电路，另外在这部分波形中也可以体现本项目中当读地址为 0 时，读出的数据为 0 的特性。

1 ALU 设计

1.1 Verilog 关键代码段

```
    //define the operation
`define DATA_WIDTH 32
```

```
`define AND 3'b000
`define OR 3'b001
`define XOR 3'b100
`define NOR 3'b101
`define ADD 3'b010
`define SUB 3'b110
`define SLT 3'b111
`define SLTU 3'b011

module alu(
   input [`DATA_WIDTH - 1:0] A,
   input [`DATA_WIDTH - 1:0] B,
   input [            2:0] ALUop,
   output                  Overflow,
   output                  CarryOut,
   output                  Zero,
   output [`DATA_WIDTH - 1:0] Result
);

   wire [31:0] temp_sub;
   assign temp_sub = A + ~B + 1;

   assign {CarryOut,Result} = (ALUop == `AND) ? A & B:
              (ALUop == `OR ) ? A | B:
              (ALUop == `XOR) ? A ^ B:
              (ALUop == `NOR) ? {0,~(A | B)}:
              (ALUop == `ADD) ? A + B:
              (ALUop == `SUB) ? A + ~B + 1:
              (ALUop == `SLT) ? (({A[31],B[31]} == 2'b10) ? 1 :
                        ({A[31],B[31]} == 2'b01) ? 0 :
                       (temp_sub[31] == 1) ? 1: 0) :
              (ALUop == `SLTU) ? (temp_sub[31] == 1) :
              0;

   assign Overflow = (ALUop == `ADD) ? ((A[31] && B[31] && ~Result[31]) || (~A[31] && ~B[31] &&
       Result[31])) :
          (ALUop == `SUB) ? ((A[31] && ~B[31] && ~Result[31]) || (~A[31] && B[31] && Result[31])) :
          0;
        //overflow in sub : +- -> -; -+ -> +
        //overflow in add : ++ -> -; -- -> +

   assign Zero = (Result == 0);
endmodule
```

在本实验中，我将 op 对应的几种功能放在了模块外边，方便更改，也希望这样能让代码可读性更好本实验的几个要点如下

1. 所有的运算、赋值都要采用组合逻辑
2. 要特别注意有符号数的大小比较

3. 要注意溢出、进位信号所代表的含义已经他们在不同的输入下所产生的情况，例如进位信号可以作为借位信号来当作比大小的信号，溢出出现在两个负数相加时，结果为正数，或者两个正数相加时，结果为负数

4. 若两个有符号数的符号相同，则它们相减，最高位的符号位即为 result 的结果

5. 另外在 SLT 的功能实现中出现了特殊的问题，会在下一板块中具体描述

在本实验中，直接采用了拼接向量的办法来同时为 CF 和 result 来赋值，减少了代码量
我对于溢出的判断是采用了一个组合逻辑电路来实现的，首先要判断 ALUop 是否是 SUB 或者 ADD，之后判断两种情况，核心方法在于比较 A，B 以及 result 的最高位的情况，枚举出了每种溢出出现的可能情况

1.2 逻辑电路结构图

由于逻辑操作的电路图比较显然，因此在这里只给出加减法以及比较基于一位全加器所对应的逻辑电路图

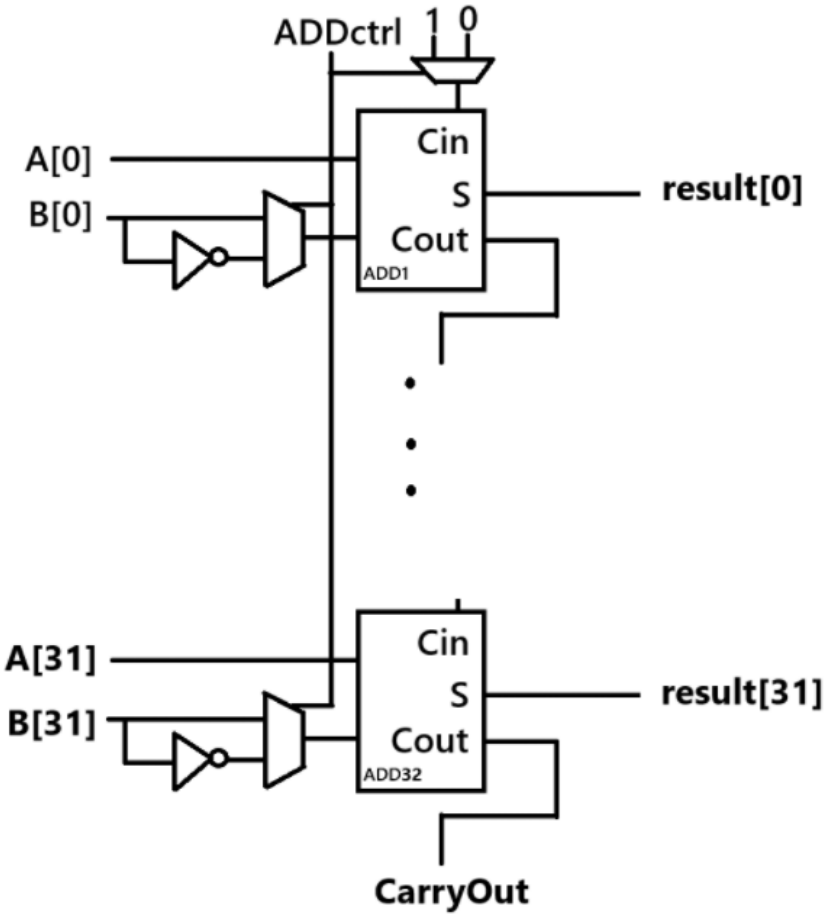

图 2: ALU 设计的部分逻辑电路结构图

在这个图中，我们可以看到 ADDctrl 作为控制信号，用于选择电路是计算 A+B 还是 A+ B+1，也就是 A-B，并且利用结果来计算 SLT 以及 SLTU

1.3 部分仿真波形

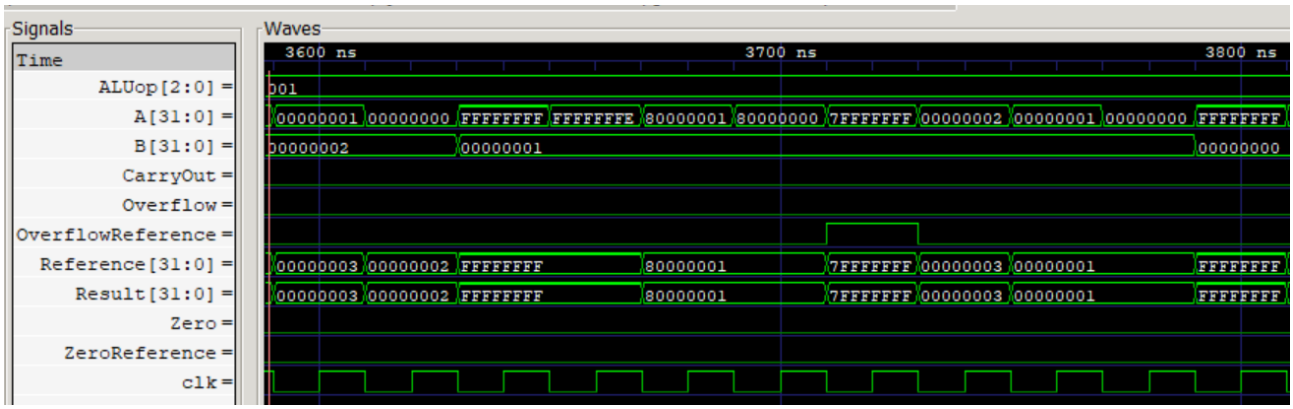

图 3: ALU 设计的 OR 功能仿真波形

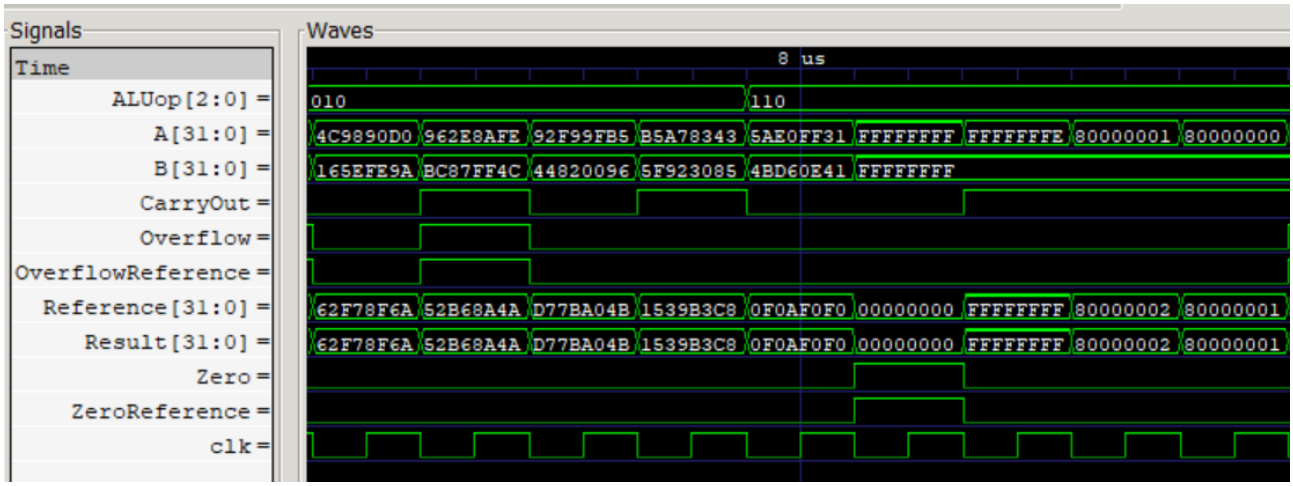

图 4: ALU 设计的 ADD 功能仿真波形

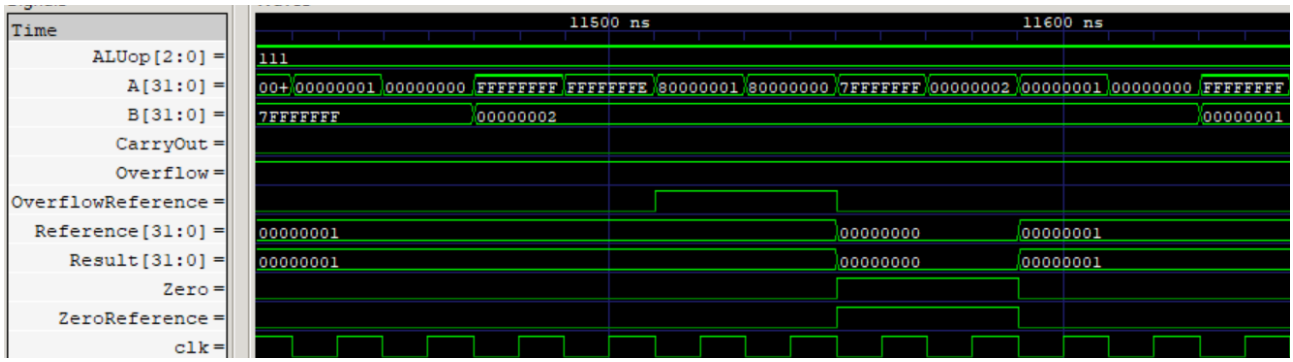

图 5: ALU 设计的 SLU 功能仿真波形

选取了几个具有代表性的功能的波形，可以看出 alu 的功能实现基本符合我们的预期

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）

2 寄存器堆设计问题

寄存器堆的设计过程中并没有遇到较大的问题，不过在本次实验中收获最大的是回忆了

```
`define DATA_WIDTH 32
`define ADDR_WIDTH 5
```

此类宏定义的使用，在减少修改成本的同时也增加了代码可读性

```
reg [`DATA_WIDTH - 1:0] myreg [`DATA_WIDTH - 1:0]; //用于定义一个32*32的寄存器堆
```

这种定义多个寄存器堆的写法

2 ALU 设计问题

在本次实验中，我一开始使用的是 always@(*) 的写法来写组合逻辑，这种写法确实更符合之前所学习的其他诸如 C 一样的编程语言，但是作为 RTL 代码，这样写出来的 verilog 程序事实上是不可综合的，所以在组合逻辑应该用如下两种 assign 语句的写法

1. 逻辑计算方法

```
assign result = (A && selectA) || (B && selectB);
```

2. 三目运算符方法

```
assign result = (selectA) ? A :
                (selectB) ? B :
                0;
```

另外，在本实验中的 SLT 一开始设计出现了问题，错误代码如下

```
SLTresult = (temp_sub[31] == 1) ? 1: 0;
```

这段代码没有考虑有符号数，将有符号数当作无符号数来处理，这样所得到结果在两个操作数符号相同时是正确的，但是在两个操作数符号不同时，会出现比较愚蠢的错误，例如任意一个整数都小于所有负数，于是修正代码如下

```
SLTresult = (({A[31],B[31]} == 2'b10) ? 1 :
            ({A[31],B[31]} == 2'b01) ? 0 :
            (temp_sub[31] == 1) ? 1: 0) :
            0;
```

另外，在本实验中所出现的另一个问题是对于向量的使用和声明，在 SLT 的计算中，我运用到了 sub 的结果的符号位来判断结果，一开始所写的错误代码是

```
(A + ~B + 1)[31];
```

但这样会产生行为错误，于是需要声明一个新的 wire 来计算这个减运算的值，在之后的计算中使用

```
wire [31:0] temp_sub;
assign temp_sub = A + ~B + 1;
temp_sub[31];
```

三、 实验所耗时间

在课后，你花费了大约____6____ 小时完成此次实验。