中国科学院大学
《计算机组成原理(研讨课)》实验报告

姓名 宋俊仪 学号 2023K8009929018 专业 计算机科学与技术

实验项目编号 1 实验名称 寄存器堆和 ALU 实现

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

- **寄存器堆设计。**

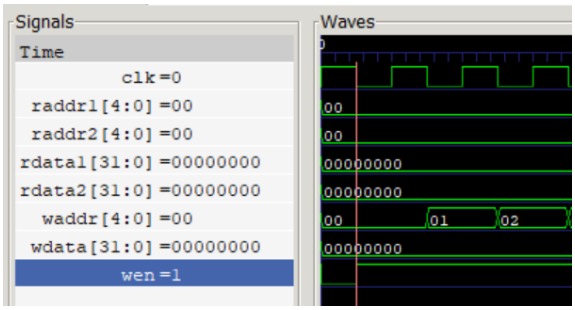
按照要求: 1. 寄存器堆需要两个读口,一个写口
2. 零号寄存器读出的值总是 32'b0
3. 利用写使能信号 wen, 当 wen=1 且 waddr 不等于 0 时, 才可以向对应的寄存器写入 wdata

使用 if 语句进行判断,当 wen 和 waddr 不为 0 时写入;使用三元运算符,确保当 raddr=0 时,读取总是 32'b0(见图 1)

```
1  `timescale 10 ns / 1 ns
2
3  `define DATA_WIDTH 32
4  `define ADDR_WIDTH 5
5
6  module reg_file(
7      input                       clk,
8      input  [`ADDR_WIDTH - 1:0]  waddr,
9      input  [`ADDR_WIDTH - 1:0]  raddr1,
10     input  [`ADDR_WIDTH - 1:0]  raddr2,
11     input                       wen,
12     input  [`DATA_WIDTH - 1:0]  wdata,
13     output [`DATA_WIDTH - 1:0]  rdata1,
14     output [`DATA_WIDTH - 1:0]  rdata2
15 );
16
17     reg [`DATA_WIDTH - 1:0] reg_file [(1<<`ADDR_WIDTH)-1:0];
18
19     always @(posedge clk)begin
20       if(wen == 1 && waddr !=0)begin
21         reg_file[waddr]<=wdata;
22         end
23     end
24
25   assign rdata1 = (raddr1 == 5'b0) ? 5'b0 : reg_file[raddr1];
26   assign rdata2 = (raddr2 == 5'b0) ? 5'b0 : reg_file[raddr2];
27
28 endmodule
```

图 1: reg_file 代码

仿真运行波形图如下:(见图 2)



(a) 零时刻波形图()

(b) 运行结果图

图 2: 波形图示例

逻辑电路设计：1.waddr 与 0 或，再与 wen 与，将最终的值作为写使能信号，保证写入数据时必须满足的两点要求 2.raddr 与 0 或，再与直接读取的数据与，保证了 raddr=1 时，读出数据恒为 0（见图 3）

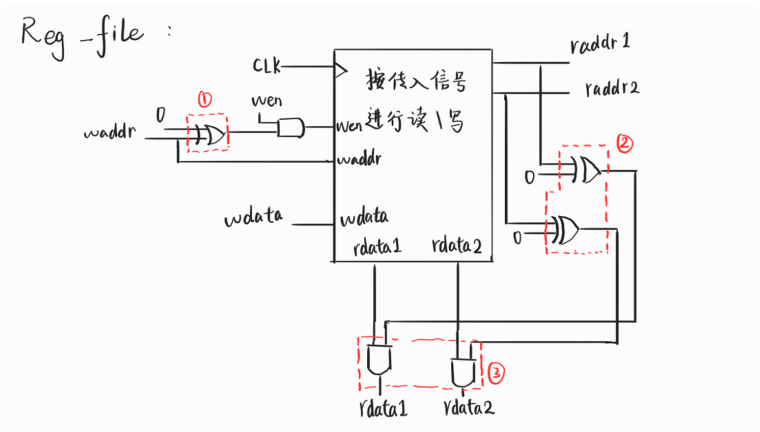


图 3: reg_file 逻辑电路简图

- **ALU 设计。**

按照要求: 1. 输入两个 32 位操作数，以及一个控制信号，利用控制信号 ALUop，选择加法、减法、按位与、按位或、比较大小五种

2.Overflow 信号，在有符号数进行加减法运算且有溢出时置 1，反之为 0

3.Carryout 信号，在无符号数进行加减法运算且有借位和进位时置 1，反之为 0

4. 运算结果全为 0 时，输出 Zero 信号

设计思路：

1. 按位与、按位或可以直接计算。
2. 通过分析，可以发现：对于有符号数和无符号数，其加减法运算并无差异，所以不需要区分，但是 Carryout 和 Overflow 信号是要区分有符号数和无符号数的。
3. 对于减法，采用加上这个数的“取反 +1”，把减法转化成加法。
4. 由于要求只能利用一个全加器完成 ALU 设计，所以我采用了加减法都对应同一个 Result，但是 Result 的计算会受到 ALUop 的控制，这样就可以只用一个全加器完成加减法操作。
5. 为了可以正确得到 Carryout 和 Overflow，可以通过在两个操作数前补一位 0 来实现。

6. 实现设计思路的逻辑电路简图如下（见图 4）。

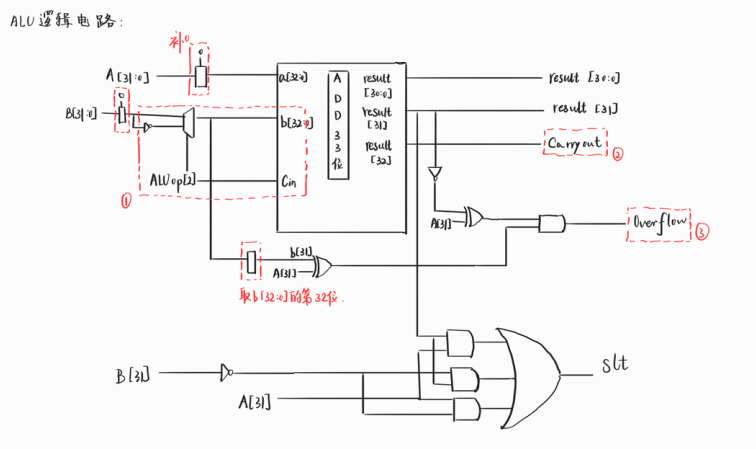


图 4: ALU 逻辑电路简图

ALU 具体代码如下：见图 5~9

cod-lab > fpga > design > ucas-cod > hardware > sources > alu > alu.v

```
`timescale 10 ns / 1 ns

`define DATA_WIDTH 32

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 [`DATA_WIDTH - 1:0] and_result;
    wire [`DATA_WIDTH - 1:0] or_result;
    wire [`DATA_WIDTH :0] add_result;
    wire slt_result;

    wire [`DATA_WIDTH:0] a = {1'b0, A};
    wire [`DATA_WIDTH:0] b = (ALUop[2]==1)?~{1'b0,B}:{1'b0,B};

    assign and_result = A&B;
    assign or_result = A|B;
    assign add_result = a+b+ALUop[2];
    assign slt_result = (A[31] == 1 && B[31] == 0)? 1 :
    (A[31] == 0 && B[31] == 1) ? 0 :
    (add_result[31] == 1) ? 1 : 0;

/*output*/
    assign Overflow =(ALUop == 3'b010||ALUop == 3'b110) ? (A[31]==b[31])&&(add_result[31]!=A[31]):1'b0;

    assign Result = (ALUop == 3'b000) ? and_result :
    (ALUop == 3'b001) ? or_result :
    (ALUop == 3'b010||ALUop == 3'b110) ? add_result[`DATA_WIDTH - 1:0] :
    (ALUop == 3'b111) ?  (slt_result ? 32'b1 : 32'b0) : 32'b0;
    assign CarryOut = add_result[`DATA_WIDTH];

    assign Zero = (Result==32'b0)? 1 : 0;

endmodule
```

图 5: ALU 代码

代码具体解释：1.（见图 6）补 33 位的零，方便记录是否有进位借位，可以利用 result 的 33 位得到 Carryout 和 Overflow，并且

2.（见图 7）按位与、按位或和加法运算直接进行，不做解释；关于减法，还是将其转化成加法，对减数采取取反加

3.（见图 8）Cayyout 信号直接与结果的 33 位相连，对于进位，考虑“正数 + 正数”和“负数 + 负数”；正数的 33

4.(见图 9) 对于 Overflow 信号，直接按照理论课讲义中的溢出判断即可：如果两个操作数符号相同且它们的结果

```
wire [`DATA_WIDTH:0] a = {1'b0, A};
wire [`DATA_WIDTH:0] b = (ALUop[2]==1)?~{1'b0,B}:{1'b0,B};
```

图 6: 补高位 0

```
assign and_result = A&B;
assign or_result = A|B;
assign add_result = a+b+ALUop[2];
assign slt_result = (A[31] == 1 && B[31] == 0)? 1 :
(A[31] == 0 && B[31] == 1) ? 0 :
(add_result[31] == 1) ? 1 : 0;
```

图 7: 五种运算代码

```
assign CarryOut = add_result[`DATA_WIDTH];
```

图 8: Carryout 信号

```
assign Overflow =(ALUop == 3'b010||ALUop == 3'b110) ? (A[31]==b[31])&&(add_result[31]!=A[31]):1'b0;
```

图 9: Overflow 信号

ALU 运行波形图示例如下：见图 10~15

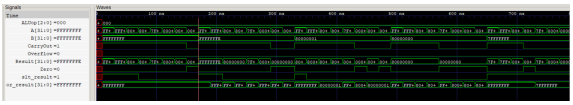


图 10: ALU 波形图示例

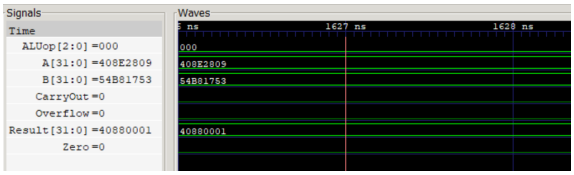


图 11: ALU 按位与波形图示例

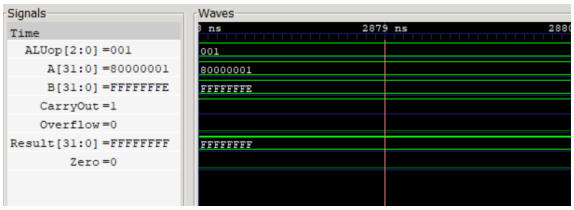


图 12: ALU 按位或波形图示例

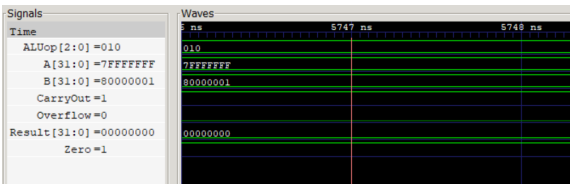


图 13: ALU 加法波形图示例

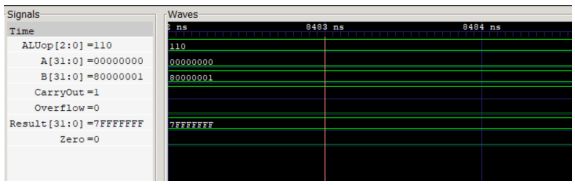


图 14: ALU 减法波形图示例

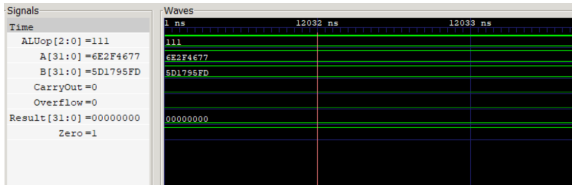


图 15: ALU 比较大小波形图示例

- **其他关键模块设计**。

  本次实验暂无

二、 实验中遇到的难点，debug 过程，思考和解决方法

- **寄存器堆设计问题**。太久没有使用 verilog 语言，对语法不够熟悉，一开始没有使用三元运算符，而是使用了 if 语句判断，导致语法错误，assign 模块中不可以使用 if 语句，复习 verilog 语法修改后通过测试。

- **ALU 设计问题**。

  设计 ALU 遇到了一下的一些问题：1. 按照讲义：当符号位进位和有效最高位进位不同时产生溢出，于是想当然的用 result 的 33 位和 32 位来表示 Overflow 信号，错误原因可以通过图 17 这个例子看出；

```
assign CarryOut = add_result[32];
    assign Overflow = add_result[32]^add_result[31];
assign Zero = (Result==32'b0)? 1 : 0;
```

图 16: 错误的 Overflow 信号

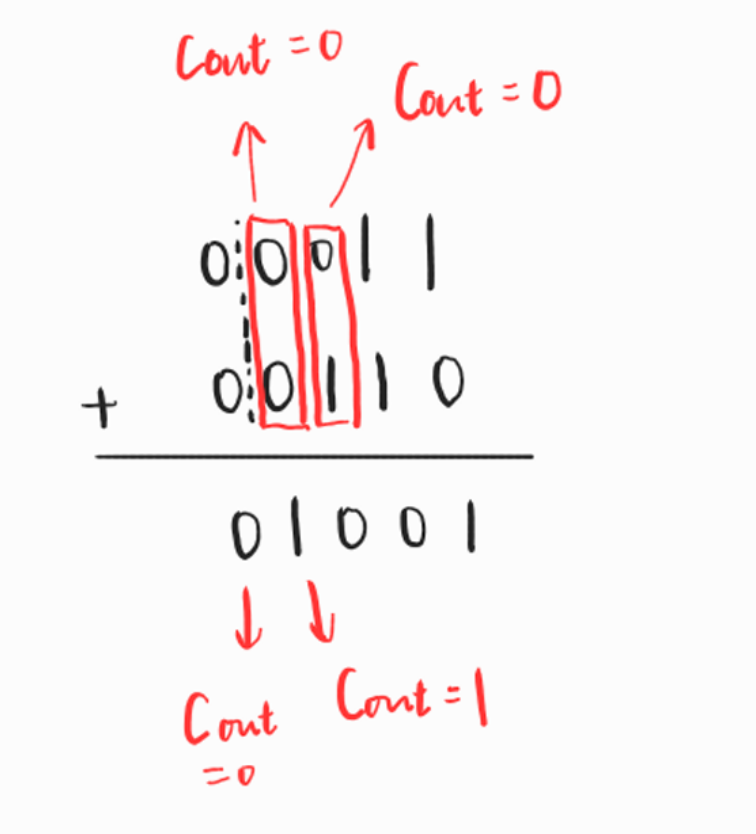


图 17: Overflow 错误例子

2.（图 18）按照理论课讲义中的溢出判断即可：如果两个操作数符号相同且它们的结果符号位与原操作数的符号位不同，那么有溢出，Overflow=1，反之为 0；但是由于马虎，忘记了减法时，这里的第二个操作数应该是补码，所以应该使用取反后的 b，而不是 B；修改后通过测试。

```
assign Overflow =(ALUop == 3'b010||ALUop == 3'b110) ? (A[31]==B[31])&&(add_result[31]!=A[31]):1'b0;
assign Overflow =(ALUop == 3'b010||ALUop == 3'b110) ? (A[31]==b[31])&&(add_result[31]!=A[31]):1'b0;
```

图 18: Overflow 代码修改

3. 通过观察 ALUop 控制不同操作的指令码，我发现了可以直接利用 ALUop[2] 来完成对第二个操作数的变形。

```
wire [`DATA_WIDTH:0] a = {1'b0, A};
wire [`DATA_WIDTH:0] b = (ALUop[2]==1)?~{1'b0,B}:{1'b0,B};
```

图 19: 创新点（自己认为的，可能也不算是创新点）

- **其他关键模块设计问题。**

本次实验暂无

三、 对讲义中思考题（如有）的理解和回答

本次实验暂无

四、 实验所耗时间

在课后，你花费了大约 12（代码完成 4 小时，学习使用 latex 完成实验报告 8 小时） 小时完成此次实验。