中国科学院大学
《计算机组成原理(研讨课)》实验报告

姓名 朱夏楠 学号 2022K8009929031 专业 计算机科学与技术

实验项目编号 1 实验名称 基本功能部件设计

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)

• 寄存器堆设计。

此次实验的关键点在于将寄存器堆的读和写功能进行区分, 写功能受使能信号的控制, 为时序逻辑; 读功能则为组合逻辑。

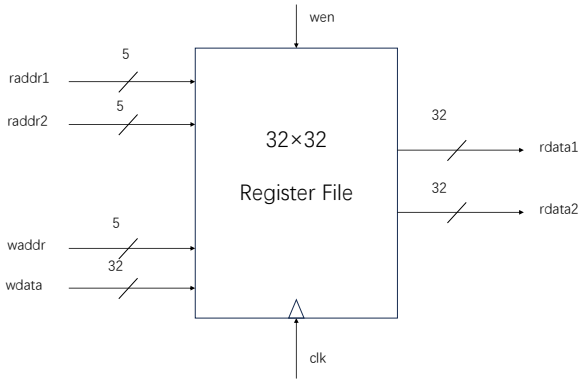

图 1: 寄存器堆逻辑电路结构

经过仿真模拟得到的波形如下,

图 2: reg_file.v 仿真波形图

从波形中可以看到，在 549ns 时，$raddr2$ 值为零，从零号寄存器中取数，得到的结果为 $rdata2 = 0$，在 553ns 时，写使能信号为高电平，向 5 号寄存器中写入数据，而在 561ns 时从 5 号寄存器中进行数据读取，得到的结果正是之前存入的值。

- **ALU 设计**。

ALU 内部为纯组合逻辑，ALU 分为按位与、按位或、加法器三个大部件，在 ALU 运算时，这三部分同时进行运算，并将计算结果输入到多路选择器中，根据操作码对最终输出值进行控制。

本次实验中 ALU 的加法部件主体为 32 个一位全加器组成的 32 位串行进位加法器，可进行两个 32 位数的加法和减法，并利用结果进行数的比较和标志位的置位。

由于计算机中数据按照补码的形式进行储存，计算 $[A]_{补} - [B]_{补} = [A]_{补} + [-B]_{补}$，因此 32 位加法器进行减法时，要对作为减数的 B 进行按位取反并加 1 的操作，在 ALU 的实现中，可通过操作码控制 2 选 1 选择器，需要进行减法时输出 B 取反后的值，并给加法器的进位输入赋值为 1。

```
//when the op is sub, get the 1's complement code of B
assign complement_B = B ^ {32{ALUop[2]}};
//add
assign {cout,sum} = {1'b0,A}+{1'b0,complement_B}+{32'b0,ALUop[2]};
```

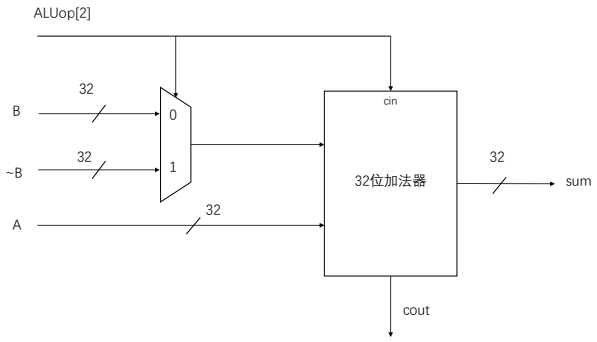

图 3: 根据 ALUop 控制 B 输入到加法器中的值

从以下的波形图中可以看出,在 $ALUop$ 变化时,得到的结果均正确。

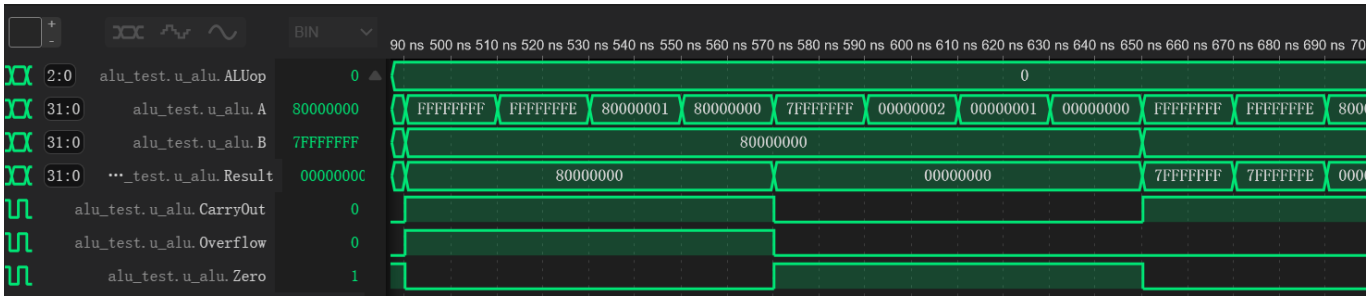

图 4: ALUop 为 000 时波形图

当 ALUop 为 000 时,做按位与操作,在 $570ns$ 时,$7FFFFFFF$ 和 80000000 进行按位与,结果为 $32h'0$,与预期结果相同

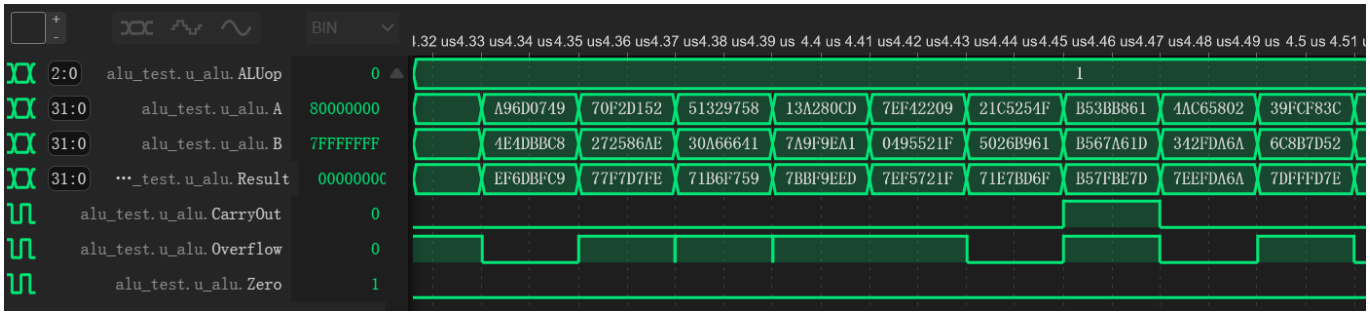

图 5: ALUop 为 001 时波形图

当 ALUop 为 001 时,做按位或操作,$4.37\mu s$ 时,两个输入进行按位或,所得结果和预期相同。

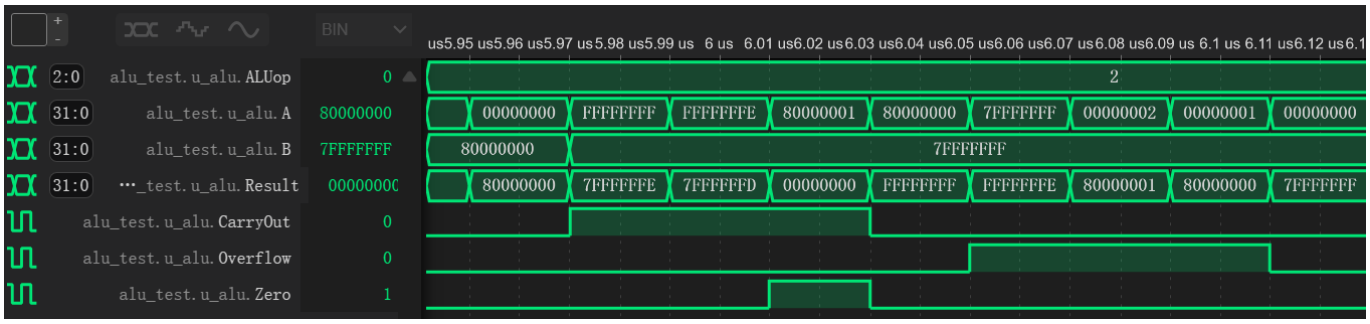

图 6: ALUop 为 010 时波形图

当 ALUop 为 010 时，做加法操作，在 6.01μs 时，$A = 80000001, B = 7FFFFFFF$，将 A 和 B 视为无符号数，则相加发生进位，$CarryOut$ 置位为 1，将其视为有符号数，则两数互为相反数，相加得零；在 6.07μs 时，$A = 00000002, B = 7FFFFFFF$，将两者视为有符号数，相加发生溢出，$OverFlow$ 被置位为 1.

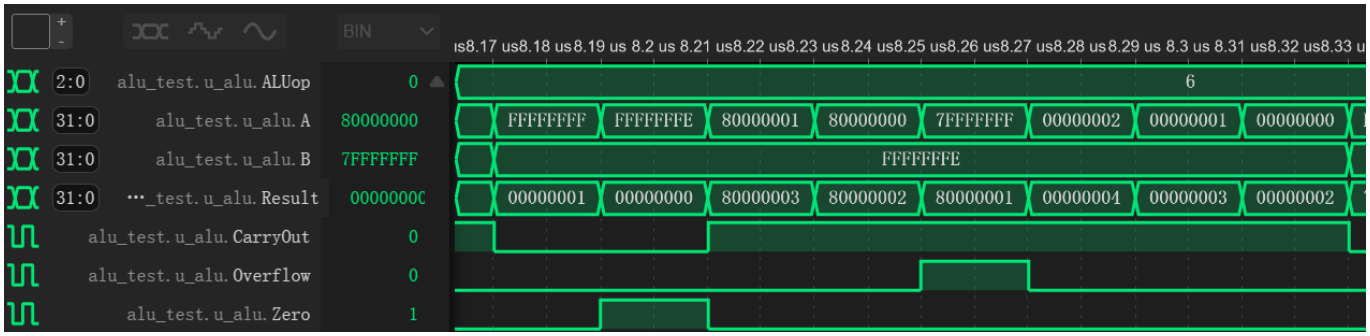

图 7: ALUop 为 110 时波形图

当 ALUop 为 110 时，做减法操作，在 8.25μs 时，$A = 7FFFFFFF, B = FFFFFFFE$，将两值视为有符号数，则 $A - B$ 产生溢出，$Overflow$ 置位为 1，将两值视为无符号数，则 $A - B$ 发生借位，$CarryOut$ 置位为 1。

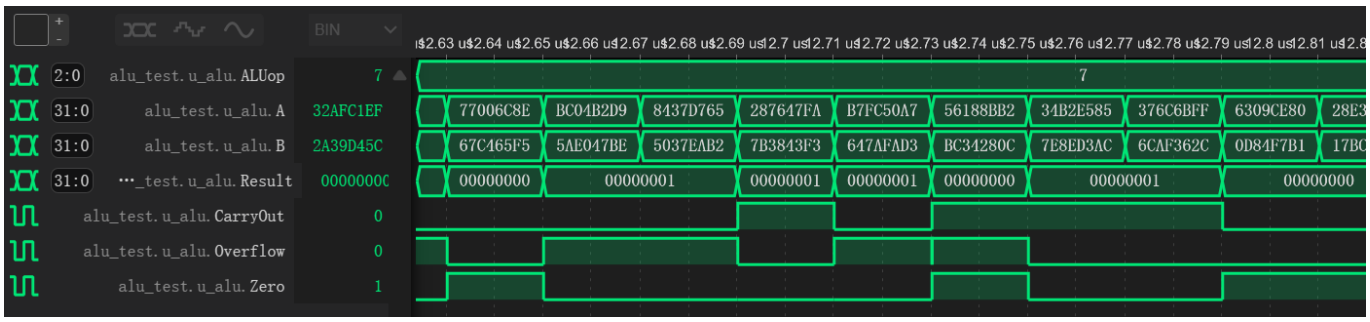

图 8: ALUop 为 111 时波形图

当 ALUop 为 111 时，做减法操作并比较，在 2.7μs 时，$A = 287647FA, B = 7B3843F3$，将两值视为有符号数，$A > B$，结果为 00000001，在 2.73$\mu s$ 时，$A = 56188BB2, B = BC34280C$，将两值视为有符号数，$A > B$，结果为 00000000。

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)

- **寄存器堆设计问题。**

在设计寄存器堆时,我最开始读数据时使用的是三目运算符进行赋值,随后在重新检查代码时,改为使用逻辑运算。我的思路是用位运算实现类似选择器的功能,从寄存器堆中读取数据,在读地址为 0 号寄存器时,数据与 32 位全 0 进行按位或运算,最终结果一定为 0;在读地址为其它寄存器时,数据与 32 位全 1 进行按位或运算,最终结果仍为从寄存器中读取的数据。为此,需要对读取到的地址的每一位进行或运算,当地址为 0 时,得到结果为 0,在其它情况下,得到结果为 1。再对其进行拼接,可得到 32 位的全 0 或全 1。

```
//when raddr is 5'b0, assign rdata 32'b0
assign rdata = RegisterData[raddr] & {32{|raddr}};
```

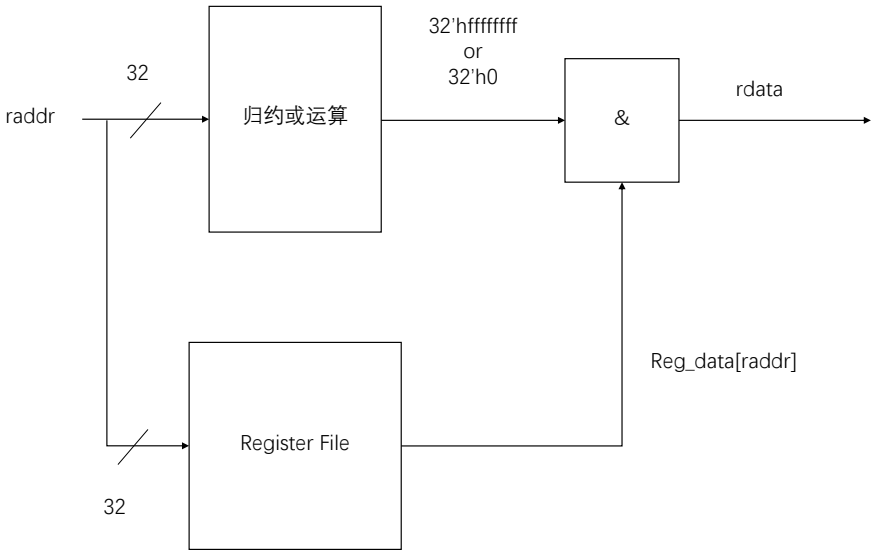

图 9: 使用逻辑运算实现读零号寄存器输出 0

- **ALU 设计问题。**

在实验过程中,我对无符号数的 $CarryOut$ 标志位如何正确置位理解的不够准确,对于无符号数减法时的借位情况感到困惑,之后我考虑将 32 位无符号数运算看作 33 位有符号数的运算,这样一来,在补码表示下,第 33 位的符号位为 1 时,可以认为两个无符号数运算后出现了借位和进位的情况。

```
assign {cout,sum} = {1'b0,A}+{ALUop[2],complement_B}+{32'b0,ALUop[2]};
assign CarryOut = cout;
```

亦可以看成 32 位加法器运算后的进位结果再与 $ALUop[2]$ 进行异或操作。

```
assign {cout,sum} = {1'b0,A}+{1'b0,complement_B}+{32'b0,ALUop[2]};
assign CarryOut = cout ^ ALUop[2];
```

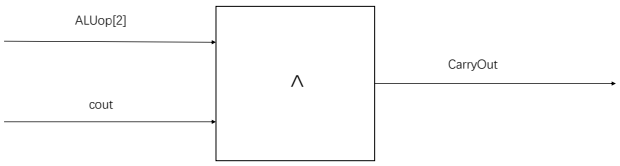

图 10: 根据加法器结果对 CarryOut 置位

三、 对实验内容的思考

在设计 ALU 的实验中,加法器可以更进一步用门级电路搭建并行进位加法器。

四、 实验所耗时间

在课后,你花费了大约____4____ 小时完成此次实验。