# 中国科学院大学
# 《计算机组成原理(研讨课)》实验报告

姓名 韦欣池 学号 2022K8009907004 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——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。 |
| 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**。

寄存器堆设计实验要求的是设计 32 个 32bit 的寄存器。具体要求为仅当 wen=1(有效)且 waddr 不等于 0 时, 才可以向 waddr 对应的寄存器写入 wdata。

对于寄存器的设计, 课件中已经给出了一个例子(8 个 1bit 寄存器组成的寄存器堆), 因此大部分的内容我们仅需仿照这个例子进行即可。需要注意的是: 我们需要通过时序逻辑电路控制数据的写入, 而实验要求只有 wen 信号为 1 时且 waddr 不为 0 时才能够进行写入, 因此我们需要在时序电路中加入一个判断, 即使用 if 条件判断来决定什么时候写入数据; 另外, 读出数据是不需要时序控制的, 所以应该用组合逻辑电路进行控制。实验要求从 0 号地址读出的值总是常量 32'b0, 因此我们需要在组合逻辑中加入判断读出的寄存器是不是 0 号, 实现该功能的方式有两种——使用三目运算符或者使用与或非逻辑进行选择。由于三目运算符会影响门延迟, 所以我选择使用与或非逻辑来完成该功能。

```
1  always @(posedge clk) begin
2          if (wen && waddr != 0) begin // 当写使能wen为1且waddr非0时写入数据
3              regfile[waddr] <= wdata;
4          end
5      end
```

图 1: reg_file 中时序逻辑注意点代码

```
// 异步读数据1
assign rdata1 = ({32{raddr1 == 0}} & `DATA_WIDTH'b0) | ({32{~(raddr1 == 0)}} & regfile[raddr1]);
// 异步读数据2
assign rdata2 = ({32{raddr2 == 0}} & `DATA_WIDTH'b0) | ({32{~(raddr2 == 0)}} & regfile[raddr2]);
```

图 2: reg_file 中组合逻辑注意点代码

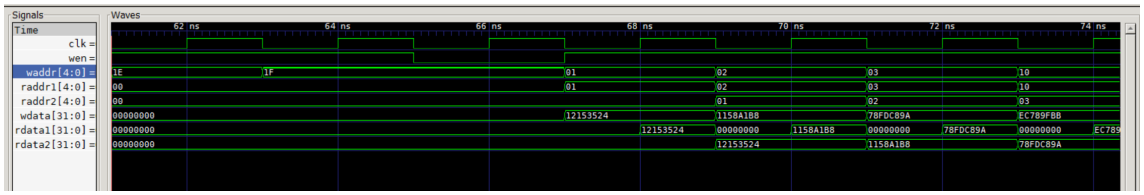



图 3: reg_file 的波形图

经过 gtkwave 软件显示波形后，我们可以看到：当时钟上升沿到达并且 wen 为 1 时，写地址对应的寄存器会存入相应的写数据；当读地址为 0 时，读出的数据为 0；当读地址变化为其他时，相对应的读数据也会变化为相应寄存器中写入的数据。

- **ALU 设计**。

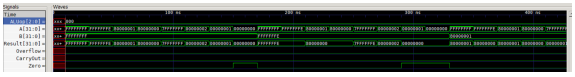
ALU 设计是实验一中的难点，虽然 ALU 只用组合逻辑电路就可以实现，但是由于其输入输出的要求，使得我们实现这个 ALU 时会存在许多障碍。

该 ALU 的要求是根据 ALUop 的输入值选择计算器的功能。同时算数加法，算数减法和有符号整数比较要求用一套加法器来进行实现，我在最初的设计中是将加法和减法分开设计的，经过询问老师后，发现了这一问题，并通过合并逻辑将这三种运算通过一套加法器实现。

另一个较为困难的地方就是我们只能通过 assign 语句来实现所有的组合逻辑，同时还需要考虑未定义的含义。在 PPT 中老师介绍了其实所有的运算都会进行，只是根据选择器的输入来选则将哪种输出出来而已，因此我们在选择器选择的模式不需要溢出和进位的时候，我们只需要随便将其赋值即可，在这里我选择的是将其赋值 0。

```
//求得加法/减法的结果以及进位
assign {re_CarryOut,re_ADDSUB} = {1'b0, A} + {1'b0, ({32{(SUB | SLT)}} & ~B) | ({32{~(SUB | SLT)}} & B)} + {31'b0, ALUop[2]};
assign re_SLT = {31'b0,(re_ADDSUB[31] ^ Overflow)}; //求得比较的结果：若最高位的值与符号位相同，则A>=B，否则A<B
```

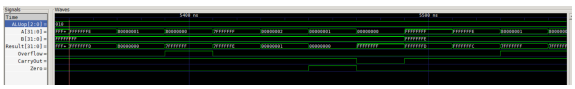
图 4: 用一套加法器实现三种功能代码



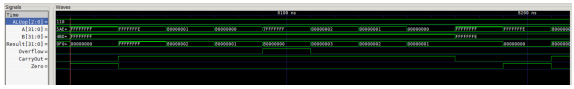
(a) ALU 实现按位与波形图

(b) ALU 实现按位或波形图

图 5: ALU 功能操作 1 2



(a) ALU 实现算数加法波形图



(b) ALU 实现算术减法波形图

图 6: ALU 功能操作 3 4

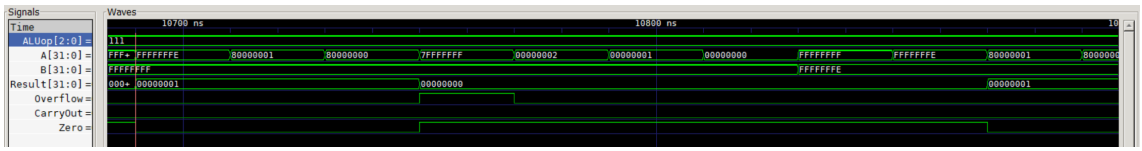



图 7: ALU 实现有符号数整数比较波形图 (ALU 功能操作 5)

以上为 ALU 实现五种不同计算的波形图。当 ALUop 为不同的五种信号的时候,Result,Overflow,Carryout,Zero 会显示不同的值,并与相应的计算方式相对应。

- **其他关键模块设计**。

  在本部分中我将绘制关键部分的门电路,并进行解释。

  首先是对于如何用一套加法器实现加法减法和比较这三种运算。

  当选择信号为加法时,加法器的结果直接为 A+B;

  当选择信号为减法时,加法器的结果为 A+ B+1,即通过 A+(-B) 的补码运算进行求解;

  当选择信号为比较信号时,取加法器结果的最高位与溢出信号进行异或,前面 31 位都置为 0。

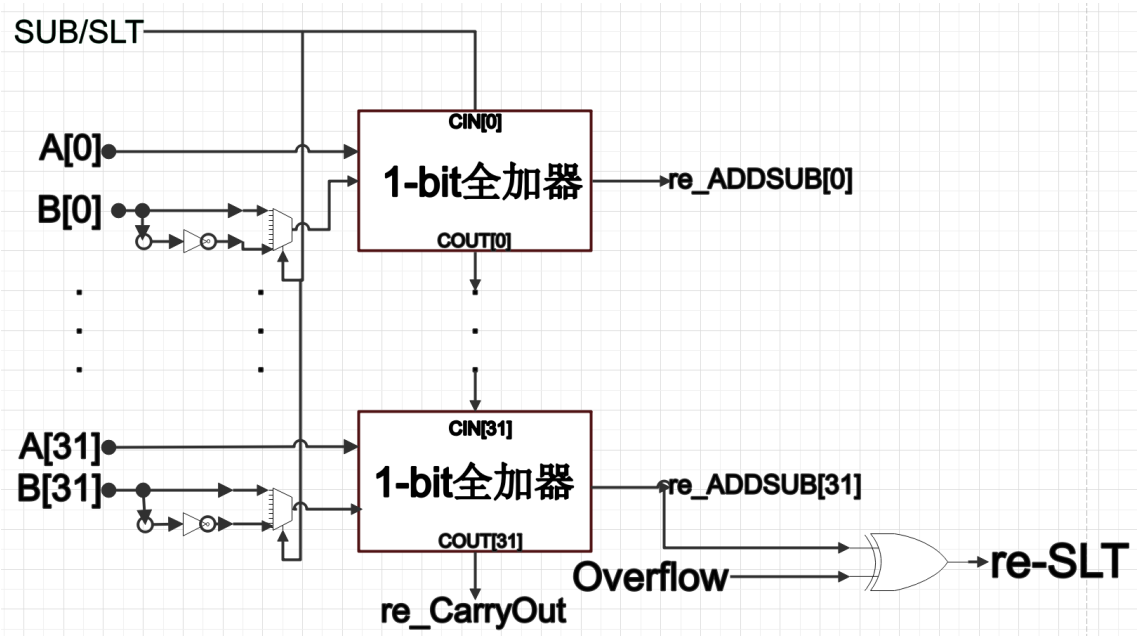



图 8: 用一套加法器实现三种计算的加法器电路图

针对有符号数大小比较的设计,从电路图中可以看出我是采用求和结果最高位与溢出位进行异或,这就首先需要了解溢出的原因:在有符号加减法过程中,如果加数和被加数(或减数和被减数)符号位相同,但是其结果的符号位与之相反,则溢出,即正数与正数求和得到负数或负数与负数求和得到正数,这些显然是错误的结果,就称为溢出。

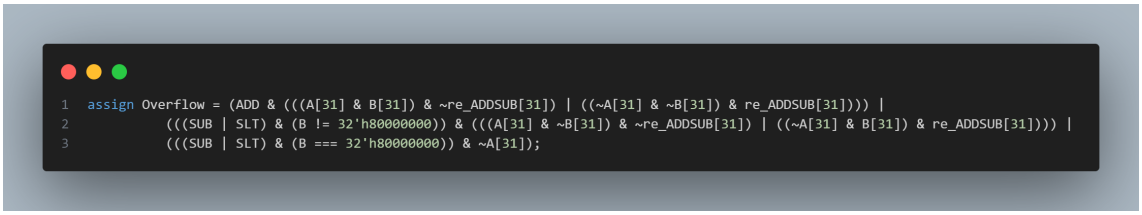



图 9: 实现 Overflow 的代码

这里面有一点我们需要注意的就是 B 是否等于 32'h80000000，这是因为在补码运算中，我们如果按照一般的规则，那么会计算出 B 为-0，但是我们规定 0 和-0 的补码都是 32'h00000000，因此我们定义 32'h80000000 为-2 的 31 次方，所以只要 A 是正数，B=32'h80000000，则 A-B 的结果一定会溢出。在 B!=32'h80000000 时，就按照正常的判断溢出的方法进行判断即可。

当我们了解了溢出的原理，加上实验要求我们去比较两个有符号整数的大小的时候，我们可以采用 A-B 的正负来进行判断，当 A-B 结果为负且计算结果是正确的时候，就可以得出 A<B。如果计算结果溢出，说明 A 和-B 符号相同，即 A 和 B 符号相反，我们根据溢出和结果的最高符号位即可判断 A 和 B 的大小，因为如果溢出，则结果最高位与 A 最高位相反，若结果最高位为 1，则 A 是正数，A>B，因此比较的结果为 0；反之，则为 1。

从电路图和源代码我们可以看出，加法运算、减法运算和比较运算仅使用了同一套加法器即可以得出三种运算的结果。

## 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**。

寄存器堆的设计的唯一难点，就是如何通过与或非逻辑，在组合逻辑电路中，进行判断在选择寄存器 0 的时候输出全 0，在选择其他寄存器的时候，按照其中存储的数据读出。

当然，因为这是本实验课的第一个正式实验，难度较低，经过回顾数字电路的门电路逻辑和 Verilog 语言的基本规则之后，就可以轻松解决这个难点。

- **ALU 设计问题**。

在 ALU 的设计过程中，遇到了如下几个问题：

1. 如何用一套加法器完成三种运算?
2. 怎么判断溢出?
3. 如何让我的代码中只用 assign 语句实现所有组合逻辑，同时不会出现- < > 等运算符?

以下是我的思考和解决方法：

1. 根据 PPT 中的 32 位加减器的电路示意图，将其写成 Verilog 代码

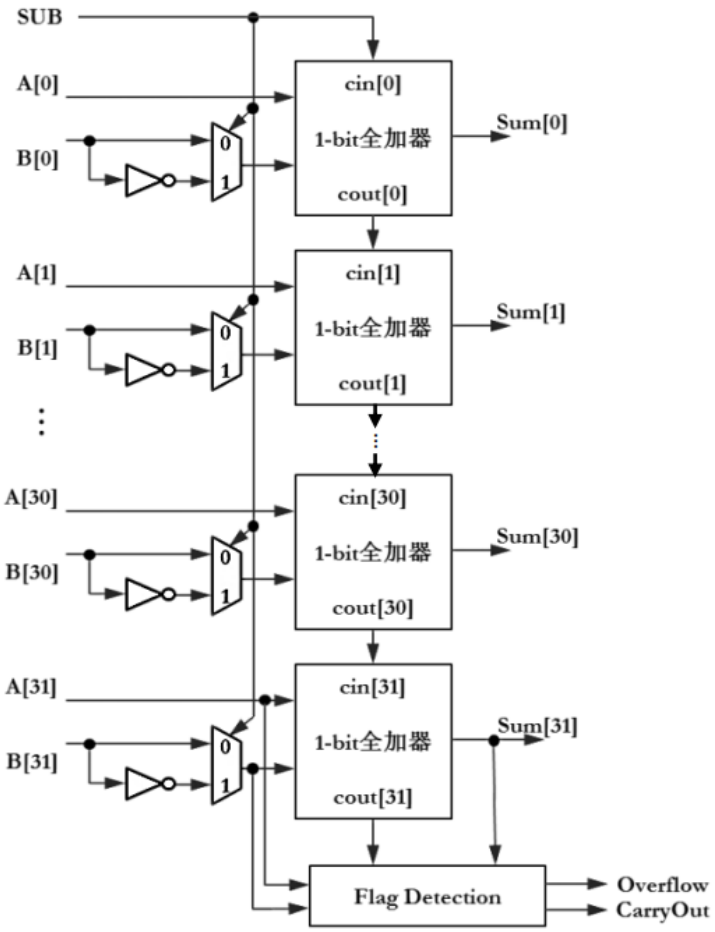



图 10: 32 位加减器电路示意图

从图中可以看出,实现加法就是正常的 32bit 串行进位加法器,而要实现减法操作,则根据 SUB 信号将 B 按位取反,同时进位 cin 的值由 SUB 决定,因为进行减法运算 A-B 等价于 A+(-B),我们根据 SUB 将-B 用补码运算规则表示即可。

对于 SLT 运算,这个在上一部分中已经详细说明,在这里不再重复。

2. 解决问题的方法(怎样判断溢出)已在上一部分中详细说明,在这里说明一下我解决这个难题的过程。首先就是根据 PPT 中的介绍:“• 两个正数的加法结果是一个负数 • 两个负数的加法结果是一个正数”,难点在于如何实现这个代码。

我在一开始的想法是正确的,即根据最高位的符号位是否相同以及与结果符号位是否相同来判断,但是却没有考虑到 A 和 B 是用补码表示,补码表示的数比原码要多出一个,就是上部分说明的 B=32'h800000000 表示最小的负数-2 的 31 次方。后来仔细阅读 PPT 之后,看到了下面这一页:

## 1.3.2 对于A、B和Result的理解

❑ 以下32-bit二进制比特串（机器数） 对应的真值（数学值）是多少?

- 1000 0000 0000 0000 0000 0000 0000 0000

❑ 不知道!!!

❑ 因为没有说明编码方式和定浮点格式!

- 补码：-2147483648
- 无符号编码：2147483648
- 原码：-0
- IEEE 754浮点数： -0.0

❑ 所以对一个比特串的解释（interpretation）有多种方式

图 11: 关于 32'h80000000 表示什么数的 slide

在思考为什么会有这一页 PPT 的时候，突然想到溢出是判断有符号数之间的加减法，而补码表示的数比原码多的正是这个 32'h80000000，然后想到任何正数减去这个数都会溢出，任何负数加上这个数也会溢出，而原来的判断逻辑无法判断这种情况，所以修改了组合逻辑，加上这个条件下的溢出判断，从而解决了这个问题。

3. 解决不允许出现- < > 符号是容易的，因为根据问题 1，我实现了加法器求减法，自然就不会出现减号，用加法器的结果来实现比较运算，也自然不会出现 < 和 >。

要只用 assign 语句来实现组合逻辑电路，不允许使用 always 语块的原因是使用 always 语块会导致出现锁存器，这个在仿真的时候是不会提示错误，但是在最终制作完 CPU 之后运行的时候会出现问题。因此我们只能通过与或非的逻辑门来实现所有组合逻辑电路功能，这就要求我需要详细复习一下数字电路实验课中学习的 Verilog 知识，加上老师和助教们制作的 PPT 加以辅助，将五种运算分而食之，解决了这个问题。

• **其他关键模块设计问题**。

在 reg 和 alu 的设计中，组合逻辑电路占据绝大部分内容，由于组合逻辑电路要求只用 assign 语句实现，加上我们有许多需要根据输入进行选择和判断的问题，我们有两种选择——使用三目运算符或者与或非逻辑，但是 PPT 中建议使用与或非逻辑来实现，因为三目运算符会加大门延时，使整个电路的时序不太好。

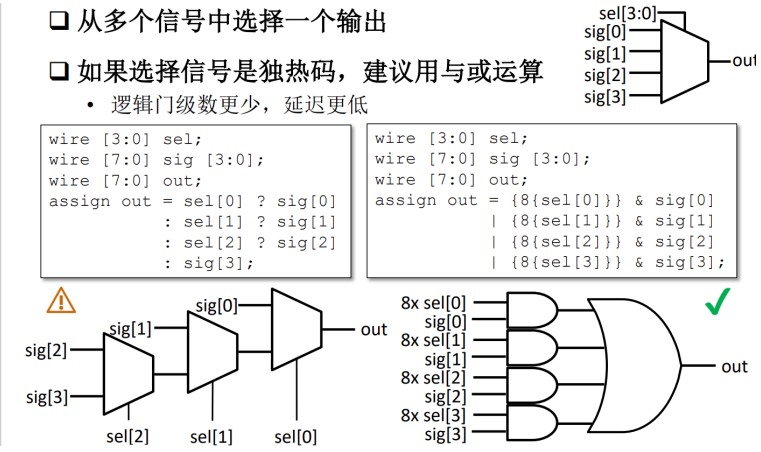



图 12: 三目运算符与与或非逻辑对比

**关于助教提出的建议**。

助教提出了两条建议：

1. 代码注意分行使其显得简介，或者定义 wire 变量代替一些较长的表示。

2.=== 与 == 虽然有区别，但是在本实验中不会存在高阻态或者不定态，如果出现则一定会有错误，因此可以用 ==，而不必用 ===。

关于助教提出的改进意见，我对自己的代码稍作修改，使其在可读性上显得更加简洁，同时将 === 修改为 ==。

## 三、 实验所耗时间

在课后，你花费了大约 3.5 小时完成此次实验。