PA1实验报告

时昌军 221220085

一、实验目的

1. 复习理论课中整数和浮点数的表示存取运算相关知识；
2. 了解 NEMU 的总体架构和工作原理；
3. 实现整数的表示，存储和运算；
4. 实现浮点数的表示和运算。

二、实验背景

现代计算机的世界是一个由0（低电平）和1（高电平）构成的世界，所有的信息都被表示成01串形式的数据。我们把世界上所有需要表示的数据大概分一分，发现可以分为两大类：非数值型和数值型。

非数值型大体上对应的是用于表征类别或个体的信息，比如汉字、英文字母、阿拉伯数字、每个学生个体等等，我们可以将每一个需要表示的对象赋予一个编号，并且让大家都遵守统一的编号规则就行了。比如ACSII编码，UTF-8编码等等。我们在PA 2开始要重点关注的指令数据，其操作码就可以认为是一种非数值型数据。

与非数值型相对应的是数值型的信息，用于进行科学计算，主要包含两种类型：整数和实数（复数总可以用两个实数来表示）。而在整数和实数上又定义了加减乘除等各种运算规则。计算机顾名思义就是要进行计算的，在这一节中我们要重点关注的就是以整数和实数为代表的数值型信息的表示、存取和运算。

在PA 1中，配合理论课的节奏，我们先通过数据的存储开始简单了解NEMU的总体结构，进而从整数和浮点数的表示、存取和运算出发，开始构建一台完整计算机的旅程。

三、实验过程

1. PA 1-1 数据在计算机内的存储

在1-1中，主要考察的是寄存器模拟和主存模拟。其中寄存器的模拟采用 union 结构，内存采用字节型数组，约定大小为128MB。

2. PA 1-2 整数的表示、存储和运算

在1-2中，考察整数的表示、存储和运算问题。

- **整数的加减操作**

在采用补码表示法后，带符号和无符号的整数加减法可以统一使用无符号整数加减法来执行。因此只需要实现无符号整数加减法即可。同时，考虑是否带进位和借位的加减法，一共需要实现四个无符号加减法函数。

心得：在写这四个函数的过程中，我理解了标志位的用法和实现，特别是OF和CF这两个标志位。

- **整数的移位操作**

整数的移位操作从方向上可以分为左移和右移两种，从对符号位的处理方式上又可以分为逻辑移位和算术移位两种，因此一共可以分为算术左移（SAL）、逻辑左移（SHL）、算术右移（SAR）和逻辑右移（SHR）四种。

心得：移位操作的实现在1-2中难度较高，特别是要先熟悉CF、OF在左右移中的含义。

- **整数的逻辑运算**

整数的逻辑运算包括整数的与（AND）、或（OR）非（NOT）、异或（XOR）操作。在实验中只需要实现与、或、异或这三个操作。

心得：较为简单

- **整数的乘除运算**

整数的乘除运算按照是否带符号可以分为两组，无符号乘除法和有符号乘除法。

心得：模拟乘法运算时得到的结果应是 64 位整型，由于 bug 没有发现也是卡住了一段时间。

```
uint64_t alu_mul(uint32_t src, uint32_t dest, size_t data_size
{
    uint64_t res=0;
    //获取计算结果
    uint64_t src64=src;
    uint64_t dest64=dest;
    res=src64*dest64;
    //设置标志位
    set_CF_mul(res,data_size);
    set_OF_mul(res,data_size);
    //返回,此时高位不能清零!
    return res;
}
```

3. PA 1-3 浮点数的表示和运算

在本实验中，我们仅将注意力集中在对浮点数运算的模拟上。

基本步骤：

1. **判断边界条件。**在进行加减运算之前，我们首先要判断一些包括参与运算的数字是`±0`、`±∞`、`NaN`的边界条件。
2. **提取尾数并对阶。**当处理的浮点数`FLOAT f`是规格化数时尾数需要加上隐藏位1，否则隐藏位为0。提取尾数至至少为32位（乘除法中用64位较为便利）的无符号整型变量中，并根据是否是规格化数补上隐藏位得到`significand`；将参与运算的两个尾数都左移3位，为`GRS bits`空出位置；确定阶小的那个数需要右移的位数`shift`；将阶小的那个数的尾数右移执行对阶操作，移出部分自然进入`G`和`R`两位，`S`位的取值需根据粘位规则额外操作。
3. **尾数相加。**浮点数的尾数部分采用的是原码表示，而浮点数的符号位保存在最高位。在做尾数相加时，只需按照符号位的取值将原码表示的尾数转变为补码表示，并做无符号整型加法即可。执行尾数相加后，再根据尾数之和的补码取值，判断结果的正负号。根据结果的符号设置运算结果浮点数的符号，并将尾数从补码转回原码表示。
4. **尾数规格化和舍入。**完成尾数相加后，我们接下来需要对尾数进行规格化和舍入。

浮点数乘除运算的基本步骤和加法类似，相比加法运算，还可以免去对阶的过程。需要注意的是，加入乘除法运算后尾数规格化处理 时需要多考虑阶码为负的情况。

四、思考题

1. **C语言中的`struct`和`union`关键字都是什么含义，寄存器结构体的参考实现为什么把部分`struct`改成了`union`？**

答：C 语言中的`struct`(结构体)与`union`(联合体)都是复合结构，由多个不同的数据类型成员组成。`struct`中所有成员占用空间是 累加的，所有成员都存在，不同成员会存放在不同的地址。在计算一个结构体变量的总长度时，其内存空间大小等于所有成员长度的和 （需要考虑字节对齐）；`union`中所有成员共用一块地址空间，即联合体只存放了一个被选中的成员，所有成员不能同时存在，不能同时占用内存空间，所以一个联合型变量的长度等于其最长的成员的长度。

在 NEMU 的模拟中，各个成员读取的是同一个寄存器的不同部分， 处于同一块内存空间，因此应当使用`union`而非`struct`。

2. **为浮点数加法和乘法各找两个例子：1）对应输入是规格化或非规格化数，而输出产生了阶码上溢结果为正（负）无穷的情况；2）对应输入是规格化或非规格化数，而输出产生了阶码下溢结果为正（负）零的情况。是否都能找到？若找不到，说出理由。**

答：在浮点数运算中，阶码上溢是指阶码从正的方向超出了阶码的表示范围（全1），阶码下溢是指阶码从负的方向超出阶码的表示范围（全0）。

1）浮点数加法中，可以找到两个数相加产生阶码上溢结果为正无穷的情况，如两个规格化浮点数相加:$1.0B \times 2^{127} + 1.0B \times 2^{127}$，对尾数进行右规，阶码加一变为 1111 1111，应将结果设置为正无穷。浮点数乘法中，如$1.0B \times 2^{127} \times 1.0B \times 2^{127}$，因为阶码直接相加，得到的和大于 1111 1111，故设为无穷大。将以上两个例子中的浮点数取负即可得到负无穷的例子。

2)浮点数加法中，两个正数相加不会出现阶码下溢，因为他们的和的绝对值大于其中任意一个数，阶码大于等于两个数中较小的那个阶码。浮点数乘法中，可能出现阶码下溢，如$1.0B \times 2^{-126} \times 1.0B \times 2^{-126}$， 阶码相加的真值为-252 超过最小范围，因此结果设为 0。将其中一个乘数取负得到负下溢的情况。

五、总结与反思

PA-1 总体难度不算太大，所用到的知识与课本结合比较密切，很多实现汪亮老师上课都提到了，重点在于对 NEMU 结构的熟悉和对整数、浮点数运算的理解上，不过中间由于对原理不熟悉等原因也卡住了一段时间，希望下一阶段的 PA 实验能够顺利进行。