南京大学

PA-1：数据的表示、存取和运算

课程名称：计算机系统基础

姓名：孙文博

学号：201830210

邮箱：201830210@smail.nju.edu.cn

实验时间：2022.3.17

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。我们把所有需要表示的数据大概分为两大类：非数值型数据和数值型数据。

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。

三、 实验过程

1. PA-1.1

这部分中我们需要熟悉 NEMU 系统内部的大致结构，了解数据的存储器体系结构，重点掌握寄存器和内存的模拟。其中寄存器的模拟采用 union 结构（原因见思考题 1），内存采用字节型数组，默认大小为 128MB（128*1024*1024）。

2. PA-1.2

这部分中我们需要复习课本上关于整数的算术和逻辑运算相关知识，了解 NEMU 中执行整数运算操作的函数以及 EFLAGS 寄存器，并根据 i386 手册完成不同运算操作中一系列标志位的模拟。例如，下面是实现加法操作的代码：

```
uint32_t alu_add(uint32_t src, uint32_t dest, size_t data_size)
{
    uint32_t res = 0;
    res = dest + src; // 获取计算结果

    // 设置标志位
    set_CF_add(res,src,data_size);
    set_PF(res);
    // set_AF();
    set_ZF(res,data_size);
    set_SF(res,data_size);
    set_OF_add(res,src,dest,data_size);

    return res & (0xFFFFFFFF >> (32 - data_size)); // 高位清零返回
}
```

其中我们调用 set_CF_add()等函数实现标志位的模拟。

值得一提的是，模拟乘法运算时我们得到的结果应是 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;
    uint64_t src_s=src;
    uint64_t dest_s=dest;
    res = dest_s * src_s; // 获取计算结果

    // 设置标志位
    set_CF_mul(src,dest,data_size,res);
    set_OF_mul(src,dest,data_size,res);
    set_PF_mul(res);
    set_ZF_mul(res,data_size);
    set_SF_mul(res,data_size);

    return res;
}
```

3. PA-1.3

这部分中我们要模拟浮点数的表示和运算，首先是浮点数的表示，NEMU 采用 IEEE 754 标准，32 位单精度浮点数由最高位 1 位符号位（sign），8 位阶码部分（exponent）和 23 位尾数部分（fraction）构成。8 位阶码部分采用移码形式，其偏置常数为 127；对于规格化数而言，23 位的尾数部分实际上表示了 24 位的有效数字，其最高位缺省为 1，而非规格化数则没有缺省 1，判断隐藏位之后的尾数称为 significand。

接着模拟浮点数的运算，主要分为判断边界条件，提取尾数并对阶，尾数相加和尾数规格化与舍入四步。

第一步判断边界条件已经在框架代码中实现；

第二步对阶时我们采用阶向大阶看齐，阶小的那个数的尾数右移，右移的位数等于两个阶的差的绝对值。但是要注意判断非规格化数，其阶码全 0 而尾数非 0，此时阶码应理解为-126 而不是-127。此外在

在对 fa 进行对阶的过程中，尾数每右移一位都会在最高位补 0，并导致最低有效位移出，为了提高运算结果的精度，我们在运算中间结果右边保留三个附加位：保护位（guard, G），舍入位（round, R）和粘位（sticky, S），其中只要舍入位右边有任何非 0 数字，粘位就置为 1，否则置为 0（符合“粘”的特点）。

第三步位数相加我们先要根据符号位 sign 的值判断尾数的正负，再对其做无符号整型加法，最后根据结果正负号设置结果浮点数的 sign 位。

第四步尾数规格化与舍入处理我们调用专门的函数 internal_normalize()，分别考虑左规和右规两种情况，注意到阶码的上溢与下溢，讲最终结果的符号阶码尾数返回。

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?

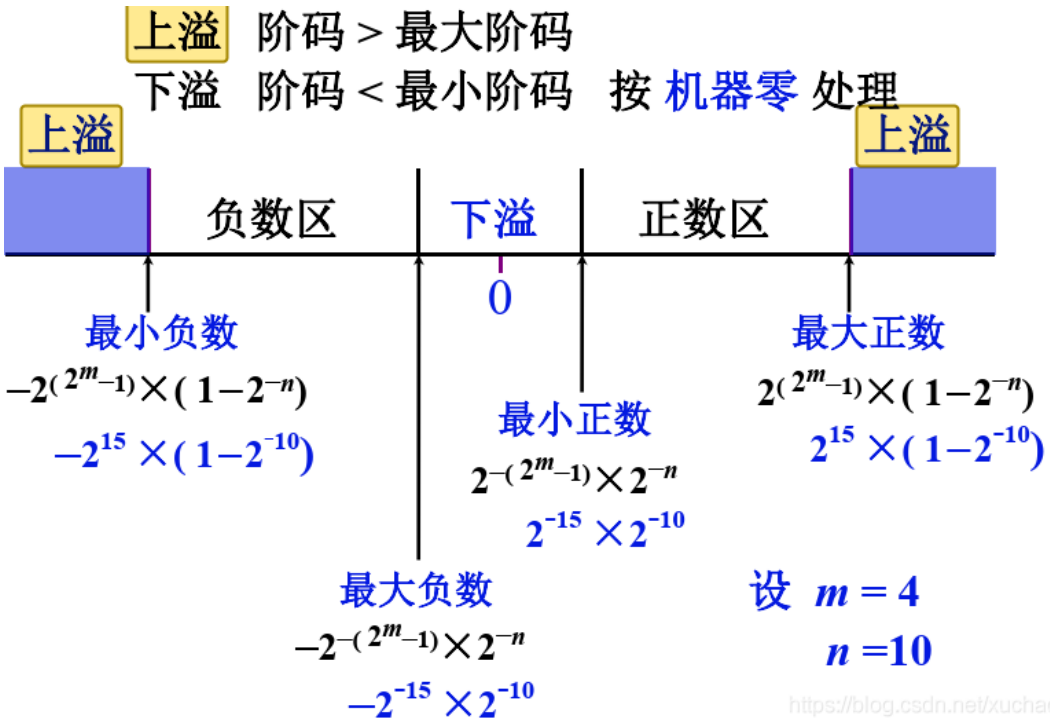
答：C 语言中的 struct（结构体）与 union（联合体）都是复合结构，由多个不同的数据类型成员组成。不同之处在于 union 中所有成员共用一块地址空间，即联合体只存放了一个被选中的成员，所有成员不能同时占用内存空间，它们不能同时存在，所以一个联合型变

量的长度等于其最长的成员的长度；而 struct 中所有成员占用空间是累加的，其所有成员都存在，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）。

在 NEMU 的模拟中，各个成员读取的是同一个寄存器的不同段落，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$1.0B \times 2^{127} + 1.0B \times 2^{127}$（机器数表示为 0 1111 1110 000 0000 0000 0000 0000 0000），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，更容易找到例子，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$1.0B \times 2^{-126} \times 1.0B \times 2^{-126}$（机器数表示为 0 0000 0001 000 0000 0000 0000 0000 0000），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

作为第一阶段的 PA 实验，PA-1 总体难度不算太大，所用知识与课本结合密切，重点在于对 NEMU 结构的熟悉和对整数浮点数运算的理解上，不过中间由于不熟悉 debug 的流程等原因也卡住了一段时间，希望下一阶段的 PA 实验能够再接再厉！💪