PA1 实验报告

1.1

C 语言中的 struct 是指结构类型，里面由若干个成员组成，成员可以是不同的数据类型。而 union 是指联合体，里面同样可以有若干成员，但这些成员共用一块内存区域，这是 struct 与 union 的重大区别。

下面具体分析为何将 struct 改成 union。

```
union{
    uint32_t _32;
    uint16_t _16;
    uint8_t _8[2];
}
```

这一层使得三个宽度的变量共用一块内存区，从而实现了寄存器按长度的划分。(一个 32 位的寄存器可分为 2 个 16 位，同时低 16 位又可分为 2 个 8 位)。

```
union{
    union{
        uint32_t _32;
        uint32_t _16;
        uint8_t _8[2];
    };
    uint32_t val;
}gpr[8]
```

在这一层中，开辟了八个通用寄存器空间，每一个通用寄存器为联合体类型。

```
union{
    union{
        union{
            uint32_t _32;
            uint32_t _16;
            uint8_t _8[2];
        };
        uint32_t val;
    }grp[8];
    struct{
        uint32_t eax,ecx,edx,ebx,esp,ebp,esi,edi;
    };
};
```

这一层中实现了通用寄存器组与 32 位整型变量 eax,ecx...的一一对应。

1.3

(1)

加法

例子一：两个加数都为 1.0*2 的 127 次方时(单精度浮点机器数表示为 0x7f000000)，阶码上溢，结果为正无穷。

(0x7f000000=0 11111110 00000...0B)
例子二：一个加数为-1.1*2 的 127 次方，另一个为-1.1*2 的 126 次方时(单精度浮点机器数表示为 0xff400000 和 0xfec00000)，阶码上溢，结果为负无穷。
(0xff400000=1 11111110 1000...0B)
(0xfec00000=1 11111101 1000...0B)
乘法
例子一：一个乘数为 1.0*2 的 127 次方，另一个为 2 的 1 次方时(单精度浮点机器数表示为 0x7f000000 和 0x40000000)，阶码上溢，结果为正无穷。
(0x7f000000=0 11111110 0000...0B)
(0x40000000=0 10000000 0000...0B)
例子二：两个乘数都为 1.0*2 的 127 次方(单精度浮点机器数表示为 0x7f000000)，阶码上溢，结果为正无穷。

(2)
加法
例子一：一个加数是 0.01*2 的-126 次方(二进制)，另一个加数是-0.1*2 的-126 次方(二进制)(单精度浮点机器数表示为 0x00200000 和 0x80400000)，阶码下溢，结果为负零。
(0x00200000=0 00000000 0100...0B)
(0x80400000=1 00000000 1000...0B)
例子二：一个加数是 0.001*2 的-126 次方(二进制)，另一个加数为-0.1*2 的-126 次方(二进制)(单精度浮点机器数表示为 0x00100000 和 0x80400000)，阶码下溢，结果为负零。
(0x00100000=0 00000000 0010...0B)

乘法
例子一：两个乘数都为 0.1*2 的-126 次方(二进制)，(单精度浮点机器数表示为 0x00400000)，阶码下溢，结果为正零。
(0x00400000=0 00000000 1000...0B)
例子二：一个乘数为 1.0*2 的-126 次方(二进制)，另一个为 0.1*2 的-126 次方(二进制)，(单精度浮点机器数表示为 0x00800000 和 0x00400000)，阶码下溢，结果为正零。
(0x00800000=0 00000001 0000...0B)

实验出现的问题、心得、及重点整理：
1.
设置 PF 时，相当于对低八位进行奇校验，即奇数个 1 为 1，偶数个 1 为 0。

2.
注意 add 与 adc，sub 和 sbb 区别，都是根据当前 CF 判断的，如果 CF 为 0，则是一样的，如果为 1，则表示进位和借位。

在有进位的加法中对 CF 的判断要小心，如果简单地调用 set_CF_add(result,src,dest+1)或是 set_CF_add(result,src+1,dest)会出问题，给出一个例子：result=0000,src=0000,dest=1111，检查时 result==src&&result==dest+1，CF 就被设为 0，而它应该是产生进位的。所以可见三个数相加，即使一个数是 1，也应该逐步考虑，判断 CF。即需要两步，set_CF_add(dest+src,src,dest)，如果此时产生进位，则总结果的 CF 一定是 1，否则再

set_CF_add(dest+src+1,dest+src,1)，这时的 CF 才是最终结果的 CF。

在 set_CF_sub 函数中，我是转化成三个加数相加来设置 CF，即 src,~dest,1，思路就类似于上面。当然注意 CF 最后还要取一次反，因为 CF=Cin xor Cout。

在 set_OF_sub 函数中，我参考了汇编讲义，通过判断减数，被减数，结果的符号来设置 OF。在 sbb 函数中设置 CF 时，我沿用了上面的思路，进行了两次减法，逐次判断。而在设置 OF 时，我发现这种思路不行(添加了一个测试用例 0x80000000)。我得到了很多导致我 OF 错误的用例。

```
oracle eflags CF = 1, PF = 0, ZF = 0, SF = 1, OF = 0
nemu   eflags CF = 1, PF = 0, ZF = 0, SF = 1, OF = 1
a = -2147483648, b= -3, res = -2147483646, res_asm = -2147483646
```

```
oracle eflags CF = 1, PF = 1, ZF = 0, SF = 0, OF = 0
nemu   eflags CF = 1, PF = 1, ZF = 0, SF = 0, OF = 1
a = 0, b= -2147483648, res = 2147483647, res_asm = 2147483647
```

```
oracle eflags CF = 1, PF = 1, ZF = 0, SF = 0, OF = 0
nemu   eflags CF = 1, PF = 1, ZF = 0, SF = 0, OF = 1
a = 0, b= -2147483648, res = 2147483647, res_asm = 2147483647
```

我发现都是因为 0x80000000 出事，于是我决定单独对 0x80000000 讨论。先判断被减数和减数符号，相同 OF 一定为 0，再判断是否为 0x0-0x80000000-1 这一特殊情况，再判断如果 dest 和 src 都不为 0x80000000，则直接 set_OF_sub(result,src,dest-1)，否则 OF 一定为 1。经过反复测试，终于通过了。

3.

对汇编课程中讲到的公式 CF=cin xor cout 重新理解：这里的 cin 应该是 sub 信号，即如果为减法，底层 alu 中将接受到一个 sub 信号，然后减数取反，和被减数，sub 信号一起进入 alu 中的加法器。而我们这里的 adc 函数中对 CF 的设置，不能用这样的方法，即我们不能理解成 cin 为现阶段的 CF 值，然后与 cout 异或，例如 1111，0000，当前 CF 为 1，则结果为 0000，cout 为 1，异或的结果为 1 xor 1=0，显然这样就错了。

4.

注意逻辑运算中 OF=CF=0。

5.

书写过程中 if 语句要有大括号(即使只有一条语句)，否则有时候会报错，不知道什么原因。

6.

对 GRS 的理解：粘位可以看作是把 1 粘住。按偶数舍入：先看最后三位即 GRS，若大于 4，则进一，小于 4，舍去，等于 4，看最低有效位，为 1，则进，否则舍。

7.

注意在浮点数加法中对阶 shift 的值，在 exponent=0 时，它要加加，因为此时它反映的是-2 的 126 次方。一开始我加法就是在这里出问题了。

8.

在浮点数乘法中中间结果阶码 exp_res=E1+E2+127-(46-23)，此处的 E1 和 E2 都以考虑了阶码为 0 的情况，关键是这里的‘-(46-23)’，因为尾数相乘是 46 位的，它实际的小数点是在左边第 46 位上，但在规格化中，我们统一看成小数点在左边第 23 位(而在送入规格化中，我们都会把尾数再左移 3 位，给 GRS 留下空间)，所以得到的阶码还要减去(46-23)。

9.

同样在浮点数除法中中间结果阶码 exp_res=fa.exponent - fb.exponent + 127 - (shift - 23)+3，注意这里已经考虑好了 a、b 阶码为 0 的情况。关键是‘-(shift-23)+3’，shift 是被除数左移和除数右移的和，相当于把结果乘了 2 的 shift 次方，同样我们把小数点看在左边第 23 位上，所以‘-(shift-23)’，最后加上 3，是此时把尾数的最后三位当作 GRS。

最后 PA1 的成果：

```
======== reg test ========
reg_test()      pass

======== alu test ========
alu_test_add()  pass
alu_test_adc()  pass
alu_test_sub()  pass
alu_test_sbb()  pass
alu_test_and()  pass
alu_test_or()   pass
alu_test_xor()  pass
alu_test_shl()  pass
alu_test_shr()  pass
alu_test_sal()  pass
alu_test_sar()  pass
alu_test_mul()  pass
alu_test_div()  pass
alu_test_imul() pass
alu_test_idiv() pass

======== fpu test ========
fpu_test_add()  pass
fpu_test_sub()  pass
fpu_test_mul()  pass
fpu_test_div()  pass
```