1-1

```
make[1]: Leaving directory '/home/user/pa2017/nemu'
user@zxr:~/pa2017$ ./nemu/nemu

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

======== alu test ========
Please implement me at alu.c
nemu: src/cpu/alu.c:5: alu_add: Assertion `0' failed.
Aborted
user@zxr:~/pa2017$
```

(1) struct：结构类型，由若干成员组成，每个成员可以是一个基本数据类型或者是另一个构造类型。

union：也是一种构造类型的数据结构，在一个联合内可以定义多种不同的数据类型，一个该联合类型的变量允许被装入该联合定义的任何一种类型的数据，这些数据共享同一段内存。

将 struct 改为 union 因为 union 类型更适合寄存器可作为 16,32 位复用的特性。

1-2

```
======== 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 ========
Please implement me at fpu.c
nemu: src/cpu/fpu.c:128: internal_float_add: Assertion `0' failed.
```

1-3

```
user@zxr:~/pa2017$ ./nemu/nemu

======== 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
user@zxr:~/pa2017$
```

1)01111111011111111111111111111111+

01111111011111111111111111111111

=inf

11111111011111111111111111111111+

11111111011111111111111111111111

=-inf

01111111011111111111111111111111*

01111111011111111111111111111111

=inf

11111111011111111111111111111111*

01111111011111111111111111111111

=-inf

2)00000000100000000000000000000010+

10000000100000000000000000000001

=+0

00000000100000000000000000000001+

10000000100000000000000000000010

=-0

00000000000000000000000000000001*

00000000000000000000000000000001

=0

00000000000000000000000000000001*

10000000000000000000000000000001

=-0

过程分析：

问题 1：adc 与 sbb 的 CF 判断

adc 的 CF 判断花了很长时间，针对各种特殊情况进行了很多条件判断且频频出错

sbb 的 CF 判断时想到了将 sbb 当做两次 sub 判断的方法，对 sub 进行复用节约了时间

问题 2：adc 与 sbb 的 OF 判断

这两个 OF 判断不应像 CF 判断一样复杂化，只要抓住 OF 的定义，根据数据与结果的符号判断

问题 3：对 data_size 的理解

在这里我将 data_size 看做不确定的数，变相地将问题复杂化了，要是能早点看出 data_size 只会局限于 8,16,32，用条件判断来写能节约很多时间

问题 4：对 data_size 作用的理解，没有敏锐地看到 data_size 是用来限制操作的数据位数，开始未能考虑到移位等操作不应改变 data_size 外的位，造成了很大麻烦

问题 5：移位操作的符号位设置

移位操作的符号位应根据 data_size 内的数据变化设置，而不是根据整个 32 位数据设置，所以这里的符号位设置函数应该另写，多传一个参数 data_size

问题 6：浮点数规格化

对隐藏位的理解不全面，最后才搞清楚粘位的设置方法

问题 7：浮点数乘除法阶码的纠正

消耗很长时间才通过小数点的位置理解浮点数乘除法阶码的纠正方法