PA5 从一到无穷大：程序与性能

- 姓名：管昀玫
- 学号：2013750
- 专业：计算机科学与技术

1 概述

1.1 实验目的

- 实现浮点数支持
- 通过整数模拟实数运算
- 理解binary scaling支持

1.2 实验内容

1. 用整数来模拟实数
2. 实现浮点数运算

2 浮点数的支持

在PA中，我们将使用整数来模拟实数运算，这种方法交binary scaling。

2.1 整数表示实数

浮点数在计算机中采取科学计数法的形式进行存储：

$$num = (-1)^s * M * 10^E$$

它能分为三个部分：

- 符号位S：s为0，符号位为正；s为1，符号位为负
- 尾数M：即一个科学计数法的小数部分
- 指数E：即科学计数法表示的指数部分

因此，FLOAT类型最高位为符号位，接下来的 15 位表示整数部分，低 16 位表示小数部分, 即约定小数点在第 15和第 16 位之间(从第 0 位开始)，如下图所示：

```
31  30                16                  0

+----+-------------------+--------------------+

|sign|      integer      |      fraction      |

+----+-------------------+--------------------+
```

对于一个实数a，他的FLOAT类型表示为：`A=a*2^16`（阶段结果的小数部分）。举例：

- 1.2表示为：

```
1.2 * 2^16 = 78643 = 0x13333

+----+------------------+--------------------+
| 0  |        1         |        3333        |
+----+------------------+--------------------+
```

- 5.6表示为：

```
5.6 * 2^16 = 367001 = 0x59999

+----+------------------+--------------------+
| 0  |        5         |        9999        |
+----+------------------+--------------------+
```

但是实际上，1.2的FLOAT类型表示的数据并不是1.2，而是$0x13333/2^{16} = 1.19999695$，同理5.6实际存储的是$0x59999/2^{16} = 5.59999084$。

而负实数要用相应正数的相反数来表示，比如-1.2会被表示为$-(1.2 * 2^{16}) = -0x13333 = 0xffffecccd$

我们首先需要实现一个用32位正数FLOAT来模拟真正的浮点数。修改`navy-apps/apps/pal/src/FLOAT/FLOAT.c`中的f2F函数。我们定义一个Union来表示浮点数，其结构包含了：

- 有效位
- 指数位
- 符号位

然后将二进制数转换为浮点数：

1. 将真实指数减去固定移码偏移量
2. 指数位大于7时小数位不足，需要左移；指数位小于7时则小数位溢出，需右移
3. 判断符号正负，负值则左移31位取负值

```
FLOAT f2F(float a) {
  /* You should figure out how to convert `a' into FLOAT without
   * introducing x87 floating point instructions. Else you can
   * not run this code in NEMU before implementing x87 floating
   * point instructions, which is contrary to our expectation.
   *
   * Hint: The bit representation of `a' is already on the
   * stack. How do you retrieve it to another variable without
   * performing arithmetic operations on it directly?
   */
  union float_ {
    struct {
      uint32_t m : 23;
      uint32_t e : 8;
      uint32_t signal : 1;
    };
    uint32_t value;
```

```
  };
  union float_ f;
  f.value = *((uint32_t*)(void*)&a);
  int e = f.e - 127;
  FLOAT result;
  if (e <= 7) {
    result = (f.m | (1 << 23)) >> 7 - e;
  }
  else {
    result = (f.m | (1 << 23)) << (e - 7);
  }
  return f.signal == 0 ? result : (result|(1<<31));
}
```

2.2 FLOAT运算

1. 加减法：由于FLOAT的是用补码的方式表示，加减法直接用整数加减法表示即可
2. 而FLOAT的乘除法发现可以采用右移、左移16位的方法(除法需要考虑符号问题)。
3. 关系运算可以直接按照整数的关系运算实现

首先实现 `F_mul_F` 函数。在乘法中，$A*B=a*2^{16}*b*2^{16}=(a*b)*2^{32}!=(a*b)*2^{16}$两个数相乘之后，需要右移16位，即将结果除以2^16，才能得到正确的结果。具体代码如下所示：

```
FLOAT F_mul_F(FLOAT a, FLOAT b) {
    assert(-((int64_t)1 << 32) < ((int64_t) a * (int64_t) b) >> 16 &&
                   ((int64_t) a * (int64_t) b) >> 16 < ((int64_t)1 << 32));
    return ((int64_t) a * (int64_t) b) >> 16;
}
```

然后实现 `F_div_F` 函数。由于$A/B=(a*2^{16})/(a*2^{16})=a/b$，除法需要进行依次左移，即类似手写除法一样计算得到结果。

```
FLOAT F_div_F(FLOAT a, FLOAT b) {
    int op = 1;
    if(a < 0) {
        op = -op;
        a = -a;
    }
    if(b < 0) {
        op = -op;
        b = -b;
    }
    int ret = a / b;
    a %= b;
    int i;
    for (i = 0;i < 16;i ++){
        a <<= 1;
        ret <<= 1;
        if (a >= b) a -= b, ret |= 1;
    }
```

```
    return op * ret;

}
```

2.3 f2i和i2f

接下来我们需要实现FLOAT和INT数据之间的转换。这一部分在`navy-apps/apps/pal/include/FLOAT.h`中实现。

`F2int`的逻辑如下：如果一个参数`a`的最高位为0，将其左移16位并返回，将低16位设为0，相当于将`a`转换为正数的`FLOAT`类型；否则，将参数`a`取反后左移16位并返回，将低16位设为0，相当于将`a`转换为负数的`FLOAT`类型。

`int2F`的逻辑正好相反。具体实现如下所示：

```
static inline int F2int(FLOAT a) {//浮点数转整数
  if ((a & 0x80000000) == 0) {//正数
    return a >> 16;//右移16位
  }
  else {
    return -((-a) >> 16);//负数
  }
}

static inline FLOAT int2F(int a) {
  if ((a & 0x80000000) == 0) {
    return a << 16;
  }
  else {
    return -((-a) << 16);
  }
}
```

2.4 FLOAT与int间的运算

这两个数据类型的运算，其实就是把int类型先转化为FLOAT类型，然后与FLOAT进行运算即可。

```
static inline FLOAT F_mul_int(FLOAT a, int b) {
  FLOAT temp = int2F(b);
  return F_mul_F(a, temp);
}

static inline FLOAT F_div_int(FLOAT a, int b) {
    FLOAT temp = int2F(b);
   return F_div_F(a, temp);
}
```

由于代码中并没有出现`int_mul_F`和`int_div_F`的情况，因此我们只需要实现`F_mul_int`和`F_div_int`即可。

2.5 Fabs

除此之外，还需要提供以下的运算：

```
FLOAT Fabs(FLOAT);
FLOAT Fsqrt(FLOAT);
FLOAT Fpow(FLOAT, FLOAT);
```

由于代码已实现`Fsqrt`和`Fpow`，我们只需要实现`Fabs`即可。在`FLOAT.c`中添加以下代码：

```
FLOAT Fabs(FLOAT a)
{
  return (a > 0) ? a : -a;
}
```

float相关的代码已添加完毕。

2.6 shrd和shld

实现了运算之后，我们还需要实现指令才能完整地实现PA5。

根据报错信息（The instruction at eip = 0x080497b7 is not implemented），需要实现的指令是位移指令。查找i386手册，看到我们需要实现的指令：

SHRD — Double Precision Shift Right

Opcode	Instruction	Clocks	Description
0F AC	SHRD r/m16,r16,imm8	3/7	r/m16 gets SHR of r/m16 concatenated with r16
0F AC	SHRD r/m32,r32,imm8	3/7	r/m32 gets SHR of r/m32 with r32
0F AD	SHRD r/m16,r16,CL	3/7	r/m16 gets SHR of r/m16 concatenated with r16
0F AD	SHRD r/m32,r32,CL	3/7	r/m32 gets SHR of r/m32 concatenated with r32

Operation

```
(* count is an unsigned integer corresponding to the last operand of the
instruction, either an immediate byte or the byte in register CL *)
ShiftAmt ← count MOD 32;
inBits ← register; (* Allow overlapped operands *)
IF ShiftAmt = 0
THEN no operation
ELSE
   IF ShiftAmt ≥ OperandSize
   THEN (* Bad parameters *)
      r/m ← UNDEFINED;
      CF, OF, SF, ZF, AF, PF ← UNDEFINED;
   ELSE (* Perform the shift *)
      CF ← BIT[r/m, ShiftAmt - 1]; (* last bit shifted out on exit *)
      FOR i ← 0 TO OperandSize - 1 - ShiftAmt
      DO
         BIT[r/m, i] ← BIT[r/m, i - ShiftAmt];
      OD;
      FOR i ← OperandSize - ShiftAmt TO OperandSize - 1
      DO
         BIT[r/m,i] ← BIT[inBits,i+ShiftAmt - OperandSize];
      OD;
      Set SF, ZF, PF (r/m);
         (* SF, ZF, PF are set according to the value of the result *)
      Set SF, ZF, PF (r/m);
      AF ← UNDEFINED;
   FI;
FI;
```

SHLD —— Double Precision Shift Left

Opcode	Instruction	Clocks	Description
0F A4	SHLD r/m16,r16,imm8	3/7	r/m16 gets SHL of r/m16 concatenated with r16
0F A4	SHLD r/m32,r32,imm8	3/7	r/m32 gets SHL of r/m32 concatenated with r32
0F A5	SHLD r/m16,r16,CL	3/7	r/m16 gets SHL of r/m16 concatenated with r16
0F A5	SHLD r/m32,r32,CL	3/7	r/m32 gets SHL of r/m32 concatenated with r32

Operation

```
(* count is an unsigned integer corresponding to the last operand of the
instruction, either an immediate byte or the byte in register CL *)
ShiftAmt ← count MOD 32;
inBits ← register; (* Allow overlapped operands *)
IF ShiftAmt = 0
THEN no operation
ELSE
   IF ShiftAmt ≥ OperandSize
   THEN (* Bad parameters *)
      r/m ← UNDEFINED;
      CF, OF, SF, ZF, AF, PF ← UNDEFINED;
   ELSE (* Perform the shift *)
      CF ← BIT[Base, OperandSize - ShiftAmt];
         (* Last bit shifted out on exit *)
   FOR i ← OperandSize - 1 DOWNTO ShiftAmt
   DO
      BIT[Base, i] ← BIT[Base, i - ShiftAmt];
   OF;
   FOR i ← ShiftAmt - 1 DOWNTO 0
   DO
      BIT[Base, i] ← BIT[inBits, i - ShiftAmt + OperandSize];
   OD;
   Set SF, ZF, PF (r/m);
      (* SF, ZF, PF are set according to the value of the result *)
   AF ← UNDEFINED;
   FI;
FI;
```

SHRD和SHLD双精度右移和双精度左移，这两条指令对于位操作和循环移位非常有用，特别是在处理大整数或实现高级数据结构时。它们允许在一个寄存器中组合或拆分位，并且可以用于实现多种算法和编码方案。

首先在`nemu/src/cpu/exec/all-instr.h`中声明函数：

```
make_EHelper(shrd);
make_EHelper(shld);
```

然后补全`nemu/src/cpu/exec/exec.c`

```
  /* 0xa4 */    IDEX(I_G2E, shld), EMPTY, EMPTY, EMPTY,
  /* 0xac */    IDEX(I_G2E, shrd), EMPTY, EMPTY, IDEX(E2G,imul2),
```

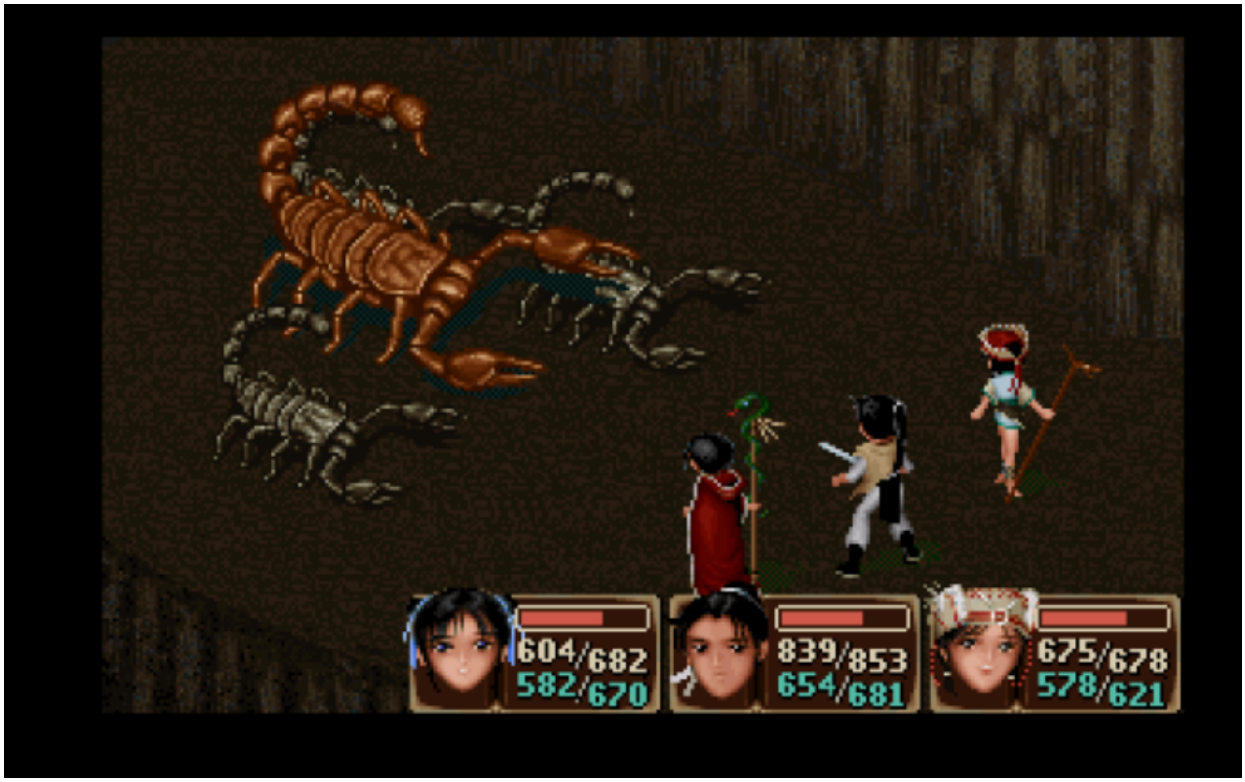
最后在`nemu/src/cpu/exec/logic.c`中根据i386手册实现完整函数体

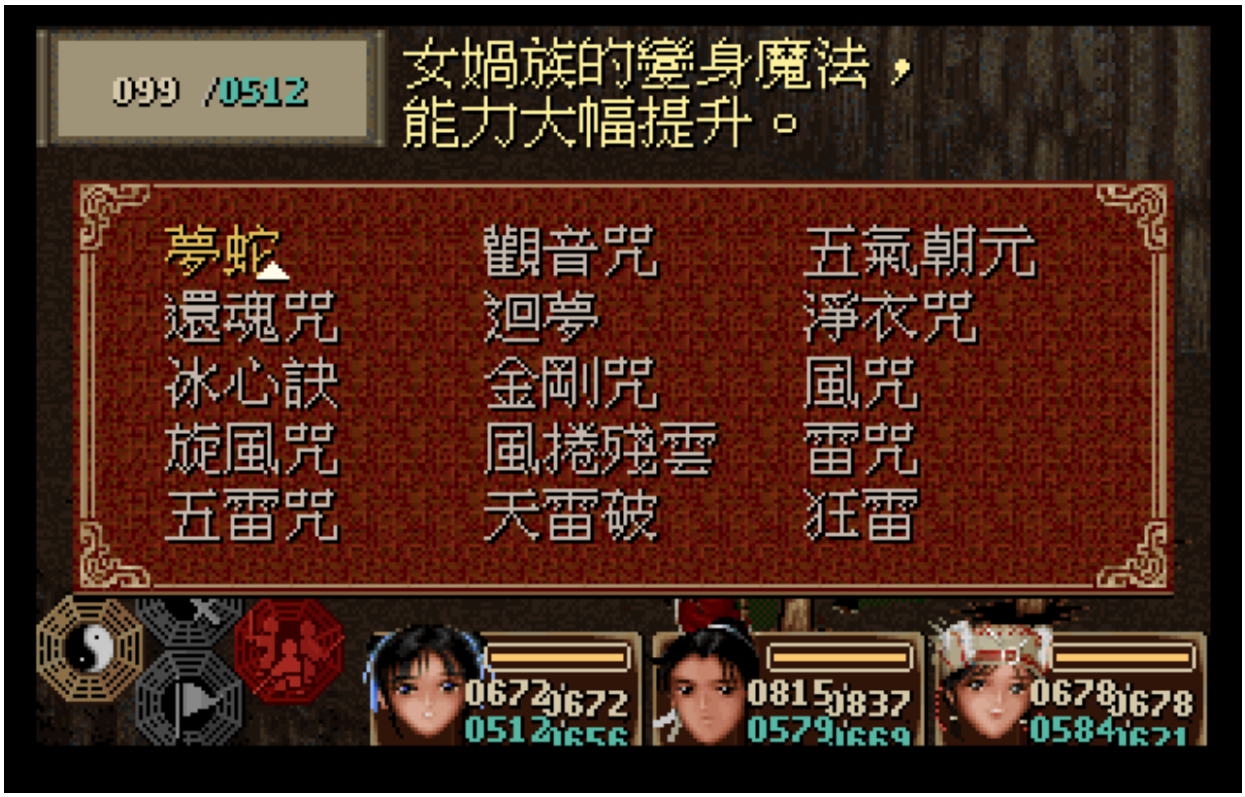
```
make_EHelper(shld)
{
  rtl_shl(&t0,&id_dest->val,&id_src->val);
```

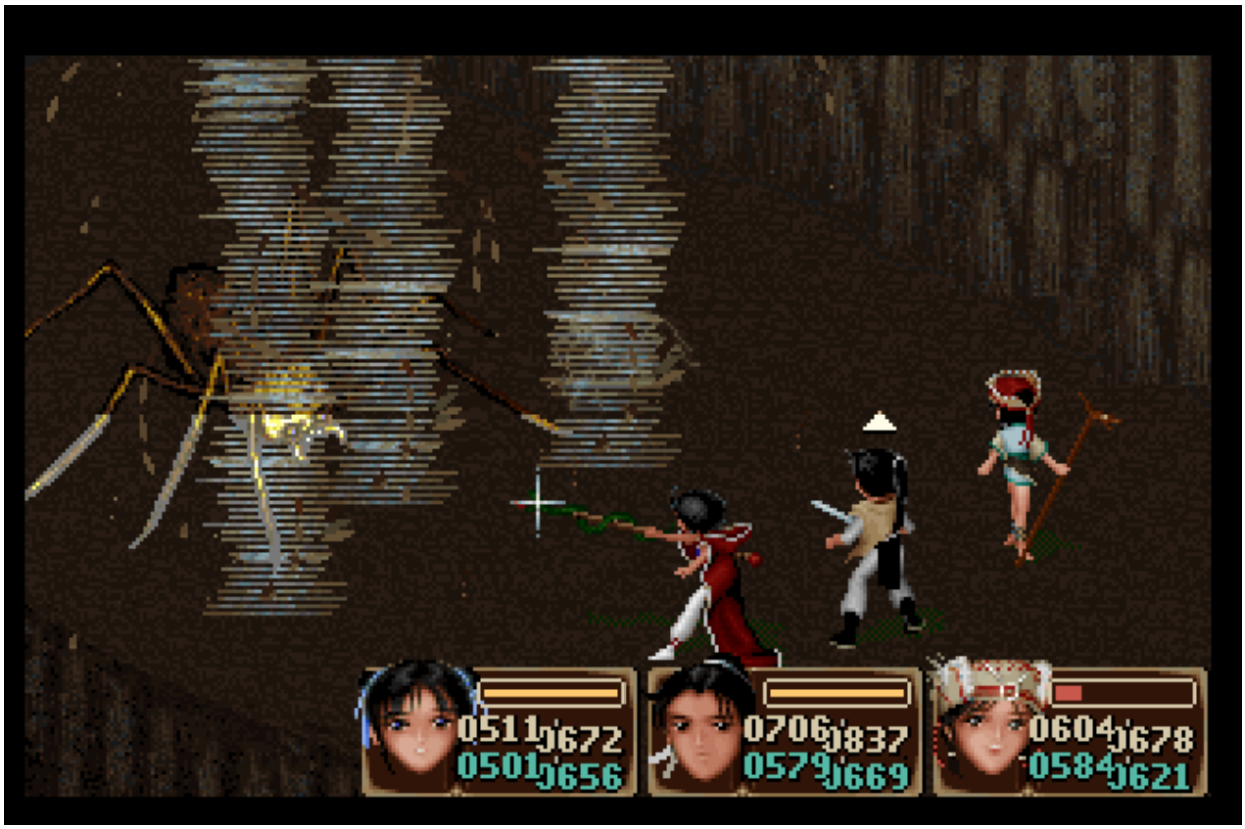
```
  rtl_li(&t2,id_src2->width);
  rtl_shli(&t2,&t2,3);
  rtl_subi(&t2,&t2,id_src->val);
  rtl_shr(&t2,&id_src2->val,&t2);
  rtl_or(&t0,&t0,&t2);
  operand_write(id_dest,&t0);
  rtl_update_ZFSF(&t0,id_dest->width);
  print_asm_template3(shld);
}

make_EHelper(shrd)
{
  rtl_shr(&t0,&id_dest->val,&id_src->val);
  rtl_li(&t2,id_src2->width);
  rtl_shli(&t2,&t2,3);
  rtl_subi(&t2,&t2,id_src->val);
  rtl_shl(&t2,&id_src2->val,&t2);
  rtl_or(&t0,&t0,&t2);
  operand_write(id_dest, &t0);
  rtl_update_ZFSF(&t0, id_dest->width);
  print_asm_template3(shrd);
}
```

至此，PA5已基本完成。

2.7 实验结果







可以看到，可以完成战斗功能。

3 实验感想

PA5的实现颇为坎坷，因为参考资料很少，内容虽少，耗费的时间还挺多。我是只有实现了`shrd`和`shld`之后代码才能跑起来，后来和同学们交流同学们告诉我这是`float.c`实现的问题，如果`float.c`实现的好则不需要添加那两个指令的代码。具体原因我还是不太了解，不过希望老师可以在下一届做PA5的时候说得更详细些。

系统设计更像是一个综合的课程，将之前所学的如计算机组成原理、操作系统等等重要的课全部都串起来了，说实话完整做出来一个PA是一个挺痛苦的过程，但是在痛苦中我也逐渐加深了对计算机的理解。非常感谢学长学姐们留下的宝贵经验，以及每一个在我遇到困难时耐心解答我的问题的人！