南开大学

Nankai University

南　　开　　大　　学

计　算　机　学　院

计算机系统设计实验报告

PA5 - 从一到无穷大: 程序与性能

邵琦

年级：2020 级

专业：计算机科学与技术

指导教师：卢冶

2023 年 6 月 3 日

一、 实验目的

1. 实现浮点数的支持
2. 通过整数来模拟实数的运算
3. 理解 binary scaling 的支持

二、 浮点数的支持

我们用一个 32 位整数来表示一个实数。我们称 binary scaling 方法表示的实数的类型为 FLOAT。我们约定最高位为符号位，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)。从这个约定可以看到，FLOAT 类型其实是实数的一种定点表示。

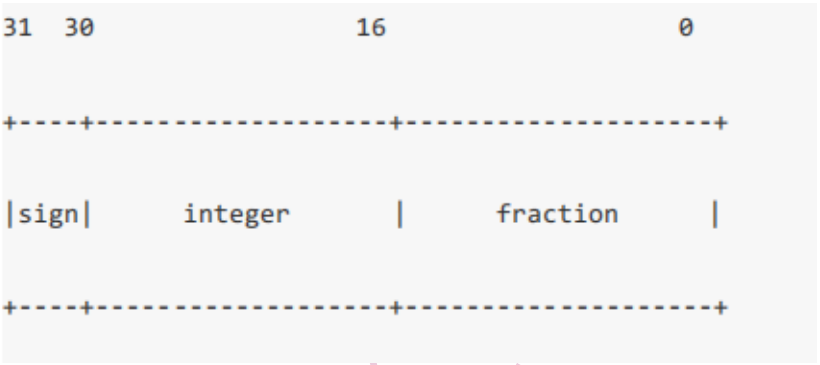

图 1: 浮点数的表示

(一) 实现 f2F 函数

FLOAT 和 float 类型的数据都是 32 位，它们都可以表示 2^{32} 个不同的数。但由于表示方法不一样, FLOAT 和 float 能表示的数集是不一样的。首先，我们需要实现用一个 32 位的整数 FLOAT 来模拟真正的浮点数。

首先我们定义一个 Union 来表示这种浮点数，包含有效位、指数位、符号位。

定义 Union

```
typedef union
{
  float value;
   struct
   {
      unsigned m:23;
      unsigned e:8;
      unsigned s:1;
   };
} float_;
```

然后我们开始编写 f2F 函数进行转化。

首先，我们使用 float_ 结构体类型将 a 转换为 value，这个结构体类型包含了浮点数的指数和尾数。接下来我们考虑两种特殊情况：首先检查指数部分是否为零，如果是零则返回整数值

0，表示浮点数为零。然后，检查 value 的指数部分是否为 0xFF，如果是 0xFF，表示该浮点数为特殊值（如 NaN 或 Infinity）。根据符号位的值，如果为真（非零），则返回负无穷的整数表示 0x80000000，否则返回正无穷的整数表示 0x7fffffff。

接着，代码通过将 value.e 减去 127 并加上 16 得到幂次 pow，表示浮点数的位移量。如果 pow 小于 0，说明计算后的位移量小于 0，此时返回整数值 0，表示浮点数无法转换为合法的整数。然后，代码将 value.m 和 1 左移 23 位的结果进行按位或运算，得到 base 值。这个 base 值将用于计算浮点数的有效数部分。接下来，代码将 pow 减去 23，并根据这个调整后的位移量来计算最终的整数结果。如果 pow 小于 0，则将 base 右移-pow 位；否则将 base 左移 pow 位。

最后，代码根据符号位 value.s 的值来确定最终结果的正负，如果为真（非零），则返回负数，否则返回正数。

f2F 函数

```
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?
     */
    float_ value = {a};
    if (value.e == 0) return 0;
    if (value.e == 0xff) return (value.s ? 0x80000000 : 0x7fffffff);
    int pow = value.e - 127 + 16;
    if (pow < 0)
        return 0;
    int base = value.m | (1 << 23);
    pow -= 23;
    return (value.s ? -1 : 1) * (pow < 0 ? (base >> -pow) : (base << pow));
}
```

(二) 实现 F_mul_F 函数

首先，我们定义一个 union 结构体为了方便进行运算，它包含了包含有效位、指数位以及符号位。

定义一个 union 结构体

```
typedef union
{
    FLOAT value;
    struct
    {
        unsigned f:16;
        unsigned i:15;
```

```
        unsigned s:1;
    };
} Float;
```

然后我们便可以进行乘法运算。首先我们计算符号位，它是通过将 a 和 b 的符号位进行异或操作得到的。如果为真（非零），表示结果为负数，否则表示结果为正数。

接着根据符号位，将 a 和 b 的值取相反数，如果 a.s 和 b.s 为真，则将其对应的值取负数。然后我们分别计算整数部分的乘积 intResult，以及两个交叉乘积 crossProduct1 和 crossProduct2。intResult 是将 a 的整数部分和 b 的整数部分相乘，并乘以 FACTOR 的常量值。crossProduct1 是将 a 的小数部分和 b 的整数部分相乘，crossProduct2 是将 a 的整数部分和 b 的小数部分相乘。

之后，我们计算浮点数部分的乘积 floatResult。它将 a 的小数部分和 b 的小数部分相乘，然后除以 FACTOR，再加上一个判断表达式的结果，即如果余数大于等于 FACTOR/2，则加上 1，否则不加。

最后将之前计算得到的整数部分乘积、交叉乘积和浮点数部分乘积相加，得到最终的结果。并且根据符号位进行判断，如果为真（非零），则返回结果的相反数，否则返回结果本身。

实现 F_mul_F 函数

```
FLOAT F_mul_F(FLOAT _a, FLOAT _b)
{
    Float a = {_a}, b = {_b};
    int sign = a.s ^ b.s;
    a.value = a.s ? -a.value : a.value;
    b.value = b.s ? -b.value : b.value;
    int intResult = a.i * b.i * FACTOR;
    int crossProduct1 = a.f * b.i;
    int crossProduct2 = a.i * b.f;
    FLOAT floatResult = (a.f * b.f / FACTOR) + (a.f * b.f % FACTOR >= FACTOR /
        2);
    FLOAT result = intResult + crossProduct1 + crossProduct2 + floatResult;
    return sign ? -result : result;
}
```

(三) 实现 F_div_F 函数

我们进行除法运算，首先我们使用 assert 断言来确保除数 b 不为零。如果断言失败（b 为零），则程序会终止执行。然后我们分别计算被除数的绝对值 x 和除数的绝对值 y。然后我们便可分别计算除法的商和余数。之后进入 for 循环，用于执行浮点数的小数部分的除法计算。在每次循环中，将 x 左移一位，将 ret 左移一位，然后检查 x 是否大于等于 y。如果是，说明可以将 y 从 x 中减去，并将 ret 的最低位加一。最后，我们根据 a 和 b 的符号位来判断最终的结果的符号，并且返回结果。

实现 F_div_F 函数

```
FLOAT F_div_F(FLOAT a, FLOAT b) {
  // assert(0);
  // return 0;
  assert(b != 0);
```

```
  FLOAT x = Fabs(a);
  FLOAT y = Fabs(b);
  FLOAT ret = x / y;
  x = x % y;

  for (int i = 0; i < 16; i++) {
    x <<= 1;
    ret <<= 1;
    if (x >= y) {
      x -= y;
      ret++;
    }
  }
  if (((a ^ b) & 0x80000000) == 0x80000000) {
    ret = -ret;
  }
  return ret;
}
```

（四） 实现 FLOAT 和 int 之间的相互转换

实现简单，此处不再赘述。

实现 FLOAT 和 int 之间的相互转换

```
static inline int F2int(FLOAT a) {
  /*
  assert(0);
  return 0;
  */
  return a>>16;
}

static inline FLOAT int2F(int a) {
  /*
  assert(0);
  return 0;
  */
  return a<<16;
}
```

（五） 实现 FLOAT 和 int 之间的运算

实现简单，此处不再赘述。

实现 FLOAT 和 int 之间的运算

```
static inline FLOAT F_mul_int(FLOAT a, int b) {
  /*
```

```
  assert(0);
  return 0;
  */
  return a*b;
}

static inline FLOAT F_div_int(FLOAT a, int b) {
  /*
  assert(0);
  return 0;
  */
  return a/b;
}
```

(六) 实验结果

最终能够实现对战。

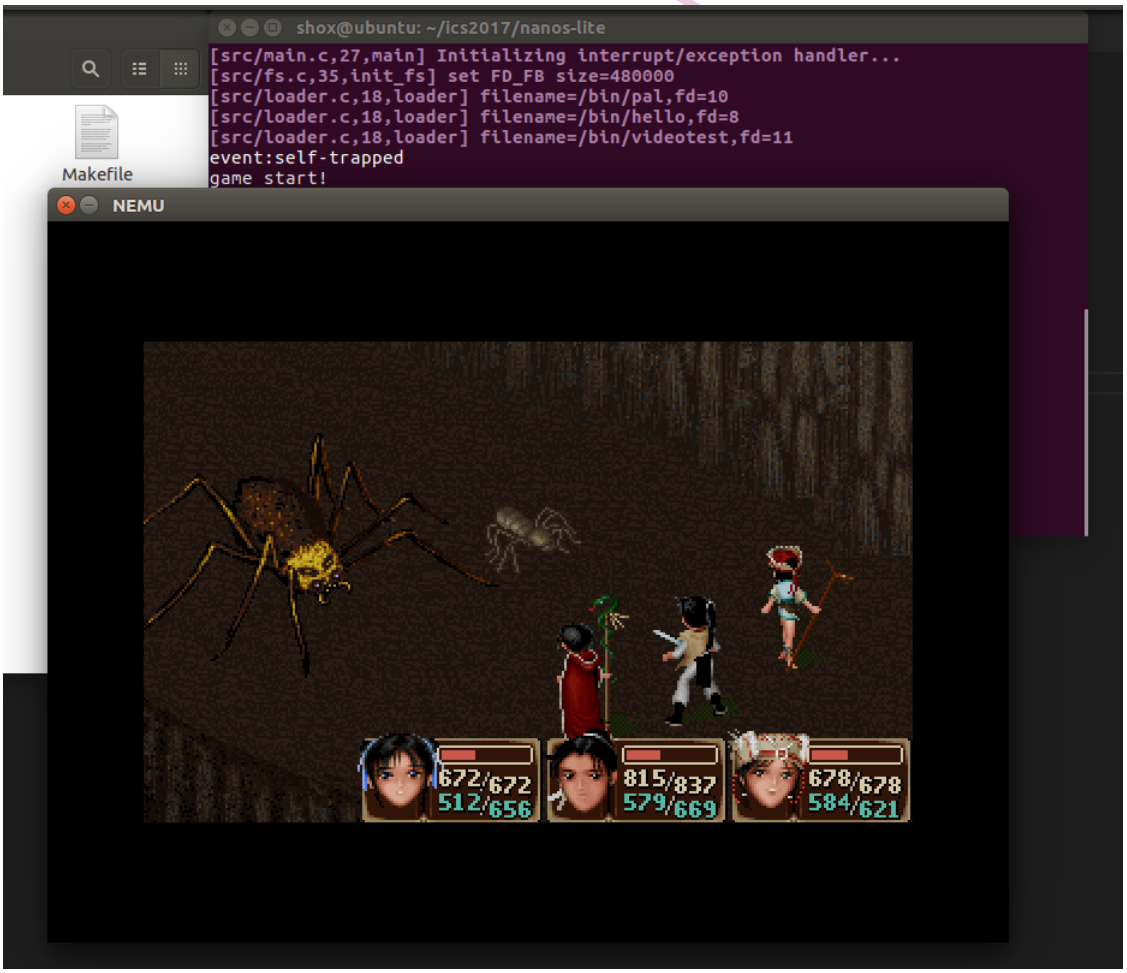

图 2: 实验结果