```
/*
 * linux/kernel/math/mul.c
 *
 * (C) 1991 Linus Torvalds
 */

/*
 * temporary real multiplication routine.
 */
/*
 * 临时实数乘法子程序。
 */

#include <linux/math_emu.h> // 协处理器头文件。定义临时实数结构和 387 寄存器操作宏等。

// 把 c 指针处的 16 字节值左移 1 位（乘 2）。
static void shift(int * c)
{
        __asm__("movl (%0),%%eax ; addl %%eax,(%0)\n\t"
                "movl 4(%0),%%eax ; adcl %%eax,4(%0)\n\t"
                "movl 8(%0),%%eax ; adcl %%eax,8(%0)\n\t"
                "movl 12(%0),%%eax ; adcl %%eax,12(%0)"
                ::"r" ((long) c):"ax");
}

// 2 个临时实数相乘，结果放在 c 指针处（16 字节）。
static void mul64(const temp_real * a, const temp_real * b, int * c)
{
        __asm__("movl (%0),%%eax\n\t"
                "mull (%1)\n\t"
                "movl %%eax,(%2)\n\t"
                "movl %%edx,4(%2)\n\t"
                "movl 4(%0),%%eax\n\t"
                "mull 4(%1)\n\t"
                "movl %%eax,8(%2)\n\t"
                "movl %%edx,12(%2)\n\t"
                "movl (%0),%%eax\n\t"
                "mull 4(%1)\n\t"
                "addl %%eax,4(%2)\n\t"
                "adcl %%edx,8(%2)\n\t"
                "adcl $0,12(%2)\n\t"
                "movl 4(%0),%%eax\n\t"
                "mull (%1)\n\t"
                "addl %%eax,4(%2)\n\t"
                "adcl %%edx,8(%2)\n\t"
                "adcl $0,12(%2)"
                ::"b" ((long) a),"c" ((long) b),"D" ((long) c)
                :"ax","dx");
}

// 仿真浮点指令 FMUL。
// 临时实数 src1 * scr2 ➔ result 处。
```

```
46 void fmul(const temp_real * src1, const temp_real * src2, temp_real * result)
47 {
48         int i,sign;
49         int tmp[4] = {0,0,0,0};
50
   // 首先确定两数相乘的符号。符号值等于两者符号位异或值。然后计算乘后的指数值。相乘时
   // 指数值需要相加。但是由于指数使用偏执数格式保存，两个数的指数相加时偏置量也被加了
   // 两次，因此需要减掉一个偏置量值（临时实数的偏置量是 16383）。
51         sign = (src1->exponent ^ src2->exponent) & 0x8000;
52         i = (src1->exponent & 0x7fff) + (src2->exponent & 0x7fff) - 16383 + 1;
   // 如果结果指数变成了负值，表示两数相乘后产生下溢。于是直接返回带符号的零值。
   // 如果结果指数大于 0x7fff，表示产生上溢，于是设置状态字溢出异常标志位，并返回。
53         if (i<0) {
54                 result->exponent = sign;
55                 result->a = result->b = 0;
56                 return;
57         }
58         if (i>0x7fff) {
59                 set_OE();
60                 return;
61         }
   // 如果两数尾数相乘后结果不为 0，则对结果尾数进行规格化处理。即左移结果尾数值，使得
   // 最高有效位为 1。同时相应地调整指数值。如果两数的尾数相乘后 16 字节的结果尾数为 0，
   // 则也设置指数值为 0。最后把相乘结果保存在临时实数变量 result 中。
62         mul64(src1,src2,tmp);
63         if (tmp[0] || tmp[1] || tmp[2] || tmp[3])
64                 while (i && tmp[3] >= 0) {
65                         i--;
66                         shift(tmp);
67                 }
68         else
69                 i = 0;
70         result->exponent = i | sign;
71         result->a = tmp[2];
72         result->b = tmp[3];
73 }
74
```