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

/*
 * temporary real addition routine.
 *
 * NOTE! These aren't exact: they are only 62 bits wide, and don't do
 * correct rounding. Fast hack. The reason is that we shift right the
 * values by two, in order not to have overflow (1 bit), and to be able
 * to move the sign into the mantissa (1 bit). Much simpler algorithms,
 * and 62 bits (61 really - no rounding) accuracy is usually enough. The
 * only time you should notice anything weird is when adding 64-bit
 * integers together. When using doubles (52 bits accuracy), the
 * 61-bit accuracy never shows at all.
 */
/*
 * 临时实数加法子程序。
 *
 * 注意！ 这些并不精确：它们只有 62 比特宽度，并且不能进行正确地舍入操作。
 * 这些仅是草就之作。原因是为了不会溢出（1 比特），我们把值右移了 2 位，
 * 并且使得符号位（1 比特）能够移入尾数中。这是非常简单的算法，而且 62 位
 * （实际上是 61 位 - 没有舍入）的精度通常也足够了。只有当你把 64 位的整数
 * 相加时才会发觉一些奇怪的问题。当使用双精度（52 比特精度）数据时，是永
 * 远不可能超过 61 比特精度的。
 */

#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。

// 求一个数的负数（二进制补码）表示。
// 把临时实数尾数（有效数）取反后再加 1。
// 参数 a 是临时实数结构。其中 a、b 字段组合是实数的有效数。
#define NEGINT(a) \
__asm__("notl %0 ; notl %1 ; addl $1,%0 ; adcl $0,%1" \
        :"=r" (a->a),"=r" (a->b) \
        :"" (a->a),"1" (a->b))

// 尾数符号化。
// 即把临时实数变换成指数和整数表示形式，以便于仿真运算。因此我们这里称其为仿真格式。
static void signify(temp_real * a)
{
// 把 64 位二进制尾数右移 2 位（因此指数需要加 2）。因为指数字段 exponent 的最高比特位是
// 符号位，所以若指数值小于零，说明该数是负数。于是则把尾数用补码表示（取负）。然后把
// 指数取正值。此时尾数中不仅包含移过 2 位的有效数，而且还包含数值的符号位。
// 30 行上：%0 - a->a；%1 - a->b。汇编指令“shrdl $2, %1, %0”执行双精度（64 位）右移，
// 即把组合尾数<b,a>右移 2 位。由于该移动操作不会改变%1（a->b）中的值，因此还需要单独
// 对其右移 2 位。
        a->exponent += 2;
        __asm__("shrdl $2,%1,%0 ; shrl $2,%1"   // 使用双精度指令把尾数右移 2 位。
                :"=r" (a->a),"=r" (a->b)
```

```
32                  :"" (a->a), "1" (a->b));
33          if (a->exponent < 0)                    // 是负数，则尾数用补码表示（取负值）。
34                  NEGINT(a);
35          a->exponent &= 0x7fff;                  // 去掉符号位（若有）。
36  }
37
    // 尾数非符号化。
    // 将仿真格式转换为临时实数格式。即把指数和整数表示的实数转换为临时实数格式。
38  static void unsignify(temp_real * a)
39  {
    // 对于值为 0 的数不用处理，直接返回。否则，我们先复位临时实数格式的符号位。然后判断
    // 尾数的高位 long 字段 a->b 是否带有符号位。若有，则在 exponent 字段添加符号位，同时
    // 把尾数用无符号数形式表示（取补）。最后对尾数进行规格化处理，同时指数值作相应递减。
    // 即执行左移操作，使得尾数最高有效位不为 0（最后 a->b 值表现为负值）。
40          if (!(a->a || a->b)) {                  // 若值为 0 就返回。
41                  a->exponent = 0;
42                  return;
43          }
44          a->exponent &= 0x7fff;                  // 去掉符号位（若有）。
45          if (a->b < 0) {                         // 是负数，则尾数取正值。
46                  NEGINT(a);
47                  a->exponent |= 0x8000;          // 临时实数添加置符号位。
48          }
49          while (a->b >= 0) {
50                  a->exponent--;                  // 对尾数进行规格化处理。
51                  __asm__("add1 %0,%0 ; adc1 %1,%1"
52                          :"=r" (a->a), "=r" (a->b)
53                          :"" (a->a), "1" (a->b));
54          }
55  }
56
    // 仿真浮点加法指令运算。
    // 临时实数参数 src1 + src2 ➔ result。
57  void fadd(const temp_real * src1, const temp_real * src2, temp_real * result)
58  {
59          temp_real a,b;
60          int x1,x2,shift;
61
    // 首先取两个数的指数值 x1、x2（去掉符号位）。然后让变量 a 等于其中最大值，shift 为指数
    // 差值（即相差 2 的倍数值）。
62          x1 = src1->exponent & 0x7fff;
63          x2 = src2->exponent & 0x7fff;
64          if (x1 > x2) {
65                  a = *src1;
66                  b = *src2;
67                  shift = x1-x2;
68          } else {
69                  a = *src2;
70                  b = *src1;
71                  shift = x2-x1;
72          }
    // 若两者相差太大，大于等于 2 的 64 次方，则我们可以忽略小的那个数，即 b 值。于是直接返
    // 回 a 值即可。否则，若相差大于等于 2 的 32 次方，那么我们可以忽略小值 b 中的低 32 位值。
```

```
    // 于是我们把 b 的高 long 字段值 b.b 右移 32 位，即放到 b.a 中。然后把 b 的指数值相应地增
    // 加 32 次方。即指数差值减去 32。这样调整之后，相加的两个数的尾数基本上落在相同区域中。
73          if (shift >= 64) {
74                  *result = a;
75                  return;
76          }
77          if (shift >= 32) {
78                  b.a = b.b;
79                  b.b = 0;
80                  shift -= 32;
81          }
    // 接着再进行细致地调整，以将相加两者调整成相同。调整方法是把小值 b 的尾数右移 shift
    // 各比特位。这样两者的指数相同，处于同一个数量级下。我们就可以对尾数进行相加运算了。
    // 相加之前我们需要先把它们转换成仿真运算格式。在加法运算后再变换会临时实数格式。
82          __asm__("shrd1 %4,%1,%0 ; shr1 %4,%1"                   // 双精度（64 位）右移。
83                  :"=r" (b.a), "=r" (b.b)
84                  :"" (b.a), "1" (b.b), "c" ((char) shift));
85          signify(&a);                                            // 变换格式。
86          signify(&b);
87          __asm__("add1 %4,%0 ; adc1 %5,%1"                       // 执行加法运算。
88                  :"=r" (a.a), "=r" (a.b)
89                  :"" (a.a), "1" (a.b), "g" (b.a), "g" (b.b));
90          unsignify(&a);                                          // 再变换会临时实数格式。
91          *result = a;
92  }
93
```