Рамиль Альварес Х.

Алгоритмы троичной арифметики

Москва
Фонд «Новое тысячелетие»
2012

УДК 16
ББК 87.4
Р21

Р21 Рамиль Альварес Х. Алгоритмы троичной арифметики. – М.,Фонд «Новое тысячелетие», 2012. – 20 с.

ISBN 978-5-86947-068-3

Оглавление

Сравнение чисел в троичной симметричной системе

Операции сравнения чисел занимают важное место в системах и языках программирования, в которых в той или иной форме имеются операции сравнения: равно, не равно, больше, меньше или равно, меньше и больше или равно. Результат этих операции – булево значение (true или false), используемое в конструкциях if – then и if – then – else. Реализация операций сравнения основана на выполнении вычитания с последующим исследованием знака вычитания для получения результата. Отметим, что при вычитании могут возникнуть переносы в старшие разряды.

Рассматривая преимущества симметрических (Шеннон К. [1]) и троичной симметричной (Кнут Д. [2]) систем счисления, не указывают еще одно важное их преимущество – возможность поразрядного сравнения целых чисел с трехзначным результатом: меньше (-1), равно (0) и больше (1). Такое сравнение производится, начиная со старших разрядов до первого несовпадения. Результат сравнения несовпадающих разрядов определяют конечный результат. При совпадении всех разрядов – числа равны. Использование результата сравнения в конструкциях альтернативного вызова одной из трех процедур в зависимости от знака значения вершины стека, введенных на машине «Сетунь 70» [3] и в диалоговой системе структурированного программирования ДССП [4], позволяют реализовать все возможные варианты сравнения. Поразрядное сравнение быстрее, чем обычное сравнение вычитанием: в среднем поразрядное сравнение требует выполнения сравнений разрядов равное половине разрядности числа.

Опишем на языке С [5] алгоритмы сравнения целых троичных чисел одинаковой и неодинаковой длины, а также сравнение чисел с плавающей точкой.

Сравнение чисел одинаковой длины

Числа одинаковой длины представляют целые числа, если точка фиксирована после младшего разряда, или дробные числа, если она фиксирована после определенного разряда.

Троичное число с заданной длиной представления можно описать на языке С как массив типа char длины N (параметра представления), элементы которого принимают значения -1, 0 и 1.

Функция `icmp(x,y)` производит сравнение чисел `x` и `y`. Сравнение тритов чисел `x` и `y` производится до первого несовпадения, которое и определяет результат. Если все триты совпали, то результат – нуль.

```
int icmp (char x[], char y[])
{int i=N;
while (x[i] == y[i] && i-- >= 0);
if (i<0) return 0;
if (x[i] > y[i]) return 1;
return -1;
}
```

При поразрядном сравнении количество требуемых сравнений тритов в среднем равно половине разрядности числа.

Сравнение чисел неодинаковой длины

Троичное число с нефиксированной длиной представления можно описать на языке С как структуру

```
struct trin {int n; char trit[K];};
```

где n – число тритов в представлении числа (для числа нуль n=0), а массив `trit` – значения тритов представления (при этом `trit[n-1]` отлично от нуля), `K` – параметр реализации. Функция `tcmp(x,y)` производит сравнение чисел `x` и `y`. Если длина числа `x` больше длины `y`, результат сравнения есть значение старшего разряда числа `x`. Если длина числа `y` больше длины `x`, результат сравнения есть обратное значение старшего разряда числа `y`. При равных длинах `x` и `y` производится сравнение тритов чисел `x` и `y` до первого несовпадения, которое и определяет результат. Если все триты совпали, то результат – нуль.

```
int tcmp (struct trin x, struct trin y)
{int i= x.n;
if (y.n<i) return x.trit[i-1];
if (y.n>i) return -y.trit[y.n-1];
i-- ;
while (x.trit[i] == y.trit[i] && i-- >= 0);
if (i<0) return 0;
if (x.trit[i] > y.trit[i]) return 1;
return -1;
}
```

Функция `acmp(w,z)` производит сравнение абсолютных величин `w` и `z`. Функция используется в алгоритме деления Кнута и его модификации. Если длина структуры `w` больше длины `z`, то результат будет 1, а если длина `w` меньше длины `z`, то результат будет -1. При равенстве длин производится сравнение тритов чисел `w` и `z` с учетом знаков чисел до первого несовпадения, которое и определяет результат. Если все триты совпали, то результат – нуль.

```
int acmp(struct trin w, struct trin z)
{int i=w.n,sw = sgn(w),sz= sgn(z);
 if (i>z.n) return 1;
 if (i-- == z.n)
{while (sw*w.trit[i] == sz*z.trit[i] &&
i-- >= 0);
if (i<0) return 0;
if (sw*w.trit[i] > sz*z.trit[i]) return 1;
} // if
return -1;
}
```

Сравнение чисел с плавающей точкой

Троичное число с плавающей точкой опишем на языке С как структуру

```
struct ftrin {char n[5]; char trit[N];};
```

где массив n (6 тритов, что равносильно трайту) – порядок числа, а массив `trit` (`N` – параметр реализации) – мантисса числа с точкой после старшего трита и он отличен от нуля, т.е. модуль мантиссы принадлежат интервалу (0.5 ; 1.5) [6]. Число нуль представляется нулевой мантиссой и с минимальным порядком (все триты массива n равны -1).

```
int fcmp (struct ftrin x, struct ftrin y)
{int j= 5;
// Сравнение порядков чисел
while (x.n[j] == y.n[j] && j-- >=0);
if (j >=0)
if(x.n[j] > y.n[j]) return x.trit[N-1];
else return -y.trit[N-1];
j= N-1;
// Порядки чисел равны – сравнение мантисс
while (x.trit[j] == y.trit[j] && j-- >=0);
if (j <0) return 0;
if(x.trit[j] > y.trit[j]) return 1;
 return -1;
}
```

Литература

1. Shannon C.E. A symmetrical notation for numbers // The American Mathematical Monthly, vol. 57, n. 2 (Feb. 1950), pp. 90-93.
2. Кнут Д. Искусство программирования, т. 2. Получисленные алгоритмы. 3-е изд. — М.: «Вильямс», 2007.
3. Брусенцов Н.П., Рамиль Альварес Х. Троичная ЭВМ «Сетунь 70» // Труды SORUCOM-2011. Вторая Международная конференция «Развитие вычислительной техники и ее программного обеспечения в России и странах бывшего СССР». – Великий Новгород, 2011. С. 71-75.
4. Брусенцов Н.П., Захаров В.Б., Руднев И.А., Сидоров С.А., Чанышев Н.А. Методические указания по программирования в ДССП. – М.: Ротапринт НИВЦ МГУ, 1988. 39 с.
5. Березин Б.И., Березин С.Б. Начальный курс С и С++. – М.: ДИАЛОГ МИФИ, 2005.
6. Брусенцов Н.П., Маслов С.П. и др. Малая цифровая вычислительная машина “Сетунь”. – М.: Изд–во МГУ, 1965.

Троичное деление целых чисел

Брусенцовым Н.П. описаны алгоритмы деления в троичной симметричной системе [1] в предположении, что значения делимого и делителя – нормализованные троичные числа, определенные в [2], т.е. равны нулю или их абсолютные величины принадлежат интервалу (0.5 ; 1.5).

Брусенцов отмечает, что критерии, является ли нулем очередная цифра частного, различаются в системе с неотрицательными цифрами и в симметричной системе. В последней возможны два варианта определения того, что очередной трит частного будет нулем:

- если очередное делимое по модулю не больше ½ делителя,
- если очередное делимое по модулю меньше ½ делителя.

Различие этих вариантов состоит в представлении половины i-того трита результата:

- при первом варианте она представляется с избытком: +-(-)… (-+(+)…);
- при втором – с недостатком: 0+(+) … (0-(-)…).

В основном цикле алгоритма для определения очередного трита частного используется удвоенное значение делимого, из модуля которого на каждом шаге вычитается модуль делителя и, если результат больше нуля, то цифра частного отлична от нуля и из результата повторно вычитается модуль делителя. Таким

образом, для получения частного требуется одно сложение (для получения удвоенного делимого), а число вычитаний равно числу разрядов частного плюс число ненулевых цифр в нем.

В работе [3] описан алгоритм деления целых чисел, являющийся модификацией алгоритма Брусенцова. При определении очередного трита частного производится сравнение модуля удвоенного делимого с модулем делителя, и если трит частного не нуль, то из удвоенного делимого вычитается или к нему прибавляется модуль удвоенного делителя.

Сравнение модулей чисел производится потритно специальной операцией, аналогичной введенной в [3] операции потритного сравнения чисел.

Таким образом, в модифицированном алгоритме Брусенцова для получения частного требуется два сложения (для удвоения делимого и делителя) и столько операций типа сложения, сколько имеется ненулевых тритов в частном.

В решении упражнения 31 раздела 4.3.1 Кнутом Д. [4] приведен изящный алгоритм деления целых чисел в троичной симметричной системе. В основном цикле определения трита частного выполняются следующие действия:

- при совпадении знаков делимого и делителя очередной трит частного равен 1 и из делимого вычитается делитель. В противном случае трит равен –1 и к делимому прибавляется делитель;
- если полученное значение больше по модулю делимого или если их модули равны и знаки необработанной части делимого и полученного значения совпадают, то трит частного равен нулю;
- при ненулевом значении трита частного делимому присваивается вычисленное значение разницы (суммы) и во всех случаях к очередному делимому приписывается справа следующий трит необработанной части делимого.

Таким образом, если на втором шаге использовать потритное сравнение чисел, в алгоритме Кнута для получения частного число операций типа сложения равно числу тритов частного.

Число операций в алгоритме Кнута может быть уменьшено, если сначала определять, является ли очередной трит частного нулем, для чего сравнивать модули удвоенного делимого и делителя.

Описание алгоритмов

Описание алгоритмов приводится на языке C [5].

Целое троичное число представляется в виде структуры

```
struct trin { int n; char trit [K];};
```

где `n` – число тритов, а элементы массива `trit` – значение числа: младший трит хранится в элементе с индексом 0, а старшие – левее, элементы массива принимают значения –1, 0 и 1. Для числа нуль `n` равно нулю. `K` – параметр алгоритма.

Вспомогательные функции

Результатом функции `sgn(w)` является знак числа `w`. Если `n` равно нулю, то результат будет нуль, в противном случае – значение элемента массива `trit` с индексом `n-1`.

```
int sgn(struct trin w)
{      if (w.n == 0) { return 0; }
       else { return w.trit[w.n-1]; }
}
```

Результатом функции `sgnk(x,k)` есть знак необработанной части делимого, представленной тритами структуры `x` с индексами с `k`-го по нулевой. При этом ищется первый слева направо ненулевой трит, который и определяет результат. В случае, когда все триты – нули, результат есть нуль.

```
int sgnk(struct trin x, int k)
{ int i=k;
       if (i >= 0)
{while (x.trit[i] == 0 && i-- >= 0);
 if (i >= 0) return x.trit[i];
}
return 0;
}
```

Функция `shift(&w,t)` определяет новое значение `w`. Если `w` не нуль, то производится сдвиг числа `w` влево на один трит, а триту с индексом нуль присваивается значение `t`. Если `w` равно нулю и `t` отлично от нуля, то триту с индексом нуль присваивается значение `t`. Если `w` и `t` равны нулю, `w` не изменяется.

```
void shift(struct trin *w, char t)
{ int j;
if (w->n != 0)
 {for (j=w->n; j>0; j--)
w->trit[j]=w->trit[j-1];
w->trit[0]=t; w->n += 1;
 }
else if (t != 0) {w->trit[0]=t; w->n=1;}
}
```

Функция `acmp(w,z)` производит сравнение модулей `w` и `z`. Если длина структуры `w` больше длины `z`, то результат будет 1, а если длина `w` меньше длины `z`, то результат будет -1. При равенстве длин производится сравнение тритов чисел `w` и `z` с учетом знаков чисел до первого несовпадения, которое и определяет результат. Если все триты совпали, то результат – нуль.

```
int acmp(struct trin w, struct trin z)
{int i=w.n,sw = sgn(w),sz= sgn(z);
 if (i>z.n) return 1;
 if (i-- == z.n)
{while (sw*w.trit[i] == sz*z.trit[i] && i-- >= 0);
if (i<0) return 0;
if (sw*w.trit[i] > sz*z.trit[i]) return 1;
} // if
return -1;
}
```

Функция `red(&res)` производит удаление старших тритов с нулевыми значениями структуры `res`, являющейся параметром функции. Она используется в функциях сложения `sum( )`, вычитания `res( )`, в алгоритме Кнута `knuth( )` и в функции собственно деления `dvs0( )`.

```
void red(struct trin *res)
{int i;
for (i = res->n-1; i >= 0 && res->trit[i] == 0; i--);
res->n=i+1;
}
```

В функциях сложения `sum(x,y,&res)` и вычитания `res(x,y,&res)` параметрами являются структуры `x` и `y` – аргументы, а структура `res` – результат.

```
void sum(struct trin x, struct trin y,
struct trin *res)
{int i, s, c1 = 0, j=max(x.n,y.n);
res->n=j;
for (i = 0; i < j; i++)
  {if (i>=x.n) s = y.trit[i]+c1; else
if (i>=y.n) s = x.trit[i]+c1; else
  s = x.trit[i]+y.trit[i]+c1;
  if (s<-1) { res->trit[i] = s+3; c1 =-1;} else
  if (s>1) { res->trit[i] = s-3; c1 =1;} else
 { res->trit[i] = s; c1 =0;}
  } // for
if (c1!=0) {res->trit[j] = c1; res->n=j+1;}
else red(res);
}

void res (struct trin x, struct trin y,
struct trin *res)
{int i, s, c1 = 0, j=max(x.n,y.n);
res->n=j;
```

```
for (i = 0; i < j; i++)
  {if (i>=x.n) s = -y.trit[i]+c1; else
if (i>=y.n) s = x.trit[i]+c1; else
  s = x.trit[i]-y.trit[i]+c1;
  if (s<-1) { res->trit[i]= s+3; c1 =-1;} else
  if (s>1) { res->trit[i] = s-3; c1 =1;} else
 { res->trit[i] = s; c1 =0;}
  } // for
if (c1!=0) {res->trit[j] = c1; res->n=j+1;}
else red(res);
}
```

Алгоритм деления Кнута

Функция `knuth(x,y,&rs,&rs1)` производит деление `x` на `y`, результатом является пара чисел: `rs` - частное и `rs1` - остаток.

Сравнение модулей очередного делимого и возможного его нового значения производится потритно (обращение к функции `acmp(w,z))`, в результате для получения частного число операций типа сложения равно числу тритов делимого.

```
void knuth(struct trin x, struct trin y,
struct trin *rs,struct trin *rs1)
{ int cwz, i, sy=sgn(y);
struct trin w, z;
if (sy == 0) printf("Division by zero");
else { w.n = 0; rs->n = x.n;
for (i=x.n-1 ; i >= 0; i--)
{shift(&w,x.trit[i]);
if (sgn(w) == sy) {res(w,y,&z);rs->trit[i]=1;}
 else {sum (w,y,&z); rs->trit[i]=-1 ;}
cwz = acmp(w,z);
if(cwz == -1||(cwz == 0 &&(sgnk(x,i-1) == sgn(z))))
rs->trit[i]=0;
else w=z;
} // for
red(rs);
*rs1= w;
} // else
}
```

Модифицированный алгоритм Кнута

Функция `mknuth(x,y,&rs,&rs1)` с теми же параметрами, что и в алгоритме Кнута.

В начале вычисляются удвоенные значения делимого `(xx)` и делителя `(yy)`, после чего производится обращение к функции собственно деления

dvs0(xx,y,yy,rs,&w1), в которой будет получено значение удвоенного остатка (w1). Для определения остатка производится обращение к функции деления на 2 (dvs2(w1,rs1)).

```
void mknuth(struct trin x, struct trin y,
struct trin *rs,struct trin *rs1)
{struct trin xx, yy,w1;
int i, k=y.n, sy=sgn(y);
if (sy == 0) printf("Division of zero ");
else
{sum(x,x,&xx);
 sum(y,y,&yy);
 dvs0(xx,y,yy,rs,&w1);
 dvs2(w1,rs1);
}
}
```

К функции dvs0(xx,y,yy,&rs,&rs1) возможны два вида обращения:

- обращение, производимое в функции mknuth, где xx – удвоенное делимое, y - делитель и yy - удвоенный делитель;
- обращение, при котором xx – делимое, y - половина делителя и yy - делитель; такое обращение производится в функции dvs2(w1,rs1) – деление на 2 и может производиться при четном делителе, например при делении на 8 и 10.

```
void dvs0(struct trin  xx, struct trin y,
struct trin yy, struct trin *rs, struct trin *rs1)
{ int cwy, i,sy= sgn(y),sw;
struct trin w ;
rs->n=xx.n;
w.n=0;
for (i=xx.n-1 ; i>=0; i--)
{shift(&w,xx.trit[i]);
sw= sgn(w);
cwy = acmp(w,y);
if(cwy == 1||(cwy == 0 && sgnk(xx,i-1) == sw))
if (sw == sy) {res(w,yy,&w);rs->trit[i]=1;}
 else {sum(w,yy,&w); rs->trit[i]=-1 ;}
else rs->trit[i]=0 ;
} // for
red(rs);
*rs1 = w;
}
```

Функции dvs2(w1,&rs1) вычисляет половину w1.

```
void dvs2(struct trin xx, struct trin *rs)
{struct trin y, yy, w;
y.n=1; y.trit[0]=1; // y=1
yy.n=2; yy.trit[1]=1; yy.trit[0]=-1; // yy=2
dvs0(xx,y,yy,rs,&w);
}
```

Заключение

В модифицированном алгоритме Кнута для вычисления частного требуется два сложения (для удвоения делимого и делителя) и операций типа сложения, число которых равно числу ненулевых тритов частного. Функция собственно деления `dvs0(xx,y,yy,&rs,&rs1)` позволяет ускорить вычисления в случае, когда делитель четный и его половина известна.

Литература

1. Брусенцов Н.П. Алгоритмы деления для троичного кода с цифрами 0, 1, –1 // Вычислительная техника и вопросы кибернетики. Вып. 10. – Л.: Изд–во ЛГУ, 1974. С. 39–44.
2. Брусенцов Н.П., Маслов С.П. и др. Малая цифровая вычислительная машина «Сетунь». – М.: Изд–во МГУ, 1965.
3. Рамиль Альварес Х. Деление целых чисел в троичной симметричной системе // Программные системы и инструменты. Тематический сборник № 12. М.: Изд-во факультета ВМК МГУ, 2011. С. 228-234.
4. Кнут Д. Искусство программирования, т. 2. Получисленные алгоритмы. 3-е изд. — М.: «Вильямс», 2007.
5. Березин Б.И., Березин С.Б. Начальный курс С и С++. – М.: ДИАЛОГ МИФИ, 2005.

Извлечение квадратного корня в троичной симметричной системе

В работе [1] описан алгоритм извлечения «в столбик» квадратного корня в троичной симметричной системе. В данной статье приводится описание алгоритма на языке С [2].

Общие правила извлечения квадратного корня

Также как при вычислении квадратного корня для систем с неотрицательными значениями цифр, подкоренное число N разбивается на пары тритов влево и вправо от точки. Обозначим через N_i – целое число, получаемое из первых слева направо i пар тритов, A_i – целое значение корня из этого числа, D_i – i–я пара тритов, d_i – значение i–го трита корня.

Алгоритм извлечения квадратного корня в «столбик» для троичной системы счисления можно описать следующими формулами:

$$N_1 = D_1,$$
$$A_1 = d_1,$$
$$B_1 = N_1 - A_1^2,$$
$$N_i = A_i^2 + B_i,$$
$$C_{i+1} = B_i \cdot 3^2 + D_{i+1},$$
$$B_{i+1} = C_{i+1} - 2 \cdot A_i \cdot 3 \cdot d_{i+1} - d_{i+1}^2 = C_{i+1} - (2 \cdot A_i \cdot 3 + d_{i+1}) \cdot d_{i+1},$$
$$A_{i+1} = A_i \cdot 3 + d_{i+1}.$$

Отметим, что действия производятся над целыми числами, а точка в результате будет после k–го трита, если в подкоренном числе она стояла после k–й пары.

Заметим, что $B_i \cdot 3^2 + D_{i+1}$ при ручном использовании алгоритма выполняется приписыванием справа к значению B_i двух тритов пары D_{i+1}, а $2 \cdot A_i \cdot 3 + d_{i+1}$ и $A_i \cdot 3 + d_{i+1}$ – приписыванием трита d_{i+1} к числам соответственно $2 \cdot A_i$ и A_i.

Определение d_1

После разбиения числа на пары тритов возможны два случая:

– первая пара состоит из одного трита. Если рассмотреть подкоренное число как число с точкой после первого трита, имеем число в интервале (.5:1.5). Корень будет, грубая оценка – в интервале (.7 : 1.23), т.е. первый трит корня d_1 будет “+”;

– первая пара содержит два трита. Если рассмотреть подкоренное число как число с точкой перед первой парой, имеем число в интервале (.166… :.5) и первый трит корня будет “0” или “+”. Если подкоренное число больше ¼, корень больше чем ½, т.е. первый трит корня d_1 будет “+”. В противном случае первый трит корня d_1 будет “0”. Заметим, что ¼ не представляется точно в

троичной симметричной системе и имеет представление в виде бесконечной последовательности .+– (+–) ... .

Определение d_{i+1}

При d_{i+1} отличном от “0” имеем

$$B_{i+1} = (C_{i+1} - 1) - 2 \cdot A_i \cdot 3 \cdot d_{i+1},$$

т.е. d_{i+1} есть частное от деления $C_{i+1}–1$ на $2 \cdot A_i \cdot 3$ и вследствие алгоритма деления в троичной системе d_{i+1} равно единице, если

$$2 \cdot |C_{i+1}–1| \geq 2 \cdot A_i \cdot 3,$$

или проще

$$|C_{i+1}–1| > A_i \cdot 3.$$

При C_{i+1} положительном и $C_{i+1} > A_i \cdot 3$ и $B_{i+1} = C_{i+1} – 2 \cdot A_i \cdot 3 – 1$ значение d_{i+1} будет “+”. При C_{i+1} отрицательном и $C_{i+1} < –A_i \cdot 3$ и $B_{i+1} = C_{i+1} + 2 \cdot A_i \cdot 3 – 1$ значение d_{i+1} будет “–”.

Рассмотрим случай $|C_{i+1}| = A_i \cdot 3$, тогда B_{i+1} можно представить в виде

$$B_{i+1} = C_{i+1} – 2 \cdot A_i \cdot 3 \cdot d + (½)^2 – ¼$$

где d равняется ½ для $C_{i+1} > 0$ и –½ для $C_{i+1} < 0$. Поэтому значение d_{i+1} зависит от значения n_{i+1} необработанной части числа N, рассматриваемой как число с точкой перед первым тритом.

То, что функция квадратного корня возрастающая, позволяет определить значение d_{i+1}. При $C_{i+1} > 0$ и $n_{i+1} > ¼$ невычисленная часть корня больше ½, т.е. d_{i+1} будет “+”. При $C_{i+1} < 0$ и $n_{i+1} <¼$ невычисленная часть корня меньше –½, т.е. d_{i+1} будет “-”.

Описание алгоритма

Целое троичное число также как и в работе [3] представляется в виде структуры

```
struct trin {int n; char trit [K];};
```

где `n` – число тритов, а массив `trit` – значение числа: младший трит хранится в элементе с индексом 0, а старшие – левее, элементы массива принимают значения –1, 0 и 1. Для числа нуль `n` равно нулю. Параметр алгоритма `K` – максимальная длина числа.

Вспомогательные функции

Результатом функции `sgn(w)` является знак числа `w`. Если `n` равно нулю, то результат будет нуль, в противном случае – значение элемента массива `trit` с индексом `n`-1.

```
int sgn(struct trin w)
{if (w.n == 0) return 0;
else return w.trit[w.n-1];
}
```

Функция `red(&res)` производит удаление старших тритов с нулевыми значениями структуры `res`, являющейся параметром функции. Она используется в функциях сложения `sum( )`, вычитания `res( )`, `shftb(&bs,x,i)` и в функции сложения трех слагаемых `sum2(x,y,&res)`.

```
void red(struct trin *res)
{int i;
for (i = res->n-1; i >= 0 && res->trit[i] == 0; i--);
res->n=i+1;
}
```

Функция `shftb(&bs,x,i)` определяет новое значение `bs`. Ненулевое значение `bs` сдвигается влево на два трита. Всегда тритам с индексами один и нуль присваивается значения тритов структуры `x` с индексами, соответственно, `i` и `i-1`, после чего производится удаление ведущих нулей.

```
void shiftb(struct trin *bs, struct trin x, int i)
{int k;
if (bs->n != 0)
for (k=bs->n; k>0; k--)
bs->trit[k+1]=bs->trit[k-1];
bs->trit[1]=x.trit[i];
bs->trit[0]=x.trit[i-1];
bs->n +=2;
red(bs);
}
```

Функция `shiftc(&cs)` сдвигается влево на один трита значение структуры `cs` и обнуляет трит с индексом нуль.

```
void shiftc(struct trin *cs, struct trin x, int i)
{int k;
if (cs->n != 0)
for (k=cs->n; k>=0; k--)
cs->trit[k]=cs->trit[k-1];
cs->trit[0]=0;
cs->n +=1;
}
```

Функция `acmp(x,y)` производит сравнение модуля `x` и положительного значения `y`. При равной длине `x` и `y` производится сравнение тритов чисел `x` и `y` с учетом знака числа `x` до первого несовпадения, которое и определяет результат. Если все триты совпали, то результат – нуль.

```
int acmp(struct trin x, struct trin y)
{int i=x.n,sx = sgn(x);
 if (i>y.n) return 1;
 if (i-- == y.n)
{while (sx*x.trit[i] == y.trit[i] && i-- >= 0);
if (i<0) return 0;
if (x.trit[i]*sx > y.trit[i]) return 1;
}
return -1;
}
```

Функция `cmp1(x,k)` производит сравнение необработанной части структуры `x` с индексами с `k` по нулевой (рассматриваемой как дробное число с точкой перед `k`-ым тритом) с числом 0.25. Число 0.25 представляется бесконечной последовательностью .+-(+-)… . Поэтому сравниваются пары тритов структуры `x` и пара `+-`.

```
int cmp1(struct trin x, int k)
{int i= k;
while (x.trit[i] == 1 && x.trit[i-1] == -1 && i>0)
i -= 2;
if (i>0 && x.trit[i]==1 && x.trit[i-1]>-1)
return 1;
else return -1;
}
```

Функция `sum2(x,y,&res)` производит вычисление по формуле `res=x-(y+y)sign(x)-1`. Практически это одновременное сложение трех слагаемых, заметим, что в этом случае перенос в старший трит, также как и при обычном сложении, может быть только одной единицы (положительной или отрицательной). Максимальное значение суммы тритов будет при совпадении знаков суммируемых тритов и трита переноса и равно 4 (“++” или “−−”), и это дает перенос единицы того же знака (“+” или “−”) в следующий трит. Отметим, что начальное значение трита переноса (переменная `c1`) равно -1, тогда как при обычном сложении трех слагаемых `c1` равно нулю.

Заметим, что трехвходовые сумматоры использовались в «Сетуни» [4] в множительном устройстве. Там для умножения 18–ти тритных чисел на первом уровне использовалось 6 трехвходовых сумматоров, на втором – 2 трехвходовых сумматора для суммирования результатов первого уровня, а на третьем уровне - один обычный (двухвходовый) сумматор для суммирования результатов второго уровня. Это позволило при последовательном выполнения операций выполнять умножения за 320 машинных тактов, притом, что сложение выполнялось за 180 тактов.

```
void sum2(struct trin x, struct trin y,
struct trin *res)
{int i, s, sx=sgn(x), c1 = -1, j=max(x.n, y.n);
res->n=j;
for (i = 0; i < j; i++)
 {s =c1;
if (i<x.n) s += x.trit[i];
if (i<y.n) s -= (y.trit[i]+y.trit[i])*sx;
if (s< -1) {res->trit[i] = s+3; c1 =-1;} else
if (s>1) {res->trit[i] = s-3; c1 =1;} else
{res->trit[i] = s; c1 =0;}

if (c1!=0) {res->trit[j] = c1; res->n=j+1;} else
red (res);
}
```

Алгоритм извлечения квадратного корня

Функция `sqrt(x,&y)` производит вычисление квадратного корня числа `x`.

```
void sqrt (struct trin x, struct trin *y)
{int i=x.n,j,sbs, k ,cmpbc;
struct trin bs,cs;
if (i == 0) y->n=0; else
if (x.trit[i-1]< 0) printf("Number less 0 \n");
else
{j= i >> 1;//j - индекс старшего трита корня
y->n= j+1;
if ((i&1) == 1 )
{// первая пара состоит из одного трита
bs.n=0; // bs=0
cs.n=1; cs.trit[0]=1; // cs=1
y->trit[j]=1; i -=2;}
else {// первая пара состоит из двух тритов
i--; // i - индекс старшего обрабатываемого трита x
if (cmp1(x,i)==1)
{// x больше ¼
bs.n=1; bs.trit[0]=-1; // bs=-1
cs.n=1; cs.trit[0]=1; // cs=1
y->trit[j]=1;}
else {y->n -= 1;
bs.n=0; // bs=0
cs.n=0; // cs=0}
}
j--;
```

```
for (i=i; i>=1; i -=2)
{ // определение j-го трита корня
shiftb(&bs,x,i);
shiftc(&cs);
sbs=sgn(bs);
cmpbc=acmp(bs,cs);
if (cmpbc ==1||cmpbc==0 && i>2 && cmp1(x,i-2)==sbs)
{ // j-й трит корня не нуль
 y->trit[j--]=sbs; sum2(bs,cs,&bs);
 if (sgn(cs)==0) cs.n=1;
 cs.trit[0]=sbs;
}
else y->trit[j--]=0 ;
} // for
} // else
} // конец функции
```

Литература

1. Рамиль Альварес Х. Троичный алгоритм извлечения квадратного корня // Программные системы и инструменты. Тематический сборник № 11. М.: Изд-во факультета ВМК МГУ, 2010. С. 98-106.
2. Березин Б.И., Березин С.Б. Начальный курс С и С++. – М.: ДИАЛОГ МИФИ, 2005.
3. Рамиль Альварес Х. Деление целых чисел в троичной симметричной системе // Программные системы и инструменты. Тематический сборник № 12. М.: Изд-во факультета ВМК МГУ, 2011. С. 228-234.
4. Брусенцов Н.П., Маслов С.П. и др. Малая цифровая вычислительная машина «Сетунь». – М.: Изд–во МГУ, 1965.

Рамиль Альварес Х.

Алгоритмы троичной арифметики

Компьютерная верстка А.Елыманов
Формат 60/90x1/16
Тираж 200 экз
Фонд «Новое тысячелетие»
www.newmillennium.ru
e-mail: newmillennium@mail.ru
тел. (495) 725-84-35
Отпечатано в ИПЦ «МАСКА», Москва, Научный проезд, д. 20, стр. 9
Тел./факс: (+7 495) 510-32-98. E-mail: info@maska.su