岡本安晴　2001.2.５;2001.8

５．多重精度算術演算[1]

Delphi における整数型は、８ビット長のもの（Shortint、Byte)、１６ビット長のもの（Shortint、Word)、３２ビット長のもの（Longint）がある。Delphi５ではさらに、３２ビット長のものとして Longword、６４ビット長のものとして Int64 が用意されている。これらの整数型より桁数の多い整数を扱うときは、必要な桁数に対応した型を用意する必要がある。用意した型のデータの処理・演算は、多重精度算術演算 (1)、(2) と呼ばれているアルゴリズムによって行うことができる。多重精度算術演算を、ここでは２進数表記に対して行う。まず、有限桁数の２進数表記の演算について説明する。

２進数表記の演算

正整数９は２進数で表わすと

$$9 = 1\times 2^3 + 0\times 2^2 + 0\times 2^1 + 1\times 0^0$$

となる。すなわち、

$$9_{10} = 1001_2$$

と表わせる。数値の右下に付けられている１０や２の数は、それぞれ１０進数表記、２進数表記であることを明記するためのものである。

整数３０は

$$30 = 1\times 2^4 + 1\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 0^0$$

と表わせる。すなわち、

$$30_{10} = 11110_2$$

と表わせる。

２進数４桁で０～１５の数が表わせる。２進数は４桁ずつ区切ることによって簡単に１６進数表記に変換できる。11110_2 を右から４桁ずつに区切ると、1と1110に分けられる（図１）。

---

[1] この解説は、TRY!PC,2000 年７月号「Delphi による数値計算プログラミングのすすめ：第５回多重精度算術演算・方程式の根の計算」の原稿をもとにしたものです。

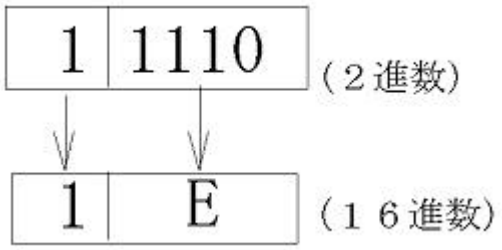

図１　２進数の１６進数への変換。E は 16 進数おいて１４を表わす。

1110 は１０進数で 8＋4＋2＝14 を表わすが、１４は１６進数では英字ｅまたはＥで表わす。したがって、

$$30_{10} = 11110_2 = 1E_{16}$$

と書ける。１６進数は、Delphi では＄を付けて＄1E と表わし、C++では先頭に 0x または 0X を付けて 0X1E と表わす。

コンピュータのメモリーは、８ビットを１バイトとして数える。１ビットは２進数の１桁に対応する。４ビットで２進数４桁、すなわち１６進数１桁、を表わすので、１バイトは１６進数で２桁を表わすことになる。

コンピュータにおいて、整数は、必要ならば左側に０を置くことにより、型によって決まった桁数（ビット数）の２進数で表わされる。Shortint 型の場合は 1 バイトすなわち８ビットで表わされるので、２進数で８桁までの数が表わされる。演算の結果が８桁を超えたときは、その超えた桁はメモリーに保持されずに消える[(3)]。

例えば、

$$(01000000_2)\times(00001000)_2 = 1000000000_2$$

の場合、右辺は８桁を超える数なので０になる（図 2）。

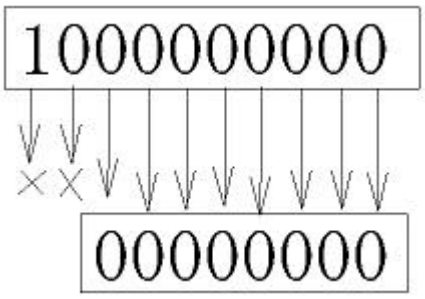

図２　Shortint は８ビットなので、演算結果が２進数で８桁を超えた部分は消える。

２進数で８桁を超えた部分が消えてしまうということは、2^8を法とする演算を行うことと同じである。すなわち、

$$(0100000)_2 \times (00001000)_2 \equiv 0 \mod 2^8$$

ということである。

2^8を法とする演算では、

$$(00000001)_2 + (11111111)_2 \equiv 0 \mod 2^8$$

となる。

2^8を法とする演算、すなわち Shortint 型での演算では、１に$(11111111)_2$を足すと０になる。したがって、$(11111111)_2$はー１と同じ働きをしている。Shortint 型では、最高位の桁が１である数は、それとの加算を行ったとき０となる数の負の値を表わすとみなす。２の負の数‐２は、

$$(00000010)_2 + (11111110)_2 = 100000000_2 \equiv 0 \mod 2^8$$

となるので、$(11111110)_2$が‐２を表わす。

コンピュータでは、数値は２進数で表わされ、演算は２のべき乗を法とする演算になっている。ｎビット（ｎ桁）の表記では、2^nを法とする演算になる。負の数は、最高位の桁が１のもので表わされる。

ｎ＝８の場合の2^nを法とする演算結果を確認するプロジェクトを PCkModular.dpr として用意した。このプロジェクトを実行すると、図３のようなフォームが表示される。

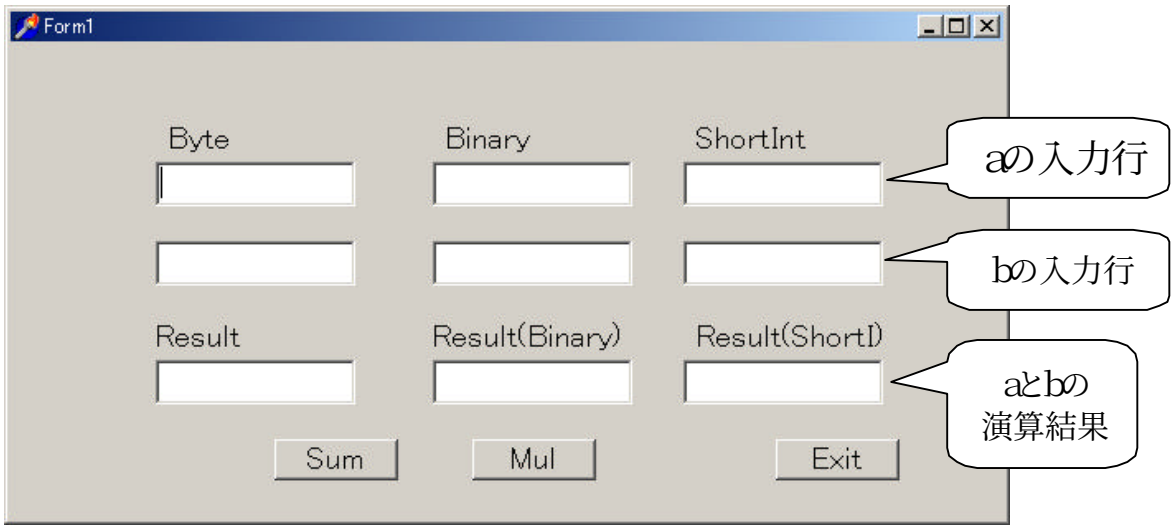

図３　PCkModular.dpr の実行開始時のフォーム

上の行と中の行に２つの整数値を設定する。設定は、Byte と ShortInt の２つの欄のいずれかで行う。Binary の表示の下の欄の Edit コンポーネントは表示専用である。Byte の下の欄は、８桁までの２進数を０～255 の範囲で設定する。この範囲を超えたり、不適当な文字が設定されると、範囲内の数値に適当に変換されたり、空白になったりする。Byte の欄に値を設定すると、そのときの値に対応した２進数表記が Binary の欄に表示される。ShortInt の欄には、８桁目を符号ビットとみた符号付き整数としての値が表示される。例えば、Byte 欄に 255 を設定すると、ShortInt 欄には－1 が表示される。逆に、ShortInt 欄に－1 を設定すると、Byte 欄には 255 が表示される。

上段と中段にそれぞれ数値を設定してから、「Sum」あるいは「Mul」ボタンをクリックすると、上段の数と中段の数の和、あるいは、積が算出されて下段に表示される。－１と－128 の和が図４に、積が図５に示されている。

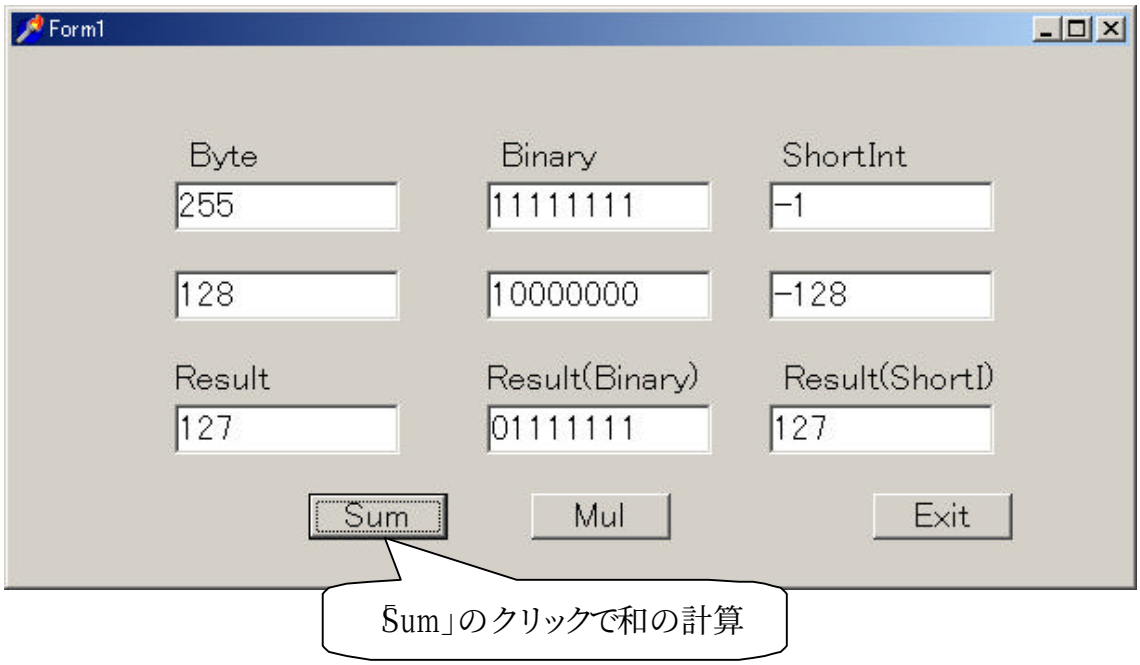

図４　2^8 を法とする和の計算

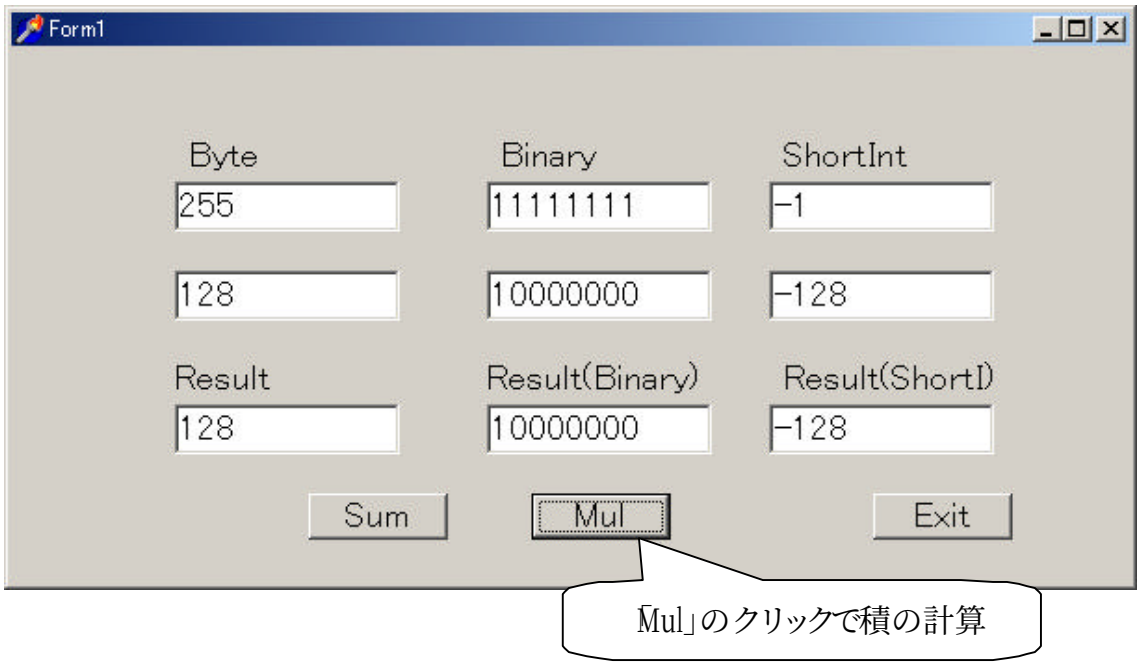

図５　2^8 を法とする積の計算

普通の１０進数での演算結果が８桁を超えるときは、このように普通の整数値の計算とは異なった感じの結果になることがある。しかし、普通の１０進数での演算結果が８桁を超えない場合、例えば－1 と－２の計算の場合は、図６a、図６ｂのように ShortInt の欄の値は普通の整数値の演算と同じ結果になっている。図６の場合、Byte 欄および Binary 欄に

おける演算結果は 2^8 を法とするものである。

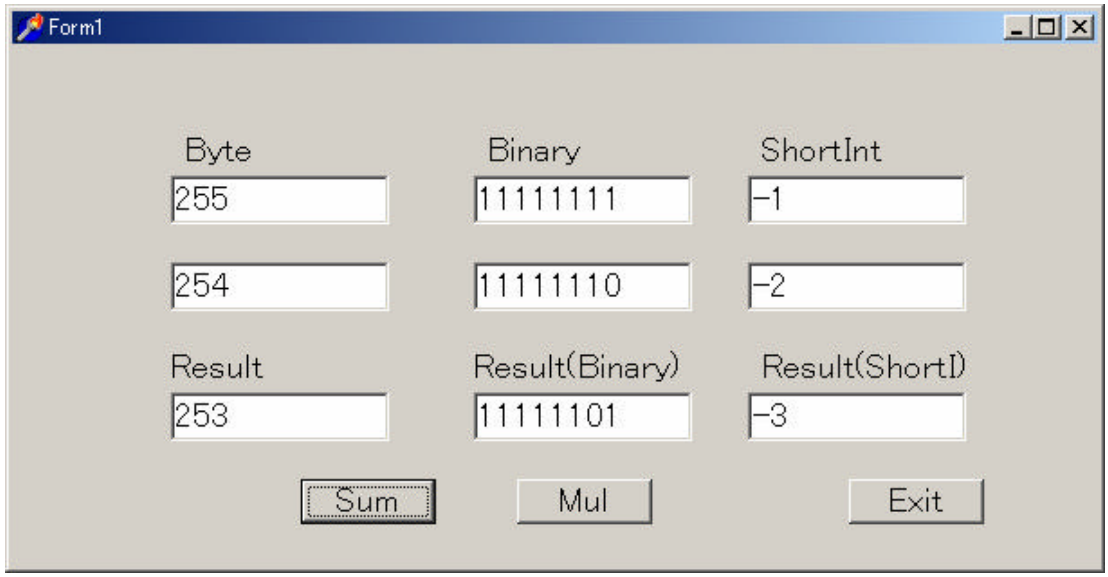

図６ａ　－１と－２の和

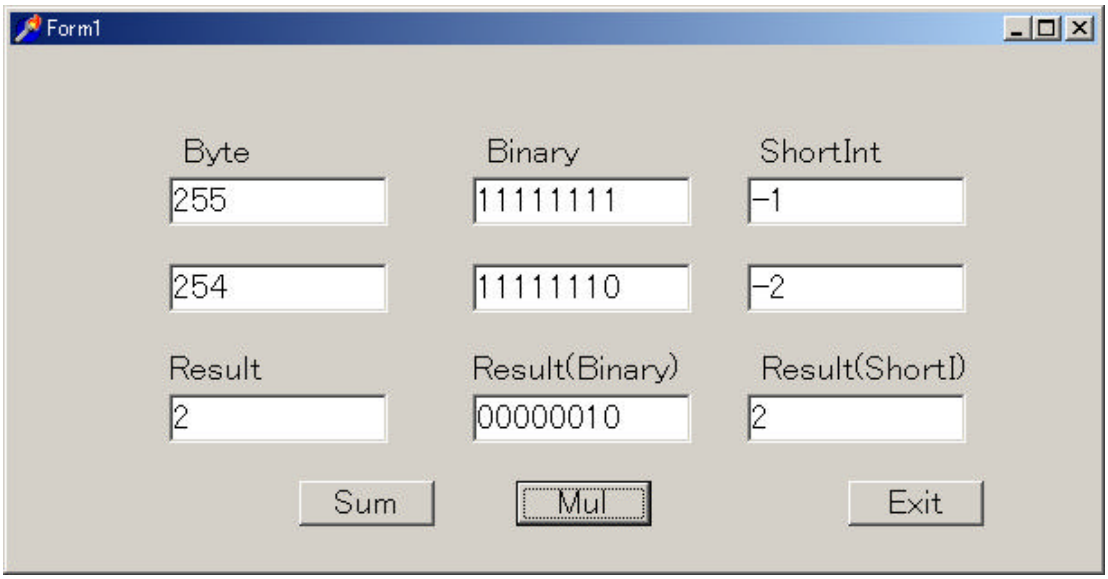

図６ｂ　－１と－２の積

Knuth(1998)［2］の表記法

正整数ｐの２進数表記

$$p = u_{n-1} \times 2^{n-1} + u_{n-2} \times 2^{n-2} + \cdots + u_1 \times 2^1 + u_0 \times 2^0$$

をここでは Knuth（1998）に従って

$$p=(u_{n-1}u_{n-2}\cdots u_1u_0)_2$$

と表わす。Knuth(1981)の訳（1986）[1]では、添字は$(u_1u_2\cdots u_n)_2$というように昇順であるが、ここではKnuth(1998)の方に従う。

正整数pがb進法で

$$p=v_{m-1}\times b^{m-1}+v_{m-2}\times b^{m-2}+\cdots+v_1\times b^1+v_0\times b^0$$

と表わされるときは

$$p=(v_{m-1}v_{m-2}\cdots v_1v_0)_b$$

と表わす。bは基数（baseまたはradix）という。

コンピュータでは、2進数４桁をまとめて、$2^4=16$を基数として整数を表わすことがある。ここでは、プログラミングが簡単になるように、２進数７桁をまとめて、$2^7=128$を基数とする。128 を基数とする１５桁の表記法は、２進数で$7\times15=105$桁の数が扱える。2^{105}は１０進数で３２桁の数なので、１２８を基数とする１５桁の表記法では、１０進数で３２桁ぐらいまでの整数が扱えることになる。

nビットの整数が１０進数による表記で何桁になるのかを求めるプロジェクトとしてPCheck.dprを用意した。プロジェクトPCheck.dprを実行すると、図７のようなフォームが表示される。

図７　ビット数から１０進数での桁数を求める

「ビット数」の右側のEditコンポーネント内にビット数を設定して「Calc」ボタンをクリックすると、「桁数」の右側に１０進数表記での桁数が表示される。

多重精度整数型 TBigInt

多重精度算術演算のための型 TBigInt をユニットファイル UBigInt.pas に用意した。TBigInt は、以下のように宣言されている。

```
type TBigInt = array[0..NBigInt-1] of byte;
```

TBigInt 型は、byte 型を要素とする配列である。byte 型の各要素の最上位のビット（MSB、Most Significant Bit）はキャリービットとして使い、１byte 中の下位７ビットで２進数の各桁を表わす。したがって、byte 型の各要素は、2^7 ＝128 進数表記での各桁に対応する（図８）。

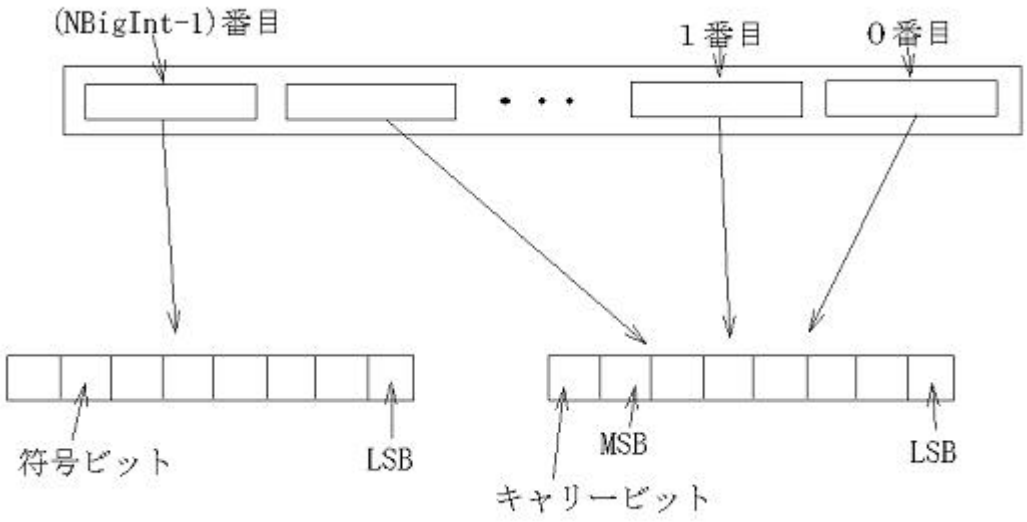

図８　TBigInt 型の構成

ユニットファイル UBigInt.pas では

```
const  NBigInt = 15;
```

と宣言されているので、7×15＝105 ビットの整数が扱える。105 ビットの中で最上位ビットは符号ビットとして使うので、-2^{104} から $2^{104}-1$ までの整数値を表わすことができる。

プロジェクト PTable.dpr は、TBigInt 型を用いて多重精度の整数値の和を求めるものである。整数値はグリッドのセルに設定する（図９）。

図９　和を求める整数値の設定

グリッドは、「追加」ボタンのクリックで行を追加することができる。データ設定用セルをクリックしてから「追加」ボタンをクリックすると、クリックしておいたセル（アクティブなセル）の下に行が追加される。不用なセルは、そのセルをクリックしてから「削除」ボタンをクリックすると削除される。全てのデータ設定用セルに数値を設定した後、「計算」ボタンをクリックすると、データの総和が下の Edit コンポーネントに表示される（図１０）。

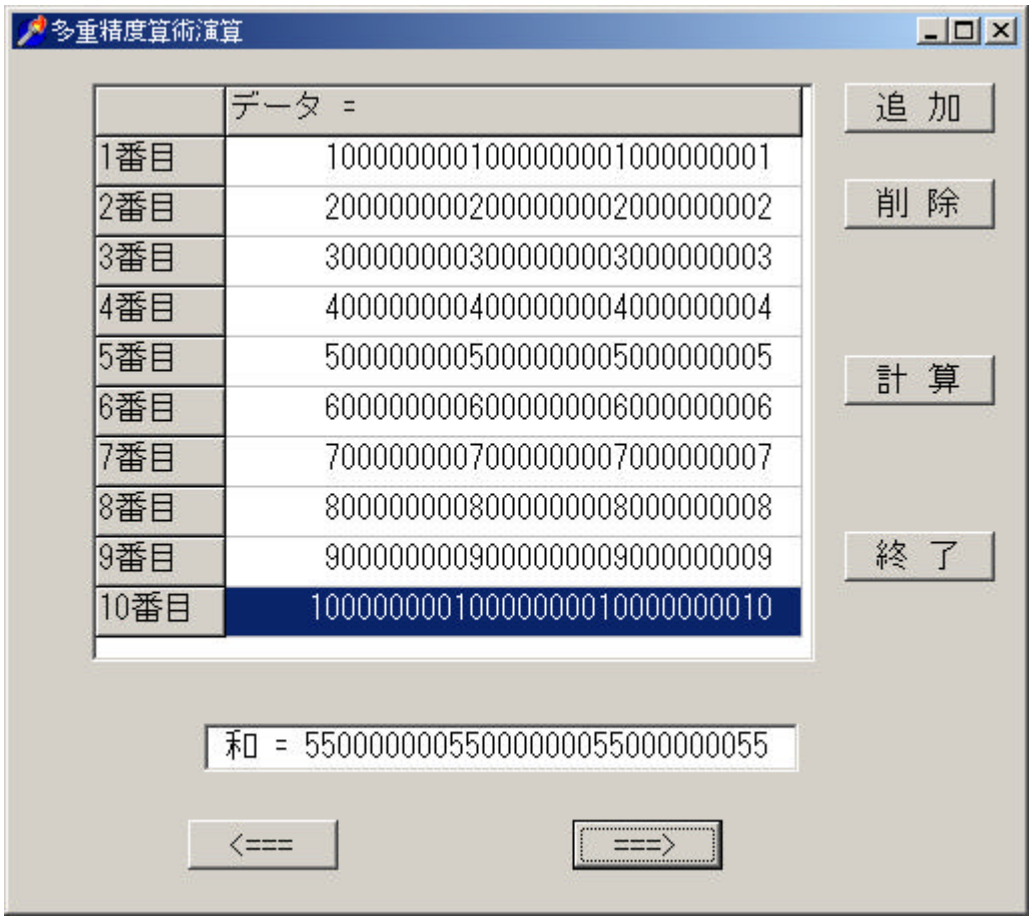

図 10　「計算」ボタンのクリックで和が求まる

「===>」ボタンをクリックすると、表示が右詰になる。左詰にするときは、「<===」ボタンをクリックする。データの訂正のときは、左詰の方がやり易い。

ユニット UBigInt

多重精度演算のためのユニットファイル UBigInt.pas には、表１の関数が用意されている。

表１　ユニット UBigInt.pas に用意されている関数

function BigAdd(a, b : TBigInt) : TBigInt;
ａとｂの和を関数値とする
function BigSub(a, b : TBigInt) : TBigInt;
ａとｂの差を関数値とする
function BigMul(a, b : TBigInt) : TBigInt;
ａとｂの積を関数値とする
function BigDiv(a, b : TBigInt) : TBigInt;
ａをｂで割った商を関数値とする
function BigMod(a, b : TBigINt) : TBigInt;
ａをｂで割った剰余を関数値とする
function StrToBig(s : string) : TBigInt;
文字列ｓの表わす整数値を TBigInt 型として返す
function BigToStr(a : TBigInt) : string;
ａの値を表わす整数値を（符号と）数字の文字列として返す。

function BigToFloat(a : TBigInt) : Extended;
aの表わす整数値を Extended 型として返す
function BigALB(a , b : TBigInt) : Boolean;
a＜bのときは True、それ以外のとき False を関数値とする
function BigALEB(a, b : TBigInt) : Boolean;
a≦bのときは True、それ以外のときは False を関数値とする
function BigAGB(a , b : TBigInt) : Boolean;
a＞bのときは True、それ以外のときは False を関数値とする
function BigAGEB(a, b : TBigInt) : Boolean;
a≧bのときは True、それ以外のときは False を関数値とする
function BigAEB(a, b : TBigInt) : Boolean;
a＝bのときは True、それ以外のときは False を関数値とする

これらの関数では、TBigInt 型の整数の加減乗除の演算を基礎として処理が行われている。これらの加減乗除の演算は、Knuth（1998）(2) のアルゴリズムに従って行われているが、基本的には紙の上で行う筆算と同じものである。以下の説明では、TBigInt 型に合わせて、基数b（base、radix）を $2^7 = 128$ とするb進数表記で整数pを表わす。

$$p = u_{n-1} \times b^{n-1} + \cdots + u_0 \times b^0 = (u_{n-1} \cdots u_0)_b = (u_{n-1} \cdots u_0)_{128}$$

型 TBigInt は

```
type TBigInt = array[0..NBigInt-1] of byte;
```

と宣言されていて、各byte型の要素で各u_iを表わす（図８）。

```
var  a : TBigInt;
```

のとき、a[NBigInt-1]が最上位桁u_{n-1},a[0]が最下位桁u_0を表わす。ただし、

```
n = NBigInt
```

である。

２数、$u=(u_{n-1}\cdots u_0)_{128}$と$v=(v_{n-1}\cdots v_0)_{128}$、の四則演算について説明する。

加算のアルゴリズム

$u=(u_{n-1}\cdots u_0)_{128}$と$v=(v_{n-1}\cdots v_0)_{128}$の加算を次のように行う（Knuth,1998, Algorithm A)。ここで、和を$w=(w_{n-1}\cdots w_0)_{128}$に求めるものとする（図１１）。

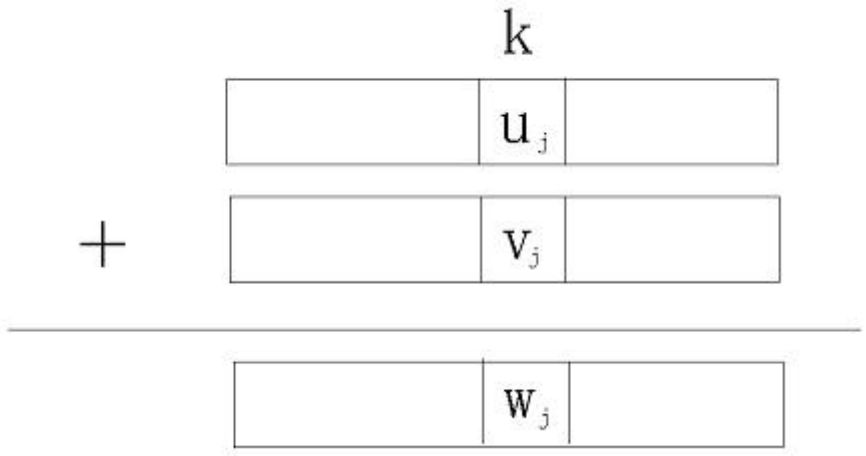

図１１　$u=(u_{n-1},\cdots,u_0)_{128}$と$v=(v_{n-1},\cdots,v_0)_{128}$の足し算。
kは桁上がりの値を表わす。

（A1）　$j\leftarrow 0$、$k\leftarrow 0$とおく。ここで、矢印←は代入を表わす。jは現在計算を行っている桁を表わし、kはキャリー（桁上がり）を表わす。これらは初期値として０とおく。

（A2）　$w_j=(u_j+v_j+k)\ \mathrm{mod}\ 128$、

$k\leftarrow(u_j+v_j+k)\ div\ 128$

とおく。

（A3）$j \leftarrow j+1$とおく。

$j < n$ならば、（A2）に戻る。

$j \geq n$ならば、このアルゴリズムを終了する。アルゴリズムの終了後、$w = (w_{n-1} \cdots w_0)_{128}$には$u$と$v$の和が格納されている。

関数 BigAdd は、上のアルゴリズムによって和を求めるものである。

減算のアルゴリズム

関数 BigSub では、ａとｂの差をａと－ｂの和として求めている。

負の数は、2^{105}を法とする演算においては、最上位のビットが１のもので表わされる。したがって、－ｂはｂを表わすビットを反転（１は０、０は１にする）してから１を加えることによって得られる。

ａと－ｂの和は、関数 BigAdd によって求めている。

乗算のアルゴリズム

２数、$u = (u_{n-1} \cdots u_0)_{128}$と$v = (v_{n-1} \cdots v_0)_{128}$、の積$w = (w_{2n-1} \cdots w_0)_{128}$は、以下の手順で求めている（Knuth, 1998, Algorithm M）。

（M1）まず、$j = n-1, \cdots, 0$に対して$w_j \leftarrow 0;$、とおく。

次に、$j \leftarrow 0$とおく。ｊは、vの128^jの位v_jによる掛け算であることを表わすものである。

（M2）$v_j = 0$ならば、$w_{j+n} \leftarrow 0$とおき、（M6）に跳ぶ。

（M3）$i \leftarrow 0$、$k \leftarrow 0$とおく。ここで、iはuの128^iの位u_iとの掛け算であることを表わし、ｋは桁上がりの数を表わすものである。これらは初期値として０とおく。

（M4）$t \leftarrow u_i \times v_j + w_{i+j} + k$とおく。ここで、ｔは例えば Word 型の変数として、桁溢れがないようにする。

$$w_{i+j} \leftarrow t \mod 128$$

$$k \leftarrow t \ div \ 128$$

とおく（図１２）。

（M5）$i \leftarrow i+1$とおく。

$i < n$ならば、（M4）に戻る。

$i \geq n$ならば、$w_{j+n} \leftarrow k$ とおく。

（M6）$j \leftarrow j+1$とおく。

$j < n$ならば、（M2）に戻る。

$j \geq n$ならば、このアルゴリズムを終了する。アルゴリズムの終了後、$w = (w_{2n-1} \cdots w_0)_{128}$には、$u$と$v$の積が格納されている。

関数 BigMul は、上のアルゴリズムによって積を求めるものである。

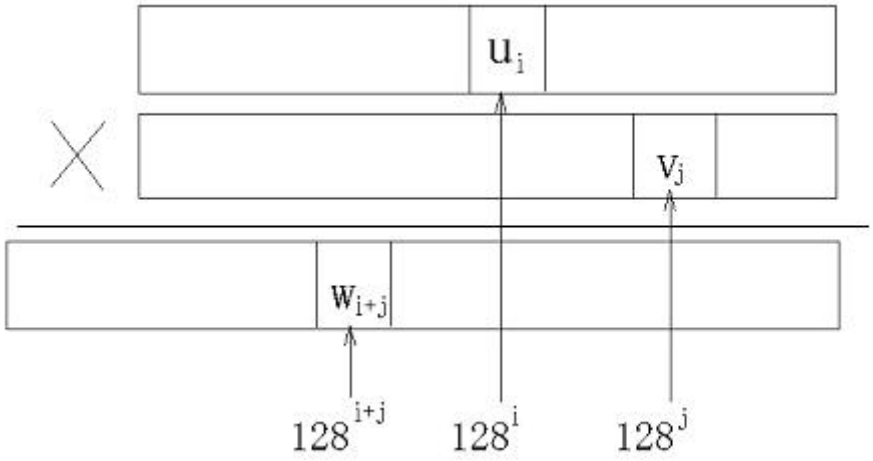

図１２　$u = (u_{n-1} \cdots u_0)_{128}$と$v = (v_{n-1} \cdots v_0)_{128}$の掛け算において、$u$の$128^i$の位$u_i$と$v$の$128^j$の位$v_j$の積は$w = u \times v$の$128^{i+j}$の位$w^{i+j}$に加えられる

除算のアルゴリズム

２数、$u = (u_{n-1} \cdots u_0)_{128}$と$v = (v_{n-1} \cdots v_0)_{128}$、の商$q = (q_{n-1} \cdots q_0)_{128}$および剰余$r = (r_{n-1} \cdots r_0)_{128}$の算出（Knuth、1998、Algorithm D）においては、まず、uおよびvが非負であるようにしてから商と剰余を求める。uあるいはvが負のときは、非負のときの商と剰余から負のときの商と剰余を求める。例えば、$u < 0,\ v > 0$のときは、

$$-u = vq + r$$

から

$$u = v(-q) + (-r)$$

として商$-q$と剰余$-r$を求める。

以下の説明では、uおよびvは非負であるとする。

$u<v$のときは、$u=v\times 0+u$となり、$q=0,\ r=u$となる。

$u\geq v$の場合について考える。

$v_j=0,\ j=n-1,\cdots,1$のときはv_0でu_iを順番に割っていけばよい。以下では、ある$j>0$に対して$v_j\neq 0$である場合について考える。

$v_j>0$である最大のjを$n-1$とおく。このとき、適当な整数mに対して、$u=(u_{m+n}\cdots u_0)_{128}$、$v=(v_{n-1}\cdots v_0)_{128}$と表わすことができる。

筆算による割り算は、上の桁から計算していく（図１３）。

$$\begin{array}{r} q_m\cdots q_0 \\ v_{n-1}\cdots v_0\overline{)\,u_{m+n}\cdots u_m\cdots u_0} \\ \ddots \end{array}$$

図１３　$(u_{m+n}\cdots u_0)_{128}\div(v_{n-1}\cdots v_0)$の筆算

最初の桁q_mの計算について考える。

$$(u_{m+n}\cdots u_m)_{128}\div(v_{n-1}\cdots v_0)_{128}<128 \qquad (1)$$

であるとする。この商が128以上のときは、被除数uの先頭に0を付け加えておく。すなわち、$u_{m+n+1}=0$、$m\leftarrow m+1$として、$(0u_{m+n}\cdots u_0)_{128}$を改めて$(u_{m+n}u_{m+n-1}\cdots u_0)_{128}$とおくことによって商が128より小さくなるようにする。いま、$\hat{q}$を次式によって定める。

$$\hat{q}=\min\{(u_{m+n}\times 128+u_{m+n-1})\ div\ v_{n-1},\ 128-1\} \qquad (2)$$

このとき、

$$q_m=(u_{m+n}\cdots u_m)_{128}\ \ div\ (v_{n-1}\cdots v_0)_{128}$$

とおくと

$$\hat{q}\geq q_m \qquad (3)$$

が成り立つ（Knuth,1998，p.271，Theorem A)。式（３）の証明は、附Aに示した。

さらに、

$$v_{n-1} \geq 128 \ div \ 2 = 64 \qquad (4)$$

であれば、

$$q_m \leq \hat{q} \leq q_m + 2 \qquad (5)$$

となる（Knuth,1998, p.271, Theorem B)。式（５）の証明は附Bに示した。

（５）式より、商q_mは$\hat{q}$、$\hat{q}-1$、$\hat{q}-2$の３つの値の１つに一致することがわかる。q_mを求めるときは、$\hat{q}$を商q_mの最初の推定値とする。この推定値が商として大き過ぎるときは適切な値になるまで１つずつ減らしていく。この減らす手続きは、式（５）より高々２回で済むことになる。商q_mの見当をつけるのに、被除数の上２桁と除数の上１桁の商$\hat{q}$で行うのが式（２）である。５４３２１÷９８の最初の桁を５４÷９によって見当をつけるということである。このとき、除数の上１桁の値が小さいと２桁目の値の影響を受け易いので、基数の1/2より大きくしておくというのが（４）式の条件である。

$$v_{n-1} < 64$$

のときは、次のように調整することによって（４）式が成り立つようにすることができる。

一般に、uをvで割ったときの商をq、剰余をrとおくと、

$$u = vq + r$$

が成り立つ。両辺にａをかけると、

$$(au) = (av)q + (ar)$$

が成り立つ。すなわち、除数と被除数を定数倍しても、商は変わらない。このことより、vが64より小さいときは、uとvを定数倍することにより（４）式（$v \geq 64$）が成り立つようにして、商を求めることができる。

上のことを踏まえて、$n>1$の場合の割り算を以下のように行う（Knuth,1998, Algorithm D)。

$u = (u_{m+n} \cdots u_0)_{128}$を$v = (v_{n-1} \cdots v_0)_{128}$で割ったときの商を$q = (q_m \cdots q_0)_{128}$、剰余を$r = (r_{n-1} \cdots r_0)_{128}$とおく。

（Ｄ１）上の式（５）の説明での条件「$v_{n-1} \geq 64$」が成り立つように、$u = (u_{m+n} \cdots u_0)_{128}$と$v = (v_{n-1} \cdots v_0)_{128}$の形を整えておく。$u$と$v$の

定数倍に用いた定数をdとおく。

（Ｄ２）$j \leftarrow m$とおく。ここで、jは商の128^jの位q_jの計算であること

を表わすものである。

（Ｄ３）$\hat{q} \leftarrow \{(u_{j+n} \times 128 + u_{j+n-1})\ div\ v_{n-1}\}$

$\hat{r} \leftarrow \{(u_{j+n} \times 128 + u_{j+n-1})\ \mathrm{mod}\ v_{n-1}\}$

とおく。

（Ｄ３ａ）$\hat{q} \geq 128$または$\hat{q}v_{n-2} > 128 \times \hat{r} + u_{j+n-2}$

ならば、

$\hat{q} \leftarrow \hat{q} - 1$、$\hat{r} \leftarrow \hat{r} + v_{n-1}$

とおく。

条件「$\hat{q} \geq 128$または$\hat{q}v_{n-2} > 128 \times \hat{r} + u_{j+n-2}$」が成り

立たないときは、（Ｄ４）に進む。

（Ｄ３ｂ）（Ｄ３ａ）に戻る。

（Ｄ４）$(u_{j+n} \cdots u_j)_{128} \leftarrow \{(u_{j+n} \cdots u_j)_{128} - \hat{q} \times (v_{n-1} \cdots v_0)_{128}\}$とおく。

（Ｄ５）$q_j \leftarrow \hat{q}$とおく。

$(u_{j+n} \cdots u_j)_{128} < 0$のときは、（Ｄ６）に進む。

$(u_{j+n} \cdots u_j)_{128} \geq 0$のときは、（Ｄ７）に跳ぶ。

（Ｄ６）$q_j \leftarrow q_j - 1$

$(u_{j+n} \cdots u_j)_{128} \leftarrow (u_{j+n} \cdots u_j)_{128} + (v_{n-1} \cdots v_0)_{128}$

とおく。

（Ｄ７）$j \leftarrow j - 1$とおく。

$j \geq 0$ならば、（Ｄ３）に戻る。

$j < 0$ならば、（Ｄ８）に進む。

（Ｄ８）$(q_m \cdots q_0)_{128}$に商が求まっている。

剰余は、（Ｄ１）での定数dで割ったもの、

$$(r_{n-1}\cdots r_0) \leftarrow \{(u_{n-1}\cdots u_0)\ \ div\ \ \ d\}$$

として得ることができる。

上のアルゴリズムによって商と剰余を求める手続きとして BigDivMod を用意した。商を求める関数 BigDiv、および剰余を求める関数 BigMod では、手続き BigDivMod を呼出している。

なお、上のアルゴリズムの（D3a）は、以下の性質に基づいている。

いま、

$$(u_{j+n}\cdots u_j)_{128} = q\times(v_{n-1}\cdots v_0)_{128} + r,\ \ 0 \le r < (v_{n-1}\cdots v_0)_{128}$$

$$u_{j+n}\times 128 + u_{j+n-1} = \hat{q}v_{n-1} + \hat{r}$$

とおく。ここで、$\hat{r} < 128$ の制約はない。

このとき、次の（A）および（B）が成り立つ。

（A）$\hat{q}v_{n-2} > 128\hat{r} + u_{j+n-2}$ ならば、$q < \hat{q}$　(Knuth,1998, p.282,Exercise 19)

（B）$\hat{q}v_{n-2} \le 128\hat{r} + u_{j+n-2}$ かつ $\hat{q} < 128$ ならば、$\hat{q} \le q+1$　(Knuth,1998,p.282,Exercise 20)

性質（A）は、$\hat{q}$ が q より大きくなる場合の条件を示している。なお、（３）式より $\hat{q}$ は q 以上の数である。

性質（B）は、$\hat{q}$ と q の差が高々１である場合の条件を示している。この性質（B）はステップ（D3）で用いている。

なお、Knuth(1998)の Algorithm D（p.272）におけるステップ D3 では、条件 $\hat{r} < 128$ のチェックが行われている。しかし、性質（B）が成り立つので、手続き BigDivMod では $\hat{r} < 128$ のチェックを省いている。

四則演算のチェック

プロジェクト PCalc.dpr は、上の四則演算の関数を用いて加減乗除の結果を表示するものである。実行時のフォームは図１４のようになっている。

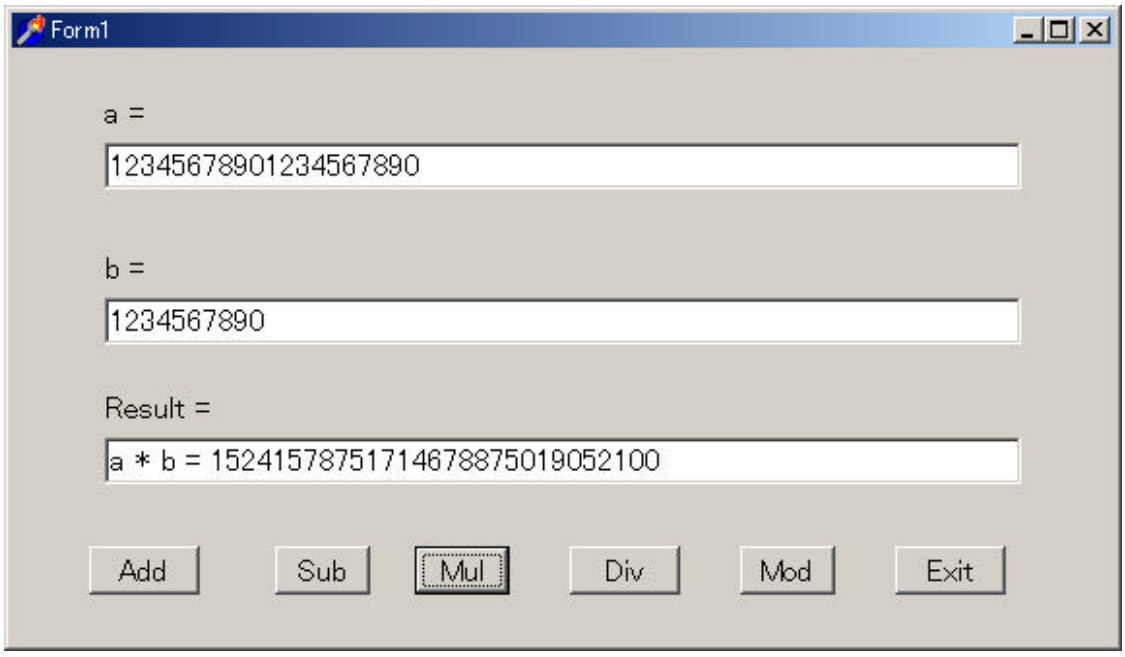

図１４　四則演算の実行例

それぞれ「ａ＝」と「ｂ＝」の下の Edit コンポーネントのところに整数値を設定して、「Add」、「Sub」、「Mul」、「Div」、「Mod」のボタンをクリックすると、ａとｂの和、差、積、商、剰余が「Result」の下の Edit コンポーネントに表示される。

附A. 式（3）の証明

条件（1）のもとでは

$$q_m \le 128-1 \qquad \text{(A.1)}$$

が成り立つ。

また、式（2）より

$$\hat{q} \le 128-1$$

が成り立つ。

$\hat{q}$の値の場合に分けて考える。

$\hat{q}=128-1$のときは、式（A.1）より次式

$$\hat{q} \ge q_m$$

すなわち（3）式が成り立つ。

その他の場合、すなわち

$$\hat{q} < 128-1 \qquad \text{(A.2)}$$

の場合について考える。

式（2）と（A.2）より

$$\hat{q} = \left[\frac{u_{m+n} \times 128 + u_{m+n-1}}{v_{n-1}} \right]$$

であることがわかる。ここで記号 [z] は、z を超えない最大の整数値を表わす。

したがって、

$$\hat{q} \le \frac{u_{m+n} \times 128 + u_{m+n-1}}{v_{n-1}} \le \hat{q} + 1 - \frac{1}{v_{n-1}}$$

が成り立つ。上式の右側の不等式より

$$\begin{aligned} u_{m+n} \times 128 + u_{m+n-1} &\le \hat{q} v_{n-1} + v_{n-1} - 1 \\ \hat{q} v_{n-1} &\ge u_{m+n} \times 128 + u_{m+n-1} - v_{n-1} + 1 \end{aligned} \qquad \text{(A.3)}$$

となる。いま、

$$u^{(m)} = (u_{m+n} \cdots u_m)_{128}$$

とおくと、

$$v = (v_{n-1} \cdots v_0)_{128}$$

なので、

$$u^{(m)} - \hat{q}v = u^{(m)} - \hat{q} \cdot (v_{n-1} \cdots v_0)_{128}$$

$$= u^{(m)} - \hat{q}(v_{n-1} \times 128^{n-1} + \cdots + v_0)$$

$$\le u^{(m)} - \hat{q}v_{n-1} \times 128^{n-1} \qquad \text{(A.4)}$$

となる。

式（A.3）と（A.4)、および$u^{(m)} = (u_{m+n} \cdots u_m)_{128}$に注意して、

$$u^{(m)} - \hat{q}v \le u^{(m)} - (u_{m+n} \times 128 + u_{m+n-1} - v_{n-1} + 1) \times 128^{n-1}$$

$$= u_{m+n} \times 128^n + u_{m+n-1} \times 128^{n-1} + u_{m+n-2} \times 128^{n-2} + \cdots + u_m$$
$$- (u_{m+n} \times 128^n + u \times 128^{n-1} - v_{n-1} \times 128^{n-1} + 128^{n-1})$$

$$= u_{m+n-2} \times 128^{n-2} + \cdots + u_m + v_{n-1} \times 128^{n-1} - 128^{n-1}$$

$$= u_{m+n-2} \times 128^{n-2} + \cdots + u_m - 128^{n-1} + v_{n-1} \times 128^{n-1} \qquad \text{(A.5)}$$

を得る。

128^{n-1}は 128 進数のｎ桁目の位なので、128 進数表記において n-1 桁までで表わされる数より大きい、すなわち

$$128^{n-1} > u_{m+n-2} \times 128^{n-2} + \cdots + u_m \qquad \text{(A.6)}$$

が成り立つ。

式（A.5）と（A.6）より

$$u^{(m)} - \hat{q}v < v_{n-1} \times 128^{n-1}$$

$$\le v_{n-1} \times 128^{n-1} + v_{n-2} \times 128^{n-2} + \cdots + v_0$$

$$= v$$

となる。すなわち、

$$u^{(m)} < \hat{q}v + v = (\hat{q} + 1)v$$

となる。上式は、$u^{(m)}$をvで割ったときの商q_mが$\hat{q}+1$より小さいこと

$$q_m < \hat{q}+1$$

を表わしている。q_mと$\hat{q}$は整数なので、上の不等式より

$$q_m \leq \hat{q}$$

が導ける。上式は（３）式と同じ内容である。

附B. 式（５）の証明

（３）式が成り立つので、（５）式が成り立つことを示すためには、（４）式の仮定

$$v_{n-1} \geq 64$$

のもとで、次式

$$\hat{q} \leq q_m + 2 \qquad \text{(B.1)}$$

が成り立つことを示せばよい。

背理法で証明する。

次式

$$\hat{q} \geq q_m + 3 \qquad \text{(B.2)}$$

が成り立つとする。

（２）式より

$$\begin{aligned}
\hat{q} &\leq \frac{u_{m+n} \times 128 + u_{m+n-1}}{v_{n-1}} \\
&= \frac{u_{m+n} \times 128^n + u_{m+n-1} \times 128^{n-1}}{v_{n-1} \times 128^{n-1}} \\
&\leq \frac{u_{m+n} \times 128^n + u_{m+n-1} \times 128^{n-1} + u_{m+n-2} \times 128^{n-2} + \cdots + u_m}{v_{n-1} \times 128^{n-1}} \\
&= \frac{u^{(m)}}{v_{n-1} \times 128^{n-1}} \qquad \text{(B.3)}
\end{aligned}$$

が導かれる。$u^{(m)}$は附.Aで定義された$u^{(m)} = (u_{m+n} \cdots u_m)_{128}$のことである。

また、

$$v = v_{n-1} \times 128^{n-1} + v_{n-2} \times 128^{n-2} + \cdots + v_0$$

$$< v_{n-1} \times 128^{n-1} + 128^{n-1}$$

すなわち、

$$v_{n-1} \times 128^{n-1} > v - 128^{n-1} \qquad \text{(B.４)}$$

が成り立つ。

式（B.3）と（B.4）より

$$\hat{q} < \frac{u^{(m)}}{v - 128^{n-1}} \tag{B.5}$$

となる。

また、

$$q_m = \left[\frac{u^{(m)}}{v} \right]$$

なので

$$q_m > \frac{u^{(m)}}{v} - 1 \tag{B.6}$$

となる。

式（B.2）と（B.6）より

$$3 \le \hat{q} - q_m < \hat{q} - \left(\frac{u^{(m)}}{v} - 1 \right)$$

となる。上式と式（B.5）より

$$\begin{aligned} 3 &< \frac{u^{(m)}}{v - 128^{n-1}} - \left(\frac{u^{(m)}}{v} - 1 \right) \\ &= \frac{u^{(m)} v - u^{(m)} (v - 128^{n-1})}{(v - 128^{n-1}) v} + 1 \\ &= \frac{u^{(m)}}{v} \cdot \frac{128^{n-1}}{v - 128^{n-1}} + 1 \end{aligned}$$

すなわち、

$$\begin{aligned} 3 - 1 &< \frac{u^{(m)}}{v} \cdot \frac{128^{n-1}}{v - 128^{n-1}} \\ \frac{u^{(m)}}{v} &> 2 \cdot \frac{v - 128^{n-1}}{128^{n-1}} \\ &= 2 \cdot \frac{v_{n-1} \times 128^{n-1} + v_{n-2} \times 128^{n-2} + \cdots + v_0 - 128^{n-1}}{128^{n-1}} \\ &\ge 2 \cdot \frac{v_{n-1} \times 128^{n-1} - 128^{n-1}}{128^{n-1}} \\ &= 2(v_{n-1} - 1) \end{aligned} \tag{B.7}$$

となる。

式（２）より

$$128-1\geq \hat{q}$$

が成り立つので、

$$\begin{aligned}128-4 &\geq \hat{q}-3\\ &\geq q_m \qquad \text{（式（B.2）より）}\\ &=\left[\frac{u^{(m)}}{v}\right]\\ &\geq [2(v_{n-1}-1)] \qquad \text{（式（B.7）より）}\\ &=2(v_{n-1}-1)\end{aligned}$$

となる。上式より

$$128-2\geq 2v_{n-1}$$

$$\frac{128}{2}-1\geq v_{n-1}$$

すなわち

$$v_{n-1}<\left[\frac{128}{2}\right]=64$$

となる。

式（B.2）の仮定のもとで上式が導かれたので、逆に

$$v_{n-1}\geq\left[\frac{128}{2}\right]=64$$

であれば

$$\hat{q}\leq q_m+2 \qquad \text{(B.1)}$$

が成り立つことになる。

参　考　文　献

（１）D.E.Knuth（著）・中川圭介（訳）「準数値算法／算術演算： The art of computer programming, 第４分冊」、Pp.536、サイエンス社、1986.

（２）D.E.Knuth.　Seminumerical algorithms:　The art of computer programming, Vol.2, third edition. Pp.762, Addison-Wesley, 1998.

（３）岡本安晴「Delphi プログラミング入門」、Pp.207,CQ 出版株式会社、1997.