RUSSIAN PEASANT MULTIPLICATION AND EGYPTIAN DIVISION IN ZECKENDORF ARITHMETIC

Garry J. Tee

Department of Mathematics, University of Auckland
University of Auckland, New Zealand

1 Introduction

Edouard Zeckendorf (1901–1983) was a Belgian amateur mathematician ([10], p.144). He invented [13] the representation of natural numbers as sums of distinct and non–consecutive Fibonacci numbers, and he shewed that each natural number has unique representation if F_2 is used to represent 1, rather than F_1 (which also equals 1). Each natural number can be represented as a Zeckendorf numeral which can be encoded as a stream of bits with index starting at 2, e.g. $27 = F_3+F_5+F_8$ can be encoded (with index increasing to the right) as 0101001 . In data transmission the most significant 1 can be followed by 1 (since 11 never occurs within a Zeckendorf numeral) to indicate the end of a number, so that 27 would get transmitted as the self-limiting bit-string 01010011 . This variable–length representation of natural numbers has been investigated as a potential method for compression of data in transmission [1][6].

In Zeckendorf arithmetic, addition by 1 is more complicated than in binary arithmetic, as was explained by Graham, Knuth & Patashnik ([8], pp. 282–283). Addition, subtraction and multiplication were discussed by Freitag & Phillips [7]; and Peter Fenwick has recently given the first treatment of all 4 basic arithmetic operations in Zeckendorf arithmetic [5], which he has tested with computer programs. Fenwick comments that "this arithmetic is unlikely to remain more than a curiosity" [5], since it is much more complex than standard binary arithmetic and the Zeckendorf numerals are bulkier than standard binary numerals.

Another set of algorithms for Zeckendorf arithmetic has been implemented by the author, as a set of procedures in THINK Pascal. These Pascal procedures, with a set of programs for testing them, are available from the author by e-mail request sent to tee@math.auckland.ac.nz .

2 Ordering Zeckendorf Numerals

To test $x = y$, just check whether all corresponding Zeckendorf coefficients in x and y are equal. To test $x > 0$, just check whether the bit–stream contains any 1.

To add 1 to a Zeckendorf numeral ([8], pp. 282–283), if its bit–stream starts with 0 then convert that to 1 (with $F_2 = 1$); but if it starts with 10 then convert that to 01 (with $F_3 = 2$). If the 1 which has been inserted is followed by 10, then

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX

that 110 must be standardized to 001 (by the 3–term recurrence relation for Fibonacci numbers); and that standardization must be repeated if necessary, i.e. any bit–sequence 11010 . . . 10100 (starting at any positive index) must be converted to 00000 . . . 00010 . The representation of any standard Zeckendorf numeral is unique. Thus, adding 1 to a k–digit Zeckendorf numeral can require $\mathrm{O}(k)$ elementary operations (of counting and Boolean operations). That is similar to adding 1 in k–bit binary arithmetic, where the carry can propagate through up to $k-1$ consecutive unit bits.

If a Zeckendorf numeral μ has its most significant 1 at index $i$then that is unchanged by adding 1, unless $\mu + 1 = F_{i+1}$, in which case the index increases by 1. Every natural number $\lambda > \mu$ can be produced from μ by repeated additions by 1, in each of which the highest index either remains constant or else increases by 1. Hence, the highest index for λ is greater than or equal to the highest index for μ. Therefore $\mu < \lambda$ if and only if, in their Zeckendorf representations with coefficients 0 or 1,

$$\lambda \;=\; \sum_{i\geq 2} Y_i F_i, \qquad \mu \;=\; \sum_{i\geq 2} Z_i F_i, \tag{1}$$

$Y_i = Z_i$ for all $i > h$ for some $h \geq 2$, but $Y_h = 1$ and $Z_h = 0$. Thus, a simple program can be written for a **Boolean function** less (x,y) which yields **true** if $x < y$ but otherwise yields **false**, doing a single downward scan through the bit–streams of Zeckendorf coefficients for x and y. From that function, other simple functions can be written to test $x > y$, $x \geq y$ and $x \leq y$:

greater:= less(y,x), ge:= **not** less(x,y), le:= ge(y,x).

3 Addition and Subtraction

3.1 Addition. In the Pascal procedure for addition, for each digit 1 at index i in y add F_i into x. When a digit 0 (at index i in x) gets 1 added into it, if that new 1 is followed by 10 then standardization must be applied from index i upwards; but otherwise, if the new bit–stream from index $i-1$ is 110, then standardization must be applied from index $i-1$ upwards. If 1 gets added to 1 at index i, then $2F_i$ must be replaced by $F_{i+1} + F_{i-2}$. To add in $F_{i+1} - 2F_i$, the bit–stream 010 from index $i-1$ must be converted to 001, and if that is followed by 1 then standardization must be applied from index $i+1$, costing $\mathrm{O}(k)$ elementary operations. And to add in F_{i-2}, then this process must be applied recursively, with $\mathrm{O}(k)$ stages. Thus, even increasing a single Zeckendorf coefficient by 1 costs $\mathrm{O}(k^2)$ elementary operations; and hence adding a pair of k–digit Zeckendorf numerals by the Pascal procedure costs $\mathrm{O}(k^3)$ elementary operations!

3.2 Subtraction. Zeckendorf subtraction by 1 is very simple. Consider $\mu = F_a + F_b + \cdots + F_z$, where the subscripts are in strictly increasing order. If $a = 2$ then just delete F_2. With $a > 2$ then $\mu - 1 = (F_a - 1) + F_b + \cdots + F_z$. From the 3–term recurrence formula,

$$\begin{aligned} F_a \;&=\; F_{a-2} + F_{a-1} \;=\; F_{a-4} + F_{a-3} + F_{a-1} \;=\; F_{a-6} + F_{a-5} + F_{a-3} + F_{a-1} \\ &=\; \ldots \;=\; \left\{ \begin{array}{l} F_2 + F_3 \text{ (for even } a) \\ F_1 + F_2 \text{ (for odd } a) \end{array} \right\} + \cdots + F_{a-7} + F_{a-5} + F_{a-3} + F_{a-1}\ ; \end{aligned} \tag{2}$$

and hence

$$F_a - 1 \;=\; \left\{ \begin{array}{c} F_3 + F_5 \text{ (for even } a) \\ F_2 + F_4 \text{ (for odd } a) \end{array} \right\} + \cdots + F_{a-7} + F_{a-5} + F_{a-3} + F_{a-1}\ . \qquad (3)$$

This produces $\mu - 1$ in standard Zeckendorf form, with no consecutive Fibonacci numbers.

Consider the subtraction of Zeckendorf numerals $x - y$, where $x \geq y$, with k digits in x. For each digit 1 at index i in y, we need to subtract F_i from x. When a digit 1 (at index i in the current value of x) gets 1 subtracted from it then that digit in x is replaced by 0 . But if the digit at index i of x is 0 and 1 is to be subtracted from it, then scan the bit-stream from index $i+1$ upwards, to find the 1 with least index $h > i$. As in (2),

$$F_h \;=\; F_{h-2j} + F_{h-2j+1} + F_{h-2j+3} + \cdots + F_{h-3} + F_{h-1}. \qquad (4)$$

Hence, when $h - i$ is even then

$$F_h - F_i \;=\; F_{i+1} + F_{i+3} + \cdots + F_{h-5} + F_{h-3} + F_{h-1}, \qquad (5)$$

but if $h - i$ is odd then

$$F_h - F_i \;=\; F_{i-1} + F_{i+2} + \cdots + F_{h-5} + F_{h-3} + F_{h-1}. \qquad (6)$$

Thus, in subtracting F_i from x, if $h - i$ is even then it can be done by (5) at the cost of $\mathrm{O}(k)$ elementary operations. But if $h - i$ is odd then it can be done by (6), which also adds in F_{i-1}, at a cost $\mathrm{O}(k^2)$.

Hence, subtracting a pair of k–digit Zeckendorf numerals by the Pascal procedure costs $\mathrm{O}(k^3)$ elementary operations.

4 Russian Peasant Multiplication

Before the Russian Revolution in 1917, some visitors to Russia were surprised to find that many illiterate Russian peasants were able to multiply quite large numbers mentally. The following algorithm (in Pascal) computes $z = xn$, where n is any non-negative integer and x is any number.

```
 if odd(n) then s:= x else s:= 0;    n:= n div 2;
while n>0 do
begin x:= x+x;   if odd(n) then s:= s + x;    n:= n div 2
end;    z:= sum .
```

This algorithm can be interpreted as generating successively the digits in the base–2 representation of n. After k executions of either if–statement ($k = 0, 1, 2, \ldots$) the current value of x is 2^k times its initial value, and the current value of n is the quotient when the initial value has been divided by 2^k. That current value of n is odd, if and only if 2^k has coefficient 1 in the binary expansion of the initial value of n; and hence the sum is to be increased by the current value of x if and only if the **Boolean** function odd(n) yields **true**. The number of additions into s is less than or equal to the number of doublings of x, which is $\lfloor \log_2 n \rfloor$.

This Russian Peasant Multiplication algorithm is independent of any written numerals, and it is independent of the base (if any) which might be used in the spoken numerals. The user needs only to be able to decide whether n is odd (and whether $n = 0$), to halve n (discarding remainder 1 when n is odd), and to add x into a number.

Less systematic versions of Russian Peasant Multiplication (without halving) were used in Ancient Egypt for multiplication and division ([4], pp. 14–20). In Ancient India, powers x^n were computed by an algorithm very similar to Russian Peasant Multiplication ([3], p.76), with addition being replaced by multiplication and the initial value 0 being replaced by 1. The RSA cryptographic algorithm requires efficient computation of $x^n \textbf{ mod } m$ for very large integers x, n, m. That has been done [11] by a version of the Ancient Indian method, using the operation of multiplication (**mod** m). More generally, the number x and the operator $+$ can be replaced by elements of any monoid with associative operator $\otimes$, with 0 being replaced by the neutral element for $\otimes$, to apply $\otimes$ ($n-1$ times) to x ([9], p.399). For example, x could be a polynomial whose coefficients are scalars or are square matrices (integer, real, complex or quaternion), and $\otimes$ could be the operation of multiplication of polynomials, to compute the nth power of the polynomial x.

If x and n are represented in binary notation, then Russian Peasant Multiplication is closely similar to the standard algorithm for multiplication in binary arithmetic. In binary arithmetic the doublings of x are done by shifting all binary digits up one place, and the binary digits of n are operated on directly.

4.1 Multiplication in Zeckendorf Arithmetic.

The existing algorithms for Zeckendorf multiplication [7] involve intricate operations upon the Zeckendorf coefficients. But Zeckendorf numerals can be multiplied more simply by Russian Peasant Multiplication, using the existing Zeckendorf algorithms (in [5], or the Pascal procedure) for addition of x. If the factor n is represented as a standard **integer** in Pascal (or in any similar programming language), then all operations on n can be done by standard **integer** operations.

4.1.1 Parity of Zeckendorf Numerals. In order to determine whether n is odd, consider the expansion

$$n \;=\; \sum_{i=1}^{k} Z_i F_i \qquad (Z_i = 0, 1), \tag{7}$$

for a suitable upper limit k, where $Z_1 = 0$ in a standard Zeckendorf numeral.

Since $F_0 = 0$ which is even, and both F_1 and F_2 equal 1 which is odd, then it follows by induction on the defining 3–term recurrence relation that F_i is even if and only if $i = 3h$ for some integer h. Therefore, replacing each coefficient Z_{3h} in (1) by 0 would not alter the parity of the sum of Fibonacci numbers. Thus the parity of n equals the parity of $\sum_{h\geq 0}(Z_{3h+1}F_{3h+1} + Z_{3h+2}F_{3h+2})$, where both F_{3h+1} and F_{3h+2} are odd. Operating **mod** 2, this becomes

$$n \;\equiv\; \sum_{h\geq 0}(Z_{3h+1} + Z_{3h+2}) \quad \textbf{mod } 2\ . \tag{8}$$

For each h, the term in parentheses will equal 1 if $Z_{3h+1} \neq Z_{3h+2}$, but otherwise that sum is even. Thus, the **Boolean** value of odd(n) can be computed as follows, with the Zeckendorf coefficients represented by a **Boolean** array z. If n is zero then odd:= **false**, but for positive n apply the algorithm:

```
odd:= false;    i:= 0;
repeat   if z[i+1]<>z[i+2] then odd:= not odd;    i:=i+3
until i>=k .
```

Determining the parity of a k–digit Zeckendorf numeral by this algorithm costs O(k) elementary operations.

4.1.2 Halving nonstandard Zeckendorf Numerals.

In order to halve n, it is convenient to work with nonstandard Zeckendorf numerals in which consecutive Fibonacci numbers can appear, so that the bit–stream can be any pattern of 0 and 1; and also F_1 is accepted, so that the index in the bit–stream starts with 1 rather than 2. The algorithm for evaluating odd(n) works for such nonstandard Zeckendorf numerals, since if $Z_{3h+1} = Z_{3h+2} = 1$ then $Z_{3h+1} + Z_{3h+2} = 2 \equiv 0 \textbf{ mod } 2$. During the operation of the algorithm for halving n, the coefficients Z_i can take the values 0, 1, 2 or 3; but at the end of the algorithm the coefficients of the new n are all 0 or 1, so that the new n can be represented by the **Boolean** array z. The index for nonstandard Zeckendorf starts at 1, since otherwise we could get $Z_2 = 2$ during the halving.

In order to halve n set j as the index of the highest 1 in n, and construct an array c of **integer** coefficients of Fibonacci numbers:

```
c[0]:= 0;      for i:= 1 to j do if z[i] then c[i]:= 1 else c[i]:= 0 .
```

Transform the coefficients c[i] so that each becomes 0 or 2, except that c[1] becomes 1 if n is odd. To do that transformation, for i from j down to 2, if c[i] is odd then F_i or $3F_i$ occurs in the sum represented by the current array c. In that case, replace F_i (once) by $F_{i-1}+F_{i-2}$, so that c[i] gets reduced to 0 or 2 and both c[i-1] and c[i-2] get increased by 1. Construct the coefficients for the new value of n by halving each coefficient c[i] (for i>1). Do not actually subtract 1 from c[i] and do not perform any arithmetic division. Rather, in terms of the **Boolean** array z:

```
for i:= k downto 2 do z[i]:= c[i]>1 .
```

Halving a k–digit Zeckendorf numeral by this algorithm costs O(k) elementary operations.

This halving algorithm operates on nonstandard Zeckendorf numerals, and so the cycle of halvings in Russian Peasant Multiplication can be performed in this manner, without needing to standardize the output.

In this manner, all of the operations in Russian Peasant Multiplication on the Zeckendorf numerals for x and n have been implemented in Pascal, in terms of addition of small integers and **Boolean** operations.

4.2 Standardizing nonstandard Zeckendorf Numerals. If the nonstandard output of the procedure for halving is to be operated on by other procedures, then it must be converted to standard Zeckendorf form. That standardization can be done by a Pascal procedure, which scans the bit–stream with index i decreasing to 1. Each time that 11 is encountered at indices i and $i+1$ then it must be standardized from index i upward, costing O(k) elementary operations, as in §3.1. After standardizing down to $i = 1$, if the bit-stream starts with 10 then that must be replaced by 01 (since $F_1 = F_2 = 1$), and if the next bit is 1 then one more standardization must be applied, from index 2 upwards. Thus, standardization costs O(k^2) elementary operations.

5 Ancient Egyptian Division

For non–negative integer numerator n and positive integer denominator d, the operation of integer division produces the unique quotient q and remainder r such that $n = qd + r$, with $q \geq 0$ and $0 \leq r < d$. The existing division algorithm [5] requires intricate operations upon the Zeckendorf coefficients. But it is simpler to use a systematic version of an ancient Egyptian method for division, as in Problem 25 of the Rhind Mathematical Papyrus ([4], p.16).

Construct (once only) an array p with $p_i = 2^i$: set i=0 and $p_i = 1$, and then repeat i:=i+1; p_i:= $p_{i-1} + p_{i-1}$ until p_i is large enough for your purposes. That array can then be used in any division of n by d by the following algorithm, which constructs (by repeated addition) an array t with $t_i = 2^i d$. The quotient q (initially set at 0) effectively gets each 1 in its binary representation inserted in decreasing place–value, from the array p.

```
r:= n ;   q:=0;   i:=0;   t[i]:= d;
while t[i]<r do begin i:= i+1;   t[i]:= t[i-1] + t[i-1];      { t[i] = $2^i$d }
                  end;                                      { t[i]≥r>t[i-1] }
repeat  while r<t[i] do i:= i-1;                        {Now, t[i+1]>r≥t[i] }
   q:= q+p[i];  r:= r-t[i]                 {Puts $q_i$=1, in its binary expansion }
until r<d  .
```

Then q is the quotient and r is the remainder.

If $n < d$ then $q = 0$ and $r = d$. But, for $n \geq d$, the number of subtractions from r equals the number of additions into q, which is less than or equal to the number of doublings of d; and that equals $\lceil \log_2 q \rceil$.

If the numbers are represented in binary notation, then the Egyptian algorithm is closely equivalent to the standard binary arithmetic algorithm for division. In binary arithmetic the doublings of t are done by shifting all digits up one place, and each digit 1 in q gets inserted directly, without needing any table of powers of 2.

5.1 Division in Zeckendorf Arithmetic. In Zeckendorf arithmetic, the additions and subtractions involving p, q, r and t can be done by the existing algorithms (in [5], or the Pascal procedures), and a simple program (3) can test for $r < d$. In this manner, the Ancient Egyptian method for division has been implemented in Pascal.

6 Signed Zeckendorf Arithmetic

For negative numbers in Zeckendorf form, Fenwick proposed a form of complementing, generalized from binary arithmetic with 2s-complement. But that representation of negative numbers proves to be rather cumbersome [5].

Accordingly, it is convenient to generalize the unsigned Zeckendorf numerals by using sign and magnitude representation. Extend the bit–stream for z to start at index 0, with z[0] = **false** for $z \geq 0$ but **true** for $z < 0$, so that signed Zeckendorf zero has all elements **false**, from index 0 up. Nonstandard Zeckendorf numerals use z[1] for F_1, but z[1] is not used in unsigned standard Zeckendorf numerals. If z does have a sign at index 0, the bit–stream from index 2 up gives $|z|$ in unsigned Zeckendorf form.

To add signed Zeckendorf numerals $a + b = c$, if a and b have the same signs set $c = |a| + |b|$, then give c the sign of a. If a and b have different signs; then if $|a| \geq |b|$ set $c = |a| - |b|$ and then give c the sign of a, but otherwise set $c = |b| - |a|$

and then give c the sign of b. To subtract signed Zeckendorf numerals $c = a - b$, if $b = 0$ then $c = a$, but otherwise reverse the sign of b and then add $a + (-b)$.

To multiply signed Zeckendorf numerals $ab = c$, set $c = |a| \times |b|$, and then fix the sign by c[0]:= a[0]<>b[0]. Then, if unsigned $c = 0$, set the sign as non–negative: c[0]:= **false**. Signed division could be handled in a similar manner.

7 Complexity of Zeckendorf Arithmetic

For numbers written in any integer base $\beta > 1$, addition or subtraction of a pair of m–digit numbers costs O(m) elementary operations on digits $0, 1, \dots, \beta - 1$, and the standard algorithms for multiplication and for division cost $O(m^2)$ elementary operations.

7.1 Daniel Bernoulli's formula for Fibonacci numbers. Daniel Bernoulli the 1st (1700–1782) published in 1732 an important paper in which (amongst much else) he published ([2], p.52) the explicit formula for F_x, as:

$$\left[\left(\frac{1+\sqrt{5}}{2}\right)^x - \left(\frac{1-\sqrt{5}}{2}\right)^x\right] : \sqrt{5}\ . \tag{9}$$

Daniel Bernoulli's formula for the Fibonacci numbers has often been mis–named (e.g. [7] and [8], p.285) after Binet, who published it 111 years after Daniel Bernoulli did [12].

In terms of the golden ratio $\gamma = (1+\sqrt{5})/2 \ = \ 1{\cdot}6180340$, Daniel Bernoulli's formula can be rewritten as

$$F_i \ = \ \left(\gamma^i - (1-\gamma)^i\right)/\sqrt{5}. \tag{10}$$

7.1.1 Golden Numbers. Denote the Zeckendorf numeral for z as

$$z \ = \ \sum_{i=2}^{m} Z_i F_i\ . \tag{11}$$

Then

$$z \ = \ \frac{1}{\sqrt{5}}\sum_{i=2}^{m} Z_i\left(\gamma^i - (1-\gamma)^i\right) \ = \ \frac{1}{\sqrt{5}}\sum_{i=2}^{m} Z_i\gamma^i - \frac{1}{\sqrt{5}}\sum_{i=2}^{m} Z_i(1-\gamma)^i \ = \ \frac{\Gamma(z)}{\sqrt{5}} - \rho(z). \tag{12}$$

Here, $\Gamma(z)$ is the base–γ number whose coefficients equal the corresponding Zeckendorf coefficients of z, and

$$\rho(z) \ = \ \frac{1}{\sqrt{5}}\sum_{i=2}^{m} Z_i(1-\gamma)^i. \tag{13}$$

The real number $\Gamma(z)/\sqrt{5}$ could be called the *Golden Number* for the positive integer z.

For all m–digit Zeckendorf numerals, the maximum modulus of $\rho(z)$ is given by $z = F_{m+1} - 1$, whose Zeckendorf coefficients are given by (3). If m is even then

$$\begin{aligned}\rho\left(F_{m+1}-1\right) \ &= \ \left[(1-\gamma)^2 + (1-\gamma)^4 + \cdots + (1-\gamma)^m\right]/\sqrt{5} \\ &< \ \frac{(1-\gamma)^2}{\sqrt{5}\left(1-(1-\gamma)^2\right)} \ = \ \frac{1-2\gamma+\gamma^2}{\sqrt{5}(2\gamma-\gamma^2)}\ .\end{aligned} \tag{14}$$

Since $\gamma^2 = \gamma + 1$, this inequality reduces to

$$\rho(z) < \frac{2-\gamma}{\sqrt{5}(\gamma-1)} = \frac{2\gamma-\gamma^2}{\sqrt{5}(\gamma^2-\gamma)} = \frac{\gamma-1}{\sqrt{5}} = \frac{\sqrt{5}(\sqrt{5}-1)}{10} = \frac{5-\sqrt{5}}{10} < 0{\cdot}3 \,. \tag{15}$$

Similarly, if m is odd then

$$\rho\left(F_{m+1}-1\right) = \left[(1-\gamma)^3 + (1-\gamma)^5 + \cdots + (1-\gamma)^m\right]/\sqrt{5}, \tag{16}$$

so that

$$-\rho(z) < \frac{(\gamma-1)(1-\gamma)^2}{\sqrt{5}\,(1-(1-\gamma)^2)} = \frac{(\gamma-1)\left(5-\sqrt{5}\right)}{10} < 0{\cdot}2 \,. \tag{17}$$

Hence, every natural number z equals the Golden Number $\Gamma(z)/\sqrt{5}$, rounded to the nearest integer.

7.2 Comparison of Zeckendorf Arithmetic with binary arithmetic. For numbers written in any integer base $\beta > 1$, the largest m–digit number is $\beta^m - 1$. For numbers written in some other base δ, $\delta^p = \beta^m$, where $p = m(\log\beta/\log\delta)$. Hence, conversion of an m–digit integer in base β to base δ requires $\mathrm{O}(m)$ digits in base δ. Hence, the Zeckendorf representation of an m–digit number in base β requires approximately $m(\log\beta/\log\gamma)$ digits in Zeckendorf representation. For example, an m–bit binary number requires approximately $\lceil 1{\cdot}46m\rceil + 2$ digits in Zeckendorf representation.

Conversion of a k–digit Zeckendorf numeral to standard binary form (or conversely) costs $\mathrm{O}(k)$ additions or subtractions of binary numbers with $\mathrm{O}(k)$ bits, or $\mathrm{O}(k^2)$ elementary operations. Hence, if a pair of Zeckendorf numerals were converted to binary form, then addition or subtraction performed in binary arithmetic would cost $\mathrm{O}(k)$ elementary operations, and multiplication or division would cost $\mathrm{O}(k^2)$. If the result of the binary arithmetic were converted to Zeckendorf form, then the cost for each basic arithmetic operation on Zeckendorf numerals (via binary arithmetic) would be $\mathrm{O}(k^2)$ elementary operations. But the Pascal procedures for addition and subtraction each cost $\mathrm{O}(k^3)$ elementary operations. In Russian Peasant Multiplication the number of additions is $\mathrm{O}(k)$, and in Egyptian division the number of additions or subtractions is $\mathrm{O}(k)$; and hence multiplication or division of k–digit Zeckendorf numerals costs $\mathrm{O}(k^4)$ elementary operations.

However, arithmetic operations on large binary integers (beyond the domain of arithmetic hardware) usually are done by software, which multiplies by a large factor the time required for elementary operations on the digits. Consequently, the actual ratio of cost of Zeckendorf addition or subtraction to cost of binary operations might be much smaller than this simple counting of elementary operations on digits might suggest.

If algorithms for addition or subtraction of Zeckendorf numerals could be devised with cost of lower order (e.g. $\mathrm{O}(k \log k)$ elementary operations) then there would be no need to consider conversion to and from binary numerals for doing arithmetic on Zeckendorf numerals.

I wish to thank Peter Fenwick, for introducing me to the concept of Zeckendorf arithmetic.

References

1. A. Apostolico A. & A. S. Fraenkel, *Robust transmission of unbounded strings using Fibonacci representations*, IEEE Trans. on Inf. Th. **IT–33** (1987)), 238-245, (cited from [5]).
2. Daniel Bernoulli, *Observationes de seriebus quae formantur ex additione vel subtraction quacunque terminorum se mutuo consequentium, ubi praesertim earundem insignis usus pro inveniendis radicum omnium aequationum algebraicarum ostenditur*, Commentarii Academiae Scientiarum Imperialis Petropolitanae t.3 (1732 for 1728), 85–100, Reprinted in: *Die Werke von Daniel Bernoulli* Band 2 (edited by David Spieser), Birkhäuser Verlag, Basel–Boston–Stuttgart (1982), pp. 49–64.
3. Bibhutibhusan Datta & A. N. Singh, *History of Hindu Mathematics*, vol. 1, Motilal Banarsidass, Lahore, 1935.
4. John Fauvel & Jeremy Gray, *The History of Mathematics - a Reader*, Macmillan Press, London, 1987.
5. Peter Fenwick, *Zeckendorf integer arithmetic*, The Fibonacci Quarterly (to appear).
6. A. S. Fraenkel & S. T. Klein, *Robust universal complete codes for transmission and compression*, Discrete Applied Mathematics **64** (1996), 31–55, (cited from [5]).
7. H. T. Freitag & G. M. Phillips, *On the Zeckendorf form of* F_{kn}/F_n, The Fibonacci Quarterly **34,** No.5 (1996), 444–446.
8. Ronald L. Graham, Donald E. Knuth & Oren Patashnik, *Concrete Mathematics*, Addison–Wesley, Reading MA, 1989.
9. Donald E. Knuth, *The Art of Computer Programming*, vol. 2, Addison–Wesley Publishing, Reading, 1969.
10. George M. Phillips, *Two Millenia of Mathematics, from Archimedes to Gauss*, Springer–Verlag, New York, 2000.
11. Garry J. Tee, *The perfect cryptographic method?*, in: *Directions For the Future: Communication. Papers given during the 49th ANZAAS Congress, University of Auckland January 1979* (ed. by Mari Davis), Trans Knowledge Associates, Melbourne, 1979, pp. 24–28.
12. Garry J. Tee, *Integer sums of recurring series*, New Zealand Journal of Mathematics **22** (1993), 85–100.
13. E. Zeckendorf, *Représentation des nombres naturels par une somme de nombres de Fibonacci ou les nombres de Lucas*, Bull. Soc. Roy. Sci. Liège **41** (1972), 179–182.

2000 *Mathematics Subject Classification* 11B39, 03H15.

2002–4–24

Department of Mathematics, University of Auckland,
Private Bag 92019, Auckland, New Zealand
E-mail address: **tee@math.auckland.ac.nz**