多重精度整数の10進法による表示アルゴリズム†

高橋 俊成††

計算機は，数値の内部表現に2進法を用いるのが普通であるが，人間にとっては10進法の方が都合の良いことが多く，出力の際には2進数を10進数に変換することが必要となる．近年では無限精度の整数をも扱うことのできるプログラミング言語が一般的になったため，基数変換の速度が重要さを増してきた．本論文では，汎用プロセッサにおいて多重精度の2進数を10進数に変換する高速アルゴリズムを，使用するプロセッサの機能に応じて，いくつか紹介する．これらのアルゴリズムの多くは，ある特殊な性質を持つ数値を巧みに利用したものであるため，他の基数法間での変換に応用することは困難であるが，逆に2進10進変換の方法としては優れたものであると言える．

1. はじめに

計算機が数値を扱う場合，内部的には通常2進数，外部的には10進数を使うので，入出力には基数変換が伴う．近年では無限精度の整数を扱うプログラミング言語等が一般的になったため，基数変換の速度が重要さを増してきた．本論文では筆者の工夫した，十分に精度の高い多重精度整数の2進10進変換アルゴリズムのいくつかを報告する．

多重精度整数の2進10進変換の最も簡単な方法の1つは，計算機の除算命令で割ることのできる最大の10のべき乗で繰り返し割ることである[1]．ところがこの方法は効率的とは言い難い．汎用のCPUでは除算よりもビット・シフトの方が高速にできるから，10が2と5の積であることを利用して，計算機の命令で一度に割ることのできる最大の5のべき乗で繰り返し割るのが良い，というのが着想である．

今日では，定数での乗除算をビット・シフト演算に置き換えて高速化する手法は，コンパイラ技術として定着している．にもかかわらず，2進10進変換に同様の手法を用いた例はほとんど見当たらない*．その理由のひとつは，レジスタ内容をビット・シフトする場合とは異なり，メモリに格納された数のビット・シフトは容易でないことにある．一方，実際にメモリ内でシフトを行わず，仮想的にシフトを行う方法も考えられるが，

- ビット・フィールド演算のような高機能な命令がないとコーディングが困難である，
- 比較的小さな多重精度整数の場合には効果が期待できない，
- ソース・コードが大きくなり，例えば言語処理系の1ルーチンとしては適当でない，

などの問題点がある．

本論文では，多重精度の2進整数が符号なしの形でメモリ内に連続的に格納されていて，そのアクセスの最小単位は1バイト（8桁）であるという最も典型的な仮定をし，各種CPU上で高速に10進変換するアルゴリズムを報告したい．

なお，以下本論文においては"除算"とは商と剰余を同時に求めることを意味し，正の整数 X, Y に対し

$$\text{quo} \leftarrow \lfloor (X/Y) \rfloor,\ \text{rem} \leftarrow X \bmod Y$$

の演算を

$$(\text{quo, rem}) \leftarrow X \div Y$$

と略記する．

以下，第2章ではアルゴリズム実現の基本となる定理を紹介し，第3章では使用するCPUの（主に除算の）機能に応じたアルゴリズムをそれぞれ示し，第4章で各アルゴリズムの評価を行う．

2. 算法

この章では本論文におけるアルゴリズムの実現に必要な定理を2つ示す．

定理 1：

$0 \leqq a, b, c < D$, $0 < e \leqq d < D$, $aD+b < dD+e$ なる整数 a, b, c, d, e, D について

$$(q, r) \leftarrow (aD^2+bD+c) \div (dD+e),$$

$$(f, g) \leftarrow (aD+b) \div d,\ R = gD+c-ef$$

とすると

$R \geqq 0$ のとき $q=f$, $r=R$

---

† Algorithms to Indicate Multiple-Precision Integer with Decimal Notation by TOSHINARI TAKAHASHI (Department of Mathematical Engineering and Instrumentation Physics, Faculty of Engineering, University of Tokyo).

†† 東京大学工学部計数工学科

* 例えば筆者の検証した Franz Lisp および KCL では，それぞれ 10^9 および 10 で繰り返し割るという古典的な方法を用いている．

$R<0$ のとき $q=f-1,\ r=R+dD+e$

である．

また特に $a=d$ の場合は

$q=D-1,\ r=(b+d-e)D+c+e$

である．

証明）

$$\begin{aligned}&(aD^2+bD+c)-(f+1)(dD+e)\\&=(aD^2+bD+c)-(aD+b-g)D-ef-dD-e\\&=(g-d)D+c-ef-e\\&\leqq -D+c-ef-e \quad \because\ g<d\\&<0 \quad \because\ c<D.\end{aligned}$$

したがって $(aD^2+bD+c)/(dD+e)<f+1$.

$$\begin{aligned}&(aD^2+bD+c)-(f-1)(dD+e)\\&=(aD^2+bD+c)-(aD+b-g)D-ef+dD+e\\&=(g+d)D-ef+c+e\\&>0 \quad \because\ e\leqq d,\ f<D.\end{aligned}$$

したがって $(aD^2+bD+c)/(dD+e)>f-1$.

両者より $f-1\leqq q\leqq f$.

また，

$$\begin{aligned}&(aD^2+bD+c)-f(dD+e)\\&=(aD^2+bD+c)-(aD+b-g)D-ef\\&=gD+c-ef\\&=R\end{aligned}$$

であるから以降は明らかである．

また $a=d$ の場合は

$$\begin{aligned}&(aD^2+bD+c)-(dD+e)D\\&=((aD+b)-(dD+e))D+c\\&\leqq -D+c \quad \because\ aD+b<dD+e\\&<0 \quad \because\ c<D.\end{aligned}$$

したがって $(aD^2+bD+c)/(dD+e)<D$.

$$\begin{aligned}&(aD^2+bD+c)-(dD+e)(D-1)\\&=(b+d-e)D+c+e\\&>0 \quad \because\ 0<e\leqq d.\end{aligned}$$

したがって $(aD^2+bD+c)/(dD+e)>D-1$.

両者より $q=D-1$.

$$\begin{aligned}r&=(aD^2+bD+c)-(D-1)(dD+e)\\&=(b+d-e)D+c+e \quad \because\ a=d.\end{aligned}$$

定理 2：

$0\leqq x<2^{i+j}\cdot y,\ 0<2^j\cdot y<2^i,$

$2\leqq i,\ 1\leqq j,\ 2\leqq k\leqq j+1$

なる整数 x, y, i, j, k について

$(q,\ r) \leftarrow x\div 2^j\cdot y,$

$(a,\ b) \leftarrow x\div 2^k,$

$(c,\ d) \leftarrow a\div 2^{j-k+1}\cdot y,$

$R=2^k\cdot d+b$

とすると

$R<2^j\cdot y$ のとき $r=R,\ q=2c$

$R\geqq 2^j\cdot y$ のとき $r=R-2^j\cdot y,\ q=2c+1$

である．

また，$(c,\ d) \leftarrow a\div 2^{j-k+1}\cdot y$ の演算は，$2i$ ビット $\div i$ ビットの符号付き除算１命令で実行できる．すなわち，

$0\leqq a<2^{2i-1}$

$0<2^{j-k+1}\cdot y<2^{i-1}$

$0\leqq c<2^{i-1}$

$0\leqq d<2^{i-1}$

である．

証明）

$x=2^k\cdot a+b,\ a=2^{j-k+1}\cdot yc+d$ だから

$$\begin{aligned}x-2c\cdot 2^j\cdot y&=2^k(2^{j-k+1}\cdot yc+d)+b-2^{j+1}\cdot cy\\&=2^k\cdot d+b\\&\geqq 0.\end{aligned}$$

よって $R<2^j\cdot y$ のとき $r=R,\ q=2c$.

また，
$$\begin{aligned}R-2^j\cdot y&=2^k\cdot d+b-2^j\cdot y\\&\leqq 2^k(2^{j-k+1}\cdot y-1)+(2^k-1)-2^j\cdot y\\&=2^j\cdot y-1.\end{aligned}$$

したがって $R\geqq 2^j\cdot y$ のとき $r=R-2^j\cdot y,\ q=2c+1$.

また $0\leqq a<2^{i+j-k}<2^{i-k}\cdot 2^i<2^{2i-2}<2^{2i-1},$

$0<2^{j-k+1}\cdot y<2^i\cdot 2^{1-k}\leqq 2^{i-1},$

$0\leqq c<(2^{i+j-k}\cdot y)/(2^{j-k+1}\cdot y)=2^{i-1},$

$0\leqq d<2^{j-k+1}\cdot y<2^{i-1}.$

3.　2 進 10 進変換の各種アルゴリズム

3.1　アルゴリズム A とその計算機上での実現

2 進整数 Z がメモリ上に与えられたとき，以下の手順によっていったん 10^{12} 進数に変換し，後にその各桁をそれぞれ 10 進数に変換する．結果を上位桁から順に知る必要がなければ 10^{12} 進数の 1 桁を知る度に 10 進数に変換しても良いが本質的には同じなので本論文では省略する．また，10^n 進数を 10 進数に変換することは，n が小さいとき容易かつ十分に短い時間でできるので，その方法について本論文では触れない．

アルゴリズム A：

2 進数 Z ($\geqq 10^{12}$) を 10^{12} 進数に変換する．

まず 10^{12} 進数を一時的に保存する作業領域を用意する*．以下ここへの書き込みは push と表記する．

A-1：$(a,\ b) \leftarrow Z\div 2^8$

* 厳密には $5\cdot\lceil(\log_{16}(Z+1))/12\rceil$ バイト必要である．

A-2 : $(Z, c) \leftarrow a \div 2^4 \cdot 5^{12}$

A-3 : $d \leftarrow 2^8 \cdot c + b$; push d

A-4 : if $Z \geqq 10^{12}$ then go to A-1

A-5 : push Z

A-1 は a と b が元々メモリ内で分かれて格納されているので実際は何もしなくて良い．A-2 は使用する CPU に依存するので後述する．A-3 は push b; push c と置き換えることができるので計算は全く不要である．A-4 での条件判断はさらに単純化することができる．改良した方法を次に示す．

アルゴリズム A′:

2進数 Z を 10^{12} 進数に変換する．

まず 10^{12} 進数を一時的に保存する作業領域を用意する*.

A′-1 : $(a, b) \leftarrow Z \div 2^8$

A′-2 : $(Z, c) \leftarrow a \div 2^4 \cdot 5^{12}$

A′-3 : $d \leftarrow 2^8 \cdot c + b$; push d

A′-4 : if $Z \geqq 2^{32}$ then go to A′-1

A′-5 : push Z

アルゴリズム A′ では A′-4 における条件判断が単なるメモリ上でのサイズの比較で行えるので容易である．ただし得られた 10^{12} 進数の最上位桁に 0 の入ることがあるのに注意を要す．また，比較的小さな整数の変換はアルゴリズムAに比べむしろ遅くなる可能性があるが，それはそもそも本論文の趣旨に反するので考えない．

次に A-2 (A′-2) の実現方法について述べる．

（1） 64 ビット÷32 ビットの符号なし整数除算のできる CPU での A-2 の実現

例えば MC 68020[2] がこれにあたる．

$2^4 \cdot 5^{12} = 0xe8d4a510 < 2^{32}$ であるから単に $0 \leqq X < 2^{36} \cdot 5^{12}$, $Y = 2^4 \cdot 5^{12}$ なる整数 X, Y による除算 $(X \div Y)$ を（筆算の要領で）上の桁から順に繰り返せば良い（$0x\cdots$ は16進表記を表す）．

（2） 64 ビット÷32 ビットの符号付き整数除算ができる CPU での A-2 の実現

例えば VAX[3] がこれにあたる．商と剰余が共に 32 ビット符号付き整数の場合，本質的には 62 ビット÷31ビットの除算しかできない．この場合は次のように除算とビット・シフトを組み合わせることによって 64 ビット÷32 ビットの除算を実現する．

アルゴリズム A1:

$0 \leqq X < 2^{36} \cdot 5^{12}$, $Y = 2^4 \cdot 5^{12}$ なる整数 X, Y について $(q, r) \leftarrow X \div Y$ を求める．

A1-1 : $(a, b) \leftarrow X \div 2^5$

A1-2 : $(c, d) \leftarrow a \div 5^{12}$

A1-3 : $r \leftarrow 2^5 \cdot d + b$; $q \leftarrow 2 \cdot c$

A1-4 : if $r \geqq 2^4 \cdot 5^{12}$ then
　　$(q \leftarrow q+1;\ r \leftarrow r - 2^4 \cdot 5^{12})$

アルゴリズム A1 の正当性は，定理2において，$i=32$, $j=4$, $k=5$, $y=5^{12}$ とおくことにより説明できる．

（3） 32 ビット÷16 ビットの符号なし整数除算のみができる CPU での A-2 の実現

例えば MC 68010[4] がこれにあたる．この場合はアルゴリズム A2 を用いた 48 ビット÷32 ビットの除算を上の桁から順に繰り返せば良い．

アルゴリズム A2:

$0 \leqq X < 2^{20} \cdot 5^{12}$, $Y = 2^4 \cdot 5^{12}$ なる整数 X, Y について $(q\ r) \leftarrow X \div Y$ を求める．

$X = aW^2 + bW + c$, $Y = dW + e$ とおく．

ただし $0 \leqq a, b, c, d, e < W = 2^{16}$

A2-1 : if $a = d$ then go to A2-6

A2-2 : $(q, f) \leftarrow (aW+b) \div d$

A2-3 : $r \leftarrow fW + c - qe$

A2-4 : if $r < 0$ then $(q \leftarrow q-1;\ r \leftarrow r+Y)$

A2-5 : exit

A2-6 : $q \leftarrow W-1$; $r \leftarrow (d-e)W + bW + c + e$

（W, d, e は定数であることに注意せよ）

アルゴリズム A2 の正当性は，定理1において $D = 2^{16}$, $aD^2 + bD + c = X$, $dD + e = Y$ とおくことにより説明できる．なぜなら，$Y = 2^4 \cdot 5^{12} = 0xe8d4a510$ により $e \leqq d$ であり，また $a < d$ の場合は A2-2 の除算が1命令で実行できるからである．

ここで，$aW + b < dW + e$ であるから，ほとんどの場合は $a < d$ である．したがって最初の2ステップは次のように直すのが実際的である．

A2-1′ : $(q, f) \leftarrow (aW+b) \div d$

A2-2′ : if overflowed then go to A2-6

また，MC 68010 のように商と剰余がレジスタの下位および上位のワードに得られる CPU の場合は，A2-3 を

A2-3′ : $r \leftarrow (fW+q) + c - q(e+1)$

のように直すのが良い．

アルゴリズム A2 では

(1 word, 2 word) ← 3 word ÷ 2 word の除算を (1 word, 1 word) ← 2 word ÷ 1 word の除算と 2 word ←

* 厳密には $5 \cdot \lceil (\log_{10}((Z+1)/2^{32}))/12 + 1 \rceil$ バイト必要である．

1 word×1 word の乗算それぞれ1回ずつで実行することができる.

3.2 アルゴリズム B とその計算機上での実現

アルゴリズムBは本質的にはアルゴリズム A′ と同じであるが, 一度に可能な除算のサイズがより大きな CPU 向きのものである. 2進整数を以下の手順によっていったん 10^{26} 進数に変換し, 後に各桁をそれぞれ10進数に変換する.

アルゴリズムB:

2進数 Z を 10^{26} 進数に変換する.

まず 10^{26} 進数を一時的に保存する作業領域を用意する*.

B-1: $(a, b) \leftarrow Z \div 2^{24}$

B-2: $(Z, c) \leftarrow a \div 2^2 \cdot 5^{26}$

B-3: $d \leftarrow 2^{24} \cdot c + b$; push d

B-4: if $Z \geqq 2^{88}$ then go to B-1

B-5: push Z

$2^2 \cdot 5^{26} = 0x52b7d2dc, c80cd2e4 < 2^{64}$ であるから B-2 は単に64ビットの整数 $2^2 \cdot 5^{26}$ による除算を繰り返せば良い. しかし現在の CPU ではまだそのような大型除算命令は一般的ではない.

3.3 アルゴリズム C とその計算機上での実現

アルゴリズム C はアルゴリズム B を現在の汎用 CPU 向きに改良したものである. 2進整数を以下の手順によっていったん 10^{24} 進数に変換し, 後に各桁をそれぞれ10進数に変換する.

アルゴリズムC:

2進数 Z を 10^{24} 進数に変換する.

まず 10^{24} 進数を一時的に保存する作業領域を用意する**.

C-1: $(a, b) \leftarrow Z \div 2^{16}$

C-2: $(Z, c) \leftarrow a \div 2^8 \cdot 5^{24}$

C-3: $d \leftarrow 2^{16} \cdot c + b$; push d

C-4: if $Z \geqq 2^{72}$ then go to B-1

C-5: push Z

C-2 の実現には $0 \leqq X < 2^{40}$, $Y = 2^8 \cdot 5^{24}$ なる整数 X, Y による除算 $(X \div Y)$ をアルゴリズム C1 を用いて上の桁から順に繰り返せば良い.

アルゴリズムCはアルゴリズムBと比べ, 変換の効率は劣るが, 64 ビット÷32 ビットの除算命令で容易に実現できる. また, すべてのデータ・アクセスが16ビット単位で行えるという利点がある.

* 厳密には $11 \cdot \lceil (\log_{10}((Z+1)/2^{88}))/26+1 \rceil$ バイト必要である.

** 厳密には $10 \cdot \lceil (\log_{10}((Z+1)/2^{72}))/24+1 \rceil$ バイト必要である.

(1) 64 ビット÷32 ビットの符号なし整数除算のできる CPU での C-2 の実現

アルゴリズム C1:

$0 \leqq X < 2^{40} \cdot 5^{24}$, $Y = 2^8 \cdot 5^{24}$ なる整数 X, Y について $(q, r) \leftarrow X \div Y$ を求める.

$X = aL^2 + bL + c$, $Y = dL + e$ とおく.

ただし $0 \leqq a, b, c, d, e < L = 2^{32}$

C1-1: $(q, f) \leftarrow (aL+b) \div d$

C1-2: if overflowed then go to C1-6

C1-3: $r \leftarrow fL + c - qe$

C1-4: if $r < 0$ then $(q \leftarrow q-1;\ r \leftarrow r+Y)$

C1-5: exit

C1-6: $q \leftarrow L-1;\ r \leftarrow (d-e)L + bL + c + e$

アルゴリズム C1 の正当性は, 定理1において $D = 2^{32}$, $aD^2 + bD + c = X$, $dD + e = Y$ とおくことにより説明できる. なぜなら, $Y = 0xd3c21bce, cceda100$ により $e < d$ であり, また $a < d$ の場合は C1-1 の演算が桁あふれなしに実行できるからである.

アルゴリズム C1 では

(2 word, 4 word) ← 6 word÷4 word の除算を (2 word, 2 word)←4 word÷2 word の除算と 4 word←2 word×2 word の乗算それぞれ1回ずつで実行することができる.

(2) 64 ビット÷32 ビットの符号付き整数除算ができる CPU での A-2 の実現

アルゴリズム C1 では, C1-1 で 64 ビット÷32 ビットの符号なし整数除算が必要である. これはアルゴリズム C2 により 64 ビット÷32 ビットの符号付き整数除算で実現できる.

アルゴリズム C2:

$0 \leqq X < 0xd3c21bce, 00000000$, $Y = 0xd3c21bce$ なる整数 X, Y について $(q, r) \leftarrow X \div Y$ を求める

C2-1: $(a, b) \leftarrow X \div 4$

C2-2: $(c, d) \leftarrow a \div 0x69e10de7$

C2-3: $r \leftarrow 4d + b;\ q \leftarrow 2c$

C2-4: if $r \geqq 0xd3c21bce$ then

$(q \leftarrow q-1;\ r \leftarrow r - 0xd3c21bce)$

アルゴリズム C2 の正当性は, 定理2において $i = 32$, $j = 1$, $k = 2$, $y = 0x69e10de7$ とおくことにより説明できる.

4. 有効性

ここでは本論文で紹介した各アルゴリズムの有効性を考察する.

4.1 乗除算回数の比較

一般に，$0 \leqq x < 2^{8c} b$, $0 < b$ なる整数 x, b による除算 $(x \div b)$ を用いて，整数 Z（サイズ n バイト）を変換したとし，1回のループごとに縮小できるサイズを c バイトとおく（A-1:, A′-1:, B-1:, C-1: において 2^{8c} で割ることを指す）．変換に要する（CPU の命令としての）除算の回数を n^2 のオーダのみ求めると，

$$\sum_{0 \leq i,\, (2^{8c}b)^i < Z} \frac{\log_2(Z/((2^{8c}b)^i))}{a}$$

$$= \frac{1}{a} \sum_{0 \leq i,\, (2^{8c}b)^i < Z} (\log_2 Z - i(\log_2 b + 8c))$$

$$\fallingdotseq \frac{(\log_2 Z)^2}{2a(\log_2 b + 8c)}$$

$$\fallingdotseq \frac{32}{a(\log_2 b + 8c)} \cdot n^2 \quad \because \ \log_2 Z \fallingdotseq 8n$$

であり，$\dfrac{32}{a(\log_2 b + 8c)} \cdot 10^4 = K(a, b, c)$ とおくとそれぞれのアルゴリズムで必要な除算命令の実行回数は，ほぼ $K(a, b, c)$ に比例する．これをまとめたのが表1である．表中，例えばアルゴリズムAにアルゴリズム A1 を用いることを A&A1 と略記した．

アルゴリズム適用の効果は使用する CPU における，除算・乗算の演算速度比およびそれらの演算サイズによる速度比にも依存する．例えば MC 68010 の場合は，n バイト（n は十分に大きい）の2進数を変換するのに，10^4 による除算の繰り返しという古典的方法[1]を用いると約 $0.15 \times n^2$ 回の除算が必要だが，アルゴリズムAにアルゴリズム A2 を用いると約 $0.05 \times n^2$ 回の除算および乗算で済む．除算は乗算の約 2.7 倍の時間を要すので，全体として後者の方法が約 2.2 倍速い．

4.2 演算時間による比較

前節では乗除算回数による検証を行ったが，近年では乗除算回路の高速化も予想され，乗除算回数の比較は必ずしもアルゴリズムの実際的な効果を示しているとは言えない．

本節では実際に基数変換プログラムを実行するのに要した時間を求める．測定するデータには

1° 3^{60}（比較的小さな数，2進数で 96 桁）

2° 1000 の階乗（2進数で 8530 桁）

3° $2^{216091}-1$（知られている最大の素数，2進数で

表 1 各アルゴリズムにおいて必要な乗除算の回数

Table 1 The number of multiplication and division in each algorithm.

$K(a,b,c)$ $(10^{-4}(\mathrm{byte})^{-2})$		演算命令 32÷16 符号無	16×16 符号無	64÷32 符号無	32×32 符号無	64÷32 符号有	96÷64 符号無	128÷64 符号無	算法のタイプ a	b	c
典型的 CPU		MC 68010		MC 68020		VAX					
アルゴリズム	10^4 による除算	1505	—	—	—	—	—	—	16	10^4	0
		—	—	—	—	1505*	—	—	16	10^4	
	10^9 による除算	—	—	334	—	—	—	—	32	10^9	
	10^{19} による除算	—	—	—	—	—	158	—	32	10^{19}	
		—	—	—	—	—	—	79	64	10^{19}	
	A	—	—	251	—	—	—	—	32	$2^4 \cdot 5^{12}$	1
	A&A 1	—	—	—	—	251	—	—	32	$2^4 \cdot 5^{12}$	
	A&A 2	502	502	—	—	—	—	—	16	$2^4 \cdot 5^{12}$	
	B	—	—	—	—	—	116	—	32	$2^2 \cdot 5^{26}$	3
		—	—	—	—	—	—	58	64	$2^2 \cdot 5^{26}$	
	C	—	—	—	—	—	125	—	32	$2^8 \cdot 5^{24}$	2
		—	—	—	—	—	—	63	64	$2^8 \cdot 5^{24}$	
	C&C 1	—	—	125	125	—	—	—	32	$2^8 \cdot 5^{24}$	
	C&C 1 &C 2	—	—	—	125	125	—	—	32	$2^8 \cdot 5^{24}$	

* 32 ビット÷16 ビットの符号無し除算をしたものとする

表 2　2進10進変換に要する時間（秒）
Table 2　Binary-decimal conversion time (in second).

CPU	アルゴリズム	測定データ		
		3^{40}	1000!	$2^{216091}-1$
MC 68010 (SUN 2/120)	10^4 による除算	9.67×10^{-4}	2.53	1600
	A′ & A 2	9.68×10^{-4}	1.35	851
MC 68020 (SUN 3/260)	*10^4 による除算	1.78×10^{-4}	0.411	261
	10^9 による除算	2.00×10^{-4}	0.156	94.7
	* A′ & A 2	2.11×10^{-4}	0.256	170
	A′	1.44×10^{-4}	0.110	76.6
	C & C 1	2.67×10^{-4}	0.100	62.3
VAX (VAX 8600)	10^4 による除算	4.13×10^{-4}	0.954	612
	* A′ & A 2	3.26×10^{-4}	0.396	254
	A′ & A 1	3.48×10^{-4}	0.251	148

* 64 ビット÷32 ビットの除算命令を使用しない場合

216091 桁）

の3つを採用した．その結果を**表2**に示す．

アルゴリズムAは除算命令が高速な CPU では効果が小さいこと，アルゴリズム A2 およびアルゴリズム C1 は除算命令に比べて乗算命令が高速な CPU で効果が大きいこと，などに注意すべきである．

4.3 言語処理系の上での性能評価

本論文で報告したアルゴリズムの一部は，実際に UtiLisp 32[5)~9)] 処理系の無限精度整数出力ルーチンで使用されている．UtiLisp 32 は現在 SUN 2 (MC 68010)，SUN 3 (MC 68020)，VAX の3種類を公開しているが，いずれもアルゴリズム A′ を用いている．ただし SUN 2 版ではアルゴリズム A2 を，VAX 版ではアルゴリズム A1 を併用している．

本アルゴリズムの実際的な効果を知るために，本アルゴリズムが適用された UtiLisp 32 と，他の言語処理系との比較を行う．1000 の階乗*を計算しておき，それを表示するのに要する時間をまとめたのが**表3**である．厳密には基数変換以外の時間（表示など）も含まれるが，無視できる程度である（表2と比べよ）．

UtiLisp 32 の場合，SUN 3 版および VAX 版に関してはアルゴリズムCを用いれば，より一層の高速化を図ることができるが，今のところその必要は生じていない．

* 既存のデータが多かった[10)]ので採用した．
** 前者に関しては，上位の数が偶数であるという性質は利用していない．

表 3　各種言語処理系での 1000! の表示時間
Table 3　Time to print 1000! on each language-system.

マシン	言語処理系	時間（秒）
SUN 2/120	UtiLisp 32	1.38
	Franz Lisp	23.1
	Franz CL	15.1
	Lucid CL	11.5
SUN 3/260	UtiLisp 32	0.117
	Franz CL	2.08
	Franz Lisp	5.34
	Lucid CL	1.74
	Kyoto CL	9.36
VAX 8600	VaxUtiLisp	0.261
	Franz Lisp	0.333
	Kyoto CL	14.5
Symbolics 3645	Symbolics CL	2.50
TI-Explorer	TI-Explorer	1.94
μVAX-II	VAXLISP	12.8

CL：Common Lisp の略

5. おわりに

本論文のアルゴリズムの実現には2つの魔法の数を用いた．それは $2^4\cdot5^{12}$（$=0xe8d4a510$）および $2^8\cdot5^{24}$（$=0xd3c21bce, cceda100$）である．これらの数は単に4バイト（後者は8バイト）で表現できる大きな数というだけでなく，5のべきと2のべきの差（前者では $12-4=8$，後者では $24-8=16$）が8の倍数であり，上位の値（前者は上位2バイト・後者は上位4バイトの値）が下位の値よりも大きく，しかも上位の値は偶数である，という特殊な性質を持つ**．したがって本論文における基数変換アルゴリズムを他の基数法へ応用するのは難しいが，2進10進変換に関してはすぐれた方法であると言える．

謝辞　日頃から有益な助言を頂いている，東京大学工学部計数工学科の和田英一教授に厚く感謝いたします．

参考文献

1) Knuth, D. E.: *The Art of Computer Programming*, Vol. 2, p. 688, Addison-Wesley, Reading, Mass. (1969).

2) MC 68020 32-Bit Microprocessor User's Manual, Motorola, N. J. (1984).
3) 日本 DEC 教育部編：VAX アーキテクチャ・ハンドブック，p. 645，共立出版，東京 (1984).
4) M 68000 16/32-BIT MICROPROCESSOR Programmer's Reference Manual, Motorola, N. J. (1984).
5) 近山　隆：Utilisp システムの開発，情報処理学会論文誌，Vol. 24, No. 5, pp. 599-604 (1983).
6) Chikayama, T.: Utilisp Manual, Technical Report, METR 81-6, Dept. of Mathematical Engineering and Instrumentation Physics, Faculty of Engineering, Univ. of Tokyo (1981).
7) 和田英一，富岡　豊：UtiLisp の MC 68000 への移植，情報処理学会記号処理研究会資料，84-29, pp. 15-21 (1984).
8) Kaneko, K. and Yuasa, K.: A New Implementation Technique for the UtiLisp System, 情報処理学会記号処理研究会資料，41-7 (1987).
9) 金子敬一：VAX 上の UtiLisp，東京大学大型計算機センター・センターニュース，Vol. 19, No. 2, pp. 35-39 (1987).
10) White, J. L.: Reconfigurable, Retargetable Bignums: A Case Study in Efficient, Portable Lisp System Building, *Proc. of the 1986 ACM Conf. on Lisp and Functional Programming*, pp. 174-191 (1986).

付　　録　アルゴリズム A の略証

A-1 における Z を $Z1$，A-2 で求める Z を $Z2$ と書くと，$Z1=2^8a+b$，$a=2^4\cdot 5^{12}\cdot Z2+c$ だから

$$Z1-(2^8\cdot c+b)=2^8(2^4\cdot 5^{12}\cdot Z2+c)+b-(2^8\cdot c+b)$$
$$=10^{12}\cdot Z2.$$

よって，$(Z2,\ 2^8\cdot c+b)=Z1\div 10^{12}$.

すなわちアルゴリズム A は，Z を 10^{12} で割って剰余を push することを繰り返すものである．

アルゴリズム B，アルゴリズム C も同様である．

(昭和 63 年 3 月 15 日受付)
(平成 元 年 7 月 18 日採録)

高橋　俊成（正会員）
1962 年 12 月 19 日（水）生．1986 年東京大学工学部計数工学科（数理工学専修）卒業．在学中はアナログ・レコードの光学式再生装置の研究・試作を経験．同年同大学院工学系研究科情報工学専門課程修士課程に入学し，計算機分野に転向．1988 年同課程修了．同年（株）東芝入社．現在，同社総合研究所情報システム研究所にて基本ソフトウェアの研究に従事．