Vol. 37 No. 3　　情　報　処　理　　Mar. 1996

連載講座

算術演算回路のアルゴリズム

3. 除算回路のアルゴリズム†

—減算シフト型か乗算型か—

高　木　直　史††

1. まえがき

第3回は除算回路および開平（平方根計算）回路のアルゴリズムについて述べる．アルゴリズムを大別すると，減算シフト型，乗算型，多項式近似（補間），その他（CORDIC法など）に分類できる．本稿では，最近のマイクロプロセッサで実際に用いられている減算シフト型と乗算型の除算法および開平法について述べる．

以前は多くのシステムで，加減算器とシフタからなる演算回路上で減算シフト型の除算および開平を実現していた．高速乗算器を持つシステムでは，初期近似値用のROMを用意することにより乗算型の除算および開平を実現できる．減算シフト型に比べ高速であるが，除算や開平の実行中，乗算器が占用され乗算などが実行できなくなる．このため，最近のマイクロプロセッサでは，減算シフト型の除算および開平を実現する専用のハードウェアを用意しているものが多くなってきている．実用されている専用ハードウェアでは，演算の途中結果を，第1回で述べた桁上げ保存形などの冗長表現で保持し，各加減算で桁上げが伝搬しないようにし，高速計算を実現している．

本稿では，次章で減算シフト型除算法，3.で乗算型除算法について述べる．4.で，減算シフト型および乗算型の開平法について述べる．

2. 減算シフト型除算法

本章と次章では，nビット符号なし2進小数の除算について考える．被除数Xと除数Yはともに正規化されており，$\frac{1}{2}\leq X$，$Y<1$を満たすものとする．本章では，説明を簡単にするため$X<Y$と仮定する．$X<Y$が保証されない場合にも容易に拡張できる．$|X/Y-Z|<2^{-n}$を満たす商Zを小数点以下nビット目まで求めるものとする．

減算シフト型除算法は，人が筆算で除算を行う場合と同様，減算とシフトの繰り返しにより除算を行うものである．商とともに剰余も求まる．nに比例する回数の減算とシフトが必要である．

基数rの減算シフト型除算法では，$R_j:=rR_{j-1}-q_jY$という漸化式に従い，商を上位から一桁ずつ求めていく．ここに，q_jは商の小数点以下j桁目，R_jはq_jを決定した後の部分剰余である．基数rをいくつにするか，q_jをどのような桁集合から選ぶか，R_jの表現に冗長表現を用いるかどうか，などにより種々のバリエーションが考えられる．減算シフト型除算については，多くの研究がなされており，文献1）に詳しくまとめられている．

2.1 基数2の減算シフト型除算法

まず，基本的な基数2の回復型除算法，非回復型除算法，SRT除算法について述べる．基数2の減算シフト型除算法では，$R_0=X$とし，

$$R_j:=2R_{j-1}-q_jY \qquad (1)$$

という漸化式に従って計算を行う．

基数2の回復型（restoring）除算法では，(1)の漸化式で，$q_j\in\{0,1\}$とし，$2R_{j-1}-Y$が非負ならq_jを1，負なら0とする．すなわち，まず$R'_j:=2R_{j-1}-Y$を計算し，$R'_j\geq 0$なら$q_j:=1$，$R_j:=R'_j$とし，$R'_j<0$なら$q_j:=0$，$R_j:=R'_j+Y$とする．常に$0\leq R_j<Y$が成り立つ．$R'_j<0$のとき，R'_jに再びYを加えて$2R_{j-1}$に戻すことから，回復型除算法あるいは引き戻し法と呼ばれる．途中で$R_j:=0$になれば，計算を終了するようにしてもよい．専用回路で実現する場合

† Algorithms for Arithmetic Circuits 3 Algorithms for Dividers by Naofumi TAKAGI (Department of Information Engineering, Graduate School of Engineering, Nagoya University).

†† 名古屋大学大学院工学研究科情報工学専攻

は，セレクタで$2R_{j-1}-Y$と$2R_{j-1}$の一方を選択するようにしてもよい．

基数2の非回復型（non-restoring）除算法では，(1)の漸化式で，$q_j\in\{-1,1\}$とし，$2R_{j-1}$が非負ならq_jを1，負なら-1とする．常に$-Y\leq R_j<Y$が成り立つ．$2R_{j-1}-q_jY$が負になっても，これをそのままR_jとすることから，非回復型除算法あるいは引き放し法と呼ばれる．商を通常の2進表現に変換する必要があるが，2進表現の商Zのj桁目z_jはq_{j+1}が1なら1，-1なら0となり，変換には特に計算の必要はない．剰余Rを$R\geq 0$とするには，$R_n\geq 0$なら$z_n:=1$，$R:=R_n$とし，$R_n<0$なら$z_n:=0$，$R:=R_n+Y$とする．途中で$2R_{j-1}=0$になれば，計算を終了するようにしてもよい．商の変換を漸化式に組み込むことも可能である．$z_1:=1$，$R_1:=2X-Y$，$R_2:=2R_1-Y$とし，$j\geq 2$について，$2R_j\geq 0$なら$z_j:=1$，$R_{j+1}:=2R_j-Y$，$2R_j<0$なら$z_j:=0$，$R_{j+1}:=2R_j+Y$とする．

Sweeney，Robertson，Tocherによりほぼ同時期に独立に考案された基数2のSRT除算法では，$q_j\in\{-1,0,1\}$とし，$q_j=0$のときR_jの計算に加減算が不要になるようにしている．常に$-Y\leq R_j<Y$が成り立つようにq_jを選ぶには，$(-2Y\leq)2R_{j-1}<0$なら$q_j:=-1$，$R_j:=2R_{j-1}+Y$，$-Y\leq 2R_{j-1}<Y$なら$q_j:=0$，$R_j:=2R_{j-1}$，$0<2R_{j-1}(<2Y)$なら$q_j:=1$，$R_j:=2R_{j-1}-Y$とすればよい．すなわち，$-Y\leq 2R_{j-1}<0$のときは，q_jを-1としても0としてもよく，$0\leq 2R_{j-1}<Y$のときは，q_jを0としても1としてもよい．そこで，$2R_{j-1}<-\frac{1}{2}$なら$q_j:=-1$，$-\frac{1}{2}\leq 2R_{j-1}<\frac{1}{2}$なら$q_j:=0$，$\frac{1}{2}\leq 2R_{j-1}$なら$q_j:=1$とする．すなわち，$2R_{j-1}$と$\pm\frac{1}{2}$との比較により$q_j$を決定する．商の選択が$2R_{j-1}$の符号を含む上位3ビットを調べるだけで行うことができる．$q_j=0$の場合は加減算の必要がないので，シフトのみの場合にクロック長を短くできるような演算器を構成すれば，計算を高速化できる．$-Y\leq R_n<Y$である．剰余を$R\geq 0$とするには，$R_n<0$のとき，商から2^{-n}を減じ，$R:=R_n+Y$とする．商が冗長2進表現で求まるので，通常の2進表現に変換する必要がある．

これら3つの除算法に基づく回路では，加減算器に何を用いるかにより，1ステップ（商の桁の選択と次の部分剰余の計算）にかかる計算時間が大きく左右される．順次桁上げ加算器を用いればnに比例し，桁上げ先見加算器を用いれば$\log n$に比例する．このステップがn回繰り返される．

2.2 部分剰余の冗長表現

部分剰余を，本講座の第1回で述べた桁上げ保存形あるいは冗長2進表現で表し，各ステップでの加減算を桁上げの伝搬なしに行えれば，計算を高速化できる．

桁上げ保存形や冗長2進表現では，正確な大小比較のためには最悪の場合全桁を調べる必要があり，大きな計算時間を要する．SRT除算法の場合と同様に，商の桁集合を$\{-1,0,1\}$とすれば，商の選択において$2R_{j-1}<0$であるか$-Y\leq 2R_{j-1}<Y$であるか$0\leq 2R_{j-1}$であるかの判定を行えばよく，$2R_{j-1}$のおおまかな値が分かればよい．すなわち，商の選択の隣り合う領域に重なりを設けることにより，商の選択において正確な大小比較を不要にすることができる．

部分剰余を桁上げ保存形で表す場合は，桁上げ保存形で表された$R_{j-1}(-1<-Y\leq R_{j-1}<Y<1)$の小数点以下2桁目まで（符号桁を含む上位3桁）の値を$\check{R}_{j-1}$として，$2\check{R}_{j-1}<-\frac{1}{2}$ならば$q_j:=-1$，$-\frac{1}{2}\leq 2\check{R}_{j-1}<0$ならば$q_j:=0$，$0\leq 2\check{R}_{j-1}$ならば$q_j:=1$とする．$\check{R}_{j-1}\leq R_{j-1}<\check{R}_{j-1}+2^{-1}$であるから，$q_j$として$-1$が選択されたときは$2R_{j-1}<0$，0のときは$-Y\leq 2R_{j-1}<Y$，1のときは$0\leq 2R_{j-1}$が保証されている．**図-1**に，$2R_{j-1}$と$R_j$の関係を示す．（この図をRobertson図という．）**図-2**に除算の例を示す．（商の-1を$\bar{1}$で表している．部分剰余の負数の表現には2の補数表示を用いている．）

部分剰余を冗長2進表現で表す場合は，R_{j-1}を1桁の整数部を持つ冗長2進数で表し，その上位3桁の値が正か零か負により，q_jを1か0か-1とする[2]．

いずれの場合も，商q_jの選択は部分剰余の上位3桁の値を調べることによって行え，次の部分

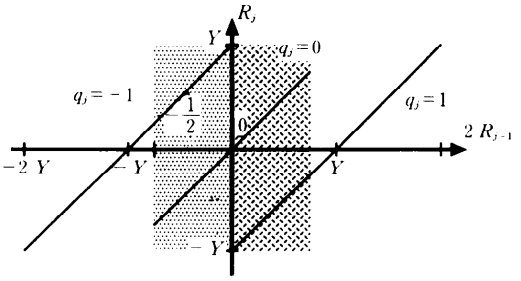

図-1　部分剰余を桁上げ保存形で表した場合の Robertson 図

被除数：$X=[0.100111]$，除数：$Y=[0.110001]$

	RS_0		0.	1	0	0	1	1	1	
	RC_0		0.	0	0	0	0	0		
	$2RS_0$	0	1.	0	0	1	1	1		
	$2RC_0$	0	0.	0	0	0	0			
$q_1=-1$	$+(-Y)$	1	1.	0	0	1	1	1	0	$+1$
	RS_1		0.	0	0	0	0	0	1	
	RC_1		0.	0	1	1	1	0		
	$2RS_1$	0	0.	0	0	0	0	1		
	$2RC_1$	0	0.	1	1	1	0			
$q_2=1$	$+(-Y)$	1	1.	0	0	1	1	1	0	$+1$
	RS_2		1.	1	1	0	1	0	1	
	RC_2		0.	0	1	0	1	0		
	$2RS_2$	1	1.	1	0	1	0	1		
	$2RC_2$	0	0.	1	0	1	0			
$q_3=-1$	$+(-Y)$	1	1.	0	0	1	1	1	0	$+1$
	RS_3		0.	0	0	1	1	0	1	
	RC_3		1.	0	1	0	1	0		
	$2RS_3$	0	0.	0	1	1	0	1		
	$2RC_3$	1	0.	1	0	1	0			
$q_4=\bar{1}$	$+Y$	0	0.	1	1	0	0	0	1	
	RS_4		0.	0	0	0	0	1	1	
	RC_4		1.	1	1	0	0	0		
	$2RS_4$	0	0.	0	0	0	1	1		
	$2RC_4$	1	1.	1	0	0	0			
$q_5=0$	$+0$	0	0.	0	0	0	0	0	0	
	RS_5		1.	1	0	0	1	1	0	
	RC_5		0.	0	0	0	0	0		
	$2RS_5$	1	1.	0	0	1	1	0		
	$2RC_5$	0	0.	0	0	0	0			
$q_6=\bar{1}$	$+Y$	0	0.	1	1	0	0	0	1	
	RS_6		1.	1	1	1	1	0	1	
	RC_6		0.	0	0	0	0	0		

商：$Z=[0.111\bar{1}0\bar{1}]=[0.110011]$

図-2　部分剰余の表現に桁上げ保存形を用いた基数 2 の除算の例

剰余 R_j の計算は桁上げの伝搬なしに行える．(冗長 2 進加算では，Y は非冗長な 2 進数であるから，通常の冗長 2 進加算より簡単である．) 1 ステップにかかる計算時間（回路の段数）は n に無関係な一定値になる．

SRT 除算の場合と同様，$R\geq 0$ とするには，$R_n<0$ のとき，商から 2^{-n} を減じ，$R:=R_n+Y$ とする．商が冗長 2 進表現で求まるので，通常の 2 進表現に変換する必要がある．また，剰余も求める必要がある場合は，剰余を通常の 2 進表現に変換する必要がある．

2.3　商の on-the-fly 変換

部分剰余を冗長表現で表すことにより計算を高速化できるが，商を冗長 2 進表現から通常の 2 進表現に変換する必要がある．冗長 2 進表現の商がすべて求まってから 2 進表現に変換するようにすると，変換に要する時間が計算時間に上積みされることになる．そこで，この変換を商の選択と並行して 'on-the-fly' に行う手法が提案されている[3]．

冗長 2 進表現で上位桁から順に求まる商 Q と 2 進表現の商 Z の小数点以下 j 桁目までをそれぞれ $Q_j(=[.q_1q_2\cdots q_j])$ と $Z_j(=[.z_1z_2\cdots z_j])$ とすると，Z_j は Q_j または Q_j-2^{-j} のいずれかの 2 進表現となる．すなわち，Q の小数点以下 $j+1$ 桁目以降が非負ならば Q_j，負ならば Q_j-2^{-j} の 2 進表現となる．したがって，各ステップで，Q_j および Q_j-2^{-j} の 2 進表現を保持しておけば，q_n が求まった時点で（$R\geq 0$ とする場合は R_n の正負が判明した時点で），Z が求まる．

Q_j と Q_j-2^{-j} の 2 進表現をそれぞれ ZP_j と ZN_j とする．各ステップで，q_j が求まれば，ZP_{j-1} と ZN_{j-1} から次のように ZP_j と ZN_j を求める．$q_j=-1$ のとき，$ZP_j=ZN_{j-1}+2^{-j}$（j 桁目に 1 を付加），$ZN_j=ZN_{j-1}$（j 桁目に 0 を付加）とする．$q_j=0$ のとき，$ZP_j=ZP_{j-1}$，$ZN_j=ZN_{j-1}+2^{-j}$ とする．$q_j=1$ のとき，$ZP_j=ZP_{j-1}+2^{-j}$，$ZN_j=ZP_{j-1}$ とする．これらは，レジスタの $j-1$ 桁目までの内容のコピーと j 桁目への値（0 または 1）の設定であり，n に無関係な一定時間で行える．$Z=ZP_n$（$R\geq 0$ とする場合は，R_n が非負なら $Z=ZP_n$，負なら $Z=ZN_n$）である．

図-3 に，商の on-the-fly 変換の例を示す．

2.4　高基数除算法

高基数（$r>2$）の除算法では，商の選択において，部分剰余と除数のいくつかの倍数との比較が必要となる．このため高速化のためには商の桁集合を冗長にし，商の選択を容易にすることが不可

$q_1=1$	ZP_1	0.	1					
	ZN_1	0.	0					
$q_2=1$	ZP_2	0.	1	1				
	ZN_2	0.	1	0				
$q_3=1$	ZP_3	0.	1	1	1			
	ZN_3	0.	1	1	0			
$q_4=\bar{1}$	ZP_4	0.	1	1	0	1		
	ZN_4	0.	1	1	0	0		
$q_5=0$	ZP_5	0.	1	1	0	1	0	
	ZN_5	0.	1	1	0	0	1	
$q_6=\bar{1}$	ZP_6	0.	1	1	0	0	1	1
	ZN_6	0.	1	1	0	0	1	0

図-3 商の on-the-fly 変換の例

欠となる．そこで，商の桁集合を $\{-a, \cdots, -1, 0, 1, \cdots, a\}$ $(\frac{r}{2} \leq a \leq r-1)$ とする[4]．

収束を保証するためには，$0<\rho\leq 1$ である定数 ρ が存在し，$-\rho Y \leq R_j < \rho Y$ で，$-r\rho Y + aY = -\rho Y$，$r\rho Y - aY = \rho Y$ が成り立たなければならない．よって，$\rho = a/(r-1)$ である．また，商の桁 q_j として c を選択してもよい rR_{j-1} の範囲を $L_c \leq rR_{j-1} < U_c$ とすると，$L_c - cY = -\rho Y$，$U_c - cY = \rho Y$ より，$L_c = (-\rho + c)Y$，$U_c = (\rho + c)Y$ であり，商の選択の領域が連続しているためには，$L_c \leq U_{c-1}$ でなければならないので，$(-\rho + c)Y \leq (\rho + c - 1)Y$，すなわち，$\rho \geq \frac{1}{2}$ でなければならない．以上より，$\frac{1}{2} \leq \rho = a/(r-1) \leq 1$ という条件を得る．$\frac{r}{2} \leq a \leq r-1$ であるから，この条件は成り立っている．

商の選択は，rR_{j-1} と Y の関数になる．この関数は，$-a \leq c \leq a$ である c について，$\{S_c\}$ を定め，$S_c \leq rR_{j-1} < S_{c+1}$ のとき $q_j := c$ とすることにより定義できる．商の選択の境界 S_c は，$L_c \leq S_c \leq U_{c-1}$，すなわち，$(-\rho + c)Y \leq S_c \leq (\rho + c - 1)Y$ を満たさなければならない．$U_{c-1} - L_c = (2\rho - 1)Y$ であるから，ρ が大きいほど，隣り合う領域の重なりが大きく，S_c の設定に自由度が大きくなる．しかし，ρ を大きくするということは a を大きくすることを意味し，多くの比較が必要になる．

S_c は，$(-\rho + c)Y \leq S_c \leq (\rho + c - 1)Y$ を満たし，生成が容易なものを選べばよい．S_c の範囲は Y に依存するので，S_c として Y の生成しやすい倍数 P_c を用いることが考えられる．たとえば，$r=4$，$a=2$ の場合，$\rho = \frac{2}{3}$ であり，$L_{-1} = -\frac{5}{3}Y \leq P_{-1} \leq U_{-2} = -\frac{4}{3}Y$ を満たす $P_{-1} = -\frac{3}{2}Y$，同様に，$P_0 = -\frac{1}{2}Y$，$P_1 = \frac{1}{2}Y$，$P_2 = \frac{3}{2}Y$ が存在し，$S_c = P_c$ とすればよい．

さらに，rR_{j-1} との比較を容易にするには，$L_c \leq \check{P}_c$ を満たす範囲で，P_c の上位数ビット $\check{P}_c$ を S_c とすればよい．S_c として P_c の小数点以下 t ビット目までを用いるとすると，t は，$P_c - 2^{-t} \geq L_c$ を満たす最小の整数にすればよい．Y の上位数ビットから P_c の小数点以下 t ビットまでの近似を計算する場合は，その近似誤差も考慮する必要がある．$Y \geq \frac{1}{2}$ であるから，$r=4$，$a-2$ の場合，$t=4$ とすればよい．この場合，S_c ($\check{P}_c$ の近似) は，近似誤差も考慮して，Y の上位5ビット（最上位の1も含む）から生成できる．($r=4$，$a=2$ の場合を基数4の SRT 除算法あるいは4進 SRT 除算法と呼ぶことがある．)

商の選択の隣り合う領域に重なりを設ければ，部分剰余の表現に桁上げ保存形や冗長2進表現などの冗長表現を用い，部分剰余の上位数桁 $r\check{R}_{j-1}$ と境界 S_c とを比較することにより商を選択することができる．$r\check{R}_{j-1} < S_c$ のとき $rR_{j-1} < U_{c-1}$，$r\check{R}_{j-1} \geq S_c$ のとき $rR_{j-1} \geq L_c$ となればよい．比較に小数点以下 t 桁目までを用いるとすると，桁上げ保存形の場合は打ち切り誤差 ε が $0 \leq \varepsilon < 2^{-t+1}$ となる．S_c として P_c の小数点以下 t ビット目までを用いるとすると，t は，$P_c - 2^{-t} + 2^{-t+1} < U_{c-1}$ かつ $P_c - 2^{-t} \geq L_c$ を満たす最小の整数にすればよい．$r=4$，$a=2$ の場合，$t=4$ とすればよい．この場合，$-\frac{8}{3}Y \leq 4R_{j-1} < \frac{8}{3}Y$ であるから $4R_{j-1}$ は整数部が符号桁も含めて3桁になる．また，S_c は Y の上位5ビット（最上位の1も含む）から生成できる．よって，商の選択は，R_{j-1} の上位7桁と Y の上位5ビット（最上位の1も含む）により行える．

商の選択の境界 S_c は一般に Y に依存するが，Y のとり得る値が狭い範囲に限られていれば，S_c として定数を用いることができる．たとえば，上述の $r=4$，$a=2$ の場合，$Y\approx 1$ であれば，$S_{-1}=-\frac{3}{2}$ とすることができる．そこで，あらかじめ除数 $Y(\frac{1}{2}\leq Y<1)$ にその逆数の（粗い）近似値を乗じ，1に近い値にしておくことを考える．これをスケーリングという[5]．被除数 X にも同じ数を乗じておけば，正しい商が得られる．

2進数の除算の場合，基数 r を通常，2のべきとする．$r=2^k$ の場合，n ビットの除算が n/k ステップで行える．基数が r の場合，商の選択に少なくとも r 個の比較が必要になる．したがって，$r=16$ 程度以上になると効率が悪い．$r=16$ 程度以上の除算は，基数2や基数4の除算を組み合わせて実現するのが実用的である．

2.5 減算シフト型除算器

減算シフト型除算器の実現法としては，1クロックに上述の除算法の1ステップ分あるいは数ステップ分を実行する逐次型実現と，全ステップ分を組合せ回路で行う組合せ回路実現が考えられる．逐次型実現の場合は，部分剰余を記憶するレジスタが必要である．

1ステップ分の計算時間（回路の段数）は，どのような加減算器を用いるかによって決まる．順次桁上げ加算器を用いれば n に比例し，桁上げ先見加算器を用いれば $\log n$ に比例する．部分剰余に冗長表現を用いれば，商の選択が若干複雑になるが，1ステップ分の計算時間は n に無関係な一定値になる．ただし，最終剰余の非冗長表現への変換や正負判定が必要な場合には，これらにかかる時間やハードウェアのコストを考慮する必要がある．また，逐次型実現の際に，部分剰余の記憶に必要なレジスタが2倍になる．

組合せ回路実現の場合，各ステップの加減算器として順次桁上げ加算器を用いれば全体の回路の段数は n^2 に比例し，必要な論理素子数，理論的なVLSI上での面積とも n^2 に比例する．1ビット分の加減算セルの規則正しい2次元配列構造を持ち，配列型除算器と呼ばれる．部分剰余に冗長表現を用い，各ステップの加算器として桁上げ保存加算器あるいは冗長2進加算器を用いれば，配列型除算器と同様，規則正しい配列構造になり，素子数，面積とも n^2 に比例し，段数は n に比例し高速になる．商の2進表現への変換は，商が上位桁から一桁ずつ順に求まることを利用し，ハードウェア量の小さな高速の桁上げ選択加算器で実現できる．

最近のマイクロプロセッサで減算シフト型除算の専用ハードウェアを持つものは，部分剰余の表現に桁上げ保存形を用いた基数4のSRT除算法を採用し，その1ステップ分を組合せ回路で実現しているものが多いようである．今後は，さらに多くのステップ分を組合せ回路で実現した専用ハードウェアが実現され，組合せ回路実現も可能になってくると考えられる．その際には，部分剰余の表現に桁上げ保存形あるいは冗長2進表現を用いた基数2あるいは4のSRT除算法が用いられるものと思われる．すでに，部分剰余の表現に冗長2進表現を用いた基数2の除算法に基づく単精度（仮数部24ビット）の除算器が組合せ回路で実現されている[6]．

3. 乗算型除算法

乗算型除算法は，乗算の繰り返しにより除算を行うもので，高速乗算器を持つシステムで用いられている．Newton法に基づくものやGoldschmidtの方法が用いられている．商のみが求まるので，剰余が必要な場合は，商から剰余を計算する必要がある．

Newton法に基づく除算法では，Newton法により Y の逆数を求める．$1/Y$ の適当な近似値 U_0 から始め，$U_{i+1}:=U_i\cdot(2-U_i\cdot Y)$ という計算の繰り返しにより，$1/Y$ を求める．1回の繰り返しに，乗算が2回必要である．$2-(U_i\cdot Y)$ は $U_i\cdot Y$ の補数をとればよい．二次の収束をし，1回の繰り返しで2倍の桁数の精度になる．必要な精度の $1/Y$ が求まれば，これに X を乗じて商を得る．Newton法に基づく除算アルゴリズムは，第2回で述べた中間積の符号変換を用いた繰り返し乗算の手法により高速化できる．

Goldschmidtの除算法では，分数の分母と分子に同じ数を乗じても値が変わらないことを利用する．$X/Y=(X\cdot D_0\cdot D_1\cdot D_2\cdots)/(Y\cdot D_0\cdot D_1\cdot D_2\cdots)$ であり，分母が1に近付くように D_i を定めると，分子は X/Y に近付く．具体的には，$D_i:=2$

$-Y_i$, $X_{i+1}:=X_i\cdot D_i$, $Y_{i+1}:=Y_i\cdot D_i$ という計算を繰り返す．1回の繰り返しに，乗算が2回必要である．$2-Y_i$ は Y_i の補数をとればよい．二次の収束をする．通常，収束を速めるため，D_0 として $1/Y$ の適当な近似値を用いる．

いずれの除算法でも必要な演算はほぼ同じであるが，Goldschmidt の除算法の方が，1回の繰り返しにおける2つの乗算の独立性が高く，パイプライン処理などにより複数の乗算が高速に実行できるシステムでは有利である．

いずれの除算法でも，$1/Y$ の近似値を用いる．その精度が m ビットのとき，およそ $\log_2(n/m)$ 回の繰り返しが必要である．初期近似値の生成は，テーブルから直接近似値を読み出す方法と，一次近似を用いテーブルから2つの係数を読み出し積和演算によって近似値を生成する方法が考えられる．いずれにしても，テーブル用の ROM が必要である．また，いずれの除算法でも，途中の乗算結果の丸め誤差を考慮すると，乗算器のビット長を商に必要な精度より数ビット多くしておく必要がある．

4. 開平回路のアルゴリズム

n ビット符号なし2進小数の開平について考える．演算数 X は，$\frac{1}{4}\leq X<1$ を満たすものとする．除算と同様，減算シフト型と乗算型の開平法がある．

減算シフト型開平法では，$R_j:=rR_{j-1}-q_j(2Q_{j-1}+q_jr^{-j})$, $Q_j:=Q_{j-1}+q_jr^{-j}$ という漸化式に従い，平方根を上位から一桁ずつ求めていく．

基数が2の場合は，$R_j:=2R_{j-1}-q_j(2Q_{j-1}+q_j2^{-j})$, $Q_j:=Q_{j-1}+q_j2^{-j}$ という漸化式に従う．2進回復型開平法では，$q_j\in\{0,1\}$ とし，$2R_{j-1}-(2Q_{j-1}+2^{-j})$ が非負のとき q_j を1，負のとき0とする．2進非回復型開平法では，$q_j\in\{-1,1\}$ とし，$2R_{j-1}$ が非負のとき q_j を1，負のとき -1 とする．SRT 除算法と同様，$q_j\in\{-1,0,1\}$ とし，$q_j=0$ のとき加減算を不要とした開平法も提案されている．これらの開平法に基づく回路では，加減算器として順次桁上げ加算器を用いれば，1ステップ分の回路の段数は n に比例し，桁上げ先見加算器を用いれば $\log n$ に比例する．

R_j を桁上げ保存形や冗長2進表現などの冗長表現で表すことにより，計算を高速化できる．R_{j-1} の上位3桁の値から $q_j(\in\{-1,0,1\})$ を選択し，R_j の計算を桁上げの伝搬なしに行う．1ステップ分の回路の段数は n に無関係な一定値になる．

平方根の 'on-the-fly' 変換は除算の場合とまったく同様に行える．各ステップで Q_j と Q_j-2^{-j} の2進表現 ZP_j と ZN_j を更新していけばよい．R_j の計算において，$-q_j(2Q_{j-1}+q_j2^{-j})$ を生成する必要がある．$q_j=-1$ の場合は $2Q_{j-1}-2^{-j}=2ZN_{j-1}+2^{-j+1}+2^{-j}$（11の付加），$q_j=1$ の場合は $-2Q_{j-1}-2^{-j}=2ZP'_{j-1}+2^{-j+1}+2^{-j}$，ただし，$ZP'_{j-1}$ は ZP_{j-1} の1の補数，とすればよい．

除算と同様，高基数（$r>2$）の開平も考えられる．q_j の選択の境界は，Q_{j-1} に依存する．すなわち q_j の選択は R_{j-1} と Q_{j-1} の関数になる．R_j の表現に桁上げ保存形や冗長2進表現などの冗長表現を用い，q_j の選択を R_{j-1} と Q_{j-1} の上位数桁により行い，R_j の計算を桁上げの伝搬なしに高速に行うことができる．

除算の場合と同様，減算シフト型開平器の実現法としては，逐次型実現と組合せ回路実現が考えられる．除算との共通点が多いので，除算と共用の回路として実現されることが多い．

乗算型の開平法としても，Newton 法に基づくものや Goldschmidt の方法が用いられている．

Newton 法に基づく開平法では，Newton 法により $\sqrt{X}$ の逆数を求める．$1/\sqrt{X}$ の適当な近似値 U_0 から始め，$U_{i+1}=U_i\cdot(3-U_i^2\cdot X)/2$ という計算の繰り返しにより，$1/\sqrt{X}$ を求める．必要な精度の $1/\sqrt{X}$ が求まれば，これに X を乗じて平方根を得る．

Goldschmidt の開平法では，$X/X^2=(X\cdot D_0^2\cdot D_1^2\cdot D_2^2\cdots)/(X\cdot D_0\cdot D_1\cdot D_2\cdots)^2$ であることを利用する．上式で，分子が1に近付くように D_i を定めると，分母は X に近付き，したがって，$X\cdot D_0\cdot D_1\cdot D_2\cdots$ は $\sqrt{X}$ に近付く．具体的には，$X_0:=X$, $Z_0:=X$ とし，$D_i:=(3-X_i)/2$, $X_{i+1}:=X_i\cdot D_i^2$, $Z_{i+1}:=Z_i\cdot D_i$ という計算を繰り返す．D_0 として $1/\sqrt{X}$ の近似値を用いる．

いずれの開平法も二次の収束をする．1回の繰り返しに，乗算が3回とビット反転およびシフト

が必要である．

5. むすび

連載講座の第3回として，除算回路および開平回路のアルゴリズムについて述べた．

今後は，部分剰余の表現に冗長表現を用いた減算シフト型の除算器および開平器が組合せ回路で実現されるようになるであろう．また，乗算型の除算および開平を実現するための専用の乗算器を設けることも可能となるであろう．減算シフト型を採用するか乗算型を採用するかは，両者を十分理解した上で，除算および開平自体の性能のみならず，システム全体の性能を考慮して決定することが重要である[7]．

参考文献

1) Ercegovac, M. D. and Lang, T.: Division and Square Root—Digit-Recurrence Algorithms and Implementations, Kluwer Academic Publishers (1994).
2) 高木，安浦，矢島：冗長2進表現を利用したVLSI向き高速除算器，電子通信学会論文誌，Vol. J 67-D, No. 4, pp. 450-457 (Apr. 1984).
3) Ercegovac, M. D. and Lang, T.: On-the-fly Conversion of Redundant into Conventional Representations, IEEE Trans. Comput., Vol. C-36, No. 7, pp. 895-897 (July 1987).
4) Atkins, D. E.: Higher-radix Division Using Estimates of the Divisor and Partial Remainders, IEEE Trans. Comput., Vol. C-17, No. 10, pp. 925-934 (Oct. 1968).
5) Tung, C.: A Division Algorithm for Signed-Digit Arithmetic, IEEE Trans. Comput., Vol. C-17, No. 9, pp. 887-889 (Sep. 1968).
6) Kuninobu, S., Nishiyama, T., Edamatsu, H., Taniguchi, T. and Takagi, N.: Design of High Speed MOS Multiplier and Divider Using Redundant Binary Representation, Proc. 8 th IEEE Symp. on Computer Arithmetic, pp. 80-86 (May 1987).
7) Soderquist, P. and Leeser, M.: Area and Performance Tradeoffs in Floating-Point Division and Square Root Implementations, Technical Report EE-CEG-94-5, School of Engineering, Cornell University (Dec. 1994). (Submitted to ACM Computing Surveys).

(平成7年7月13日受付)

高木　直史（正会員）

1959年生．1983年京都大学大学院工学研究科修士課程情報工学専攻修了．1984年同大工学部助手．1990～91年スタンフォード大学客員研究員．1991年京都大学工学部助教授．1994年名古屋大学工学部助教授，現在に至る．工学博士．論理回路，ハードウェアアルゴリズム，算術演算回路などの研究に従事．本会坂井記念特別賞，日本IBM科学賞，本会論文賞，電子情報通信学会論文賞などを受賞．電子情報通信学会，IEEE各会員．