URR 浮動小数点数演算のための指数仮数高速分離・結合回路方式とその URR プロセッサへの応用

大山光男[†] 浜田穂積[†,*]

数値演算におけるオーバフロー，アンダフローの発生を避けるため，指数部を可変長とした浮動小数点表現が各種提案されている．しかし，指数部可変長の浮動小数点表現では，指数部と仮数部の境界が指数の大きさにより変動し，データフォーマットも複雑となる．このため，指数部固定長の浮動小数点表現に比べて指数仮数の分離結合の手続きが複雑であり，その高速化が高速演算実現のキーポイントである．本論文では，二重指数分割に基づく実数値表現法である URR 浮動小数点数の高速演算を目的とした，指数仮数の高速分離結合回路方式を示す．URR から指数仮数を分離復元するには，まず指数部と仮数部の境界位置を検出してコード化する．次にそれを基に，バレルシフタ，論理演算回路，ビットパターン発生器を制御，指数仮数を切り出して復元する．また，URR の生成では，最初に指数の桁数を調べ，生成する URR の指数部と仮数部の境界位置を決定する．次にそれを基に，前記演算器を制御して指数部と仮数部を生成し，所定の位置に埋め込む．これらの手続きは，ビットパラレルに行うこと，組合せ回路のみで実現することで高速化を図っており，処理のパイプライン化も容易である．そして本回路方式を実際に 64 ビット URR プロセッサの試作に適用，その実現性と有効性を確認した．

A Circuit to Separate and to Connect the Exponent Part and the Mantissa Part for URR Floating Point Arithmetic and its Application to a URR Processor

Mitsuo Ooyama[†] and Hozumi Hamada[†,*]

To overcome overflows and underflows in computation, several new floating-point arithmetics have been proposed. But in these new arithmetics, a boundary between the exponent part and the mantissa part moves according to the exponent value and data formats are usually more complicated. So, fast separation and connection of the exponent and the mantissa is a key technology to achieve fast computation of these new arithmetics. In this paper, a circuit scheme to separate and connect the exponent and the mantissa for URR floating-point arithmetic based on double exponential cut is presented. To separate the exponent from the mantissa, this circuit scheme firstly detects the boundary and the position is encoded to a short code. Then barrel shifters, logical operation circuits, and bit pattern generators controlled by the code separate the exponent and the mantissa from URR. To form URR from the exponent and the mantissa, a number of bits of the exponent is counted and encoded to a short code. The boundary can be settled by this code, and using the same circuits controlled by the code, the exponent part and the mantissa part are generated and combined into URR. To achieve fast execution, these processes are executed by the combination logic circuits in bit paralell manner and meet to pipelined architectures. We applied this circuit scheme to an experimental 64 bits URR processor and verified its realization. Its performance was also evaluated.

1. はじめに

コンピュータによる数値演算におけるオーバフロー，アンダフローの処理は厄介な問題である．オーバフロー，アンダフローが発生した場合，正常に演算を継続するにはスケーリングを行う必要があるが，これがプログラミングを複雑にし，かつ演算の実効速度を低下させる要因となる．このため，実質的にオーバフロー，アンダフローの発生を避けることのできる数値表現法が研究されてきた[1)~4),7)]．これらの表現法は，いずれも指数部可変長の浮動小数点表現であり，指数部を長くすることにより指数部固定長の表現，例えば IEEE-754 規格の表現ではオーバフロー，アンダフローする数でも表現可能である．なかでも二重指数

† (株)日立製作所中央研究所
Central Research Laboratory, Hitachi Ltd.
* 現在 電気通信大学情報工学科
Department of Computer Science, The University of Electro-Communications

分割に基づく浜田の方式（以下 URR (Universal Representation of Real numbers) と呼ぶ）は，オーバフロー，アンダーフローが事実上起こらないことに加えて，表現仕様がデータの長さに依存しない（データ長独立）などの優れた特徴を持つことから注目され，主として表現精度の面から評価が加えられている[4)~7)]．

ところで，これらの新しい表現方式の実用化を図るには，ハードウェア化により実用的な演算速度を実現することが必須である．しかし現在のところ，これらの表現による数値データの高速演算に関する報告は見当たらない．Azmi らは Morris によって提案された，指数部の長さを記述するフィールドを設けることにより指数部を可変長とした TFP (Tapered Floating Point) の演算方式について考察している[8)]．これは，TFP 表現による数を，従来の指数部固定長浮動小数点表現に変換してから演算し，結果を再度 TFP 表現に変換するもので，シフトレジスタを用いたビットシリアル演算を想定しており，高速な演算は難しいと思われる．

筆者らは URR を高速で演算するためのハードウェア方式を検討，試作を行ってきた[9)]．ごく短い語長で可能なテーブルルックアップ方式を除けば，URR を直接演算する方法は知られておらず，演算にあたっては URR から指数，仮数を分離復元し，指数，仮数ごとに演算した後，指数，仮数を結合して URR を生成する必要がある．URR は指数部の長さが可変長であることから，指数，仮数の分離復元と結合は，従来の指数部固定長浮動小数点数に比べて手続きが複雑であり，処理に時間がかかることが予測される．このため，URR 演算の高速化を実現するには，指数，仮数の分離復元と結合の高速化が必須となる．

本論文では，URR 演算高速化実現のキーポイントである指数，仮数の高速分離復元，結合回路方式を提案する．本回路方式は，ビットパラレルに演算を行うこと，組合せ回路のみで実現することにより演算の高速化を目指している．さらに本回路方式を実際に URR プロセッサに適用，評価した結果について報告する．

2. URR 演算の手順

2.1 URR の概要

URR の演算手順の説明に入る前に，URR の表現形式について簡単に説明する．厳密な数学上の定義等は，文献3)を参照されたい．

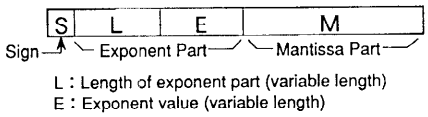


図1 URR フォーマット
Fig. 1 URR data format.

URR は，2重指数分割に基づく実数値表現方式であるが，これを従来行われている符号，指数部，仮数部の連結したものと解釈することもできる．**図1**に URR 表現によるデータフォーマットを示す．URR 表現のデータフォーマット上の特徴は，指数部が可変長であることである．指数部は，指数部の長さを記述するL部と，指数の大きさを表すE部とから成り，指数部自体が自己の長さを記述する機能を持つ．このため，指数部固定長の従来の浮動小数点表現ではオーバフローやアンダフローとなる数値も，指数部を長くすることで表現可能となる．

つぎに URR の表現形式を具体的に説明する．今，実数 x を，

$$x=2^e\times f$$

ただし，e は整数

f は，$x\geqq 0$ のとき $1\leqq f<2$

$x<0$ のとき $-2\leqq f<-1$

とする．このとき f は2進数表現（2の補数）で

$$f=\cdots 01.f_1f_2f_3\cdots f_n \quad (f\geqq 0)$$

$$f=\cdots 10.f_1f_2f_3\cdots f_n \quad (f<0)$$

と表されるので，小数点以下のビット列

$$f_1f_2f_3\cdots f_n$$

を仮数部Mとする．

次に，指数部の表現は，

（1）$x\geqq 0$ のとき

① $e\geqq 0$ で e が2進数表現でちょうど m 桁で表されるとき，

$$e=\cdots 01e_1e_2e_3\cdots e_{m-1}$$

と表せるので，ビット列 $e_1e_2e_3\cdots e_{m-1}$ をE部，その前に指数部の長さを記述するために $m+1$ 個の1と区切りの0を付加してL部，併せて指数部とする．

$$\text{指数部}:\underbrace{111\cdots 10}_{m+1\text{個}}e_1e_2e_3\cdots e_{m-1}$$

（L部 | E部）

② $e<0$ で e が2進数表現でちょうど m 桁で表されるとき，

$$e=\cdots 10e_1e_2e_3\cdots e_{m-1}$$

と表せるので，ビット列 $e_1e_2e_3\cdots e_{m-1}$ をE部，その前

に指数部の長さを記述するために $m+1$ 個の0と区切りの1を付加してL部，併せて指数部とする．

指数部：$\overbrace{000\cdots01}^{m+1\text{個}}e_1e_2e_3\cdots e_{m-1}$

（2）$x<0$ のとき

$x\geqq 0$ の場合と同じ手順でビット列を求め，1の補数（0と1を反転）をとって指数部とする．

以上に求めた仮数部M，指数部L，E，仮数の符号Sを，S，L，E，Mの順に並べることにより図1に示す URR 表現によるデータフォーマットを得る．例えば，正の整数 89 は，

$89=\cdots 01.011001\times 2^{110}$

と表されるので，仮数部Mは 011001 である．また指数部は，$x\geqq 0$，$e\geqq 0$ であり，ちょうど3桁で表されるので，L部は 11110，E部は 10 となる．仮数は正であるので正の整数 89 の URR 表現は，

0111101001100 10…0
S L E M

となる．

2.2 URR 演算の手順

URR を直接演算する方式が現在のところ知られていないので，URR の演算では，**図2**に示すように，指数と仮数を分離復元し，それぞれ演算を行った後，演算結果の指数，仮数から URR を生成するのが一般的手順となる．指数，仮数ごとの演算は，従来の指数部固定長の浮動小数点演算に共通であるから，URR 演算の特徴は，指数，仮数の分離復元と指数，仮数からの URR の生成にある．図1に示すように，URR では，指数部が自己の長さ記述能力を持ち，可変長であることから，L部とE部，E部とM部の境界位置は，指数の桁数により変動する．このため，URR からの指数，仮数の分離復元，および指数，仮数からの URR 生成では，まずそれらの境界位置を見つけることが必要であり，従来の指数部固定長の浮動小数点表現に比べて手続きが複雑となる．したがって，その高速化が URR の高速演算表現のキーポイントであることがわかる．

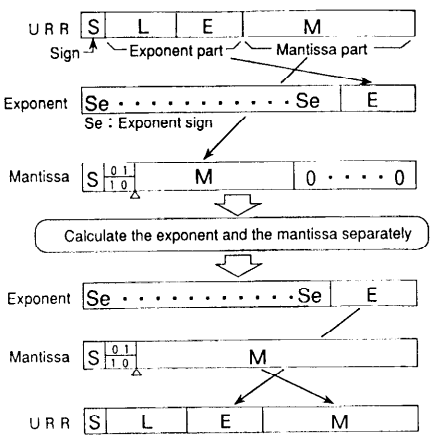


図2 URR の演算手順
Fig. 2 Computation scheme of URR.

以下，第3章では指数と仮数の分離復元の手順の，第4章では指数，仮数の結合の手順の考察を行う．そして，その高速化に必要なハードウェアの機能を明らかにし，実現回路方式を示す．なお，以下の説明では，データ長は一定とし，丸めはハードウェア試作の都合上，RM (Round to Minus infinity, $-\infty$ 方向への丸め）を行うものとする．

3. 指数，仮数の高速分離復元の手順

3.1 指数仮数の分離復元方式

URR から指数仮数を分離復元する手順を**図3**に示し，指数，仮数の分離復元方式を説明する．

① 最初に指数部L，E，および仮数部 M の境界位置を検出し，仮数の符号ビットSとL部を指数の符号ビット Se に変換する．まず，L部の左端のビットを基準に右へビット列を調べ，左端のビットと異なるビットが最初に表れる位置を検出する．これがL部の右端のビットである．L部とE部の境界位置がわかればE部の長さがわかり，E部とM部の境界位置が決定できる．

第2章で述べたように，L部の長さとビットパターンは，仮数の符号S，指数の符号 Se，および指数の桁数で決定されるので，逆にSとL部の左端のビット，およびL部の長さがわかれば，SとL部を指数の符号 Se に変換できる．変換は，仮数の符号SとL部の左端のビット，およびL部の長さから決定するビットパターンと URR の EOR（排他的論理和）演算により行う．

例えば正の整数 89 の URR 表現，

0111101001100 10…0
S L E M

では，S=0，L部の左端のビット=1であるので，指数の符号 Se は0である．また，変換後のE部には，ケチ表現で省略された先頭の1ビットをL部の区切りビットから再生して含めることとし，L部のうち，E部の長さを記述する4個の1を0，区切りビットの0を1に変換するビットパターンは，

0111110……0

と決定され，演算結果は，

000001100110010…0

となる．

② 次に，算術右シフトによりM部をシフトアウトし，E部を右端に揃えて，指数の分離復元を終了する．このとき，必要なシフトビット数は，E部とM部の境界の位置から決定できる．

次に，仮数の分離復元を説明する．

③ まずE部とM部の境界位置から決定するビットパターンとURRのAND（論理積）演算により，指数部（L，E）をゼロにクリアする．

例えば正の整数89のURR表現では，E部は左端から8ビット目までなので，指数部（L，E）をゼロにクリアするビットパターンは，指数部に対応するビットを0，他のビットを1として

100000001……1

演算結果は，

000000000110010…0

となる．

④ 次に，M部の左端ビットが左から4ビット目に位置するように，算術左シフトする．このとき，左から2ビット目，3ビット目に，いわゆるケチ表現で省略されていたビットを挿入する．小数点の位置は，左から3ビット目と4ビットの間にある．以上により仮数の分離復元を完了する．

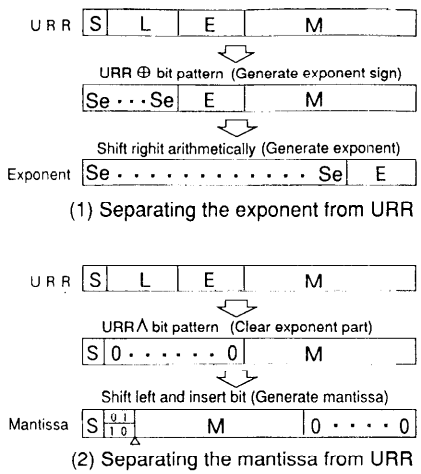


図 3 URR からの指数と仮数の分離
Fig. 3 Separation scheme of the exponent and the mantissa from URR.

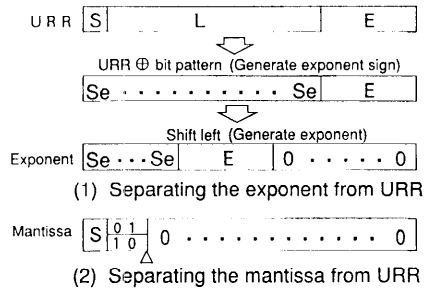


図 4 URR（仮数部なし）からの指数と仮数の分離
Fig. 4 Separation scheme of the exponent and the mantissa from URR with no mantissa part.

以上はURRが仮数部Mをもつ場合で，通常はこの範囲で演算が行われると考えられる．しかし，きわめて大きい，あるいは小さい数であって，一定のデータ長の範囲では仮数部Mが存在せず，さらに指数の大きさを表すビット列Eの下位ビットも存在しない場合も，フォーマットとしては存在するので，演算可能としておきたい．このような場合の演算の手順を**図4**に示す．基本的にはM部が存在する場合と同様な手順で演算可能であるが，指数の分離復元時のシフトが左シフトとなること，仮数として正数は＋1，負数は－2を与えるところが異なる．

次に，以上に説明した方式の高速演算回路の実現について考察する．

3.2 高速分離復元回路方式

以下に高速分離復元に必要な回路機能について列挙する．なお，以下の説明ではデータ長は64ビット一定とする．

（1）指数部，仮数部の境界検出回路

指数部L，Eと仮数部Mの境界位置，指数の長さ記述部Lと指数の値の記述部Eの境界位置は，指数，仮数の分離復元過程における論理演算で使用するビットパターンの決定要素の一つであり，また，シフトビット数の決定要素でもある．したがって，指数，仮数の分離復元にあたっては，最初にそれらの位置を検出せねばならないが，具体的にはL部とE部の境界位置を検出する．これは，サインビットSの右隣のビットを基準として順次右に調べ，最初の反転ビットを検出すればよく，第2章で説明したように，検出した最初の反転ビットがL部の右端のビットである．E部の長さがmビットのときL部の長さは$m+2$ビットあるので，L部とE部の境界位置がわかれば，E部と仮数部Mの境界位置も定まる．

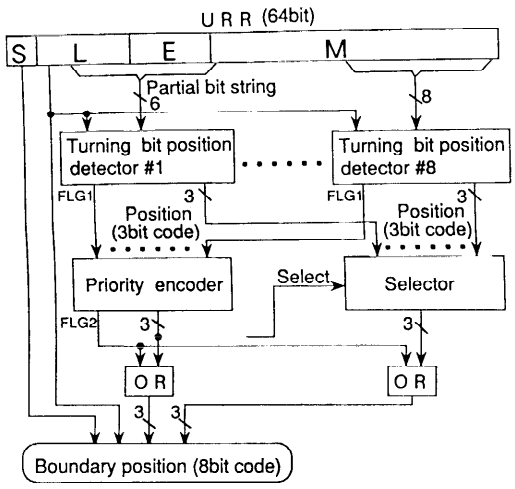


図 5 高速境界位置検出回路
Fig. 5 A circuit for fast detecting and encoding the boundary position.

以上の手順をビットパラレルで高速に行うための具体的な回路を**図5**に示す．この回路は，64ビットURR のL部の右端ビットの位置を検出して6ビットの位置コードを作成し，仮数の符号と指数の符号を決定するために，さらにL部の左端ビットと仮数の符号ビットSを上位に付加，8ビットコードとして出力する．サインビットの右隣のビットを基準として最初の反転ビット位置は，論理としては63ビット入力のプライオリティエンコーダで検出できる．しかし多入力のゲートを必要とすることから，図5に示すように，実際の回路構成ではURRを8個の8ビットの部分ビット列に分割して処理する．ここで，FLG1は入力された部分ビット列内の反転ビットの有無（ありのとき1）を示し，FLG2=1 はプライオリティエンコーダの入力がすべて0であることを示す．これにより，反転ビットが存在しない場合は，OR ゲートを用いて位置コード6ビットを，ビット位置としては存在しない全ビット1として出力し，他のコードと区別する．

表 1 URR ビットパターンと高速境界検出回路の出力コード
Table 1 Input bitpatterns (URR) and output codes of the circuit to detect and encode the boundary position.

URRビットパターン(64bit) (MSB) #30 (LSB)	出力コード (HEX.)	意味
0 0 0 0・・・・・・・・・・・・・・・・0 0	3 F	0（非数）
0 0 0 0・・・・・・・・・・・・・・・・0 1	0 0	＋0（非数）
0 0 0 0・・・・・・・・・・・・・・・0 1 X	0 1	仮数部無し
0 0 0 0・・・・・0 1 X・・・・・・・X	1 E	仮数部無し
0 0 0 0・・・・0 1 X・・・・・・・・X	1 F	仮数部有り
0 0 1 X・・・・・・・・・・・・・・・X	3 D	仮数部有り
0 1 0 X・・・・・・・・・・・・・・・X	7 D	仮数部有り
0 1 1・・・・・1 0 X・・・・・・・・X	5 F	仮数部無し
0 1 1・・・・・・1 0 X・・・・・・・X	5 E	仮数部無し
0 1 1 1・・・・・・・・・・・・・・・1 0	4 0	正数
0 1 1 1・・・・・・・・・・・・・・・1 1	7 F	＋∞（非数）
1 0 0 0・・・・・・・・・・・・・・・0 0	B F	∞（非数）
1 0 0 0・・・・・・・・・・・・・・・0 1	8 0	－∞（非数）
1 0 0 0・・・・・・・・・・・・・・・0 1 X	8 1	仮数部無し
1 0 0 0・・・・・0 1 X・・・・・・・X	9 E	仮数部無し
1 0 0 0・・・・0 1 X・・・・・・・・X	9 F	仮数部有り
1 0 1 X・・・・・・・・・・・・・・・X	B D	仮数部有り
1 1 0 X・・・・・・・・・・・・・・・X	F D	仮数部有り
1 1 1 1・・・・1 0 X・・・・・・・・X	D F	仮数部無し
1 1 1 1・・・・・1 0 X・・・・・・・X	D E	仮数部無し
1 1 1 1・・・・・・・・・・・・・・・1 0	C 0	負数
1 1 1 1・・・・・・・・・・・・・・・1 1	F F	－0（非数）

注） X は Don' t care を表す。

64 ビット URR の入力パターンと図5に示した高速境界検出回路の出力の関係を**表1**に示す．存在するすべてのL部の右端ビットの位置，仮数の符号，指数の符号，さらにURRで定義される6種の非数，すなわち0（これは本来非数とは言えないが，他のものとの兼ね合いで，非数に分類しておくほうが便利なので，ここに含めておく），±0，∞，±∞が8ビットの異なるコードとして出力されるので，これを用いて指数，仮数の分離復元過程における論理演算で使用するビットパターン，およびバレルシフタのシフト方向とシフトビット数を決定できる．

（2）論理演算ビットパターンの発生

指数の分離復元における指数の符号生成，および仮数の分離復元における指数部ゼロクリア時の論理演算で使用するビットパターンを発生する．発生するビットパターンは，指数部，仮数部境界位置検出回路の出力（8ビットコード）から決定できる．

（3）バレルシフタ

指数のビット位置あわせ，仮数のビット位置あわせにおいて，多数ビットの算

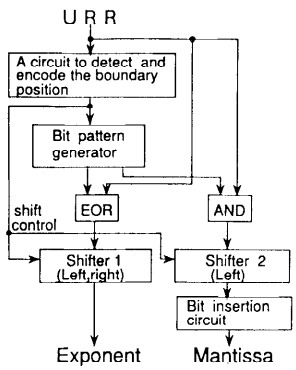


図 6 指数，仮数の高速分離復元回路
Fig. 6 A circuit for fast separation of the exponent and the mantissa from URR.

術右シフト，左シフトを行う．バレルシフタのシフト方向とシフトビット数も指数部，仮数部境界検出回路の出力から決定できる．

上記(1)(2)(3)の回路要素を用いて構成した．指数，仮数高速分離復元回路の構成を**図6**に示す．図6において，シフタ1は指数の桁合わせを，シフタ2は仮数の桁合わせを行い，図3に示した指数，仮数の分離手順をビットパラレルに実行する．

4. 指数，仮数の高速結合

指数と仮数に分離された URR は，指数，仮数ごとに演算し，演算結果の指数と仮数から，再び URR を生成する．そこで，URR 演算の高速化を実現するには，この指数，仮数からの URR 生成も高速化することが必要である．

4.1 指数，仮数の結合方式

図7に指数，仮数からの URR 生成の手順を示し，URR の生成方式を説明する．

① 最初に指数の桁数Eを検出する．その結果，生成する URR 指数部Eの位置と大きさが定まるので，指数を左シフトし，生成する URR の指数部Eの位置に合わせる．このとき，ゼロを右端からシフトインして仮数部Mとなるパートをゼロにクリアしておく．

② 次に，指数の長さ記述部Lを生成し，同時に左端の仮数の符号ビットの位置をゼロにクリアする．これは，指数の符号と桁数，および仮数の符号から決まるビットパターンと，シフト後の指数との EOR 演算により行う．

以上の処理により，指数部LとEが所定の位置に生成され，それ以外の部分のビットはゼロにクリアされる．

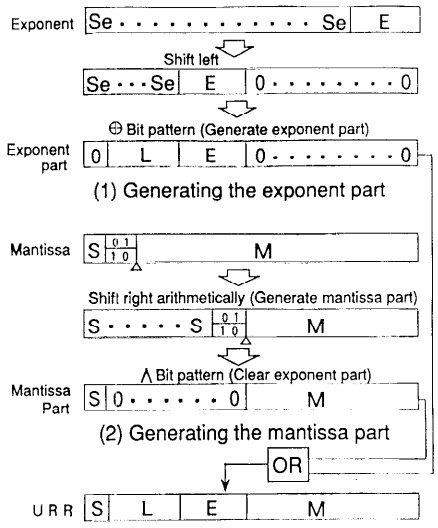


図 7 指数，仮数からの URR の生成
Fig. 7 Generation scheme of URR from the exponent and the mantissa.

次に仮数部を以下の手順で生成する．

③ 仮数を算術右シフトし，仮数部Mを生成するURRの仮数部の位置に合わせる．

④ 次に指数部LとEとなるビットをゼロにクリアする．これは，指数部LとEの位置は，指数の桁数から決定されるので，指数の桁数から定まるビットパター

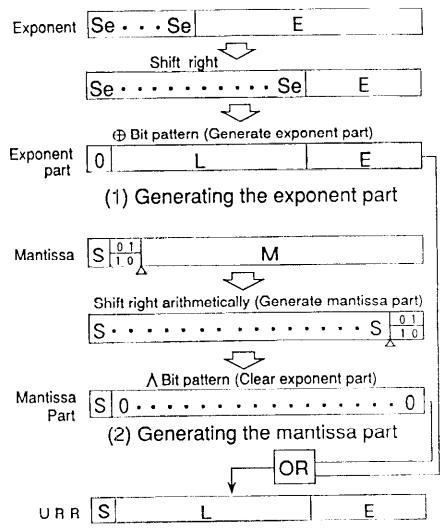


図 8 指数，仮数からの URR（仮数部なし）の生成
Fig. 8 Generation scheme of URR (with no mantissa part) from the exponent and the mantissa.

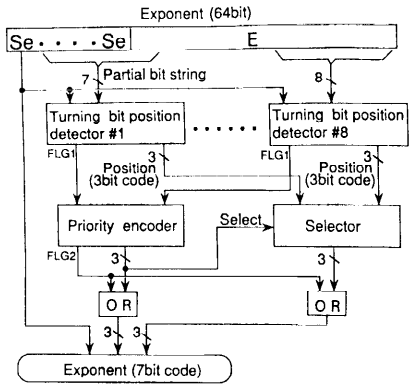


図 9 高速指数桁数検出回路
Fig. 9 A circuit for fast counting and encoding a number of bits of the exponent.

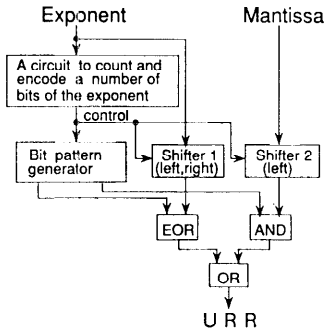


図 10 指数，仮数の高速結合回路
Fig. 10 A circuit for fast generation of URR from the exponent and the mantissa.

ンと，算術右シフトした仮数との AND 演算により行う．

⑤ そして，生成した指数部と仮数部の OR 演算により URR を生成する．

以上は生成する URR が仮数部Mを有する場合についてであるが，演算結果がきわめて大きい，あるいはきわめて小さい数のとき，所定のビット数の範囲に，仮数部Mが存在しなくなる場合がある．このような場合の手順を**図 8** に示す．図 7 に示した仮数部Mが存在する場合の手順に比べて，指数が右算術シフトされること，仮数Mは全ビット右にシフトアウトされることが異なるが，同一手順で URR を生成できる．

次に，以上に説明した方式に基づく URR 高速生成回路の実現について述べる．

4.2 高速結合回路方式

URR の高速生成に必要な回路機能を以下に列挙する．

（1）指数桁数検出

生成する URR の指数部ＬとＥ，および仮数部Mの長さと位置は，第 2 章で述べたように指数の桁数から決定できる．したがって，まず指数の桁数を高速に検出する必要がある．その回路構成を**図 9** に示す．図 9 に示す回路では，64 ビットの指数から指数の大きさを桁数で検出し，指数の符号を左端に加えて 7 ビットのコードとして出力する．回路は図 5 に示した指数仮数境界位置検出回路と基本的に同じ構成で，同様に動作するが，ビット反転の基準ビットが左端の指数符号ビットであること，出力が反転ビットの位置コード 6 ビットの左端に指数の符号 1 ビットを加えた 7 ビットであるところが異なる．本回路の出力である 7 ビットのコードは，URR 生成過程におけるバレルシフタのシフトの方向とシフトビット数，論理演算で使用するビットパターンの決定に用いる．指数桁数検出回路の出力コードと 64 ビット指数のビットパターンの関係を**表 2** に示す．表 2 に示すよう

表 2 指数のビットパターンと指数桁数検出回路の出力コード
Table 2 Input bitpatterns (exponent) and output codes of the circuit to detect and encode a number of bits of the exponent.

指数のビットパターン (64bit) (MSB) #29 (LSB)	出力コード (HEX.)	URR生成時の仮数の有無
0 0 0 0・・・・・・・・・・・・・・・0 0	3 F	仮数部有り
0 0 0 0・・・・・・・・・・・・・・・0 1	0 0	
0 0 0 0・・・・・・・・・・・・・・0 1 X	0 1	
・	・	
0 0 0 0・・・・・・0 1 X・・・・・・X	1 D	
0 0 0 0・・・・・0 1 X・・・・・・・X	1 E	
・	・	
0 0 0 1 X・・・・・・・・・・・・・X	3 C	
0 0 1 X・・・・・・・・・・・・・・X	3 D	非数
0 1 X・・・・・・・・・・・・・・・X	3 E	仮数部無し
1 0 X・・・・・・・・・・・・・・・X	7 E	
1 1 0 X・・・・・・・・・・・・・・X	7 D	
1 1 1 0 X・・・・・・・・・・・・・X	7 C	
・	・	
1 1 1 1・・・・・1 0 X・・・・・・・X	5 E	仮数部有り
1 1 1 1・・・・・・1 0 X・・・・・・X	5 D	
・	・	
1 1 1 1・・・・・・・・・・・・・・1 0	4 0	
1 1 1 1・・・・・・・・・・・・・・1 1	7 F	

注） X は Don't care を表す。

に，出力コードからは，生成するURRの仮数部の有無，非数かどうかについても判定できる．

（2）論理演算ビットパターンの発生

指数部の指数の長さ記述部L部の生成と左端ビット（仮数の符号ビットの位置）のゼロクリア，指数部位置のゼロクリアを行う論理演算で使用するビットパターンを発生する．発生するビットパターンは，指数桁数検出回路の出力コードから決定する．

（3）バレルシフタ

指数のシフトによる指数部の位置合わせ，仮数のシフトによる仮数部の位置合わせを行う．バレルシフタの動作も指数桁数検出回路の出力コードによる制御される．

上記(1)(2)(3)の回路機能を用いて構成した指数，仮数の高速結合回路を**図10**に示す．シフタ1が指数部の位置合わせを，シフタ2が仮数部の位置合わせを行い，図7，図8に示した手順に従って指数と仮数を結合して URR を生成する．そしてこれらの演算は全てビットパラレルに行われる．

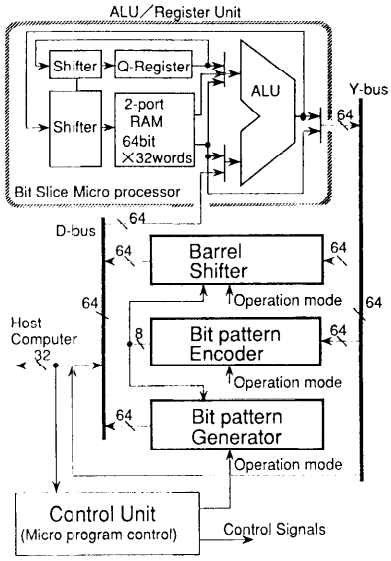


図 11 64ビット URR プロセッサの構成
Fig. 11 Block diagram of a 64 bit URR processor.

5. URR 演算プロセッサへの適用と評価

以上に述べた指数仮数の高速分離復元回路，高速結合回路の実現性と有効性を確認するため，URR プロセッサハードウェアの試作に適用し，評価を行った．

5.1 URR プロセッサのアーキテクチャ

図11に URR プロセッサの構成を示す．試作した URR プロセッサは，64ビット URR の基本的算術演算をビットパラレルに行う．市販のビットスライスマイクロプロセッサを用いてALU/レジスタユニットを構成し，URR 演算に特徴的な指数仮数の分離復元，結合を高速に行うために，ビットパターンエンコーダ，ビットパターンジェネレータ，およびバレルシフタを専用演算器として付加した．この中でビットエンコーダは，指数仮数の境界検出と指数の桁数検出で共用，論理演算は ALU で行い，マイクロプログラム制御により図6，図10の回路

表 3 64ビット URR プロセッサの主要命令実行時間と指数，仮数の分離，結合に要する時間
Table 3 Execution times of major instructions of a 64 bit URR processor and times required for separating the exponent and the mantiss from URR and generating URR from the exponent and the mantissa.

命 令		命令実行時間	指数,仮数の分離,結合に要する時間
ADD (Add)	Rd+Rs → Rd	6.2 μs (31)	3.8 μs (19)
SUB (Subtruct)	Rd−Rs → Rd	6.2 μs (31)	3.8 μs (19)
MLT (Multiply)	Rd×Rs → Rd	18.4 μs (92)	3.8 μs (19)
DIV (Divide)	Rd÷Rs → Rd	18.8 μs (94)	3.8 μs (19)
MLT 2 (Multiply by 2)	Rs×2 → Rd	3.8 μs (19)	2.6 μs (13)
DIV 2 (Divide by 2)	Rs÷2 → Rd	3.8 μs (19)	2.6 μs (13)
MANT (Mantissa)	Rd÷2^n → Rd	3.0 μs (15)	1.2 μs (6)
REM (Remainder)	Rd÷Rs → Rs (整数) 余り(URR) → Rd	16.4 μs (82)	3.8 μs (19)
SCAL (Scale)	Rd×2^n → Rd	4.6 μs (23)	2.6 μs (13)
SCALS (Scale Short)	Rd×2^n → Rd	5.6 μs (28)	2.6 μs (13)
CMP (Compare)	Rd−Rs, set CC	1.0 μs (5)	0 (分離せず)
NEG (Negative move)	−Rs → Rd	0.4 μs (2)	0 (分離せず)

注1） カッコ内はマシンサイクル数．1マシンサイクルは200ns.
注2） Rd : destination register 又はその内容，Rs : source register の内容。
n：整数，CC : condition code

機能を実現している．ただしハードウェア量を抑えるため，各演算では演算器を共用したので指数仮数の分離復元，結合に複数のマシンサイクルが必要となったが，ビットパラレル演算は実現されている．

5.2 動作環境

URR プロセッサはワークステーションのシステムバスに接続された付加装置として動作する．ホストのワークステーションから URR プロセッサのレジスタにデータを書き込んでおいてから起動し，ワークステーションは，演算終了割込みを確認してから演算結果を URR プロセッサのレジスタから読みだす．これらの詳細については文献10)，11)を参照されたい．

5.3 性能評価

表3に主要命令の実行時間と，実行時間に占める指数仮数の分離復元，結合に要する時間を示した．カッコ内はマシンサイクル数である．例えば，加算命令では，命令実行時間 6.2 μs (31 マシンサイクル) のうち，二つの URR から指数，仮数を分離復元するのに要する時間は 2.4 μs (12 マシンサイクル)，演算結果の指数，仮数を結合して URR を生成するのに要する時間は 1.4 μs (7 マシンサイクル) である．これら指数，仮数の分離復元，結合に要するマシンサイクル数は，演算器を共用したため，図6，図10の各演算がそれぞれ1マシンサイクルを要することによる．なお，比較 (compare) 命令，および符号反転 (negative move) 命令では，URR の特徴として，指数仮数に分離することなく直接演算可能であることに注目されたい．

6. おわりに

2重指数分割に基づく数値表現 URR の演算高速化を目的として，URR からの指数と仮数の分離復元，指数と仮数からの URR 生成の高速回路方式を示した．URR の演算高速化を実現するには，指数と仮数の分離復元，結合の高速化がキーポイントとなる．本論文で提案した回路方式は，ビット並列に演算すること，組合せ回路のみで実現することにより高速化を目指すものであり，URR プロセッサの試作に適用して実現性と有効性を確かめた．

今回適用したプロセッサでは，ハードウェア規模の制約から演算器を共用したため，指数，仮数の分離復元，演算結果の指数，仮数からの URR 生成にそれぞれ数マシンサイクルを要している．しかし，LSI 化すれば演算器を専用化することが可能となるので，必要なマシンサイクル数をさらに少なくすることができる．また，パイプライン処理を行えば，演算スループットは指数部固定長の浮動小数点表現に比べても，ほぼ同等の性能が期待できると考えるが，その具体化の検討と定量的評価は今後の課題である．指数，仮数の分離復元，URR の生成の手続きが複雑な分，ハードウェアが増加するのは事実であるが，LSI の集積度は年々大きく向上している．これらの課題は，オーバフロー，アンダフロー対策から開放されるなどの URR のメリットとの兼ね合いから評価されるべきであろう．

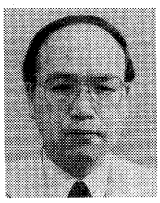
謝辞 本回路方式の評価の機会を与えて頂いた，早稲田大学中島勝也教授，ご討論頂いた東京農工大学高橋延匡教授，元立教大学教授故島内剛一先生，東京農工大学高橋研究室中原雅彦氏（現日立製作所システム開発研究所），日立製作所中央研究所武市宣之主任研究員（現日立情報システムズ開発研究所主管研究員），並びに和田健一主任研究員（現日立製作所システム開発研究所第七部長）に感謝いたします．

参考文献

1) Morris, R.: Tapered Floating Point: A New Floating-Point Representation, *IEEE Trans. on Comput.*, Vol. TC-20, No. 6, pp. 1578-1579 (1971).
2) 松井正一，伊理正夫：あふれのない浮動小数点表示方式，情報処理学会論文誌，Vol. 21, No. 4, pp. 306-313 (1980).
3) 浜田穂積：二重指数分割に基づくデータ長独立実数値表現法Ⅱ，情報処理学会論文誌，Vol. 24, No. 2, pp. 149-156 (1983).
4) 中森真理雄，土井 孝：3重指数分割による数値表現方式について，電子情報通信学会論文誌，Vol. J71-A, No. 7, pp. 1468-1469 (1988).
5) 横尾英俊：自然数の立場から見た多重指数分割浮動小数点表示方式，情報処理学会論文誌，Vol. 30, No. 6, pp. 792-794 (1989).
6) 横尾英俊：自然数の表現に基づく多重指数分割浮動小数点表示方式のクラス，電子情報通信学会論文誌，Vol. J72-A, No. 12, pp. 1998-2004 (1989).
7) 中森真理雄，萩原洋一，高田正之：変動式多重分割による自然数表現法，情報処理学会論文誌，Vol. 31, No. 6, pp. 939-941 (1990).
8) Azmi, A. M. and Lombardi, F.: On a Tapered Floating Point System, *Proc. of IEEE Symposium on Comput. Arith.*, pp. 2-9 (1989).
9) 大山光男，浜田穂積：URR 演算のための指数・仮数高速分離・結合回路方式，電子情報通信学会春季全国大会論文集，分冊6，p. 49 (1989).

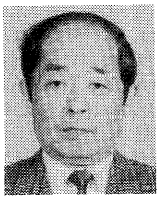
10) 中原雅彦，中川正樹，高橋延匡：URR 浮動小数点演算コプロセッサのアーキテクチャの検討と言語C処理系 cat による利用方式，第36回情報処理学会全国大会論文集，pp. 219-220 (1988).
11) 中原雅彦，早川栄一，岡野裕之，並木美太郎，高橋延匡：複数の浮動小数点表現法を処理するシステム環境の設計と実現，情報処理学会論文誌，Vol. 33, No. 4, pp. 481-490 (1992).

(平成6年1月24日受付)
(平成6年4月21日採録)



大山 光男（正会員）

1972 年名古屋工業大学工学部電子工学科卒業．同年(株)日立製作所中央研究所入社．計算機ハードウェア，アレイ型ディスク記憶装置等の研究開発に従事．現在，同所超高速プロセッサ部主任研究員．電子情報通信学会会員．



浜田 穂積（正会員）

1964 年京都大学工学部数理工学科卒業．同年(株)日立製作所中央研究所入社．1989 年より電気通信大学情報工学科教授．理学博士．計算機アーキテクチャ，数値計算等の研究に従事．著書「PASCAL 入門」等．日本応用数理学会，日本数学会，ACM, IEEE 各会員．