A COMPARATIVE STUDY OF ALGORITHMS FOR MULTIPLE-PRECISION RADIX-CONVERSIONS

By KRIPASINDHU SIKDAR
Indian Statistical Institute

SUMMARY. A survey of the algorithms for converting integers, fractions and floating point numbers from one radix to another is presented in this paper. A new algorithm, called "radix-difference algorithm" is proposed for converting integers as well as fractions.

A comparative evaluation of all these algorithms is made in terms of the number of arithmetic operations, as well as the available arithmetic facility. For this sake, numbers in the signed-magnitude form, as well as in the signed-complementary form are considered and the minor changes needed in the algorithms to suit a given representation are mentioned.

1. Introduction

A number X in a positional number system with (base) radix p is conventionally represented as

$$X = a_n p^n + a_{n-1} p^{n-1} + \ldots + a_0 p^0 + a_{-1} p^{-1} + \ldots + a_{-m} p^{-m} \qquad \ldots \quad (1.1)$$

where $0 \leqslant a_i \leqslant p-1$, for all i, and n and m are positive integers.

It is often required to convert numbers from one radix to another either by built-in algorithms or by an external programme. Several techniques are available in literature (see references at the end of the paper) for converting integers, fractions and floating-point numbers; but as such neither a comprehensive survey nor a comparative evaluation of these methods are available for computer users. It is the object of this paper to fill up these needs. In addition to this a new algorithm, called "radix-difference algorithm", for conversion is suggested here. The survey presented here deals, in particular, with the suitability of each algorithm for a given type of arithmetic with particular emphasis on the economy of the number of arithmetic operations when a computer has a specified arithmetic facility. For this purpose, number representations in signed-magnitude as well as in signed-complementary forms are considered.

2. Conversion algorithms for integers (from radix p to radix q)

Let $\qquad N_p = a_n p^n + \ldots + a_0 p^0$

and $\qquad N_q = b_m q^m + \ldots + b_0 q^0$

be the representations of an integer N in radices p and q respectively.

Two methods are available (Booth and Booth, 1956; Ledley, 1960; Richards, 1960) for this purpose. These are described in Sections 2.1 and 2.2. A new algorithm,

4

called "radix-difference algorithm" is described in Section 2.3. A modification of the method of Section 2.1 is presented in Section 2.4. A comparative evaluation of these different algorithms is given in Section 2.5.

2.1. *Algorithm 1 : Multiplication method (radix-q arithmetic).*

Case (a) $p > q$. In this case one expresses p in radix q in the form,

$$p_q = \sum_{i=0}^{k} d_i q^i \qquad \dots \quad (2.1.1)$$

where $q^k \leqslant p < q^{k+1}$, and the coefficiet a_i as a_{iq} in radix q.

Then N_q is evaluated using the recursive scheme

$$Z_i = Z_{i-1} p_q + a_{(n-i)q}, \qquad \dots \quad (2.1.2)$$

for $i = 1, 2, \dots, n$, with $Z_0 = a_{nq}$ and the recursion terminates at the n-th step yielding $Z_n = N_q$.

Naturally, evaluation of (2.12) demands facilities for expressing a_i initially in radix q as well as multiplication and addition in radix q.

Example (binary arithmetic) : $p = 10$ and $q = 2$. Convert $(256)_{10}$ to binary.

Using (2.1.2), we have,

$$Z_0 = 0010$$

$$Z_1 = (0010 \times 1010) + 0101$$

$$= 11001$$

$$Z_2 = (11001 \times 1010) + 0110$$

$$= (100000000)_2.$$

Case (b) $p < q$. In this case a_i need not be expressed as a_{iq} as they are already valid digits in radix q. So, only the facility for adding and multiplying in radix q is needed to execute the recursive scheme

$$Z_i = Z_{i-1} p_q + a_{n-i} \qquad \dots \quad (2.1.3)$$

for $i = 1, 2, \dots, n$ with $Z_0 = a_n$ and the recursion terminates at the n-th step yielding $Z_n = N_q$.

Example (decimal arithmetic) : $p = 2$ and $q = 10$. Convert $(1101)_2$ to decimal.

Using (2.1.3), we have,

$$Z_0 = 1$$

$$Z_1 = 1 \times 2 + 1 = 3$$

$$Z_2 = 3 \times 2 + 0 = 6$$

$$Z_3 = 6 \times 2 + 1 = (13)_{10}.$$

2.2. *Algorithm 2 : Division method (radix-p arithmetic).*

Case (a) $p > q$. In this case we use the recursion

$$Z_i = Z_{i+1} q + r_i, \quad i = 0, 1, \ldots, m \qquad \ldots \quad (2.2.1)$$

with $Z_0 = N_p$ obtaining a quotient Z_{i+1} and a remainder r_i. The recursion terminates when the quotient $Z_i = 0$. The remainders r_i which are valid digits in radix q form the digits b_i of N_q. These are to be recorded in a reverse manner starting from the final step to the initial step. This method demands division facility in radix p.

Example (decimal arithmetic) : $p = 10$ and $q = 2$. Convert $(285)_{10}$ to binary. Using (2.2.1), we have computations as shown in Table 1.

TABLE 1

i	Z_i	$b_i = r_i$
0	285	1
1	142	0
2	71	1
3	35	1
4	17	1
5	8	0
6	4	0
7	2	0
8	1	1
9	0	—

So, $(285)_{10} = (1000\ 111\ 01)_2$

Case (b) $p < q$. In this case we need to express q in radix p as

$$q_p = \sum_{i=0}^{k} C_i p^i \qquad \ldots \quad (2.2.2)$$

where $p^k \leqslant q < p^{k+1}$, and use the recursion

$$Z_i = Z_{i+1} q_p + r_i, \qquad \ldots \quad (2.2.3)$$

for $i = 0, 1, \ldots, m$ with $Z_0 = N_p$ obtaining a quotient Z_{i+1} and a remainder r_i performing division in radix p. The recursion terminates at $Z_i = 0$. The remainders r_i which are valid digits in radix q are in p-coded form and form the digits b_i of N_q which are obtainable by translating these p-coded digits into radix q.

Note that the division method described here is just a reverse of the multiplication method, considered earlier.

Example (binary arithmetic): $p = 2$, $q = 10$ *and* $q_p = 1010$. Convert $(11011)_2$ to decimal.

Using (2.2.3), computations are given in Table 2.

TABLE 2

i	Z_i	$b_i = r_i$
0	11011	111
1	10	10
2	0	—

So, $(11011)_2 = 10-111$ in binary coded decimal

$= (27)_{10}$

2.3. *Radix-difference algorithms.*

2.3.1. *Case* (a) $p > q$: *Algorithm* 3 (*radix-q arithmetic*). Let $N_p = a_n p^n + \ldots + a_0 p^0$.

Let a_i be coded in radix-q system as a_{iq} and let r be the required number of radix-q digits needed to code radix-p digits. Let X denote N_p thus coded. Let us denote by Y the number formed by the coefficients of X, in the radix q^r viz.

$$Y = a_{nq}(q^r)^n + \ldots + a_{0q}(q^r)^0. \qquad \ldots \quad (2.3.1)$$

Let Z denote the converted number in radix q. Taking Y as the converted number in radix q^r (2.3.1) we overestimate X and hence to obtain Z we must subtract a quantity ΔY from Y, given by,

$$\Delta Y = \{(q^r)^n - p^n\}a_{nq} + \{(q^r)^{n-1} - p^{n-1}\}a_{(n-1)q} + \cdots + \{(q^r)^1 - p^1\}a_{1q}. \qquad \ldots \quad (2.3.2)$$

Factoring out, one obtains from (2.3.2)

$$\begin{aligned} \Delta Y &= (q^r - p)(a_{1q} + p\, a_{2q} + \ldots + p^{n-1} a_{nq}) \\ &\quad + (q^r - p)q^r(a_{2q} + p\, a_{3q} + \ldots + p^{n-2} a_{nq}) \\ &\quad + \ldots + (q^r - p)(q^r)^{n-1} a_{nq}. \end{aligned} \qquad \ldots \quad (2.3.3)$$

To compute ΔY numerically we use radix-q system. It is interesting to note that ΔY can be obtained in a recursive fashion as outlined below.

Let X_i denote the truncated X with truncation performed up to $(n-i)$-th digit. Thus,

$$X_i = \sum_{j=n-i}^{n} a_{jq} p^{j-n+i}, \qquad \ldots \quad (2.3.4)$$

for $i = 0, 1, \ldots, n$.

Let Y_i denote the number formed by the coefficients of X_i in radix q^r. Thus,

$$Y_i = \sum_{j=n-i}^{n} a_{jq}(q^r)^{j-n+i}, \qquad \dots \quad (2.3.5)$$

for $i = 0, 1, \dots, n$.

Let Z_i denote the converted number in radix q equivalent to X_i.

Then,

$$Z_i = Y_i - C_i, \quad \text{for } i = 1, \dots, n$$

where

$$C_i = (q^r - p)\Big[\sum_{j=1}^{i} Z_{i-j}(q^r)^{j-1}\Big], \qquad \dots \quad (2.3.6)$$

which follows from (2.3.2) and

$$Z_0 = a_{nq}.$$

Clearly, Z_n, Y_n and C_n are Z, Y and ΔY respectively.

C_i can be obtained by the recursive relation

$$C_i = q^r C_{i-1} + (q^r - p) Z_{i-1} \qquad \dots \quad (2.3.6)$$

for $i = 1, \dots, n$ with $C_0 = 0$.

Multiplication of C_{i-1} by q^r is equivalent to a shift of C_{i-1} by r digits (in radix q) to the left and multiplication of Z_{i-1} by $(q^r - p)$ can be obtained by shifting to the left by appropriate number of digits and adding. Thus all the computations to obtain Z are performed in radix-q arithmetic. However, initially one needs facilities to code digits in radix p into radix q.

An example is given below to illustrate this algorithm.

Example (binary arithmetic): $p = 10$, $q = 2$, $r = 4$ *and* $q^4 = 16$. Convert $(125)_{10}$ to binary. $(125)_{10} = 0001\ 0010\ 0101$ in binary-coded decimal.

Using Algorithm 3, computations are presented in Table 3.

TABLE 3

i	Y_i	$C_i = 16C_{i-1} + 6Z_{i-1}$	$Z_i = Y_i - C_i$
0	0001	0000	0001
1	0001 0010	0110	0000 1100
2	0001 0010 0101	1010 1000	00000 0111 1101

So, $(125)_{10} = (111\ 1101)_2$.

Notice that in binary form multiplication by 16 is equivalent to a shift by 4 bits to the left and multiplication by 6 is equivalent to two shifts and one addition.

2.3.2. *Case* (b) $p < q$: *Algorithm* 4 (*radix-p arithmetic*). Consider

$$N_p = a_n p^n + \ldots + a_0 p^0.$$

Let r be the maximum positive integer such that $p^r \leqslant q$. Group the radix-p digits $a_n, a_{n-1}, \ldots, a_0$ into m groups of r digits each starting from a_0 and introducing zeros beyond a_n for perfect grouping which is necessary only when $n+1 \neq mr$, for any m. Let $b_0, b_1, \ldots, b_{m-1}$ be the magnitudes of these m groups of radix-p digits considered as valid digits in radix p^r. $b_{m-1}, \ldots, b_0$ are, however, valid p-coded digits in radix q also. So,

$$N_p = \sum_{i=0}^{m-1} b_i (p^r)^i. \quad \ldots \quad (2.3.8)$$

Let

$$Y = \sum_{i=0}^{m-1} b_i q^i \quad \ldots \quad (2.3.9)$$

and Z denote the converted number in radix q. Then, as in the case (a),

$$Z = Y - \Delta Y \quad \ldots \quad (2.3.10)$$

where,

$$\begin{aligned}\Delta Y = & (q-p^r)\{b_1 + p^r b_2 + \ldots + (p^r)^{m-2} b_{m-1}\} \\ & + (q-p^r) q \{b_2 + p^r b_3 + \ldots + (p^r)^{m-3} b_{m-1}\} \\ & + \ldots + (q-p^r) q^{m-2} b_{m-1}. \quad \ldots \quad (2.3.11)\end{aligned}$$

Computation of (2.3.10) is performed in the same fashion as in the case (a) and is outlined below.

Let

$$X_i = \sum_{j=m-1-i}^{m-1} b_j (p^r)^{j-m+i+1} \quad \ldots \quad (2.3.12)$$

$$Y_i = \sum_{j=m-1-i}^{m-1} b_j q^{j-m+i+1} \quad \ldots \quad (2.3.13)$$

Z_i = converted number in radix q equivalent to X_i,

for $i = 0, 1, \ldots, m-1$.

Then,

$$Z_i = Y_i - C_i \quad \ldots \quad (2.3.14)$$

for $Z_0 = b_{m-1}$, where, using (2.3.11),

$$\begin{aligned}C_i &= (q-p^r) \sum_{j=1}^{i} Z_{i-j} q^{j-1} \\ &= qC_{i-1} + (q-p^r) Z_{i-1} \quad \ldots \quad (2.3.15)\end{aligned}$$

for $i = 2, \ldots, m-1$ and

$$C_1 = (q-p^r)Z_0.$$

Multiplication of C_{i-1} by q in radix-q arithmetic can be accomplished by shifts.

Example (decimal arithmetic): $p = 2$, $q = 10$, $r = 3$ *and* $p^r = 8$. Convert $(101\ 011\ 111)_2$ to decimal.

Using (2.3.14) and (2.3.15), we get from Table 4 $(101\ 011\ 111)_2 = (537)_8 = (351)_{10}$

TABLE 4

i	Y_i	$C_i = 100C_{i-1}+2Z_{i-1}$	$Z_i = Y_i - C_i$
0	5	—	5
1	53	10	43
2	537	186	351

Another scheme for conversion when $p < q$ is given in Section 2.4.

2.4. *Algorithm 5 (radix-q arithmetic).* To convert N_p to N_q group radix-p digits into m groups of r digits each as in Algorithm 4 (Section 2.3.2) and then evaluate N_q using the recursive scheme

$$Z_i = Z_{i-1}p^r + b_{m-1-i}, \qquad \ldots \quad (2.3.16)$$

for $i = 1, 2, \ldots, m-1$, with $Z_0 = b_{m-1}$ and the recursion terminates at $(m-1)$-th step yielding $Z_{m-1} = N_q$.

Example (decimal arithmetic): $p = 2$, $q = 10$, $r = 3$ *and* $p^r = 8$. Convert $(101\ 011\ 111)_2$ to decimal.

Using (2.3.16),

$$Z_0 = 5$$

$$Z_1 = 5 \times 8 + 3 = 43$$

$$Z_2 = 43 \times 8 + 7 = (351)_{10}.$$

2.5. *Comparison of algorithms.* In Table 5 is presented a comparison of several algorithms for conversion of integers described earlier. Note that the number of $+$ or $-$ operations, $\times$ or $\div$ operations and shifts needed to carry out a given algorithm is easily obtained from the recursive scheme used.

In case of Algorithm 4 and Algorithm 5 binary digits are to be grouped in groups of 3 bits and each group is to be treated as a BCD digit.

TABLE 5. COMPARISON OF CONVERSION ALGORITHMS OF $(n+1)$ DIGITED INTEGERS FROM DECIMAL TO BINARY AND VICE VERSA

	algorithm	no. of + or − operations	no. of × or ÷ operations	no. of shifts by 1 digit	arithmetic
Case (a) : decimal to binary	1	n	n	—	binary
	2	—	$[(n+1)\log_2 10]+1$*	—	decimal
	3††	$2n-1$	$2n-1$		binary
		$2n-1$**	n**	$4n-4$*	
Case (b) : binary to decimal assume $(n+1) = 3(m+1)$ for some m.	1	n	n	—	decimal
	2	—	$[(n+1)\log_{10} 2]+1$*	—	binary
	4††	$\frac{2}{3}(n+1)-3$	$\frac{2}{3}(n+1)-3$		decimal
		$\frac{2}{3}(n+1)-3$†	$\frac{1}{3}(n+2)-1$†	$\frac{1}{3}(n-5)$†	
	5	$\frac{n+1}{3}$	$\frac{n+1}{3}$	—	decimal

*$[x]$ denotes the lower integral part of x. In general for conversion from radix p to radix $q[(n+1)\log_q p]+1$ × or ÷ operations are needed.

**If multiplication by 10 is effected by shifts.

†If multiplication by 10 is effected by shift.

††Facilities for coding decimal digits to BCD and vice versa are needed.

3. Conversion algorithms for fractions (from radix p to radix q)

Let

$$N_{pf} = a_{-1}p^{-1}+a_{-2}p^{-2}+\ldots+a_{-m}p^{-m} \quad \ldots \quad (3.1)$$

and

$$N_{qf} = b_{-1}q^{-1}+b_{-2}q^{-2}+\ldots+b_{-s}q^{-s} \quad \ldots \quad (3.2)$$

be the representations of a fraction N_f with respect to radices p and q respectively.

It is to be noted that, unlike the case of integers the conversion process for fractions could be non-terminating when the fraction N_{pf} (in radix p) has no exact or finite representation in radix q. Thus, the extent to which the conversion process for fractions has to be carried out is decided by the accuracy requirements.

Two methods (Booth and Booth, 1956; Leadley, 1960; Richards, 1960) are available for this purpose. These are described in Sections 3.1 and 3.2. As in the case of integers (Section 2.3) radix-difference algorithms for conversion of fractions are described in Section 3.3.

A modification of the algorithm of Section 3.1 is presented in Section 3.4. A comparative evaluation of these algorithms is given in Section 3.5.

3.1. *Algorithm 6: Multiplication method (radix-p arithmetic).*

Case (a) $p < q$. Express q as

$$q_p = \sum_{i=0}^{k} d_i p^i \qquad \ldots \quad (3.1.1)$$

where k is an integer such that $p^k \leq q < p^{k+1}$.

The digits $b_{-1}, b_{-2}, \ldots, b_{-s}$ of N_{qf} (3.2) are obtained by the following recursive scheme:

$$f_{-i} q_p = b_{-i} + f_{-(i+1)} \qquad \ldots \quad (3.1.2)$$

for $i = 1, 2, \ldots, s$, where s is decided by the accuracy needed, so that the recursive process is not a finite terminating process, and $b_i = [f_{-i} q_p]$, $[x]$ denoting the lower integral part of the number x, with

$$f_{-1} = N_{pf}.$$

This can be easily verified as follows.

$$N_{pf} \cdot q_p = \left(\sum_{i=0}^{k} d_i p^i \right) \left(\sum_{i=1}^{k} a_{-i} p^{-i} \right)$$

$$= \sum_{i=0}^{l} a_i' p^i + \sum_{i=1}^{m} a_{-i}' p^{-i}, \quad \text{(say)}$$

$$< q \qquad \ldots \quad (3.1.3)$$

and

$$N_{qf} \cdot q = b_{-1} + b_{-2} q^{-1} + \ldots \text{ etc.} \qquad \ldots \quad (3.1.4)$$

From (3.1.3) and (3.1.4), equating the integral parts separately we get,

$$b_{-1} = \sum_{i=0}^{l} a_i p^i \qquad \ldots \quad (3.1.5)$$

$$= \text{Carry past the radix point.}$$

The process can be repeated and the other digits $b_{-2}, b_{-3}, \ldots$, etc. can be similarly obtained.

The digits $b_{-1}, b_{-2}, \ldots$, of N_{qf} are obtained in p-coded form and these must be translated into radix-q digits. The arithmetic is in radix p.

Example (binary arithmetic): $p = 2$, $q = 10$ *and* $q_p = 1010$. Convert $(.110101)_2$ to decimal.

Using (3.1.2), we have computations in Table 6.

TABLE 6

i	f_{-i}	b_{-i}
1	.110101	1000
2	.010010	10
3	.110100	1000
4	.001000	1
5	.010000	10
6	.100000	101
7	.000000	—

Therefore $(.110101)_2 = 1000-0010-1000-0001-0010-0101$ (in binary coded form)

$$= (.828125)_{10}$$

Case (b) $p > q$. The procedure here is similar except that no translation of radix-q digits in radix-p coded form is needed as q is a valid digit in radix p.

Example (decimal arithmetic) : $p = 10$ *and* $q = 2$. Convert $(.825)_{10}$ to binary.

Using (3.1.2), we have computations in Table 7.

TABLE 7

1	.825	1
2	.650	1
3	.300	0
4	.600	1
5	.200	0
6	.400	0
7	.800	1
-	—	-
-	—	-
-	—	-

So, $(.825)_{10} = (.1101001)_2$ upto seven significant bits.

3.2. *Algorithm* 7 : *Division method (radix-q arithmetic).*

Case (a) $p < q$. N_{qf} is obtained from N_{pf} using the following recursive scheme :

$$Z_i = Z_{i-1}p^{-1} + a_{-(n-i)}, \qquad \ldots \quad (3.2.1)$$

for $i = 1, 2, ..., n-1$, with $Z_0 = a_{-n}$ and the recursion terminates at $(n-1)$-th step yielding $Z_{n-1} = N_{qf} \cdot p$ so that

$$N_{qf} = Z_{n-1} \cdot p^{-1}. \quad \dots \quad (3.2.2)$$

Note, however, p^{-1} may not have exact representation in radix q and an approximate value of this is to be used for computation purposes.

Example (decimal arithmetic): $p = 2$ *and* $q = 10$. Convert $(.1101)_2$ to decimal.

Using (3.2.1) and (3.2.2), we have

$$Z_0 = 1$$
$$Z_1 = .5+0 = .5$$
$$Z_2 = .25+1 = 1.25$$
$$Z_3 = .625+1 = 1.625$$

So, $$N_{qf} = (.8125)_{10}.$$

Case (b) $p > q$: Express a_{-i} as a_{-iq} and p as p_q in radix q. Then N_{qf} is obtained by the recursive scheme given by (3.2.1) and (3.2.2).

Example (binary arithmetic): $p = 10$ *and* $q = 2$. Convert $(.875)_{10}$ to binary.

Using (3.2.1) and (3.2.2), we have,

$$Z_0 = 0101$$
$$Z_1 = .1+0111 = 0111.1$$
$$Z_2 = .11+1000 = 1000.1$$

So, $$N_{qf} = 1000.11 \,.\, 1010$$
$$= (.111)_2.$$

3.3. *Radix-difference algorithms.*

3.3.1. *Case* (a) $p > q$: *Algorithm* 8 (*radix-q arithmetic*). Let $X = a_1 p^{-1} + \dots + a_m p^{-m}$ be a fraction in a radix p. To obtain the radix-q equivalent Z of X the following method, which is similar to that of Section 2.3.1, is used. Construct a number Y in radix q^r with the coefficients a_i representing X in q-coded form, denoted by a_{iq}, that is, let

$$Y = \sum_{i=1}^{m} a_{iq} q^{-ir}; \quad \dots \quad (3.3.1)$$

here r denotes the number of radix-q digits needed to code radix-p digits. Thus, as in Section 2.3.1, we can write,

$$Z = Y + \Delta Y, \quad \dots \quad (3.3.2)$$

where,

$$\Delta Y = a_{1q}(p^{-1}-q^{r})+\ldots+a_{mq}(p^{-m}-q^{-mr}). \qquad \ldots \quad (3.3.3)$$

Factoring out, one obtains from (3.3.3),

$$\begin{aligned}\Delta Y = {} & (q^{r}-p)p^{-1}(a_{1q}q^{-r}+a_{2q}q^{-2r}+\ldots+a_{mq}q^{-mr}) \\ & +(q^{r}-p)p^{-2}(a_{2q}q^{-r}+a_{3q}q^{-2r}+\ldots+a_{mq}q^{-(m-1)r}) \\ & +\ldots+(q^{r}-p)p^{-m}a_{mq}q^{-r} \qquad \ldots \quad (3.3.4)\end{aligned}$$

ΔY can be obtained by the following recursive scheme.

Let $$X_i = \sum_{j=0}^{i-1} a_{m-i+j+1}p^{-j-1}, \qquad \ldots \quad (3.3.5)$$

$$Y_i = \sum_{j=0}^{i-1} a_{m-i+j+1}q^{-(j+1)r}, \qquad \ldots \quad (3.3.6)$$

and $Z_i =$ the radix-q equivalent of X_i, for $i = 1, 2, \ldots, m$.

Then $$Z_i = Y_i+\Delta Y_i \qquad \ldots \quad (3.3.7)$$

where,

$$\Delta Y_i = (q^{r}-p)\sum_{j=0}^{i-1} Y_{i-j}p^{-j-1} \qquad \ldots \quad (3.3.8)$$

which follows from (3.3.4).

(3.3.8) can be written in the recursive form

$$\Delta Y_i = p^{-1}\{Y_{i-1}+(q^{r}-p)Y_i\} \qquad \ldots \quad (3.3.9)$$

for $i = 1, 2, \ldots, m$, with $Y_0 = 0$. The recursion terminates at $i = m$ giving $\Delta Y_m = \Delta Y$. Z is then obtained by adding Y_m to Y.

Example (binary arithmetic): $p = 10$ *and* $q = 2$. Convert $(.875)_{10}$ to binary. To use the Algorithm 8 the arithmetic will have to be carried out in binary. However, for the sake of clarity the example is worked out using decimal fractional arithmetic as given in Table 8.

TABLE 8

i	Y_i	Y_i
1	$\frac{5}{16}$	$\frac{6}{10}\cdot\frac{5}{16} = \frac{3}{16}$
2	$\frac{7}{16}+\frac{5}{16^2} = \frac{117}{16^2}$	$\frac{1}{10}\left\{\frac{3}{16}+6.\frac{117}{16^2}\right\} = \frac{75}{256}$
3	$\frac{8}{16}+\frac{117}{16^3} = \frac{2165}{16^3}$	$\frac{1}{10}\left\{\frac{75}{256}+6.\frac{2165}{16^3}\right\} = \frac{1419}{16^3}$

So,
$$Z = \frac{2165}{16^3}+\frac{1914}{16^3}$$

$$= \frac{3584}{16^3} = \frac{7}{8} = (.111)_2$$

3.3.2. *Case (b)* $p < q$: *Algorithm* 9 (*radix-q arithmetic*). Let $X = a_1 p^{-1} + \ldots + a_n p^{-n}$ be a fraction in radix p. To obtain the radix-q equivalent Z of X the following method, which is similar to that of Section 2.3.2, is used.

Let r be the maximum positive integer such that $p^r \leqslant q$. Group the radix-p digits $a_1, a_2, \ldots, a_n$ into m groups of r digits each starting from a_1 and introducing zeros after a_n for perfect grouping which is necessary only when $n \neq mr$, for any m. Let $b_1, \ldots, b_m$ be the magnitudes of these m groups of radix-p digits considered as valid digits in radix p^r. $b_1, \ldots, b_m$ are, however, valid p-coded digits in radix q also. Then X can also be written as

$$X = \sum_{i=1}^{m} b_i p^{-ir}. \qquad \ldots \quad (3.3.10)$$

Let
$$Y = \sum_{i=1}^{m} b_i q^{-i} \qquad \ldots \quad (3.3.11)$$

Thus, as in Section 3.3.1,

$$Z = Y + \Delta Y, \qquad \ldots \quad (3.3.12)$$

where,

$$\Delta Y = (p^{-r} - q^{-1})b_1 + \ldots + (p^{-mr} - q^{-m})b_m. \qquad \ldots \quad (3.3.13)$$

Factoring out, one obtains from (3.3.13),

$$\begin{aligned}\Delta Y = &(q-p^r)p^{-r}(b_1 q^{-1} + \ldots + b_m q^{-m}) \\ &+ (q-p^r)p^{-2r}(b_2 q^{-1} + \ldots + b_m q^{-(m-1)}) \\ &+ \ldots + (q-p^r)p^{-mr} b_m q^{-1}.\end{aligned} \qquad \ldots \quad (3.3.14)$$

As in Section 3.3.1 ΔY is obtained using the recursive scheme

$$\Delta Y_i = p^{-r}\{\Delta Y_{i-1} + (q-p^r)Y_i\}, \qquad \ldots \quad (3.3.15)$$

for $i = 1, 2, \ldots, m$, where Y is given by

$$Y_i = \sum_{j=0}^{i-1} b_{m-i+j+1}\, q^{-j-1} \qquad \ldots \quad (3.3.16)$$

and $\Delta Y_0 = 0$.

The recursion terminates at m-th step yielding $\Delta Y_m = \Delta Y$.

Example (decimal arithmetic) : $p = 2$, $q = 10$ *and* $r = 3$. Convert $(.101\ 110)_2$ to decimal $(.101\ 110)_2 = (.50)_{8}$.

Using (3.3.15) and (3.3.16), we have computations in Table 9.

TABLE 9

i	Y_i	Y_i
1	$\frac{6}{10}$	$\frac{1}{8}.2.\frac{6}{10} = \frac{6}{40}$
2	$\frac{5}{10}+\frac{6}{10^2} = \frac{56}{10^2}$	$\frac{1}{8}\left\{\frac{6}{40}+2.\frac{56}{10^2}\right\} = \frac{127}{800} = .15875$

So,

$$Z = .56+.15875$$
$$= (.71875)_{10}.$$

Another scheme for converting fractions from radix p to radix q, for $p < q$, is given in Section 3.4. This is similar to Algorithm 5.

3.4. *Algorithm* 10 *($p < q$ radix-q arithmetic).* To convert a fraction $X = a_1p^{-1}+\ldots+a_np^{-n}$ in a radix p into its equivalent Z in radix-p digits $a_1, \ldots, a_n$ into m groups of r digits each as in Algorithm 9 (Section 3.3.2) and then evaluate Z using X as the recursive scheme

$$Z_i = Z_{i-1}p^{-r}+b_i \qquad \ldots \quad (3.4.1)$$

for $i = 1, 2, \ldots, m$, with $Z_0 = 0$ and the recursion terminates at m-th step yielding $Z{\cdot}q^r = Z_m$ whence

$$Z = Z_mp^{-r}. \qquad \ldots \quad (3.4.2)$$

Example (decimal arithmetic) : $p = 2$, $q = 10$ *and* $r = 3$. Convert $(.101\ 110)_2$ to decimal $(.101\ 110)_2 = (.56)_8$

Using (3.4.1) and (3.4.2), we have,

$$Z_1 = 5$$

$$Z_2 = \frac{5}{8}+6 = \frac{53}{8} \div 8 = (.71875)_{10}.$$

3.5. *Comparison of the algorithms.* In Table 10 is presented a comparison of the algorithms described in Sections 3.1, 3.2, 3.3 and 3.4. The number of $+$ or $-$operaions, $\times$ or $\div$ operations needed to carry out a given algorithm is easily obtained from the recursive scheme used. It is worth noting that Algorithms 7, 8, 9 and 10 are finite terminating processes although p^{-1} may not have an exact representation in radix q and an approximate value of this is to be used for computation purposes and hence the converted number may not be exact. Algorithm 6, on the other hand, can be carried through any arbitrary large number of steps to a given degree of precision of the converted number.

TABLE 10. COMPARISON OF ALGORITHMS FOR CONVERTING n PRECISION FRACTIONS FROM DECIMAL TO BINARY AND VICE VERSA

	no. of algorithm	no. of + or − operations	no. of × or ÷ operations	arithmetic
Case (a)	6	—	d*	binary
decimal	7	$n-1$	n	decimal
to				
binary	8	n	$2n$	binary
Case (b)	6	—	d*	decimal
binary	7	$n-1$	n	binary
to decimal	9†	$\frac{n}{3}$	$\frac{2n}{3}$	decimal
assume $n = 3m$ for some m	10*	$\frac{n}{3}$	$\frac{n}{3}$	decimal

*d is equal to the number of digits in the converted fraction in radix-q desired.

†Binary digits are grouped in groups of three bits each and each group is treated as BCD digit.

4. Conversion algorithms using radix-complement arithmetic (from radix p to radix q)

4.1. *For integers* (E. V. Krishnamurthy et al, 1963). If a number N_p is in true complementary form in radix p the usual method to get N_q in radix q is to reconvert the number N_p in true form, convert it to radix q and then put it back in the complementary form in radix q.

This would mean that we would have once to go up the characters to find the sign of the string and then come down the characters, recomplement and then perform conversion.

A more economic procedure is to convert the most significant character alone to true form by taking its p's complement, translate into radix q and then evaluate

$$[((\bar{a}_{nq} \cdot p_q - a_{(n-1)q})\, p_q - a_{(n-2)q})p_q - \ldots \text{ etc.}].$$

Using the recursive scheme

$$Z_i = Z_{i-1}p_q - a_{(n-i)q} \qquad \text{for} \quad i = 1, 2, \ldots, n \qquad \ldots \quad (4.1.2)$$

with $Z_0 = \bar{a}_{nq}$ and the recursion terminates at the n-th step yielding $Z_n = N_q$ in true form in radix q. This can be easily proved as outlined below.

Since when N_p is in complementary form represented by

$$N_p = a_n^* p^n + \ldots + a_0^* p^0 \qquad \ldots \quad (4.1.3)$$

it has a value given by

$$N_p = -\{(p-a_n^*-1)p^n + \ldots + (p-a_1^*-1)p^1 + (p-a_0^*)p^0\}$$

or,

$$N_p = -\{(p-a_n^*)p^n+(-p)^n+(p-a_{n-1}^*)p^{n-1}+\ldots+(p-a_0^*)\}$$
$$= -\bar{a}_n^* p^n - a_{n-1}^* p^{n-1} - \ldots - a_0^* p^0\}$$

therefore, $$N_p = -(\bar{a}_n^* p^n - a_{n-1}^* p^{n-1} - \ldots - a_0^* p^0) \quad \ldots(4.1.4)$$

where the bar indicates the complement in radix q. This polynomial is to be evaluated in the usual manner (4.1.2). Then the complement of this result in radix q is taken.

Example (binary arithmetic) : $p = 10$ *and* $q = 2$. Convert $N_{10} = 7^* \; 5^*$ to binary in complementary form. $N_{10} = (-25)_{10} = -(11001)_2$.

Using (4.1.2), we have

$$Z_0 = 1010-111 = 11$$
$$Z_1 = 11\times 1010-101$$
$$= 11001$$

So, $$N_2 = -(11001)_2 = (0^* \; 0^* \; 1^* \; 1^* \; 1^*)_2.$$

4.2. *For fractions.* Conversion algorithm for fractions in the complementary from in radix p to fractions in complementary form in radix q is simpler than that for integers.

Let f_p^* be the true complementary representation of a fraction $-f_p$ in radix p and let f_q^* be the representation of f_p^* in radix q. Then $f_q^* = 1-f_q$, where f_q is the representation of f_p^* in radix q. This is because $f_p^* = (1-f_p)_p$, and also converting both sides to radix q we shall get

$$f_q^* = (1-f_q)_q.$$

Since the true complementary representation for a fraction in any radix is obtained by subtracting the fraction from unity, which has the same representation in all radices, f_q^* is obtained directly from f_p^* using any of the algorithms described in Section 3. Note that this is different from the case of integers.

Example : $p = 10$ *and* $q = 2$. Convert $f_{10}^* = (.7^* \; 5^*)_{10}$ to binary in complement form

$$f_{10}^* = (.7^* \; 5^*)_{10} = (-.25)_{10} = (-.01)_2 = (.1^* \; 1^*)_2$$

and $$(.7^* \; 5^*)_{10} = (.1^* \; 1^*) = f_2^*$$

So, f_2^* can be directly obtained from f_{10}^* in the true complementary binary form.

5. Conversion algorithms for floating point numbers (from radix p to radix q) (Krishnamurthy *et al*, 1963; Mancino, 1966)

There are two different schemes available for converting a floating point number in radix p to a floating point number in radix q. In practice, one can use either of the two schemes or a combination of these. These schemes are described below.

5.1. *Polynomial approximation method.* Let the representations of a number X be $N_p \cdot p^{e_p}$ and $N_q \cdot q^{e_q}$ in radices p and q respectively. Given N_p, e_p expressed in radix p to get N_q and e_q expressed in radix q. N_p can be considered as either a pure integer or a pure fraction with proper exponent e_p. N_q and e_q, such that

$$X = N_p \cdot p^{e_p} = N_q \cdot q^{e_q} \qquad \ldots(5.1.1)$$

is satisfied, can be obtained as follows. Equating $p^{e_p} = q^{e_q^*}$, and taking logarithm to the base q we get,

$$\begin{aligned} e_q^* &= e_p \cdot \log_q p \\ &= l+f \text{ (say)} \end{aligned} \qquad \ldots \quad (5.1.2)$$

where l is an integer and f is a fraction. Take $e_q = l$. N_q can be obtained in any one of the following two ways.

(a) Obtain N_q^* converting N_p to radix q and then get $N_q = N_q^* \cdot q^f$.

(b) Obtain $N_p^* = N_p \cdot q^f$ and then get N_q converting N_p^* to radix q.

Conversion of N_p(or N_p^*) to N_q^*(or N_q) can be performed using suitable algorithms for integral and fractional parts of N_p (or N_p^*) separately. The value of the single constant $\log_q^p$ is to be stored and q^f may be obtained by an appropriate polynomial approximation using stored constants. The arithmetic in obtaining N_q is in radix q (or in radix p) if N_q is obtained using (a) (or (b)). These are illustrated in the following examples.

Examples : Case (a) : $p > q$, $p = 10$ *and* $q = 2$ (*binary arithmetic*). We can approximate the function of 2^f to 8 significant digits by using the following polynomial approximation (Lysternik, Chervonenkis and Yanpolski, 1965).

$$(2^f)^* = \left\{\left[\left(\sum_{k=0}^{4} a_k f^k\right)^2\right]^2\right\}^2, \qquad \ldots \quad (5.1.3)$$

where,

$a_0 = 1$

$a_1 = 0.086\ 643\ 306\ 773$

$a_2 = 0.003\ 753\ 501\ 712$

$a_3 = 0.000\ 108\ 419\ 178\ 11$

$a_4 = 0.000\ 023\ 481\ 769\ 517.$

Thus by using a table with the coefficients $a_0, \ldots, a_4$ in fixed point binary form and the relation

$$N_2 = N_2^* \cdot (2^f)^*, \qquad \ldots \quad (5.1.4)$$

where N_2^* is the binary equivalent of N_{10}, it is possible to obtain N_2 in fixed-point binary form evaluating the polynomial (5.1.3) using binary arithmetic only.

Case (b) : $p < q$, $p = 2$ *and* $q = 10$ (*binary arithmetic*) The function 10^f can be approximated to 8 significant digits (Hastings, Hayaward and Wong, 1955; Lyusternik, Chervonenkis and Yanpolski, 1965) by

$$(10^f)^* = [a_0+a_1f+a_2f^2+\ldots+a_7f^7]^2, \qquad \ldots \quad (5.1.5)$$

where,

$$
\begin{aligned}
a_0 &= 1\\
a_1 &= 1.151 \quad 292 \quad 776 \quad 03\\
a_2 &= 0.662 \quad 730 \quad 884 \quad 29\\
a_3 &= 0.254 \quad 393 \quad 574 \quad 84\\
a_4 &= 0.072 \quad 951 \quad 736 \quad 60\\
a_5 &= 0.017 \quad 421 \quad 119 \quad 88\\
a_6 &= 0.002 \quad 554 \quad 917 \quad 96\\
a_7 &= 0.000 \quad 932 \quad 642 \quad 67
\end{aligned}
$$

So by using a table with the coefficients $a_0, a_1, \ldots a_7$ in fixed-point binary form and the relation

$$N_2^* = N_2 \cdot (10^f)^* \qquad \ldots \quad (5.1.6)$$

it is possible to obtain N_2^* in fixed-point binary form evaluating the polynomial (5.1.5). Then N_{10} can be obtained converting N_2^* using binary arithmetic only.

5.2. *Table look-up method.* The usual method of conversion of a floating-point decimal number to a normalized floating-point binary number and vice versa is often performed by a sub-routine that converts from radix 10 to radix 2 using a table of the powers 10^f and from radix 2 to radix 10 by using a table of the coefficients of a polynomial approximation of $10^x (0 \leqslant x < 1)$ [c.f. (5.1.5)]. Mancino (1966) suggested a modification of these conversion schemes and has shown that conversion in both directions can be performed by using a single small table of the powers 10^i. We briefly describe the modified schemes of Mancino for conversion of a floating point number $d = N_p \cdot p^{e_p}$ in radix p to a normalized floating-point number $b = N_q \cdot q^{e_q}$ in radix q and vice versa using a single small table of the powers p^i and radix-q arithmetic.

5.2.1. *Conversion from radix p to radix q.* By suitably choosing the exponent e_p we can make N_p an integer which is converted into a normalized floating-point number h_q in radix q. Then the final exponent e_p of d is converted into a radix-q integer 2_q. Finally b is obtained

$$b = \begin{cases} h_q \cdot p^{i_q}, & \text{if } i_q \geqslant 0 \\ h_q / p^{|i_q|}, & \text{if } i_q < 0 \end{cases} \qquad \ldots \quad (5.2.1)$$

using normalized floating point radix-q arithmetic- the value of $p^{|i_q|}$ being available in a table in normalized floating point radix-q form.

5.2.2. *Conversion from radix q to radix p.* A radix p fraction f_q and a radix p integer i_q are determined so that

$$N_q \cdot q^{e_q} = f_q \cdot p^{i_q}. \qquad \ldots \quad (5.2.2)$$

Then f_q and i_q are converted into their radix-p equivalents. Mathematically the computation of f_q and i_q is based on two formulas obtained as follows. Let u and v be the integral and fractional parts of the product e_q by $\log_p^q$. Then.

$$N_q \cdot q^{e_q} = N_q \cdot p^v \cdot p^u. \qquad \ldots \quad (5.2.3)$$

Setting $x = v$ and $j = u$ if $n \geqslant 0$ or $x = v+1$ and $j = u-1$ if $n < 0$ (5.2.3) may be written as

$$N_q \cdot q^{e_q} = N_q \cdot p^x \cdot p^j \quad \dots \quad (5.2.4)$$

where $0 \leqslant x < 1$ and j is a relative integer. Comparing (5.2.2) and (5.2.4) one has

$$f_q = N_q \cdot p^x \quad \dots \quad (5.2.5)$$

$$i_q = j. \quad \dots \quad (5.2.6)$$

Usually f_q is obtained from (5.2.5) using a polynomial approximation for p^x, but in Mancino's scheme f_q is obtained as

$$f_q = N_q \cdot q^{e}/p^{i_q}.$$

So.

$$f_q = \begin{cases} b/p^{|i_q|}. & \text{if } e_q > 0 \\ b \cdot p^{|i_q|}. & \text{if } e_q \leqslant 0. \end{cases} \quad \dots \quad (5.2.7)$$

5.23. *Use of single small table of the powers p^i in Mancino's scheme.* As described in sub-Sections 5.2.1 and 5.2.2 conversion in both directions can be performed using a single table of the powers p^i in floating-point radix-q form and using normalized floating point radix-q arithmetic.

Moreover, if l denotes the greatest positive integer such that p^l is expressible exactly as a single precision normalized floating-point radix-q number, the division of $|i_q|$ by l leads to an integral quotient g and to an integral remainder r such that

$$p^{|i_q|} = (p^l)^g \cdot p^r. \quad \dots \quad (5.2.8)$$

Hence the tabulation of the single-precision normalized floating-point radix-q equivalents of $p^1, \dots, p^l$ suffices for the execution of (5.2.1) and (5.2.7). Finally, from (5.2.5)

$$\frac{q}{2} \leqslant f < p. \quad \dots \quad (5.2.9)$$

This implies that the exponent s of f_q can assume only the values 0, 1, ..., t, where t is the minimum integer such that $q^t > p$, so that the mantissa of f_q, left-shifted of s-places, gives f_q in fixed-point radix-q from which can be converted into radix p as N_p.

Acknowledgement

The author wishes to express his heartfelt thanks to Dr. E. V. Krishnamurthy for the many useful discussions with him in the preparation of this survey. He also wishes to thank Dr. J. Roy, Head of the Computer Science Unit, Indian Statistical Institute, for encouragement.

REFERENCES

BOOTH, A. D. and BOOTH, K. H. V. (1956) : *Automatic Digital Calculators*, Butterworths.

DEAN, K. J. (1967) : Binary-decimal encoder using integrated circuits. *Electronic Engineering*, 39, 744-747.

——— (1966) : A binary-decimal converter using integrated circuits. *Electronic Engineering*, 38, 662-664.

FURSOV, K. YA. (1965) : Conversion from one number system to another. *USSR C.M.* and *M.P.*, 5, No. 1, 225-227.

HASTINGS, C., HAYWARD, J. T. and WONG, J. P. (1955) : *Approximations for Digital Computers*, Princeton University Press, Princeton, N. J., 144.

KEYS, D. F. and MOORE, D. P. (1963) 9 Multiple-precision binary to decimal integer conversion using only addition and subtraction. *Comm. ACM.*, 6, No. 8, 439.

KRISHNAMURTHY, E. V., *et al.* (1963) : System, logical and arithmetic organisation of the project computer ISIJU. (Unpublished).

LEDLEY, R. S. (1960) : *Digital Computer and Control Engineering*, McGraw Hill.

LYUSTERNIK, L. A., CHERVONENKIS, O. K. and YANPOLSKI, A. R. (1965) : *Hand Book for Computing Elementary Functions*, Pergamon Press, 42.

MANCINO, O. G. (1966) : Multiple-precision floating-point conversion from decimal to binary and vice versa. *Comm. ACM.*, 9, No. 3, 347-350.

RICHARDS, R. K. (1960) : *Arithmetical Operations in Digital Computers*, D. Von Nostrand.

YARBROUGH, L. D. (1963) : Decimal-to-binary conversion of short fields. *Comm. ACM.*, 6, 63-64.

Paper received : June, 1968.