On Translation Algorithms in Residue Number Systems

DILIP K. BANERJI AND JANUSZ A. BRZOZOWSKI

Abstract—**This paper considers translation problems in residue number systems. The conversion from a fixed-base representation to a residue representation can be done using residue adders only; we show that relatively simple combinational logic can be used to replace one level of residue addition. In the reverse translation problem, we examine the conditions under which base extension can be used to compute the fixed-base digits from a residue code number, and we study the efficiency of the algorithm.**

Index Terms—**Base extension, input translation, modular arithmetic, output translation, residue number systems.**

I. Introduction

In this paper we examine algorithms for converting a fixed-base number to a residue or modular representation, and vice versa. In Section II we consider the problem of translating a base b integer X to a modular representation [1]–[4], with $M=\{m_1, m_2, \cdots, m_n\}$ as the set of moduli. This is called the "input translation" problem. The reverse problem (of translating from a modular representation to a base b representation) called the "output translation" problem is discussed in Section III. The base b digits of X as well as the least nonnegative residues of X with respect to the moduli m_i are assumed to be represented in the binary code. A schematic diagram illustrating both translation problems is shown in Fig. 1. In the figure, we use the example $b=10$, $M=\{2, 3, 5, 7\}$, and $X=197$. In general, we denote the base b digits of X by $(a_{q-1}, a_{q-2}, \cdots, a_0)$ and the residues of X with respect to $m_1, m_2, \cdots, m_n$ by $(x_1, x_2, \cdots, x_n)$. We also use the notation $|X|_m$ to denote the least nonnegative residue of X modulo m.

II. Input Translation

For input translation it is sufficient to consider only one modulus m, since each residue $x=|X|_m$ of a base b number X can be computed independently of the other residues. Let

$$X=\sum_{i=0}^{q-1} a_i b^i$$

and let $c_i=a_i b^i$ for each i. Then

$$x=|X|_m=\left|\sum_{i=0}^{q-1} c_i\right|_m=\left|\sum_{i=0}^{q-1}|c_i|_m\right|_m. \quad (1)$$

Manuscript received June 22, 1970; revised January 31, 1972. This work was supported in part by the National Research Council of Canada under Grant A-1617. Most of this work was performed at the University of Waterloo, Waterloo, Ont., Canada.

D. K. Banerji is with the Department of Computer Science, University of Ottawa, Ottawa, Ont., Canada.

J. A. Brzozowski is with the Department of Applied Analysis and Computer Science, University of Waterloo, Waterloo, Ont., Canada.

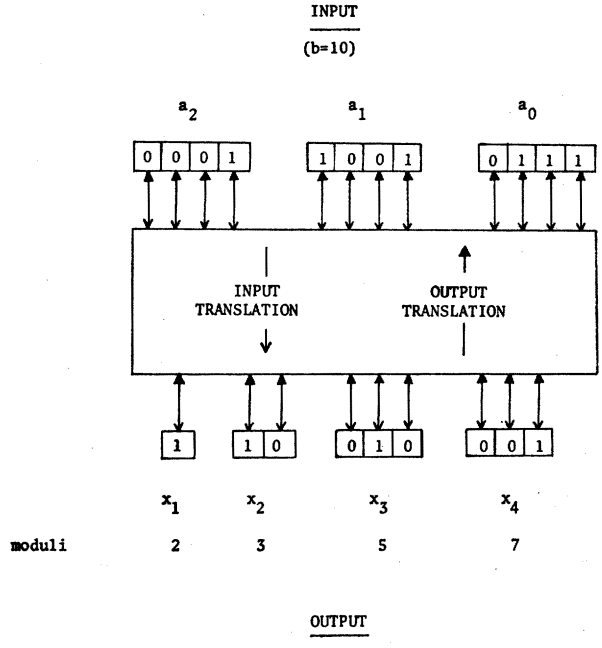

Fig. 1. Illustrating the translation problems.

The use of (1) for input translation has been considered before in [7]. It was shown there that $|c_i|_m$ can be obtained using magnetic matrix circuits; the summation of these terms can then be performed by a mod m adder, yielding x. The magnetic matrix method requires that the base b digits be decoded, in the sense that a binary signal s_v is available for each value v of the digit a_i, such that $s_v=1$ if $a_i=v$ and $s_v=0$, otherwise.

A second method for input translation was also proposed in [7]. This method assumes the availability of special equipment (a circuit translating base b digits to modular form) and is based on the inclusion of a "super modulus." Also the algorithm is sequential in nature, handling one base b digit at a time.

In [6] it was pointed out for the case $b=2$ that the computation of $|c_i|_m$ can be accomplished using only modular adders. Our first remark about the input translation problem is that this method can be easily extended to an arbitrary base b.

Let $\alpha_{i,j}$ be the bits in the representation of a_i, i.e., let

$$a_i=\sum_{j=0}^{k-1}\alpha_{i,j}2^j.$$

Then

$$|c_i|_m=|a_i b^i|_m=\left|\sum_{j=0}^{k-1}|\alpha_{i,j}2^j b^i|_m\right|_m=\left|\sum_{j=0}^{k-1}|d_{i,j}|_m\right|_m$$

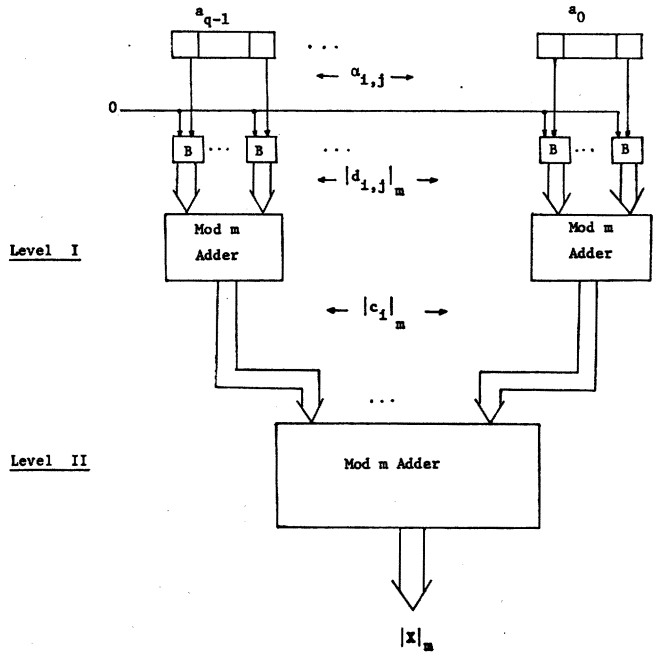

Fig. 2. Computation of $|X|_m$ using mod m adders.

where we have defined $d_{i,j}$ to be $d_{i,j}=\alpha_{i,j}2^jb^i$. This shows that $|c_i|_m$ can be obtained by adding the $|d_{i,j}|_m$ in a k-input mod m adder.

Next consider the computation of $d_{i,j}$. Since $\alpha_{i,j}$ is a binary variable, we have

$$|d_{i,j}|_m = \alpha_{i,j}|2^jb^i|_m.$$

Now for a *fixed* i and j, let the binary representation of $|2^jb^i|_m$ be $(\gamma_{p-1}, \gamma_{p-2}, \cdots, \gamma_0)$. Then the binary representation of $|d_{i,j}|_m$ is $(\alpha_{i,j}\gamma_{p-1}, \alpha_{i,j}\gamma_{p-2}, \cdots, \alpha_{i,j}\gamma_0)$. Thus the binary code for $|d_{i,j}|_m$ can be obtained by simple wire splitting: all the components $\alpha_{i,j}\gamma_r$ for which $\gamma_r=1$ are obtained from $\alpha_{i,j}$ and all others from a constant 0 source.

The method is illustrated in Fig. 2, where the boxes labeled B represent the wire splitting circuits. Note that the mod m adders in level 1 are somewhat special. In a general mod m adder with k inputs there can be m^k different input combinations. However, since the inputs $d_{i,j}$ can take only two values, the number of allowed input combinations is 2^k. This implies the presence of DON'T CARE entries and can lead to simpler logic.

Our second observation about input translation is that combinational logic should be considered for the computation of the bits of $|c_i|_m$ directly from the bits of a_i. In a typical case with $b=10$, each a_i requires four bits. In a modular system it is possible to get a number range exceeding 4.8×10^{12} with 31 as the largest modulus. In this case a maximum of five bits is required to represent $|c_i|_m$. Thus the most complex combinational circuit for finding $|c_i|_m$ has four inputs and five outputs. Note also that a_i and hence $|c_i|_m$ is defined for only 10 out of the 16 possible combinations of the $\alpha_{i,j}$, leaving 6 DON'T CARE entries. Some DON'T CARE entries will always be present, unless $b=2^k$. Table I shows the computation of $|c_2|_{13}$ for the case $b=10$.

TABLE I

a_2	a_210^2	$\lvert a_210^2\rvert_{13}$	a_2 in Binary α_3	α_2	α_1	α_0	$\lvert c_2\rvert_{13}=\lvert a_210^2\rvert_{13}$ in Binary β_3	β_2	β_1	β_0
0	0	0	0	0	0	0	0	0	0	0
1	100	9	0	0	0	1	1	0	0	1
2	200	5	0	0	1	0	0	1	0	1
3	300	1	0	0	1	1	0	0	0	1
4	400	10	0	1	0	0	1	0	1	0
5	500	6	0	1	0	1	0	1	1	0
6	600	2	0	1	1	0	0	0	1	0
7	700	11	0	1	1	1	1	0	1	1
8	800	7	1	0	0	0	0	1	1	1
9	900	3	1	0	0	1	0	0	1	1
			other combinations correspond to DON'T CARE conditions							

TABLE II

Modulus m	Number of Gates (AND/OR) Required to Compute $\lvert c_i\rvert_m=\lvert a_ib^i\rvert_m$	Maximum Fan-In
2, $i\geq0$	nil	nil
3, $i\geq0$	8	4
4, $i\geq0$	nil	nil
5, $i=0$	11	3
$i\geq1$	nil	nil
8, $i\geq0$	nil	nil
9, $i\geq0$	2	2
10, $i\geq0$	nil	nil
13, $i=0$	nil	nil
$i=1$	15	3
$i=2$	13	4
$i=3$	11	3
$i=4$	14	3
16, $i=0$	nil	nil
$i=1$	3	2
$i\geq2$	nil	nil
17, $i=0$	nil	nil
$i=1$	17	4
$i=2$	12	4
$i=3$	19	4
$i=4$	16	3
19, $i=0$	nil	nil
$i=1$	6	3
$i=2$	15	3
$i=3$	17	4
$i=4$	20	4
23, $i=0$	nil	nil
$i=1$	20	4
$i=2$	14	3
$i=3$	15	3
$i=4$	20	4
$i=5$	12	4
$i=6$	10	3
$i=7$	20	4

The relative simplicity of the combinational approach is shown in Table II. Minimal sum-of-products forms have been used to estimate the combinational logic. With shared logic some further reductions are possible but the improvement in most cases is not significant.

Finally, we point out that the complexity of the computation of $|c_i|_m$ and $|X|_m$ depends on the relationship

between b and m. Several special cases are mentioned below.

Case 1: $b=Km+1$, for some $K\geq 1$. In this case the computation of $|c_i|_m$ requires identical circuits for all i, since $|c_i|_m=|a_i|_m$.

Case 2: $b=Km-1$, for some $K\geq 1$. If i is even, then $|c_i|_m=|a_i|_m$; otherwise, $|c_i|_m=|-a_i|_m$. Thus only two types of circuits are needed.

Case 3: $b^i=Km$, $1\leq i<q-1$. Then

$$|X|_m=\left|\sum_{k=0}^{q-1}a_kb^k\right|_m=\left|\sum_{k=0}^{i-1}a_kb^k\right|_m.$$

Thus to find $|X|_m$ only $i<q$ quantities have to be added. Note, in particular, that if $b=Km$, then $|X|_m=|a_0|_m$.

Case 4: $m=2^p$, $p\geq 1$ and $|b^i|_m=2^h$, for some i, $h\geq 0$. Then $|c_i|_m$ does not require any gates.

Proof:

$$|c_i|_m=|a_ib^i|_m=|a_i|b^i|_m|_m=|a_i2^h|_m$$
$$=\left|\sum_{j=0}^{k-1}\alpha_{i,j}2^{j+h}\right|2^p.$$

If $k+h-1\geq p$, some terms in the sum involve 2^{j+h}, where $j+h\geq p$. These terms vanish modulo 2^p and we have

$$|c_i|_m=\left|\sum_{j=0}^{p-1-h}\alpha_{i,j}2^{j+h}\right|2^p=\sum_{j=0}^{p-1-h}\alpha_{i,j}2^{j+h}.$$

Let $(\beta_{i,p-1},\cdots,\beta_{i,0})$ denote the binary representation of $|c_i|_m$. Then

$$\begin{aligned}\beta_{i,p-1}&=\alpha_{i,p-1-h}\\&\cdots\\\beta_{i,h+1}&=\alpha_{i,1}\\\beta_{i,h}&=\alpha_{i,0}\\\beta_{i,h-1}&=0\\&\cdots\\\beta_{i,0}&=0.\end{aligned}$$

If $k+h-1<p$, then

$$|c_i|_m=\left|\sum_{j=0}^{k-1}\alpha_{i,j}2^{j+h}\right|_{2^p}=\sum_{j=0}^{k-1}\alpha_{i,j}2^{j+h}$$

and again no logic is required.

III. Output Translation

We now consider the translation of a number X from its residue representation $(x_1, x_2, \cdots, x_n)$ to its base b representation $(a_{q-1}, a_{q-2}, \cdots, a_0)$. This problem has been considered before in [7]. The method proposed in [7] is based on computing an auxiliary function $A(X)$. The computation of $A(X)$ requires introducing a redundant modulus (called the "super modulus"). In addition to this redundancy, the method of [7] also requires more operations in general than the scheme proposed here.

It is also possible to implement output translation using the mixed-radix conversion process [1], [4]. One can first obtain the mixed-radix representation and then translate it to the base b representation. This means that facility for base b addition and multiplication must be provided just for this purpose. This may not always be desirable in a residue mode machine. Alternatively, the mixed-radix digits can be used to compute the base b digits iteratively, in which case this method offers no special advantage over the iterative scheme proposed in this paper.

The method proposed here is based on the standard algorithm for conversion of an integer from one base to another and requires only the standard residue arithmetic operations. Furthermore, it is pointed out that the number of operations required in computing the successive base b digits can be progressively reduced.

In the standard base conversion algorithm, the base b digits are obtained as follows. For $0\leq j\leq q-1$

$$a_j=|X_j|_b$$

where

$$X_0=X$$

and for $j>0$

$$X_j=\frac{X_{j-1}-|X_{j-1}|_b}{b}.$$

In our case X is assumed to be available only in its residue code. To compute $|X_j|_b$, one can use the method of base extension [3], [4].

The computation of the base b digits from $(x_1, x_2, \cdots, x_n)$ depends on the relationship between b and the moduli m_i. This is discussed case by case now.

Case 1—gcd $(b, m_i)=1$, for all $i=1, 2, \cdots, n$: In this case, letting $m_{n+1}=b$, we can use the base extension process to compute a_0. This requires $2n$ basic operations (subtractions and multiplications).

The first step in computing a_1 is to subtract a_0 from $(x_1, x_2, \cdots, x_n)$. Thus we must obtain $|a_0|_{m_i}$, $i=1, \cdots, n$. For any $m_i>b$, $|a_0|_{m_i}=a_0$ and, therefore, $|a_0|_{m_i}$ can be obtained by simple wire splitting. For any $m_i<b$, $|a_0|_{m_i}$ can be obtained by combinational logic. The next step is division of $X-a_0$ by b. For the case under consideration, $|1/b|_{m_i}$ exists for all $i=1, 2, \cdots, n$. Therefore, division by $|b|_{m_i}$ is equivalent to multiplication by $|b^{-1}|_{m_i}$, where $|b^{-1}|_{m_i}$ denotes the unique multiplicative inverse of b modulo m_i. a_1 is then obtained from $X_1=X-a_0/b$, by base extension. Similarly, other higher order digits can be computed.

Case 2—$b=m_i$ for some i, $1\leq i\leq n$: The computation of a_0 is trivial since $a_0=|X|_b=|X|_{m_i}=x_i$. This reduces the output translation time since, in general, $2n$ basic operations are required for computing a_0. (This case also covers the situation when M contains all factors of b. Let $m_1, m_2, \cdots, m_j$ denote the j moduli that are factors of b. Then the residues $x_1, x_2, \cdots, x_j$ contain all the information required to compute a_0, and the combinational logic involved is rather simple.)

If $b=m_i$, then $|b^{-1}|_{m_i}$ does not exist and it appears that X_1 cannot be computed, since $|X-a_0/b|_{m_i}$ is undefined. However, we can resolve this problem as follows. Since $X<m_1m_2\cdots m_n$, we have

$$\frac{X-a_0}{b}<\frac{m_1\cdots m_im_{i+1}\cdots m_n}{b}$$

or

$$X_1<m_1\cdots m_{i-1}m_{i+1}\cdots m_n.$$

Thus the $|X_1|_{m_j}$, $1\leq j\leq n$, $j\neq i$, uniquely define X_1 and base extension can be used on the $n-1$ moduli $m_1, \cdots, m_{i-1}, m_{i+1}, \cdots, m_n$ to compute a_1.

For computing other higher order digits, a similar process is used.

Case 3—gcd $(b, m_i)\neq 1$ for exactly one modulus m_i, but $b\neq m_i$: In this case both $|1/m_i|_b$ and $|1/b|_{m_i}$ do not exist and the method must be modified. This is because the computation of a_0 by base extension requires the quantity $|1/m_i|_b$. However, we can resolve this problem under certain conditions to be described.

It has been pointed out in [4] that the last step in the base extension process is unnecessary. If we take the multiplication by $|1/m_i|_b$ as the last step of base extension (by relabeling the moduli, if necessary) then a_0 can be computed. This is illustrated by the following example.

Example 1: Let $m_1=3$, $m_2=7$, $m_3=8$, and $(x_1, x_2, x_3)=(2, 5, 7)$. Let $b=10$. Then we compute $a_0=|X|_{10}$ by the following base extension process.

Moduli	3	7	8	10	Mixed-Radix Digits
residues	2	5	7	$\|X\|_{10}$	$r_1=2$
subtract r_1	2	2	2	2	
	0	3	5	$\|\|X\|_{10}+8\|_{10}$	
multiply by $\left\|\frac{1}{3}\right\|_{m_i}$		5	3	7	$r_2=1$
		1	7	$\|7\|X\|_{10}+6\|_{10}$	
subtract r_2		1	1	1	
		0	6	$\|7\|X\|_{10}+5\|_{10}$	
multiply by $\left\|\frac{1}{7}\right\|_{m_i}$			7	3	$r_3=2$
			2	$\|\|X\|_{10}+5\|_{10}$	
subtract r_3			2	2	
			0	$\|\|X\|_{10}+3\|_{10}$	

Using the scheme of [4] we get

$$\|X|_{10}+3|_{10}=0$$

or

$$|X|_{10}=|-3|_{10}=7$$

which is correct since (2, 5, 7) in this residue system corresponds to 47. Thus a_0 can be computed even though gcd $(b, m_3)\neq 1$.

We now examine the conditions under which the above process can be carried out. Base extension as described in [4] requires the extra modulus to be relatively prime with each of the moduli, although this condition is not explicitly stated in [4]. The use of base extension in our example, when gcd $(8, 10)\neq 1$, is still valid for the following reasons.

When the extra modulus m_{n+1} is relatively prime with each of the moduli $m_1, m_2, \cdots, m_n$, the new range of representation becomes 0 to $\prod_{i=1}^{n+1}m_i-1$. However, when m_{n+1} is not relatively prime with one of the moduli, the new range equals lcm $(m_1, \cdots, m_n, m_{n+1})$ [5]. Since $m_1, \cdots, m_n$ are pairwise relatively prime,

$$\text{lcm}\,(m_1, \cdots, m_{n+1})\geq\prod_{i=1}^{n}m_i,$$

and in the mixed-radix expression

$$X=r_{n+1}\prod_{i=1}^{n}m_i+\cdots+r_2m_1+r_1,$$

r_{n+1} is still 0 for any X such that

$$0\leq X\leq\prod_{i=1}^{n}m_i-1.$$

Hence base extension is still valid in this case.

In the present case, computation of higher order digits presents problems unless $b>m_i$. This is because computation of X_1 in residue form requires the quantity $|1/b|_{m_i}$, which is undefined. Hence $|X_1|_{m_i}$ cannot be computed. Base extension cannot be used to compute $|X_1|_{m_i}$ since this requires that

$$X_1<m_1\cdots m_{i-1}m_{i+1}\cdots m_n.$$

In general this condition does not hold for $b<m_i$. However, for $b\geq m_i$, the condition is satisfied and X_1 is uniquely defined by the residues $|X_1|_{m_j}$, $1\leq j\leq n$, $j\neq i$. Hence base extension can be used with $n-1$ moduli $m_1, \cdots, m_{i-1}, m_{i+1}, \cdots, m_n$ in order to compute a_1. By the same reasoning $a_2, a_3, \cdots, a_{q-1}$ can also be computed. Except for the computation of a_0, this case is the same as Case 2, provided the condition $b\geq m_i$ is satisfied.

In using the present output translation scheme, fewer operations are required as the higher order digits are computed. We discuss this reduction case by case in a general system of n moduli $m_1, m_2, \cdots, m_n$.

In Case 1, computation of a_0 requires $2n$ operations. The number of operations required in the computation of subsequent base b digits depends on the relative size of b with respect to the moduli.

If there exist j moduli $m_1, m_2, \cdots, m_j$, which are less than b, then the computation of any a_i, $1\leq i\leq j$, requires $2(n-i)$ operations. This is because

$$X_i<\frac{m_1m_2\cdots m_im_{i+1}\cdots m_n}{b^i}$$

and

$$m_1m_2\cdots m_i<b^i.$$

Hence

$$X_i < m_{i+1} \cdots m_n.$$

Therefore, a_i can be computed by base extension on the $n-i$ moduli $m_{i+1}, \cdots, m_n$ which requires $2(n-i)$ operations. If b is such that $b^2 > m_{j+1}, \cdots, m_n$ (this would be the case if $b=10$ and $m_{j+1}, \cdots, m_n \leq 99$, for example) then, the calculation of a_{j+k} requires a maximum of $2(n-j-[k-1/2])$ operations, for $1 \leq k \leq n-j$.

The same general comments apply to Case 2. a_0 is readily available in this case. If $m_1, m_2, \cdots, m_j < b$, then, in general, the computation of any a_i, $1 \leq i \leq j$, requires $2(n-i)$ operations and any a_{j+k} requires a maximum of $2(n-j-[k/2])$ operations. The computational requirements for Case 3 are the same as in Case 2, except for the computation of a_0 which requires $2n$ operations in Case 3.

Finally, it is to be noted that computing a_{q-1} from X_{q-1} does not require any operations. This is because

$$X_{q-1} < \frac{m_1 m_2 \cdots m_n}{b^{q-1}}$$

and, since the base b representation of X consists of q digits

$$m_1 m_2 \cdots m_n < b^q.$$

Hence

$$X_{q-1} < \frac{b^q}{b^{q-1}} \quad \text{or} \quad X_{q-1} < b.$$

Therefore,

$$a_{q-1} = \left| X_{q-1} \right|_b = X_{q-1}.$$

We now consider an example to illustrate how the number of operations required in computing the higher order base b digits decreases progressively.

TABLE III

Base b Digits a_i	Number of Operations for Computing a_i from X_i
a_0	nil
a_1	8
a_2	8
a_3	6
a_4	4
a_5	nil
Total	26

Example 2: Consider the set of moduli $\{10, 13, 17, 19, 23\}$ and $b=10$. Then $q=6$, i.e., we must compute a_0, a_1, a_2, a_3, a_4, and a_5 from a given residue representation $(x_1, x_2, x_3, x_4, x_5)$. Table III lists the number of operations required for computing these digits from X_i. The total number of operations required for output translation in this system is 36. (This includes the computation of all X_i.)

Acknowledgment

The authors wish to thank the referees for their useful comments and suggestions regarding this work.

References

[1] H. L. Garner, "The residue number system," *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 140–147, June 1959.
[2] I. Flores, *The Logic of Computer Arithmetic.* Englewood Cliffs, N. J.: Prentice-Hall, 1963.
[3] A. Svoboda, "Computer progress in Czechoslovakia II. The numerical system of residual classes," in *Digital Information Processors*, W. Hoffmann, Ed. New York: Wiley, 1962.
[4] N. S. Szabo and R. I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology.* New York: McGraw-Hill, 1967.
[5] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers.* New York: Oxford, 1954.
[6] D. K. Banerji and J. A. Brzozowski, "Sign detection in residue number systems," *IEEE Trans. Comput.*, vol. C-18, pp. 313–320, Apr. 1969.
[7] H. Aiken and W. Semon, "Advanced digital computer logic," Comput. Lab., Harvard Univ., Cambridge, Mass., Rep. WADC TR-59-472, July 1959.