DIGITAL COMPUTER ARITHMETIC: A UNIFIED ALGORITHMIC SPECIFICATION

Algirdas Avižienis
UCLA Computer Science Department, University of California, Los Angeles, CA

The paper presents a method for the description of digital-computer arithmetic in terms of a set of primitive *digit algorithms*. Digit algorithms are specified as arithmetic operations on the values of individual digits of *digit vectors* which represent numerical values in the computer. Complete arithmetic operations in digital processors are described as *vector algorithms* which are composed from digit algorithms.

I. INTRODUCTION

A. Objectives of the Specification

The field of digital-computer arithmetic has evolved as an aspect of the logic design and system architecture of digital computers. This heuristic development has been of practical rather than theoretical nature. Consequently, digital arithmetic has not been recognized as a separate aspect of the theory of computing. The principal difficulty in the presentation of digital arithmetic and in its evaluation has been the lack of a clear distinction between the arithmetic algorithm and its logic implementation. The numerous details of logic design which are included in the description of a given algorithm severely obscure its arithmetical properties. (For an illustration, see [1].) The assessment of the originality of an algorithm and the measurement of its relative efficiency (in terms of speed and cost) with respect to other algorithms for the same arithmetical operation become difficult tasks. The comparison is usually limited to only one class of logic elements.

More recently, initial steps have been taken in the theoretical evaluation of arithmetic algorithms [2, 3, 4]. The general properties of arithmetic algorithms are being identified, and there is a need to formulate digital-computer arithmetic in terms which will provide a unified viewpoint for both the theoretician and the system designer. The benefits of such an algorithmic formulation are expected to be:

(1) A simplification of the description of arithmetic processors
(2) A foundation for the comparison and evaluation of arithmetical algorithms
(3) The separation of a discipline of "arithmetic design" from logic design.

Presented at the Symposium on Computers and Automata
Polytechnic Institute of Brooklyn, April 13–15, 1971.

The objective of this paper is to formulate a methodology for the description of algorithms for digital arithmetic [5]. The goals listed above have been considered as objectives of the formulation. The contributions of Markov [6], Iverson [7], Patterson [8], Robertson [9], and Garner [10] have been important stimuli in this effort. Many aspects of Iverson's notation [7] are used in the subsequent sections.

B. The User's View of Arithmetic Processors

A digital arithmetic processor can be described from two different viewpoints: the viewpoint of the *user*, and the viewpoint of the *designer*. The user sees the arithmetic processor as a "black box" shown in Figure 1. It is described in terms of its inputs, outputs, and operation times.

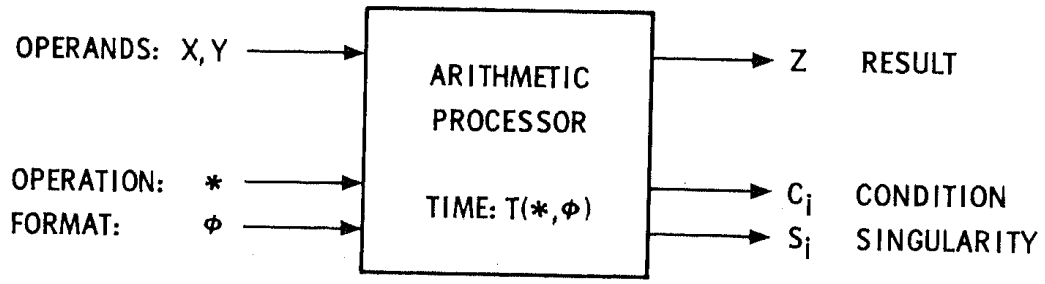

Fig. 1. User's specification of an arithmetic processor.

(1) *The inputs* to the processor are:
  (a) One or more *numerical values* $X, Y, Z, \cdots$
  (b) An *operation symbol* $*$
  (c) A *format symbol* ϕ

The input numeric values (*operands*) are characterized by three properties:

(a.1) They are members of a *finite set* $\mathfrak{N}$ of numerical values with a specified *range*

$$N_{\min} \leq X \leq N_{\max} ,$$

(a.2) They are known within a specified *precision*

$$X - \Delta X_L \leq X \leq X + \Delta X_H ,$$

(a.3) They are stated by means of a defined *number representation*, consisting of an n-tuple of symbols

$$x_{n-1}, \cdots, x_i, \cdots, x_0 ,$$

and a set of rules for the interpretation of this n-tuple as the numerical value X.

The *operation symbols* represent a set of defined arithmetic operations. The set $\{+, -, \times, \div\}$ is commonly available; however, both more extensive and more limited sets may be encountered. When more than one *format* (type of representation) is allowable for the input operands, the operation symbol is supplemented by a *format symbol* ϕ.

(2) *The outputs* of the processor are:
   (a) One or more *numerical results* $Z, Y, X, \cdots$
   (b) One or more condition symbols $C_0, C_1, C_2, \cdots$
   (c) One or more *singularity symbols* $S_0, S_1, S_2, \cdots$
   The *numerical output results* possess similar properties as the operands (range, precision, and representation); however, they may differ from those of the operands in details.

   The *condition symbols* identify specific conditions associated with the value of the result, such as whether the result is negative, positive, zero, etc.

   The *singularity symbols* are issued when the input operands and operation symbol do not yield a "legal" result. The singularity symbols may include:
   (c.1) "Overflow," i.e., the result value is not representable in the range allowed for output values
   (c.2) Excessive loss of precision (significant digits) in floating-point operations
   (c.3) Error caused by a logic fault in the hardware of the processor.
   A *pseudo-result* $Z(S_i)$ instead of the expected result Z appears at the output together with the singularity symbol S_i.

(3) An *operation time* $T(*)$ is specified for every allowable operation symbol of the processor. When more than one format is allowed, separate operation times $T(*, \phi)$ are specified for every format ϕ. Operation times may be either *fixed*, or *variable* over a specified range $T(*)_{\min} \leq T(*) \leq T(*)_{\max}$.

Here we note two key aspects of the user's viewpoint:

(1) The definition of an "arithmetic processor" includes a broad spectrum of "black boxes," from a simple counter with "add one" as the only operation, to a processor which computes trigonometric functions or the fast Fourier transform
(2) The internal structure of the arithmetic processor is specified only in terms of the execution time associated with the operation symbol $*$ and the format of the operands. It must be emphasized that while Fig. 1 is given in terms of two operands and one result, the variations of single-operand, multiple-operand, and multiple-result arithmetic operations are also possible and fit equally well within the given specification.

C. The Designer's View of Arithmetic Processors

The detailed specification of the internal structure of the processor is the responsibility of the designer. The designer's viewpoint must consider both the detailed arithmetic algorithms and the logic structure of the processor. At this point we establish a distinction between arithmetic design and logic design.

Arithmetic design begins with the overall system specifications of the processor (as given in the "user's viewpoint") and translates them into detailed specification of arithmetic operations at the level of individual digits of the number representation. These digit-level algorithms serve as a complete truth-table

specification for the *logic design*. The logic designer specifies the logic nets using the currently preferred logic elements and design techniques, for example, medium-scale or large-scale integrated circuits.

The principal advantage of separating arithmetic design is that it makes the description of algorithms separate and independent of any given set of logic building blocks.

The designer's "black-box" specification of an arithmetic processor is obtained by making the following changes in the user's specification of Figure 1:

(1) *The inputs* are operand-digit vectors

$$\mathbf{x} \equiv \mathbf{x}_{n-1}, \cdots, \mathbf{x}_i, \cdots, \mathbf{x}_0 \qquad \text{(representing } X\text{)} ,$$

$$\mathbf{y} \equiv \mathbf{y}_{m-1}, \cdots, \mathbf{y}_i, \cdots, \mathbf{y}_0 \qquad \text{(representing } Y\text{)} ,$$

and an (operation-and-format) code, abbreviated as "o&f code," vector

$$\Omega \equiv \Omega_t, \cdots, \Omega_i, \cdots, \Omega_0 ,$$

representing the operation and format symbols $*$ and ϕ.

(2) *The outputs* are the result-digit vector

$$\mathbf{z} \equiv \mathbf{z}_{k-1}, \cdots, \mathbf{z}_i, \cdots, \mathbf{z}_0 \qquad \text{(representing } Z\text{)} ,$$

a condition-code vector

$$\varepsilon \equiv \varepsilon_q, \cdots, \varepsilon_i, \cdots, \varepsilon_0 \qquad \text{(representing } C_i\text{)} ,$$

and a singularity-code vector

$$\sigma \equiv \sigma_p, \cdots, \sigma_i, \cdots, \sigma_0 \qquad \text{(representing } S_i\text{)} .$$

(3) The *objective of arithmetic design* is to develop for every o&f-code vector Ω a *vector algorithm* A (Ω) which specifies the value of every result digit $\mathbf{z}_i$ by one of two methods:

(1) Directly

(2) As an arithmetic function of operand digits $(\mathbf{x}_j, \mathbf{y}_j)$ from one or more positions j of the operand digit vectors $\mathbf{x}$ and $\mathbf{y}$.

The values of digits $\mathbf{z}_i$ must be such that the result numerical value $Z \equiv X * Y$ is represented by the digit vector $\mathbf{z}$ in the specified number representation system. If the representation is not possible due to range limits or other constraints, the corresponding singularity-code vector σ is to be generated, and the vector $\mathbf{z}$ must represent the corresponding pseudo-result $Z(S_i)$. Finally, the condition code vector ε must be specified in terms of the result digits $\mathbf{z}_i$ to represent the required condition symbol C_i of the result value Z.

II. DIGITAL REPRESENTATION OF NUMERICAL VALUES

A. Digit Vectors

Numerical values in a digital arithmetic processor are represented by an n-tuple $\mathbf{x}$ of symbols $\mathbf{x}_i$, called *digits*

$$\mathbf{x} \equiv \mathbf{x}_a, \cdots, \mathbf{x}_i, \cdots, \mathbf{x}_b .$$

The n-tuple $\mathbf{x}$ will be called a *digit vector* (abbreviated "*d vector*") in the subsequent discussion. The *length* n of the d vector is the count of its components, designated by $\rho\ \mathbf{x}$

$$\rho\ \mathbf{x} \equiv |a - b| + 1 \quad .$$

In addition to its length, a d vector has two other explicit properties: the indexing system and the range of values for each digit.

The first explicit property of a d vector is the indexing employed to identify its component digits

$$\mathbf{x} \equiv \mathbf{x}_a, \mathbf{x}_{a\pm1}, \cdots, \mathbf{x}_i, \cdots, \mathbf{x}_{b\mp1}, \mathbf{x}_b \quad .$$

The index set of $\mathbf{x}$ consists of consecutive integers

$$\{a, a \pm 1, \cdots, i, \cdots, b \mp 1, b\} \quad .$$

The index set is described by its *origin*, defined as the lesser of the two end values

$$\text{origin} \equiv \min(a, b) \quad ,$$

and by the direction (with respect to the origin) in which the indices increase: *leftward* or *rightward*. For example, $\{3, 2, 1, 0\}$ is zero-origin leftward indexing, while $\{1, 2, 3, 4\}$ is one-origin rightward, and $\{2, 1, 0, -1, -2, -3\}$ is minus-three-origin leftward indexing. Several index origins and both directions have been used in the practice of computer design and description, following the author's preference. The main need in indexing is consistency in adhering to one indexing system. Inconsistency in indexing still occurs in practice and leads to severe difficulties in understanding an algorithm or a design. Zero-origin leftward indexing $\{n - 1, \cdots, i, \cdots, 0\}$ will be used throughout this discussion unless explicitly otherwise specified.

The second explicit property of a d vector is the range of values of its digits. The digit $\mathbf{x}_i$ is a variable ranging over a set of allowed values $\Sigma_i \equiv \{a, b, c, \cdots, k\}$. In cases of practical interest this set consists of two or more consecutive integers, for example

$$\{0, 1\}; \quad \{\overline{1}, 0, 1\}; \quad \{0, 1, 2, \cdots, 9\} \quad , \text{ etc.}$$

The *overbar* (e.g., $\overline{1}$, $\overline{9}$, $\overline{14}$) will be used to designate negative-digit values. Digit values greater than 9 will always be given in the conventional decimal (radix-10) notation. The distinction between canonical, noncanonical, and redundant sets of digit values is discussed in the next section.

B. Number-Representation Systems

A *number-representation system* is specified by a rule which associates one numerical value X with every "legal" d vector $\mathbf{x}$. (All digits of a "legal" d vector assume only allowed values.) When the value X is different for every allowed d vector $\mathbf{x}$, the number system is *nonredundant*. When two or more distinct d vectors represent the same value X, the number system is *redundant*. For example, the d vectors

$$\mathbf{x} \equiv 0, 1, 1, 1 \quad \text{and} \quad \mathbf{y} \equiv 1, 0, 0, \overline{1}$$

both represent the value "seven" in the radix-2 system with allowed digit values $\{\bar{1}, 0, 1\}$. Redundancy of representation has not been useful in manual arithmetic; however, redundant number systems have been found to possess desirable attributes in automatic digital-computer arithmetic.

First, we assume a zero-origin leftward indexing of the d vector $\mathbf{x}$ and consider the explicit representation of numerical values by digit vectors.

The most commonly-used number representation systems are *weighted* number systems in which an *explicit value* x is associated with the d vector $\mathbf{x}$ according to the rule

$$x \equiv \sum_{i \equiv 0}^{n-1} \mathbf{x}_i \times \mathbf{w}_i \quad , \quad \text{or} \quad x \equiv +/\mathbf{x} \times \mathbf{w} \quad ,$$

where $\mathbf{w}$ is the *weight-vector* associated with the d vector $\mathbf{x}$. The weight vector is obtained from the *radix vector* $\mathbf{r}$, which specifies a radix $\mathbf{r}_i$ for every component $\mathbf{x}_i$ of the d vector $\mathbf{x}$. The components of the weight vector are defined as follows

$$\mathbf{w}_0 \equiv 1 \quad ,$$

$$\mathbf{w}_i \equiv \mathbf{w}_{i-1} \times \mathbf{r}_{i-1} \qquad (\text{for } 1 \leq i \leq n-1) \quad .$$

The magnitude $|\mathbf{r}_i|$ of the i^{th} radix in a nonredundant number system is an integer equal to the number of allowed values of the digit $\mathbf{x}_i$.

In a *fixed-radix* number system all components of the radix vector $\mathbf{r}$ have the same value, such as 10, 2, or $\bar{2}$. In a *mixed-radix* number system more than one radix value occurs in $\mathbf{r}$. Representation of elapsed-time values in terms of weeks, days, hours, minutes, and seconds is a system which employs the mixed-radix vector

$$\mathbf{r} \equiv 10, 10, 7, 24, 60, 60 \quad ,$$

when the range of 100 weeks is chosen for the system.

Conventional number systems are fixed-radix systems with the same positive integer radix $\mathbf{r}_i \geq 2$ for all digits $(0 \leq i \leq n-1)$, and with the *canonical* set of digit values

$$\Sigma_i \equiv \{0, 1, 2, \cdots, \mathbf{r}_i - 1\} \qquad (\text{for } 0 \leq i \leq n-1) \quad ,$$

In design practice attention has been focused on conventional systems with radices 10, 2, 4, 8, and 16. Practical applications have been found for *noncanonical* digit sets such as the *nonredundant* radix-10 set

$$\{\bar{4}, \bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3, 4, 5\} \quad ,$$

in decimal multiplier recoding, as well as the several *redundant* sets, such as the radix-2 sets

$$\{\bar{1}, 0, 1\} \quad \text{and} \quad \{0, 1, 2\} \quad ,$$

in fast binary arithmetic [10] and the radix-10 and radix-16 sets, respectively

$$\{\bar{6}, \cdots, 0, \cdots, 6\} \quad \text{and} \quad \{\bar{9}, \cdots, 0, \cdots, 9\} \quad ,$$

in signed-digit arithmetic [11].

Nonconventional weighted number systems with negative radices [12], and complex radices [13], have been postulated and investigated, but thus far they have not found wider acceptance by practicing designers.

Nonweighted number systems are those in which the relative position i of the digit $\mathbf{x}_i$ does not convey "weighting" information (other than the value of the radix $\mathbf{r}_i$) in the expression for the explicit value x being represented. The interchange of any two digits (together with their associated radices and sets of allowed values) will not change the value x. The best known example is the residue number systems (RNS) [14], in which the radices $\mathbf{r}_i$ are mutually pairwise-prime positive integers (7, 9, 11) and the digit values which represent the explicit value x are

$$\mathbf{x}_i \equiv \mathbf{r}_i \mid x \quad ,$$

where $\mathbf{r}_i \mid x$ means "the modulo $\mathbf{r}_i$ residue of x." The "most-significant digit" and "the least-significant digit" do not exist in a nonweighted representation. The rule for the explicit value x of a RNS d vector can be considered to be that of a weighted number representation under a generalization defined by Garner [10].

C. Implicit Values of Single *d* Vectors

The preceding discussion dealt exclusively with the *explicit value* x of the d vector $\mathbf{x}$. The explicit value x is a function of the digit values $\mathbf{x}_i$, radices $\mathbf{r}_i$, and indices i of the d vector under the assumption of *normal* (zero-origin leftward) indexing. The explicit values are either integers (assuming that fractional radices and digit values are excluded), or natural numbers when negative radices and digit values are not allowed (e.g., in conventional number systems). The explicit values cover a finite *explicit range* of the number system represented by the vectors $\mathbf{x}$, $\mathbf{r}$, and Σ.

Explicit values x provide the most direct interpretation of the d vector $\mathbf{x}$ with the associated vectors $\mathbf{r}$ and Σ. However, a further interpretation of x is frequently specified to describe the numerical value X (called the "*implicit value*" of $\mathbf{x}$) which the user wishes to represent by means of the d vector $\mathbf{x}$.

The *implicit value* X of the d vector $\mathbf{x}$ is an arithmetic function of the explicit value x which is defined by further interpretations attached to the d vector $\mathbf{x}$. The major implied attributes of $\mathbf{x}$ which are used to define the implicit value X are:

(a) The location of the radix point
(b) The representation of negative values
(c) The presence of nonsignificant digits in $\mathbf{x}$ [15]
(d) Encoding in an error-detecting or error-correcting code [16].

The variety of possible implicit values of a radix-2 d vector is shown by examples in Table I.

Two key points are to be noted in the discussion of explicit and implicit values:

(1) The implicit value is the only value of interest to the *user* of the computer

(2) The *arithmetic designer* must devise algorithms which operate on operand digits representing the explicit values. These algorithms generate digits of the result such that the correct implicit value of the result is represented.

For an illustration of (2), consider a left shift (multiplication by two) of the operand $x \equiv 11011$ ($x \equiv 27$) used in Table I, line 6 (fraction, "one's complement," with $X = -{}^{4}/_{16}$). The shift will yield $y \equiv 10111$ as a result, since an "end-around" shift is the specified algorithm. The new explicit value is $y \equiv 23$, and the implicit value is $Y \equiv 2^{-4} \times (-2^5 + 1 + y) = -{}^{8}/_{16}$, which is the correct implicit value. The operation

$$y \equiv (x - 16) \times 2 + 1 \quad ,$$

on the explicit value x caused the operation

$$Y = 2 \times X \quad ,$$

on the implicit value X.

TABLE I. Some implicit values of radix-2 d-vector $x \equiv 1\ 1\ 0\ 1\ 1$.

IMPLIED ATTRIBUTES: RADIX POINT, NEGATIVE NUMBER REPRESENTATION, OTHERS	EXPRESSION FOR IMPLICIT VALUE X AS A FUNCTION OF EXPLICIT VALUE x	NUMERICAL IMPLICIT VALUE X (IN DECIMAL)
Integer, Magnitude	$X = x$	27
Integer, "Two's Complement"	$X = -2^5 + x$	−5
Integer, "One's Complement"	$X = -(2^5 - 1) + x$	−4
Fraction, Magnitude	$X = 2^{-5} \times x$	27/32
Fraction, "Two's Complement"	$X = 2^{-4} \times (-2^5 + x)$	−5/16
Fraction, "One's Complement"	$X = 2^{-4} \times (-2^5 + 1 + x)$	−4/16
Integer, Magnitude, "AN" Error Detection Encoding with $A = 3$ [16]	$X = x \div 3$	9
Fraction, "Two's Complement," Floating-Point Coefficient in Significant-Digit Arithmetic [15]	$X = \left(2 \times \left(2^{-4} \times (-2^5 + x)\right)\right) \times 2^{-1}$	$(-5/8) \times 2^{-1}$

D. Variable-Length and Multiple *d* Vectors

The most common format of d vectors in computers is the *fixed-length, single d vector* format which has been used in the discussion of implicit values of the preceding section. *Variable-length d* vectors are allowed in a number of computers as well, in which the number of digits may range from one to some maximum length n_{max}. The additional cost of such representation is the *length indicator* which must be provided for each d vector. The methods of length indication of d vectors are:

(1) Special symbols (flag bits, etc.) are used to identify the ends of the d vector

(2) A separate fixed-length d vector λ representing the length of the variable-length d vector is attached at one end (the *reference* end, usually the least significant end) of the d vector.

The method (2) above makes use of two separate d vectors $(\mathbf{x}, \lambda)$ to represent a numerical value X for the purposes of arithmetical processing. Other forms of *multiple d* vectors (double, triple, etc.) also have been used for the representation of numbers. The most common forms are:

(1) The use of a *sign tag* σx (values 0, 1) with a d vector $\mathbf{x}$ representing a magnitude X to form a *sign-and-magnitude d* vector

$$\mathbf{y} \equiv (\sigma x, \mathbf{x}) \quad \text{represents} \quad Y \equiv (-1)^{\sigma x} \times X$$

(2) The use of an *exponent* e to form a *floating-point d* vector

$$\mathbf{y} \equiv (\mathbf{e}, \mathbf{x}) \quad \text{represents} \quad Y \equiv X \times R^E$$

where $R \equiv \mathbf{r}_i^k$, given a fixed radix $\mathbf{r}_i$ of $\mathbf{x}$ and an integer $k \geq 1$

(3) The use of a *significance tag* vector $\mathbf{s}$ with a floating-point d vector to give a *significant-digit d* vector

$$\mathbf{y} \equiv (\mathbf{e}, \mathbf{x}, \mathbf{s}) \quad \text{represents} \quad Y \equiv X \times R^E$$

where X is known to S significant digits (the length of $\mathbf{x}$ usually exceeds S)

(4) The use of two or more d vectors to give *multiple-precision* forms, which represent the implicit value to a higher precision than provided by a single d vector

$$\mathbf{y} \equiv (\mathbf{x}, \mathbf{x}') \quad \text{represents} \quad Y \equiv X \times r^n + X'$$

where n is the length of $\mathbf{x}'$ and r is the radix (fixed) of $\mathbf{x}$ and $\mathbf{x}'$.

(5) The use of two d vectors to represent an *interval number* for interval arithmetic

$$\mathbf{z} \equiv (\mathbf{x}, \mathbf{y}) \quad \text{represents} \quad X \leq Z \leq Y$$

(6) Error-detection encoding by a separate code using a *check vector* $\mathbf{c}$ to form the *residue-encoded d* vector

$$\mathbf{y} \equiv (\mathbf{x}, \mathbf{c}) \quad \text{represents} \quad X \text{ iff } c \equiv A \mid x$$

where: A is the check modulus, and $A \mid x$ means "modulo-A residue of x."

In the use of multiple d vectors to represent one implicit value the expression for this implicit value involves the implicit values of all the component d vectors, which are processed separately when an arithmetical algorithm is carried out. It should be noted that these component d vectors themselves can be multiple; for example, sign-and-magnitude format may be used for floating-point d vectors $\mathbf{e}$ and $\mathbf{x}$ in (2) above, and floating-point d vectors may be used as the two interval-number representation d vectors in (5).

III. SPECIFICATION OF DIGITAL ARITHMETIC

A. The Digit-Algorithm Approach

The types of d vectors which are employed in digital computers and their interpretations in terms of implicit values have been presented in the preceding

part of this paper. At this point we set out to provide a basis for arithmetic design by describing a set of primitive operations, called *digit algorithms*, which are carried out on the individual digits of one or more *d* vector *operands* and yield values for individual digits of intermediate or final *result d* vectors.

One digit is the least indivisible component of a *d* vector in the arithmetic processing of numerical information. Each digit $\mathbf{x}_i$ of the *d* vector $\mathbf{x}$ is identified by its index i, its associated radix $\mathbf{r}_i$, and its set of allowed values Σ_i. Algorithms for arithmetical processing of *d* vectors consist of rules for transforming the digits of the operands into the digits of the result. These rules for the handling of individual digits of the operands and for the generation of the digits of a result are called *digit algorithms*. The rules for the handling of entire-digit vectors are called *vector algorithms*; they are composed of combinations or sequences of digit algorithms.

Digit algorithms for various number systems and arithmetical operations differ in their complexity. Whenever the set Σ_i contains more than two allowed digit values, a single binary storage element is not sufficient to store the digit $\mathbf{x}_i$. In this case the digit values are specified in terms of an *encoding* which establishes the equivalences between the allowed values of $\mathbf{x}_i$ and combinations of the logic values 1 and 0. Each element of the *d* vector is represented by a logic vector. Examples are the various encodings which have been devised for digit values in radix-10 arithmetic processors (8-4-2-1, excess-three, etc.).

A digit algorithm may be implemented either as a *table-lookup* operation (implemented by a combinational-logic network) or as a sequence of *subalgorithms*. When a radix-*r* digit value is specified by a logic vector (the states of several Boolean variables), it may be convenient to consider the radix-*r* digit to be composed of radix-2 *subdigits*. At this microscopic level of processing each radix-*r* digit is one *d* vector in a radix-2 number system, and the radix-*r* digit algorithm can be further specified as a combination or sequence of radix-2 subalgorithms. Finally, at the level of logic design, every digit algorithm or subalgorithm is expressed in terms of combinational or sequential Boolean functions. The specific form of these Boolean functions reflects the constraints imposed by the characteristics of currently available building blocks (fan-out, fan-in, cascading, delays, etc.). An example of a digit algorithm implemented in terms of subalgorithms is the one-decimal-digit adder in binary coded decimal (8-4-2-1) or in "excess-three" arithmetic processor.

Digit algorithms are classified according to the number of digits with the same index i which enter as operands to form the i^{th} result digit. For example

$$\mathbf{z}_i \longleftarrow 9 - \mathbf{x}_i \;; \quad \text{and} \quad \mathbf{z}_i \longleftarrow \mathbf{x}_{i-1} \;,$$

are *one-digit* algorithms, while

$$\mathbf{cg}_i \longleftarrow \lfloor (\mathbf{x}_i + \mathbf{y}_i) \div 10 \;,$$

is a *two-digit* algorithm for radix-10, conventional arithmetic, and

$$\mathbf{s}_i \longleftarrow \left(\sum_{j \equiv 1}^{11} \mathbf{x}_i^{(j)} \right) - 10 \times \mathbf{t}_{i+1} \;,$$

is an eleven-digit algorithm in a minimal-redundancy signed-digit radix-10 arithmetic [17].

Another important property of a digit algorithm for the i^{th} position

$$z_i \longleftarrow f(x_j, y_j) \ ,$$

is the number of positions j which affect z_i. When j has only *one* value, the digit algorithm is *local*. For examples, see the four preceding examples. When j has k different values, the digit algorithm is *k-dependent*. For example

$$z_i \longleftarrow 10 \mid (x_i + y_i + c_i) \ ,$$

with

$$c_i \longleftarrow \lfloor (x_{i-1} + y_{i-1} + c_{i-1}) \div 10 \ ,$$

is an $(i + 1)$-dependent digit algorithm in radix-10 conventional arithmetic.

B. Digit Algorithms for Three Classes of Number Systems

The concept of digit-algorithms is illustrated with the sets of digit-algorithms for three classes of number representation systems:

(1) *Conventional* (symbol CO) with a fixed positive radix $r_i \geq 2$ and the canonical digit-value set $\{0, 1, 2, \cdots, r_i - 1\}$
(2) *Signed-Digit* (symbol SD) with a fixed positive radix $r_i \geq 3$ and the redundant digit-value set $\{-a, \cdots, -1, 0, 1, \cdots, a\}$ in which a is an integer satisfying $r_i \div 2 < a < r_i$ [11, 17]
(3) *Residue* (symbol RE) with a set of positive radices r_i such that $\text{GCD}(r_i, r_j) \equiv 1$ for every pair of radices, and the digit-value sets $\{0, 1, 2, \cdots, r_i - 1\}$ for every r_i [14].

The Iverson ("APL") [7] notation is used to state digit algorithms, and zero-origin leftward indexing is employed. The notation is summarized in Table II. Note that the order of execution of APL statements is from right to left except as indicated by parentheses.

Table III lists digit algorithms which use one or two operand digits x_i, y_i (per position i) to form a single result-digit. The "SYMBOL" column contains a symbol used to identify the digit algorithm in vector algorithm specifications. The rightmost column "DEP.k" lists the dependency k of the digit algorithm as defined in the preceding section. Table IV lists the more complex digit algorithms which produce two result-digits (except for the RE number system) and may receive more than two operand-digits $(x^1, x^2, x^3, \cdots, x^m)$, or $(x, y, u^1, \cdots, u^q)$. All these digit algorithms are local. They are used to implement fast multiplication and multioperand summation vector algorithms, such as the "carry-save" addition for CO numbers.

TABLE II. Summary of notation.

SYMBOL	NAME	EXPLANATION
$\mathbf{x}$	d vector	n-tuple of digits
$\mathbf{x}_i$	digit	i^{th} component of $\mathbf{x}$
x	explicit value of $\mathbf{x}$	an integer, function of $\mathbf{x}$ and $\mathbf{r}$, defined by number system spec.
X	implicit value of $\mathbf{x}$	arithmetic function of x, defined by implied properties of $\mathbf{x}$
$\mathbf{r}_i$	radix of i^{th} digit	integer (except: $0, 1, -1$)
$\mathbf{w}_i$	weight of i^{th} digit	$\mathbf{w}_0 \equiv 1$; $\mathbf{w}_i \equiv \mathbf{r}_{i-1} \times \mathbf{w}_{i-1}$
Σ_i	value set of i^{th} digit	set of $\mathbf{r}_i$ or more consecutive integers, including 0
$\rho\mathbf{x}$	size (length) of $\mathbf{x}$	count of digits in $\mathbf{x}$
$X \mathbin{@} Y$	relational operator $@ \equiv \{< \leqslant = \geqslant > \neq\}$	result is 1 if the relation @ holds, and 0 otherwise
$\lvert X$	magnitude of X	
$Y \mid X$	residue	modulo Y residue of X
$\lfloor X$	floor	integer such that $\lfloor X \leqslant X < 1 + \lfloor X$ holds
$\times X$	signum	$\times X \equiv (X > 0) - (X < 0)$ values are $+1, 0, -1$
$X * Y$	power	X to the Y^{th} power
$\equiv$	identity	$X \equiv Y$ asserts identity; not to be confused with operator =

TABLE III. One-digit and two-digit algorithms with one-digit results.

ALGORITHM NAME	NO. SYSTEM "NS"	SYMBOL: D_{xx}/NS	DIGIT ALGORITHM: ARITHMETIC SPECIFICATION	DEP. k
TRANSFER	All	$\longleftarrow$	$\mathbf{z}_i \longleftarrow \mathbf{x}_i$	1
LEFT SHIFT (m positions)	CO, SD	DLm/NS	$\mathbf{z}_{i+m} \longleftarrow \mathbf{x}_i$	1
	RE		not defined	
RIGHT SHIFT (m positions)	CO, SD	DRm/NS	$\mathbf{z}_{i-m} \longleftarrow \mathbf{x}_i$	1
	RE		not defined	
INVERSE	CO, RE	DI/NS	$\mathbf{z}_i \longleftarrow (r_i - 1) - \mathbf{x}_i$	1
	SD	DI/SD	$\mathbf{z}_i \longleftarrow 0 - \mathbf{x}_i$	1
CARRY (2 digits)	CO	DC/CO	$\mathbf{c}_{i+1} \longleftarrow (\mathbf{x}_i + \mathbf{y}_i + \mathbf{c}_i) \geqslant \mathbf{r}_i$	$i+1$
	SD	DC/SD	$\mathbf{t}_{i+1} = (\times\mathbf{x}_i + \mathbf{y}_i) \times (a \leqslant \lvert (\mathbf{x}_i + \mathbf{y}_i))$	1
	RE		not defined	
SUM (2 digits)	CO	D+/CO	$\mathbf{s}_i \longleftarrow \mathbf{r}_i \mid (\mathbf{x}_i + \mathbf{y}_i + \mathbf{c}_i)$	$i+1$
	SD	D+/SD	$\mathbf{s}_i \longleftarrow \mathbf{x}_i + \mathbf{y}_i + \mathbf{t}_i - \mathbf{t}_{i+1} \times \mathbf{r}_i$	2
	RE	D+/RE	$\mathbf{s}_i \longleftarrow \mathbf{r}_i \mid (\mathbf{x}_i + \mathbf{y}_i)$	1
BORROW (2 digits)	CO	DB/CO	$\mathbf{b}_{i+1} \longleftarrow ((\mathbf{x}_i - \mathbf{y}_i) - \mathbf{b}_i) < 0$	$i+1$
	SD	DB/SD	$\mathbf{t}_{i+1} \longleftarrow (\times\mathbf{x}_i - \mathbf{y}_i) \times (a \leqslant \lvert (\mathbf{x}_i - \mathbf{y}_i))$	1
	RE		not defined	
DIFFERENCE (2 digits)	CO	D−/CO	$\mathbf{d}_i \longleftarrow \mathbf{r}_i \mid (\mathbf{x}_i - \mathbf{y}_i) - \mathbf{b}_i$	$i+1$
	SD	D−/SD	$\mathbf{d}_i \longleftarrow (\mathbf{x}_i - \mathbf{y}_i) + \mathbf{t}_i - \mathbf{t}_{i+1} \times \mathbf{r}_i$	2
	RE	D−/RE	$\mathbf{d}_i \longleftarrow \mathbf{r}_i \mid (\mathbf{x}_i - \mathbf{y}_i)$	1

TABLE IV. Local-digit algorithms with multidigit inputs and/or two-digit results.

ALGORITHM NAME; NOTES	NO. SYSTEM "NS"	SYMBOL: D*xx*/NS	DIGIT ALGORITHM: ARITHMETIC SPECIFICATION
PRODUCT (2 digits)	CO	D×/CO	$\mathbf{ps}_i \leftarrow \mathbf{r}_i \mid (\mathbf{x}_i \times \mathbf{y}_i)$ $\mathbf{pc}_{i+1} \leftarrow \lfloor (\mathbf{x}_i \times \mathbf{y}_i) \div \mathbf{r}_i$
	SD	D×/SD	$\mathbf{ps}_i \leftarrow (\mathbf{x}_i \times \mathbf{y}_i) - \mathbf{pc}_{i+1} \times \mathbf{r}_i$ $\mathbf{pc}_{i+1} \leftarrow (\times \mathbf{x}_i \times \mathbf{y}_i) \times \lfloor (((\mathbf{r}_i - 1) - a) + \mid (\mathbf{x}_i \times \mathbf{y}_i)) \div \mathbf{r}_i$
	RE	D×/RE	$\mathbf{z}_i \leftarrow \mathbf{r}_i \mid (\mathbf{x}_i \times \mathbf{y}_i)$
SUM (m digits) [1]for CO, SD: $m \leqslant \mathbf{r}_i + 1$ [2]for RE: m, q are arbitrary	CO[1]	D+m/CO	$\mathbf{ms}_i \leftarrow \mathbf{r}_i \mid (\mathbf{x}_i^1 + \cdots + \mathbf{x}_i^m)$ $\mathbf{mc}_{i+1} \leftarrow \lfloor (\mathbf{x}_i^1 + \cdots + \mathbf{x}_i^m) \div \mathbf{r}_i$
	SD[1]	D+m/SD	$\mathbf{ms}_i \leftarrow (\mathbf{x}_i^1 + \cdots + \mathbf{x}_i^m) - \mathbf{mc}_{i+1} \times \mathbf{r}_i$ $\mathbf{mc}_{i+1} \leftarrow \mathbf{sn}_i \times \lfloor (((\mathbf{r}_i - a) - 1) + \mid (\mathbf{x}_i^1 + \cdots + \mathbf{x}_i^m)) \div \mathbf{r}_i$ where: $\mathbf{sn}_i \leftarrow \times (\mathbf{x}_i^1 + \cdots + \mathbf{x}_i^m)$
	RE[2]	D+m/RE	$\mathbf{z}_i \leftarrow \mathbf{r}_i \mid (\mathbf{x}_i^1 + \cdots + \mathbf{x}_i^m)$
PRODUCT-AND-SUM (2 and q digits) [3]for CO: $q \leqslant 2$ [4]for SD: $q \leqslant \mathbf{r}_i - a + 1$	CO[3]	D×+q/CO	$\mathbf{ns}_i \leftarrow \mathbf{r}_i \mid (\mathbf{u}_i^1 + \mathbf{u}_i^2 + \mathbf{x}_i \times \mathbf{y}_i)$ $\mathbf{nc}_{i+1} \leftarrow \lfloor (\mathbf{u}_i^1 + \mathbf{u}_i^2 + \mathbf{x}_i \times \mathbf{y}_i) \div \mathbf{r}_i$
	SD[4]	D×+q/SD	$\mathbf{ns}_i \leftarrow (\mathbf{u}_i^1 + \cdots + \mathbf{u}_i^q + \mathbf{x}_i \times \mathbf{y}_i) - \mathbf{nc}_{i+1} \times \mathbf{r}_i$ $\mathbf{nc}_{i+1} \leftarrow \mathbf{sn}_i \times \lfloor (((\mathbf{r}_i - a) - 1) + \mid (\mathbf{u}_i^1 + \cdots + \mathbf{u}_i^q + \mathbf{x}_i \times \mathbf{y}_i)) \div \mathbf{r}_i$ where: $\mathbf{sn}_i \leftarrow \times (\mathbf{u}_i^1 + \cdots + \mathbf{u}_i^q + \mathbf{x}_i \times \mathbf{y}_i)$
	RE[2]	D×+q/RE	$\mathbf{z}_i \leftarrow \mathbf{r}_i \mid (\mathbf{u}_i^1 + \cdots + \mathbf{u}_i^q + \mathbf{x}_i \times \mathbf{y}_i)$

C. Vector Algorithms for Conventional Number Systems

In order to perform arithmetic, the digit algorithms are organized into *vector algorithms*. A vector algorithm is a prescription for generating the result(s) from the given operand(s). The *operands* are d vectors (**x**, **y**, etc.), employing zero-origin leftward indexing and characterized by their length n, the radix vector **r** and the sets of allowed digit values Σ_i for $0 \leq i \leq n-1$. In the conventional number systems the fixed radix $\mathbf{r}_i > 1$ and the canonical set of digit values are employed for every position. The weight $\mathbf{r}_i^i$ is associated with the digit $\mathbf{x}_i$.

The *result* may be either a digit vector (**z**, **p**, **q**, etc.), or a condition-code vector ε. A d vector will be generated when an arithmetic operation (for example: addition, subtraction, multiplication, division) is specified by the operation code. A two-valued logic variable ε_i (1 for true and 0 for false) will be the result for magnitude comparison, zero detection, overflow detection, range estimate, and similar algorithms.

The vector algorithms describe the digits of the result(s) in terms of the operand digits. Some digits of the result(s) may be specified directly as a function of the particular number system (for instance, the right-end digit in a left shift, etc.). If the result is a condition-code logic variable, its value is specified in terms of all or some digits of the operand(s).

The arithmetic algorithms usually specified by the set of operation codes are: transfer, additive inverse (sign change), absolute value, addition, subtraction,

multiplication, roundoff, arithmetic shifts (for normalization, overflow correction, alignment in floating point) and division. Occasionally inter-radix conversions, square root, exponentiation, trigonometric functions and other more-elaborate algorithms may be specified, as well as intersystem or interformat conversions (residue to binary, etc.). The extension and contraction of range, although not explicitly stated, are implied in the above list. The preceding algorithms all yield numeric results; besides them logic results are obtained from the sign test, zero test, comparison and range-estimate algorithms.

Table V lists a typical set of vector algorithms with one or two operands. This is the "user's viewpoint" in which only the implicit values and the lengths of the results are specified. The "Statement" column uses APL function notation to describe the vector algorithms. The dollar sign \$ represents a number-system identifier which uniquely identifies the details (both explicit and implied) of the number representation system. In addition to operand length, radix vector, indexing system, it also describes the implied properties, such as those listed in Table I. A program specifying the result-digits by means of digit algorithms exists for every identifier.

Tables VI and VII illustrate vector algorithm descriptions for some of the

TABLE V. Common one-vector and two-vector algorithms: symbols and "user's" specifications.

VECTOR ALGORITHM	STATEMENT[1]	LENGTH ρz[2]	IMPLICIT VALUE Z
TRANSFER	z ⟵ x	n	$Z \longleftarrow X$
RANGE EXTENSION[3]			
m positions	z ⟵ m REX\$ x	$n+m$	$Z \longleftarrow X$
RANGE CONTRACTION			
m positions	z ⟵ m RCN\$ x	$n-m$	$Z \longleftarrow X$
LEFT ARITH. SHIFT[4]			
m positions	z ⟵ m LSH\$ x	n	$Z \longleftarrow X \times r_i * m$
RIGHT ARITH. SHIFT[4]			
m positions	z ⟵ m RSH\$ x	n	$Z \longleftarrow X \times r_i * (-m)$
ADDITIVE INVERSE	z ⟵ INV\$ x	n	$Z \longleftarrow 0 - X$
ROUNDOFF[5]			
m positions	z ⟵ m RND\$ x	$n-m$	$Z \longleftarrow (X - D)$
SUM	z ⟵ x SUM\$ y	n	$Z \longleftarrow X + Y$
DIFFERENCE	z ⟵ x DIF\$ y	n	$Z \longleftarrow X - Y$
PRODUCT	z ⟵ x PRD\$ y	$n+n$	$Z \longleftarrow X \times Y$
QUOTIENT[6]	q ⟵ x QNT\$ y	n	$Q \longleftarrow (X - W) \div Y$
REMAINDER[6]	w ⟵ x REM\$ y	n	$W \longleftarrow X - Q \times Y$
ZERO TEST[7]	ε_0 ⟵ ZRO\$ x		$\varepsilon_0 \longleftarrow X = 0$
POSITIVE TEST[7]	ε_1 ⟵ POS\$ x		$\varepsilon_1 \longleftarrow X > 0$
NEGATIVE TEST[7]	ε_2 ⟵ NEG\$ x		$\varepsilon_2 \longleftarrow X < 0$
RANGE TEST (A, B)[7]	ε_3 ⟵ A, B RTS\$ x		$\varepsilon_3 \longleftarrow (X < A) \vee (X > B)$

[1]The symbol \$ represents the number system identifier for the algorithm.

[2]Operands x, y have the length n unless explicitly noted otherwise.

[3]If radix of x is not fixed, replace m by a vector listing the m new radices to be used.

[4]Applicable to fixed-radix ($r_i \geq 2$) operands only.

[5]The range of D is $D_{min} < D < D_{max}$; the range is a function of the m deleted digits and their radices.

[6]Length of operand x is $n+n$ digits. The remainder W has the same sign as the dividend X and satisfies $(|W) < (|Y)$.

[7]The tests yield two-valued results ε_i, which form the condition-code vector ε.

algorithms listed in Table V. The CO (conventional) number system with n-digit operands and an even fixed radix $\mathbf{r}_i$ has been selected. Two methods to represent negative numbers are employed:

(1) The symbol RC stands for Range Complement (often called two's or ten's complement when $\mathbf{r}_i$ is 2 or 10, respectively) in which negative numbers are represented as complements with respect to $(\mathbf{r}_i)^n$

(2) The symbol DC stands for Digit Complement (colloquially: one's, or nine's complement when $\mathbf{r}_i$ is 2 or 10, respectively) in which negative numbers are represented as complements with respect to $(\mathbf{r}_i)^n - 1$.

Table VI specifies vector algorithms which generate a digit vector as the result. The rightmost column identifies and labels the singularities which may occur with the given algorithm. Three common variants of the roundoff algorithm are specified. The "exact round" singularity for method 2 identifies the case in

TABLE VI. Vector algorithm specifications.

VECTOR ALGORITHM	NUMBER SYSTEM ID. "$"	DIGIT POSITIONS IN z	DIGIT ALGORITHMS FOR SPECIFIED POSITIONS OF z	NAME OF SINGULARITY (SEE TABLE VII)
$\mathbf{z} \leftarrow m$ REX$ x	RC, DC	$0, \cdots, n-1$	$\mathbf{z}_i \leftarrow \mathbf{x}_i$	None
	RC, DC	$n, \cdots, n+m-1$	$\mathbf{z}_i \leftarrow (\mathbf{r}_i - 1) \times (\mathbf{x}_{n-1} \geqslant \mathbf{r}_i \div 2)$	
$\mathbf{z} \leftarrow m$ RCN$ x	RC, DC	$0, \cdots, n-m-1$	$\mathbf{z}_i \leftarrow \mathbf{x}_i$	Overflow OF/RCN$
	RC, DC	$n-m, \cdots, n-1$	$\mathbf{z}_i \leftarrow \iota 0$ (null)	
$\mathbf{z} \leftarrow m$ LSH$ x	RC, DC	$m, \cdots, n-1$	DLm/CO	Overflow OF/LSH$
	RC	$0, \cdots, m-1$	$\mathbf{z}_i \leftarrow 0$	
	DC	$0, \cdots, m-1$	$\mathbf{z}_i \leftarrow (\mathbf{r}_i - 1) \times (\mathbf{x}_{n-1} \geqslant \mathbf{r}_i \div 2)$	
$\mathbf{z} \leftarrow m$ RSH$	RC, DC	$0, \cdots, n-m-1$	DRm/CO	Truncation TR/RSH$
	RC, DC	$n-m, \cdots, n-1$	$\mathbf{z}_i \leftarrow (\mathbf{r}_i - 1) \times (\mathbf{x}_{n-1} \geqslant \mathbf{r}_i \div 2)$	
$\mathbf{z} \leftarrow$ INV$ x	DC	$0, \cdots, n-1$	DI/CO	Inverse Anomaly AN/INV$
	RC	$0, \cdots, n-1$	D+/CO with: $\mathbf{x}_i \leftarrow 0$, $\mathbf{y}_i \leftarrow (\mathbf{r}_i - 1) - \mathbf{x}_i$	
	RC	0	$\mathbf{c}_0 \leftarrow 1$	
$\mathbf{z} \leftarrow \mathbf{x}$ SUM$ y or $\mathbf{z} \leftarrow \mathbf{x}$ DIF$ y	RC, DC	$0, \cdots, n-1$	D+/CO or D−/CO	Overflow OF SUM$ or OF/DIF$
	RC	0	$\mathbf{c}_0 \leftarrow 0$ or $\mathbf{b}_0 \leftarrow 0$	
	DC	0	$\mathbf{c}_0 \leftarrow \mathbf{c}_n$ or $\mathbf{b}_0 \leftarrow \mathbf{b}_n$	
$\mathbf{z} \leftarrow m$ RND$ x	RC, DC	$0, \cdots, m-1$	$\mathbf{z}_i \leftarrow \iota 0$ (null)	
(1) Truncation	RC, DC	$m, \cdots, n-1$	$\mathbf{z}_i \leftarrow \mathbf{x}_i$	None
(2) Local Adjust.	RC, DC	m	$\mathbf{z}_m \leftarrow \mathbf{x}_m + (\mathbf{x}_m < \mathbf{r}_i \div 2)$	Exact Round EX/RND2$
		$m+1, \cdots, n-1$	$\mathbf{z}_i \leftarrow \mathbf{x}_i$	
(3) Sum Adjust.	RC, DC	m	D+/CO with: $\mathbf{c}_m \leftarrow 0$, $\mathbf{y}_m \leftarrow (\mathbf{x}_{m+1} \geqslant \mathbf{r}_i \div 2)$	Overflow OF/RND3$
		$m+1, \cdots, n-1$	D+/CO with: $\mathbf{y}_i \leftarrow 0$	

TABLE VII. Vector algorithms for tests and singularities.

VECTOR ALGORITHM	NUMBER SYSTEM	ALGORITHM SPECIFICATION
TESTS:		
$\varepsilon_0 \leftarrow$ ZRO\$ x	RC	$\varepsilon_0 \leftarrow (0=+/\mathbf{x})$
	DC	$\varepsilon_0 \leftarrow (0=+/\mathbf{x}) \vee ((+/\mathbf{x})=(\mathbf{r}_i-1)\times\rho\mathbf{x})$
$\varepsilon_1 \leftarrow$ POS\$ x	RC, DC	$\varepsilon_1 \leftarrow (\mathbf{x}_{n-1}<\mathbf{r}_i \div 2)$
$\varepsilon_2 \leftarrow$ NEG\$ x	RC, DC	$\varepsilon_2 \leftarrow (\mathbf{x}_{n-1}\geqslant\mathbf{r}_i \div 2)$
SINGULARITIES:		
$\sigma_0 \leftarrow$ OF/mRCN\$ x $\sigma_0 \leftarrow$ OF/mLSH\$ x	RC, DC	$\sigma_0 \leftarrow ((\mathbf{x}_{n-m-1}<\mathbf{r}_i \div 2) \wedge (0\neq+/(m\uparrow\mathbf{x})) \vee$ $(\mathbf{x}_{n-m-1}\geqslant\mathbf{r}_i \div 2) \wedge (\rho\mathbf{x}\times(\mathbf{r}_i-1))\neq+/(m\uparrow\mathbf{x})$
$\sigma_1 \leftarrow$ TR/mRSH\$ x $\sigma_1 \leftarrow$ EX/RND2\$ x	RC	$\sigma_1 \leftarrow (0\neq+/(-m\uparrow\mathbf{x}))$
	DC	$\sigma_1 \leftarrow (0\neq+/(-m\uparrow\mathbf{x})) \vee (\rho\mathbf{x}\times(\mathbf{r}_i-1))\neq+/(-m\uparrow\mathbf{x})$
$\sigma_2 \leftarrow$ AN/INV\$ x	RC	$\sigma_2 \leftarrow (\mathbf{x}_{n-1}=\mathbf{r}_i \div 2) \wedge ((+/\mathbf{x})=(\mathbf{r}_i \div 2))$
	DC	$\sigma_2 \leftarrow (0=+/\mathbf{x})$
$\sigma_3 \leftarrow$ OF/SUM\$ x, y $\sigma_3 \leftarrow$ OF/DIF\$ x, y	RC, DC	$\sigma_3 \leftarrow ((\mathbf{x}_{n-1}<\mathbf{r}_i \div 2) \wedge (\mathbf{y}_{n-1}<\mathbf{r}_i \div 2) \wedge (\mathbf{z}_{n-1}\geqslant\mathbf{r}_i \div 2)) \vee$ $((\mathbf{x}_{n-1}\geqslant\mathbf{r}_i \div 2) \wedge (\mathbf{y}_{n-1}\geqslant\mathbf{r}_i \div 2) \wedge (\mathbf{z}_{n-1}<\mathbf{r}_i \div 2))$
$\sigma_4 \leftarrow$ OF/RND3\$ x	RC, DC	$\sigma_4 \leftarrow ((\mathbf{x}_{n-1}<\mathbf{r}_i \div 2) \wedge (\mathbf{z}_{n-1}\geqslant\mathbf{r}_i \div 2)) \vee$ $((\mathbf{x}_{n-1}\geqslant\mathbf{r}_i \div 2) \wedge (\mathbf{z}_{n-1}<\mathbf{r}_i \div 2))$

which truncation should take place regardless of the value of $\mathbf{x}_m$. The sum and difference algorithms represent the "ripple" carry or borrow implementation. The methods of fast addition (lookahead, conditional sum, etc.) can also be specified in a similar fashion. The specifications of fast addition, multiplication, and division will appear together with their comparative evaluations in a subsequent paper.

Table VII specifies the algorithms which generate test results ε_i and singularity indicators σ_i. Two new operator symbols appear in this table. The sum reduction operator $+/\mathbf{x}$ forms the sum $\mathbf{x}_0 + \mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_{n-1}$ of all digits of $\mathbf{x}$. The "take" operator $k\uparrow\mathbf{x}$ takes the leftmost k digits of $\mathbf{x}$ if k is positive, and the rightmost k digits of $\mathbf{x}$ if k is negative.

In the singularity list, σ_0 indicates that a left shift or range contraction cannot be done without losing an "overflow" part of the result. The signal σ_1 indicates that a right shift will result in an imprecise (truncated) result. The signal σ_2 indicates an anomaly in the inverse—for RC, it results in an overflow, and for DC the "minus zero" is generated. Signals σ_3 and σ_4 detect overflow after add, subtract and round (with addition) operations.

The main conclusion which can be drawn from Tables III–VII is that a purely arithmetical specification of digital computer processors offers a compact and technology-independent method of describing the operations which are needed. The logic designer takes over at this point and chooses the algorithms which are most effective for his present set of logic circuitry.

ACKNOWLEDGEMENT

This research was supported in part by the U. S. Atomic Energy Commission, Contract No. AT(11-1) Gen 10, Project 14.

Thanks are due to the many graduate students in the UCLA Computer Science Department Course 225A "Computer System Design: Arithmetic Processors" whose comments and constructive criticism on the description of computer arithmetic have stimulated the evolution of this paper.

REFERENCES

[1] D. L. MacSorley, "High Speed Arithmetic in Digital Computers," *Proc. IRE*, 49, No. 1, 67–91 (January 1961).

[2] S. Winograd, "On the Time Required to Perform Addition," *J. ACM*, 12, No. 2, 277–285 (1965).

[3] S. Winograd, "On the Time Required to Perform Multiplication," *J. ACM*, 14, No. 4, 793–802 (1967).

[4] A. Avižienis, "On the Problem of Computational Time and Complexity of Arithmetic Functions," *Proc. ACM Symp. Theory of Computing*, Los Angeles, California, 255–258 (May 5–6, 1969).

[5] A. Avižienis, "Theory of Digital-Computer Arithmetic," Notes for UCLA Course Engineering 225A, Ch. 1–3 (to be published).

[6] A. A. Markov, *Theory of Algorithms*, USSR Academy of Sciences: MOSCOW, 1954. English translation published by the Israel Program for Scientific Translations, PST Catalog No. 219, Jerusalem (1961).

[7] K. E. Iverson, *A Programming Language* (New York: Wiley, 1962).

[8] G. W. Patterson, "Algebraic Foundations of Number-Representation Systems," Univ. of Pennsylvania, The Moore School of Electrical Engineering, Report No. 59-00 (September 1, 1958).

[9] J. E. Robertson, "Redundant Number Systems for Digital-Computer Arithmetic," Notes for the University of Michigan Eng. Summer Conf., "Topics in the Design of Digital Computing Machines," Ann Arbor, Mich. (July 6–10, 1959).

[10] H. L. Garner, "Number Systems and Arithmetic," *Advances in Computers*, 6, 131–194 (Academic Press, 1965).

[11] A. Avižienis, "Signed-Digit Number Representations for Fast Parallel Arithmetic," *IRE Trans. Electr. Comput.*, EC-10, No. 3, 389–400 (September 1961).

[12] G. F. Songster, "Negative-Base Number Representation Systems," *IEEE Trans. Comput.*, EC-12, No. 3, 274–277 (June 1963).

[13] D. E. Knuth, "An Imaginary Number System," *Comm. ACM*, 3, 245–247 (April 1960).

[14] A. Svoboda, "The Numerical System of Residual Classes," *Digital Information Processors*, 543–574 (W. Hoffman, Ed., New York: Interscience Publishers, 1962).

[15] N. Metropolis and R. L. Ashenhurst, "Significant-Digit Computer Arithmetic," *IRE Trans. Elec. Comput.*, EC-7, No. 4, 265–267 (December 1958).

[16] W. W. Peterson, *Error-Correcting Codes*, 236–244 (Massachusetts: The MIT Press, 1961).

[17] A. Avižienis and C. Tung, "A Universal Arithmetic Building Element (ABE) and Design Methods for Arithmetic Processors," *IEEE Trans. Comput.*, C-19, No. 8, 733–745 (August 1970).