NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Technical Report 32-1476

Digital Fault Diagnosis by Low-Cost Arithmetical Coding Techniques

Algirdas Avižienis

JET PROPULSION LABORATORY
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA

August 1, 1970

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Technical Report 32-1476

Digital Fault Diagnosis by Low-Cost Arithmetical Coding Techniques

Algirdas Avižienis

JET PROPULSION LABORATORY
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA

August 1, 1970

Prepared Under Contract No. NAS 7-100
National Aeronautics and Space Administration

Preface

The work described in this report was performed by the Astrionics Division of the Jet Propulsion Laboratory and appeared originally in the *Proceedings of the Purdue Centennial Year Symposium on Information Processing*, Vol. I, pp. 81–91, Purdue University, Lafayette, Ind., Apr. 28–30, 1969. The author is also Associate Professor at the Computer Science Department of the University of California at Los Angeles.

Acknowledgment

The author wishes to acknowledge stimulating discussions with Mr. J. J. Wedel of JPL, who has conducted some early explorations of hybrid product codes.

Contents

Abstract

Error-detecting and correcting codes may be employed to diagnose (i.e., detect and locate) logic faults in a digital computer concurrently with its normal operation. Arithmetic error codes must be used if the same code is to be used in the entire computer. A class of arithmetic codes with a low-cost check algorithm is analyzed which possesses partial fault location properties. Complete fault location (i.e., single error correction) is then attained by multiple encodings. The results are applied to both residue and product (or An) arithmetic error codes.

Digital Fault Diagnosis by Low-Cost Arithmetical Coding Techniques

I. Application of Coding Techniques in Computers

The most common approach currently to the diagnosis of digital computer logic utilizes diagnostic programs. The normal operation of the computer is interrupted at a selected time or upon some indication of an error, and the diagnostic program is used to test the logic circuits for the presence of faults. Special-purpose hardware has been added to several contemporary computer systems in order to facilitate error detection and the running of the diagnostic programs.

An alternate approach to fault diagnosis is offered by error-detecting and correcting codes. Encoded data words (using parity, Hamming, residue, and some other codes) are found in numerous digital systems. Most frequently these codes are used as the means for error detection which is followed by the use of a diagnostic program. Recently, error-detecting codes have been employed to encode the instruction words as well as the data words of an experimental computer (Ref. 1). The most common use of error codes is for data transfers within the computer; an error-correcting code will identify the bit index (or indices) i of the altered bits and thus locate the fault to a bit-position in the logic which delivered the incorrect word. Such location either will suffice to indicate a replaceable logic package, or will abbreviate the subsequent diagnosis carried out by a program.

If the arithmetic operations as well as data transfers are to be validated, the error code must be preserved during arithmetic, and the class of applicable codes is limited to the arithmetic error codes (Ref. 2). This report covers some recent results on fault-location properties of arithmetic codes which have been obtained as a part of a continued study of the application of arithmetic codes in digital systems (Refs. 3–5).

For the purposes of this report, a *fault* is defined as the deviation of a logic variable from the "perfect" value, i.e., from the value which has been specified in the design. Faults are caused by component failures or temporary malfunctions, and by external interference, such as

power transients, electromagnetic radiation, etc. An *error* is the symptom of a fault. *Arithmetic* errors cause deviations in the values of machine words; *logic* errors alter individual logic (control) variables and lead to incorrect algorithms. Coding techniques have been applied to detect and to locate both types of errors. This report considers fault detection and location for both permanent and transient faults which result in arithmetic errors.

II. Properties of Arithmetic Error Codes

The two principal classes of arithmetic error codes considered here are the *residue* codes (Ref. 6) and the *product* (or *An*, Ref. 7) codes. Both classes of codes detect faults by detecting fault-induced deviations from the perfect (i.e., design-specified) results.

Given the radix r n-digit perfect result $X(x_{n-1}\cdots x_i\cdots x_0)$, an arithmetical error is observed when the actual result is $X^*(x^*_{n-1}\cdots x^*_i\cdots x^*_0)$ such that $x^*_i \neq x_i$ holds for one or more positions ($0 \leq i \leq n-1$) of the actual result X^*. The arithmetical error is characterized by its *error number* E composed of n digits $(e_{n-1}\cdots e_i\cdots e_0)$ which have the values

$$e_i = x^*_i - x_i, \qquad \text{for } 0 \leq i \leq n-1.$$

The magnitude, the weight, and the distribution of nonzero digits in the error number provide the means to relate the error to the fault which caused it (Refs. 2 and 4).

The undetectable error magnitudes (to be called *misses*) for both residue and product codes are all integer multiples kA of a positive integer $A > 1$. The integer A is called the *check constant*. Its choice determines both the effectiveness and the cost of the checking method. In some cases the identification of the error value and an error correction is possible. Both methods employ the same theoretical basis for checking, but they differ considerably in their logic implementation.

In the *residue code* every operand X forms a pair with the number X', which is called its *check symbol*. The value of X' is the least positive modulo A residue of X, which will be denoted by $A|X$ in this report. The number X is considered to be an unsigned integer. The check constant A is called the *check modulus* in this method. When an algorithm is to be performed with the operands (X,X') and (Y,Y'), then their check symbols X' and Y' are sent to a separate *check processor*. The operands X, Y enter the *main processor*, which computes the result Z. The *checking algorithm* for the main processor computes the residue $A|Z$. The check processor independently computes the check result Z', and compares its value to $A|Z$. If the values are equal, either the perfect result has been obtained, or a miss has occurred. Disagreement indicates a fault in either the main or the check processor; this uncertainty precludes fault location and error correction without supplementary procedures. An exception in the check procedure occurs for division $X \div Y$ which produces the quotient Q and the remainder P. The checking algorithm computes both $A|Q$ and $A|P$. The check processor computes the value $(A|Q) \cdot Y' + (A|P)$ and compares it to X' for equality.

The *modified residue code* differs from the residue code in only one respect: the value of the check symbol X'' which forms the pair (X,X'') with the operand X has the value $X'' = A - (A|X)$; that is, X'' is the complement with respect to A of $A|X$. The algorithms of the check processor operate on the check symbols X'', Y'' to compute the check result Z'' which has the value $Z'' = A - (A|Z)$ when an error has not occurred. The checking algorithm computes $A|Z$ for the result Z from the main processor and adds it to Z'' modulo A to get the check sum $F = A|[(A|Z) + Z'']$, where $F = 0$ indicates that either the result is correct, or a miss has occurred. The modified residue codes have definite advantages in the implementation of fault detection and fault location for both single-use and multiple-use faults.

A *product* (or *An*) *code* is obtained when every operand X in a conventional number system is multiplied by the *check factor* $A > 1$. The checking algorithm computes the residue $A|Z$, where Z is a product-coded result produced by the processor. The result $A|Z = 0$ indicates either a perfect result, or a miss (an error value $E = kA$). A nonzero value of $A|Z$ indicates a fault; for certain choices of A the nonzero value of $A|Z$ indicates the error value E and makes error correction possible. The algorithms of the processor are designed to compute with product-coded numbers (Ref. 8). All intermediate steps of the algorithms must preserve product coding of operands and partial results in order to retain the error-checking properties in the result. The product code is nonseparable and its hardware cost is found in the greater complexity of the main processor. The cost of the residue code is in the addition of the check processor. The checking algorithm to compute $A|Z$ and the undetectable error magnitudes kA remain the same for both classes of codes.

III. A Class of Low-Cost Binary Codes

The case of most immediate interest is the checking of binary arithmetic; it also presents the most direct relationship between a fault and the resulting change in the value of a result. A *local fault* causes an error in only one digital position either of an operand or of a result in a one-use algorithm (transfer, complementation, shifting, addition, which use the faulty circuit only once). The effects of more damaging faults may be expressed as the composite effect of a set of local faults (Ref. 4). In parallel transfer, shifting and bit-wise complementation of an n-bit number, a local fault generates an error magnitude 2^j, which has the weight 1. In modulo N (with $N = 2^n$ or $2^n - 1$) parallel addition (or subtraction), the error magnitude is either 2^j or $N - 2^j$. In multiplication and division, the parallel adder and associated circuits are used repeatedly, and a local fault will contribute an error number during some or all uses of the faulty circuit. The repeated-use faults are also encountered in transfers, shifts and additions in variable-field-length (series-parallel) computers.

The usual criterion for the effectiveness of a given arithmetical code is the guaranteed detection of weight-1, weight-2, and weight-3 error magnitudes (Ref. 9). This criterion is generally used in transmission codes. A more general criterion which includes an algorithmic arithmetic processor is the probability of detection of a local fault by the application of the checking algorithm to the results of the entire set of algorithms of the processor. The second criterion which is essential in the choice of an arithmetical code is the total (in time and hardware) *cost of checking*, which depends on two factors:

(1) The compatibility of the code with the algorithms of the processor.

(2) The direct cost of the checking algorithm.

It is evident that a practical checking method must be acceptable from the viewpoints of both cost and effectiveness; simultaneous optimization is an interesting objective. For radix-2 numbers, any odd integer $A > 1$ will detect error magnitudes of weight 1. Values of A which provide detection of all error magnitudes of weights 2 and 3 and correction of weight 1 magnitudes within a limited range of X have been described (Refs. 7, 9, 10) as well as values of A for burst-error and large-distance error detection and correction (Refs. 11, 12, 13). The cost of checking and the detection of errors arising from repeated use of faulty circuits were not considered in these codes.

The search for values of A which are characterized by high compatibility with binary arithmetic and a low-cost checking algorithm (Ref. 3) led to the choice of *low-cost* arithmetic codes which employ check constants A of the form

$$A = 2^a - 1, \qquad \text{with integer } a > 1.$$

The parameter a is called the *group length* of the code. Since division is a complex arithmetic algorithm, the checking algorithms for odd $A > 1$ are relatively costly and slow. An exception occurs for the check constant $2^a - 1$, since the congruence

$$K_i r^i \equiv K_i \text{ modulo } (r - 1), \qquad \text{with } r = 2^a,$$

permits the use of modulo $2^a - 1$ summation of the k groups (a-bit segments of value K_i, with $0 \leq K_i \leq 2^a - 1$) which compose the ka-bit number Z to compute the *check sum* which is the least positive residue of Z modulo $2^a - 1$. In this special case division is replaced by a simple "end-around carry" addition algorithm, which may be executed either sequentially or simultaneously for all k groups. Implementations which do not require addition have been described for the $a = 2$ case (Refs. 14 and 15). The "one's complement" algorithms are more compatible with low-cost coding because $2^{ka} - 1$ is divisible by $2^a - 1$, while 2^{ka} is not, and difficulties arise in implementing "two's complement" arithmetic.

IV. Partial Fault-Location by the Low-Cost Codes

To attain the complete location of a local fault, or to effect the correction of weight-1 error magnitudes and their "one's" complements, it is necessary to locate the *bit index* i and to determine the sign of 2^i for the error value pairs $\{2^i, -2^{ka}+1 + 2^i\}$ and $\{-2^i, 2^{ka}-1 - 2^i\}$ which are caused by a one-use local fault during a transfer, "one's" complementation, shift or parallel addition. The modulo 2^a-1 residues of these error values are therefore of interest and will be determined. The length of ka bits is assumed for the operands, that is, the bit index i is in the range $0 \leq i \leq ka - 1$.

The error value $+2^i$ corresponds to a binary error number containing a single nonzero digit $e_i = 1$. Stating 2^i as a base 2^a number, we have

$$2^i = 2^{i-ja}(2^a)^j = 2^h\,(2^a)^j$$

The *intra-group index* $i-ja$ will be designated by h ($0 \leq h \leq a - 1$), and the index j will be called the

group index ($0 \leq j \leq k-1$). The check sum obtained for the value 2^i will contain a single "one" digit ($f_h = 1$) in the position $h = i - ja$, since 2^i is congruent to 2^{i-ja} modulo (2^a-1), and $(2^a-1) \mid 2^i = 2^h$; the other $a-1$ digits are zeros. Consequently, the check sum F for a faulty operand $Z^* = Z + 2^i$, where Z is any perfect operand, with $(2^a-1) \mid Z = Z'$, will be obtained as follows:

$$F = (2^a-1) \mid (Z + 2^i) = (2^a-1) \mid (Z' + 2^h)$$

The other error value $-(2^{ka}-1) + 2^i$ yields a faulty operand $Z^* = Z + 2^i - (2^{ka}-1)$, since the error caused an erroneous "end-around carry" in an addition; that is, $Z \leq (2^{ka}-1)$, but $(Z + 2^i) > (2^{ka}-1)$. The check sum is the same as for the error value 2^i above, since $(2^a-1) \mid (2^{ka}-1) = 0$ cancels the contribution of $-(2^{ka}-1)$.

Given the error value -2^i, its check sum will be computed as follows:

$$(2^a-1) \mid (-2^i) = (2^a-1) \mid (-2^{i-ja})$$
$$= (2^a-1) \mid (-2^h) = (2^a-1) - 2^h$$

where i, j, and the word length ka are as defined earlier. This check sum will contain only one "zero" digit $f_h = 0$ ($h = i - ja$); all $a-1$ other digits will be "ones." The faulty operand $Z^* = Z - 2^i$ will yield the check sum

$$F = (2^a-1) \mid (Z-2^i) = (2^a-1) \mid [Z' + (2^a-1) - 2^h]$$

where $Z' = (2^a-1) \mid Z$. When the error value -2^i inhibits a required "end-around carry," the resulting faulty operand Z^* has the value $Z^* = Z + (2^{ka}-1) - 2^i$, and the check value F is the same as above, since $(2^a-1) \mid (2^{ka}-1) = 0$. It is observed that errors in the "end-around carry," which may occur in the modulo $2^{ka}-1$ ("one's" complement) adder, do not affect the recognition of the intra-group index $h = i-ja$ or of the sign of the error values $\pm 2^i$.

The preceding analysis established the values for the modulo 2^a-1 residues of operands subjected to elementary faults which added one of four possible error values. These results apply to both product (An) and residue codes. The following analysis considers the details of the fault location problem for product, residue, and modified residue codes.

If the *product* (An) *codes* are employed to encode the operands, then the check algorithm consists of the modulo 2^a-1 summation of the k groups of a bits length. Every perfect operand value Z will yield the zero residue $Z' = 0$, represented by the "all ones" check result (e.g., "1111" for $a = 4$), except the "all zeros" operand which yields the "0000" check result. The faulty operand $Z^* = Z + 2^i$ will yield the residue 2^h which indicates that $i = ja + h$ ($j = 0,1,\cdots,k-1$); for example, 0100 indicates $h = 2$ when $a = 4$. The faulty operand $Z^* = Z - 2^i$ will yield the residue $(2^a-1) - 2^h$; for example, 1110 indicates $h = 0$ when $a = 4$.

If the *residue codes* are employed to encode the operands, then the modulo 2^a-1 summation is used to generate the residue of the result X

$$F(X) = (2^a-1) \mid X$$

The check processor separately computes the residue X', and the check algorithm is completed by the modulo 2^a-1 addition of the "one's" complement $(2^a-1) - X'$ of this residue to $F(X)$

$$F = (2^a-1) \mid [F(X) + (2^a-1) - X']$$
$$= (2^a-1) \mid [F(X) - X']$$

If an error has not occurred, then $F(X) = X'$, and $F = 0$ (represented by "1111") is the check sum. If the faulty value is in the result $X^* = X \pm 2^i$, and the residue X' is computed correctly, then the check algorithm gives the result with $F(X^*) = X' \pm 2^h$, and the check sum is

$$F = (2^a-1) \mid [F(X^*) - X'] = (2^a-1) \mid (X' \pm 2^h - X')$$
$$= (2^a-1) \mid (\pm 2^h)$$

If the error is in the residue $(X')^* = (2^a-1) \mid (X' \pm 2^h)$, and the result X is computed correctly, then the check algorithm gives the result with $F(X) = X'$, and the check sum is

$$F = (2^a-1) \mid [F(X) - (X')^*]$$
$$= (2^a-1) \mid [X' - (X' \pm 2^h)] = (2^a-1) \mid (\mp 2^h)$$

The sign of 2^h has been inverted here; that is, the check sum $F = 1000$ indicates the error $+2^{h+ja}$ with $h = 3$ if the error is in the result, but the error is $-2^h = -2^3$ if it occurs in computing the residue X' (the error $+2^3$ in the residue will give the check sum 0111). We note that for residue codes the sign information is not complete, since the fault may have occurred either in the main processor or in the check processor.

The *modified residue codes* use and operate with the residue complements $X'' = (2^a-1) - X'$ instead of the residues X' in the check processors. An error in computing X'' will give $(X'')^* = (2^a-1) \mid (X'' \pm 2^h)$. The check algorithm is completed by modulo 2^a-1 addition

$$F = (2^a-1) \mid [F(X) + (X'' \pm 2^h)] = (2^a-1) \mid (\pm 2^h)$$

because $F(X) + X'' = 2^a-1$ is the requirement of the modified residue codes. The sign information does not suffer the ambiguity which was observed for ordinary residue codes; it is a strong advantage of the modified residue code.

The preceding results demonstrate that the error values which are caused by local faults yield check sums which indicate the *sign* and the *intra-group index* $h = i-ja$, with $0 \leq h \leq a-1$, and $0 \leq i \leq ka-1$. The *group index* j, with $0 \leq j \leq k-1$ remains unknown; there are k acceptable choices of j. For complete location of the bit index i of the local fault (or for error correction) the group index j remains to be determined. It is noted that the concept of the intra-group index is also very useful in the analysis of repeated-use fault symptoms.

V. Complete Fault-Location by Multiple Low-Cost Encodings

The preceding discussion of partial fault-location suggests that the complete identification of the bit index i can be attained by using multiple groupings in such a way that each bit of the operand has a unique set of intra-group indices $\{h_1, h_2, \cdots, h_m\}$. Restricting attention to the low-cost codes with the check constants $A_i = 2^{a_i}-1$, $(a_i > 1)$, we note that the group lengths are a_i bits. The choice of two group lengths of a_1 and a_2 bits, respectively, will provide $a_1 \cdot a_2$ unique pairs of intra-group indices (h_1, h_2) with $0 \leq h_1 \leq a_1 - 1$ and $0 \leq h_2 \leq a_2 - 1$ when a_1 and a_2 have no common divisors. For example, $a_1 = 3$ and $a_2 = 4$ will yield twelve pairs of indices:

$$h_1 = \mid 2,1,0 \mid 2,1,0 \mid 2,1,0 \mid 2,1,0 \mid$$
$$h_2 = \mid 3,2,1,0 \mid 3,2,1,0 \mid 3,2,1,0 \mid$$

The same observation extends directly to sets of three and more group lengths $\{a_1, a_2, \cdots, a_m\}$ which have no common divisors. The length of the binary number with unique sets of intra-group indices $\{h_1, h_2, \cdots, h_m\}$ is then p bits, and s bits are required by the code, with

$$p = \prod_{i=1}^{m} a_i, \qquad \text{and } s = \sum_{i=1}^{m} a_i$$

For example, the choice of $a_1 = 3$, $a_2 = 4$, and $a_3 = 5$ will give $3 \cdot 4 \cdot 5 = 60$ unique sets of three intra-group indices, with $3 + 4 + 5 = 12$ bits used for encoding.

The application of multiple low-cost encoding for fault location will now be considered for both varieties of arithmetic codes. In *multiple product (An) codes* the encoding with m check factors gives the coded operand AX, where

$$AX = (2^{a_1}-1)(2^{a_2}-1) \cdots \cdot (2^{a_m}-1) \cdot X$$

Complete location of an elementary fault (single-error correction) is provided for the $p = a_1 \cdot a_2 \cdot \cdots \cdot a_m$ bits long encoded operand AX, which has $s = a_1 + a_2 + \cdots + a_m$ code bits and $p-s$ information bits. It is very important to note that the checking algorithm consists of m independent modulo $2^{a_i}-1$ checks $(i=1, 2, \cdots, m)$, therefore the low-cost checking is retained. Furthermore, only one modulo $2^{a_i}-1$ check is needed to detect the presence and sign of an elementary fault. The remaining checks need to be carried out only when a fault has been indicated by the first check, therefore the same checking hardware may be shared by all checks sequentially when single-error correction is performed.

The effect of the multiple low-cost product code with m check factors of check lengths $\{a_1, a_2, \cdots, a_m\}$ with respect to the location of one local fault [the error values $\pm 2^i$ and $\pm(2^n-1-2^i)$] is equivalent to the effect of a product code with a single check factor 2^p-1, which would require p check bits, while only s check bits are needed by the multiple product code. With respect to the detection of two local faults (double-error detection) the multiple product code detects all those weight-2 errors and their "one's" complements which are detected by the single check factor 2^p-1. With respect to a burst of local faults in b adjacent positions $i+1, \cdots, i+b$ (or burst errors of length b) and their "one's" complements, the multiple product code with $s = a_1 + a_2 + \cdots + a_m$ detects all bursts up to and including $b = s-1$ adjacent positions.

In summary it is noted that the local fault location and detection properties of multiple product codes with the preferred check factors $2^{a_i}-1$ give the effect of a check factor 2^p-1, while using only s check bits, and retaining the simple individual check algorithms for each check factor. The choice of coded word length $n = p$ gives

complete one-fault location, two-fault detection, and $s-1$ adjacent-fault detection in all algorithms in which the faulty element(s) are used once, including parallel transfer, shift, "one's" complementation and "one's complement" (modulo 2^n-1) addition. For example, $a_1 = 3$, $a_2 = 4$, and $a_3 = 5$ give $p = 60$ and $s = 12$ for the encoding of 48-bit operands in the triple low-cost product code.

The application of *multiple low-cost residue codes* follows very closely the results of the multiple low-cost product codes, except that here the check moduli $(A_1, A_2, \cdots, A_m)$ are chosen such that $A_i = 2^{a_i}-1$, and the a_i have no common divisors. The same parameters $p = a_1 \cdot a_2 \cdot \cdots \cdot a_m$ and $s = a_1 + a_2 + \cdots + a_m$ determine the effectiveness of the codes. The m separate check processors now compute the set of m residues X'_i for $i = 1,2,\cdots, m$. The checking algorithm generates the m modulo A_i residues of the main result X, designated as $F_i(X) = A_i \mid X$. The set of check sums F_i is obtained by adding the "one's" complements $A_i - X'_i$ to the $F_i(X)$, as described in the preceding Section IV for a single residue:

$$F_i = A_i \mid [F_i(X) + (A_i - X'_i)] = A_i \mid [F_i(X) - X'_i]$$

One check result is sufficient to detect the presence of a local fault, while the remaining check results will supply the remaining intra-group indices for the unique recognition of the bit index i. The use of more than one check modulus solves the problem of recognition whether the fault occurred in the main, or in the check processor; if only one check processor indicates the fault, it contains the fault itself; however, if all check processors indicate the fault, then it is traced to the bit i in the main processor by the set of intra-group indices. The sign ambiguity which existed in the case of one residue is now eliminated, and correction can be made either in the main result, or in the incorrect residue according to the rules of the preceding section.

Multiple low-cost modified residue codes are applicable in exactly the same manner as the ordinary residue codes; however, the advantage of unambiguous sign information noted in the preceding section for single encodings is now available for ordinary residue codes as well.

An important difference between multiple low-cost residue and multiple low-cost product codes is found in the length of the uncoded information word. The nonseparable product codes allow $p-s$ information bits, while the separable residue codes allow p information bits, with the s check bits being added on as separate check symbols. The residue codes with the same number of check bits provide the same fault-detection and location performance for a longer information word. The separability of the residue codes also permits a simpler design of the main arithmetic processor which deals with uncoded operands, rather than with multiples of the check constant. These advantages also exist for single-residue codes, but they become even more important when multiple codes are employed.

VI. Multiple Hybrid Codes for Fault Location (Error Correction)

The preceding discussion of multiple encodings was limited to the low-cost codes with $A = 2^a-1$, which have the advantage of a simple checking algorithm. A natural extension of multiple codes leads to *multiple hybrid codes* which employ a set of check constants $\{A_1, A_2, \cdots, A_m\}$ which includes one or more low-cost check constants A_i as well as one or more non-low-cost check constants (identified as A_j^*) with the properties of error correction (Refs. 7 and 9–13).

A multiple hybrid code (for example, the double code with A, A^*) offers two advantages over the use of the single non-low-cost check constant A^*. First, the low-cost code alone is sufficient for the detection of faults which are corrected by A^*, therefore only the low-cost (modulo $A = 2^a-1$) check algorithm is applied to every operand. Only in the case of a fault is it necessary to compute the modulo A^* residue of the operand which provides the fault-location (error-correction) information. Second, certain choices of the pairs (A, A^*) which are discussed below permit the use of the partial error-locating property of A in combination with the error-correcting property of A^*. The intra-group index h of the low-cost code which has the values $\{0, 1, \cdots, a\text{-}1\}$ can be applied to extend the range covered by A^*. For example, the choice of $A^* = 23$ gives distinct values of the residue $23 \mid (\pm 2^i)$ for $0 \leq i \leq 10$, and consequently identifies the index i and the sign for single-error correction (one-fault location) for an eleven-bit operand (Ref. 7). In combination with a given $A = 2^a-1$, the length of the operand with unique one-fault location becomes $11a$ if 11 and a have no common divisors.

In general, the following observations are made for the double hybrid codes (A, A^*) where $A = 2^a-1$ and A^* is an odd prime which has the "minimum distance 3" property (Ref. 9). First, if -2 (but not 2) is a primitive

root of A^*, then the distinct values of the residues $A^* \mid 2^i$ and $A^* \mid -2^i$ repeat with the period of $(A^*-1)/2$ and the low-cost A extends the length of the operand to $a(A^*-1)/2$ bits (as in the example with $A^* = 23$) as long as $(A^*-1)/2$ and a have no common divisors. Second, if 2, or both 2 and -2, are primitive roots of A^*, then the distinct values of the residues $A^* \mid 2^i$ and $A^* \mid -2^i$ repeat with the period of A^*-1 each. The low-cost A extends the length to $a(A^*-1)$ bits when A^*-1 and a have no common divisors and when the low-cost check determines the sign of the error value $\pm 2^i$. If a has common divisors with $(A^*-1)/2$ and A^*-1, respectively, in the two cases, then the length of the operand is equal to the least common multiple of a and $(A^*-1)/2$ or A^*-1. The use of more than one low-cost check constant $(A_1, A_2, \cdots, A_k)$ with some A^* will give the combined effect of the k-multiple low-cost code (as described in the preceding section) with the error-correcting properties of A^*, with the same constraints on common divisors as before.

The implementations of fault-location using both classes of hybrid codes (product and residue) remain to be discussed. The *multiple hybrid product codes* AA^*X follow closely the implementation of other varieties of product codes. The principal advantage is provided by the low-cost modulo A checking algorithms which provide partial location of the fault (error detection); therefore the costlier and slower modulo A^* check needs to be carried out only if the low-cost checks indicate a fault. The disadvantages are found in the much more difficult (costlier and slower) implementation of arithmetic algorithms which results when A^* is not a low-cost check factor.

The *multiple hybrid residue codes* avoid most of the difficulties of the product codes because the residue codes are separable. The use of more than one check modulus also resolves the problem of distinguishing whether the fault occurred in the main or in the check processor. In a double residue code with the check moduli (A, A^*) the low-cost modulo A check is carried out each time to indicate the presence of a fault. The modulo A^* check is carried out when a fault is indicated. If it does not indicate a fault, then the modulo A check processor is faulty, and fault location follows according to the results of Section IV. If the modulo A^* check processor also indicates a fault, then a fault in the main processor is suspected. The check sum for A^* is

$$F^* = A^* \mid [F^*(X) - (X')^*]$$

which is obtained the same way as for the multiple low-cost residue codes in Section V. Since the fault is indicated in the main processor, then the check processor output $(X')^*$ is assumed to be correct, and F^* is the modulo A^* residue of the error value $(\pm 2^i)$. The value of F^* together with the intra-group index h and the sign which were obtained in the modulo A check will supply the unique location and sign for the bit index i in the operand.

Special attention is required by the modulo N $(= 2^{ka}-1)$ addition algorithm in the main processor. When the check modulus A^* is not a multiple of N, then the subtraction of $2^{ka}-1$ (end-around carry) in the main processor must effect the subtraction of $A^* \mid (2^{ka}-1)$ in the modulo A^* check processor. An incorrectly caused end-around carry will be compensated in the modulo A^* check processor, and the modulo A check processor is erroneously identified as being faulty. This problem may be resolved by the use of two low-cost check moduli (A_1, A_2) along with A^*. This will permit a unique recognition of the above case (A_1 and A_2 processors indicate a fault, while A^* processor and the main processor agree).

It is necessary to note that an error in the modulo A^* check processor result will not be detected by the usual means when the modulo A^* check is not carried out each time. Alternate methods for error detection in the modulo A^* processor are needed. One possible solution is the repetition of the modulo A^* arithmetic operation with interchanged operands, since the operation time of the check processor is short with respect to the main processor. A duplication of the modulo A^* check processor or a switched conversion of the low-cost (modulo A) check processor to duplicate the modulo A^* operation also are feasible. However, these solutions do not detect errors in the modulo A^* residue arriving at the processor. A parity bit for the modulo A^* residue can solve this problem for transmission and storage. To check both storage and processing a short low-cost (for example, two bits for modulo 3) check residue may be employed to encode and check the modulo A^* residues. The problem of validating the modulo A^* encoding does not exist for the hybrid product codes.

An interesting variation of the hybrid codes are the *mixed* (product + residue) *hybrid codes*. Two attractive variants are: (1) low-cost product-coded operands AX with modulo A^* residue encoding, and (2) low-cost modified-residue (modulo A) encoding for product-coded operands A^*X. The first variant gives simple algorithms

in the main processor, but must resolve the problem (discussed for hybrid residue codes) of checking the error-correcting residue when the modulo A^* check is used only after detection using low-cost A. The second variant gives simple residue checking for fault detection, but requires rather complex algorithms in the main processor which operates on multiples of the non-low-cost check constant A^*. *Mixed low-cost multiple codes* with low-cost product and residue encodings are also possible and offer all desirable properties of low-cost codes. However, an advantage over multiple residue codes is not immediately apparent. A study of both varieties (low-cost and hybrid) of mixed codes is being continued with emphasis on differences in their cost of implementation and in their effectiveness compared to the uniform low-cost and hybrid multiple codes.

In conclusion it is noted that the proposed use of multiple low-cost and hybrid arithmetic encodings opens a considerable variety of attractive implementations. Fault location and error correction by means of multiple encodings employs the low-cost codes alone as well as the means to extend the range covered by error-correcting codes. Although the discussion of the latter codes was in terms of the "distance 3" check constants, the principles of multiple hybrid encodings apply to the large-distance codes (Refs. 11–13) as well. It is also important to observe that multiple encodings open the use of residue codes for error correction, since they distinguish whether the error is in the main operand, or in one of the check symbol residues. This information is not available with a single residue, and the generally less convenient product (An) codes had to be used.

References

1. Avižienis, A., "An Experimental Self-Repairing Computer", *Proceedings of the IFIP Congress 1968*, pp. 872–877. North-Holland Publishing Co., 1969.

2. Avižienis, A., "Concurrent Diagnosis of Arithmetic Processors", *Digest of the First Annual IEEE Computer Conference*, pp. 34–37, Chicago, Ill., Sept. 6–8, 1967.

3. Avižienis, A., *A Set of Algorithms for a Diagnosable Arithmetic Unit*, Technical Report 32-546. Jet Propulsion Laboratory, Pasadena, Calif., 1964.

4. Avižienis, A., *A Study of the Effectiveness of Fault-Detecting Codes for Binary Arithmetic*, Technical Report 32-711. Jet Propulsion Laboratory, Pasadena, Calif., 1965.

5. Avižienis, A., "Codes for Fault Detection in Digital Arithmetic Processors", *Information Processing 1965*, Vol. 2. W. A. Kalenich, Editor. Spartan Books, Inc., Washington, D.C., 1966.

6. Garner, H. L., "Generalized Parity Checking", *IRE Trans. Electron. Computers*, Vol. EC-7, pp. 207–213, 1958.

7. Brown, D. T., "Error Detecting and Correcting Codes for Arithmetic Operations", *IRE Trans. Electron. Computers*, Vol. EC-9, pp. 333–337, 1960.

8. Avižienis, A., "The Diagnosable Arithmetic Processor", *Supporting Research and Advanced Development*, Space Programs Summary 37-37, Vol. IV, pp. 76–80. Jet Propulsion Laboratory, Pasadena, Calif., 1966.

9. Peterson, W. W., *Error Correcting Codes*, pp. 236–244. The Massachusetts Institute of Technology Press, and John Wiley and Sons, Inc., New York, 1961.

10. Bernstein, A. J., and W. H. Kim, "Linear Codes for Single Error Correction in Symmetric and Asymmetric Computational Processes", *IRE Trans. Inform. Theory*, Vol. IT-8, pp. 29–34, 1962.

11. Chien, R. T., "On Linear Residue Codes for Burst-Error Correction", *IEEE Trans. Inform. Theory*, Vol. IT-10, pp. 127–133, 1964.

12. Barrows, J. T., Jr., *A New Method for Constructing Multiple Error Correcting Linear Residue Codes*, Report R-277. Coord. Science Lab., University of Illinois, Chicago, Ill., Jan., 1966.

13. Mandelbaum, D., "Arithmetic Codes with Large Distance", *IEE Trans. Inform. Theory*, Vol. IT-13, No. 2, pp. 237–242, Apr., 1967.

14. Rothstein, J., "Residues of Binary Numbers Modulo Three", *IRE Trans. Electron. Computers*, Vol. EC-8, p. 229, 1959.

15. Germeroth, J. H., "Casting Out Threes in Binary Numbers", *IRE Trans. Electron. Computers*, Vol. EC-9, p. 373, 1960.

NASA — JPL — Coml., L.A., Calif.