SU-SEL-72-0[illegible]

Separate Non-Homomorphic Checking Codes for Binary Addition

by

Stephen G. Kolupaev

July 1972

Technical Report No. 35

This work was supported by the
National Science Foundation
under Grant G J-27527

DIGITAL SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES

STANFORD UNIVERSITY · STANFORD, CALIFORNIA

SEL-72-033

SEPARATE NON-HOMOMORPHIC CHECKING CODES FOR BINARY ADDITION

by

Stephen G. Kolupaev

July 1972

Technical Report no. 35

DIGITAL SYSTEMS LABORATORY

Department of Electrical Engineering   Department of Computer Science

Stanford University

Stanford, California

This work was supported by the National Science Foundation under Grant GJ-27527.

ABSTRACT

In this paper, necessary and sufficient conditions for successful detection of errors in a binary adder by any separate code are developed. We demonstrate the existence of separate checking codes for addition modulo 2^n ($n \geq 4$) and modulo 2^n-1 ($n > 5$, n even), which are not homomorphic images of the addition being checked. A non-homomorphic code is constructed in a regular fashion from a single check symbol with special properties. Finding all such initial check symbols requires an exhaustive search of a large tree, and results indicate that the number of distinct codes for a particular modulus grows rapidly with n. In an appendix, we examine a modulo 2^n adder where the carry out of the high position is also presented to a checker.

TABLE OF CONTENTS

	Page
Abstract	i
Table of Contents	ii
Table of Figures	iii
Introduction	1
Detectable Errors	3
Necessary and Sufficient Conditions	6
Construction of a Code	9
Codes for Modulus $R=2^n$	12
Codes for Modulus $R=2^n-1$	13
Finding Other Codes	14
Some Examples	17
Conclusion	20
Appendix	21
References	24

LIST OF FIGURES

Page

1 Residue checking scheme 1

2 General separate checking scheme 2

3 a. Ripple adder 3

b. Carry bypass adder 3

4 Full adder with common circuitry for sum and carry out 5

5 Tuple of differences 16

6 The tree which is searched to find all 3 element sets from Z_{31} which have the composite distance property with themselves 17

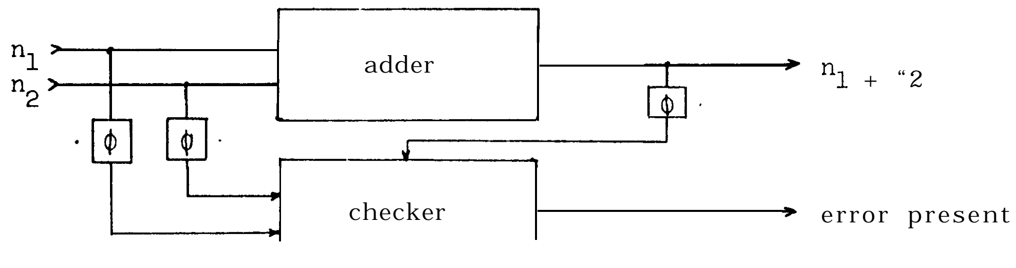
7 Check symbol encoding for Z_{32} regular code 19

SEPARATE NON-HOMOMORPHIC CHECKING

CODES FOR BINARY ADDITION

Introduction

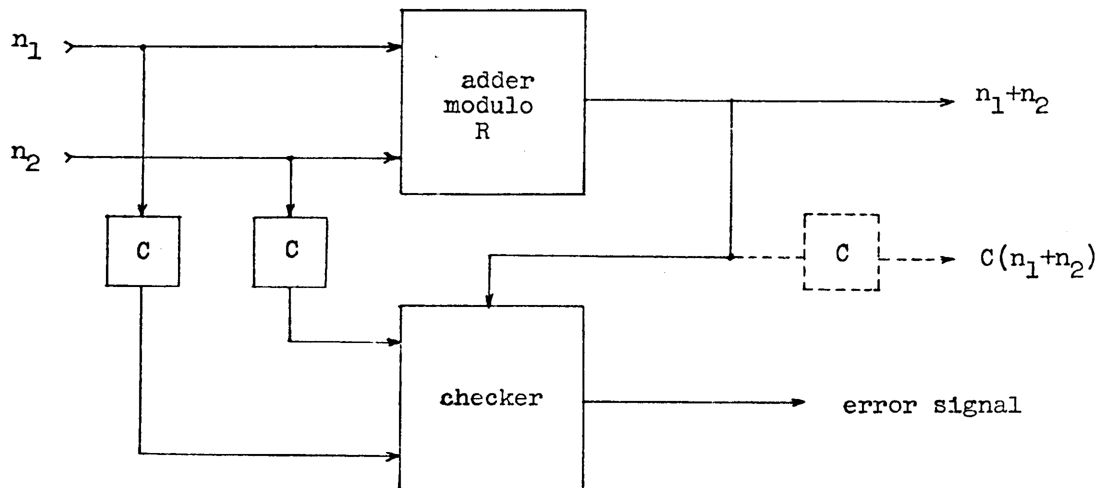
It has previously been believed that in order to check a binary adder using a separate code (check symbols), it is necessary to construct a code which is some homomorphic image of the addition operation being checked. This has been proven [1] for the checking scheme of Fig. 1



Fig, 1 Residue checking scheme

which requires $\phi(n_1)*\phi(n_2) = \phi(n_1+n_2)$, where * is some suitable check symbol operator. However, one may still retain the separateness of a checking code without performing the same check symbol transformation (ϕ) on the adder output, as is shown in Fig, 2. Here we present the checker with the full binary

output number from the adder, and deduce properties for the check symbol map C which will allow a checker to detect any of a set of detectable errors, The classical separate checking scheme is included as a special case of the generalized scheme of Fig. 2, since the output of the adder may be transformed by the C mapping inside the checker, and before the actual check operation takes place,



Fig, 2 General separate checking scheme

The scheme of Fig. 1 has the advantage that the check symbol for the adder output $\phi(n_1+n_2)$ is available at the output also, It could then be stored in memory along with the sum n_1+n_2, at some extra memory cost, Then the check symbols for the inputs $\phi(n_1)$ and $\phi(n_2)$ would presumably have been stored in memory along with n_1 and n_2, and the input check symbol transformations would be unnecessary. The same method of storing the check symbol along with the number can be used with the scheme of Fig. 2 by adding a check symbol generator C at the output. However the check symbol would not be checked by the checker since the checker does not examine the output check symbol, but rather the output itself, Consequently a failure in the output check symbol generator would be undetected until the output number was fetched back from memory as an input operand for the adder,

Detectable Errors

We must decide exactly which errors are to be detectable, It is, of course, most effective if we choose the most likely errors to be the detectable errors. The errors which are most likely to occur are those arising from the least number of simultaneous failures in the adder hardware (a single fault). We consider the ripple-carry adder where sum and carry out of each digit position are determined by a full adder, and the full adders for the various digit positions are independent. Other widely used adders employ the carry-bypass and the carry look-ahead techniques. The ripple adder and carry-bypass adder (Fig. 3) may be treated in the same way, since a single fault in the carry bypass circuitry can only affect the carry out of the group, while a faulty interdigit carry from a full adder propagates in the same way for both adders. For the carry look-ahead adder, such is not the case. The look-ahead adder will not be treated here, for there are several implementations, each affected differently by single faults [2]. The error caused by a single fault in the carry circuitry for the ripple adder and carry bypass adder has the effect of adding or subtracting a simple power of two, as is shown below.

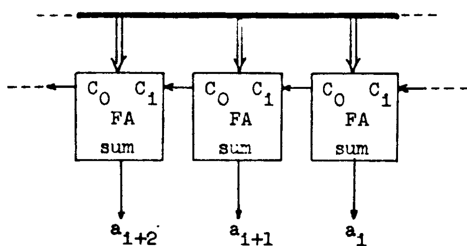


Fig-3a. Ripple adder.

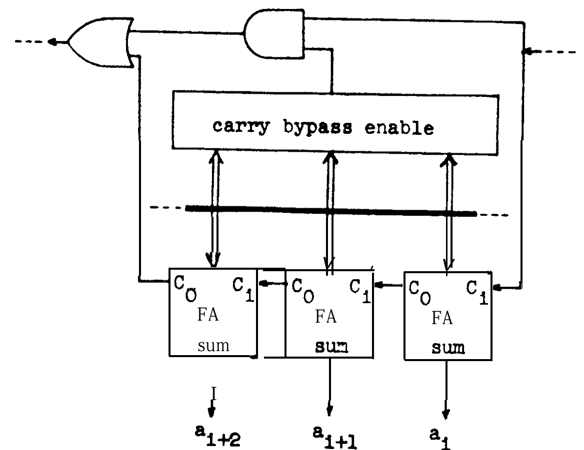


Fig. 3b. Carry bypass adder

In this paper we consider the adder to be performing addition modulo R on two inputs, the inputs being binary encodings of two integers, which are also modulo R ($R=2^n$ or $R=2^n-1$). The adder delivers the sum in the same binary-encoded form,

$$T = \sum_{k=0}^{n-1} a_k * 2^k \quad .$$

In the study of transmission codes, the severity of a transmission error is measured by how many bits are inverted (incorrect), since a single error affects only one bit, However, a single fault in a binary adder may affect several bits in its output, and in different ways depending on the input operands to the adder, When studying arithmetic codes for adders, it is more useful to measure the severity of an arithmetic error by the nature of the arithmetic difference between the correct adder output and the adder output under the fault, The reason is that the severity of the hardware failure causing the fault is most closely related to the arithmetic difference.

Definition: An <u>additive error</u> s has occurred in a modulo R adder when the erroneous adder output t' is related to the correct adder output t by

$$t' = (t+s) \bmod R, \text{ where } s \in Z_R.$$

Thus if a failure changes bit ak from 0 to 1, the erroneous sum is $t'=t+2^k$. Similarly, if a_k changes from 1 to 0 the erroneous sum is $t'=t-2^k$. Note that subtracting 2^k is the same as adding $R-2^k$ or $(-2^k) \bmod R$. Thus the additive error is $(R-2^k)$. If an interdigit carry or intergroup carry out of the kth position changes from 0 to 1 or 1 to 0, the erroneous sums become $t'=t+2^{k+1}$ and $t'=t-2^{k+1}$, respectively. Finally, if both the sum bit and carry out bit from the kth full adder are inverted, the erroneous sum becomes $t'=t \pm 2^k \pm 2^{k+1}$, depending on the nature of the double inversion. The error $t'=t \pm 3*2^k$, which occurs when the sum and carry out bits are inverted with the same sense, would be most troublesome.

Indeed, most studies of error detection in binary adders do not specifically allow for detecting this sort of error, However, all the MSI full adders which we have examined are equivalent to the full adder shown in Rig 4. When the carry out is used to generate the sum output in this way, we can show that any failure in the carry circuitry causes an error which is a simple power of two.

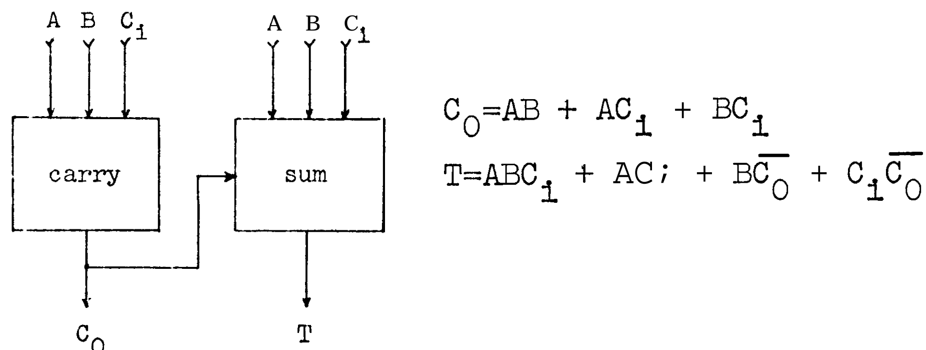


Fig. 4 Full adder with common circuitry for sum and carry out.

Suppose a failure in the carry circuit changes C_0 from 0 to 1, while the sum circuit is fault-free. Referring to the Boolean expression for the sum output T in Fig. 4, if A, B, and C_i are such that T changes under the fault, T can change only from 1 to 0, Similarly if C_0 changes from 1 to 0, T can only change from 0 to 1. Thus a single failure in the carry circuit of the kth full adder can only cause an additive error of 2^{k+1}, $2^{k+1}-2^k$, -2^{k+1}, or $-2^{k+1}+2^k$. Simplifying, these errors are 2^{k+1}, 2^k, -2^{k+1}, and -2^k. Any single failure in such a full adder (except power leads) gives an additive error of $(\pm 2^i) \bmod R$. There is no possibility of an additive error which is not a simple power of two,

With this justification, we shall require that the checking code detect all additive errors in the set S,

$$\text{where } S \triangleq \{s \mid s=2^i, \text{ or } s=R-2^i, i\in\{0,1,2, \dots, n-1\}\}$$
$$\text{and } n \text{ is the least integer } \ni n \geq \log_2 R.$$

Example: For Z_{32}, R=32, n=5.
Then S={1,2,4,8,16,24,28,30,31}.
For s=28=(-4)mod 32, the 2^2 bit has changed from 1 to 0.

Considering then the modulus R and the set of detectable errors S, we deduce necessary and sufficient conditions for the check symbol mapping C to allow a checker to detect the occurrence of any of the additive errors in S. This leads directly to the checking scheme, and an algorithm for finding all check symbols which meet the necessary conditions for inclusion in a check symbol map C. The check symbols found must then be combined into a single-valued map C, and the question of whether this can always be done efficiently is left unanswered. This algorithm will discover, in passing, any residue codes which exist for the modulus R. We discover a particular mapping C of the same general form for all moduli $R=2^n$, $n \geq 4$, and a similar mapping for all moduli $R=2^n-1$, $n>5$, n even.

Necessary and Sufficient Conditions

We make only the following initial restrictions on the checking code: we restrict the check symbol transformation C such that $C(n_1)$ and $C(n_2)$ provide the checker with sufficient information about the inputs to the adder, such that it can detect the occurrence of any error in the set S of detectable additive errors. We require that C be a single-valued mapping from the integers modulo R to check symbols

$$C: Z_R \rightarrow \{H_0, H_1, H_2, \ldots, H_{m-1}\}.$$

The checker is presented with two check symbols H_i and H_j which are derived from the binary inputs to the adder. The check symbols H_i are equivalent to sets B_i of integers, where B_i is the set of all integers modulo R whose binary encodings map under C to the ith check symbol H_i:

$$B_i \triangleq \{x \in Z_R \mid C(x)=H_i\} .$$

It will be much more convenient throughout to describe the code in terms of these equivalence classes B_i. Now suppose that the adder is given two integers x and y to add,

where $C(x)=H_i$,
$C(y)=H_j$.

All that the checker knows about the adder inputs is that one input was an integer in set B_i, and one was in set B_j. Then the checker must accept as the correct sum all the integers

$$t \in \{(a+b) \bmod R \mid a \in B_i,\ b \in B_j\}.$$

These sums arise from all the possible input pairs a,b which produce the check symbol pair H_i, H_j. We formalize this sum set with the binary set operator$\oplus$, which forms the set of all possible sums modulo R of one integer taken from the first set and one integer taken from the second set:

$$A \oplus B \triangleq \{(a+b) \bmod R \mid a \in A,\ b \in B\}.$$

Then the checker must accept as the correct sum all the integers $t \in B_i \oplus B_j$ when the check symbols presented are H_i and H_j. Likewise, the checker would deliver an error indication when $t \notin B_i \oplus B_j$. Thus the operation of the checker must be to verify that the sum t is in the set $B_i \oplus B_j$ defined by the check symbols H_i and H_j which are presented to it,

Lemma 1: Given a separate checking scheme for an adder of binary encoded integers modulo R, where the checker examines the adder output directly, and receives two check symbols H_i and H_j derived from the adder inputs, Then the checking scheme is capable of detecting the occurrence of any single additive error s taken from a set S of detectable errors iff

$$\forall H_i, H_j,\ \forall x,y \in B_i \oplus B_j,\quad (x-y) \bmod R \notin S$$

where $C\colon\ Z_R \to \{H_0,\ H_1,\ \ldots,\ H_{m-1}\}$

$$B_i \triangleq \{x \in Z_R \mid C(x)=H_i\}$$

$$A \oplus B \triangleq \{(a+b) \bmod R \mid a \in A,\ b \in B\}.$$

Proof

$\Leftarrow$ (sufficiency)

Let $a,b \in Z_R$ be inputs to the adder, where

$$C(a)=H_i, \quad C(b)=H_j.$$

Let the check symbol map C have the property

$$\forall H_i,H_j \in Im(C), \forall x,y \in B_i \oplus B_j, \quad (x-y) \bmod R \notin S.$$

Let the checker accept as correct all the integers in $B_i \oplus B_j$, and reject the others, Then by the definition of $\oplus$, the correct sum

$$t = (a+b) \bmod R \in B_i \oplus B_j.$$

But $\nexists$ $t' \in B_i \oplus B_j$ such that $(t'-t) \bmod R \in S$.
Then $\forall s \in S$, $t'=(t+s) \bmod R \notin B_i \oplus B_j$, and all the errors in S are detectable.

QED

$\Rightarrow$ (necessity)

Let the checker be presented with input check symbols H_i and H_j. Then the checker must accept as correct all the integers

$$t \in B_i \oplus B_j, \text{ since } \forall t \in B_i \oplus B_j, \exists\, a,b \in Z_R \text{ such that } C(a)=H_i,\ C(b)=H_j, \text{ and } (a+b) \bmod R = t.$$

But suppose for some $x,y \in B_i \oplus B_j$, $\exists\, s \in S$ such that

$$s=(x-y) \bmod R.$$

Then $\exists\, a,b,c,d \in Z_R$ such that $C(a)=H_i$, $C(b)=H_j$, $C(c)=H_i$, $C(d)=H_j$, $(a+b) \bmod R=x$, $(c+d) \bmod R=y$, where $x=(y+s) \bmod R$. Then if c,d are inputs to the adder, the checker must accept either x or y as the correct sum, while x is an incorrect answer resulting from the error s. In this instance the error s may go undetected, which contradicts the error detecting capability required. Thus by contradiction we must have

$$\forall H_i,H_j \in Im(C), \forall x,y \in B_i \oplus B_j, \quad (x-y) \bmod R \notin S.$$

QED

For convenience we define the property required by Lemma 1 as the composite distance property.

Definition: The sets A and B of integers modulo R have the <u>composite distance property</u>, $A \square B$, relative to the Set S of detectable additive errors iff $\forall x,y \in A \oplus B$, $(x-y) \bmod R \notin S$.

The composite distance property is always relative to a modulus R and set S of detectable errors. The modulus and set of detectable errors will not be mentioned whenever their presence is clearly understood.

With this definition, the code mentioned in Lemma 1 can detect any of the detectable errors iff $\forall B_i, B_j,\ B_i \square B_j$. The composite distance property is a convenient and compact criterion for selecting check symbol equivalence classes while constructing a code.

For example, consider the integers Z_{32}, The set of integers $\{0,3,6,9\}$ could be used as a check symbol equivalence class B_0 in a separate checking code for an adder modulo 32 since

$$S = \{1,2,4,8,16,24,28,30,31\}$$
$$B_0 = \{0,3,6,9\}$$
$$B_0 \oplus B_0 = \{0,3,6,9,12,15,18\}$$
$$\{(x-y) \bmod 32 \mid x,y \in B_0\} = \{18,15,12,9,6,3,0,14,17,20,23,26,29\}.$$

This set of differences has an empty intersection with S, so $B_0 \square B_0$.

Construction of a Code

To construct a code for a particular modulus R and its associated set of forbidden distances S, we must divide the integers Z_R into equivalence classes $B_0, B_1, \ldots, B_{m-1}$ such

that each pair B_i, B_j has the composite distance property (Lemma 1). We will not be concerned with the implementation of the check symbol transformation C or with the operation of the checker here, since they may be implemented by table look-up, if by no other means. The implementation for a special code which recurs in the same form for all moduli 2^n will be discussed in a subsequent section. Once the check symbol transformation C is defined, the structure of the code is completely specified.

Once a set B of integers modulo R has been found which has the composite distance property with itself, several other sets may be found directly from B which have the composite distance property with B, and with each other.

Definition: B' is a rotation of the set B of integers modulo R iff $\exists\, k \in Z_R$ such that $B' = B \oplus \{k\}$.

Example: $\{1,4,7,10\}$ is a rotation of $\{0,3,6,9\}$ for R=32.

Lemma 2 Given two sets A,B of integers modulo R (not necessarily distinct). If $A \square B$ then $A' \square B'$ where A' is a rotation of A and B' is a rotation of B.

Proof

Since $A \square B$, then $\forall a,b \in A \oplus B$, $(a-b) \bmod R \notin S$. That is $\forall a_1, a_2 \in A$, $\forall b_1, b_2 \in B$,

$$((a_1+b_1)-(a_2+b_2)) \bmod R \notin S.$$

Now consider

$$A' \oplus B' = \{(a'+b') \bmod R \mid a' \in A',\ b' \in B'\}$$
$$= \{((a+k_1)+(b+k_2)) \bmod R \mid a \in A,\ b \in B\}$$
$$\text{for some } k_1, k_2 \in Z_R.$$

The difference mod R of any two integers in $A' \oplus B'$ is

$$((a_1+b_1+k_1+k_2)-(a_2+b_2+k_1+k_2)) \bmod R$$
$$= ((a_1+b_1) - (a_2+b_2)) \bmod R, \text{ for some } a_1, a_2 \in A,\ b_1, b_2 \in B.$$

But this is identical to the difference mod R of two

integers in $A \oplus B$, and since $A \sqcap B$, that difference is not one of the forbidden distances in S. Then $\forall x,y \in A' \oplus B'$, $(x-y) \bmod R \& S$ and $A' \sqcap B'$.

QED

This suggests building a code from a single basic set B, as B, $B \oplus \{i_1\}$, $B \oplus \{i_2\}$, ... , $B \oplus \{i_k\}$, where each of the numbers in ZR occurs in exactly one of the sets B_i. This can be easily done for the "regular" codes which we will define shortly, In other cases, difficulties may arise with duplication of numbers in different sets B_i, in which case one can remove numbers at will from $B \oplus \{i\}$ and leave only the subset desired, It is obvious that doing so will not harm the composite distance property. It is also possible to build a code with any rotations of distinct non-overlapping sets, so long as the sets have the composite distance property with themselves and with each other,

When constructing a code from a known set of check symbol equivalence classes $\{B_i\}$, the situation becomes rather chaotic, for we have been able to impose no other restriction upon the choice of equivalence classes than that of Lemma 1. For example, one might suppose that $\sqcap$ is transitive, i.e. $A \sqcap B$, and $B \sqcap C \Rightarrow A \sqcap C$. We can immediately dispense with this proposition by the counterexample from Z_{64}

$$S=\{1,2,4,8,16,32,48,56,60,62,63\}$$

$A=\{0,9,18\}$ $\quad A \oplus A=\{0,9,18,27,36\} \Rightarrow A \sqcap A$

$B=\{1,4,7\}$ $\quad B \oplus B=\{2,5,8,11,14\} \Rightarrow B \sqcap B$

$C=\{2,25,48\}$ $\quad C \oplus C=\{4,9,27,32,50\} \Rightarrow C \sqcap C$

$A \oplus B=\{1,4,7,10,13,16,19,22,25\} \Rightarrow A \sqcap B$

$B \oplus C=\{3,6,9,26,29,32,49,52,55\} \Rightarrow B \sqcap C$

$A \oplus C=\{2,11,20,25,34,43,48,57\}$

but $(43-11)=32 \in S$ so $\quad A \not\sqcap C$

Further, one might suppose that $A \sqcap B$ iff $A=B \oplus \{k\}$, i.e. A has the composite distance property with B only if (we proved the if part in Lemma 2) A is a rotation of B.

But B and C of the example above. have the composite distance property, yet they are not rotations of each other,

Codes for Modulus $R=2^n$

For a modulus $R=2^n$, $n \geq 4$, we show that the set

$$B = \{0,3,6, \ldots , 3*(2^{n-3}-1)\}$$

has the composite distance property with itself, and that

$$C: Z_R \rightarrow \{B, B\oplus\{2^{n-3}\}, B\oplus\{2*2^{n-3}\}, \ldots , B\oplus\{7*2^{n-3}\}\}$$

constitutes a code, This is the "special form'" or '"regular" code mentioned above, For this modulus, the set of forbidden distances (detectable additive errors) is

$$s = \{1,2,4, \ldots , 2^{n-1}, 2^n-2^{n-2}, 2^n-2^{n-3}, \ldots , 2^n-1\}$$

But $B\oplus B=\{0,3,6,9, \ldots , (2^{n-3}-1)*3*2\}$, and the distance between any two numbers in $B\oplus B$ is a multiple of 3, and in fact does not exceed the $(2^{n-3}-1)*2^{th}$ multiple of 3. The smallest forbidden distance which might be a multiple of 3 is 2^n-2^{n-2}. However

$$(2^{n-3}-1)*3 < 2^{n-3}*3$$
$$(2^{n-3}-1)*3*2 < 2^{n-2}*3 = 2^n-2^{n-2}$$

Then the smallest forbidden distance which is a multiple of 3 must be still greater than the largest distance in $B\oplus B$. Then B has the composite distance property with itself.

Moreover, any one of the B_i is a rotation of any other, so all possible pairs B_i, B_j have the composite distance property,, Further, any number $n \in Z_R$ is mentioned in exactly one of the B_i, for assuming the converse leads to a contradiction as follows

> Suppose $\exists n \in Z_R \ni n \in B_i,\ n \in B_j,\ B_j=B_i\oplus\{k*2^{n-3}\},\ 1 \leq k \leq 7$.
> Then $(n-k*2^{n-3}) \in B_i$, and B_i has distance $k*2^{n-3}$.
> But by construction of B, B_i has only distances
>
> $m*3$, where $1 \leq m \leq 2^{n-3}-1$.

Consequently, for some m, $m*3 = k*2^{n-3}$, and 3 divides k, $1 \leq k \leq 7$

Case 1	Case 2
$k=3$	$k=6$
$m*3=3*2^{n-3}$	$m*3=6*2^{n-3}$
$m=2^{n-3}$	$m=2^{n-3}$
contradiction	contradiction

This completes the proof, and we have a code of the same form for all moduli $R=2^n$, $n \geq 4$.

Codes for Modulus $R=2^n-1$

For a modulus $R=2^n-1$, n even, we show that the set

$$B = \{0,3,6, 000, (2^{n-3}-2)*3\}$$

has the composite distance property with itself, and that the set

$$\{B,\ B\oplus\{1\},\ B\oplus\{2\},\ B\oplus\{k+3\},\ B\oplus\{k+4\},\ B\oplus\{k+5\},\ B\oplus\{2k+6\},\ B\oplus\{2k+7\},\ B\oplus\{2k+8\}\}$$

where $k = (2^{n-3}-2)*3$

can form a code after an ad hoc procedure of truncating duplicated numbers from the last 3 sets,

Again, the distance between any two numbers in $B\oplus B$ is a multiple of 3, and does not exceed the $(2^{n-3}-2)*2^{th}$ multiple. The set of forbidden distances is

$$S = \{1,2,4, 000, 2^{n-1}, 2^n-1-2^{n-1}, 2^n-1-2^{n-2}, \text{etc.}\}.$$

Now since n is even, 3 does not divide $2^{n-1}-1=2^n-1-2^{n-1}$. Then the smallest forbidden distance which may be a multiple of 3 is $2^n-1-2^{n-2}=3*2^{n-2}-1$. One can easily show that $3*2*(2^{n-3}-2) < 3*2^{n-2}-1$, so no pair of numbers in $B\oplus B$ differs by a forbidden distance, Thus B has the composite distance property with itself,

We can construct the nine rotations of B as above, and since (the largest number in the sixth rotation) $2k+5 = 2(2^{n-3}-2)*3+5 < 2^n-1$, the first 6 rotations mention the numbers 0,1, 000 , 2k+5 once and only once, Since $2k+8 < 2^n-1$, we can begin using the last 3 rotations of B. Since they are successively offset by 1, there is no overlap between them, and it only remains to truncate away any high order members of the 3 sets which wrap around the modulus $R=2^n-1$.

Finding Other Codes

A code is the division of the integers ZR into several non-overlapping sets, where any set has the composite distance property with any other set, and with itself, So the first step in finding a new code is finding a set B having the composite distance property with itself, Then a code may be constructed using only rotations of the same set, Also, if two distinct sets can be found, both being self-composite, and having the composite distance property with each other, a code may be constructed using any rotations of the two starting sets, This may be extended to any number of original sets,

We have devised a computer program which will find all self-composite sets with a given number of elements and containing the element 0, The sets are constructed by adding new elements one by one to a self-composite set with less than the required number of elements. The new elements are appended in ascending numerical order. All-the sets found, including the intermediate ones, are assigned to the nodes of a tree, The tree has as many levels as elements in the sets to be constructed, for the algorithm starts at the top of the tree (level 1), with the single element 0 in a starting set, The next level contains all sets of 2 elements, and so on, until the lowest level of the tree has all sets of the desired number of elements. The nodes are arranged so that any node is contained in its successors. Thus, each successor is formed by appending a new element

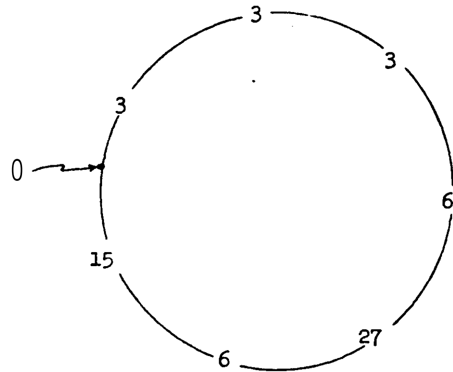
to the elements of its immediate predecessor, One can see that this is a huge tree, It is searched depth-first for nodes having the composite distance property with themselves, and the downward search stops as soon as a node is reached which is not self-composite, For, if this is true, no successor node can be self-composite, The search then continues at another node on the same level, This searching algorithm was implemented recursively in ALGOL W. It was later coded in IBM 360 assembly language, increasing speed approximately tenfold, Even so, 7 or 8 is the largest number of levels which can be searched exhaustively in a reasonable length of time.

This algorithm has the fault that several representatives of what is essentially the same set may be discovered. In Particular, a set may be represented by the differences between adjacent-numbers in the set. Then, depending on the degree of asymmetry of the tuple of differences, there may be up to 2*p distinct unordered sets 'of integers corresponding to the same tuple of differences, where p is the number of elements in the set. These are the various rotations and flips of the geometric shape of the tuple of differences, All of the sets arising from a particular ordered tuple of differences are self-composite or not, in the same way, since it is immaterial which element is rotated to the zero position,

For example, the set $B=\{0,3,6,12,39,45,60\}$ for R=63 was found to have the composite distance property with itself, The ordered tuple of differences of adjacent elements for this set is $\langle 3,3,6,27,6,15,3\rangle$. The geometric shape of Fig, 5 can be flipped and rotated in 14 ways to produce all the sets

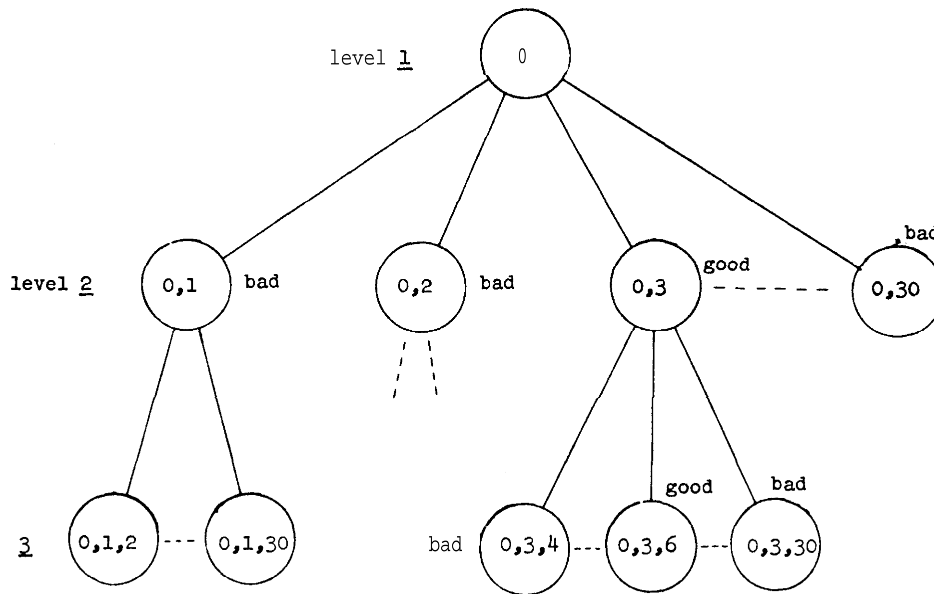
{0,3,6,9,15,42,48}
{0,3,6,12,39,45,60}
{0,3,9,36,42,57,60}
{0,6,33,39,54,57,60}
{0,27,33,48,51,54,57}
{0,6,21,24,27,30,36}
{0,15,18,21,24,30,57}

{0,3,6,9,24,30,57}
{0,3,6,21,27,54,60}
{0,3,18,24,51,57,60}
{0,15,21,48,54,57,60}
{0,6,33,39,42,45,48}
{0,27,33,36,39,42,57}
{0,6,9,12,15,30,36}.



Fig, 5 Tuple of differences

Unfortunately, there is no known algorithm for generating the distinct tuples of differences under rotation and flipping which may be formed by rearranging a given set of integral differences, The best we can do is enumerate them via Polya's counting theory,



Fig, 6 The tree which is searched to find all 3 element sets from Z_{31} which have the composite distance property with themselves

Some Self-Composite Sets Which Have Been Found

# elements	modulus R	prototype set	# rotations and flips
3	31	0,3,6	3
3	31	0,3,17	3
3	31	0,6,12	3
3	31	0,7,14	3
3	31	0,7,19	3
4	31	none exist	
4	32	0,3,6,9	4
7	63	over 2000 found	
8	64	0,3,6,9,12,15,18,21	8

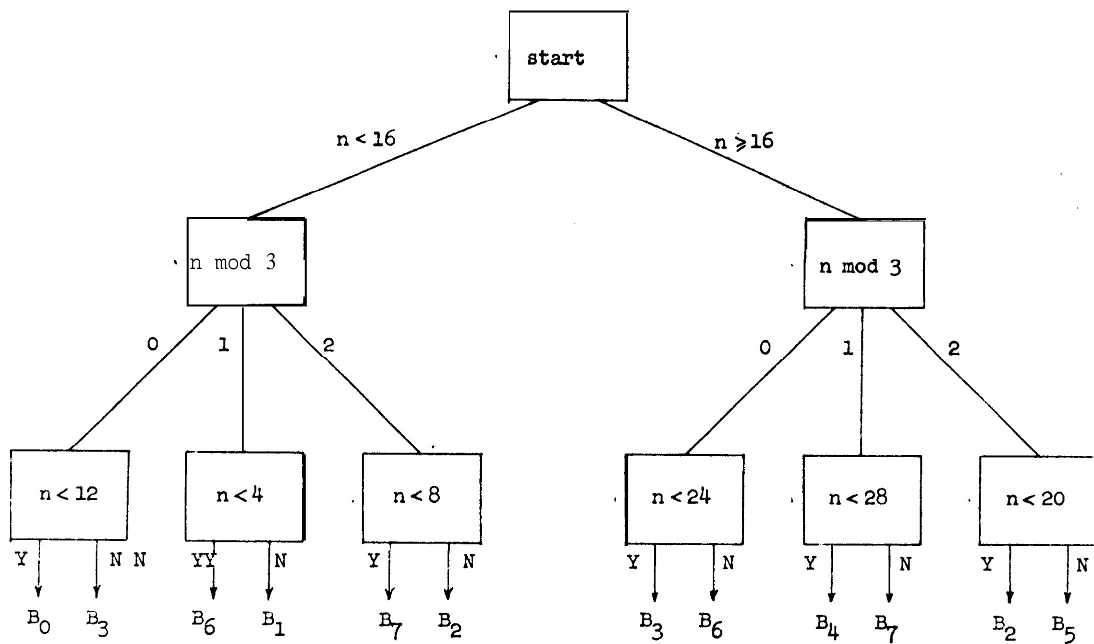
Implementing the Regular Form Code for Z_{32}

check symbol equivalence class	sum sets
$B_0=\{0,3,6,9\}$	$B_0 \oplus B_0 = B_6 \oplus B_2 = \ldots = \{0,3,6,9,12,15,18\} = A_0$
$B_1=\{4,7,10,13\}$	$B_1 \oplus B_0 = B_4 \oplus B_5 = \ldots = \{4,7,10,13,16,19,22\} = A_1$
$B_2=\{8,11,14,17\}$	$\{8,11,14,17,20,23,26\} = A_2$
$B_3=\{12,15,18,21\}$	$\{12,15,18,21,24,27,30\} = A_3$
$B_4=\{16,19,22,25\}$	$\{16,19,22,25,28,31,2\} = A_4$
$B_5=\{20,23,26,29\}$	$\{20,23,26,29,0,3,6\} = A_5$
$B_6=\{24,27,30,1\}$	$\{24,27,30,1,4,7,10\} = A_6$
$B_7=\{28,31,2,5\}$	$\{28,31,2,5,8,11,14\} = A_7$

We have listed the check symbol classes $B_0, B_1, \ldots, B_7$ and then listed the sum sets $B_i \oplus B_j$. Only 8 distinct sum sets occur, and they overlap. For example, 8 appears in both A_2 and A_7. Thus the sum 8 might result from the summands 4 and 4 from B_1 and B_1 (sum in A_2), or it might result from the summands 6 and 2 from B_0 and B_7 (sum in A_7). Note that the elements within any A_i do not differ by a forbidden distance (detectable additive error), as indeed they must not, The distances in any A_i are all multiples of 3, so the errors which may be detected in this case are $\pm 1, \pm 2, \pm 4, \pm 5, \pm 7, \pm 8, \pm 10, \pm 11, \pm 13, \pm 14$, and ± 16. These include several double errors, and the triple errors $\pm 11, \pm 13$.

The sum sets have been named so that if the check symbol classes from the input are B_i, B_j, then the output sum from the adder should lie in $A_{(i+j) \bmod 8}$. Then to perform the check, the checker examines the adder output and verifies that the sum is in the correct sum set, The sum may also lie in another sum set, but this is immaterial.

The processes of check symbol mapping and output verification are similar. A decision tree for the input encoding is shown in Fig. 7.



Fig, 7 Check symbol encoding for Z_{32} regular code

The adder output verification for the Z_{32} regular code is given in the following table,

sum set	residue condition	range
A_0	n mod 3=0	$n < 21$
A_1	n mod 3=1	$1 < n < 25$
A_2	n mod 3=2	$5 < n < 29$
A_3	n mod 3=0	$n > 9$
A_4	n mod 3=1	$n > 13$
A_4	n mod 3=2	$n > 5$
A_5	n mod 3=2	$n > 17$
A_5	n mod 3=0	$n < 9$
A_6	n mod 3=0	r-021
A_6	n mod 3=1	$n < 13$
A_7	n mod 3=1	$n > 25$
A_7	n mod 3=2	$n < 17$

Similar constructions hold for the regular (0,3,6, etc.) codes for moduli 2^n, all $n > 5$. The encoding and checking processes shown here are necessarily more complex than the homomorphic residue codes. However, this code is not necessarily representative of all the non-homomorphic codes, and further investigation is needed before making a comparison to the residue codes, Also of interest here is the fact that there is no homomorphic residue code for checking addition modulo 2^n which will detect all single errors. This is so because the residue must be taken modulo a power of two, which corresponds to a single error which is not detectable, A residue code can only be used for a modulo 2^n adder if the carry out or extend bit is given to the checker, so that the adder can be considered as modulo 2^{n+1}, with no possibility of overflow, This technique is investigated in Appendix 1.

Conclusion

We have. described non-homomorphic separate codes for checking modulo addition, for the important moduli 2^n, and for 2^n-1 with even n. These codes have essentially the same form for any suitable value of n larger than 5. This was done mainly to establish their existence, The most general non-homomorphic codes do not have this systematic form, and we expect them to vary widely, as they arise from number theoretic arguments, rather than from a defining algebraic structure.

The process of finding a non-homomorphic code is exhausting, even for a fast computer, We are currently improving the computer search technique, in order to exhaustively examine several representative moduli. This must be done in order to make meaningful comparisons between the cost-effectiveness of non-homomorphic separate checking codes and the more well-known methods,,

APPENDIX

Checking an adder where carry out of the highest bit position is available

In a modulo 2^n adder, the carry out of the highest bit position is usually available, If that bit is also presented to the checker, the checking code can be simplified considerably, In effect, the checker is then checking a modulo 2^{n+1} addition, where the adder inputs are restricted to be smaller than 2^n. Then the sum never wraps around the modulus 2^{n+1}, so we may assume that the adder is not modulo any number, but rather performs ordinary addition, The development of a separate checking code proceeds in mubh the same fashion,

<u>Additive error</u>: An additive error s is said to have occurred if the adder output under fault t' is related to the correct adder output t by

$$t' = t + s \ , \text{ where } -2^n \leq s \leq +2^n.$$

<u>Detectable errors</u>: $S = \left\{ \pm 2^i \mid i \in \{0, 1, \ldots, n\} \right\}$

<u>Check symbol map</u>: $c: \quad Z_{2^n} \rightarrow \{H_0, H_1, 000, H_{m-1}\}$

<u>Check symbol equivalence classes</u>: $B_i \triangleq \left\{ x \in Z_{2^n} \mid C(x) = H_i \right\}.$

<u>Set operator $\oplus$</u>: $A \oplus B \triangleq \left\{ a+b \mid a \in A, \ b \in B \right\}.$

<u>Error detection</u>:

Checker indicates error if sum $t \notin B_i \oplus B_j$

Checker indicates no error if sum $t \in B_i \oplus B_j$

<u>Necessary and sufficient conditions</u> on C for the checker to detect the occurrence of any error in set S:

$$\forall B_i, B_j, \ \forall x, y \in B_i \oplus B_j, \quad (x-y) \notin S.$$

<u>Constructing a code</u>: We can remove half the elements from S by noting that if for some $x,y \in B_i \oplus B_j$, $(x-y) = -2^i$, then $(y-x) = 2^i$. So we need only check the sum sets $B_i \oplus B_j$ for pairs whose difference is $+2^i$.

As an example, we construct a code for a modulo 32 adder, A similar code exists for any modulus 2^n.

$$S = \{1, 2, 4, 8, 16, 32\}$$

$$B_0 = \{0,3,6,9, \ldots, 30\}$$
$$B_1 = \{1,4,7,10, \ldots, 31\}$$
$$B_2 = \{2,5,8,11, \ldots, 29\}$$

$$B_0 \oplus B_0 = \{0,3,6, \ldots, 60\}$$
$$B_0 \oplus B_1 = \{1,4,7, \ldots, 61\}$$
$$B_0 \oplus B_2 = \{2,5,8, \ldots, 59\}$$
$$B_1 \oplus B_1 = \{2,5,8, \ldots, 62\}$$
$$B_1 \oplus B_2 = \{3,6,9, \ldots, 60\}$$
$$B_2 \oplus B_2 = \{4,7,10, \ldots, 58\}$$

Now if the adder is given two inputs from B_0, the correct output is some multiple of 3 between 0 and 60 inclusive, Since the number 63 differs from any of these by a multiple of 3 (which cannot be a power of two), we could just as well allow the checker to pass as correct the sum 63. We can similarly extend the sets $B_0 \oplus B_2$, $B_1 \oplus B_2$, and $B_2 \oplus B_2$ so the checker operates as shown in the following table

Check symbol pair	Accept
H_0, H_0	$\{0,3,6, \ldots, 63\}$
H_0, H_1	$\{1,4,7, \ldots, 61\}$
H_0, H_2	$\{2,5,8, \ldots, 62\}$
H_1, H_1	$\{2,5,8, \ldots, 62\}$
H_1, H_2	$\{0,3,6, \ldots, 63\}$
H_2, H_2	$\{1,4,7, \ldots, 61\}$

It can be seen that this checker is a simple residue-checker, It corresponds to the widely known fact that any residue which is not a power of two can be used to check a modulo 2^n addition, provided that the carry out of the high digit is available to the checker.

REFERENCES

[1] Peterson, W.W., "On checking an adder," IBM Journal of Res. and Dev., vol. 2, no.2, April 1958.

[2] Langdon, F.F. and C.K. Tang, "Concurrent error detection for group look-ahead binary adders," IBM Journal of Res. and Dev., vol. 14, Sept. 1970.