

THE UNIVERSITY OF MICHIGAN
INDUSTRY PROGRAM OF THE COLLEGE OF ENGINEERING

ERROR CHECKING AND THE STRUCTURE OF BINARY ADDITION

By
Harvey Louis Garner

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in the
University of Michigan
1958

July, 1958

IP-305

Enagn
UMR
1565

Doctoral Committee:

Professor Alan B. Macnee, Co-Chairman
Professor Norman R. Scott, Co-Chairman
Professor Arthur W. Burks
Professor Harry H. Goode
Professor Gunnar Hok

ACKNOWLEDGMENT

The author wishes to express his gratitude to all the members of his doctoral committee. In particular, the author is indebted to the committee for the critical review afforded the first draft copy of this dissertation.

The author wishes to express his gratitude for the encouragement and aid given by his wife during the period in which this dissertation was being written and prepared for publication.

Last, an expression of thanks is due to the Industry Program of the College of Engineering for the excellent services rendered in the preparation and reproduction of this dissertation.

TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGMENTS.....	ii
LIST OF TABLES.....	v
LIST OF FIGURES.....	vi
NOMENCLATURE.....	vii
I. INTRODUCTION.....	1
II. GENERALIZED PARITY CHECKING.....	7
Introduction.....	7
Congruences.....	8
Generalization of Parity.....	9
Arithmetic Invariance of the Digital Parity Check..	12
Examples of Arithmetic Checking ($m=b=10$).....	18
Numerical Checking, Mod $m(m \neq b)$, of Arithmetic	
Operations.....	20
The Diminished Base Numerical Check.....	21
The Augmented Base Numerical Check.....	23
Examples of the Augmented Base Check.....	25
Conclusion.....	26
III. VECTOR SPACES AND BINARY ADDITION.....	27
Introduction.....	27
The Elementary Concepts of Vector Spaces.....	28
Vector Representation of Binary Numbers.....	34
Vector Length and Parity.....	37
The Vector Space of Binary Addition.....	39
The Carry Vector.....	42
Methods of Generation of the Carry Vector.....	46
Synthesis by Vector Space Operations.....	54
Numerical Relationships.....	57
Addition Algorithms.....	60
Constraints and Some Properties of the Addition	
Algorithms.....	62
Evaluation of the k_p and the V_p Logic.....	69
IV. THE ERROR STRUCTURE OF BINARY ADDITION AND THE	
EFFECTIVENESS OF THE PARITY CHECK.....	73
Introduction.....	73
The Error Structure of the Carry Logic of	
Standard Addition.....	75

TABLE OF CONTENTS (CONT'D)

	<u>Page</u>
The Nature of the Propagated Error.....	82
The Average Length of Carry Propagation.....	86
The Error Structure of the k, ρ and V Generating Logic for Standard Addition.....	89
The Error Structure of the rc Sum Logic.....	96
The Effectiveness of Parity Checking.....	98
 V. THE RESIDUE CODE.....	 105
An Extension of Numerical Checking.....	105
Residue Addition and Multiplication.....	110
Subtraction and the Representation of Negative Numbers.....	114
Conversion From a Residue Code to a Normal Number Representation.....	117
Greater Than or Less Than Relations for Residue Codes.....	120
Redundancy.....	125
Division.....	135
Conclusions.....	139
 VI. CONCLUSIONS AND RECOMMENDATIONS.....	 141
 APPENDIX.....	 148
Sign and Overflow Convention for One's and Two's Complement Arithmetic.....	148
 BIBLIOGRAPHY.....	 153

LIST OF TABLES

<u>Table</u>		<u>Page</u>
I	Relation Between a_j , b_j , and v_j , k_j	57
II	Frequency of Occurrence of Different Types of Carry Logic Malfunctions.....	79
III	Summary of Errors.....	99
IV	Error Type.....	100
V	Error Distribution (Normalized).....	100
VI	Redundancy of a Non-Relatively Primed Base Representation.....	106
VII	Natural Numbers and Corresponding Residue Numbers..	108
VIII	Number of States and Digits Associated with a Residue Representation.....	109
IX	Mod 2 and Mod 3 Sums and Products.....	111
X	Residue Code Efficiency.....	133





LIST OF FIGURES

<u>Number</u>		<u>Page</u>
1	Venn Diagram.....	32
2	Carry Vector.....	43
3	Logic for Independent, Sequential Generation of One Carry Element.....	49
4	Standard Carry Logic.....	53
5	Sum Dependent Carry Logic.....	56
6	Half Adder.....	70
7	Realization of the k_p Addition Algorithm.....	70
8	Carry Logic for the Standard Addition Algorithm....	71
9	Carry Logic for the v_p Addition Algorithm.....	71
10	Adder Logic.....	74
11	Probability that $c_1 = 1$	78
12	Carry Logic.....	80
13	Flow Diagram for the Average Length of Carry Propagation.....	88
14	Logic for the Determination of the Greater or Less Than Relationship.....	124
15	Relative Efficiency of the Residue Code.....	134

NOMENCLATURE

$[A]$	- the matrix A
$[A^{-1}]$	- the inverse matrix of A
$[A^t]$	- the transpose matrix of A
$D(x)$	- the dimension of x
d_{10}	- diode number one, open
d_{1S}	- diode number one, short
eq	- is equivalent to
$[G]$	- the metric matrix
g_{ij}	- an element of the metric matrix
k	- the result of $X \otimes Y$
k_j	- an element of k
$p(\alpha)$	- the probability of n one digits in a vector α
$p(\bar{r}_j)$	- the probability that r_j , an element of the vector is zero
$p\left(\prod_{r_j=1}^{c_{j+1}}\right)$	- the probability that c_{j+1} , an element of the carry vector, is unity when r_j is constrained to a value of unity
$p\left(\Delta \prod_{r_j=1}^{c_{j+1}}\right)$	- the probability that c_{j+1} , an element of the carry vector is changed due to a constraint $r_j=1$
$p(\Delta^* c_j)$	- the probability that the last element altered by a propagated error is c_j
$p_{r_1}(\bar{k}_1)$	- the probability that $k_1 = 0$ if $r_1 = 0$
r_j	- an element of ρ
s_j	- an element of σ

NOMENCLATURE (CONT'D)

u	- mean (statistical)
V	- the result of $X \vee Y$
v	- the connective for inclusive disjunction
v_j	- an element of V
var.	- variance (statistical)
\bar{x}	- diminished radix complement, also the bar is used to indicate logical negation
x	- radix complement or additive inverse
Z	- zero or null vector
δ_{ij}	- Kronecker delta
ϵ	- "an element of"
ρ	- the result of $X \bullet Y$
σ	- the sum vector
\equiv	- congruence relation $A \equiv a \pmod b$ reads A is congruent to a modulo b
\oplus	- modulo addition sign, also the symbol is used to indicate the exclusive disjunction which is addition modulo 2
\odot	- modulo multiplication or logic conjunction which is multiplication modulo 2
\leftarrow	- is replaced by
\leftrightarrow	- is in one to one correspondence with
\supset	- material implication connective; $A \supset B$ is read <u>if</u> A <u>then</u> B
	- and gate
	- or gate
	- exclusive or gate
	- negation element

CHAPTER I

INTRODUCTION

The modern electronic digital computing machine owes its origin to the concepts presented by Charles Babbage in 1835.⁽¹⁾ However, it was not until 1946 that technology reached a sufficiently advanced state to permit the development of a large scale digital electronic computer. The ENIAC,⁽²⁾ completed in 1946, consisted, in part, of 18,000 vacuum tubes. Successful computation was performed on the ENIAC during an era in which some difficulty was encountered in obtaining consistent operation of radar and communication circuitry consisting of two hundred or so vacuum tubes.

The state of the art in regard to computer reliability has made tremendous advances in recent years. The efforts to obtain more reliable computing systems have followed three general lines of attack. (1) The past decade has witnessed the introduction of new devices which offer increased component reliability. Furthermore, the operational components of a computer such as the flip-flops have been subjected to a thorough engineering study. The results of such detailed engineering studies have led to optimal design techniques in terms of known tolerance levels. (2) A second basic approach to the problem of increased reliability is through the use of redundant components. The work in this field ranges from consideration of increased reliability obtained by duplication up to n-tuplication considered by Von Neumann.⁽³⁾ The efforts in this area are divided roughly into two general classes, those which employ circuit redundancy with no error

detection capability and those which use redundant circuitry employing error detection schemes. The error detecting schemes frequently utilize a majority element which must function perfectly. The majority element determines the correct output on the basis of the majority output of the redundant circuits. It has been shown that the redundancy is most effective if applied at the lowest level. (3) The third approach to increased computer reliability is the utilization of redundant codes. The subject of redundant information coding has received considerable attention in recent years.⁽⁴⁻⁹⁾ However, most of the effort in this field has been directed toward error detecting and correcting schemes pertinent to communication systems. Error detection and correction for arithmetic operations in a digital computer require a code which is arithmetically invariant. This is not a characteristic of most of the error correcting and detecting codes developed for communication applications. One exception is the Hamming code.⁽⁵⁾ Error correction by means of the Hamming code for a general purpose computer of the Princeton class has been studied by Robertson.⁽¹⁰⁾ Another class of arithmetically invariant codes, the residue code, was used in the RAYDAC.* The checking procedure used in this machine has been described by Block,^(11,12) but effectiveness of the check is not considered in this paper. The material of this thesis provides further information on the utility of the residue code.

* (RAYDAC) Raytheon Digital Automatic Computer.

There have been many computers which have incorporated a simple parity checking system. Without exception the parity check has been used only to determine the validity of the computer storage mechanism and no attempts have been made to utilize the arithmetic invariant properties of the digital parity check. In fact, it appears that the true nature of the digital parity check has not been clearly understood. This is largely due to a cumbersome definition involving the oddness or evenness of the number of one digits of the representation.

The main problem attacked in this thesis is concerned with the effectiveness of parity checking procedures in the detection of errors in the arithmetic operations of a binary computer. In the course of the investigation certain topics pertinent to the process of binary arithmetic were discovered and this material is also included. Particular emphasis is given to an extension of the numerical parity check which leads to an arithmetic system without carry.

In Chapter II two types of parity checks are defined. The checks are the digital check which is a function of the digit symbols and the numerical check which may be a function of both the digit symbols and the positional weights. In this chapter congruence notation is introduced and the arithmetic invariance of the digital check is proved for the operation of addition. The results are then extended to the other arithmetic operations. The class of errors detectable by either type of check is discussed and examples are presented. Also in Chapter II congruences are used to generalize the concept of the digital parity check to include non-base-two number systems.

It is necessary to determine the error structure of the binary addition process before the effectiveness of the parity checking schemes can be evaluated. The binary addition process is complicated by the fact that the sum digits are not independent. The lack of independence is due to the propagated carry which is a feature of all weighted number systems. In Chapter III the structure of binary addition is investigated with particular emphasis given to the carry process. The mathematics of vector spaces was found to be particularly appropriate to the study of this problem. The general concept and properties of vector spaces are introduced and the mechanisms of binary addition are defined in terms of vector space concepts. Considerable use is made of matrix notation. The carry process is completely defined by a particular matrix. A study of the carry matrix is made and various circuits yielding different degrees of carry element independence are presented. Specific configurations are considered and it is shown that in general the price that must be paid in terms of time and equipment to obtain independence of the carry digits is too high. All of the different carry schemes which have been used in binary arithmetic units are derivable from the carry matrix presented in this chapter. The use of vector space concepts in connection with binary number representations sheds new light on certain problems. In particular, the conversion from one number code to another may be regarded simply as a change of basis. The length of a vector in an n dimensional space which corresponds to an n digit binary number is shown to be equivalent to the magnitude of the digital parity check.

The concepts of dimensionality of vector spaces are used extensively to define possible checking procedures and also to provide alternative definitions of the addition algorithm. In particular, it is shown that identical sums are obtained from: (1) the binary addition of two operands (2) the binary addition of the conjunction of the two operands and the inclusive disjunction of the two operands. (3) the binary addition of the exclusive disjunction of the two operands and two times the conjunction of the two operands (4) the binary subtraction of the exclusive disjunction of the two operands from two times the inclusive disjunction of the two operands. It is shown that the carry process associated with the last two definitions has different characteristics than the carry process associated with the standard addition algorithm, but performance-wise the new algorithms offer no advantages not associated with the standard algorithms.

In Chapter IV the results of Chapter III are applied to the detailed error structure of the binary addition process. The error structure is considered for the standard addition algorithm. The effects of logical malfunctions of the components which comprise the arithmetic circuitry are considered and classified. The probability of success of specific digital and numerical parity checks is determined in respect to the different classes of errors due to circuit malfunctions. It is found that the numerical check is 100% effective for single addition operations.

The concept of the numerical check is extended in Chapter V. If a sufficient number of numerical parity checks are employed, it is

possible to obtain a representation consisting of the numerical parity check digits which may be substituted for the original representation. In fact, the use of an excess number of parity check digits leads to an alternative representation containing redundancy. A number representation so derived is termed a residue code. The arithmetic properties and the utilization of redundant information are considered.

In Chapter VI conclusions and recommendations are presented. Appendix I is a description of a binary arithmetic sign and overflow convention pertinent to the material of Chapter IV.

CHAPTER II

GENERALIZED PARITY CHECKING

Introduction

The concept of the parity check⁽⁵⁾ is widely known and has frequently been used to detect and in some cases to correct errors in digital systems. The parity check is obtained by associating with each representation of information an auxiliary digit or group of digits known as the parity check digit(s). The proper choice of the relationship between the check digits and the information digits permits the establishment of mathematical operations, which may indicate erroneous digits or provide correction data. Parity checking is usually defined for the binary system in terms of the odd or evenness of the number of ONE digits in a specified block of binary digits. If the number of ONE digits is even the parity check digit is a zero. If the number of ONE digits is odd the parity check digit is a one. This definition is sufficient for most practical applications of the parity concept. However, the definition gives no insight into the fundamental principles of parity. A restatement of parity in terms of linear congruences provides a mathematical basis for the parity concept and permits a generalization which includes number systems of radix $r \neq 2$. Congruence notation also provides the mathematical tool required to prove the parity check invariant to addition for any consistently weighted number system if the carries are processed correctly.

Congruences

The remainder of this paper will make considerable use of congruence notation. A short presentation of pertinent concepts and relationships is included at this point. Additional material on the subject may be found in any standard text on Number Theory.*

The congruence

$$A \equiv \alpha \text{ Mod } b$$

is read "A is congruent to α Modulo b." The congruence relationship states that the equation

$$A = \alpha + bt$$

is valid for some value of t, where A, α , b and t are integers. α is called the residue and b the base or modulus of the number A.

As examples of congruences consider:

$$10 \equiv 7 \text{ Mod } 3$$

$$10 \equiv 4 \text{ Mod } 3$$

$$10 \equiv 1 \text{ Mod } 3$$

In these examples the integers 7, 4 and 1 form a residue class of 10 Mod 3. Of particular importance is the least positive residue of the class which in this example is one. The least positive residue is that residue for which $0 \leq \alpha \leq b$.**

* Hardy and Wright⁽¹³⁾ gives an excellent presentation on congruences.

** The equality sign may exist on only one side of the inequality.

Unless otherwise specified the term residue as used in the remainder of this paper will mean the least positive residue.

Congruences for the most part may be treated in the same manner as the equality sign in ordinary arithmetic. The main exception pertains to division. The following properties of congruences will be used in this paper and are presented below without proof.

1. Congruences to the same modulus may be added and the result is a valid congruence.

$$\sum_{i=1}^n A_i \equiv \left(\sum_{i=1}^n \alpha_i \right) \text{ Mod } b$$

2. Congruences to the same modulus may be multiplied and the result is a valid congruence.

$$\prod_{i=1}^n A_i \equiv \left(\prod_{i=1}^n \alpha_i \right) \text{ Mod } b$$

3. Congruences are transitive. If $A \equiv B$ and $B \equiv C$ then $A \equiv C$.

4. The following congruence relationships provide a basis for numerical checking procedures

$$(a) \sigma b^n \equiv \sigma \text{ Mod } (b-1)$$

$$(b) \sigma b^n \equiv +\sigma \text{ Mod } (b+1) \quad n \text{ even}$$

$$\equiv -\sigma \text{ Mod } (b+1) \quad n \text{ odd}$$

Generalization of Parity

A parity check need not necessarily include every digit of the number. In the general case multiple parity checks may be employed,

each check concerning only specific digits. However, for purposes of simplification, only the case of a single parity check over all digits will be considered here. The parity of a number from a consistently weighted number system is now defined. Consider the number

$$\sigma_n \sigma_{n-1} \dots \sigma_2 \sigma_1$$

This is the usual shorthand notation for the polynomial representation of a number N in a consistently weighted number system, base b:

$$N = \sigma_n b^{n-1} + \sigma_{n-1} b^{n-2} + \dots + \sigma_2 b^1 + \sigma_1 b^0$$

The parity check digit p associated with number N is defined by:

$$F(N) \equiv p \text{ Mod } m$$

where p is the least positive residue.

In this paper two different functions of N will be considered.

The first function is

$$F(N) = N$$

and the parity check digit p is given by

$$N \equiv p \text{ Mod } m \quad m \neq b$$

This type of check is based directly on the numerical properties of linear congruences and the check digit is a function of the magnitude of the number N. This type of check is called a numerical parity check.

If the check base m is identical to the number base b then

$$N \equiv \sigma_1 \text{ Mod } b$$

The check is unsatisfactory because it is a function of only the low order digit of the number representation. This is due to the fact that

$$\sigma_i b^n \equiv 0 \text{ Mod } b \quad n > 0$$

A check for the case $m = b$ is obtained if

$$F(N) = \sigma_n + \sigma_{n-1} + \dots + \sigma_2 + \sigma_1$$

The parity check digit, p , for this type of check which is termed digital checking is defined by the relation

$$\sum_{i=1}^n \sigma_i \equiv p \text{ Mod } b$$

An equivalent definition may be given in terms of ring addition. Ring addition modulo b is specified by the symbol \oplus . Ring addition is simply normal addition without carry, for example

$$(b-1) \oplus 1 = 0$$

In terms of ring addition modulo b the parity digit p is given as

$$p = \sigma_n \oplus \sigma_{n-1} \oplus \dots \oplus \sigma_2 \oplus \sigma_1$$

If the number system is binary, and the check base is two then the previous definition for the digital check provides a parity digit which makes the total number of one digits of the number plus check representation even in correspondence with the usual parity procedures. The definition is easily extended to an odd parity check. For this case the parity check digit is given by \bar{p} , the diminished radix complement of p defined as

$$\bar{p} \oplus p = b - 1$$

As an example of the parity definition for $m = b$, consider the following:

Given 12894, a number in base 10

then $(1 + 2 + 8 + 9 + 4) \equiv p \text{ Mod } 10$

or $p = 1 \oplus 2 \oplus 8 \oplus 9 \oplus 4 = 4$

Both types of parity checks are restricted in application to the detection of certain classes of errors in the number representation. Error correction is not possible unless multiple parity checks are employed. The digital parity check, $m = b$, detects all alterations except the alterations in which the sum of the digit perturbations is congruent to 0 modulo b . Let the decimal number of the previous example be corrupted by 5 in the first column and by -5 in the third column. The result is an erroneous representation but the error is obviously undetectable.

$$1 \oplus 2 \oplus 3 \oplus 9 \oplus 9 \oplus = p = 4$$

A numerical parity check Mod m , $m \neq b$, is insensitive to an error if the magnitude of the error is congruent to zero Mod m . Thus the error is detectable only if the check base m is not a divisor of the magnitude of the error. The magnitude of the error in the previous example is 495. The error is not detected by Mod 3, 5, 9 or 11 checks.

Parity checks are particularly effective in detecting errors for which only one digit has been modified. For this class of errors a parity check $m = b$ is always effective. If m is less than b then perturbations congruent to zero modulo m are undetectable.

Arithmetic Invariance of the Digital Parity Check

In the arithmetic system employed in most digital machines a one-to-one correspondence is established between the machine digit representation and a set of positive and negative fractions. The rules of machine arithmetic operations are set up in such a way as to preserve

this one-to one correspondence under the operation of addition. Addition is the fundamental arithmetic operation of the machine and the other arithmetic operations, subtraction, multiplication and division are defined in terms of addition and the auxilliary operations, shift and complement.

It is obvious that the digital parity check is invariant to the shift operation if no digits are deleted or added to the representation. This is true because the digital parity check is a check function of the modulo sum of the individual digit rather than a function of magnitude. Also, the parity check is independent of the position of the radix point. For this reason the remaining discussion will not explicitly consider whether the representations are in correspondence with fractions or integers.

Consider now the addition of two consistently weighted numbers in base b representation. The check base is, of course, also equal to b .

$$A + D = S$$

or

$$\begin{array}{r} a_n \dots a_1 \\ + d_n \dots d_1 \\ \hline c_{n+1} s_n \dots s_1 \end{array}$$

now

$$s_i = a_i \oplus d_i \oplus c_i$$

and

$$(a_i + d_i + c_i) = s_i + c_{i+1}b$$

$$c_1 = 0$$

$$c_i = 0 \text{ or } c_i = 1 \text{ if } i \neq 1$$

The parity check digit p for the sum is given as

$$c_{n+1} \oplus s_n \oplus s_{n-1} \oplus \dots \oplus s_1 = p_S$$

or

$$c_{n+1} \oplus a_n \oplus d_n \oplus c_n \oplus \dots \oplus a_1 \oplus d_1 = p_S$$

but

$$a_n \oplus \dots \oplus a_1 = p_A$$

and

$$d_n \oplus \dots \oplus d_1 = p_D$$

therefore

$$c_{n+1} \oplus \dots \oplus c_2 \oplus p_A \oplus p_D = p_S$$

The last expression provides a means of checking the addition operation in terms of the parity check digits of the sum, the addend, ~~and the augend and the generated carries.~~ Alternatively the parity check may be in terms p', the radix complement or additive inverse of p defined as

$$p' \oplus p = 0$$

If p' is used as the check digit rather than p the check procedure for addition is given as

$$c_n \oplus \dots \oplus c_1 \oplus p'_S = p'_A \oplus p'_D$$

The class of addition errors detected by the digital parity check is the same as the class of representation errors defined in the previous section. However, it must be remembered that the logic of the usual addition circuits is such that the various carry digits are not independent. The parity checking procedure is valid only if the carry digits are correct.

Consider the decimal addition of the following numbers:

$$\begin{array}{r} 66941 \overset{p}{|} 6 \\ 32451 \overset{p}{|} 5 \\ \hline 99392 \overset{p}{|} 2 \end{array}$$

Due to the occurrence of one carry, the check of addition is

$$\begin{aligned} 6 \oplus 5 \oplus 1 &= 2 \\ 2 &= 2 \end{aligned}$$

The check of the complement operation is straightforward.

The diminished radix complement of a representation "s" is obtained by substituting for each digit σ_i the diminished radix complement, $\bar{\sigma}_i$.

$$\sigma_i \oplus \bar{\sigma}_i = b-1$$

given:
$$\sigma_n \oplus \dots \oplus \sigma_1 = p_s$$

then
$$\bar{\sigma}_n \oplus \bar{\sigma}_n \oplus \dots \oplus \bar{\sigma}_1 \oplus \bar{\sigma}_1 = p_s \oplus \bar{\sigma}_n \oplus \dots \oplus \bar{\sigma}_1$$

let
$$\bar{\sigma}_n \oplus \dots \oplus \bar{\sigma}_1 \oplus p_s = p_{\bar{s}} \oplus p_s = f$$

then
$$f \equiv n(b-1) \pmod{b}$$

$$(f+n) \equiv nb \pmod{b}$$

$$(f+n) \equiv 0 \pmod{b}$$

then
$$p_s + p_{\bar{s}} + n \equiv 0 \pmod{b}$$

or
$$p_s \oplus p_{\bar{s}} \oplus n \pmod{b} = 0$$

A check of the diminished radix complement consists of the modulo b sum of the parity check of the normal representation, the parity check of the complemented representation and the number of digits comprising the representation. In the special case where $b = 2$, the

check of the complemented representation is equal to the check of the normal representation if n is even and opposite if n is odd.

A check of the radix complement operation is somewhat more complex than that required for the diminished radix complement. The complications are due to the definition of the radix complement rather than to the nature of the parity check procedure. Two checking procedures are possible. The first is based on the fact that the radix complement s' is obtained from the diminished radix complement \bar{s} by the addition of one to the lowest order digit. This method requires cognizance of the carries produced when the one is added to the lowest order digit. The check is given by the relationship

$$p_{s'} = 1 \oplus p_{\bar{s}} \oplus \sum_{i=2}^{n+1} c_i \text{ Mod } b$$

The second procedure for obtaining the radix complement check is based upon the well-known rule for obtaining the digitwise radix complement: "Starting with the low order digit, radix complement the first non-zero digit; the remaining higher order digits are changed by diminished radix complementation." This method requires a knowledge of the number of zeros preceding the first non-zero digit. Let this number be q .

Given $\sigma_n \oplus \dots \oplus \sigma_1 = p_s$

then $\sigma_n \oplus \bar{\sigma}_n \oplus \dots \oplus \sigma_{q+1} \oplus \sigma'_{q+1} \oplus \sigma_q \oplus \dots =$
 $p_s \oplus \sigma_n \oplus \dots \oplus \sigma'_{q+1}$

$c_i = 0$ if $i = q$

$$p_{S'} = \bar{\sigma}_n \otimes \dots \otimes \bar{\sigma}_{q+2} \otimes \sigma'_{q+1}$$

$$p_S + p_{S'} \equiv [(n-q-1)(b-1) + b] \text{ Mod } b$$

$$p_S + p_{S'} + (n - q) \equiv 1 \text{ Mod } b$$

or

$$p_S \otimes p_{S'} \otimes (n - q) \text{ Mod } b = 1$$

Multiplication consists of repetitive addition with appropriate shift operations. The result is a double-length product. The digital parity of the double-length product is found from consideration of the addition operations since parity is not affected by the shift operation. The linear congruence is multiplicative. The check procedure for the product $P = A \times D$ is

$$p_P = (p_A \otimes p_D) \otimes \sum c_i \text{ Mod } b$$

or

$$(p_A \times p_D) + \sum c_i \equiv p_P \text{ Mod } b$$

The symbol \otimes indicates multiplication, modulo b . It is observed that the digital parity check provides no protection against faulty shift operations. Parity protection against this type of error is obtained if multiple checks are employed.

The division algorithm commonly employed is easily explained in terms of the remainder theorem.

Consider
$$\frac{X}{Y} = Q + \frac{R}{Y}$$

or

$$X = QY + R$$

$$R = X - QY$$

where

$$Q = q_n b^{n-1} + \dots + q_j b^{j-1}$$

then

$$R = X - (q_n b^{n-1} + \dots + q_j b^{j-1}) Y$$

The subtraction of Y from X, q_i times is accomplished by the addition of \bar{Y} , q_i times. Carries which are generated must be considered. The parity check expression is given as

$$p_R = p_X \oplus (p_Q \oplus p_{\bar{Y}}) \oplus \sum c_i \text{ Mod } b$$

This check may be employed for each division operation or only as a check of the overall division process. In any case the check is insensitive to shift errors.

Examples of Arithmetic Checking (m = b = 10)

Addition

$$\begin{array}{r|l} 3 & 4 & 6 & | & \textcircled{3} \\ + & 4 & 5 & 8 & | & \textcircled{7} \\ \hline & 8 & 0 & 4 & | & \textcircled{2} \end{array}$$

○ the circled digits are the check digits

Check

$$p_A \oplus p_D \oplus \sum c_i \text{ Mod } b = p_S$$

$$3 \oplus 7 \oplus 2 = 2$$

Complement

nines complement

$$A = 0345 \textcircled{2}$$

$$\bar{A} = 9654 \textcircled{4}$$

Check

$$p_{\bar{A}} \oplus p_A \oplus n \text{ Mod } b = 0$$

$$4 \oplus 2 \oplus 4 = 0$$

ten's complement

$$A = 034500 \text{ (2)}$$

$$A' = 965500 \text{ (5)}$$

Check

$$p_{A'} \oplus p_A \oplus (n-q) \text{ Mod } b = 1$$

$$5 \oplus 2 \oplus 4 = 1$$

Product

$$\begin{array}{r} 346 \text{ (3)} \\ \underline{213 \text{ (6)}} \end{array}$$

$$\begin{array}{r} 346 \\ \underline{346} \end{array}$$

$$\begin{array}{r} 692 \\ \underline{346} \end{array}$$

$$\begin{array}{r} 1038 \\ \underline{346} \end{array}$$

5 carries generated

$$\begin{array}{r} 4498 \\ \underline{346} \end{array}$$

$$\begin{array}{r} 39098 \\ \underline{346} \\ \hline 73698 \text{ (3)} \end{array}$$

Check

$$p_p = (p_A \oplus p_D) \oplus \sum c_i \text{ Mod } b$$

$$3 = (3 \oplus 6) \oplus 5$$

$$3 = 8 \oplus 5 = 3$$

Division

$$Q = \frac{073699 \text{ (4)}}{0346 \text{ (3)}} = \frac{X}{Y}$$

the ten's complement of 0346 (3) is 9645 (4)

	Carries	
$\begin{array}{r} 073699 \\ \underline{9654} \\ 039099 \\ \underline{9654} \\ 004499 \\ \underline{9654} \\ 969899 \\ \underline{04499} \\ 9654 \\ \underline{01039} \\ 9654 \\ \underline{97579} \\ 1039 \\ \underline{9654} \\ 0693 \\ \underline{9654} \end{array}$	2 2 3 1 2	<div style="text-align: right; margin-right: 20px;"> \rightarrow </div> $\begin{array}{r} 0347 \\ \underline{9654} \\ 0001 \\ \underline{9654} \\ 9655 \\ \underline{001} \end{array}$ <p style="margin-left: 40px;"> $Q = 213 \text{ (6)}$ $R = 1$ $\sum c_i = 13$ </p>

Check $P_R = P_X \oplus (P_Q \oplus P_{\bar{Y}}) \oplus \sum c_i \text{ Mod } b$

$1 = 4 \oplus (6 \oplus 4) \oplus 3$

Numerical Checking, Mod m (m ≠ b), of Arithmetic Operations

The modulus of the numerical parity check is not the same magnitude as the radix of the number representation. Congruence relationships are valid for any modulus. However, only certain moduli yield procedures which permit straightforward computation of the parity digits. There exist in particular two interesting and useful checks based on procedures congruent modulo (b-1) and congruent modulo (b+1). These checks will be called the diminished base numerical check and the augmented base numerical check respectively. The simplest checks of the numerical class depend directly on certain congruence relationships. Numerical checking has one very desirable feature not associated with digital checking procedures. Numerical checking procedures do not require a tally or cognizance of the carries generated by the addition process.

The Diminished Base Numerical Check

The diminished base numerical check is dependent on the following properties of linear congruences:

$$\sigma_i b^n \equiv \sigma_i \text{ Mod } (b-1)$$

This relationship permits parity calculation independent of the digit position or weight. Consider the sum

$$X + Y = S$$

$$X = x_n b^{n-1} + \dots + x_1 b^0$$

$$Y = y_n b^{n-1} + \dots + y_1 b^0$$

$$S = c_{n+1} b^n + s_n b^{n-1} + \dots + s_1 b^0$$

Since congruences are additive we have

$$X = \sum_{i=1}^n x_i b^{i-1} \equiv \sum_{i=1}^n x_i \text{ Mod } (b-1)$$

The parity check digit p_X associated with the representation X is defined by the term $\sum_{i=1}^n x_i$ reduced to the least positive residue of the class. The parity check digit p_X may also be defined by

$$p_X = x_n \oplus \dots \oplus x_1$$

for either definition

$$X = \sum_{i=1}^n x_i b^{i-1} \equiv p_X \text{ Mod } (b-1)$$

The check digit p_Y is given as

$$Y = \sum_{i=1}^n y_i b^{i-1} \equiv p_Y \text{ Mod } (b-1)$$

Linear congruences are additive, therefore:

$$X + Y = \sum_{i=1}^n (x_i + y_i) b^{i-1} \equiv (p_X + p_Y) \text{ Mod } (b-1)$$

and

$$p_S = p_X \oplus p_Y$$

The modulo $(b-1)$ check is numerical and hence does not require cognizance of the generated carries. The check is, nevertheless, sensitive to errors in carry generation and propagation. The check detects all digit alterations for which the modulo $(b-1)$ sum of the changes does not equal zero. This is equivalent to the statement that an alteration is detectable only if $b-1$ is not a factor of the magnitude of the arithmetic error. A single carry failure causes an alteration of minus one and is therefore detectable. However, the error is not correctable on the basis of the parity information.

A check of the complement operation is obtained from the basic definition of the complemented representation. The definitions which follow are for representations of $n + m + 1$ digits. The representation has n digits to the left of the radix point, m digits to the right of the radix point and one sign digit. The diminished radix complement of X is \bar{X} where

$$X + \bar{X} = b^{n+1} - b^{-m}$$

The radix complement X' is given as

$$X + X' = b^{n+1}$$

The usual congruence relationships are defined for integer values. It is, therefore, convenient to consider the radix point on the extreme right of the representation. The check is obtained by observing that

$$b^m(b^{n+1} - b^{-m}) \equiv 0 \text{ Mod } (b-1)$$

and

$$b^m b^{n+1} \equiv 1 \text{ Mod } (b-1)$$

Then the check for the diminished radix complement is

$$p_X \oplus p_{\bar{X}} = 0$$

and the check for the radix complement is

$$P_X \oplus P_{X'} = 1$$

Congruences are multiplicative. Therefore, a check of the product

$$P = R \times S$$

is given simply as

$$P_P = P_R \otimes P_S$$

with no cognizance of carries required.

A check involving the division of congruences is not practical since the correct division procedure is dependent upon whether the divisor and (b-1) have any common factors. However, this is of little consequence since machine division is usually obtained by a process consisting of repetitive add and shift operations.

$$\frac{X}{Y} = Q + \frac{R}{Y}$$

$$X = QY + R$$

The check of the division process is independent of carries and must be true at every step in the division process. The check is

$$P_X = (P_Q \otimes P_Y) \oplus P_R$$

The Augmented Base Numerical Check

The diminished base numerical check avoids the need for carry cognizance and for that reason offers a simpler check procedure than is obtainable by means of digital checking. However, for the binary number system the diminished base check is meaningless. One solution is the

augmented base check. This system is also carry independent and parity check digits are relatively easy to obtain. The augmented base check is dependent on the following property of congruences.

$$\begin{aligned}\sigma_i b^j &\equiv +\sigma_i \text{ Mod } (b+1) \text{ if } j \text{ is even} \\ &\equiv -\sigma_i \text{ Mod } (b+1) \text{ if } j \text{ is odd}\end{aligned}$$

The check digit for representation A is given as

$$(\dots + a_5 - a_4 + a_3 - a_2 + a_1) \equiv p \text{ Mod } (b+1)$$

where $A = \dots + a_5 b^4 + a_4 b^3 + a_3 b^2 + a_2 b^1 + a_1 b^0$

The checking procedures employed for the arithmetic processes are identical to the procedures for the diminished radix except that ring addition \oplus and ring multiplication \odot are defined modulo $(b+1)$.

In the binary case the number base is two and the check base is three. The parity of a representation A is given as

$$(\dots + a_5 + 2a_4 + a_3 + 2a_2 + a_1) \equiv p \text{ Mod } 3$$

or $\dots \oplus a_5 \oplus 2a_4 \oplus a_3 \oplus 2a_2 \oplus a_1 = p$

Direct subtraction is avoided by using the radix complement of one with respect to the base three.

Note that for the binary case the augmented base check is identical to the diminished base check, base four.

Examples of the Augmented Base Check

Let $b = 2$ then check is modulo 3

Addition $X + Y = S$

0 0 0 1 1 1 0 1	⓪	Check	$P_X \oplus P_Y = P_S$
0 1 0 1 0 1 0 1	⓪		
0 1 1 1 0 0 1 0	⓪		

Complement

$A = 00011101$ ⓪

Diminished Radix Complement

$\bar{A} = 11100010$	⓪	Check	$P_A \oplus P_{\bar{A}} = 0$

Radix Complement

$A' = 11100011$	⓪	Check	$P_A \oplus P_{A'} = 1$

Multiplication $R \times S = P$

1011	⓪	Check	$P_R \oplus P_S = P_P$
x 1001	⓪		
1011			
101100			
110011	⓪	$10 \oplus 00 = 00$	

Division $\frac{X}{Y} = Q + \frac{R}{Y}$

0 110011	Check	$P_X = (P_Q \oplus P_Y) \oplus P_R$
1 0101		
0 000111		
		$= 10 \oplus 10 = 00$

The above is a check of the first step in the division process.

Conclusion

The congruence notation used in this paper provides a fundamental basis for the concept of both the digital and the numerical type of parity check. Congruence notation also provides the mathematical tools needed to consider the arithmetic properties of the parity checks and the extension of the parity check concept to non-binary systems.

In the final analysis the effectiveness of a parity check is dependent upon the error structure of the carry process. It is necessary to consider in detail the mechanism of binary addition before the effectiveness of the parity check may be evaluated. The process of binary addition is investigated in the next chapter and the effectiveness of the parity check is discussed in Chapter IV.

CHAPTER III

VECTOR SPACES AND BINARY ADDITION

Introduction

The concepts, representations and operations associated with vector spaces are particularly appropriate to the study of the process of binary addition. Vectors or n -tuples correspond to binary numbers. The vector addition of two vectors corresponds to the logical addition of two binary numbers. The length of a vector is shown to correspond to the digital definition ($m=b$) of parity. A study of the dimensionality properties of vectors facilitates the establishment of various numerical relationships pertinent to checking procedures. These and other relationships are discussed in the material which follows.

The application of the vector space concepts and the use of matrix notation represent a basic and unified approach to the subject of binary addition. The vector space approach to binary addition shows clearly that the basic problem of binary addition from the viewpoint of both error structure and speed of addition is the generation of the carry. The different logical configurations employed to generate the carry may be obtained from the matrix representation of the carry process. This particular approach also facilitates the study of the differences in the expected performance of the various known methods of carry generation.

The representation of binary numbers as vectors is not a new concept. Extensive use of vector space concepts has been made by

Muller⁽²²⁾ in the problem of logical design and coding. Reed⁽⁸⁾ has used vector space concepts to discuss a particular code. More recently Campeau⁽²¹⁾ has made extensive use of Boolean matrices as a logical design tool. The earliest application of matrices to the problem of synthesis of different logic configurations is due to Lunts⁽¹⁹⁾. The study of binary addition by means of vector spaces presented in this chapter is new as are the applications of the concepts of dimensionality and length.

In the next section the basic properties of vector spaces are considered. The application of these concepts to the problem of binary addition is considered in the remaining sections.

The Elementary Concepts of Vector Spaces

Ordered n-tuples of scalar functions, a_i of a field F , are termed vectors of n dimensions if the operations of scalar multiplication and vector addition are closed. A set of n -tuples in which vector addition and scalar multiplication are closed is called a vector space. The dimension of the space is equal to the number of elements of an n -tuple.

Given n -tuples α and β and a scalar k

$$\alpha = (a_1, a_2, a_3 \dots a_n)$$

$$\beta = (b_1, b_2, b_3 \dots b_n)$$

$$k \in F$$

Scalar multiplication is defined in terms of the multiplication operation, \odot , of the field.

$$k \odot \alpha = (k \odot a_1, k \odot a_2, k \odot a_3, \dots k \odot a_n)$$

In order to secure compactness of notation, multiplication appropriate to the field under consideration will be assumed and the notation for scalar multiplication abbreviated to the form

$$k\alpha = (ka_1, ka_2, ka_3, \dots ka_n)$$

Vector addition is defined in terms of the rules of addition of the field elements (scalars).

$$\alpha \oplus \beta = (a_1 \oplus b_1, a_2 \oplus b_2, \dots a_n \oplus b_n)$$

Algebraically the vectors and the scalars of a vector field are associative, commutative, and distributive with respect to the operations of vector addition and scalar multiplication. This follows from the definition of the operations and the fact that the field has the associative, commutative and distributive properties.

A vector of an n dimensional space is usually specified as a function of the basis of the vector space. The basis may be any set of n linearly independent vectors. A set of n vectors $\alpha_1, \dots, \alpha_n$, is linearly dependent if there exists in F, n scalars $k_1 \dots k_n$, not all zero, such that:

$$k_1\alpha_1 \oplus k_2\alpha_2 \oplus \dots \oplus k_n\alpha_n = 0$$

or

$$k_1a_{11} \oplus k_2a_{21} \oplus \dots \oplus k_na_{n1} = 0$$

$$k_1a_{12} \oplus k_2a_{22} \oplus \dots \oplus k_na_{n2} = 0$$

$$k_1a_{13} \oplus k_2a_{23} \oplus \dots \oplus k_na_{n3} = 0$$

$$\begin{array}{cccccc}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{array}$$

$$k_1a_{1n} \oplus k_2a_{2n} \oplus \dots \oplus k_na_{nn} = 0$$

Note, in particular, that a suitable basis requires only linearly independent base vectors. The base vectors need not be orthogonal.

If the inner product is defined, then the conditions for linear dependence may be written in matrix notation as

$$[A] k =, Z \quad Z \text{ denotes the null vector}$$

The convention used in this paper for matrices is as follows: An $m \times n$ matrix A is bracketed, $[A]$. A row or column matrix or vector is not bracketed. This convention eliminates needless brackets and leads to more readable equations. The expression $[A] k$ always represents a matrix product and should not be taken as a scalar product. In general vectors are represented by letters from the Greek alphabet. The inner product of two vectors, α and β , with respect to a basis γ is defined as

$$= (a_1\gamma_1 \oplus \dots \oplus a_n\gamma_n) (b_1\gamma_1 \oplus \dots \oplus b_n\gamma_n) =$$

$$\sum_{i,j=1}^n a_i b_j (\gamma_i, \gamma_j) = \sum_{i,j=1}^n a_i b_j g_{ij}$$

The result is a scalar function. If the basis is orthonormal then

$$g_{ij} = \delta_{ij}$$

and

$$= a_1 b_1 \oplus \dots \oplus a_n b_n$$

The inner product is associative, commutative and also distributive with respect to addition.

The inner product of a vector α with itself in vector space spanned by an orthonormal basis defines the square of the vector length.

The inner product gives the sum of each orthogonal component squared.

$$\alpha\alpha = a_1a_1 \oplus a_2a_2 \oplus \dots \oplus a_na_n$$

The basis may be changed by a not necessarily orthogonal linear transformation.

$$\alpha_1 = [A]\alpha$$

then

$$\alpha = [A^{-1}]\alpha_1$$

$$\alpha^t\alpha = \alpha_1^t [A^{-1}]^t [A^{-1}] \alpha_1$$

let

$$G = [A^{-1}]^t [A^{-1}]$$

The square of the vector length defined by the inner product remains invariant if

$$\alpha\alpha = \sum_{1,j=1} a_1a_j g_{1j} \quad \text{where } g_{1j} \in [G]$$

The length as such may not be obtainable since this requires the inclusion of the square root in the field.

The inner product may be used to determine the orthogonality of a pair of vectors. A vector pair is orthogonal if and only if the inner product vanishes. This follows from the definition of the inner product given as

$$\alpha\beta = |\alpha| \cdot |\beta| \cos \theta$$

A theorem concerning the dimensionality of vector spaces which will be used later in this paper is now considered. The dimension of a vector space or subspace is equal the minimum number of base vectors required to span the space or subspace.

Theorem: If S and T are any two subspaces of a vector space Q , then the subspace dimensions satisfy

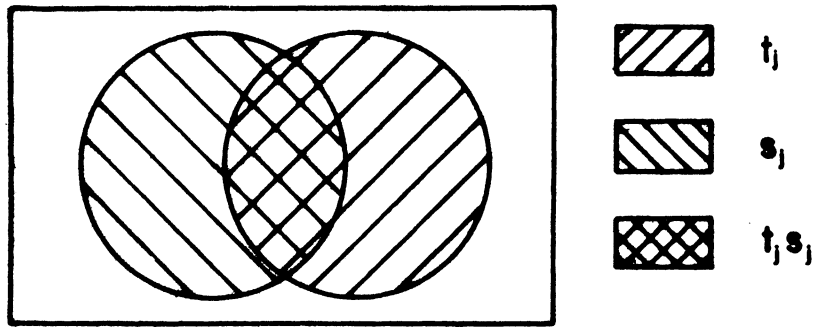
$$D(S) + D(T) = D(S \cap T) + D(S \cup T)$$


Figure 1. Venn Diagram.

This theorem is readily verified. Consider the Venn Diagram shown in Figure 1. The objects of class s_j are contained by the left-hand circle, the objects of class t_j by the right-hand circle. The objects common to both classes are contained by the conjunction of s_j and t_j designated as $s_j \cap t_j$. The objects of either class s_j or t_j , written $s_j \cup t_j$, and called the inclusive disjunction may be calculated by summing the number of objects in class s_j and the number in class t_j and subtracting the number objects contained by the conjunction. The subtraction of the conjunction is required since it has been summed twice. The result is

$$N(s_j \cup t_j) = N(s_j) + N(t_j) - N(s_j \cap t_j)$$

A formula of the above form could be written for every base vector under consideration. A given base vector either spans or does not span a

particular vector subspace. Therefore, $N(x)$ may have only two values, zero and one. If the resulting set of equations are summed the result would be

$$\sum_{j=1}^m N(s_j \vee t_j) = \sum_{j=1}^m N(s_j) + \sum_{j=1}^m N(t_j) - \sum_{j=1}^m N(s_j \cap t_j)$$

The dimension of a vector subspace is defined as

$$D(x) = \sum_{j=1}^m N(x_j)$$

so

$$D(S \vee T) = D(S) + D(T) - D(S \cap T)$$

A single vector is a vector subspace. Therefore, the fundamental dimensionality relationship, which involves two vector subspaces, must be valid for vector pairs. The dimension of a single vector, with elements from the binary field is equal to the number of elements having the value one. The dimension concept can be extended to include the vector addition connective

$$D(S \oplus T) = D(S \vee T) - D(S \cap T)$$

A simple and rapid proof of the above relationship is obtained from consideration of the definition of the connectives or the Venn diagram and by virtue of the fact that the dimension relationships must be valid for any set of bases including a single base. Two additional dimensional relationships are obtained by linear combination of the previous dimension formula.

$$D(S) + D(T) = D(S \oplus T) + 2D(S \cap T) = 2D(S \vee T) - D(S \cap T)$$

The preceding material has presented informally the concepts of the vector spaces necessary for an understanding of the remainder of

this paper. For a formal presentation of this material the reader is referred to any standard text on the subject of vector spaces.*

Vector Representation of Binary Numbers

The binary number system as employed in the digital computer is closed. The digit symbols are the elements of the modulo two field. Vector addition and scalar multiplication are obviously closed. Therefore, the closed set of binary numbers may be represented by a set of vectors in a vector space of finite dimensions.

The basis of the vector space may be any set of linearly independent vectors. The basis employed in the usual binary number representation is orthogonal. The base vectors are the set of n-tuples or binary numbers representing the powers of two from zero to m-1, where m is the dimension of the vector space. The number of binary digits is also equal to m. Let the base vectors be designated μ_i . Then the familiar polynomial representation of a binary number A

$$A = a_n 2^{n-1} + a_{n-1} 2^{n-2} + \dots + a_2 2^1 + a_1 2^0$$

is in one to one correspondence to the vector

$$= a_n \mu_n \oplus a_{n-1} \mu_{n-1} \oplus \dots \oplus a_2 \mu_2 \oplus a_1 \mu_1$$

* See for example Birkhoff and MacLane. (14)

where

$$\begin{aligned} \mu_1 &= (0\ 0\ 0\ \dots\ 0\ 0\ 0\ 0\ 1) \\ \mu_2 &= (0\ 0\ 0\ \dots\ 0\ 0\ 0\ 1\ 0) \\ \mu_3 &= (0\ 0\ 0\ \dots\ 0\ 0\ 1\ 0\ 0) \\ &\vdots \\ \mu_{n-1} &= (0\ 1\ 0\ \dots\ 0\ 0\ 0\ 0\ 0) \\ \mu_n &= (1\ 0\ 0\ \dots\ 0\ 0\ 0\ 0\ 0) \end{aligned}$$

In both expressions the a_i coefficients are elements of the modulo 2 field.

The cyclically permuted binary code is now considered as an illustration of a non-orthogonal basis.

$$C = [T] \alpha$$

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix}$$

Each column in the T matrix is one of the base vectors. The vector representation of the cyclically permuted number is

$$C = a_n \delta_n \oplus a_{n-1} \delta_{n-1} \oplus \dots \oplus a_2 \delta_2 \oplus a_1 \delta_1$$

where

$$\begin{aligned} \delta_1 &= (0\ 0\ 0\ 0 \dots 0\ 0\ 0\ 1) \\ \delta_2 &= (0\ 0\ 0\ 0 \dots 0\ 0\ 1\ 1) \\ \delta_3 &= (0\ 0\ 0\ 0 \dots 0\ 1\ 1\ 0) \\ &\vdots \\ \delta_{n-2} &= (0\ 0\ 1\ 1 \dots 0\ 0\ 0\ 0) \\ \delta_{n-1} &= (0\ 1\ 1\ 0 \dots 0\ 0\ 0\ 0) \\ \delta_n &= (1\ 1\ 0\ 0 \dots 0\ 0\ 0\ 0) \end{aligned}$$

The conversion from a cyclically permuted code to a binary code is defined by

$$\gamma = [T^{-1}] c$$

where

$$[T^{-1}] = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & \dots & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & \dots & 1 & 1 & 1 \\ \vdots & & & & & & & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix}$$

Thus the binary number A is also in one to one correspondence with

$$\gamma = c_n \epsilon_n \oplus c_{n-1} \epsilon_{n-1} \oplus \dots \oplus c_2 \epsilon_2 \oplus c_1 \epsilon_1$$

where

$$\begin{aligned} \epsilon_1 &= (0\ 0\ 0 \dots 0\ 0\ 1) \\ \epsilon_2 &= (0\ 0\ 0 \dots 0\ 1\ 1) \\ &\vdots \\ \epsilon_{n-1} &= (0\ 1\ 1 \dots 1\ 1\ 1) \\ \epsilon_n &= (1\ 1\ 1 \dots 1\ 1\ 1) \end{aligned}$$

The vector representation of the binary number is unique. Therefore,

$$A \leftrightarrow \alpha$$

and

$$A \leftrightarrow \gamma$$

$$\alpha = \gamma$$

The α and γ vectors are identical and are representations of the same number A with respect to different bases.

Vector Length and Parity

The square of the length of a vector α is given by the inner product of α with itself. Let us consider the vector representation, α , of a binary number, A, with respect to an orthogonal set of bases, μ .

Let $A = a_n \dots a_1$

then $= a_n \mu_n \oplus \dots \oplus a_1 \mu_1$

and $= L^2 = a_n a_n \oplus \dots \oplus a_1 a_1$

The last equation expressed in congruence notation is

$$\sum_{k=1}^n a_k a_k \equiv L^2 \pmod{2}$$

The only elements of the mod 2 field are zero and one. Therefore, the length and the square of the length are identical. The length, L, of the vector representing the binary number, A, is identical to the parity of A modulo 2.

$$(\alpha\alpha) \text{ eq } (L^2) \text{ eq } (L) \text{ eq } p$$

The square of the length of a vector, γ , with respect to a non-orthogonal basis is

$$\gamma\gamma = \sum_{i,j=1}^n a_i a_j g_{ij}$$

As an example the length of the vector with respect to the cyclic permuted code basis is determined.

$$[G] = [T^{-1}]^t [T^{-1}] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & \dots \\ 1 & 0 & 0 & 0 & 0 & 0 & \dots \\ 1 & 0 & 1 & 1 & 1 & 1 & \dots \\ 1 & 0 & 1 & 0 & 0 & 0 & \dots \\ 1 & 0 & 1 & 0 & 1 & 1 & \dots \\ 1 & 0 & 1 & 0 & 1 & 0 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

The G matrix* is symmetrical, $g_{ij} = g_{ji}$. It follows from the symmetry of the G matrix and the identity

$$k \oplus k = 0$$

that the off diagonal matrix elements do not contribute to L^2 . The diagonal elements have a value

$$i = g_{ii} \text{ Mod } 2$$

The length of the vector in terms of the cyclically permuted code basis is

$$L = c_1 \oplus c_3 \oplus c_5 \oplus \dots$$

which must be equal to

$$L = a_1 \oplus a_2 \oplus a_3 \oplus \dots \oplus a_n = p$$

* The G matrix is the metric matrix. A good reference is Brillouin L. (15)

The invariance of the length permits a check of the operation of conversion from one number system to another. A check procedure involving the even digits of the cyclically permuted code may be obtained if c_1 and a_1 are deleted from the respective representations by a replacement of the form.

$$a_1 \leftarrow a_{i+1}$$

$$c_1 \leftarrow c_{i+1}$$

a_2 and c_2 are the low order elements of A and C respectively. Therefore, the following relations must be valid

$$L' = c_2 \oplus c_4 \oplus c_6 \oplus \dots$$

$$L' = a_2 \oplus a_3 \oplus a_4 \oplus \dots \oplus a_n$$

The Vector Space of Binary Addition

The standard procedure for the addition of two m digit binary numbers may be described in a vector space in terms of the vector addition of three vectors. The three vectors are the augend vector, α , the addend vector, β , and the carry vector, γ . The sum vector is denoted by

$$\sigma = \alpha \oplus \beta \oplus \gamma = \rho \oplus \gamma$$

If the basis of the vector space is orthogonal and in one to one correspondence with the powers of two from zero to $m + 1$, then the carry vector may be defined recursively as

$$c_1 = 0$$

$$c_2 = a_1 b_1$$

.

.

$$c_{j+1} = c_j(a_j \oplus b_j) \oplus a_j b_j = c_j r_j \oplus k_j$$

where $r_j = a_j \oplus b_j$

and $k_j = a_j b_j$

In column form the addition process is

$$\begin{array}{r}
 r_m \dots\dots r_3 r_2 r_1 \\
 \oplus \quad \underline{c_{m+1} \ c_m \ \dots\dots \ c_3 c_2 c_1} \\
 s_{m+1} \ s_m \ \dots\dots \ s_3 s_2 s_1
 \end{array}$$

Practical arithmetic circuitry is often designed to accept a c_1 element of unity. For example the insertion of a one digit in the low order column facilitates two complement arithmetic in a parallel machine. However, the arithmetic structures considered in this paper will be the type for which c_1 is always equal to zero.

Let the augend and the addend vectors have a maximum dimension of m . The vector addition of the addend and augend vector is closed and the partial sum, ρ , has a maximum dimension of m . The carry vector as defined above has $m + 1$ elements but the first element of the carry vector, c_1 , is always equal to zero. Therefore, the maximum dimension of the carry vector is m . The sum is obtained from the vector addition of the partial sum vector and the carry vector. The actual dimension of the vector addition of the partial sum vector and the carry vector is at most $m - 1$ since $c_1 = 0$ and $r_{m+1} = 0$. The sum vector, σ , has a maximum dimension of $m + 1$. The elements of the

sum vector are

$$\begin{aligned}
 s_{m+1} &= c_{m+1} \\
 s_1 &= r_1 && i \neq 1 \\
 &&& j \neq m + 1 \\
 s_j &= r_j \oplus c_j
 \end{aligned}$$

A vector space of dimension $m + 1$ is required if the augend vector, the addend vector and the carry vector are to be considered simultaneously. It is seldom necessary to consider all three vectors at once. The vector space usually considered in the remaining sections will be a subspace and have a dimension of m or $m - 1$. The need for a dimension of $m + 1$ to represent the sum is in exact correspondence with the binary addition of two m digit binary numbers. The $m + 1$ dimension corresponds to the $m + 1$ digit which is the overflow digit.

The actual mechanization of the process of binary addition is relatively straightforward except for the generation of the carry vector. The logic required for the formation of the partial sum e by the vector addition of the addend and augend vector, and the vector addition of e and the carry vector δ is essentially singular. The absence of intercoordinate relationships is characteristic of the process of vector addition. The carry vector elements on the other hand have a high degree of intercoordinate dependence for

$$c_{j+1} = F(a_j, b_j, \dots, a_1, b_1)$$

The lack of independence of the carry elements is the basic factor to be considered in the design of a binary adder. The logic of various

digital parallel adders differs only in the means used to generate the carry. Furthermore, the error structure of the adder is determined primarily by the characteristics of the carry vector.

The Carry Vector

The recursive definition of the carry vector is adequate for many applications including the design of certain binary adders. However, for the purposes of this paper the expression of the carry vector in matrix notation has considerable utility and constitutes a unified approach to the problem of carry generation. In matrix notation the carry vector γ is given as

$$\gamma = [D] k$$

The matrix equation in expanded form is shown in Figure 2.

The fact that the k vector and ρ vector are always orthogonal is an important characteristic of the process of carry vector generation. The proof of this orthogonality is as follows: By the definition of ρ and k we have

$$r_j = 1 \supset k_j = 0$$

$$k_j = 1 \supset r_j = 0$$

Both statements, by the definition of implication, are equivalent to,

$$\bar{r}_j \vee \bar{k}_j = 1$$

The proof is completed by application of De Morgan's Theorem.

$$\overline{\bar{r}_j \vee \bar{k}_j} = \bar{1}$$

$$r_j k_j = 0$$

$$\begin{array}{c}
 \left[\begin{array}{c} c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ c_n \\ c_{n+1} \end{array} \right] = \begin{array}{c} \left[\begin{array}{cccc} 1 & & & \\ r_2 & 1 & & \\ r_2 r_3 & r_3 & 1 & \\ r_2 r_3 r_4 & r_3 r_4 & r_4 & \dots \\ r_2 \dots r_5 & r_3 r_4 r_5 & r_4 r_5 & \dots \\ r_2 \dots r_6 & r_3 \dots r_6 & r_4 r_5 r_6 & \dots \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ r_2 \dots r_{n-1} & r_3 \dots r_{n-1} & r_4 \dots r_{n-1} & \dots 1 \\ r_2 \dots r_n & r_3 \dots r_n & r_4 \dots r_n & \dots r_n \dots 1 \end{array} \right] \left[\begin{array}{c} k_1 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \\ k_6 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ k_n \end{array} \right]
 \end{array}$$

where $r_j = a_j \oplus b_j$

and $k_j = a_j b_j$

Figure 2. Carry Vector.

Therefore:

$$e \cdot k = 0$$

The form of c_j , the j th element of the carry vector is

$$c_j = k_{j-1} \oplus k_{j-2}r_{j-1} \oplus k_{j-3}r_{j-2}r_{j-1} \oplus k_{j-4}r_{j-3}r_{j-2}r_{j-1} \oplus \dots$$

It may be concluded from the orthogonality of the k and e vectors that at most only one term in the above expression for c_j may be different from zero. This fact is seen immediately if c_j is expressed in the following form.

$$c_j = k_{j-1} \oplus r_{j-1} \left\{ k_{j-2} \oplus r_{j-2} \left[k_{j-3} \oplus r_{j-3} \left(k_{j-4} \oplus \dots \right. \right. \right.$$

In addition to giving information of the structure of the carry vector, the orthogonality of the e and k vectors has very practical consequences. In particular, since the conjunction of e and k never occurs, the convenient mathematical connective, ring addition or exclusive disjunction \oplus , may be replaced by the physically simpler connective, inclusive disjunction. The recursive definition is then

$$c_{j+1} = (a_j \oplus b_j) c_j \vee a_j b_j$$

The recursive definition may be further simplified by the use of the identities

$$a_j \oplus b_j = (a_j \vee b_j) \overline{a_j b_j}$$

and

$$\overline{xy} \vee y = x \vee y$$

The result is

$$c_{j+1} = (a_j \vee b_j) c_j \vee a_j b_j$$

The physical logic associated with this definition is somewhat simpler than that associated with the definition of the carry using exclusive disjunction connectives. In fact the carry definition in terms of the inclusive disjunction is the most frequently used definition and might have been chosen as the starting point for this work. This was not done for two reasons. First the definition in terms of exclusive disjunction is mathematically more convenient. The convenience is primarily due to the fact that a variable is self inverse for the exclusive disjunction connective. Secondly, the definition in terms of the exclusive disjunction is more restrictive than the definition in terms of inclusive disjunction. Thus, more is learned about the carry vector structure if the most restricted case is considered first. Later the restrictions may be removed one by one and the effects observed.

Let v_j denote the inclusive disjunction of the j th augend and addend elements. If r_j is replaced by v_j then any particular row of the carry matrix may contain more than one, ONE element since

$$c_j = k_{j-1} \vee k_{j-2}v_{j-1} \vee k_{j-3}v_{j-2}v_{j-1} \vee \dots$$

and

$$k_n \Rightarrow v_n$$

In the discussion which follows ρ and V may usually be used interchangeably. Special note will be given to situations where interchangeability does not exist.

Methods of Generation of the Carry Vector

The matrix relationship

$$\gamma = [D]k$$

completely defines the generation of the carry vector in terms of the k and the ρ or V vectors. Essentially instantaneous generation of the carry vector is achieved if the elements of the physical logic correspond one by one to the logical connectives found in the matrix definition of the carry vector. This particular realization of the carry logic is characterized by the fact that the elements of the carry vector are independent. The elements of the carry vectors are independent in the sense that if k and ρ or V are correct then a single error in the structure of the carry logic will cause an error in at most one element of γ . Unfortunately, for all practical purposes the number of logical elements required to realize an instantaneous and independent type of carry is prohibitive. A study of the carry vector reveals that, in addition to the logic required to generate the elements of ρ or V and k , the carry logic required of each c_j element is one OR gate and $j-2$ AND gates; $j \geq 3$. Let m be the number of elements comprising the augend and addend vectors then k_j must drive $m + 1 - j$ gate inputs and r_j or v_j must drive $(j-1) \cdot (m+1-j)$ inputs. The last formula is not valid for $j = 1$. r_1 does not influence the carry process and is not considered. r_1 is identical to the sum element s_1 . The total carry logic of the instantaneous and independent type of carry requires

with $m - 1$ OR gates
 $\frac{m(m+1)}{2} - 1$ inputs

$\frac{m(m-1)}{2}$ AND gates

with $\frac{1}{2} \sum_{j=3}^{m+1} (j - j - 2) = \frac{(m+1)(m+2)}{6} - m$ inputs

The number of the logical elements required for the instantaneous and independent carry can best be illustrated by a numerical example. The equipment needed for the carry logic of a 30 bit binary adder is

- 29 OR gates with 464 inputs
- 435 AND gates with 4930 inputs
- k_1 drives 30 gate inputs
- k_2 drives 29 gate inputs
-
-
-
- k_{30} drives 1 gate input
- * r_2 drives 29 gate inputs
- r_3 " 56 " "
- r_4 " 81 " "
- r_5 " 104 " "
-
-
-
- r_{15} " 224 " "
- r_{16} " 225 " "
- r_{17} " 224 " "
-
-
-

* r_j or v_j

r₂₉ drives 56 gate inputs
 r₃₀ " 29 " "

The independence of the elements of the carry vector may be maintained and the number of logical elements reduced if the instantaneous feature of the previous carry logic is not required. The elements of any row of the D matrix may be generated sequentially. The logic for one carry digit c_j is shown in Figure 3. The logic required for each c_j is 2^{j-5} AND gates and one OR gate. For m digits the total carry logic consists of m-1 OR gates with a total of $\frac{m(m+1)}{2}$ inputs and

$$\sum_{j=3}^{m+1} 2^j - 5 = (m+1)(m-3) + 4$$

AND gates. Each AND gate has exactly two inputs. An element of k, k_j, and an element of ρ, r_j, must each drive m-j+1 inputs. This scheme of carry generation requires the following components if m is equal to thirty as in the previous example.

29 OR gates with 464 inputs
 841 AND gates with 1682 inputs
 k₁ drives 30 gate inputs
 .
 .
 .
 k₃₀ " 1 " "
 .
 .
 *r₂ " 29 " "
 .
 .
 .
 *r_j or v_j r₃₀ " 1 " "

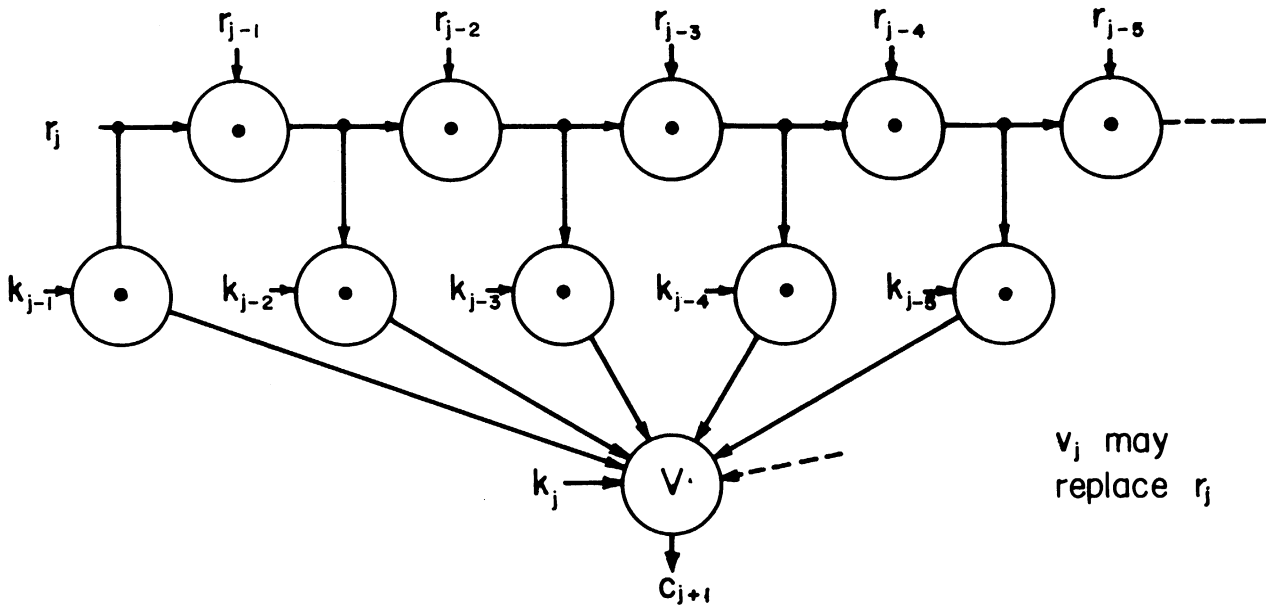


Figure 3. Logic for Independent, Sequential Generation of One Carry Element.

It is observed that realization of a carry, with independent elements, by the above procedure is by no means economical. Furthermore, the method suffers considerable time delay due to the sequential method employed to generate the elements of the D matrix. The sacrifice in time does permit a significant reduction in the required drive capabilities of the r_i or v_i elements. All in all a high price is paid in terms of either components, or time, or both, to achieve independence of the carry elements.

The generation of the carry vector in a synchronous logical structure in general leads to more extensive logic or requires more time than a corresponding asynchronous circuit which permits carry

rippling. This problem of carry generation for synchronous circuitry has been studied by Weinberger and Smith.⁽¹⁶⁾ The paper details a particular technique used to simplify the logical configuration of the D matrix. There is no indication that the technique used by Weinberger and Smith is optimal though the results are reasonable.

The carry vector may be considered to be composed of a set of sub-carry vectors according to the following relationship

$$\gamma = \gamma_m \oplus \gamma_{m-1} \oplus \dots \oplus \gamma_2 \oplus \gamma_1$$

In particular let γ_j be chosen as the j th set of diagonal elements from the matrix definition of the carry. Then

$$\begin{aligned} \gamma_1 &= (k_m, k_{m-1}, \dots, k_2, k_1, 0) \\ \gamma_2 &= (k_{m-1}r_m, k_{m-2}r_{m-1}, \dots, k_1r_2, 0, 0) \\ \gamma_3 &= (k_{m-2}r_{m-1}r_m, k_{m-3}r_{m-2}r_{m-1}, k_1r_2r_3, 0, 0, 0) \\ &\vdots \\ \gamma_m &= (k_1r_2r_3\dots r_m, 0, 0, \dots, 0, 0, 0) \end{aligned}$$

The set of carry sub-vectors as defined above form an orthogonal set and therefore the sum of the sub-vector may be obtained by the inclusive disjunction connective.

$$\gamma = \gamma_m \vee \gamma_{m-1} \vee \dots \vee \gamma_2 \vee \gamma_1$$

\oplus is replaceable by \vee if and only if the inclusive disjunctive connective is used to sum the sub-carry vectors. The sub-carry vectors may be defined recursively. The recursive definition of the sub-carry vectors corresponds to a carry realization procedure economically feasible in equipment but time consuming. The recursive definition is

$$\begin{aligned} \gamma_1 &= [2] k \\ \gamma_2 &= [2] ([2] k) = [2] (\gamma_1 e) \\ &\vdots \\ \gamma_j &= [2] (\gamma_{j-1} e) \end{aligned}$$

The notation, $[2]$, corresponds to a left shift of one element. In matrix notation

$$[2] = \begin{bmatrix} 0 & . & . & . & . \\ 1 & 0 & . & . & . \\ 0 & 1 & 0 & . & . \\ . & 0 & 1 & 0 & . \\ . & . & 0 & 1 & 0 \\ . & . & . & . & . \end{bmatrix}$$

The multiplications specified in the recursive definition of the sub-carry vectors are non-associative. The product γ_{j-1} is formed first, followed by the left shift of one element. An addition process which obtains a carry vector by means of operations utilizing the sub-vectors is sometimes termed "programmed addition" because the process can easily be executed on a basic automatic computer in the absence of an addition instruction. The generation of the carry vector from the sub-vectors was employed by Robertson⁽¹⁰⁾ as a means of preventing the propagation of errors. Error propagation is prevented if suitable error detection and correction procedures are applied after the creation of each partial sum of the carry vector. Robertson employed a Hamming

Code⁽⁵⁾ to obtain the redundancy needed to detect and correct single errors which might occur at each step in the carry generation procedure.

The simplest logic which generates the carry vector is obtained if the logic corresponds to the recursive definition of the carry vector,

$$c_2 = k_1$$

$$c_j = c_{j-1} \vee_{j-1} \vee k_{j-1}$$

or

$$c_j = c_{j-1} \vee_{j-1} \vee k_{j-1}$$

This definition may be represented by the matrix equation

$$k = [D^{-1}] \gamma$$

D^{-1} is the inverse of the D matrix which has been used previously to define the carry process. The inverse matrix exists only for the definition of D in terms of ρ .

$$[D^{-1}] = \begin{bmatrix} 1 & 0 & \cdot & \cdot & \cdot \\ r_2 & 1 & 0 & \cdot & \cdot \\ 0 & r_3 & 1 & 0 & \cdot \\ \cdot & 0 & r_4 & 1 & \cdot \\ \cdot & \cdot & 0 & r_5 & \\ \cdot & \cdot & \cdot & & \end{bmatrix}$$

The logic of this type of carry generation is shown in Figure 4. The logic is characterized by the interdependence of the carry digits and by the fact that different carry sequences may propagate simultaneously. The carry digit interdependence is completely characterized by the k and

ρ vectors. In all cases a propagation of carries is triggered by a k_j element equal to one. The carry is propagated through the successive elements only if successive elements of ρ are equal to one. The carry propagation sequence stops at r_{j+n} , the first non-unity element of the ρ vector succeeding r_j . Of course, a carry sequence may have been initiated by k_{j+n} but this constitutes a new carry sequence and not a continued propagation of a carry sequence. Due to the orthogonality of the ρ and k vectors no carry sequences overlap. If k_i initiates a carry then r_i must stop the propagation of any previous sequence. If V is substituted for ρ then the sequence may propagate through points of carry generation. This will affect the error structure of the adder but will not affect the ordinary addition process. It has been shown in Burks et al. (17) that if m is equal to 40 then the average length of the maximum carry propagation is 4.62 elements. The characteristics of the carry propagation and generation is considered in a later section.

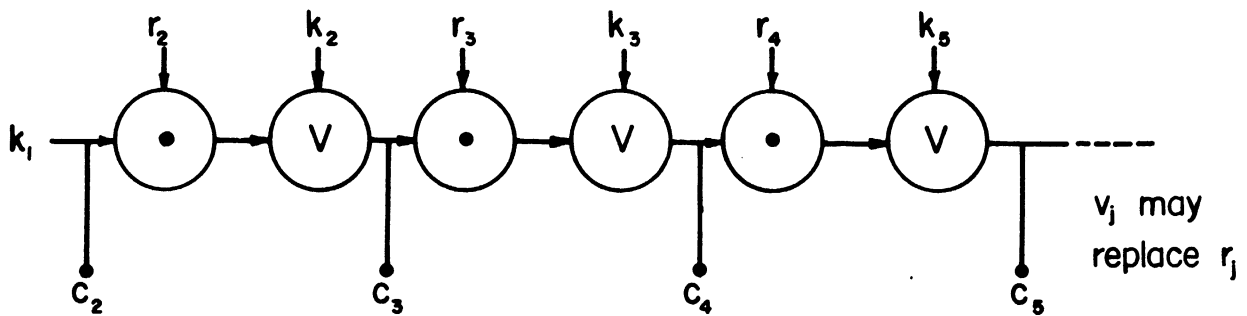


Figure 4. Standard Carry Logic.

Synthesis by Vector Space Operations

The process of binary addition for all sum digits except the first is completely defined by the matrix equation

$$\sigma = \rho \oplus [D] k$$

In the language of vector spaces the binary addition process consists of an affine transformation of the vector k . The vector k is rotated as a result of the multiplication of k by the D matrix. The result of the multiplication is then translated by the vector addition of the ρ vector. The whole process is complicated by the fact that the D matrix is a function of the ρ vector.

New logical configurations for the process of binary addition may be synthesised by either of two procedures: (1) by the factorization of the D matrix and (2) by a change of basis transformation over the whole addition process. The different logical configurations discussed in the previous section were obtained as a result of different factorizations of the D matrix.

One factorization of the D matrix follows from the definition of the D matrix as

$$D = [D^{-1}]^{n-1} [D]^n$$

The resulting carry logic has a logical circuit depth of two. If a value of n equal to 5 is chosen the resulting logic is considerably simpler than instantaneous carry logic but not as simple as the multi-depth logic proposed by Weinberger and Smith.⁽¹⁶⁾ A factorization in this manner can lead to fast carry logic which is economical in terms

of the number of logical components. However, the factorization does not lead to a carry logic particularly appropriate for error checking.

The difficulties encountered when a change of basis transformation on the whole addition process is attempted is due to the fact that the D matrix is a function of the ρ vector. Consider the following transformation

$$[T] \sigma = [T] \rho \oplus [T] [D] k$$

While σ , ρ and k may be considered in terms of a new basis the elements of the D matrix are given with respect to the original basis. Thus the transformed addition process defined as

$$\sigma' = \rho' \oplus [D] k'$$

cannot be expected to lead to a simpler definition of the carry process because of the functional dependence of the D matrix on the ρ vector. Thus the transformed addition process is to be regarded in terms of the original basis. A simple transformation of this type and in all probability the only practical transformation of this type is obtained if the transformation matrix is identical to the inverse D matrix. The multiplication of the matrix definition of the sum by the inverse D matrix yields

$$[D^{-1}] \sigma = [D^{-1}] \rho \oplus k$$

The indices have the same range as specified above. The matrix equation involving the inverse D matrix specifies a recursive definition for the sum elements.

$$\begin{aligned}
 s_2 &= r_2 \oplus k_1 \\
 s_3 &= r_2 (1 \oplus s_2) \oplus r_3 \oplus k_2 \\
 &\vdots \\
 &\vdots \\
 s_j &= r_{j-1} (1 \oplus s_{j-1}) \oplus r_j \oplus k_{j-1}
 \end{aligned}$$

As a result of the orthogonality of k and ρ the equations are reduced to the form

$$s_j = (r_{j-1} \bar{s}_{j-1} \vee k_{j-1}) \oplus r_j$$

The logic corresponding to this particular method is shown in Figure 5. The circuit has a very prominent weakness. A propagated carry must pass through four logical elements per stage rather than the two elements per stage required in the conventional scheme of Figure 4. The count is four logical elements per stage rather than three because the usual realization of the exclusive disjunction requires a cascade of two logical elements. The circuit was included here to illustrate a particular method of generating alternative logical structures by matrix manipulation.

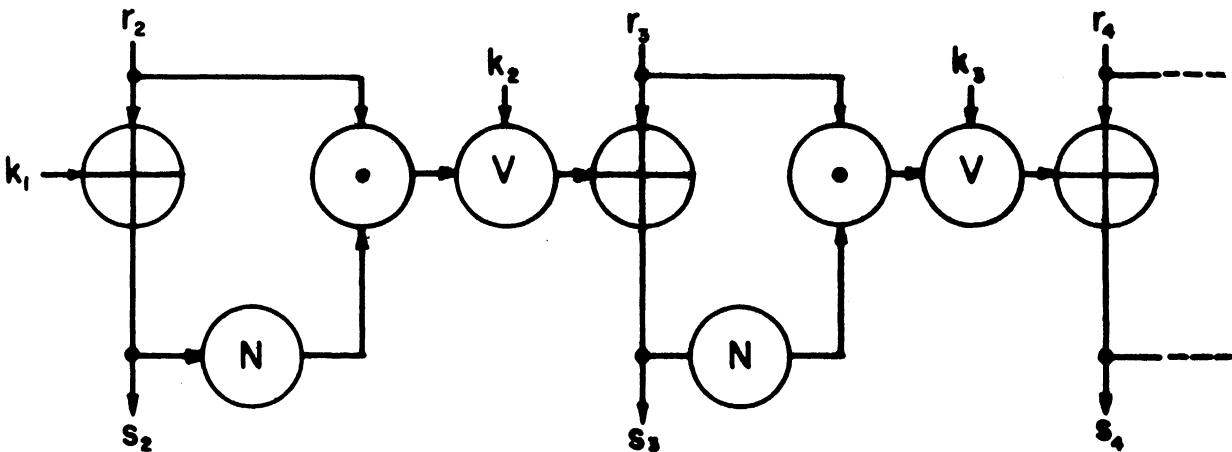


Figure 5. Sum Dependent Carry Logic.

Numerical Relationships

In this section the dimensionality concept is used to establish particular numerical relationships which exist between the results of specific logical operations. One important consequence of the existence of the numerical relationships is the feasibility of using numerical checking and correcting procedures to control the errors produced by faulty elements in a logical structure. The numerical relationships will be used in the next section to specify alternate algorithms of the addition process.

Consider the dimensionality relationship

$$D(\alpha) + D(\beta) = D(V) + D(k)$$

where

$$V = \alpha \vee \beta$$

and

$$k = \alpha \beta$$

The dimensionality relationship must be valid for any set of coordinates. If a single coordinate is considered then the following relationships exist between the values of a_j , b_j and v_j , k_j .

TABLE I

RELATION BETWEEN a_j , b_j and v_j , k_j

a_j	b_j	v_j	k_j
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Digitally the difference between a_i , b_i and v_i , k_i is a possible zero and one interchange between columns as shown in row two of Table I.

The interchange occurs if v_j and k_j are substituted for a_j and b_j but there is no numerical effect since

$$0 + 1 = 1 + 0$$

Therefore, the dimensionality relationship implies the numerical relationship

$$\sigma = \alpha + \beta = V + k$$

A second numerical relationship is obtained from the dimensionality equation.

$$D(\rho) + D(k) = D(V)$$

where

$$\rho = \alpha \oplus \beta$$

The numerical relationship is

$$\rho + k = V$$

The validity of this equation is established by the following argument. The ρ and k vectors are orthogonal and therefore the magnitude of the sum of a particular coordinate, $r_j + k_j$, cannot exceed unity. The addition of ρ and k can never produce a carry. This fact and the implication

$$k_j \supset v_j$$

assure the validity of the numerical relationship between ρ , k and v .

The dimension relationships may be expanded to include more than a pair of variables. In particular, the following dimension equation is obtained by considering a Venn diagram with three overlapping classes or by application of the previously derived dimension

formula of two variables.

$$D(\alpha v \beta v \gamma) = D(\alpha) + D(\beta) + D(\gamma) - D(\alpha\beta) - D(\alpha\gamma) - D(\beta\gamma) + D(\alpha\beta\gamma)$$

The corresponding numerical relationship is

$$(\alpha v \beta v \delta) = \alpha + \beta + \delta - \alpha\beta - \alpha\delta - \beta\delta + \alpha\beta\delta$$

The following examples are given to illustrate the numerical relationship previously described.

Let $\alpha = (1\ 0\ 1\ 1\ 0\ 1) = 45$

and $\beta = (1\ 1\ 1\ 0\ 0\ 1) = 57$

then $V = (1\ 1\ 1\ 1\ 0\ 1) = 61$

$$k = (1\ 0\ 1\ 0\ 0\ 1) = 41$$

$$\rho = (0\ 1\ 0\ 1\ 0\ 0) = 20$$

now $\alpha + \beta = V + k$

$$45 + 57 = 61 + 41 = 102$$

and $\rho + k = V$

$$20 + 41 = 61$$

Now let us consider the extension to three variables.

Let $\delta = (1\ 1\ 0\ 0\ 1\ 0) = 50$

then $(\alpha v \beta v \delta) = \alpha + \beta + \delta - \alpha\beta - \alpha\delta - \beta\delta + \alpha\beta\delta$

$$63 = 45 + 57 + 50 - 41 - 32 - 48 + 32$$

$$63 = 63$$

In this section we have shown the existence of two relationships which can be employed to check the generation of the ρ , k and V elements which are functions of the addend and augend elements. The most important characteristic of the derived checks is the fact that

the checks are numerical. This means that the checks may be applied on a digit by digit basis, as a check on certain parts of a representation or as a check of all elements of the representation. Both check processes are of the type which detect the occurrence of a single error. The check

$$\alpha + \beta = V + k$$

will detect malfunctions in the V and the k elements if the α and β elements are correct. Likewise the check

$$\rho + k = V$$

may be used to check errors in the ρ elements only when the k and the V elements are correct.

It is logical to ask whether similar check procedures exist for the carry generation process. Unfortunately the answer is no. The checks given above owe their existence to either the absence of carries or the presence of identical carry processes appearing on opposite sides of the equality sign. The carry generation process is characterized by the presence of constraints and the dependent nature of the elements of the carry vector. A check of the carry generation process can only be made by either duplicating the carry generation or by partitioning the carry generation process into checkable parts as is done for the carry generation process used in programmed addition.

Addition Algorithms

The numerical relationships derived from the dimension concept in the previous section may be used to obtain alternate algorithms for binary addition.

The relationship

$$\sigma = \alpha + \beta = V + k$$

$$V = \alpha \vee \beta$$

$$k = \alpha \beta$$

states that the sum of the augend and addend is identical to the sum of the disjunction and the conjunction of the augend and the addend.

The same carry vector is associated with either algorithm for

$$\rho = \alpha \oplus \beta = V \oplus k$$

$$k = \alpha \beta = kV$$

The numerical relationships

$$\sigma = \alpha + \beta = V + k$$

and

$$\rho + k = V$$

$$\rho = \alpha \oplus \beta$$

may be combined linearly to give two binary addition algorithms of the form

$$\sigma = k + k + \rho = 2k + \rho$$

$$\sigma = V + V - \rho = 2V - \rho$$

In the remaining part of this paper the following terminology is adopted. The addition algorithm described by the first equation above is called $k\rho$ addition. The second process is called $V\rho$ addition. The usual addition procedure is called standard addition and defined by $\sigma = \alpha + \beta$. The process $\sigma = V + k$ is termed Vk addition.

The concept of dimensionality has led to the definition of alternative algorithms for addition. The important aspect of the $k\rho$

and V_p algorithms is that the carry or borrow vectors associated with these processes differ from the carry vector associated with the standard and the V_k addition processes. The fact that the k_p carry vector and the V_p borrow vector are different from the standard carry vector is sufficient reason to investigate in detail the addition algorithms. The characteristics of the carry or borrow vector determine both the propagation of errors and the average addition time of a parallel type binary adder with asynchronous carry.

Constraints and Some Properties of the Addition Algorithms

Constraints play an important role in the determination of the characteristics of the k_p and V_p addition processes. The constraints and some of the effects of the constraints are discussed in this section.

The constraint of the k_p addition process is

$$k_j p_j = 0$$

Consider the addition process as expressed in column form

$$\begin{array}{rcccc}
 & \dots\dots\dots & k_3 & k_2 & k_1 \\
 + & \dots\dots\dots & r_4 & r_3 & r_2 & r_1 \\
 \hline
 & \dots\dots\dots & s_4 & s_3 & s_2 & s_1
 \end{array}$$

The sum of any column is given by the vector sum of the k , and carry elements.

$$\sigma = 2k \oplus p \oplus \delta$$

The effect of the constraint is to limit the number of elements with a value of one in any column. The number of ones in any column cannot exceed two. The proof of this statement is obtained by considering the following situation. Let k_{j-1} and r_j both have a value of one so that a carry is generated. Then r_{j-1} must be equal to zero and no carry from a coordinate or column less than $j-1$ is propagated through $j-1$. It is also obvious that a carry cannot be generated at column $j-1$. The number of ones in columns $j-1$ and j is at most two. In column $j+1$, k_j is constrained to a value of zero and c_{j+1} the carry element of the column has a value of one. If r_{j+1} is equal to zero the $j+1$ column has only one digit of value one. If r_{j+1} is equal to one there are two one digits in the column. Thus, three one digits never occur in any column and a two input type of adder is sufficient. However, no overall saving of components results since a two input adder is required to form each ρ , k element pair. One other effect of the constraint in 1. ρ type addition is that carry propagation is determined solely by the elements of ρ . If the carry is propagated then r_{j+1} must be equal to one. But if this is true, then k_{j+1} is equal to zero and r_{j+2} must equal one to propagate the carry. The carry continues to propagate until r_n is equal zero.

The following example is presented as an illustration of the $k\rho$ algorithm

$$\begin{array}{l} \text{Given:} \quad \alpha = (1\ 1\ 0 \quad 0\ 1\ 1 \quad 1\ 0\ 1) \\ \quad \quad \quad \beta = (1\ 1\ 0 \quad 0\ 0\ 0 \quad 1\ 0\ 1) \\ \text{then} \quad \quad \rho = (0\ 0\ 0 \quad 0\ 1\ 1 \quad 0\ 0\ 0) \\ \quad \quad \quad k = (1\ 1\ 0 \quad 0\ 0\ 0 \quad 1\ 0\ 1) \end{array}$$

the sum is obtained by the addition

$$\begin{array}{r} \sigma = 2k + \rho \\ \quad \quad \quad 000\ 011\ 000 \\ + \\ \quad \quad \quad \underline{1\ 100\ 001\ 01} \\ \quad \quad \quad 1\ 100\ 100\ 010 \end{array}$$

the carry vector of the process is

$$\gamma = (0\ 000\ 110\ 00)$$

For this example the carry vector associated with standard addition is

$$(1\ 100\ 111\ 01)$$

The $V\rho$ algorithm uses subtraction to obtain the sum. It is therefore appropriate to discuss the generation and propagation of borrows rather than carries. A borrow is generated only if the minuend element has a value zero and the corresponding subtrahend element a value one. The borrow propagates through succeeding elements if the subtrahend and minuend elements have identical values. The borrow h_j is given the equation

$$h_j = (x_{j-1} \text{ eq } y_{j-1}) h_{j-1} \vee \bar{x}_{j-1} y_{j-1}$$

The above form of the equation is appropriate for probabilistic consideration since the causes of the propagation and the generation of the borrows are separated. In practice a more convenient logic is

$$h_j = (\bar{x}_{j-1} \vee y_{j-1}) h_{j-1} \vee \bar{x}_{j-1} y_{j-1}$$

The $V\rho$ addition process is constrained by the relations:

$$\begin{aligned} & r_j \vee v_j \\ \text{or} \quad & \bar{r}_j \vee v_j = 1 \end{aligned}$$

In column form the addition process is represented by

$$\begin{array}{rcccc} & \dots\dots & v_3 & v_2 & v_1 \\ - & \dots\dots & r_4 & r_3 & r_2 & r_1 \\ \hline & \dots\dots & s_4 & s_3 & s_2 & s_1 \end{array}$$

The constraint has the following effects. A borrow is generated if

$$\bar{v}_{j-1} r_j = 1$$

The borrow propagates the succeeding column if r_{j+1} is of value unity. v_j does not influence the borrow propagations since the value of v_j is constrained to one if r_j is equal to one. Thus the constraining relationship propagates with the carry. In the normal subtraction process the borrow is propagated if equality exists between the minuend and the subtrahend elements. In the $V\rho$ algorithm a borrow is propagated only if both the minuend and the subtrahend are unity. In practice only a two input subtractor is required but a two input adder is required to generate each pair of ρ and V elements.

The previous example is considered in terms of the $V\rho$ algorithm.

$$\alpha = (1\ 1\ 0\ \quad 0\ 1\ 1\ \quad 1\ 0\ 1)$$

$$\beta = (1\ 1\ 0\ \quad 0\ 0\ 0\ \quad 1\ 0\ 1)$$

$$\rho = (0\ 0\ 0\ \quad 0\ 1\ 1\ \quad 0\ 0\ 0)$$

$$V = (1\ 1\ 0\ \quad 0\ 1\ 1\ \quad 1\ 0\ 1)$$

$$\begin{array}{r} 1\ 1\ 0\ 0\ \quad 1\ 1\ 1\ \quad 0\ 1 \\ -\quad\quad\quad 0\ 0\ 0\ \quad 0\ 1\ 1\ \quad 0\ 0\ 0 \\ \hline 1\ 1\ 0\ 0\ \quad 1\ 0\ 0\ \quad 0\ 1\ 0 \end{array}$$

In this example the borrow vector was the null vector. Consider as a second example

$$\alpha = (1\ 1\ 1\ \quad 1\ 1\ 0)$$

$$\beta = (0\ 0\ 1\ \quad 0\ 0\ 1)$$

$$\rho = (1\ 1\ 0\ \quad 1\ 1\ 1)$$

$$V = (1\ 1\ 1\ \quad 1\ 1\ 1)$$

$$\begin{array}{r} 1\ 1\ 1\ 1\ \quad 1\ 1 \\ -\quad\quad\quad 1\ 1\ 0\ \quad 1\ 1\ 1 \\ \hline 1\ 0\ 0\ 0\ \quad 1\ 1\ 1 \end{array}$$

The borrow vector H is

$$H = (0\ 0\ 0\ \quad 1\ 1\ 1)$$

An additional algorithm for binary addition consisting of both $k\rho$ and $V\rho$ addition processes may be verified by the comparing

of the carry and borrow characteristics of k_e and V_e addition respectively. The new algorithm will be called $k_e V_e$ addition and is now described. The process requires the generation of vectors e , $2k$ and $2V$. The addition process is started with the k_e algorithm at the low order column. The k_e algorithm is applied to succeeding columns until a column is found which would generate a carry. Either the k_e sum or the V_e sum is recorded for the carry generating column but the next column is summed using the V_e algorithm. The V_e algorithm is used until a column is found which generates a borrow. The k_e or the V_e sum is recorded for this column but the sum for all succeeding columns, until a carry is generated, is found by the k_e algorithm. At the columns which generate a carry or a borrow both k_e and V_e addition give the same result. The procedure is continued until the final digit is summed. The validity of the algorithm is seen in terms of the constraints of the k_e and V_e addition processes. The first column never generates a carry for the k_e algorithm since no element of $2k$ corresponds to r_1 . The k_e algorithm is then applied to the second column. Let us assume that column j will generate a carry, i.e., $(r_j = 1, k_{j-1} = 1)$. The constraints dictate that $v_{j-1} = 1$, $r_{j-1} = 0$ and $v_j = 1$. For column j the k_e and V_e algorithms give the same sums since

$$r_j \oplus k_{j-1} = r_j \oplus v_{j-1} = 0$$

The V_e algorithm may be substituted for the k_e algorithm at the j th column only if there is no borrow propagation into the j th column.

This is assured since $r_{j-1} = 0$ and $v_{j-2} = 0$ if no borrow is propagated from the $j-2$ column and $v_{j-2} = 1$ if a borrow is propagated from the $j-2$ column. The $k\rho$ algorithm may be substituted for the $V\rho$ algorithm when a borrow occurs at the p th column if there is no carry propagation into the p th column. The borrow occurs when $r_p = 1$ and $v_{p-1} = 0$. Therefore, $k_{p-1} = 0$, and $r_{p-1} = 0$. If $k_{p-2} = 1$ there can be no carry sequence into the $p-1$ or the p column. Thus it has been shown that due to the constraint the specified algorithm change is valid.

At the moment the $k\rho V$ algorithm is only of academic interest. No practical logical configuration has been found for the $k\rho V$ algorithm which is not identical to some simpler algorithm. In fact the circuits examined contained unnecessary redundancy. However, the $k\rho V$ algorithm does constitute an interesting description of the addition process. An example of the $k\rho V$ algorithm is now given

Let	$\alpha =$	$(1\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1)$																								
	$\beta =$	$(0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0)$																								
then	$k =$	$(0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0)$																								
	$V =$	$(1\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0)$																								
	$\rho =$	$(1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1)$																								
<table style="margin-left: 100px; border-collapse: collapse;"> <tr> <td style="padding-right: 20px;">$0\ 1\ 0\ 0$</td> <td style="padding-right: 20px;">$0\ 1\ 0$</td> <td style="padding-right: 20px;">$1\ 0$</td> <td>$2k$</td> </tr> <tr> <td>$1\ 1\ 1\ 1$</td> <td>$0\ 1\ 1$</td> <td>$1\ 0$</td> <td>$2V$</td> </tr> <tr> <td>$1\ 0\ 1$</td> <td>$1\ 0\ 0$</td> <td>$1\ 0\ 1$</td> <td>ρ</td> </tr> <tr> <td colspan="3" style="border-top: 1px solid black; padding-top: 2px;"></td> <td></td> </tr> <tr> <td style="border-top: 1px solid black; padding-top: 2px;">$1\ 0\ 0\ 1$</td> <td style="border-top: 1px solid black; padding-top: 2px;">$1\ 1\ 1$</td> <td style="border-top: 1px solid black; padding-top: 2px;">$0\ 0\ 1$</td> <td style="border-top: 1px solid black; padding-top: 2px;">$k\rho$</td> </tr> <tr> <td colspan="3" style="border-top: 1px solid black; padding-top: 2px;"></td> <td style="border-top: 1px solid black; padding-top: 2px;">$V\rho$</td> </tr> </table>			$0\ 1\ 0\ 0$	$0\ 1\ 0$	$1\ 0$	$2k$	$1\ 1\ 1\ 1$	$0\ 1\ 1$	$1\ 0$	$2V$	$1\ 0\ 1$	$1\ 0\ 0$	$1\ 0\ 1$	ρ					$1\ 0\ 0\ 1$	$1\ 1\ 1$	$0\ 0\ 1$	$k\rho$				$V\rho$
$0\ 1\ 0\ 0$	$0\ 1\ 0$	$1\ 0$	$2k$																							
$1\ 1\ 1\ 1$	$0\ 1\ 1$	$1\ 0$	$2V$																							
$1\ 0\ 1$	$1\ 0\ 0$	$1\ 0\ 1$	ρ																							
$1\ 0\ 0\ 1$	$1\ 1\ 1$	$0\ 0\ 1$	$k\rho$																							
			$V\rho$																							

Evaluation of the $k\rho$ and the $V\rho$ Logic

The generation of the k , V or ρ elements required by the $k\rho$ or the $V\rho$ addition processes can easily be realized by the conventional half adder circuitry shown in Figure 6. Additional logic must be provided to generate and propagate carries or borrows and also to perform the vector addition of the k , ρ , and carry elements or the vector subtraction of the V , ρ , and the borrow elements. Due to the effect of the constraints the vector addition or subtraction operations require only one modified half adder or modified half subtractor per adder stage. The logic required for the $k\rho$ algorithm is shown in Figure 7. The logic required for the $V\rho$ algorithm is shown in Figure 9.

It is necessary to compare the $k\rho$ and the $V\rho$ logical configurations with the standard adder configurations in order to determine whether the new algorithms have any merit. The logic of a standard adder configuration is shown in Figure 8. A comparison of the logic shown in Figure 8 and that of Figure 7 reveals immediately the minor differences between the $k\rho$ logic and the standard logic. The difference is in the point at which the carry is obtained from the carry logic. For the $k\rho$ logic the carry is obtained from the output of an AND gate, For the standard logic the carry is obtained from the output of an OR gate. The change in the point at which the carry is derived has no effect on the performance of the adder circuit. We conclude therefore that the $k\rho$ adder offers no advantage over the standard adder

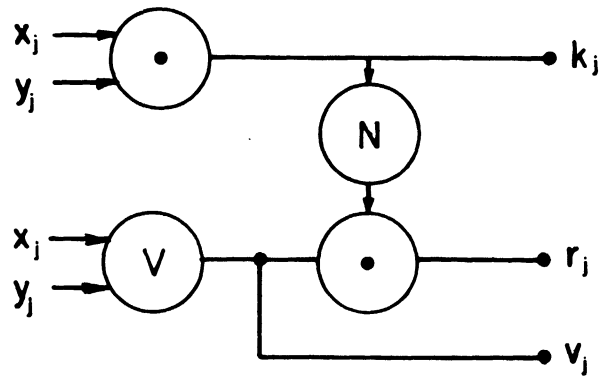


Figure 6. Half Adder.

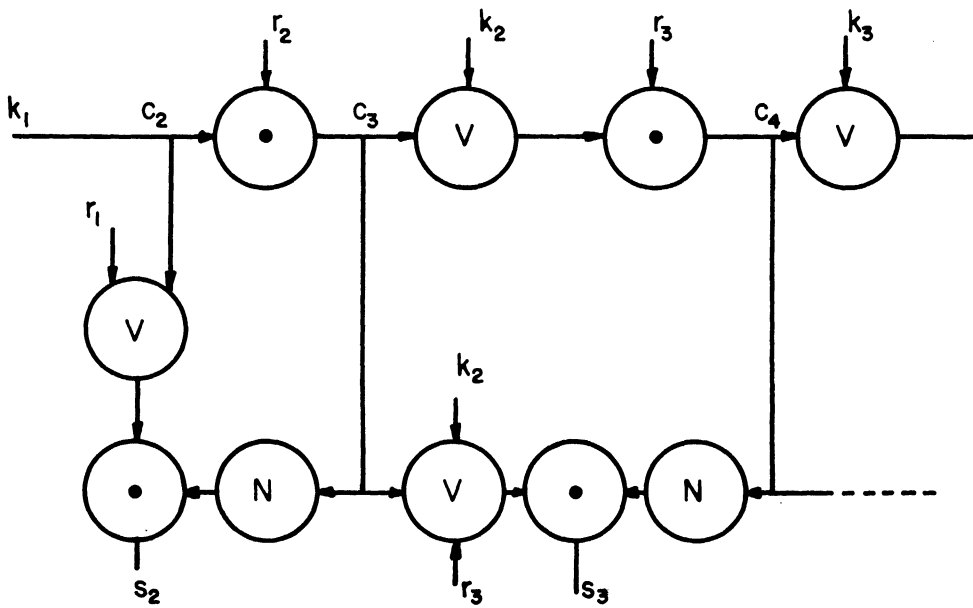


Figure 7. Realization of the k_p Addition Algorithm.

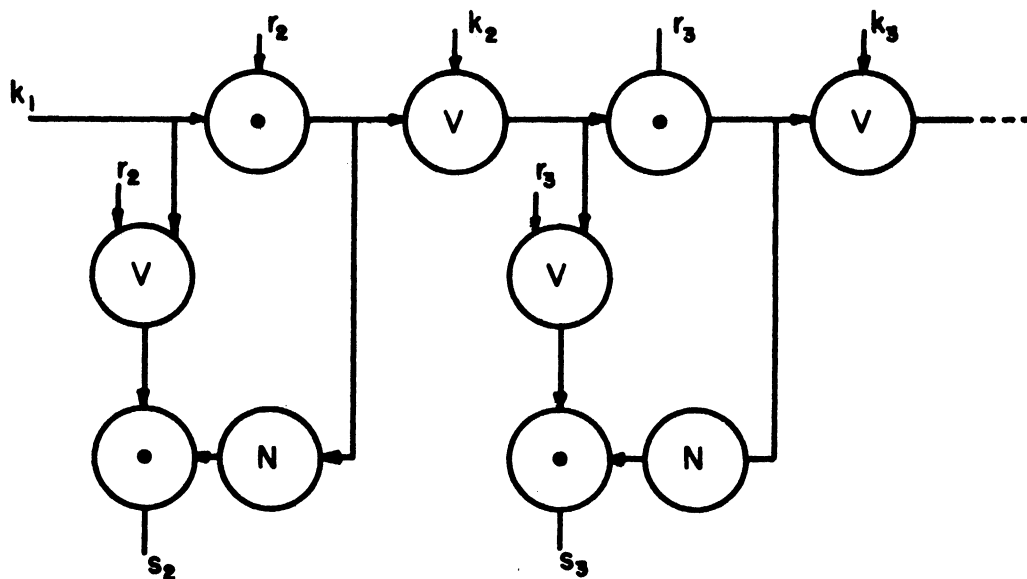


Figure 8. Logic for the Standard Addition Algorithm.

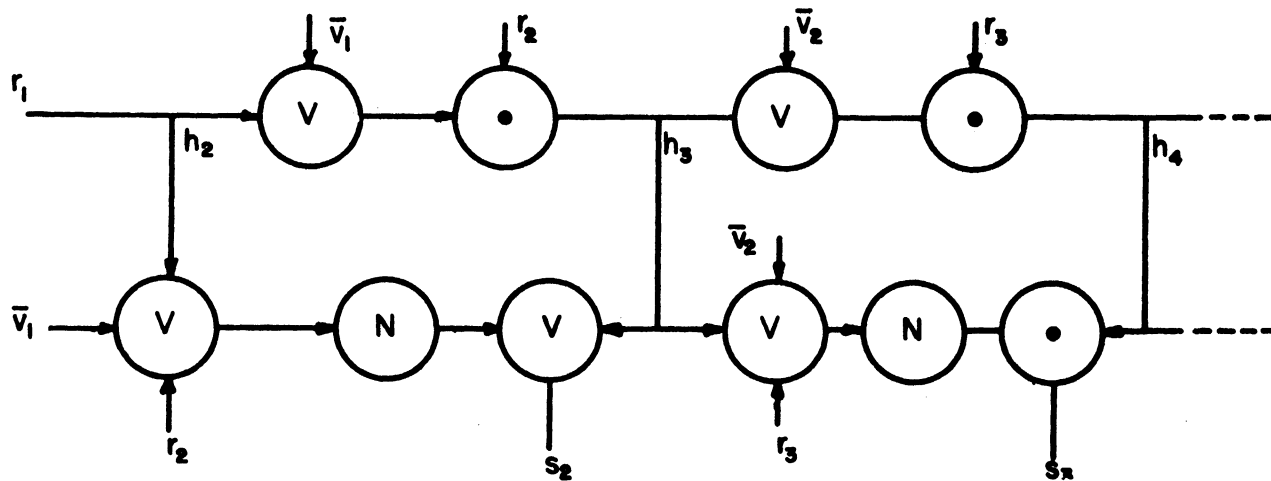


Figure 9. Logic for the vp Addition Algorithm.

configurations. The same is true of the $V\rho$ adder when compared against standard half adder - half subtractor configurations. However, the change in the point of carry derivation does change the statistics of the carry generation process. An analysis of the statistics of the carry process for $k\rho$ or $V\rho$ addition would reveal that the probability of carry initiation is equal to $1/8$ and the probability of carry propagation is equal to $1/2$. For the standard addition algorithm the probability of carry initiation is $1/4$ and the probability of carry propagation is equal to $1/2$. However, these statistics can be very misleading. The probabilities associated with the inputs to the AND gate from which the sum element is obtained are identical for all the addition algorithms. In view of this we must conclude that the new addition algorithms presented here form interesting alternate descriptions of the addition process. But no logical configurations have been found which differ substantially from the standard logical configurations. In view of this the discussion in the next chapter is restricted to the standard addition algorithm.

CHAPTER IV

THE ERROR STRUCTURE OF BINARY ADDITION AND THE EFFECTIVENESS OF THE PARITY CHECK

Introduction

The sections which follow contain a detailed investigation of the effects of logical malfunctions on the correct sum representation for standard binary addition. The carry logic is the most important single factor which determines the error structure. If it were not for the carry process, the sum elements could be completely independent and the error structure of the addition process would be extremely simple.

The logic of binary addition is divided into three parts. The first part consists of a half adder circuit which obtains the k , r and v elements from the addend and augend elements. This logic will be referred to as the generating logic. The second part of the logic of the adder is the carry logic. A standard carry logic will be considered. The carry logic requires one AND gate and one OR gate per digit. Initially the carry logic is assumed to have no logical elements in common with any other part of the adder logic. The third part of the adder logic performs the vector addition of a carry element and an element of the ρ vector to produce the sum element and is called the rc sum logic.

The form of the standard adder logic under consideration is shown in Figure 10. Conventionally gates g_5 and g_6 are combined since

the output of gate g_6 provides the required input to gate g_4 . In the immediate discussion which follows the separation of g_5 and g_6 will be assumed, though consideration will be given to the possibility that the input of g_5 is either r_i or v_i .

In the last sections of this chapter the effectiveness of both digital and numerical error detecting methods is considered for the different error structures associated with particular malfunctions of circuit components.

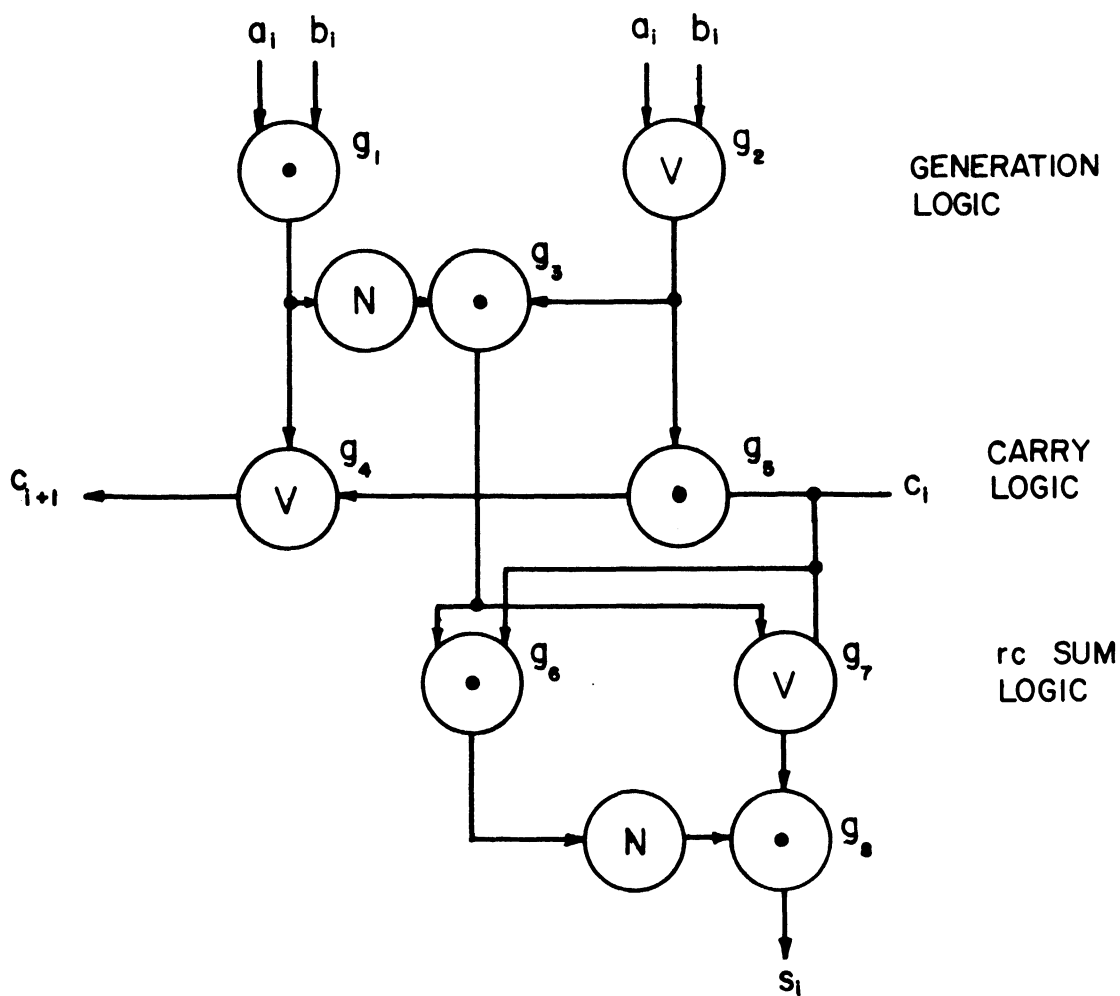


Figure 10. Adder Logic.

The Error Structure of the Carry Logic of Standard Addition

In this section the effects of various malfunctions in the standard carry logic are considered. The matrix representation of the carry logic is particularly appropriate for the study of the carry error structure. The elements of the vectors of the addition process are considered as random variables. The probability that an element k_i of the k vector equals one is $1/4$. The probability that an element r_i of the ρ vector equals one is $1/2$. If these values are substituted into the carry matrix shown in Figure 2 the result will be the matrix shown in Figure 11. The matrix defines the probability $p(c_i)$ that a given element c_i of the carry vector has a value of unity. The matrix representation of the carry vector in terms of the k and ρ elements is particularly appropriate for probabilistic studies since the elements of any row of the carry matrix are mutually exclusive. The probability that a given carry element is of value unity is obtained simply by summing the corresponding row elements and multiplying the sum by $1/4$. If the carry matrix were defined in terms of the k and V elements it would be necessary to consider the fact that the matrix elements in terms of k and V vector elements are not mutually exclusive. However, the values for $p(c_i)$ would be identical, since the same carry vector must be obtained regardless of whether the carry matrix is defined in terms of ρ and k or V and k vector elements. It is simpler to use the carry matrix in terms of k and ρ vector elements because of the mutually exclusive properties. However,

the carry vector determined by k and ρ vector elements will have a different error structure from the carry vector defined in terms of k and V vector elements. Both definitions provide the same results if there are no logical malfunctions. It is only when circuit malfunctions are present that the difference in the error structure of the two schemes becomes apparent. For reasons of mathematical convenience the present studies of the error structure of the carry vector are continued in terms of elements of the ρ and k vectors. $p(c_i)$, the probability that $c_i = 1$, obtained from the matrix representation of Figure 11, is

$$\begin{aligned} p(c_j) &= 1/4 (2 - 2^{-i+2}) \\ &= 1/2 - 2^{-i} \quad 2 \leq i \leq m+1 \end{aligned}$$

Consider the carry logic shown in Figure 12. The logic represents that required for standard addition. The logic is implemented using semi-conductor diodes. A study of the circuit will reveal that the following correspondences exist between open diodes and k_i or r_i constraints

d_{40}	corresponds to	$k_i = 0$
d_{20}	"	$r_i = 1$
d_{30}	"	$r_i = 0$
d_{10}	"	$c_i = 1$

But $c_i = 1$ if $k_{i-1} = 1$

Thus the effect of a certain diode malfunction may be analyzed by a constraint on the k_i or r_i elements.

The result of diode shorts is now determined for the carry logic of Figure 12. In the practical circuitry the parameters k_i and r_i are usually derived from voltage sources. The effect of a short is, therefore, somewhat dependent on clipping levels. If d_4 is shorted the effect is the same as obtained for d_3 open. If d_3 is shorted, the result is indeterminate since the circuit will probably continue to function until d_2 or d_4 malfunctions. A short in d_2 renders the succeeding carry elements independent of the previous carry elements and c_{i+1} becomes a function of only r_i and k_i which are orthogonal or mutually exclusive. The same effect is obtained if d_1 has an open. If d_1 shorts, the carry propagation of the i th stage in the carry logic is independent of r_i and is equivalent to d_2 open. The effects of shorted diodes are now summarized.

d_{4s}	corresponds to $r_i = 0$
d_{3s}	indeterminate
d_{2s}	corresponds to $k_{i-1} = 1$
d_{1s}	corresponds to $r_i = 1$

Let the probability of a short or an open be identical and neglect the inferred result of a short in d_3 . The frequency of occurrence of particular values of k_{i-1} , k_i and r_i in correspondence with the diode malfunctions is shown in Table II.

$$\begin{bmatrix} p(c_2) \\ p(c_3) \\ p(c_4) \\ p(c_5) \\ p(c_6) \\ \cdot \\ \cdot \\ \cdot \\ p(c_i) \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ p(c_{m+1}) \end{bmatrix} = \begin{bmatrix} 1 & & & & & & & \\ 1/2 & 1 & & & & & & \\ 1/4 & 1/2 & 1 & & & & & \\ 1/8 & 1/4 & 1/2 & 1 & & & & \\ 1/16 & 1/8 & 1/4 & 1/2 & 1 & & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & & \\ p(c_i) & 2^{-n+2} & 2^{-n+3} & 2^{-n+4} & 2^{-n+5} & 2^{-n+6} & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & & \\ p(c_{m+1}) & 2^{-m+1} & 2^{-m+2} & 2^{-m+3} & 2^{-m+4} & 2^{-m+5} & \dots\dots\dots & 1 \end{bmatrix} \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ 1/4 \end{bmatrix}$$

Figure 11. Probability that $c_1 = 1$.

TABLE II

FREQUENCY OF OCCURRENCE OF DIFFERENT
TYPES OF CARRY LOGIC MALFUNCTIONS

Element Value	Frequency
$k_i = 0$	1/8
$k_{i-1} = 1$	1/4
$r_i = 0$	1/4
$r_i = 1$	1/4
no effect	1/8

The changes in the carry error structure if V is substituted for ρ are simple. In reference to the diode logic of Figure 12 only one circuit malfunction is sensitive to the replacement of ρ by V. If diode d_4 is open, 1/8 of the time an error will not be observed. This is due to the fact that if $c_i = 1$ and $k_i = 1$ then $v_i = 1$ but $r_i = 0$. The value of $p(c_i) = 1/2$ and $p(k_i) = 1/4$. The probability that the malfunction does not cause an error is the product $p(k_i) p(c_i)$ and is equal to 1/8. The result of the substitution of V for ρ is a change in the frequency of occurrence of the $k_i = 0$ and the "no effect" type of error. The frequency of $k_i = 0$ is 7/64 and the frequency of the "no effect" condition is given as 9/64.

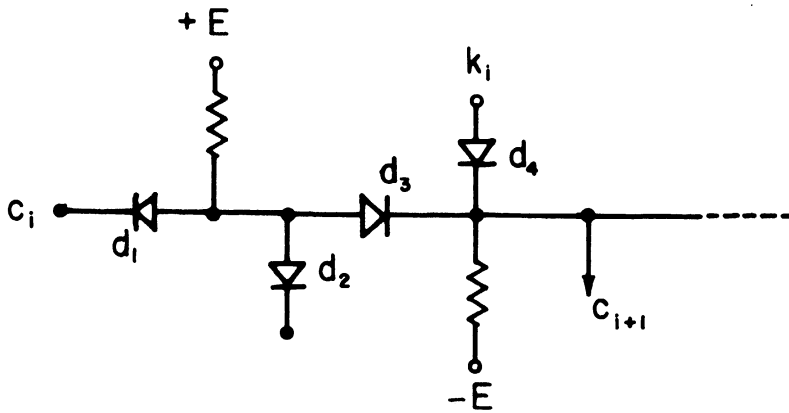


Figure 12. Carry Logic.

A malfunction in the carry logic does not necessarily cause an error in the element of the carry representation. A malfunction which does produce at least one error is termed effective. We now proceed to calculate the effectiveness of the various logical malfunctions in terms of the k_i or r_i constraints. In the following equations $p(\Delta c_{i+1})$ denotes the probability that the $i + 1$ th element of the carry vector is changed due to a malfunction represented by some constraint.

$r_i = 1$ type of malfunction

$$p(c_{i+1}) = p(k_i) + p(r_i) p(c_i)$$

$r_i = 1$ then

$$p(c_{i+1}) = p(k_i) + p(c_i) - p(k_i) p(c_i)$$

$$p(\Delta c_{i+1}) = p(c_i) - p(k_i) p(c_i) - p(r_i) p(c_i)$$

$$= p(c_i) \{ 1 - p(k_i) - p(r_i) \}$$

$$= p(c_i) \{ 1 - [p(k_i) + p(r_i)] \}$$

$$= 1/4 p(c_i) = 2^{-2}(2^{-1} - 2^{-1}) \cong 1/8$$

$r_i = 0$ type of malfunction

$$p(c_{i+1}) = p(k_i) + p(r_i) p(c_i)$$

$r_i = 0$ then

$$p(c_{i+1}) = p(k_i)$$

$$p(\Delta c_{i+1}) = p(r_i) p(c_i) = (2^{-2} - 2^{-i-1}) \cong 1/4$$

$k_i = 1$ type of malfunction

$$p(c_{i+1}) = p(k_i) + p(r_i) p(c_i)$$

$$p(c_{i+1}) = 1$$

$$p(\Delta c_{i+1}) = 1 - p(k_i) - p(r_i) p(c_i)$$

$$= (2^{-1} + 2^{-i-1}) \cong 1/2$$

$k_i = 0$ type of malfunction

$$p(c_{i+1}) = p(r_i) p(c_i)$$

$$p(\Delta c_{i+1}) = p(k_i) = 1/4$$

The Nature of the Propagated Error

In the remaining sections of this chapter a number of different logical malfunctions will be considered. All malfunctions of the adder logic may result in a propagated error. We shall at this time investigate the characteristics of the propagated error.

The mechanism for the propagation of an error is independent of the cause of the error. The type of malfunction does determine whether a zero or a one is propagated. Constraints of the type $r_i = 1$ and $k_i = 1$ may initiate erroneous one elements while the constraints $r_i = 0$ and $k_i = 0$ initiate erroneous zero elements in the carry vector representation. An element c_{i+2} of the carry vector succeeding an erroneous element c_{i+1} is also in error, if $r_{i+1} = 1$ and no carry is initiated by k_{i+1} . If $r_{i+1} = 1$ then k_{i+1} is constrained to zero. Hence, propagation and initiation never occur simultaneously at the same element. The condition for error propagation is simply $r_{i+1} = 1$ and the probability of propagation is one half. The probability of not propagating is also equal to one half. The probability that an error initiated at the i +1th element by a r_i or k_i type malfunction propagates to the j th element of the carry vector and no further is

$$p(i+1 \dots \dots \dots j) = 2^{-j+i} \quad j \geq i+1$$

An error in the j th element of the carry vector caused by a malfunction of the k_i or r_i type implies an error in all carry elements between and including i +1th element and the j th element. Thus

$$p(\Delta^* c_j) = p(\Delta c_{i+1}) 2^{-j+i} \quad j \geq i+1$$

The asterisk is used to indicate that the propagation of the error stopped at the jth element. The probability that the error includes the j element is

$$p(\Delta c_j) = p(\Delta c_{j+1}) 2^{-j+i+1}$$

If only effective malfunctions are considered the expected number of carry elements in error is a function of the mechanism of carry propagation and is independent of the frequency of occurrence of effective errors of each type. Then

$$p(\Delta^* c_j) = 2^{-j+i}$$

Consider an effective malfunction in the ith stage of the carry logic. The carry element c_{i+1} is then in error. The error propagates only if $r_{i+1} = 1$ regardless of whether the erroneous c_{i+1} element has a value of one or zero. The propagation of the error is halted if $r_{i+n} = 0$. This situation is illustrated below. An erroneous carry element is denoted by \bar{c} and an erroneous sum element by \bar{s} .

$$\begin{array}{cccccc} r_{i+5} & 0 & 1 & 1 & 1 & r_i \\ c_{i+5} & \bar{c}_{i+4} & \bar{c}_{i+3} & \bar{c}_{i+2} & \bar{c}_{i+1} & c_i \\ \hline s_{i+5} & \bar{s}_{i+4} & \bar{s}_{i+3} & \bar{s}_{i+2} & \bar{s}_{i+1} & s_i \end{array}$$

The sum elements in question may assume one of two representations where one representation is correct and the other is erroneous. Which is erroneous and which is correct depends on the correct value of the c_{i+1} element. For our purposes the question of which representation is correct is of no importance. The two representations are

... s_{i+5} 0 1 1 1 s_i ... Type A error
... s_{i+5} 1 0 0 0 s_i ... Type B error

If the error doesn't propagate then $r_{i+1} = 0$ and only one element of the sum is changed.

One other situation must be considered. This is the case where the error propagates to the last element of the representation. There exist only certain conditions for which an error may propagate to c_{n+1} , the last element of the sum representation. The element c_{n+1} is usually ignored if negative numbers are represented in two's complement notation but the c_{n+1} digit may not be ignored if a numerical check is used. If the negative numbers are represented in one's complement notation, then c_{n+1} indicates the need of the end around carry associated with the ones complement addition of two negative numbers or with the addition of positive and negative operands which produce a positive sum. The n th element of the representation is the sign element. An overflow may be detected for either system of negative number representation by an examination of the sign digit of the sum. The details of the problem of an overflow representation are not elaborated at this time but are presented in Appendix I. In regard to the addition of two numbers the following obtains for two's complement arithmetic.

addition of two positive operands

normal	$r_n=0$	overflow	$r_n=0$
	$c_n=0$		$c_n=1$
	$c_{n+1}=0$		$c_{n+1}=0$

addition of two negative operands

normal	$r_n=0$	overflow	$r_n=0$
	$c_n=1$		$c_n=0$
	$c_{n+1}=1$		$c_{n+1}=1$

addition of a positive and a negative operand

positive sum	$r_n=1$	overflow cannot occur
	$c_n=1$	
	$c_{n+1}=1$	
negative sum	$r_n=1$	
	$c_n=0$	
	$c_{n+1}=1$	

For one's complement arithmetic the following situations obtain before the end-around carry correction. The addition of two positive operands results in the same value of r_n , c_n and c_{n+1} as specified above for two's complement arithmetic.

addition of two negative operands

normal	$r_n=0$	overflow	$r_n=0$
	$c_n=0$ or 1		$c_n=0$
	$d_{n+1}=1$		$c_{n+1}=1$

addition of a positive and a negative operand

positive sum	$r_n=1$
	$c_n=1$
	$c_{n+1}=1$
negative sum	$r_n=1$
	$c_n=0$
	$c_{n+1}=0$

If an effective malfunction in the carry logic occurs in the n th stage a single carry digit c_{n+1} will be in error. A malfunction which occurs prior to the n th stage cannot propagate an error to the $n+1$ carry element unless $r_n=1$. The only time the element $r_n=1$ is for the addition of a negative and a positive pair of operands. This is true for both one's complement and two's complement addition. For this special case where the error may propagate to the end of the representation the resulting erroneous representation is either a Type A or Type B error. In short, if the c_{n+1} element is considered as part of the sum representation it is impossible to propagate an error consisting of all ones or zeros to the end of the representation. This fact is of little significance for the digital check but is of paramount importance for the numerical check.

The Average Length of Carry Propagation

In order to indicate the general range of effect of a propagated error we shall consider here the average length of the propagated

carries resulting from the addition of two m digit binary numbers. The carry sequence is initiated by a unity element of the k vector and is propagated if the successive ρ vector elements are equal to unity. The average length of carry propagation is defined by the total number of digits involved in the propagation divided by the total number of propagation sequences. Note in particular that a carry initiation not followed by propagation is not counted as either a propagation sequence or as a propagation digit. A carry initiation followed by propagation is counted as a propagating sequence, but the carry initiation digit is not counted as a propagating digit.

The calculation of the average length of carry propagation involves the consideration of various conditional probabilities. The problem is easily described in terms of a flow diagram such as that shown in Figure 13. The flow diagram may be considered as a representation of the process of examination of m augend and addend digit pairs. The examination procedure starts with the lowest order digit and proceeds consecutively toward the higher order digits. State I corresponds to carry initiation. State P corresponds to carry propagation. State O corresponds to digit pairs which are not a part of either the carry initiation or propagation process. The square boxes on the flow diagram which have been labeled A and B are counters. The counter A indicates the total number of propagation sequences while the counter B indicates the total number of digit places involved

in the propagation minus the total number of propagating sequences. Thus the ratio $\frac{A+B}{A}$ is the average length of carry propagation.

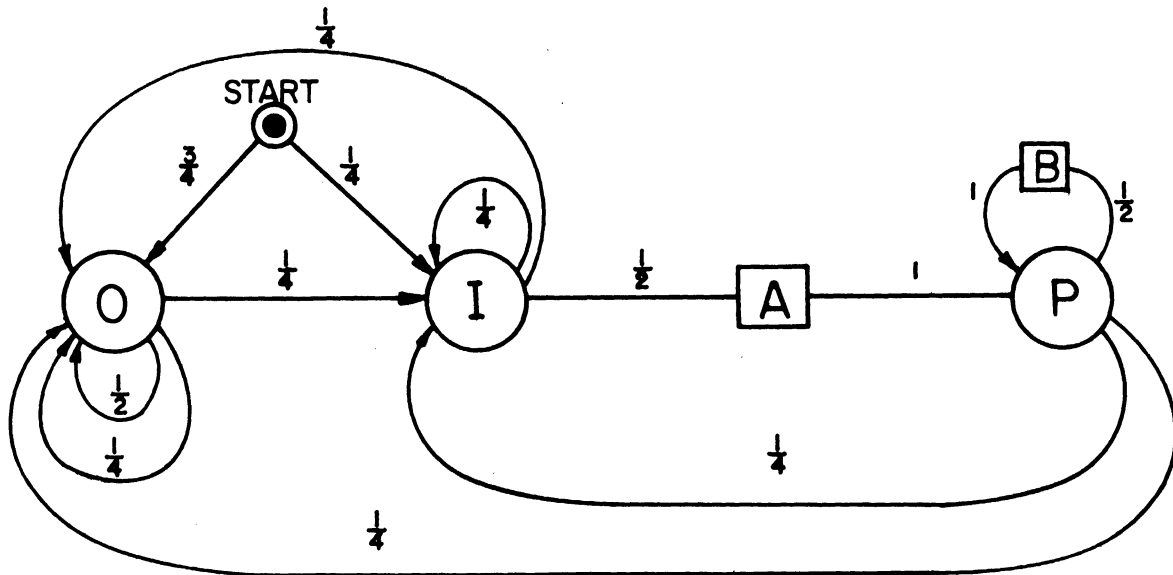


Figure 13. Flow Diagram for the Average Length of Carry Propagation.

The calculation of the average length of carry from the flow diagram is complicated by the fact that the sequence of digit pairs under consideration is finite rather than infinite. It is readily seen for an infinite sequence that the expected value of B for each unit of A is one half. Thus for infinite sequences $\frac{A+B}{A} = \frac{1+0.5}{1} = 1.5$. One would expect the truncation of the sequence to have very little effect on the value of the average length of carry propagation. This is indeed the case. All possible propagating sequences for m digit pairs, for m

equal to 3 and m equal to 4 were examined and the average propagation length was computed. For m equal to 3 the average propagation length is 1.33. For m equal to 4 the average propagation length is 1.43. For computer applications the range of m is typically 30 to 50 digits. The effects of truncation are certainly negligible and the average length of carry propagation is very nearly 1.5 digits.

The Error Structure of the k , ρ and V
Generating Logic for Standard Addition.

In the previous sections the effect of malfunctions in the carry logic has been considered in detail. The analysis is now extended to consider the effects of errors in the logic which generate the k_i and r_i or v_i elements. The input to the generating logic is, of course, the addend and augend elements. The generating logic is implemented with a basic half adder circuit. Each stage of the generating logic is independent of every other stage and the dependence of the sum elements is a function of only the carry logic. However, the various outputs of one stage of the generating logic are not necessarily independent. The generating logic may produce r_i , v_i and k_i and the exact dependence of the ρ , k and V elements is a function of the type of half adder circuit employed.

For the moment, the possible dependence of the outputs of the generating logic is ignored and the effect of single ρ or k malfunctions on the sum representation is considered. An error in a particular k_i element can disturb only the carry vector. Therefore,

the sum error structure resulting from a faulty k_i element is identical to the sum error structure previously associated with a malfunction in the carry logic of the k_i type. If the carry logic is a function of ρ and k then an error in a r_i element can yield an erroneous ρ vector and may also cause an error in the carry vector. An effective r_i element error results in at least one erroneous sum element s_i and the error may propagate to element $i+n$ in precisely the same manner as the error due to a r_i type of carry logic malfunction. The sum element s_i is the vector sum of r_i and c_i while c_i is a function of r_{i-1} , k_{i-1} and c_{i-1} . If $c_i = 0$ the error in r_i affects only s_i . If $c_i = 1$ the error affects s_i and also propagates. The propagation of an erroneous one requires $r_i = 0 \leftarrow r_i = 1$. The erroneous sum element s_i is then zero and if error propagation occurs it is of Type B. The propagation of an erroneous zero requires $r_i = 1 \leftarrow r_i = 0$ and the erroneous sum element s_i is a one. The propagated error is of Type A. The probability that an $r_i = 0$ element type of malfunction is effective is one half. The probability that the $r_i = 0$ malfunction changes s_{i+1} is given by the probability that $c_i = 1$, which is approximately equal to one half.

$$p(\Delta s_i) = p(r_i) = 1/2$$

$$p(\Delta^* s_i) = p(r_i) p(\bar{c}_i) \cong 1/4$$

$$p(\Delta s_{i+1}) = p(r_i) p(c_i) \cong 1/4$$

$$p(\Delta^* s_{i+1}) = p(r_i) p(c_i) p(\bar{r}_{i+1}) \cong 1/8$$

The following probabilities are obtained for an element malfunction of the type $r_i = 1$.

$$\begin{aligned}
 p(\Delta s_i) &= p(\bar{r}_i) = 1/2 \\
 p(\Delta^* s_i) &= p(\bar{r}_i) p(\bar{c}_i) + p(c_i) p(k_i) \\
 &= 1/2 \left[1/2 + 1/2 (1/4) \right] \cong 5/16 \\
 p(\Delta s_{i+1}) &= p(\bar{r}_i) p(c_i) p(\bar{k}_i) \cong 3/16 \\
 p(\Delta^* s_{i+1}) &= p(\Delta s_{i+1}) p(\bar{r}_{i+1}) \cong 3/32
 \end{aligned}$$

There exist half adder circuits which yield independent v and k elements. However, typical of the half adder circuits used to generate the r_i , v_i and k_i element is the circuit shown in Figure 6. For this circuit, errors in the v_i and k_i elements are independent of the r_i element but the r_i element is dependent on both v_i and k_i . If v_i is erroneous then r_i is erroneous unless $k_i = 1$. The probability that $k_i = 1$ is $1/4$ so $3/4$ of the time an error in v_i will also give rise to a r_i element error. Similarly, if k_i is in error r_i will be in error unless $v_i = 0$. $p(\bar{v}_i) = 1/4$. The error in k_i affects the r_i element three fourths of the time. The constraining relationships associated with the particular half adder under discussion are described by the following relationships.

- (1) $k_i = 1 \supset r_i = 0$ and $c_{i+1} = 1$
- (2) $k_i = 0 \supset r_i = v_i$
- (3) $v_i = 1 \supset r_i = \bar{k}_i$
- (4) $v_i = 0 \supset r_i = 0$

We consider first an adder which employs only ρ and k elements. The form of the sum errors resulting from an element error $k_i = 1$ which constrains r_i to a value of zero is now determined. An error occurs if the correct r_i element has a value of one, $p(r_i) = 1/2$. If $c_i = 1$, $p(c_i) \cong 1/2$, the result is an erroneous sum representation in which only one element s_i is in error. For this situation the $k_i = 1$ constraint initiates the proper carry element at c_{i+1} and hence corrects the error in the carry vector due to $r_i = 0$. If $c_i = 0$, $p(\bar{c}) = 1/2$, then an error $s_i = 0$ occurs and a faulty carry sequence is initiated at c_{i+1} . Error propagation, if present, is Type B. If the correct value of r_i is zero the effect of a type $k_i = 1$ malfunction is the same as that associated with a $k_i = 1$ malfunction for the carry. The following probabilities are associated with an element generating malfunction of the type $k_i = 1$ which constrains r_i to a value of zero.

$$\begin{aligned} p(\Delta s_i) &= p(r_i) = 1/2 \\ p(\Delta^* s_i) &= p(r_i) p(c_i) \cong 1/4 \\ p(\Delta s_{i+1}) &= 1 - p(c_{i+1}) \cong 1/2 \\ p(\Delta^* s_{i+1}) &\cong 1/4 \end{aligned}$$

The malfunction of type $k_i = 0 \Rightarrow r_i = v_i$ will result in an error only when the correct value for k_i is unity. r_i will then have an erroneous value of unity since $k_i \Rightarrow v_i$. The carry c_{i+1} which is required will not be initiated unless $c_i = 1$. So if $c_i = 1$ only one sum element, s_i , is in error. If $c_i = 0$ then s_i is erroneous with a value of one and the required carry is not initiated at c_{i+1} . Error propagation if present, is Type A.

The error probabilities associated with the $k_i = 0 \supset r_i = v_i$ type of constraint are

$$\begin{aligned} p(\Delta s_i) &= p(k_i) = 1/4 \\ p(\Delta^* s_i) &= p(k_i) p(c_i) = 1/8 \\ p(\Delta s_{i+1}) &= p(k_i) p(\bar{c}_i) = 1/8 \\ p(\Delta^* s_{i+1}) &\cong 1/16 \end{aligned}$$

The form of the errors resulting from a malfunction of the type $v_i = 1 \supset r_i = \bar{k}_i$ may be described in terms of a single $r_i = 1$ element error though the probabilities associated with the error are changed. The constraint results in an error whenever the correct value of both r_i and k_i is zero.

$$\begin{aligned} p(\Delta s_i) &= p(\bar{r}_i) p(\bar{k}_i) = 1/4 \\ p(\Delta^* s_i) &= p(\bar{r}_i) p_{\bar{r}}(\bar{k}_i) p(\bar{c}_i) = 1/8 \\ p(\Delta s_{i+1}) &\cong 1/8 \\ p(\Delta^* s_{i+1}) &\cong 1/16 \end{aligned}$$

The errors due to the constraint $v_i = 0 \supset r_i = 0$ are identical with the errors resulting from a generating element malfunction of the type $r_i = 0$.

The preceding discussion of constraints has been in terms of an adder logic employing only k and ρ elements. However, if the four constraining relationships are considered in terms of an adder employing k , ρ and V elements the same error structure will be obtained.

One aspect of the structure of the errors associated with the generating logic must yet be investigated. This is the error structure due to malfunctions in the input logic of the generating circuits. The input logic is identified in Figure 10 as gates g_1 and g_2 . The input logic is symmetrical with respect to the input addend and augend elements a_i and b_i . It is therefore sufficient to consider the results of constraints applied to only one of the variables. The results obtained apply equally well to the other variable. The input logic in terms of the addend and augend elements is described by the following equations:

$$(a_i \vee b_i) (\overline{a_i b_i}) = r_i$$
$$a_i b_i = k_i$$

If a_i is constrained to one then

$$r_i = \overline{b_i}$$
$$k_i = b_i$$

and if a_i is constrained to zero

$$r_i = b_i$$
$$k_i = 0$$

The malfunction of type $a_i = 1$ is effective only when the correct value of the element a_i is zero. Two different types of error result. The type of error is determined by the value of b_i . If $b_i = 0$ then the correct value of k_i is obtained but the result $r_i = 1$ is in error. The resulting error in the sum representation consists

of only one altered element if $c_i = 0$. If $c_i = 1$ the error may propagate. If propagation occurs the error is Type B. The probabilities of the error process are

$$\begin{aligned} p(\Delta s_i) &= p(\bar{a}_i) p(\bar{b}_i) = 1/4 \\ p(\Delta^* s_i) &= p(\Delta s_i) p(\bar{c}_i) \cong 1/8 \\ p(\Delta s_{i+1}) &= p(\Delta s_i) p(c_i) \cong 1/8 \\ p(\Delta^* s_{i+1}) &= p(\Delta s_{i+1}) p(\bar{r}_{i+1}) \cong 1/16 \end{aligned}$$

If $b_i = 1$ then both k_i and r_i assume erroneous values, $k_i = 1$ and $r_i = 0$. When $c_i = 1$ only one element s_i is changed. When $c_i = 0$ the malfunction may propagate an error of Type B. The probabilities associated with the error process are

$$\begin{aligned} p(\Delta s_i) &= p(\bar{a}_i) p(b_i) = 1/4 \\ p(\Delta^* s_i) &= p(\Delta s_i) p(c_i) \cong 1/8 \\ p(\Delta s_{i+1}) &= p(\Delta s_i) p(\bar{c}_i) \cong 1/8 \\ p(\Delta^* s_{i+1}) &= p(\Delta s_{i+1}) p(\bar{r}_{i+1}) \cong 1/16 \end{aligned}$$

The malfunction of the type $a_i = 0$ is effective only when the correct value of a_i is one. If at the time of the constraint the value of b_i is zero the resulting error is the generating element type $r_i = 0$. If $b_i = 1$ both the k_i and the r_i outputs will be in error. The erroneous outputs are $k_i = 0$ and $r_i = 1$. The resulting error may be described completely in terms of the constraint $k_i = 0 \supset r_i = 1$ which has already been discussed.

The Error Structure of the rc Sum Logic

In reference to Figure 10 the rc sum generating logic consists of gates g_6 , g_7 and g_8 . The error structure of the rc sum logic is perfectly straightforward if the carry logic and the rc sum logic are separated as shown in Figure 10. For this configuration a malfunction in the rc sum logic may cause at most one error in the sum representation. The propagation of errors is determined only by the carry logic and this precedes the rc sum logic.

In the usual logic of the binary adder the rc sum logic and the carry logic are not independent. In Figure 10 both gates g_5 and g_6 generate the conjunctive function needed for the carry logic. Thus g_5 may be eliminated and g_6 is then part of both the carry logic and the rc sum logic. Malfunctions in g_6 may affect both the carry vector and the rc sum process. The effect of a malfunction of the gate g_6 on the carry vector is completely described in terms of the results of carry logic malfunctions. We shall now consider the possible carry vector errors due to malfunctions of the components of gate g_6 . The carry vector error and the rc sum process error combined determine the sum error.

The diodes associated with gate g_6 are diodes d_1 and d_2 shown in Figure 12. In a previous section the following correspondences were obtained between diode malfunctions and representative constraints of the carry logic inputs

$$d_{1s} \dots r_i = 1$$

$$d_{2s} \dots c_i = 1$$

$$d_{10} \dots c_i = 1$$

$$d_{20} \dots r_i = 0$$

The malfunction represented by $r_i = 1$ applied only to gate g_6 is effective only if $r_i = 0$ and $c_i = 1$. The resulting sum error is Type B. If $c_i = 0$ no error is produced.

$$p(\Delta s_i) = p(\bar{r}_i) p(c_i) \cong 1/4$$

$$p(\Delta^* s_i) = 0$$

$$p(\Delta s_{i+1}) = p(\Delta s_i) \cong 1/4$$

$$p(\Delta^* s_{i+1}) = p(\Delta s_i) \cong 1/8$$

Malfunctions corresponding to the constraint $c_i = 1$ result in one erroneous sum element s_i if $r_i = c_i = 0$. If $c_i = 0$ and $r_i = 1$ a Type B error is produced by the constraint $c_i = 1$.

$$p(\Delta s_i) = p(r_i) p(\bar{c}_i) \cong 1/4$$

$$p(\Delta^* s_i) = 0$$

$$p(\Delta s_{i+1}) = p(\Delta s_i) \cong 1/4$$

$$p(\Delta^* s_{i+1}) = p(\Delta s_i) p(\bar{r}_{i+1}) \cong 1/8$$

The malfunction represented by the constraint $r_i = 0$ produces an error only when $r_i = 1$ and $c_i = 1$. The resulting propagated error is Type A.

The Effectiveness of Parity Checking

The effectiveness of the digital and numerical parity checks is found by considering the probability that a logical malfunction will cause an undetectable change in the sum representation. The effectiveness of a checking procedure can best be evaluated in terms of effective malfunctions. In this case only the form and the propagation of errors are considered and the rate of initiation of errors is ignored.

A summary of the characteristics of the logical malfunctions associated with the logic of a binary adder is given in Table III. The table was compiled from the information derived in the previous sections. The information needed to study the effectiveness of the various parity checks is the normalized error distribution. The error distribution Δ^*s_{i+n} is the probability that the sum elements s_i to s_{i+n} have been modified due to some effective error. For our purposes the important aspect of the error distribution is the ratio of the various Δ^*s_{i+n} values. It is convenient to normalize the error distribution so that $\sum_{n=0}^t \Delta^*s_{i+n} = 1$. Then Δ^*s_i is the probability that the malfunction produces an error in only one element. Δ^*s_{i+1} is the probability that the resulting error disturbs two and only two successive elements of the sum. Δ^*s_{i+2} corresponds to only three successive errors, etc. Four different error distributions have been found. The different distributions shown in Table IV are designated E.D. 1, E.D. 2, E.D. 3 and E.D. 4.

In the calculations which follow the error distribution series are considered infinite in length. The approximation is valid since the

TABLE III

SUMMARY OF ERRORS

Constraint	Other Conditions	Error Type	Error Distribution
GEM $v_1 = 0$ or CLM $r_1 = 0$		A	1
GEM $v_1 = 1$ or CLM $r_1 = 1$		B	1
GEM or CLM $k_1 = 0$		A	1
GEM or CLM $k_1 = 1$		B	1
GEM $r_1 = 0$ or	$c_1 = 1, k_1 = 0$	A and $\bar{s}_1 = 1$	1
GEM $v_1 = 0 \Rightarrow r_1 = 0$	$c_1 = 0$	$\bar{s}_1 = 0$	—
GEM $r_1 = 1$	$c_1 = 1, k_1 = 0$	B and $\bar{s}_1 = 0$	2
	$c_1 = 1, k_1 = 1$	$\bar{s}_1 = 0$	—
	$c_1 = 0$	$\bar{s}_1 = 1$	—
GEM $k_1 = 1 \Rightarrow r_1 = 0$	$r_1 = c_1 = 1$	$\bar{s}_1 = 1$	
	$r_1 = 1, c_1 = 0$	B and $\bar{s}_1 = 0$	4
	$r_1 = 0$	B	1
GEM $k_1 = 0 \Rightarrow r_1 = v_1$	$k_1 = c_1 = 1$	$\bar{s}_1 = 0$	
	$k_1 = 1, c_1 = 0$	A and $\bar{s}_1 = 1$	1
GEM $v_1 = 1 \Rightarrow r_1 = \bar{k}_1$	$k_1 = c_1 = 0$	$\bar{s}_1 = 1$	
	$k_1 = 0, c_1 = 1$	B and $\bar{s}_1 = 0$	1
GEM $a_1 = 1$	$c_1 = 0$	$\bar{s}_1 = 1$	
	$c_1 = 1$	B and $\bar{s}_1 = 0$	1
GEM $a_1 = 0$	$b_1 = 0$	same as GEM $r_1 = 0$ with $k_1 = 0$	1
	$b_1 = 1$	same as GEM $k_1 = 0 \Rightarrow r_1 = v$	1
CGM $r_1 = 1$	$c_1 = 1$	B and $\bar{s}_1 = 0$	3
CGM $c_1 = 1$	$r_1 = 0$	$\bar{s}_1 = 1$	
	$r_1 = 1$	B and $\bar{s}_1 = 0$	3
CGM $r_1 = 0$	$c_1 = 1$	A and $\bar{s}_1 = 1$	3

GEM generating element malfunction

CLM carry logic malfunction

CGM common gate malfunction

TABLE IV

ERROR TYPE				
	s_{i+n}	$s_{i+n-1} \dots s_{i+2}$	s_{i+1}	- sum element
A	0	1.....1	1	
B	1	0.....0	0	

TABLE V

ERROR DISTRIBUTION (NORMALIZED)				
	$\Delta * s_i^n$	$\Delta * s_{i+1}^n$	$\Delta * s_{i+2}^n$	$\Delta * s_{i+3}^n$
1	1/2	1/4	1/8	1/16
2	5/8	3/16	3/32	3/64
3	0	1/2	1/4	1/8
4	1/3	1/3	1/6	1/12

number of elements usually involved in computer arithmetic operations is thirty to forty bits. The fact that the series is not truncated will affect most severely the error distributions resulting from the initiation of errors in the first few high order elements. The number of elements so affected is small compared to the total number of elements involved.

The effectiveness of a simple digital parity check employing one check digit is dependent on the probability that the error in the sum representation involves an odd or an even number of elements. If the number of erroneous sum digits is even the check fails. The check is successful if the number of erroneous sum digits is odd. The probability that the check succeeds $p(S)$ or fails $p(F)$ for each of the four error distributions is

$$p_1(S) = 1/2 + 1/8 + 1/32 + \dots = 2/3$$

$$p_1(F) = 1/4 + 1/16 + 1/64 + \dots = 1/3$$

$$p_2(S) = 5/8 + 3/32 + 3/128 + \dots = 3/4$$

$$p_2(F) = 3/16 + 3/64 + 3/256 + \dots = 1/4$$

$$p_3(S) = 1/4 + 1/16 + 1/64 + \dots = 1/3$$

$$p_3(F) = 1/2 + 1/8 + 1/32 + \dots = 2/3$$

$$p_4(S) = 1/3 + 1/6 + 1/24 + \dots = 5/9$$

$$p_4(F) = 1/3 + 1/12 + 1/48 + \dots = 4/9$$

The effectiveness of a digital parity check consisting of two digits is now determined. The check is constructed so that elements in odd positions are incorporated in one check and the elements in the even positions in the other check. The check is judged successful if as a result of an effective error either check fails.

$$p_1(S) = 1/2 + 1/4 + 1/8 + 1/32 + 1/64 + \dots = 14/15 \cong .92$$

$$p_1(F) = 1/16 + 1/256 + 1/4096 + \dots = 1/15$$

$$p_2(S) = 5/8 + 3/16 + 3/32 + 3/128 + 3/256 + 3/512 + \dots = 19/20 = .95$$

$$p_2(F) = 3/64 + 3/1024 + \dots = 1/20$$

$$p_3(S) = 0 + 1/2 + 1/4 + 1/16 + 1/32 + 1/64 + \dots = 13/15 \cong .87$$

$$p_3(F) = 1/8 + 1/128 + \dots = 2/15$$

$$p_4(S) = 1/3 + 1/3 + 1/6 + \dots = 41/45 \cong .91$$

$$p_4(F) = 1/12 + 1/192 + \dots = 4/45$$

The effectiveness of any simple digital parity check constructed as above and consisting of n check digits may be found by evaluating a recurring binary fraction consisting of repeated groups of $2^{2n}-1$ one digits and one zero digit. The magnitude of the non terminated fraction is $p_1(S)$. $p_1(S)$ for a representation of the type

$$.1111 \dots 10 \quad 1111 \dots 10 \quad 1111 \dots 10 \dots$$

is

$$p_1(S) = \frac{2^{2n}-2}{2^{2n}-1} \text{ for } n \text{ check digits}$$

$p_2(S)$, $p_3(S)$ and $p_4(S)$ are obtainable if $p_1(S)$ is known.

$$p_2(S) = 1/4 + 3/4 p_1(S)$$

$$p_3(S) = 2 p_1(S) - 1$$

$$p_4(S) = 4/3 p_1(S) - 1/4$$

Numerical checking requires the residue of the sum representation with respect to the check base. Check bases of magnitude $2^n - 1$, $n > 0$, are particularly appropriate for binary representations.

The process of numerical checking requires the assignment of a check weight associated with each element of a representation. The check weight is the least positive residue of the normal element weight with respect to the check base. The following examples illustrate the weighting scheme for several possible check bases.

$$. . . . 2 1 2 1 2 1 \quad m = 3$$

$$. . . . 4 2 1 4 2 1 \quad m = 7$$

$$. . . . 2 1 8 4 2 1 \quad m = 15$$

It is observed that an error in a single digit is detectable by any numerical check. Furthermore, it has been shown that the errors due to logical malfunctions which change more than one sum element are restricted in form. The permitted error forms consist of all zeros except the last erroneous sum digit which is a one, Type B, or an erroneous representation consisting of all ones except the last digit which is a zero, Type A. The permitted error forms are always detectable by a numerical check with a check base of $2^n - 1$.

Consider the following example which employs a modulo 3 numerical check.

$$S = 11\ 00\ 01 - 10\ 11\ 10 = x - y = x + y'$$

0 11 00 01 x	Ⓚ	parity
1 01 00 10 y'	Ⓚ	digit
1 10 00 11 p		
<u>11 10 00 0 c</u>		
10 00 00 11 S	Ⓜ	

Consider now an error in the carry logic such that the error propagates to c_{n+1} , for example, an effective malfunction which changes c_n .

It follows then that c_{n+1} is also changed since in this example $r_n = 1$. The erroneous carry representation is 00 10 00 0 and the erroneous sum is 01 00 00 11. The modulo 3 parity check bit associated with the erroneous sum is clearly one and the error is detected.

As a result of the various constraints which exist in the process of binary addition the simple modulo 3 numerical parity check is 100% effective if the c_{n+1} element of the sum is considered. The check requires only two bits. For operations consisting of repetitive addition such as multiplication or subtraction, the numerical parity check is 100% effective only if applied at each addition step. It is possible to obtain an accumulated error due to several addition operations which is undetectable by the numerical check.

CHAPTER V

THE RESIDUE CODE

An Extension of Numerical Checking

If a sufficient number of numerical parity checks are employed in a number system the parity check representation may ultimately contain as much information as that possessed by the number system being checked. If this is the case it then is possible to replace the original number representation by the check representation. The extended check representation is termed a residue number system. We shall now proceed to study the construction and properties of the residue number system.

The first requirement for a residue number system is that the bases of the different elements of the representation must be relatively prime. If the residue number system is considered as an evolution of the numerical checking procedures the above statement may be reinterpreted as a requirement that the different check bases be relatively prime to each other. If a pair of bases are not relatively prime the effect is the introduction of redundancy. The following example will illustrate this fact. Contrast the combination of a modulo 2 and a modulo 6 check against the combination of a modulo 3 and a modulo 4 check. In the first case the two check bases are not relatively prime while in the second case the check bases are relatively prime. The combination of a modulo 2 and a modulo 6 check is unique for only 6 states while the combination of the modulo 3 and modulo 4 check provides a unique residue representation for 12 states. This is further clarified by Table VI.

TABLE VI

REDUNDANCY OF A NON-RELATIVELY PRIMED BASE REPRESENTATION

Number	Least Positive Residue			
	Mod 2	Mod 6	Mod 3	Mod 4
0	0	0	0	0
1	1	1	1	1
2	0	2	2	2
3	1	3	0	3
4	0	4	1	0
5	1	5	2	1
6	0	0	0	2
7	1	1	1	3
8	0	2	2	0
9	1	3	0	1
10	0	4	1	2
11	1	5	2	3
12	0	0	0	0
13	1	1	1	1
14	0	2	2	2

If the bases of the residue number system are relatively prime then the number of states uniquely represented without redundancy is equal to the product of the magnitude of the bases. The most efficient residue number system representation is obtained by a consecutive sequence of prime numbers starting with the integer two. Since only a relatively prime relationship is desired between the bases it would be possible to employ one base equal to a non-prime integer as long as this base is relatively prime to the other bases. However, there are certain advantages which are associated with the prime number base but are not associated with the non-prime base. In particular, the residues with respect to a non-prime base form a ring, while the residues with respect to a prime base form a field. The characteristics of rings and fields which are important to the present discussion are the following: Two operations, multiplication and addition are defined for both rings and fields. Also an additive inverse exists for both rings and fields. This means that subtraction is possible as well as the defined operations of addition and multiplication. The ring does not require a multiplicative inverse. The field must contain a unique multiplicative inverse for every element. Furthermore, the field admits the solution to linear algebraic equations.

The residue number system differs from the standard consistently weighted number systems in many ways. The most important difference is that the residue number system, being based upon a number of

fields or rings, has two defined operations. It will be recalled that arithmetic operations in the consistently based number system usually employed in the digital computer are based upon the definition of one operation, that operation being addition. The other difference is that the residue representation possesses digital independence of the elements comprising the representation. There exists no carry to constrain the elements. The absence of a carry plus the fact that multiplication is defined gives rise to the expectation that the residue number system should be amenable to simple error detecting and correcting schemes. It should also be possible to execute rapid addition and multiplication.

An example of a residue number system is presented in Table VII. The number system shown in Table VII uses the prime bases 2,3,5 and 7. The number system therefore contains 210 states. The 210 states may correspond to the positive integers 0 to 209. Table VII shows the residue number representation corresponding to the positive integers 0 to 29. Additional integers of the number system may be found by

TABLE VII

NATURAL NUMBERS AND CORRESPONDING RESIDUE NUMBERS

N.N.	2357	N.N.	2357	N.N.	2357
0	0000	10	0103	20	0206
1	1111	11	1214	21	1010
2	0222	12	0025	22	0121
3	1033	13	1136	23	1232
4	0144	14	0240	24	0043
5	1205	15	1001	25	1104
6	0016	16	0112	26	0215
7	1120	17	1223	27	1026
8	0231	18	0034	28	0130
9	1042	19	1145	29	1241

congruence operations. Let a,b,c and d be the digits associated with the bases 2,3,5 and 7 respectively. The following congruences define a,b,c and d for the residue representation of the number N:

$$N \equiv a \pmod{2}$$

$$N \equiv b \pmod{3}$$

$$N \equiv c \pmod{5}$$

$$N \equiv d \pmod{7}$$

The residue number system is readily extended to include more states. For example, if a base 11 is added to the representation it is then possible to represent 2310 states. Table VIII shows the product and sum of the first ten consecutive primes greater than or equal to 2. The product of the primes indicates the number of states of the number system while the sum of the primes is a measure of the size of the representation in terms of digits. Table VIII also includes the number of bits required to represent each prime base in the binary number system.

TABLE VIII

NUMBER OF STATES AND DIGITS ASSOCIATED
WITH A RESIDUE REPRESENTATION

i	p_i	$\sum_{i=1}^n p_i$	$\prod_{i=1}^n p_i$	p_i bits	$\sum p_i$ bits
1	2	2	2	1	1
2	3	5	6	2	3
3	5	10	30	3	6
4	7	17	210	3	9
5	11	28	2,310	4	13
6	13	41	30,030	4	17
7	17	58	510,510	5	22
8	19	77	9,699,699	5	27
9	23	103	223,092,670	5	32

Residue Addition and Multiplication

The residue number representation consists of several elements and is assumed to be in one to one correspondence with some positive integers of the real number system. The elements of the residue representation are the least positive residues of these real positive integers with respect to the different moduli which form the bases of the residue representation. It follows as a direct consequence of the structure of the residue number system and the properties of linear congruences that the operations of addition and multiplication are valid in the residue number system subject to one proviso. The proviso is that the residue system must possess a number of states sufficient to represent the generated sum or product. If the residue number system does not have a sufficient number of states to represent the sums and the products generated by a particular finite set of real integers then the residue system will overflow and more than one sum or product of the real number system may correspond to one residue representation. For a residue number with a sufficient number of states an isomorphic relation exists with respect to the operations of addition and multiplication in the residue system and a finite system of real positive integers.

Each element of the residue number system is obtained with respect to a different base or modulus. It follows therefore, that the rules of arithmetic associated with each element will be different. For example, the addition and multiplication of the elements associated with the moduli 2 and 3 follow rules specified in Table IX.

TABLE IX

MOD 2 AND MOD 3 SUMS AND PRODUCTS

\oplus 0 1		\oplus 0 1 2		\oplus 0 1 2
0 0 1		0 0 1 2		0 0 0 0
1 1 0		1 1 2 0		1 0 1 2
sum Mod 2		2 2 0 1		2 0 2 1
\oplus 0 1		sum Mod 3		product Mod 3
0 0 0				
1 0 1				
product Mod 2				

No carry tables are necessary since the residue number system does not have a carry mechanism. Addition of two residue representations is effected by the modulo addition of corresponding elements of the two representations. Corresponding elements must have the same base or modulus. Modulo addition of elements which have different bases is not defined. Multiplication in the residue system is effected by obtaining the modulo product of corresponding elements. The operations of addition and multiplication of two residue numbers are indicated by the following notation:

$$S = A \oplus B$$

$$p = A \odot B$$

Consider a residue number representation with base 2,3,5 and 7. We assume an isomorphic relation between the residue number system

and the real positive numbers 0 to 209. An isomorphic relation then exists for the operations of multiplication and addition only if the product or sum is less than 209. The following examples employing residue numbers illustrate the addition and multiplication operations and the presence of an isomorphism or the lack of isomorphism in the case of overflow. Residue numbers will be distinguished by the use of parentheses.

$$29 + 27 = S = 56$$

$$29 \leftrightarrow (1\ 2\ 4\ 1)$$

$$27 \leftrightarrow (1\ 0\ 2\ 6)$$

$$56 \leftrightarrow (0\ 2\ 1\ 0)$$

$$\oplus \begin{array}{r} (1\ 2\ 4\ 1) \\ (1\ 0\ 2\ 6) \\ \hline (0\ 2\ 1\ 0) \end{array}$$

The following operations are considered in performing the addition of the two residue representations.

$$1 + 1 \equiv 0 \text{ Mod } 2$$

$$2 + 0 \equiv 2 \text{ Mod } 3$$

$$4 + 2 \equiv 1 \text{ Mod } 5$$

$$1 + 6 \equiv 0 \text{ Mod } 7$$

Consider the addition of two numbers which produce a sum greater than 209.

$$S = 100 + 200$$

$$\oplus \begin{array}{r} (0\ 1\ 0\ 2) \\ (0\ 2\ 0\ 4) \\ \hline (0\ 0\ 0\ 6) \end{array}$$

The residue representation (0006) corresponds to the real positive number 90. In this particular example the sum has overflowed the residue representation. The resulting sum is the correct sum modulo 210.

$$300 \equiv 90 \text{ Modulo } 210$$

Finite real number systems and residue number systems have the same overflow characteristics. The sum which remains after the overflow is the correct sum with respect to a modulus numerically equal to the number of states in the finite number system.

The following is presented as an example of the process of residue multiplication.

$$\begin{aligned} p &= 10 \times 17 = 170 \\ 10 &\leftrightarrow (0 \ 1 \ 0 \ 3) \\ 17 &\leftrightarrow (1 \ 2 \ 2 \ 3) \\ 170 &\leftrightarrow (0 \ 2 \ 0 \ 2) \\ & \\ & \begin{array}{r} (0 \ 1 \ 0 \ 3) \\ \oplus \ (1 \ 2 \ 2 \ 3) \\ \hline (0 \ 2 \ 0 \ 2) \end{array} \end{aligned}$$

The process of multiplication involved consideration of the following relations for each element.

$$\begin{aligned} 1 \times 0 &\equiv 0 \text{ Mod } 2 \\ 1 \times 2 &\equiv 2 \text{ Mod } 3 \\ 0 \times 2 &\equiv 0 \text{ Mod } 5 \\ 3 \times 3 &\equiv 2 \text{ Mod } 7 \end{aligned}$$

An overflow resulting from a multiplication is no different than the overflow resulting from an addition. Consider the product obtained from the residue multiplication of the real numbers 10 and 100. The result in the modulo 210 number system corresponds to the real number 160 which is 1000 modulo 210.

Subtraction and the Representation of Negative Numbers

The process of subtraction is obtainable in the residue number system by employing a complement representation consisting of the additive inverses of the positive residue representation. Since the elements of the residue representation are elements of a field, the additive inverse always exists. There is no basic problem associated with the subtraction operation. There is, however, a problem associated with the representation of negative numbers. In particular, some mechanism must be included in the number system which will distinguish positive and negative differences. This problem will be discussed in detail in a later section.

The additive inverse of a residue number is defined by the following:

$$a \oplus a' = 0$$

The formula may be considered to apply to an element of the residue system or equally well to the whole residue representation. Consider the following examples with reference to the modulo 210 residue number system.

a = (1 2 4 1)
then a' = (1 1 1 6)
since

$$\oplus \begin{array}{r} (1\ 2\ 4\ 1) \\ (1\ 1\ 1\ 6) \\ \hline (0\ 0\ 0\ 0) \end{array}$$

The following examples have been chosen to illustrate the subtraction process and to some extent the difficulties associated with the sign of the difference.

$$D = A \ominus B = A \oplus B'$$

We consider first the case where the magnitude of A is greater than B.

$$\text{Let } A = 200 \qquad B = 100$$

In residue representation

then $B' = (0 \ 2 \ 0 \ 5)$

and
$$\oplus \begin{array}{r} (0 \ 2 \ 0 \ 4) \\ \hline (0 \ 2 \ 0 \ 5) \\ (0 \ 1 \ 0 \ 2) \end{array}$$

The residue representation of the difference corresponds to positive 100 in the real number domain. We consider next the case where the magnitude of B is greater than the magnitude of A.

$$A' = (0 \ 1 \ 0 \ 3)$$

then $D = A' \oplus B$

and
$$\oplus \begin{array}{r} (0 \ 1 \ 0 \ 3) \\ \hline (0 \ 1 \ 0 \ 2) \\ (0 \ 2 \ 0 \ 5) \end{array}$$

The difference (0 2 0 5) is the additive inverse of (0 1 0 2). Unless additional information is supplied the correct interpretation of the representation (0 2 0 5) is in doubt. (0 2 0 5) may correspond to either +110 or -100.

The difficulties associated with whether a residue representation corresponds to positive or negative integer can be partially

removed by the division of the residue number range into two parts. This is exactly the scheme that is employed to obtain a machine representation of positive and negative natural numbers. For the system of natural numbers two different machine representations of the negative numbers may be obtained and are commonly designated the radix complement representation of negative numbers and the diminished radix complement representation of negative numbers.

The complement representation for a residue code is defined in terms of the additive inverse. Thus the representation of negative A is A' where $A \oplus A' = 0$, and the range of A is restricted to approximately one half of the total possible range of the residue representation. This can be illustrated by consideration of a specific residue code. The residue representation employing bases of magnitude 2, 3, 5, and 7, is divided into two parts. The residue representations corresponding to the natural numbers 0 to 104 are considered positive. The residue representations corresponding to the natural numbers 105 to 209 are considered negative. The range of this particular number system is from -105 to +104. The arithmetic rules pertaining to sign and overflow conventions for this particular number system are the same rules normally associated with radix complement arithmetic.

The complement representation does eliminate in principle any ambiguity concerning the sign of the result of an arithmetic operation. However, there is a practical difficulty. The determination of the sign of a residue representation requires the determination of the

magnitude of the representation relative to the magnitude of the representation which separates the positive and negative representations. The determination of relative magnitude for a residue representation is discussed in a later section. It will be shown that the determination of relative magnitude is not a simple problem.

Conversion From a Residue Code to a Normal Number Representation

It is frequently desirable to determine the natural number associated with a particular residue representation. The need for this conversion occurs frequently when investigating the properties of the residue system. The residue representation is constructed in such a manner that magnitude is not readily obtainable. The presence of column weights in the normal number representation greatly facilitates the determination of magnitude. It is possible to assign a weight to each digit of the residue representation in such a manner that the modulo m sum of the digit-weight products is the real natural number in consistently weighted representation. m is the product of all the bases employed in the residue representation. The conversion technique is known as "The Chinese Remainder Theorem". The material which follows describes the conversion technique but omits the proof. A simple and straightforward proof is found in Dickson.⁽²⁰⁾ The proof does not refer specifically to residue number systems but rather to a system of linear congruences. If so regarded, a system of congruences defines a component of a residue number system.

Consider a residue number system with bases $m \dots m_t$. The corresponding digits are labeled $a_1 \dots a_t$. The following equations define the conversion process.

$$a_1 A_1 \frac{m}{m_1} + \dots + a_t A_t \frac{m}{m_t} \equiv S \text{ Mod } m$$

where

$$A_i \frac{m}{m_i} \equiv 1 \text{ Mod } m_i$$

and

$$m = \prod_{j=1}^t m_j$$

The conversion formula for a particular residue number system is now obtained.

$$m_1 = 2 \quad m_2 = 3 \quad m_3 = 5 \quad m_4 = 7$$

$$105 A_1 \equiv 1 \text{ Mod } 2 \quad \text{so } A_1 = 1$$

$$70 A_2 \equiv 1 \text{ Mod } 3 \quad \text{so } A_2 = 1$$

$$42 A_3 \equiv 1 \text{ Mod } 5$$

$$2 A_3 \equiv 1 \text{ Mod } 5 \quad \text{so } A_3 = 3$$

$$30 A_4 \equiv 1 \text{ Mod } 7$$

$$2 A_4 \equiv 1 \text{ Mod } 7 \quad \text{so } A_4 = 4$$

$$105 a_1 + 70 a_2 + 126 a_3 + 120 a_4 \equiv S \text{ Mod } 210$$

The conversion formula is now used to determine the natural number corresponding to the residue representation (1 2 0 4).

$$105 (1) + 70 (2) + 126 (0) + 120 (4) = 725$$

$$725 \equiv S \text{ Mod } 210$$

$$S = 95$$

Other conversion techniques exist. In particular it is possible by means of a deductive process to determine the magnitude of a particular residue representation. This requires both a knowledge of the nature of the residue system and the natural number representation associated with at least one residue representation.

Due to the deductive nature of the process it is more suitable for human computation than for machine computation. The process is explained using the residue number of the previous example (1 2 0 4). The knowledge of the residue representation for unity which is (1 1 1 1) is assumed. Consider the effect of changing the second digit from one to two. The change adds the product $m_1 m_3 m_4 = 70$ to the number since 70 is congruent 1 modulo 3. The resulting residue representation (1 2 1 1) corresponds to 71. The effect of changing the third digit is to change the magnitude by some multiple of the product $m_1 m_2 m_4 = 42$. The correct change in magnitude is $42x$ where $42x \equiv 4 \pmod{5}$. So $42x = 84$ and the residue representation 1 2 0 1 corresponds to 155. The fourth digit is modified by the addition of a three. The effect of this change is determined by $30x \equiv 3 \pmod{7}$. The magnitude change is 150. The sum of 150 and 155 modulo 210 yields the correct result 95, in correspondence with (1 2 0 4).

The conversion formulae provide a possible means of achieving conversion from a residue code to an analog representation. The conversion would involve first the transformation of the residue code to a normal weighted representation and then a transformation to analog representation by means of standard techniques. Alternatively the conversion

process need not involve the normal representation if the analog equipment were capable of precise subtraction of multiples of m where m is the product of the bases. A third conversion process would employ counters for both the residue system and the normal number system. Comparison would be employed to determine the accumulation of the desired number of counts. At the time of comparison one set of counters would contain the residue representation and the other the desired normal representation.

Greater Than or Less Than Relations for Residue Codes

Some of the arithmetic operations for the residue code are dependent on the determination of a greater than or less than relationship. A possible technique might involve the conversion techniques described in the previous section. Such a scheme would involve the standard comparison techniques associated with the determination of the relative magnitude of two numbers represented in a weighted representation. The procedure described in this section does not require an explicit conversion from residue code to normal representation. Consider a residue code consisting of t digits. The t digits of the residue code are associated with t congruence relationships as follows:

$$S \equiv a_i \text{ Mod } m_i \quad 1 \leq i \leq t$$

S is the magnitude of the number expressed in normal representation.

It is also possible to express the number S as

$$S = a_1 + A_1 m_1$$

A_1 is the integer part of the quotient of S divided by m_1 . In regard to a greater or less than relationship, the determination of A_1 divides the range of the residue representation into m/m_1 parts. We proceed to calculate A_1 from the set of t equations given above.

Let
$$S = a_t + A_t m_t \quad A_t < \frac{m}{m_t}$$

This equation is then used to replace S in the remaining t-1 equations, yielding t-1 equations of the form

$$A_t m_t \equiv (a_1 + a_t^i) \text{ Mod } m_1 \quad 1 \leq i \leq t-1$$

or
$$A_t \equiv (a_1 + a_t^i) / m_t^i \text{ Mod } m_1$$

$$A_t \equiv d_t^i \text{ Mod } m_1$$

where $/m_t^i$ is the multiplicative inverse of m_t with respect to base m_1 .

The multiplicative inverse is defined as

$$x_t / x_t^i \equiv 1 \text{ Mod } m_1$$

d_t^i is the least positive residue of $(a_1 + a_t^i) / m_t^i$ with respect to base m_1 .

a_t^i is the additive inverse of a_t .

Let A_t be expressed as

$$A_t = d_t^{t-1} + A_{t-1} m_{t-1} \quad A_{t-1} < \frac{m}{m_t m_{t-1}}$$

If this expression is substituted for A_t a set of $t-2$ equations remain.

The equations are of the form

$$A_{t-1} \equiv \left[d_t^i + (d_t^{t-1})^i \right] / m_{t-1} \text{ Mod } m_i \quad 1 \leq i \leq t-2$$

$$A_{t-1} \equiv d_{t-1}^i \text{ Mod } m_i$$

The system of equations shown below is generated by repetition of the above substitution process until no equations remain.

$$S = a_t + A_t m_t$$

$$A_t = d_t^{t-1} + A_{t-1} m_{t-1}$$

$$A_{t-1} = d_{t-1}^{t-2} + A_{t-2} m_{t-2}$$

·
·
·

$$A_3 = d_3^2 + A_2 m_2$$

$$A_2 \equiv d_2^1 \text{ Mod } m_1$$

The equations are combined to yield:

$$\begin{aligned}
 S &= a_t + m_t \left\{ d_t^{t-1} + m_{t-1} \left[d_{t-1}^{t-2} + m_{t-2} (d_{t-2}^{t-3} + \dots \right. \right. \\
 &= a_t + m_t d_t^{t-1} + m_t m_{t-1} d_{t-1}^{t-2} + m_t m_{t-1} m_{t-2} d_{t-2}^{t-3} + \dots + \frac{m}{m_1} d_2^1
 \end{aligned}$$

where $A_t < m_t$

$$d_{t-n}^{t-n-1} < m_{t-n-1}$$

Therefore, S is never equal to or greater than m and d_2^1 divides the range into m_1 parts, d_3^2 divides each of the m_1 parts into m_2 parts, d_4^3 divides each of the m_2 parts into m_3 parts, etc.

The determination of the less than or greater than relationship consists of the successive comparison of the d_{t-n}^{t-n-1} constants corresponding to two residue representations. Let the representations be designated E and F. The first step of the greater than or less than determination is the comparison of $d_2^1(E)$ and $d_2^1(F)$. If the two constants are different the process may be terminated and the larger number is associated with the larger constant. If the constants are equal in value the comparison process must consider the pair of constants $d_3^2(E)$ and $d_3^2(F)$. The process is continued in this manner until a set of non-identical constants are found. If all of the d constants are identical a final comparison is made on the basis of the pair of t th digits of the two residue representations.

The formulae which define the greater than, less than process may be applied recursively to obtain a formula for a greater number of digits. The process has been extended to five variables and the results are shown in Figure 14.

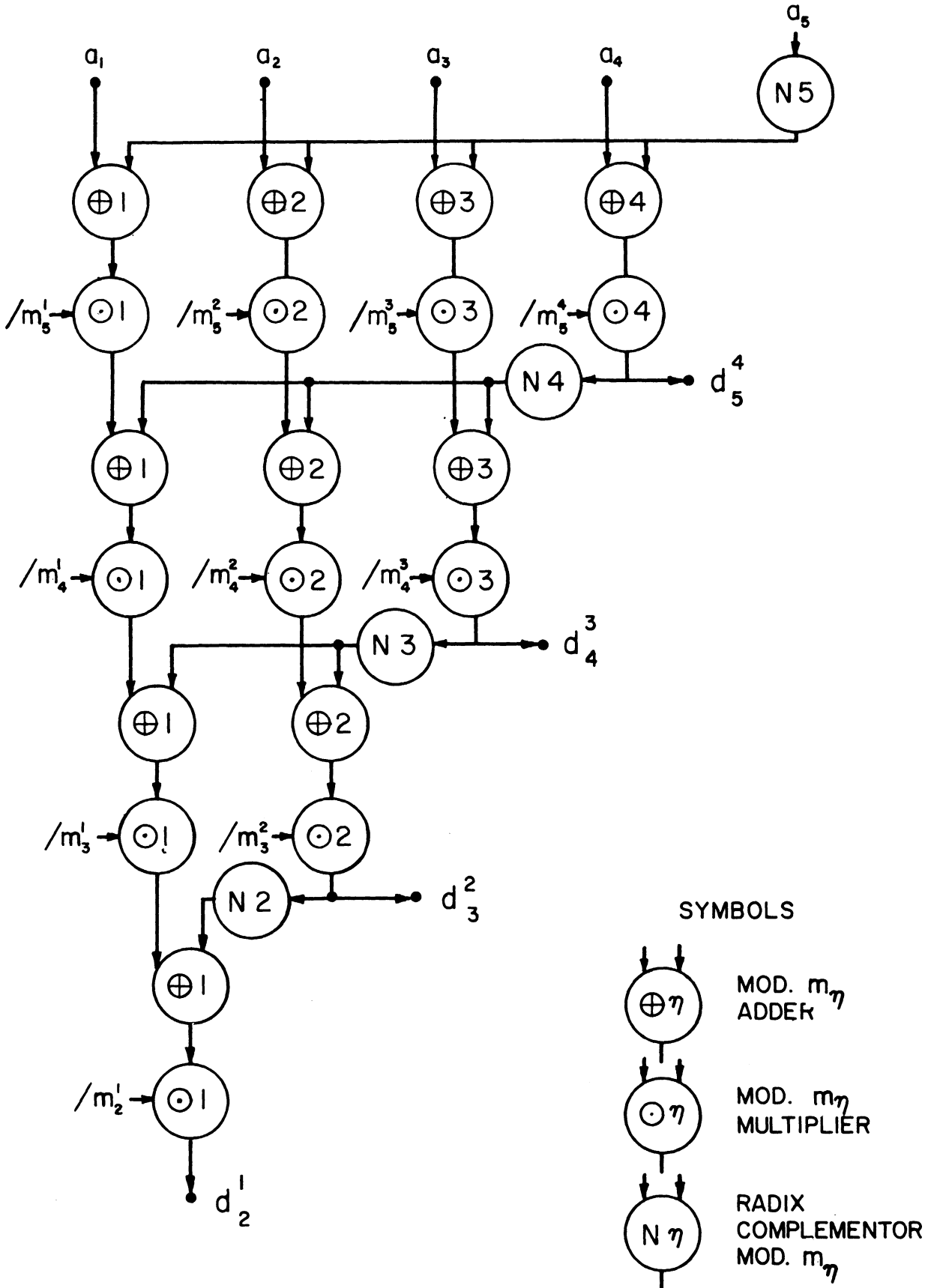


Figure 14. Logic for the Determination of the Greater or Less Than Relationship.

Admittedly the process required to obtain a greater than or less than relationship leaves much to be desired. One presumed advantage of the residue number was the absence of a carry process. The greater or less than process is essentially sequential and is in many ways similar to the carry process.

Redundancy

A residue code contains redundant information if the product of the bases exceeds the number of states required to represent a specific range of numbers. In particular, if sufficient redundancy is introduced the residue code may be made independent of one or more digits. Redundancy is obtained by adding one or more additional digits to the structure of a residue code sufficient to represent the desired number range. Consider the addition of a prime base m_u to a residue code with bases $m_1 \dots m_t$. The resulting code is redundant. Any one digit of the code may be deleted and the code will still contain the required amount of information. Furthermore, any number of digits may be deleted if the product of the remaining bases is greater than the magnitude of the represented number. This means that normally it is possible to delete more than one low order digit but in general only one high order digit may be deleted. Any degree of redundancy is obtainable by the addition of more redundant digits.

Redundancy in the residue code is easily obtained. The real problem concerns the realization of the procedures for error correction and detection. This problem may be considered from the viewpoint of

the Chinese Remainder Theorem. A residue representation of $t+1$ digits has one redundant digit. The Chinese Remainder Theorem may be applied using only t digits of the residue representation. In fact the theorem may be applied once for each of the $t+1$ combinations. Each combination consists of t digits. $t+1$ solutions are obtained and if the redundant residue representation contains one erroneous digit then only one of the $t+1$ solutions is correct. The correct solution is, of course, the solution which does not involve the erroneous digit. Unfortunately, there is no way in which the correct and the incorrect solution may be distinguished within the framework of the residue code with a single redundant digit. We now consider the question of whether two of the incorrect solutions may be identical. Given a residue code with bases $m_1 \dots m_{t+1}$, we assume the j th digit a_j is erroneous by d units. The t erroneous solutions obtained by means of the Chinese Remainder Theorem deviate from the correct solution by Δ where

$$\Delta \equiv d \pmod{m_j}$$

$$\Delta_i = \frac{x_i m}{m_j m_i}$$

Let the correct solution be S and a pair of erroneous solutions S_i and S_k , then

$$S + \frac{x_i m}{m_j m_i} \equiv S_i \pmod{\frac{m}{m_i}}$$

$$S + \frac{x_k m}{m_j m_k} \equiv S_k \pmod{\frac{m}{m_k}}$$

$$S + \frac{x_i m}{m_j m_i} = S_i + a \frac{m}{m_i} \quad S < \frac{m}{m_i}$$

$$S + \frac{x_k m}{m_j m_k} = S_k + b \frac{m}{m_k} \quad S < \frac{m}{m_k}$$

$$a < 2; \quad b < 2$$

$$a = 0, 1; \quad b = 0, 1$$

Assume $S_i = S_k$ then

$$\frac{x_i}{m_i} - \frac{x_k}{m_k} = \left[\frac{a}{m_i} - \frac{b}{m_k} \right] m_j$$

We consider the four combinations of values of a and b

$$(1) \quad \frac{x_i}{m_i} - \frac{x_k}{m_k} = 0 \quad \text{if} \quad \begin{array}{l} x_i = m_i f \\ x_k = m_k f \end{array}$$

f is an integer

$$(2) \quad \neq \frac{m_j}{m_i}$$

$$(3) \quad \neq -\frac{m_j}{m_k}$$

$$(4) \quad \frac{x_i}{m_i} - \frac{x_k}{m_k} = \frac{m_j}{m_i} - \frac{m_j}{m_k}$$

$$x_i m_k - x_k m_i = m_j m_k - m_j m_i$$

$$m_k (x_i - m_j) = m_i (x_k - m_j)$$

The equality is satisfied if

$$x_i = m_j + m_i f \quad \text{f is an integer}$$

$$x_k = m_j + m_k f$$

$$m \left(\frac{m_j + m_i f}{m_j m_i} \right) \equiv d \pmod{m_j}$$

$$m \left(\frac{m_j + m_k f}{m_j m_k} \right) \equiv d \pmod{m_j}$$

$$\frac{m}{m_i} \equiv 0 \pmod{m_j}$$

$$\frac{m}{m_k} \equiv 0 \pmod{m_j}$$

Thus both equations reduce to

$$\frac{mf}{m_j} \equiv d \text{ Mod } m_j$$

The same solution is obtained for $x_i = m_i f$ and $x_k = m_k f$ (combination one) therefore:

$$\frac{x_i}{m_i} - \frac{x_k}{m_k} = \left(\frac{m_j}{m_i} - \frac{m_j}{m_k} \right) \text{ or } 0$$

implies a single value of d and hence $S_i = S_k$ only if $d_i = d_k$. It follows that none of the solutions are identical. However, unless some auxiliary error detecting scheme is employed it is impossible to separate the correct from the incorrect solutions.

If two redundant digits are employed it becomes possible to distinguish between the correct and the incorrect solutions. The representation consists of $t+2$ digits and the number of solutions is the number of combinations of $t+2$ digits taken t at a time. When only one digit is in error the number of correct solutions will be equal to the number of combinations of $t+1$ things taken t at a time. The number of correct solutions is $t+1$. Since the total number of solutions is $\frac{(t+1)(t+2)}{2}$ the ratio of correct to incorrect solutions is given as $\frac{2}{t+2}$. We shall now show for a residue code with two redundant digits that none of the erroneous solutions are identical. Typically Δ is of the form

$$\Delta = \frac{x_{ik} m}{m_j m_i m_k} \equiv d \text{ Mod } m_j$$

A pair of erroneous solutions S_{ik} and S_{rs} are defined by

$$S + \frac{x_{ik}m}{m_j m_i m_k} = S_{ik} + a \frac{m}{m_i m_k}$$

$$S + \frac{x_{rs}m}{m_j m_r m_s} = S_{rs} + b \frac{m}{m_r m_s}$$

$$a = 0, 1; \quad b = 0, 1$$

Let us assume $S_{ik} = S_{rs}$ then

$$\frac{x_{ik}}{m_i m_k} - \frac{x_{rs}}{m_r m_s} = \frac{am_j}{m_i m_k} - \frac{bm_j}{m_r m_s}$$

If $i = r$ then the relationship becomes

$$\frac{x_{ik}}{m_k} - \frac{x_{rs}}{m_s} = \frac{am_j}{m_k} - \frac{bm_j}{m_s}$$

This situation has already been considered for the case of one redundant digit. We need, therefore, only consider the case where $i \neq r$.

Four combinations exist since the possible values of a and b are zero and one,

$$(1) \quad \frac{x_{ik}}{m_i m_k} - \frac{x_{rs}}{m_r m_s} = 0$$

$$(2) \quad \neq \frac{m_j}{m_i m_k}$$

$$(3) \quad \neq \frac{m_j}{m_r m_s}$$

$$(4) \quad = \frac{m_j}{m_i m_k} - \frac{m_j}{m_r m_s}$$

The first combination is satisfied only if

$$x_{ik} = f m_i m_k$$

$$x_{rs} = f m_r m_s$$

The fourth combination is satisfied only if

$$x_{ik} = m_j + f m_i m_k$$
$$x_{rs} = m_j + f m_r m_s$$

The results indicate that identical erroneous solutions must correspond to the same d . Erroneous solutions are not identical.

Since the erroneous solutions are not identical, the presence of two identical solutions is sufficient to indicate the correct solution. It is then possible to use a smaller set of solutions involving less computation. The set of solutions is chosen so that for every base there exist at least two solutions which are not functions of that base. Let the total number of digits of a residue code be designated as n . This includes both information and redundant digits. If n is even the above requirement may be met with n properly chosen solutions. If n is odd then $n+1$ solutions are required. There exists considerable flexibility in the choice of the solutions. If the number of solutions used is somewhat more than the required minimum the range of the representation is extended. This fact may be illustrated by an example. Consider a residue representation of six digits. The bases are consecutive primes. The lowest base is two. Six solutions are chosen with the following base pairs deleted: $a_1 a_6, a_2 a_6, a_1 a_5, a_2 a_4, a_3 a_5, a_3 a_4$. The range of the representation is given by the expression

$$0 \leq R < \frac{m}{m_i m_j}$$

where $m_1 m_j$ is the largest product of bases corresponding to the deleted pairs of digits. For this example the largest product of deleted base digits corresponds to $m_3 m_5$ and is equal to 55. The range of the representation if six solutions are used is zero to 551. If the code does not have the property of single error correction the largest number represented is 30030. The range of the representation can easily be extended by the addition of only one solution. The solution which did not involve a_3 and a_5 is replaced by a solution deleting a_5 and a_1 or a_2 and a solution deleting a_3 and a_1 or a_2 is added. By the addition of one solution the largest number represented for the example above is $(m/m_2 m_6) - 1$ which is equal to 770.

The maximum range of the representation of an n digit residue code with single error correction is given as

$$0 \leq R_m < \frac{m}{m_2 m_n}$$

An upper limit for the number of solutions required to realize this range is obtained by considering the deletion of the digits a_3 to a_n with the deletion of a_1 and a_2 . This scheme will yield $2n-4$ solutions and represents the absolute maximum number of solutions required to realize the above range. Usually the required number of solutions will be less than the maximum.

The efficiency of the residue code for the correction and detection of single digit errors is now compared with the theoretical efficiency predicted by Shannon's theory.⁽⁴⁾ It is known that a single error detecting and correcting code of the Hamming type obtains the

maximum theoretical efficiency if the total number of digits of the representation is $2^m - 1$, where m is any non zero positive integer. The Hamming code is least efficient when the total number of digits is equal to 2^m . The residue code will be considered from two viewpoints. First, the number of represented states will be translated to bits without regard to the form of the final code. Secondly the residue code will be considered in terms of a corresponding binary code for each residue digit. The binary coded residue representation does not use the information capacity of the binary code in the most efficient manner. However, the binary coded residue representation is one form in which the residue code would be used in practice.

The data of Table X indicates the efficiency of a residue code constructed from consecutive prime bases starting from two. The code has error detecting and correcting properties and the efficiency is defined as the ratio of the logarithm base two of the number of states of the representation corresponding to information to the logarithm base two of the total number of states of the representation.

The theoretical maximum efficiency for single error correction and detection is obtained for the assumption that for an m digit code $m+1$ events may occur with equal probability. The possible events are any single digit in error or no error.

TABLE X
RESIDUE CODE EFFICIENCY

Number of Residue Digits	Efficiency of the Residue Code	Efficiency of the Binary Encoded Residue Representation
4	.43	.37
5	.55	.47
6	.65	.56
7	.70	.60
8	.75	.64
9	.78	.68
10	.80	.71
11	.83	.74

The following formula due to Shannon specifies the maximum efficiency.

$$C = \log_2 N - \sum_{i=1}^{m+1} p_i \log_2 p_i$$

The bits of information per binary symbol is given by

$$\frac{C}{m} = 1 - \frac{1}{m} \log_2 p_i$$

The results of Table X and the above formula are compared graphically in Figure 15. The graphical presentation permits a rapid comparison of the relative efficiencies of the various codes in spite of the fact that the total number of binary digits varies. The variation in the number of binary digits associated with a particular number of residue digits is due to the inefficient use of binary digits when each digit of the residue code is expressed by a binary code.

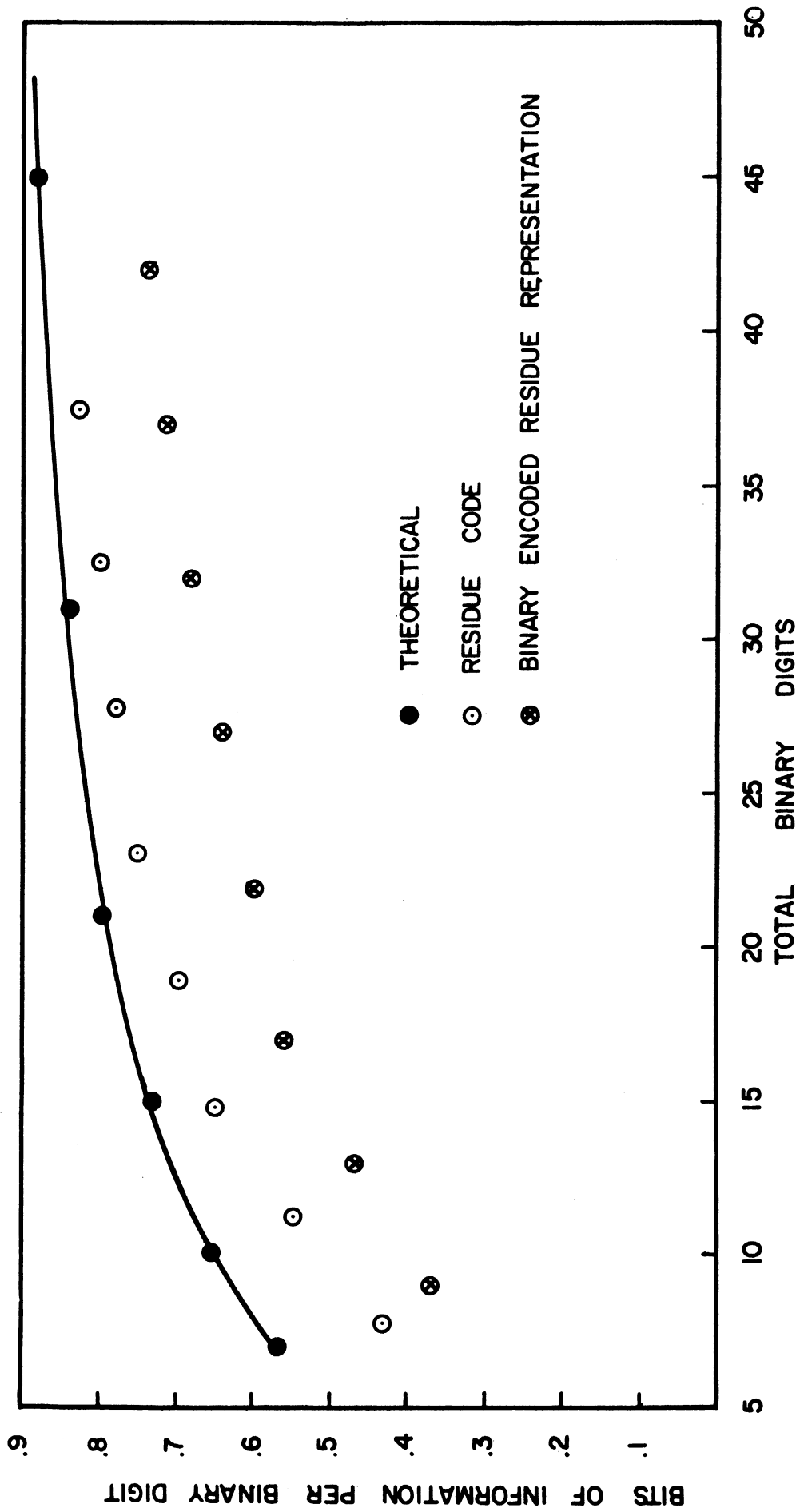


Figure 15. Relative Efficiency of the Residue Code.

Division

In any number system the operation of division is a problem. This is particularly true of the residue system. Every element of a field except zero is associated with a unique multiplicative inverse. This is one of the requirements which must be met by a mathematical system if it is to be a field. The multiplicative inverse of a field element x is denoted by x^{-1} and is defined by the following relationship

$$x x^{-1} \equiv 1 \text{ Mod } m_x$$

The definition also requires closure.

The division process for residue codes is complicated by two factors. The first is the absence of a multiplicative inverse for the zero element. The second difficulty is the fact that residue division and the normal division process are in one to one correspondence only when the resulting quotient is an integer value. We shall consider first the problem of residue division of the elements of a single field and shall consider later the elements of several fields considered as a residue code. The division process represented in equation form as

$$\frac{a}{b} = q$$

implies the following equation:

$$a = bq$$

The difference between normal arithmetic and residue arithmetic is that in residue arithmetic the product bq need not necessarily be equal to a , only the congruence of a and bq is required.

$$bq \equiv a \text{ Mod } m_n$$

Multiplication by the multiplicative inverse of b obtains

$$q \equiv a/b \pmod{m_n}$$

The correct interpretation of q in the above equation is that the number a is obtained by forming the sum consisting of b , q representations. The sum is carried out in a closed number system of base m_n . Thus q corresponds to the quotient only when the quotient has an integer value. Examples may be obtained from the consideration of a modulo 5 field.

$$\frac{4}{2} = q$$

$$2q \equiv 4 \pmod{5}$$

$$q \equiv 2 \pmod{5}$$

$$\frac{4}{3} = q$$

$$3q \equiv 4 \pmod{5}$$

$$q \equiv 3 \pmod{5}$$

note $3 \times 3 \equiv 4 \pmod{5}$

$$\frac{3}{4} = q$$

$$4q \equiv 3 \pmod{5}$$

$$q \equiv 2 \pmod{5}$$

In the above examples q corresponds to the quotient only in the first example.

The residue code representation of a number consists of many elements. Each digit of the representation is associated with a different prime base. In other words each digit of the representation is an element of a different field. The division of two numbers in residue code may be expressed by a system of congruences. The solution $A = (q_1, q_2, \dots, q_n)$ must satisfy all the congruence relationships of the system. A zero digit in the divisor $B = (b_1, b_2, \dots, b_n)$ must be given special consideration. The congruence corresponding to any $b_i = 0$ must be removed from the system. The deletion of the congruences for which $b_i = 0$ is consistent with the residue definition of division given above. In general, if some b_i equal zero then

$$QB \not\equiv A \text{ Mod } m$$

For the special case in which $b_i = 0$ and

$$QB \equiv A \text{ Mod } m$$

it is also true that Q has an integer value. The deletion of the congruence associated with $b_i = 0$ results in a relationship of the following form if there are no elements $b_j = 0$.

$$QB_1 \equiv A_1 \text{ Mod } \frac{m}{\sum_{i=1}^n m_i}$$

The examples which follow are presented to clarify the process of residue division. Consider a residue code with bases 2, 3, 5 and 7. Since some confusion might exist between the normal number representation and the residue representation the residue representations

are enclosed in parenthesis.

$$q = \frac{9}{4} \quad \left(\frac{1042}{0144} \right) = (q) \quad (1042) = q(0144)$$

$$q = (x014)$$

in terms of congruences

$$4q \equiv 9 \pmod{105}$$

$$324 \equiv 9 \pmod{105}$$

$$q = \frac{29}{11} \quad (q) = \left(\frac{1241}{1214} \right) = (1142) \leftrightarrow 79$$

$$79 \times 11 \equiv 29 \pmod{210}$$

$$q = \frac{7}{6} \quad (q) = \left(\frac{1120}{0016} \right) = (xx 20)$$

$$6 \times 7 \equiv 7 \pmod{35}$$

$$q = \frac{24}{8} \quad (q) = \left(\frac{0043}{0231} \right) = (x 223)$$

$$8 \times 3 \equiv 24 \pmod{105}$$

The process of residue division has certain interesting properties and quite possibly has applications in respect to special problems. Unfortunately, the residue division process is not a substitute for normal division. It appears that the only way in which division can be effected in the residue code is by the utilization of techniques similar to those employed for division in a consistently weighted number system. The division process then requires trial and error subtraction or addition and the greater than or less than relationship. The division algorithm could also include trial multiplication since in the residue system addition and multiplication require the same period of time.

Conclusions

The material of this chapter forms a preliminary investigation of the applicability of residue number systems to the arithmetic operations of digital computers. The residue system has been found attractive in terms of the operations of multiplication and addition. It is possible to realize practical logical circuitry to yield the product in the same operation time as for the sum since the product is not obtained by the usual procedure of repetitive addition. The main disadvantages of the residue number system are associated with the necessity of determining absolute magnitude. Thus the division process, the detection of an overflow and the determination of the correct sign of a subtraction operation are operations which at this stage of the investigation seem to involve considerable complexity. Nevertheless there are certainly many special purpose applications well suited to the residue code. In particular there exists a class of control problems characterized by the absence of the need for division, the existence of a well defined range for the variables and also by the fact that the sign of the variables is known. For the problems of this class, the use of the residue code should result in a reduction of the overall computation period and give a computer with a higher bandwidth than obtainable with the conventional number system. It seems that one appropriate application of the residue code would be in the design of a concurrent or parallel digital differential analyzer.

It has been shown that the residue code with redundancy can never obtain the theoretical efficiency predicted by Shannon. The code does have the obvious advantage that an erroneous digit does not result in a propagated error. It is not unlikely that the residue code could have important applications in the field of communications. This statement is based on the fact that a sine wave may be residue encoded by phase modulation.

The ultimate usefulness of the residue code will probably be determined largely by the success of the circuit designer in perfecting circuitry ideally suited for residue code operations. In this respect there is one recent development in permanent storage devices which should revolutionize some of the standard concepts of computer design. It is not unlikely that the computer of the future will consist in part of a large number of addressable permanent tables. The standard computer operations would be executed by a series of table look-ups. The designer of such a computer should give serious consideration to the use of a residue code. It appears that the combination of residue coding and permanent addressable tables obtains numerous advantages. The combination would permit arithmetic without carry and the inclusion of the redundancy needed for error detection and correction. The permanent storage tables would greatly simplify the problems associated with the determination of signs and relative magnitude and the error correcting and detecting routine.

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

This thesis has dealt primarily with techniques appropriate to error checking in a computer arithmetic unit. Particular attention has been given to the logic and error structure associated with the binary adder. The conventional digital parity check has been shown to have the properties of arithmetic invariance required to check arithmetic operations. The digital parity check has been applied previously to storage and transmission systems but to the author's knowledge the check has never been applied to an arithmetic system. Previous applications of the digital parity check in computer systems have determined the parity of the result of an arithmetic operation by a means independent of the parity of the operands. A parity checking scheme which provides a check of the arithmetic operations requires cognizance of the number of generated carries. This fact is one decided disadvantage of the digital checking system. The digital parity check is not necessarily restricted to binary number systems. The digital parity check may be defined and applied to check arithmetic operations for any consistently weighted number system regardless of the base. The essential features of the digital parity check are the dependence of the check on the magnitude of the symbols of a number representation and the independence of the check on the magnitude of the weight associated with the symbols. The

definition of the digital parity check in terms of the sum of the symbols modulo b where b is the base of the number system is a much more fundamental definition than the usual definition of the digital parity check given for the binary number system. The usual definition of the digital parity check is given in terms of the oddness or the evenness of the number of ones contained in the representation.

The numerical check is based on the concept of linear congruences. The check base may therefore be any number except the magnitude of the number base. The main advantage of the numerical check is that carry cognizance is not required. The check is, however, sensitive to carry malfunctions.

The question of the effectiveness of the different parity checks in the detection of errors is closely related to the logical structure of the adder. A cursory examination of the problem reveals immediately that the major difficulty is associated with the propagation of carries in the addition process. If it were not for carry propagation it would be easy to obtain independence of the elements involved in the addition process. If the elements were independent then the simple digital parity check would be one hundred percent effective against single component malfunctions. It is possible to design arithmetic structures in which each carry element is determined independently. However a severe penalty in terms of time and equipment is associated with structures of this type. For example the carry logic normally associated with a 30 bit binary adder with non-independent carry elements consists of 29 AND gates and 29 OR gates each gate

having two inputs. The generation of instantaneous and independent carry elements would require 29 OR gates with 464 inputs and 435 AND gates with 4930 inputs. One example of a non instantaneous but independent carry logic requires 29 OR gates with 464 inputs and 481 AND gates with 1682 inputs. These figures illustrate the high price paid to obtain independence of the carry elements. There may exist some carry logic design in between the extremes illustrated by the examples given. In particular it should be possible to design a carry logic for which the carry elements are not completely independent but are less dependent than the carry elements of the standard carry logic. It should be possible to synthesize carry logic of this type by an extension of the vector space concepts which have been used to investigate the structure of binary addition. The extension of the vector space concepts to the problem of the synthesis of arithmetic structures has been given only brief consideration and should be an important area for future work. Vector space concepts have been found particularly appropriate for the analysis of the process of binary addition. The author found that the study of the binary addition process by means of vectors tended to clarify the problems associated with the structure of the binary adder. In particular the introduction of the carry matrix permitted a unified approach to the study of different configurations of carry logic.

A natural extension of the vector space concept of measure has resulted in the specification of three new addition algorithms.

The algorithms provide an interesting description of the addition process but do not lead to logical circuits having any advantage over the conventional circuits.

The study of the structure of the errors for a standard parallel adder has shown that the propagated errors due to single logical element malfunctions exist in only two forms. The two forms are a series of zeros terminated by a one or a series of ones terminated by a zero. The form of the error sequence is due largely to the fact that the propagation of an error to some element s_{i+1} requires that $r_i = 1$. Error propagation is halted when some element of the ρ vector, $r_j = 0$, where $j > i$. The probability of error propagation has been found to be a function of the type of malfunction which initiated the error. The study of the error structure of the standard adder has revealed the existence of only four different classes of error propagation probabilities corresponding to all of the possible single logical malfunctions. Thus the effectiveness of the digital parity check is a function of the type of malfunction. A simple digital parity check consisting of only one parity check bit has a probability of success of $2/3$, $3/4$, $1/3$ and $5/9$ for each of the four possible error propagation probabilities. The digital check may be considerably more effective in the detection of errors if more than one parity check digit is employed. For example if two digits are employed in such a manner that alternate digits are associated with one check bit and the other digits with the other check digit then the probabilities associated with the success of the check

become $14/15$, $19/20$, $13/15$ and $41/45$ respectively. The extension of the digital check to include additional check digits is not particularly fruitful since the numerical parity check has been shown to be 100% effective. Furthermore the numerical check has the advantage of not requiring carry cognizance. The figures which have been stated for the effectiveness of the various checks apply only to the results of single addition-like operations. Multiplication and division, which consist of repetitive addition, must be checked at each addition step in order to obtain the parity check effectiveness stated above. The parity effectiveness for a sequence of addition operations has not been investigated in this paper. This would be an excellent topic for future study.

The numerical check is superior to the digital check for the purposes of error detection. However there are certain difficulties associated with numerical checking if error correcting properties are desired. The extension of the numerical check to include additional check digits will not provide the information needed to correct an erroneous representation unless the check representation contains all of the information of the original representation. In this case it becomes possible to carry out the arithmetic operations in terms of the check representation and the original representation may be ignored. We have called the numerical check representation for this case a residue number system. There appears to be a fundamental limitation in regard to the application of numerical checking procedures for the purposes of error correcting since the numerical check is dependent on

magnitude. For example a numerical check modulo 35 consisting of check bases 7 and 5 would provide error correction information for any error of magnitude less than 35. If the magnitude of the error is greater than or equal to 35 the correction data is ambiguous. It would seem that there exists an interesting possible combination of error detecting techniques used in conjunction with a reasonable amount of circuit duplication.

The residue number system has been investigated in detail. The main advantage of the number system appears to be the complete independence of the digits of a residue representation. Addition and multiplication are both defined operations and are executed without carry between the residue digits. However, complete element independence may not be obtained within a residue digit when the digit is represented in binary code unless special logical circuitry is employed. Independent addition elements must be employed within the adder logic of a single residue digit to give complete bit by bit independence. Also if standard carry logic were employed within a residue digit the addition time while still less than that of the standard adder would not be significantly less. Multiplication in a residue code would be significantly faster than that presently achieved by standard techniques even if standard carry logic were employed within the residue digits. However, it is doubtful that the residue number system will ever be used extensively for general purpose computation due to the difficulties involved in performing the division operation. The basic difficulty

pertains to the determination of magnitude. The ultimate success of the residue code for general purpose computation is very much contingent upon the development of special components suitable to the residue code. At the present time the most probable application of the residue code is in the realm of special purpose computation. The residue code is unique in the sense that the residue digits are independent. Furthermore, the code may obtain error correcting capabilities simply by the addition of more residue digits providing of course that the base of the added digits is relatively prime to the bases of the other digits. The error correcting routine required by the residue code is moderately complicated when considered in terms of equipment. The success of the residue code as a means of error correction is dependent on the development of components suited to the properties of the residue code.

It appears that the control of computation errors due to circuit malfunctions may be obtained by means of error detecting and correcting codes which are arithmetically invariant. In this thesis the simple digital parity check and the numerical parity check have been considered in detail. While the numerical check is very effective in error detection it is not particularly suited for error correction in a general computing system. It is suggested that perhaps the most effective control of errors is to be obtained by the use of a hybrid system employing both component redundancy and error coding.

APPENDIX

SIGN AND OVERFLOW CONVENTION FOR ONE'S AND TWO'S COMPLEMENT ARITHMETIC

The following material is a detailed description of one possible sign and overflow convention for binary arithmetic. The material is pertinent to the discussion of the form of errors which occur in the process of binary addition. This subject was discussed in Chapter IV. We consider here the addition and the subtraction processes for both one's complement and two's complement arithmetic.

It is conventional to speak of a binary representation to the left of the radix point as consisting of n digits plus a sign. In this case the sign would be the n th plus one digit and the last digit or element of the carry vector would be the n th plus two digit. However, the convention adopted in Chapter III is somewhat different than this. The convention of Chapter III will be used in the remaining discussion. In this convention the number including sign is represented by n digits or elements to the left of the radix point and m digits to the right of the radix point.

For two's complement arithmetic the complement representation of a negative number $-A$ is defined by the following relationship

$$A' = 2^n - A \quad \text{where } |A| < 2^{n-1}$$

In two's complement arithmetic zero is represented as a positive entity. The largest negative number is -2^{n-1} and is represented by 2^{n-1} in the two's complement representation. The largest positive number representable by the $n+m$ elements is $2^{n-1} - 2^{-m}$. We now proceed to examine

the results of the addition process for four different augend, addend combinations.

Consider first the case where both the augend and addend are positive. The resulting sum must also be positive. Therefore, any sum which carries to the n th element constitutes an overflow of the representation. It is readily verified that every sum of two positive operands which overflows will cause a one to appear in the n th element of the sum. The smallest overflow is 2^{n-1} while the largest overflow obtained by the addition of the two largest positive numbers representable by n digits is $2^n - 2^{m+1}$. Both of these overflow sums contain a one in the n th digit place.

We consider next the addition of a positive and a negative operand. In regard to the overflow problem there is no difficulty since an overflow is not possible. An important question is whether the last carry element may assume a value of unity. The last carry element corresponds to the $n + 1$ element of the sum. It is obvious that if the resulting sum is negative then the $n + 1$ element cannot equal zero for the n th element of the sum is a one and r_n equals one and c_n must equal zero. It is therefore, impossible for c_{n+1} to equal unity. On the other hand if the resulting sum is positive it is possible for the c_{n+1} digits to have a value of unity. Normally in two's complement arithmetic this digit is ignored. The positive sum obtained by the addition of a pair of positive and negative operands extends over a range defined by the addition of the largest positive

number and, in absolute magnitude, the smallest negative number to the number zero. Zero is obtained by the addition of a positive number and a negative number identical in absolute magnitude.

$$2^n \leq R \leq 2^n + 2^{n-1} - 2^{-m}$$

It is observed that the n th + 1 element is always a one over the range of the positive sum. Furthermore, the n th element has a value of zero.

The last case is the addition of two negative operands. In this case the n th element of both operands has a value of unity and hence, the n th + 1 element of the sum and the carry vector will always be unity. The investigation here concerns the behavior of the n th digit. The sum of two negative operands can overflow. We consider first the range of the non overflow sum obtained by the addition of the two smallest negative numbers (absolute magnitude) and the sum of any two negative numbers which produces the largest (absolute magnitude) negative number which can be represented. This range is given as

$$2^{n-1} \leq R \leq 2^n - 2^{-m}$$

If the sum of the two negative operands overflows the result must lie in the range specified by the sum of the two negative numbers largest in absolute magnitude and the sum of the largest negative number in absolute magnitude and minus one.

$$2^n \leq 0 \leq 2^n + 2^{n-1} - 2^{-m}$$

It is observed that if no overflow occurs the n th digit of the sum is correct. The sum is negative and the n th digit is a one. If an overflow occurs the n th digit becomes a zero which is the incorrect sign. The n th + 1 digit is a one for both situations.

In ones complement arithmetic the minimum negative number in absolute magnitude is zero given as $2^n - 2^{-m}$. The magnitude of the largest negative number is $2^{n-1} - 2^{-m}$ and is represented as $2^n - 2^{n-1}$. The largest and smallest positive numbers are $2^{n-1} - 2^{-m}$ and 2^{-m} .

The overflow representation, due to the addition of two positive numbers, lies in the range

$$2^{n-1} \leq 0 \leq 2^n - 2^{-m+1}$$

The overflow does not affect the $n+1$ digit but does produce a one in the n th digit. The overflow is detected by the erroneous sign. The normal addition of two positive operands produces a sum in the range

$$2^{-m+1} \leq R \leq 2^{n-1} - 2^{-m}$$

The addition of two negative numbers always requires an end around carry correction. Before correction the range of the correct sum and the overflow sum are

$$2^{n+1} - 2^{n-1} - 2^{-m} \leq R \leq 2^{n+1} - 2^{-m+1}$$

$$2^{n+1} - 2^n \leq 0 \leq 2^{n+1} - 2^{n-1} - 2^{-m+1}$$

Before correction the overflow representation and the normal representation will always have a one value for the correction digit. The sign digit for the correct representation is either a zero or a one. The sign digit of the overflow representation is always zero. After correction the overflow and normal range of the sum is

$$2^{n+1} - 2^{n-1} \leq R \leq 2^{n+1} - 2^{-m}$$

$$2^{n+1} - 2^n + 2^{-m} \leq 0 \leq 2^{n+1} - 2^{n-1} - 2^{-m}$$

After correction the correction digit remains a one for both the normal and the overflow representation. The sign digit for the normal sum is

a one while the sign for the overflow representation is a zero which is incorrect.

The addition of a positive and a negative operand can never produce an overflow but the sum may be either positive or negative. If the sum is positive an end around correction is required. This is verified by

$$\begin{aligned} S &= A + \bar{B} && \text{where } |A| > |B| \\ &= 2^n - B - 2^{-m} + A && A \leq 2^{n-1} - 2^{-m} \\ 2^n &\leq S \leq 2^n + 2^{n-1} - 2^{-m+1} \end{aligned}$$

Thus an end around correction is always required for a positive sum and the sign is correct both before and after the correction. After correction the representation is in the range.

$$2^n + 2^{-m} \leq S \leq 2^n + 2^{n-1} - 2^{-m}$$

The negative sum is obtained for $S = A + \bar{B}$, if $|A| \leq |B|$

$$2^{n-1} \leq S \leq 2^n - 2^{-m}$$

The negative sum obtained has a zero value for the correction digit and a one value for the sign digit. End around carry correction is not required.

BIBLIOGRAPHY

1. Babbage, H., Calculating Engines, E. and F. N. Spon Co., London, Eng.; 1889.
2. Goldstine, A., Goldstine, H. H., "The Electronic Numerical Integrator (ENIAC)," Mathematical Tables and Other Aids to Computation, Vol. 1, pp. 97-110; July, 1946.
3. Von Neumann, J., "Probabilistic Logics and the Synthesis of Reliable Organisms From Unreliable Components," Automata Studies, Edited by Shannon C. E. and McCarthy, J., Princeton University Press, 1956, pp. 43-98.
4. Shannon, C. E., Weaver, W., The Mathematical Theory of Communications, University of Illinois Press, Urbana, Ill., 1949, pp. 44-48.
5. Hamming, R. W., "Error Detecting and Correcting Codes," The Bell Telephone System Technical Journal, Vol. XXVI, No. 2, pp. 147-160, April, 1950.
6. Golay, M. J. E., "Notes on Digital Coding," Proceedings of the I.R.E., Vol. XXXVII, No. 6, p. 657, June, 1949.
7. Ulrich, W., "Non Binary Error Correcting Codes," The Bell Telephone System Technical Journal, Vol. XXXVI, No. 6, pp. 1341-1387, Nov., 1957.
8. Reed, I. S., "A Class of Multiple Error Correcting Codes and the Decoding Scheme," Transactions of the 1954 Symposium on Information Theory, I.R.E. Professional Group on Information Theory, pp. 38-49, Sept., 1954.
9. Golay, M. J. E., "Binary Coding," Transactions of the 1954 Symposium on Information Theory, I.R.E. Professional Group on Information Theory, pp. 23-28, Sept., 1954.
10. Robertson, J. E., "Error Detection and Correction in Binary Parallel Digital Computers," Electronic Digital Computer, Internal Report No. 37, University of Illinois Digital Computer Laboratory, pp. 70-81, 1952.
11. Block, R. M., Cambell, R. V. D., Ellis, M., "The Logical Design of the Raytheon Computer," Mathematical Tables and Aids to Computation, Vol. III, No. 24, pp. 286-296, 317-323, Oct., 1948.
12. Block, R., Patent #2,634,052.

BIBLIOGRAPHY (CONT'D)

13. Hardy, G. H., Wright, E. M., An Introduction to the Theory of Numbers, Oxford University Press, London, England, 1956.
14. Birkhoff and MacLane, A Survey of Modern Algebra, Macmillan Co., New York, N. Y., 1948.
15. Brillouin, L., Les Tenseurs en Mecanique et en Elasticite, Chapter VI, Dover Publications, New York, N. Y., 1946.
16. Weinberger and Smith, "One Microsecond Adder Using One Megacycle Circuitry," I.R.E. Transactions on Electronic Computers, Vol. EC-2, No. 2, June, 1956.
17. Burks, A. W., Goldstine, H. H., Von Neumann, J., "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," Part 1, Vol. 1, pp. 10-11, Princeton: Institute for Advanced Study, 1947, Second Edition.
18. Gilchrist, B., Pomerene, J. H., Wong, S. Y., "Fast Carry Logic for Digital Computers," Transaction of I.R.E. Professional Group on Electronic Computers, Vol. EC-4, No. 4, December, 1955.
19. Lunts, A. G. "The Application of Boolean Matrix Algebra to the Analysis and Synthesis of Relay Contact Networks," Doklady Akademii Nank SSSR 70, pp. 421-23, 1950.
20. Leonard, E. D., Modern Elementary Theory of Numbers, p. 19, University of Chicago Press, Chicago, Ill., 1939.
21. Cameau, J. O., "The Synthesis and Analysis of Digital Systems by Boolean Matrices," Transactions of the I.R.E. Professional Group on Electronic Computers, Vol. EC-3, No. 3, pp. 6-11; September, 1954.
22. Muller, D.E., "Application of Boolean Algebra to Switching Circuit Design and to Error Detection," Transactions of the I.R.E. Professional Group on Electronic Computers, Vol. EC-3, No. 3, pp. 6-11; Sept., 1954.

UNIVERSITY OF MICHIGAN



3 9015 02826 0852