1974

Performance variations of the binary (15.7) BCH code related to changes in channel crossover probability and decoding method /

Dwight Wingham Aten
Lehigh University

PERFORMANCE VARIATIONS OF THE BINARY (15,7) BCH CODE RELATED TO CHANGES IN CHANNEL CROSSOVER PROBABILITY AND DECODING METHOD

by

Dwight Wingham Aten

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Electrical Engineering

Lehigh University

1974

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

May 2, 1974
(date)

Joseph C. Mizell
Professor in Charge

[illegible]
Chairman of Department

ACKNOWLEDGMENTS

My thanks to my advisor, Dr. Joseph Mixsell, who provided the idea behind the central thrust of this thesis, and who always made himself readily available to supply the counsel and support which enabled me to complete the work.

I am also most grateful for the enthusiasm and encouragement of my parents, without which this work might not have been undertaken.

Dwight W. Aten

TABLE OF CONTENTS

ABSTRACT

This work examines through experimental means the capabilities of the binary (15,7) BCH code, which is guaranteed by algebraic coding theory to correct two or fewer errors in a fifteen bit block. A program was written for the PDP-8 minicomputer to create and encode information sequences, and to simulate transmission of the resulting codewords over a binary symmetric channel with variable crossover probability. In addition, the program decodes each received message with maximum likelihood, majority logic, and Berlekamp decoders.

When three or more errors are present in a fifteen bit block, the results indicate variations in the performance of the code depending on the decoder used. Under certain circumstances the maximum likelihood decoder was capable of correcting more than 20% of the messages having three or more errors. Under the same conditions the majority logic decoder corrected more than 10% of such messages, while decoding faster than either of its two competitors. The Berlekamp decoder gave the poorest performance, but was still able to correct about 5% of the messages having three or more errors.

The results showed variation in the code's

performance when viewed from a statistical standpoint. However, each decoder extracted more error correcting ability from the code than the minimum promised by algebraic coding theory. Additionally, the majority logic decoder was shown to be highly practical for decoding the binary (15,7) BCH code.

INTRODUCTION

Coding provides a means for correcting errors introduced into data during its transmission. This ability is measured by the largest number of errors per block of data which a code is guaranteed to correct. That number, while it is one of the key figures of merit offered by algebraic coding theory, really delimits the minimum ability of the code. As a practical matter, a code's error correcting ability is influenced by the decoding method used to recover the information. This work is an effort to explore through experimental means the nature of this influence in a specific case, that of the binary (15,7) BCH code.

The binary (15,7) BCH code is a linear, cyclic block code. It has a block length of fifteen bits, with seven information bits and eight parity check bits. Regardless of the decoding method, the binary (15,7) BCH code can correct any pattern of two errors within a block, since it has a minimum distance of five. However, some codewords are separated from all others by distances greater than five. A decoding method taking advantage of this fact should offer improved performance.

The binary (15,7) BCH code provides a good subject for this work because it is amenable to decoding by a variety of methods. As it is a binary code, it can be decoded by a simplified version of Berlekamp's algorithm[1]. It yields to maximum likelihood decoding, which usually is impractical, with only modest effort, there being but 128 codewords. Also, it is the only BCH code known which is one step majority logic decodable. An additional favorable feature of the binary (15,7) BCH code is its fifteen bit block length, which is sufficiently short to permit single codeword containment in a minimum number of computer words. This is important because a study such as this must as a practical matter be computer aided. The computer employed was the Digital Equipment Corporation's PDP-8.

The overall strategy involves, first, pseudo-random generation of an information sequence, followed by its encoding. The codeword is transmitted over a binary symmetric channel with crossover probability p_e. It is then decoded in turn by each of the three decoding methods mentioned previously. The decoded information is compared with the source sequence. Records are kept of the number of times each decoder fails to recover the transmitted information sequence from the received message.

The writing of the program for the PDP-8 which accomplishes the procedure just outlined was the major endeavor of this thesis work. It can process roughly nine thousand codewords in less than ten minutes, making possible this study on a large statistical sample.

BCH CODES AND THE BERLEKAMP DECODING ALGORITHM

BCH codes are a class of linear, cyclic, random error correcting codes. Discovered by Hocquenghem(2) in 1959, and independently by Bose and Chaudhuri(3) in 1960, they have received wide exposition in coding theory literature and have been used in many practical applications. The decoding technique credited to Berlekamp is inherantly relevant in a generic discussion of BCH codes because of its universal applicability. The summary which follows includes an outline of the special version of the Berlekamp decoding algorithm used in this work.

BCH codes are characterized by the following parameters:

Block length:	$n = 2^m - 1$
Number of parity check digits:	$n - k \leq mt$
Minimum distance:	$d \geq 2t + 1$

where m and t are positive integers and $t < 2^{m-1}$. They can correct any pattern of t or fewer errors in a block. The generator polynomial for a BCH code is given by

$$g(X) = \mathrm{LCM}(m_1(X), m_3(X), \ldots\ldots, m_{2t-1}(X))$$

where $m_i(X)$ is the minimum polynomial of α^i, and α^i is a primitive element of the Galois field $GF(2^m)$.

Let

$$v(X) = v_0 + v_1X + v_2X^2 + \dots + v_{n-1}X^{n-1}$$

represent a transmitted code vector and

$$r(X) = r_0 + r_1X + r_2X^2 + \dots + r_{n-1}X^{n-1}$$

represent the received code vector. Define a syndrome vector S, with 2t components such that each

$$S_i = r(\alpha^i)$$

for $i = 1, 2, \dots, 2t$. The received code vector $r(X)$ is the sum of the error vector

$$e(X) = e_0 + e_1X + e_2X^2 + \dots + e_{n-1}X^{n-1}$$

and the transmitted code vector:

$$r(X) = v(X) + e(X).$$

Rewriting the expression for S_i in light of this:

$$S_i = v(\alpha^i) + e(\alpha^i).$$

However, each syndrome vector component corresponding to the transmitted vector $v(X)$ is zero. This leaves

$$S_i = e(\alpha^i)$$

for i = 1, 2, ..., 2t. The S_i, which are computed from the received code vector, are the known values in a set of equations having the error location numbers as unknowns:

$$S_1 = e_0 + e_1(\alpha) + \dots\dots\dots + e_{n-1}(\alpha^{n-1})$$
$$S_2 = e_0 + e_1(\alpha^2) + \dots\dots\dots + e_{n-1}((\alpha^2)^{n-1})$$
$$\cdot$$
$$\cdot$$
$$S_{2t} = e_0 + e_1(\alpha^{2t}) + \dots\dots + e_{n-1}((\alpha^{2t})^{n-1})$$

Although this set of equations does not in general have a unique solution, the solution yielding an error pattern with fewest errors is the one sought. This solution will correctly specify the error location numbers provided there are t or fewer errors(4).

The error correction process entails solution of the above set of equations to obtain the error location numbers. The latter are the reciprocals of the roots of a polynomial, $\sigma(X)$, referred to as the error location polynomial. Berlekamp devised an iterative procedure for finding the error location polynomial, given the values of the S_i. This procedure, which involves finding 2t - 1 sets of table values, is applicable to the decoding of all BCH codes. The version of the procedure outlined here, which involves finding only t - 1 sets of table values, applies only to the decoding of binary BCH codes.

Set up the following table:

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$2\mu - l_\mu$
$-\frac{1}{2}$	1	1	0	-1
0	1	S_1	0	0
1				
2				
.				
.				
t				

Assuming all rows are filled up to and including the μth row, the $(\mu+1)$st row is completed as follows:

(1) If $d_\mu = 0$, then $\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X)$.

(2) If $d_\mu \neq 0$, find another row preceding the μth, say the ρth, such that the number $2\rho - l_\rho$ in the last column of the table is as large as possible and $d_\rho \neq 0$. Then

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X) + d_\mu d_\rho^{-1} X^{2(\mu-\rho)} \sigma^{(\rho)}(X).$$

In either case, $l_{\mu+1}$ is exactly the degree of $\sigma^{(\mu+1)}(X)$, and

$$d_{\mu+1} = S_{2\mu+3} + \sigma_1^{(\mu+1)} S_{2\mu+2} + \sigma_2^{(\mu+1)} S_{2\mu+1} + \dots + \sigma_{l_{\mu+1}}^{(\mu+1)} S_{2\mu+3-l_{\mu+1}},$$

where the $\sigma_i^{(\mu+1)}$ are the coefficients of $\sigma^{(\mu+1)}(X)$.

The value obtained for $\sigma^{(t)}(X)$ in the last row of the table is the error location polynomial, provided there are t or fewer errors. The reciprocal of each root of $\sigma^{(t)}(X)$ is an error location number.

THE BERLEKAMP DECODER

While much work is associated with software implementation of the Berlekamp procedure, the first problem is one of strategy. It is possible for the program to mimic precisely the procedural steps, computing interim error location polynomials until finally the correct one is arrived at. This is an intricate, therefore time consuming process. Alternately, determination can be made in advance of all the error location polynomials which can evolve from the procedure. These depend on the values of the S_i, and on the particular BCH code being decoded. A program, which is written for a particular code, can evaluate the S_i and select the correct error location polynomial accordingly. This method becomes increasingly advantageous if the number of possible error location polynomials is small.

In decoding the (15,7) binary BCH code the algorithm begins with three unfilled table rows. The first unfilled row is completed based solely on the value of S_1. The value of $\sigma^{(2)}(X)$ in the final table row depends on the values of both S_1 and S_3. The various possibilities can be seen in the following table:

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$2u - l_\mu$
$-\frac{1}{2}$	1	1	0	-1
0	1	S_1	0	0
1	$1 + S_1X$	$S_1S_2 + S_3$	0 (1) 1 (2)	2 (1) 1 (2)
2	$1 + S_1X + (S_3/S_1 + S_2)X^2$			

(1) for $S_1 = 0$
(2) for $S_1 \neq 0$

S_2 is treated as a dependent variable in this discussion because it can be shown that $S_2 = S_1^2$ (5). If $S_1 \neq 0$ and $S_3 = 0$, the error location polynomial is

$$\sigma^2(X) = 1 + S_1X + S_2X^2,$$

whereas if both $S_1 \neq 0$ and $S_3 \neq 0$, then

$$\sigma^2(X) = 1 + S_1X + (S_3/S_1 + S_2)X^2.$$

Because these are the only error location polynomials which arise in decoding the binary (15,7) BCH code, the second decoding strategy is selected.

This analysis might appear to have overlooked two possibilities. First, there is the case where $S_1 = 0$ and $S_3 = 0$. Technically, this case gives rise to a third error location polynomial, $\sigma^2(X) = 1$. This indicates no errors have been detected. More interesting is the case where $S_1 = 0$ and $S_3 \neq 0$, which indicates an uncorrectable error pattern has been encountered. To see this, consider the terms available to the summing process which specifies S_1. Since $S_1 = r(\alpha)$, these terms are the non zero elements of $GF(2^4)$, or $1, \alpha, \alpha^2, \ldots, \alpha^{14}$. In the case of $S_3 = r(\alpha^3)$, the available terms are $1, \alpha^3, \alpha^6, \alpha^9$, and α^{12}, each available thrice. If only one term is selected for summing in each case then, clearly, both S_1 and S_3 differ from zero. This happens when there is one error. In addition, no two distinct terms of $GF(2^4)$ sum to zero. Thus, when there are two errors S_1 is non zero. In order for S_3 to be non zero while S_1 is zero, three or more terms must be summed to get $S_1 = 0$. This means there are three or more errors, i.e., an uncorrectable error pattern. An example will help to illustrate the latter case. Suppose the error vector $e(X)$ is given by

$$e(X) = (0,0,0,0,1,0,1,0,0,0,0,0,1,0,0)$$

Then S_1 is

$$S_1 = e(\alpha) = \alpha^4 + \alpha^6 + \alpha^{12} = 0.$$

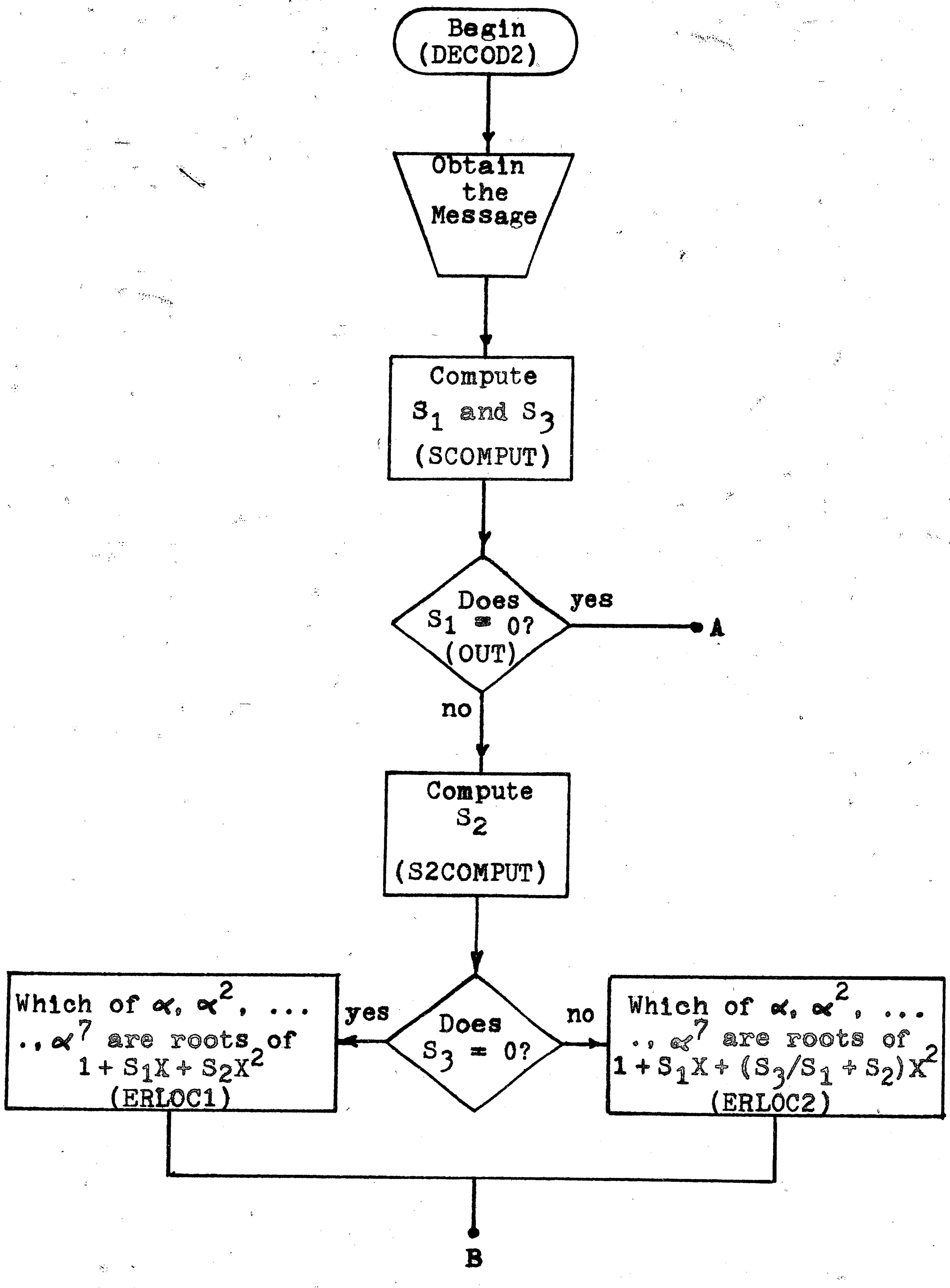
However, S_3 for this error pattern is

$$S_3 = e(\alpha^3) = \alpha^{12} + \alpha^3 + \alpha^6 = \alpha^7.$$

There are uncorrectable error patterns which, based upon the values of S_1 and S_3, appear to be correctable. In these instances correction is attempted. However, the decoded information is then compared with the transmitted information, which has been retained for checking purposes. Should a discrepancy exist, the decoder is charged with a failure to decode successfully.

While the preceding discussion outlines general procedures for the Berlekamp decoder, that which follows describes it with specific references to the program. Comments included in the program itself should be referred to for more detail. In addition, a flow chart for the Berlekamp decoder is shown in Figure 1. When a process shown on the flow chart is associated with a particular starting address, the latter is given in parenthesis near the bottom of the chart symbol for that process.

The Berlekamp decoder begins at address DECOD2 with the clearing of storage locations and initializing of counts. The first task is computation of S_1 and S_3. Subroutine SCOMPUT handles this, working in turn on the parity check and information portions of the received message. SCOMPUT uses tables TABLS1 and TABLS3 to selectively add terms 1 through α^{14} to form S_1, and terms 1, α^3, α^6, α^9, and α^{12} to form S_3, according to the non zero


Begin (DECOD2)
Obtain the Message
Compute S_1 and S_3 (SCOMPUT)
Does $S_1 = 0$? (OUT)
yes
A
no
Compute S_2 (S2COMPUT)
Does $S_3 = 0$?
yes
Which of $\alpha, \alpha^2, \ldots, \alpha^7$ are roots of $1 + S_1X + S_2X^2$ (ERLOC1)
no
Which of $\alpha, \alpha^2, \ldots, \alpha^7$ are roots of $1 + S_1X + (S_3/S_1 + S_2)X^2$ (ERLOC2)
B

Continued On Next Page

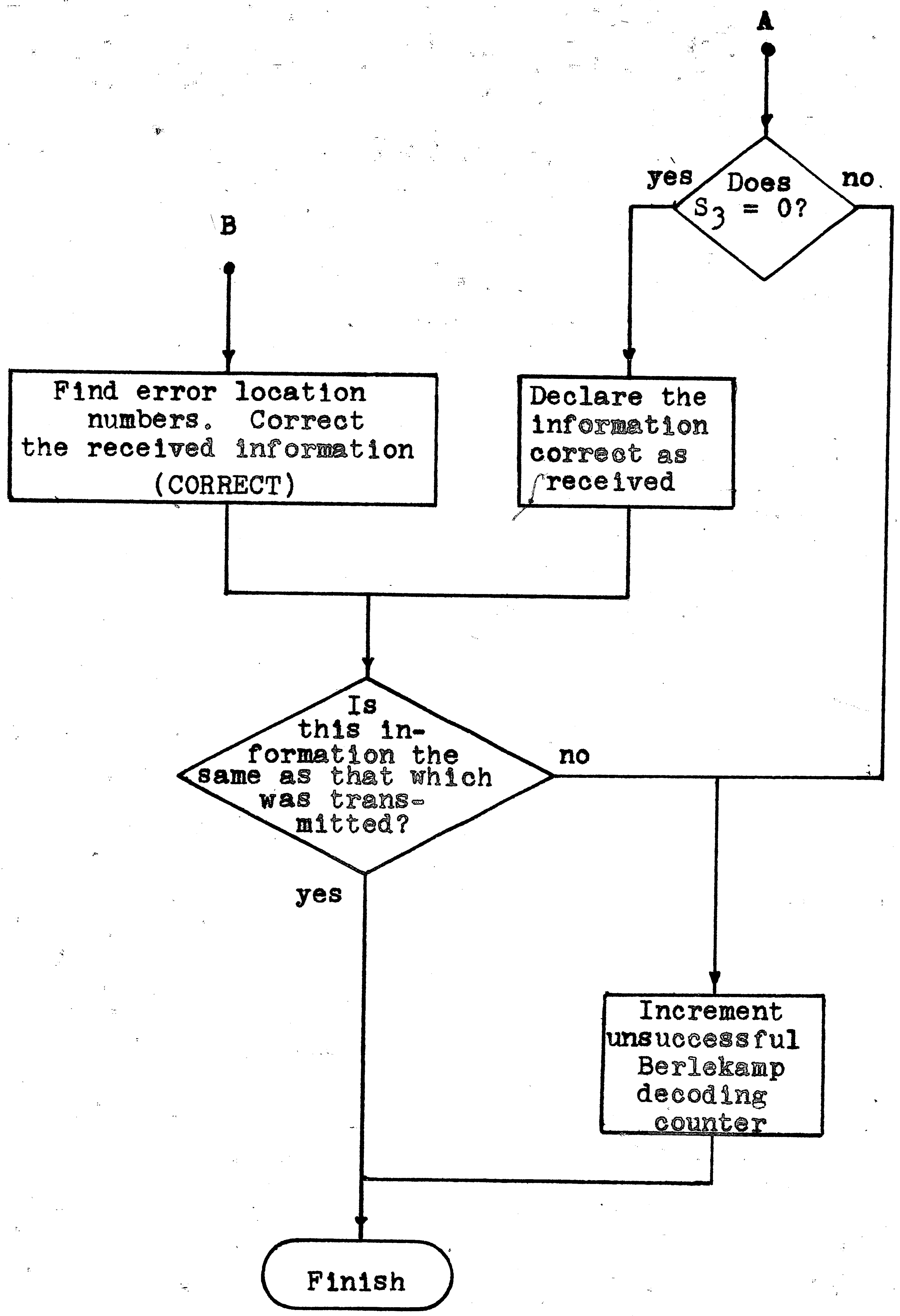

A
yes
Does $S_3 = 0$?
no
B
Find error location numbers. Correct the received information (CORRECT)
Declare the information correct as received
Is this information the same as that which was transmitted?
no
yes
Increment unsuccessful Berlekamp decoding counter
Finish

Figure 1. Berlekamp Decoder Flow Chart

bits of the message. This process is facilitated because the tables represent these terms as sums of the following: $1, \alpha, \alpha^2$, and α^3. This additive form for the terms from $GF(2^4)$ permits summing by a logical exclusive or subroutine, XOR.

Results of the addition are screened at address OUT. S_1 is tested first. If it is zero, S_3 is immediately tested. If S_3 is zero, the program branches to address HOME, where the information portion of the message is compared with the transmitted information. If S_3 is non zero, the counter which keeps track of unsuccessful decodings for the Berlekamp decoder is incremented.

If S_1 is non zero, S_2 is immediately computed, since it will be needed regardless of the value of S_3. This computation begins at location S2COMPUT. Entries in TABLS1 and TABLS3 contain, in addition to the additive representation already mentioned, a multiplicative representation of the terms from $GF(2^4)$. The latter is a binary number between zero and fourteen in the four most significant bits of each table entry. To determine S_2, TABLS1 is searched by subroutine SFORM to find the entry corresponding to the additive form of S_1. The multiplicative portion of this table entry is the output of SFORM. By doubling this number, which is actually an exponent, S_1 is squared. The result is processed to reduce it to its simplest form, a number between one and fourteen. This number is S_2 in multiplicative form, which is stored by the instruction at address CC.

S_3 is tested next. Depending on its value, the program branches either to ERLOC1 or ERLOC2, thereby selecting one of the two error location polynomial calculation routines. Each of the latter calls subroutine ERLOC, where $\alpha, \alpha^2, \alpha^3, \ldots\ldots, \alpha^7$ are substituted in turn for X in $S_1X + S_2X^2$. This is the entire polynomial in the case of ERLOC1. For ERLOC2, the remaining $(S_3/S_1)X^2$ term is computed separately and added to the output of ERLOC. When the result of any such computation is 1, an error location has been found, and the contents of FACTOR are set to 0001_8. Otherwise, FACTOR is reset.

The correction process is handled by subroutine CORRECT. Only the first seven message bits are corrected, since these alone constitute the information. Correction is done on a bit by bit basis, depending on the current value of FACTOR, which is recomputed for each bit. For example, recalling that the reciprocals of error location polynomial roots specify errors, α is substituted into the error location polynomial to ascertain whether the r_{14} bit is in error.

Following correction the program transfers to address HOME, where the corrected information is compared with that which was transmitted. If the correction process has left the result in error, the Berlekamp decoder is charged with failure to decode successfully.

MAJORITY LOGIC DECODING

THE MAJORITY LOGIC DECODING ALGORITHM

Early work in majority logic decoding was done by Reed(6), Muller(7), and Massey(8). Today this decoding technique is widely known and understood. It depends on the existence of a special set of parity check sum equations, written either in terms of the error vector bits or the received message vector bits. This set of equations must be orthogonal in the sense that each equation checks a set of bits different from those checked by the other equations, with the exception that one bit is checked by all the equations. Such a set of equations is said to be orthogonal on the bit which is checked by each equation. For a particular code, there must be at least twice as many equations in the set, numerically speaking, as the number of errors the code can correct. In the case of cyclic codes, such as the binary (15,7) BCH code, the existence of a set of equations orthogonal on any one bit ensures that such a set can be found for any other bit. This is due to the cyclic symmetry of such codes.

The binary (15,7) BCH code is the only one-step majority logic decodable BCH code known. This property permits decoding with a single majority gate, considerably simplifying the computer program required to implement the decoder. In addition, the majority logic decoding scheme using the received codeword, called a Type II decoder, was chosen for this work because it promised a less complicated implementation which would execute faster on the computer.

The Type II decoder relies on the fact that parity check sums of certain received bits are equivalent to check sums of error bits. Certain sets of the former can be found which are orthogonal on a given error bit. To illustrate this, let v, e, and r be n element transmitted, error, and received code vectors respectively. Let w be any vector in the row space of H, the parity check matrix. Then

$$rw = (v+e)w = vw + ew.$$

But $vw = 0$ because v is a codeword, so that

$$rw = ew$$

where rw is a parity check sum. The problem of finding a set of parity check sums orthogonal on e_{n-1} is now reduced to finding a set of vectors $w^{(i)}$, $i = 1, 2, \ldots J$, in the row space of H, such that the set of check sums $A_1, A_2, \ldots, A_J$ is orthogonal on r_{n-1}, where $A_i = rw^{(i)}$, and J for a one-step majority logic decodable code is numerically one less than the minimum distance. Thus, for the binary (15,7) BCH code, four check sums are required.

To find the $w^{(i)}$ for the binary (15,7) BCH code the parity check matrix H is needed:

$$H = \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \end{bmatrix} = \begin{bmatrix} 1&0&0&0&0&0&0&0&1&1&0&1&0&0&0 \\ 0&1&0&0&0&0&0&0&0&1&1&0&1&0&0 \\ 0&0&1&0&0&0&0&0&0&0&1&1&0&1&0 \\ 0&0&0&1&0&0&0&0&0&0&0&1&1&0&1 \\ 0&0&0&0&1&0&0&0&1&1&0&1&1&1&0 \\ 0&0&0&0&0&1&0&0&0&1&1&0&1&1&1 \\ 0&0&0&0&0&0&1&0&1&1&1&0&0&1&1 \\ 0&0&0&0&0&0&0&1&1&0&1&0&0&0&1 \end{bmatrix}$$

The $w^{(i)}$ can now be found by inspection:

$$\begin{aligned} w^{(1)} &= h_3 &&= (0,0,0,1,0,0,0,0,0,0,0,1,1,0,1) \\ w^{(2)} &= h_1 + h_5 &&= (0,1,0,0,0,1,0,0,0,0,0,0,0,1,1) \\ w^{(3)} &= h_0 + h_2 + h_6 &&= (1,0,1,0,0,0,1,0,0,0,0,0,0,0,1) \\ w^{(4)} &= h_7 &&= (0,0,0,0,0,0,0,1,1,0,1,0,0,0,1) \end{aligned}$$

These vectors specify the parity check sums of the received bits:

$$A_1 = r_3 + r_{11} + r_{12} + r_{14}$$

$$A_2 = r_1 + r_5 + r_{13} + r_{14}$$
$$A_3 = r_0 + r_2 + r_6 + r_{14}$$
$$A_4 = r_7 + r_8 + r_{10} + r_{14} .$$

This set of equations is orthogonal on r_{14}. Two errors in bits other than r_{14} can result in at most two equations summing to 1. If r_{14} is in error, and at most one other received bit is in error, then at least three of the equations will sum to 1. Thus, the decoding rule says r_{14} is in error only when a clear majority of the check sums sum to 1. This holds provided there are J/2 or fewer errors.

Figure 2 shows a Type II majority logic decoder for the binary (15,7) BCH code. Initially, gate 1 is on and gate 2 is off. The received codeword is loaded into the shift register so that bit r_{14} is in stage r_{14}, bit r_{13} is in stage r_{13}, and so forth. Gate 1 is then turned off and gate 2 turned on. The logical exclusive or of bit r_{14} and the majority gate output, which is the function of the check sum's sum described above, is the corrected version of r_{14}. This appears immediately at the output, and on the input line to stage r_0. The shift register is then shifted one place to the right. Because the code is cyclic, the shifted data should also be a codeword. In general, parity is rechecked by again solving the equations orthogonal on r_{14-j} for

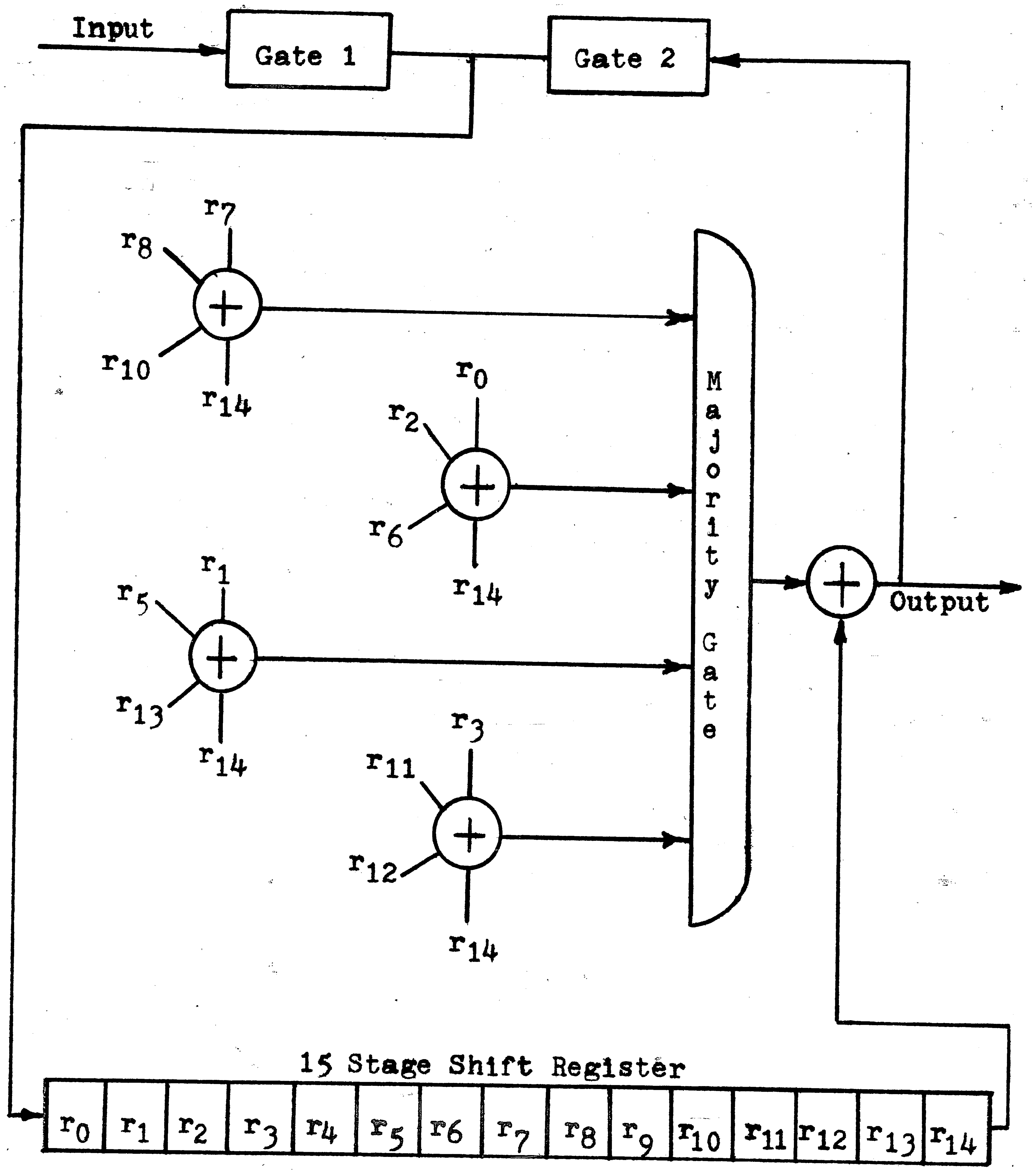


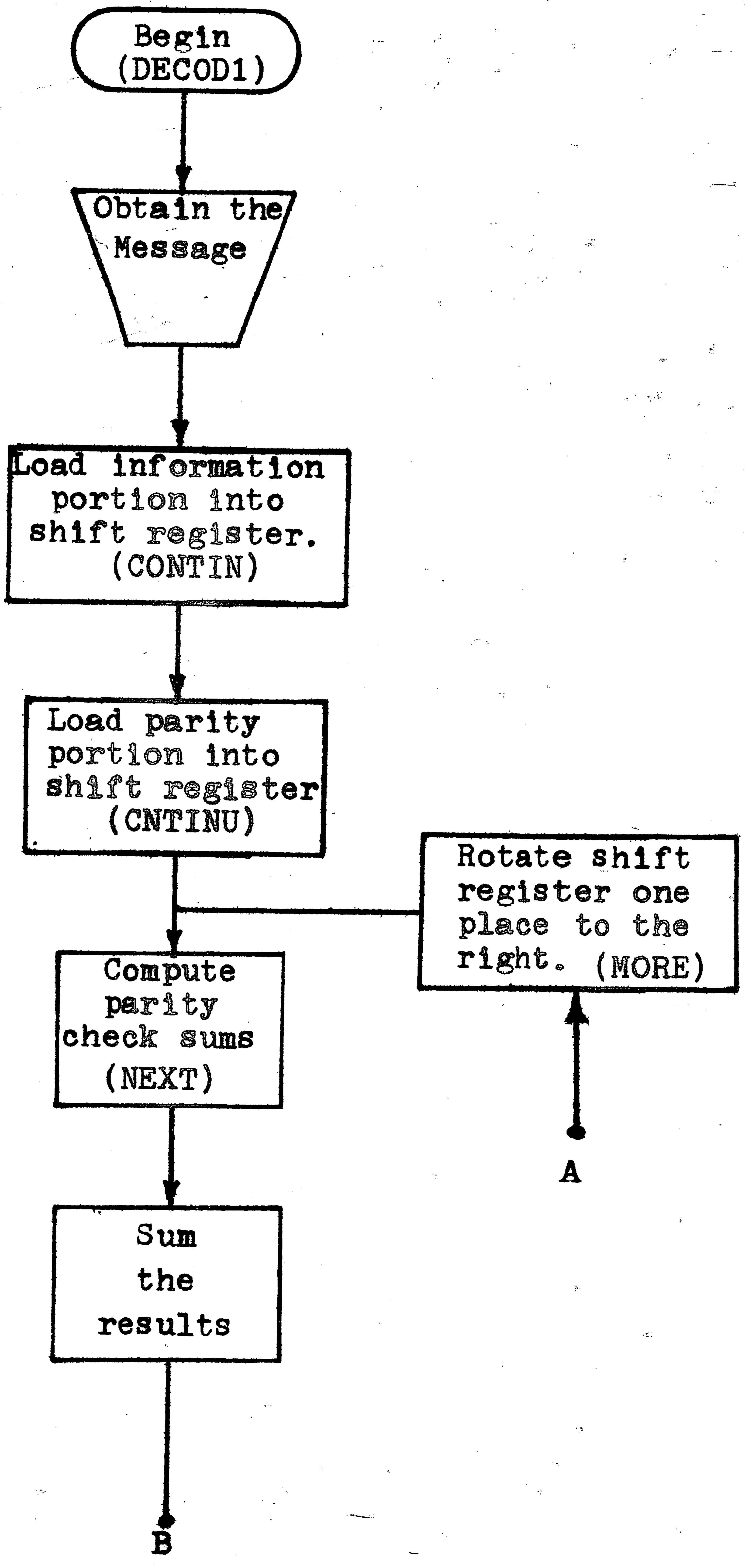
Figure 2. Majority Logic Decoder.

$j = 1, \ldots, (k-1)$. In the present case, $k = 7$, and the first output followed by the output data stream over six successive shifts is the corrected information.

THE MAJORITY LOGIC DECODER

The majority logic decoding algorithm described above was implemented on the PDP-8, beginning at address DECOD1. A flow chart for the program, with address references, is given in Figure 3. At the outset, counts and addresses are initialized. The shift register (actually sequential storage locations) is cleared. At address CONTIN, the information portion of the received message is loaded into the shift register. This is followed by the loading of the parity check portion at address CNTINU. The parity check sums are formed starting at address NEXT. The check sum totals are added together, and the result, a number between zero and four, is stored at address MAJ. When the process is complete, this number is tested. If it exceeds two, the message bit stored at address REG14 (which is r_{14}) is inverted. At address SHIFT a corrected version of the message is assembled, one bit at a time. The routine which shifts the shift register starts at address MORE. When all the information bits have been subjected to the correction process, the corrected information is compared with the transmitted information, which has been retained for checking purposes. If the transmitted information has not been recovered by the decoding process, the majority logic decoder is charged with a failure to decode successfully.

8


Begin
(DECOD1)
Obtain the Message
Load information portion into shift register. (CONTIN)
Load parity portion into shift register (CNTINU)
Rotate shift register one place to the right. (MORE)
Compute parity check sums (NEXT)
Sum the results
A
B

Continued On Next Page

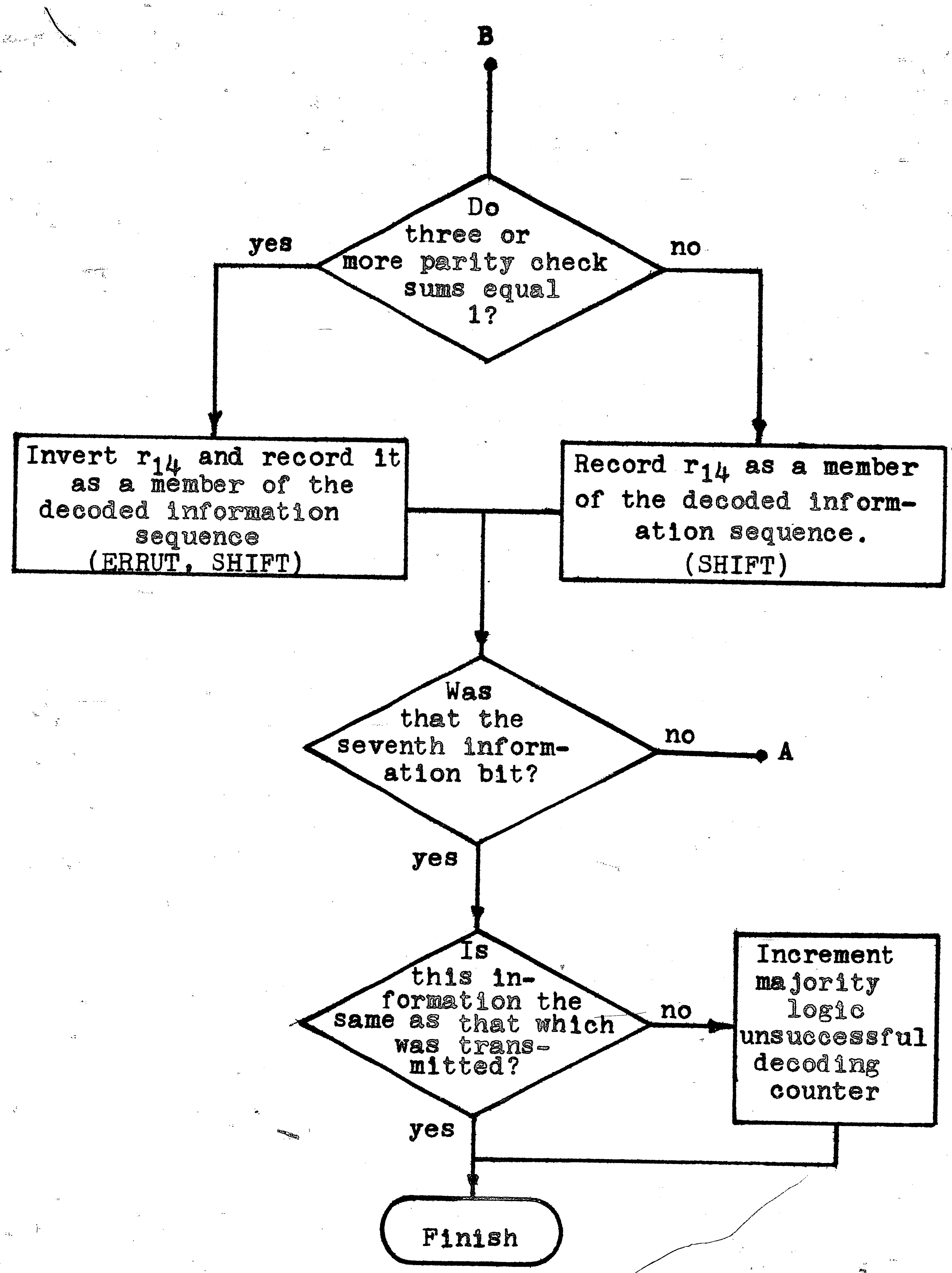

B
Do three or more parity check sums equal 1?
yes
no
Invert r_{14} and record it as a member of the decoded information sequence (ERRUT, SHIFT)
Record r_{14} as a member of the decoded information sequence. (SHIFT)
Was that the seventh information bit?
no
A
yes
Is this information the same as that which was transmitted?
no
Increment majority logic unsuccessful decoding counter
yes
Finish

Figure 3. Majority Logic Decoder Flow Chart

MAXIMUM LIKELIHOOD DECODING

THE MAXIMUM LIKELIHOOD DECODING ALGORITHM

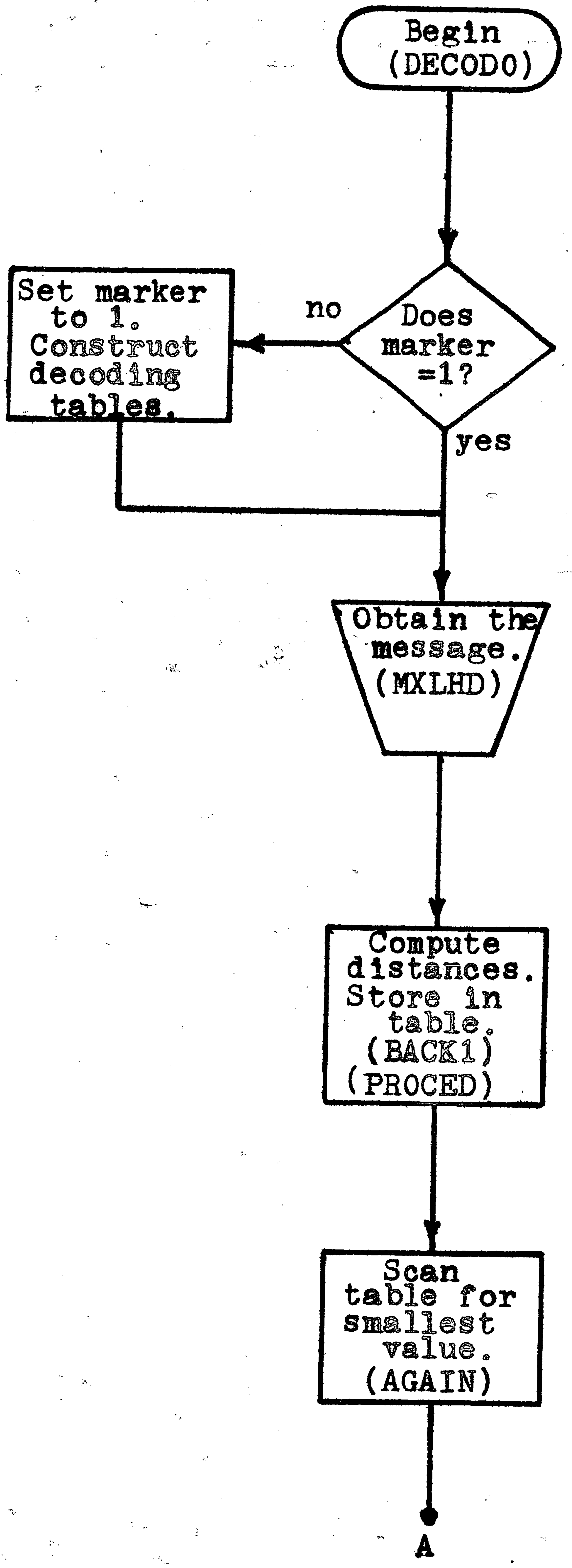
Maximum likelihood decoding is conceptually simple and straightforward. Compared with the deterministic techniques already discussed, maximum likelihood decoding is probabilistic in nature. It usually serves as a way to compare the performance of decoding algorithms more amenable to implementation than itself.

Maximum likelihood decoding requires computation, given the received sequence X, of the conditional probabilities $p(X/V_i)$, for each codeword V_i, where $i = 1, 2, \ldots, n$. V_j is selected as the transmitted codeword if $p(X/V_j) > p(X/V_i)$ for all $i \neq j$. If the probability of transmission is the same for each codeword, the decoding rule selects as the transmitted codeword the one which is least distant from the received message sequence. Thus, the Hamming distances between the received sequence X and all codewords V_i must be calculated. For example, in the case of the binary (15,7) BCH code which has $2^7 = 128$ codewords, 128 distances must be calculated.

THE MAXIMUM LIKELIHOOD DECODER

The maximum likelihood decoder is depicted in flow chart form in Figure 4. It begins at address DECODO. Initially, tables are established of each possible information sequence and its associated parity check bits. There are separate tables for information and parity because the word length of the PDP-8 is only twelve bits, not long enough to accomodate a fifteen bit codeword in a single memory location. A marker is set to indicate the tables are complete. When the maximum likelihood decoder is used the second (and succeeding) times, the marker causes a branch around the table setup routines to address MXLHD. This is possible, since the tables are used in a read-only mode.

At address MXLHD further initializing is done to prepare locations to receive the distance calculation results. Starting at BACK1 the distances are computed and stored. Actual computation is done by subroutine PROCED, the output of which is the Hamming distance between the received message and a particular codeword expressed as a binary number. At address AGAIN the table of distances just compiled is scanned for its smallest entry. This procedure is complicated by the possibility that there might not exist a unique smallest entry. Should this condition be detected, the program transfers


Begin (DECODO)
Does marker =1?
no
yes
Set marker to 1. Construct decoding tables.
Obtain the message. (MXLHD)
Compute distances. Store in table. (BACK1) (PROCED)
Scan table for smallest value. (AGAIN)
A

Continued On Next Page

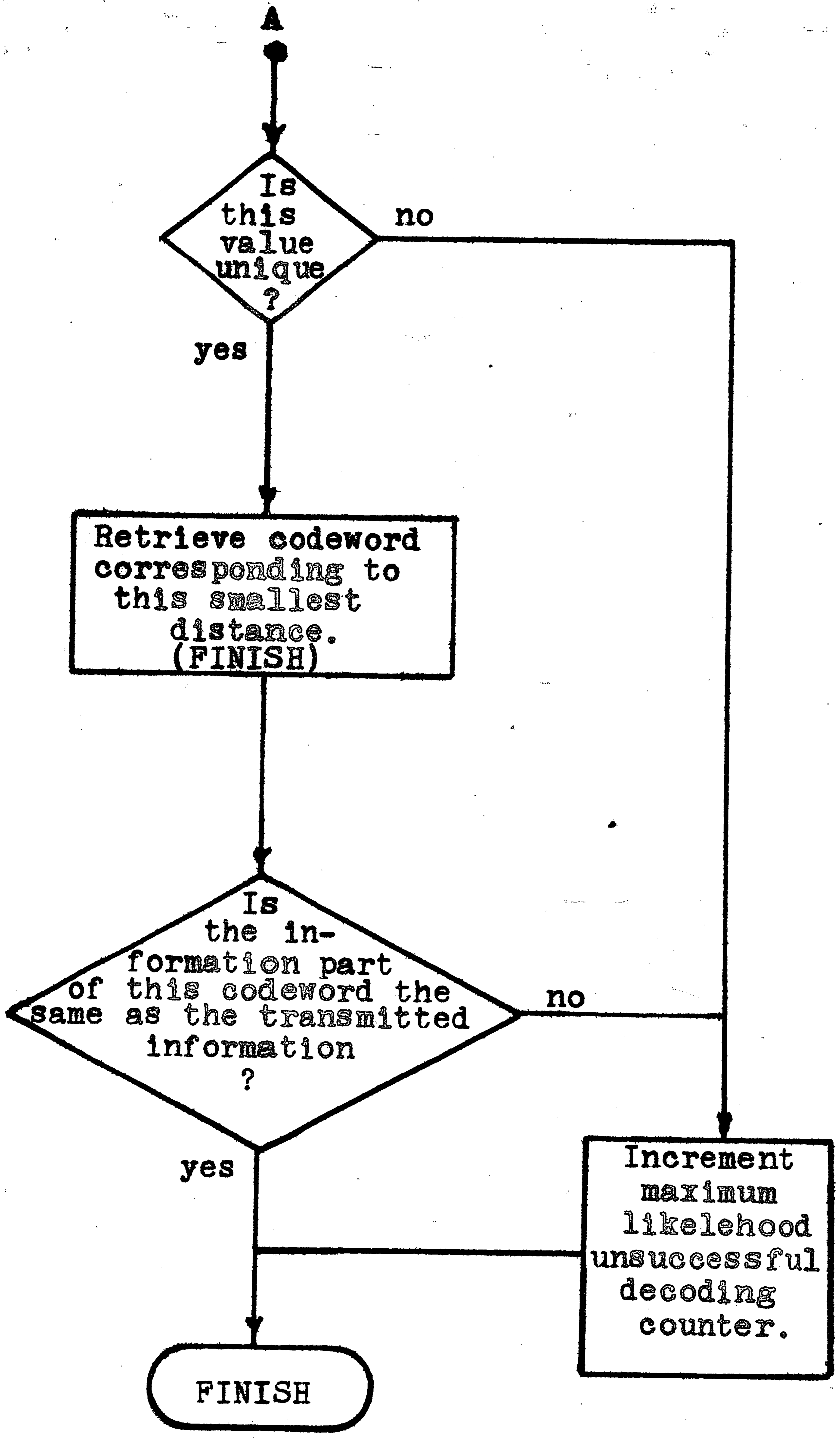

A
Is this value unique ?
no
yes
Retrieve codeword corresponding to this smallest distance. (FINISH)
Is the information part of this codeword the same as the transmitted information ?
no
yes
Increment maximum likelehood unsuccessful decoding counter.
FINISH

Figure 4. Maximum Likelihood Decoder Flow Chart

to address ERRBELL, where the maximum likelihood decoder is charged with failure to decode successfully. This is done because failure to find a unique smallest table entry means two or more codewords have equal probability.

If a unique smallest table entry is found during the search procedure, the information portion of the corresponding codeword is retrieved starting at address FINISH. This corrected information is compared with the transmitted information which has been retained for checking purposes. If a discrepancy is found, the maximum likelihood decoder is charged with failure to decode successfully.

THE SUPERVISORY PROGRAM

A means is needed to create and encode information sequences, create errors and inject them into the message sequences, and supervise the various decoding processes and record the results. These services are provided by the supervisory program, a flow chart for which is shown in Figure 6.

Information sequences are created by a seven bit maximum length shift register. This device, shown in Figure 5, produces all seven bit sequences pseudo-randomly, except for the all zero sequence. Repetition begins after 127 different information sequences. This

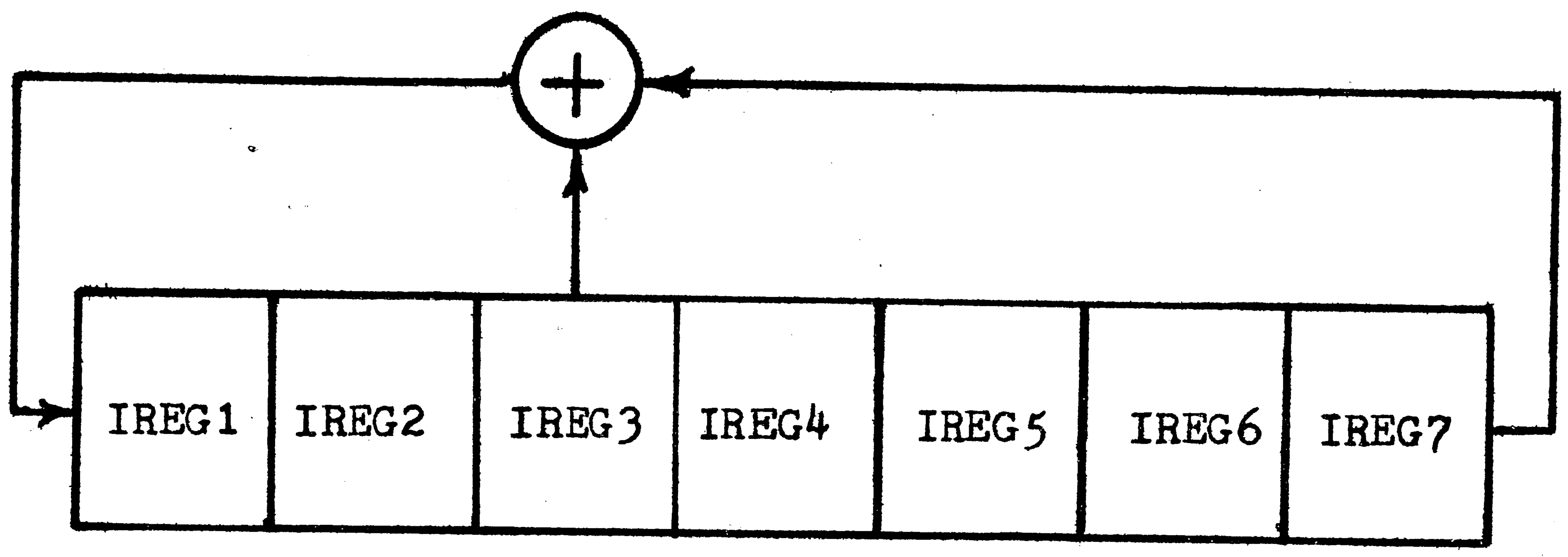
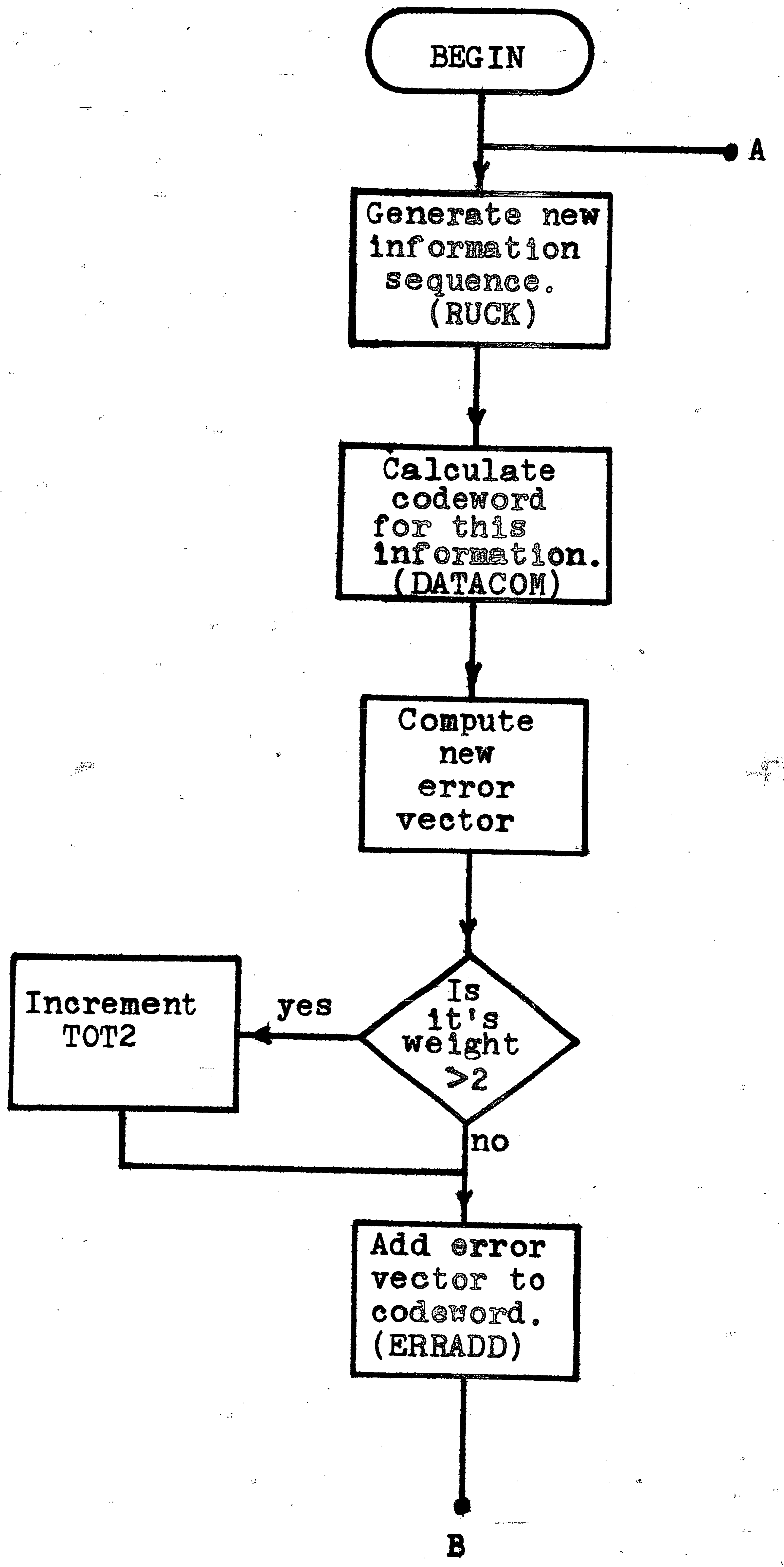


Figure 5. The Information Register

information generation process starts at address RUCK. The information register (IREG1 through IREG7) is initially loaded with all 1's. These always form the first information sequence. The information register is then


BEGIN
A
Generate new information sequence. (RUCK)
Calculate codeword for this information. (DATACOM)
Compute new error vector
Is it's weight >2
yes
Increment TOT2
no
Add error vector to codeword. (ERRADD)
B

Continued On Next Page

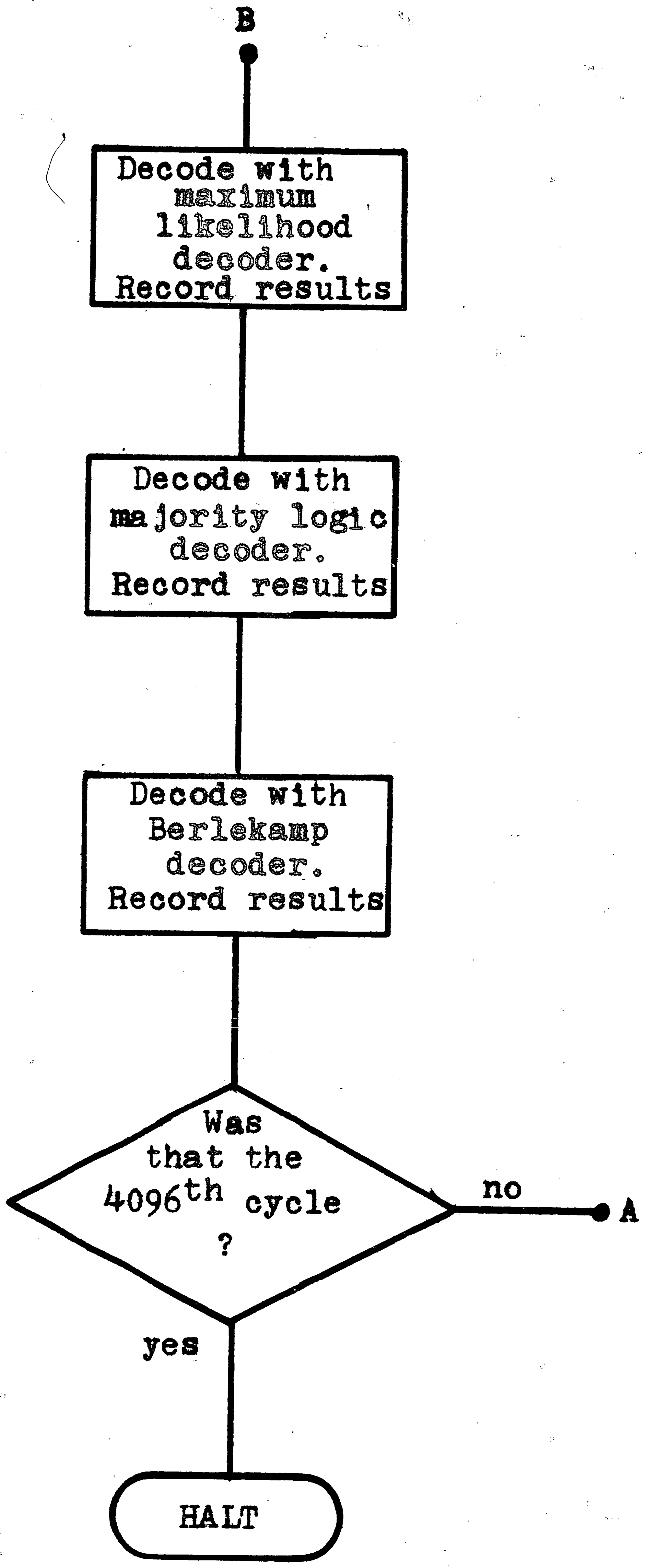

B
Decode with maximum likelihood decoder. Record results
Decode with majority logic decoder. Record results
Decode with Berlekamp decoder. Record results
Was that the 4096th cycle ?
no
A
yes
HALT

Figure 6. Supervisory Program Flow Chart

shifted, with outputs from stages three and seven added to form the input to the first stage. The feedback connections correspond to the primitive, irreducible polynomial $1 + X^3 + X^7$, which guarantees the maximum length properties of the shift register.(9)

Each information sequence is encoded in systematic form using a $k = 7$ stage shift register encoding circuit, which is connected according to the parity polynomial $h(X) = 1 + X^4 + X^6 + X^7$. Codewords are formed starting at address DATACOM.

Beginning with address ERRADD, each codeword is first stored in its original (or transmitted) form, and then corrupted. The corruption process relies on the use of six independent maximum length shift registers, each seventeen stages in length. The content of the last stage of each register is designated as its output. The probability of any register's output equalling 1 is very close to 1/2, differing only because no register ever contains all zeros. By logical anding together q such outputs, all of which are independent, a variable is obtained which has the value 1 with the probability $(1/2)^q$. For example, letting Y be the variable, for $q = 4$, we have

$$p(Y = 1) = (1/2)^4 = 1/16.$$

The number of outputs to be used, q, is set up in binary on the PDP-8 switch register before the program is started.

The technique just described allows generation of a random error with prespecified probability. The value of this random error (either 0 or 1) is recalculated for each bit in the codeword. The six maximum length shift registers are each shifted once prior to a calculation. This procedure is handled by subroutine ERCMPT, whose output is a seven bit error sequence for the information, and an eight bit error sequence for the parity portion of each codeword. These error sequences are added to the appropriate portion of the codeword by the XOR (logical exclusive or) subroutine. The resulting message, which now includes corruption, is then decoded by each of the three decoders.

It is important that errors be created on a random basis. This is achieved by cycling the six maximum length shift registers, which actually are pseudo-random generators, only once per data run. The correlation of a seventeen stage maximum length shift register used in this manner is essentially zero, differing by only one part in 131,070.(10) Earlier in the course of this work, shorter maximum length shift registers (having nine stages) were used, with several cycles per data run.

This effort, though unsuccessful, created awareness of the problem of initial bias which tends to interfere with randomness.

Initial bias is most easily seen when two or more registers start with identical contents. This affects the result when outputs are logical anded together because the outputs of two or more registers will remain identical despite shifting. Every effort is made, of course, to randomize the initial loading of the six error registers, and no registers begin with the same contents. However, a problem remains.

The seventeen stage maximum length shift registers each have feedback from the third and seventeenth stages (according to the primitive, irreducible polynomial $1+X^3+X^{17}$) added to form the input for the first stage. This means the output values for the first fourteen shifts are solely determined by the initial contents of the registers. To the extent the initial loading is not random, a bias is introduced. The effect of this bias is minimized by not recycling the set of shift registers. In addition, more than one set of initial shift register contents is used, and the results are averaged.

The initial contents of the six error registers are loaded from the same paper tape used to load the program, and the information generating register always begins with all 1's. By thus fixing the initial conditions it is possible to repeat data runs, thereby checking the results.

The supervisory program keeps records of unsuccessful decoding efforts for each decoder. These results are stored at addresses ERRST0 through ERRST2. Another portion of the supervisory program determines and records the total number of error vectors whose weight exceeds two. This routine begins at address AZURE1. It does not affect the encoder or decoders, and serves only as a monitor.

RESULTS AND INTERPRETATION

EXPERIMENTAL DETERMINATION OF DECODING ABILITY

The binary (15,7) BCH code is guaranteed to correct two errors in a fifteen bit message block. This is the minimum ability of the code. An indication of the binary (15,7) BCH code's actual ability to correct errors is presented here.

The total number of failures to decode successfully is made available by the program for each decoder at the conclusion of a data run. An estimate of the expected number of decoding failures is needed to check the validity of these results. To this end, it is possible to calculate the expected number of decoding failures by making the assumption that addition to a codeword of any error vector having weight exceeding two will cause such a failure. This is reasonable, since the code is guaranteed to correct only two or fewer errors per block.

The addition of errors to codewords, in the manner previously discussed, simulates the action of a binary symmetric channel with crossover probability $p_e = (1/2)^q$ (recall that q is specified by the program user). The expected number of decoding failures is a function of p_e. For example, let $p_e = 1/64$, and let p_t be the probability that within a block of fifteen bits, three or more will be in error. Then

$$p_t = 1 - (p(0) + p(1) + p(2))$$

where p(0), p(1), and p(2) are the probabilities that zero, one, and two bits respectively will be in error within a fifteen bit block. These probabilities are easily calculated:

$$p(0) = \binom{15}{0}(63/64)^{15} = 0.7896021$$

$$p(1) = \binom{15}{1}(1/64)(63/64)^{14} = 0.1880005$$

$$p(2) = \binom{15}{2}(1/64)^{2}(63/64)^{13} = 0.0208888$$

so that

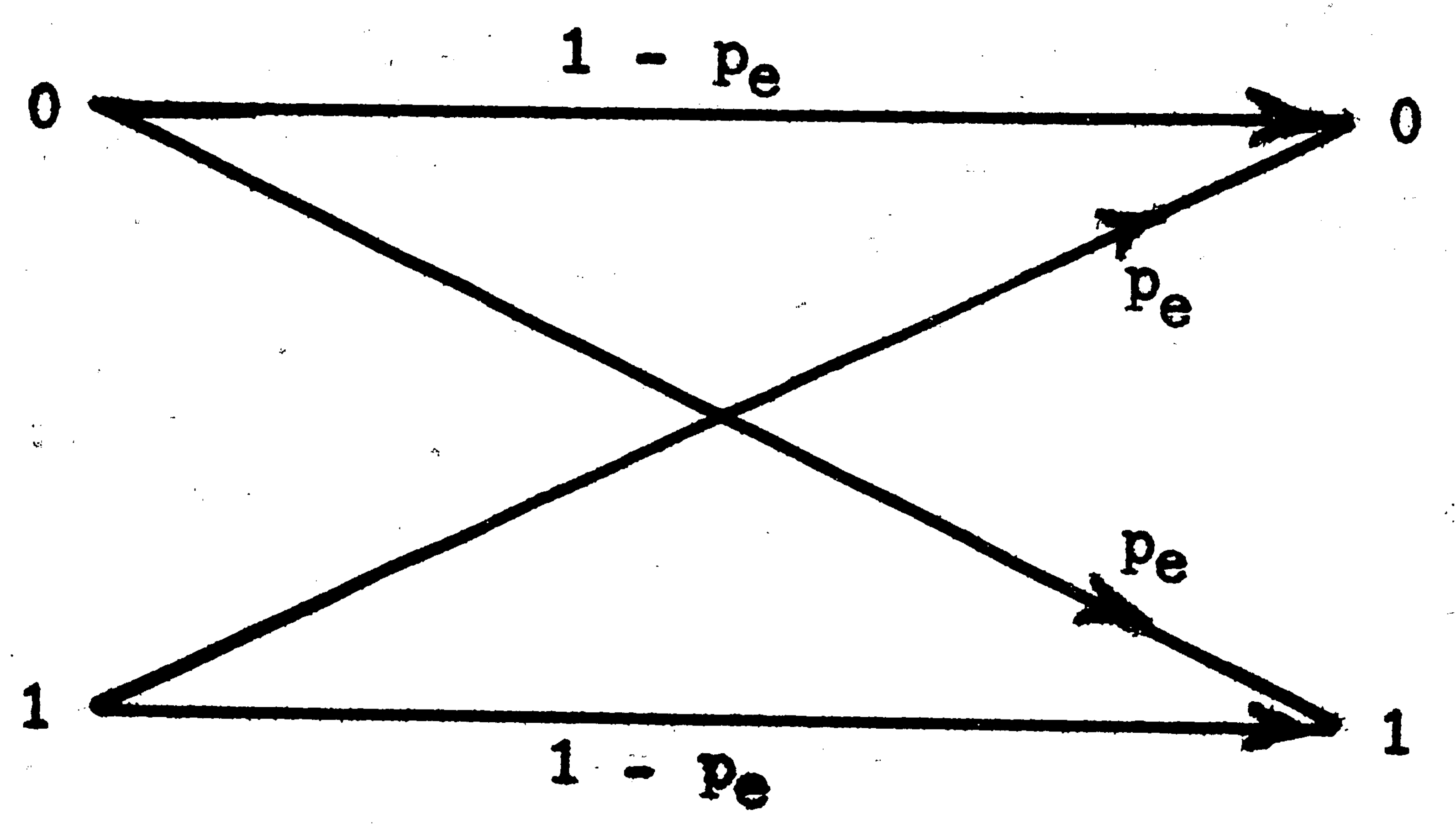
$$p_t = 0.0015086.$$

The number of codewords processed per data run was chosen to ensure against recycling of the error registers. The seventeen stage maximum length shift registers permit $2^{17} - 1$ or 131,071 shifts per cycle. Each shift specifies the error input for one codeword bit. Since

$$(131{,}071/15) = 8738.066,$$

there are 8738 fifteen bit codewords per data run. The expected number of error vectors having weight exceeding two during a data run of this length is 8738(p_t). For $p_e = 1/64$, 8738(0.0015086) or about 13.2 such error vectors can be expected. This number varies, of course, with changes in the crossover probability. The right-hand column of Table 1 shows results of such calculations based on the various crossover probability values for which data was obtained. Since the code is guaranteed to correct only two or fewer errors per block, these results specify an upper bound on the expected number of decoding failures for each crossover probability. This serves as a valuable guide to the credibility of the data obtained experimentally.

The program analyzes each error vector as it is formed. Those vectors having weight exceeding two are counted. It is possible, therefore, to compare minimum and actual performance of the decoders straightaway. This comparitive data is given in Table 1, above which the binary symmetric channel is depicted. The values tabulated are averages from three data runs, each of which was made with different initial error register contents. Figure 7 displays this same comparitive data graphically. The probability of decoded error plotted on the ordinate is the ratio of unsuccessful decodings to the total number of messages decoded. The probability that transmission over the channel will occur without



The binary symmetric channel

	Unsuccessful Decoding Attempts			Error Vector Data	
p_e	Maximum Likelihood Decoder	Majority Logic Decoder	Berlekamp Decoder	Error Vectors Having Weight ≥ 3	Expected Error Vectors With Weight ≥ 3
1/64	7.7	7.3	9.3	9.7	13.2
1/32	64.0	74.7	84.3	88.7	91.6
1/16	411.0	468.3	516.0	544.7	551.6
1/8	2023.3	2226.7	2379.7	2496.3	2505.9

Table 1. Experimental Results and Expected Error Vectors With Weight ≥ 3.

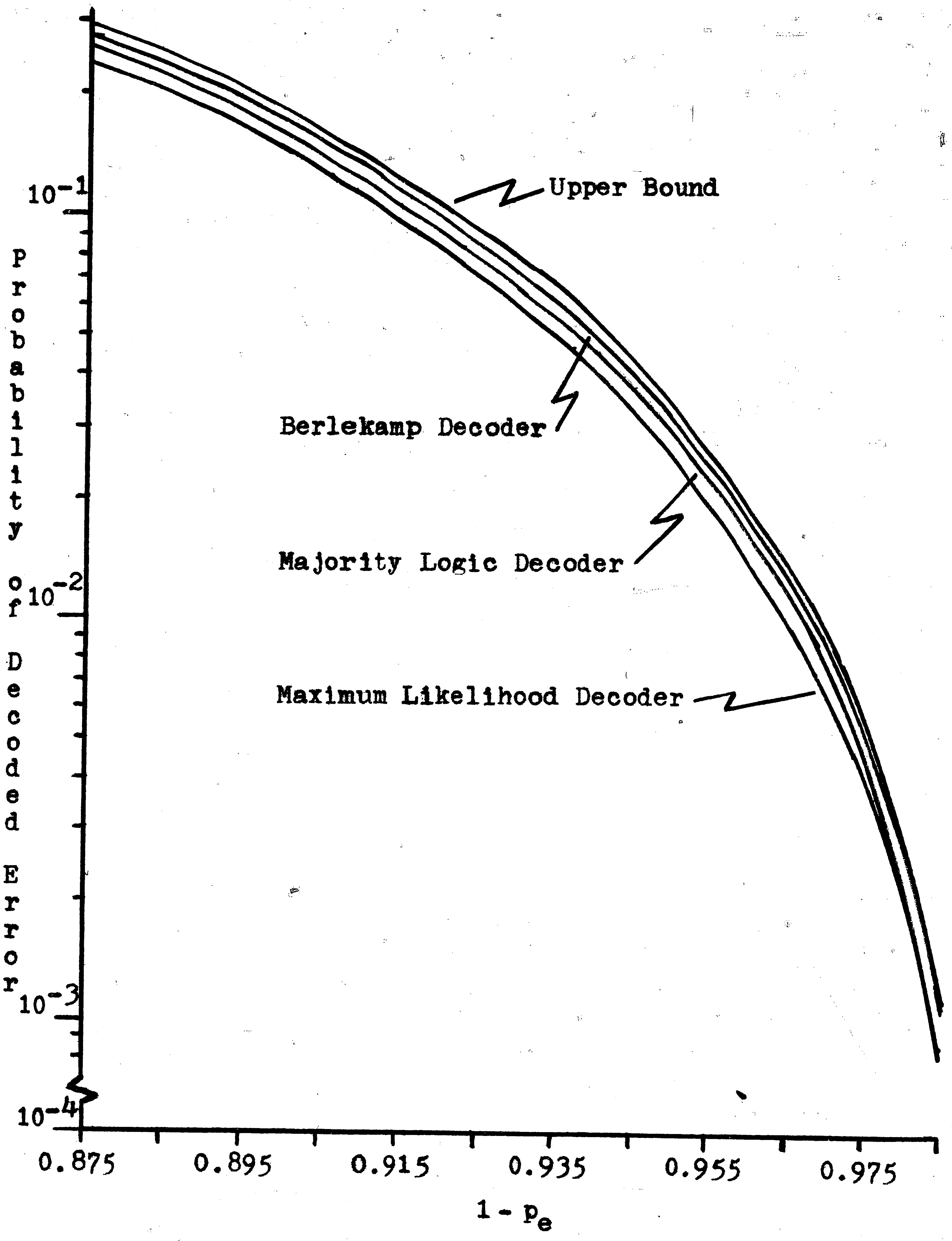


Figure 7. Probability of Decoded Error vs. $1 - p_e$.

error, $1 - p_e$, for each bit is plotted on the abscissa.

Referring to Table 1, in all instances the decoders were successful in recovering the information from some messages having more than two errors. With crossover probabilities of 1/32 and 1/16, for example, the maximum likelihood decoder recovered the information in more than 23% of the messages having three or more errors. Even the Berlekamp decoder recovered the information in at least 5% of the messages under the same conditions, and recovery by the majority logic decoder exceeded 13%. Thus, when viewed from a statistical standpoint, the ability of the code to correct errors is a function both of its structure and the decoding algorithm employed.

The enhanced error correcting ability of the maximum likelihood decoder is attributable to the fact that not all codewords have only minimum distance separation from all others. In some of these cases, the probability $p(X/V_i)$ computed by the maximum likelihood decoder will still be greatest for the correct sequence V_i, despite the presence of three or more errors in the message. The decoder will correctly select the transmitted codeword in these cases.

Those elements which contribute to the majority logic decoder's ability to correct some patterns of three or more errors are more veiled. Recall first the choice of vectors $w^{(i)}$ from the row space of parity check matrix H, such that $rw^{(i)} = ew^{(i)}$ for $i = 1, 2, \ldots, J$. This shows the parity check sums A_i used in the majority logic decoding process, written previously in terms of the received vector, can also be expressed in terms of the error vectors. Thus

$$\begin{aligned} A_1 &= e_3 + e_{11} + e_{12} + e_{14} \\ A_2 &= e_1 + e_5 + e_{13} + e_{14} \\ A_3 &= e_0 + e_2 + e_6 + e_{14} \\ A_4 &= e_7 + e_8 + e_{10} + e_{14} \end{aligned}$$

Notice that errors in positions e_4 and e_9 do not enter into the process of check sum computation for the correction of bit 14. In other words, the process is blind to errors in these positions. Additionally, certain pairs of errors are self canceling. For example, if e_0 and e_2 are simultaneously 1 (and e_6 and e_{14} are 0), check sum A_3 still sums to zero.

The two mechanisms just outlined acting in concert can cause the majority logic decoder to render a successful decoding even if more than two errors exist. Consider, for example, the following three error situation: Suppose $e^{(0)}$, the unshifted error vector, is given by

$$e^{(0)} = (0,1,1,0,0,1,0,0,0,0,0,0,0,0,0).$$

Because bits e_5 and e_1 are 1, A_2 is 0. In fact, only A_3 sums to 1 in this case (this is due to e_2). Since the majority result is zero, no change is made to e_{14}. Next the vector is shifted once to obtain $e^{(1)}$. The check sum results for this and subsequent shifts are given below. Only check sum totals of 1 are listed:

$e^{(1)} = (0,0,1,1,0,0,1,0,0,0,0,0,0,0,0)$, $A_1 = 1$

$e^{(2)} = (0,0,0,1,1,0,0,1,0,0,0,0,0,0,0)$, $A_1 = 1$, $A_4 = 1$

$e^{(3)} = (0,0,0,0,1,1,0,0,1,0,0,0,0,0,0)$, $A_2 = 1$, $A_4 = 1$

$e^{(4)} = (0,0,0,0,0,1,1,0,0,1,0,0,0,0,0)$, $A_2 = 1$, $A_3 = 1$

$e^{(5)} = (0,0,0,0,0,0,1,1,0,0,1,0,0,0,0)$, $A_3 = 1$

$e^{(6)} = (0,0,0,0,0,0,0,1,1,0,0,1,0,0,0)$, $A_1 = 1$

The actions of error cancellation (in $e^{(0)}$, $e^{(1)}$, $e^{(5)}$, and $e^{(6)}$) and error blindness (in $e^{(2)}$, $e^{(3)}$, and $e^{(4)}$) have combined in such a way that in no case do three or more of the check sums sum to one. Because of this, none of the information bits is altered. Since the information portion of the message was without error to begin with (refer to $e^{(0)}$), the majority logic decoder is credited with a successful decoding.

It is more difficult in the case of the Berlekamp decoding algorithm to identify specific mechanisms which promote correct decoding in the presence of more than two errors. However, as shown by the experimental results, there are circumstances where three or more errors can exist without disruption of the decoding process. The following example illustrates this point.

Consider transmitted, error, and received vectors given respectively by the following:

$$v = (0,1,1,0,0,1,1,1,0,0,1,0,0,0,0)$$
$$e = (1,1,0,0,0,0,0,1,0,0,0,0,0,0,0)$$
$$r = (1,0,1,0,0,1,1,0,0,0,1,0,0,0,0)$$

From the received vector S_1 and S_3 can be found as follows:

$$S_1 = 1+\alpha^2+\alpha^5+\alpha^6+\alpha^{10} = \alpha^3$$
$$S_3 = 1+\alpha^6+ \ 1+\alpha^3+ \ 1 = \alpha^8$$

Based on these values, $(S_3/S_1 + S_2) = \alpha^9$. Tabulated below are the results found by substituting α through α^7 for X in the error location polynomial:

X	$S_1X + (S_3/S_1 + S_2)X^2$
α	$\alpha^4 + \alpha^{11} \neq 1$
α^2	$\alpha^5 + \alpha^{13} \neq 1$
α^3	$\alpha^6 + 1 \neq 1$
α^4	$\alpha^7 + \alpha^2 \neq 1$
α^5	$\alpha^8 + \alpha^4 \neq 1$
α^6	$\alpha^9 + \alpha^6 \neq 1$
α^7	$\alpha^{10} + \alpha^8 \neq 1$

Because $(S_1X + (S_3/S_1 + S_2)X^2) \neq 1$ for $X = \alpha, \alpha^2, \ldots, \alpha^7$, no errors are specified in the information bit positions, α^{14} through α^8, which are checked by substituting these values for X. However, since these bits were transmitted error free (refer to the error vector above) the Berlekamp decoder is credited with a successful decoding.

DECODING SPEED

The time required for each decoder to process 4,096 codewords was measured. The results, presented in Table 2, serve as a guide to the relative practical merits of each decoding scheme. In each case, program optimization could reduce the running time slightly. In addition, fairness requires mention of the considerable penalty imposed on the maximum likelihood decoder by the twelve bit word length of the PDP-8, although this was an impediment for the other decoders as well. In each case p_e was 1/16, causing roughly one error per message, on the average.

Decoder	Time to Decode 4096 Blocks	Decoding Rate (information bits/sec)
Maximum Likelihood	157 sec.	182.6
Majority Logic	6.5 sec.	4411.1
Berlekamp	18 sec.	1592.9

Table 2. Decoding Speed.

SUMMARY AND CONCLUSIONS

This work demonstrated the ability of the binary (15,7) BCH code to correct in excess of two errors per message. This ability was seen to be related to the decoding algorithm used to recover the information. Although it performed best, the maximum likelihood decoder was much less efficient in terms of running time than either of its competitors. The Berlekamp decoder was fast, though least capable of recovering information from messages having three or more errors. The majority logic decoder was impressive in recovery of information, and in addition had the best performance in terms of running time.

With the advent of commercially available microprocessors, results such as those obtained here could be important in the selection of codes and decoding schemes to be implemented by these devices in a wide variety of practical applications. Microprocessors, after all, are much like the minicomputer used in this work, the key differences being slower speed (on the bad side) and vastly reduced size and cost (on the good side). Based on the results obtained in this work, for example, the majority logic decoder for the binary (15,7) BCH code would be the best choice in a microprocessor system.

It decodes faster while requiring less storage space than either of the other decoders. This is important because the microprocessor program might be stored in a read only memory of limited size.

The fact that this work treated just one code makes generalization difficult. The results, however, suggest the possible existence of codes whose performances viewed from a statistical standpoint may differ significantly from the minimum performances predicted by algebraic coding theory. To the extent these differences may be advantageously utilized in an engineering situation, they are important. At the least, there should be an awareness of their existence.

The considerable time required to write and debug the computer program used in this work, which ultimately permitted examination of but a single code, suggests the need for analytical techniques to facilitate similar inquiries into other codes and decoding methods. Such techniques would permit more accurate statistical characterizations of specific code-decoder systems than those now readily derivable.

REFERENCES

(1) Lin, Shu, An Introduction to Error Correcting Codes, Prentice Hall, 1970, p. 125.

(2) Hocquenghem, A., "Codes Correcteurs D'erreurs", Chiffres, 2, 1959, pp. 147-156.

(3) Bose, R. C., and Ray-Chaudhuri, D. K., "On a Class of Error Correcting Binary Group Codes", Information and Control, 3, March 1960, pp. 68-79.

(4) Op. Cit., Lin, p. 120.

(5) Op. Cit., Lin, pp. 20-21.

(6) Reed, I. S., "A Class of Multiple-Error-Correcting Codes and the Decoding Scheme", IRE Transactions, IT-4, September 1954, pp. 38-49.

(7) Muller, D. E., "Applications of Boolean Algebra to Switching Circuit Design and Error Detection", IRE Transactions, EC-3, September 1954, pp. 6-12.

(8) Massey, J. L., Threshold Decoding, The M. I. T. Press, Cambridge, Massachusetts, 1963.

(9) Davies, W. T., "Generation and Properties of Maximum Length Sequences", Control, July 1966, p. 364.

(10) Op. Cit., Davies, August 1966, p. 431.

APPENDIX

VITAE

Dwight W. Aten, son of Helen E. Roberts Aten and Dwight C. Aten, was born June 29, 1944 at Garden City, New York. He is a native of Washington, New Jersey, where as a high school student in 1961 he was selected to attend the National Youth Conference on the Atom, in Chicago. Mr. Aten graduated in 1967 from Rensselaer Polytechnic Institute, Troy, New York, with the Bachelor of Electrical Engineering degree. He joined the Singer Company at their Research and Advanced Development Laboratory, where he worked in the areas of digital control systems, threshold logic, and computer simulation of permanent magnet motors. Later, he worked for Van Dyk Research Corporation, where as Supervisory Electrical Engineer he managed the company's efforts in matters pertaining to the electrical and electronic systems of a large, high speed copier/duplicator. Mr. Aten holds two patents in his field and is a member of Eta Kappa Nu.