P. R. Daher

# Automatic Correction of Multiple Errors Originating in a Computer Memory

**Abstract: An error correction unit has been installed in an IBM RAMAC® 305 to demonstrate automatic correction of burst errors originating in a computer memory. Employing a cyclic code requiring two percent redundancy, the unit has successfully corrected all four-bit and most five-bit error bursts in a 100-character record stored on a defective disk. Details of the error correction unit and test results are presented; two special cases involving block length are also discussed.**

## Introduction

In reading information from a computer memory such as a disk file, errors may be caused by dust particles, minute scratches, or defects in the oxide coating. One solution to this problem is suggested by the fact that such errors occur in bursts and lend themselves to automatic correction by cyclic error-correcting codes.[1] A simple device was constructed which employs a code requiring two-percent redundancy to correct all four-bit and most five-bit error bursts (that is, in a string of five bits, the first bit and any combination of the four bits remaining are in error). Evaluation of this device has shown that it is feasible to equip certain computer memories to perform corrections without need for additional storage or programming steps or for interruption of machine timing or data flow.

In the following sections some essential features of the computer and of error correction are presented summarily and the error correction unit and test results are discussed more fully. Details of system logic and two special cases of block length are also considered.

## The computer

Since error correction requires data storage during a decoding operation, a computer having a buffer between its memory and output is an ideal system in which to incorporate a correction device. In the IBM RAMAC 305[2,3] chosen for this purpose data enters and leaves the computer via a drum; the information to be stored is transferred from the drum to a buffer and subsequently written in the disk file. Readout progresses in reverse order. A stored message may contain up to 100 eight-bit characters, of which six bits per character are information; one other, the "start" bit, is used for clock synchronization, and another, the "parity" bit, is used for error detection. If an even number of ONES is counted during a parity check, a machine shutdown occurs.

## Error correction

A convenient approach to the correction of burst errors employs a sequence generator (feedback shift register) implementation[1,4,5,6] of a suitable cyclic code.[4-8] Some essential steps are summarized here to allow comparisons with the special cases that follow. Typically, a shift register, initially reset to ZERO, is used as an encoder by adding modulo 2 (EXCLUSIVE OR) its feedback, as determined by the code, to the data being transmitted, thus developing a data-dependent input to the register. When the last data bit has been transmitted, the feedback is diverted to the output line to provide a number of redundant bits cor-

responding to the length of the register. As a decoder, the register would operate similarly while the received data are being stored. The redundant bits are also processed, then the data line is removed from the register input. If the shift register is filled with ZEROS at the end of the decoding cycle, no error (or no detectable error) has occurred and no correction will be made. Otherwise, some sequence will be generated in the register during the correction cycle while the data is being read from storage. When the register contains $000 \cdots 00XXX \cdots X$, its feedback is diverted and used to invert the message bits leaving storage. (The $X$'s may be either ZEROS or ONES; their number is equal to the number of bits in the largest correctable burst.) If the length of the encoded message is equal to that of the code cycle, the bits inverted will be those which are in error (unless the burst exceeds the capability of the system).

Since correcting errors on time requires that the length of the coded message be equal to that of the code cycle, one approach for the special case of messages having a block length shorter than code cycle length would be to delay by an appropriate amount the data being read from storage during the correction cycle; another approach would be to fill the message with ZEROS to bring it to the necessary length. Unless the message and cycle lengths are reasonably close, however, the delay introduced by both these methods would be objectionable.

A third alternative is to correct the message from the end rather than from the beginning. Under this approach, the encoder and the encoding cycle are, with one exception, the same as for the case where message length is equal to the code cycle length; the exception is that some means is required to signal the end of the data block, at which time the shift register feedback is diverted to the output so that the redundancy will immediately follow the last data bit of the message. The decoding cycle is also unchanged except for the requirement of an end of message signal. After the last bit of the encoded message is shifted into decoder storage, the shift register feedback is changed and the register shifts in reverse while the data is simultaneously read from storage in reverse. When $XXX \cdots XX000 \cdots$ appears, the feedback is diverted from the register, as before, and used to invert the bits in error being shifted out of storage. The new feedback connection for this altered correction cycle is determined by the reciprocal code[5] to that used for encoding and decoding.

A fourth alternative, suitable for fixed length messages, does not require correcting from the end of a message. Basic to this approach is the fact that in general all binary combinations, modulo 2, of the last $b$ characters in the sequence beginning with $1000 \cdots$ (generated by the register) correct the corresponding $b$-bit burst patterns. For example, the "correcting character" to be recognized by the logic for a double adjacent pattern is derived by adding, modulo 2, the last two characters of this sequence. For a given code, each correctable error is associated with a unique sequence and the location of that error in the message determines which character of the sequence will appear in the register at the completion of the decoding cycle. During the correction cycle, when the error appears at the storage output, the sequence, having started with this character, will have progressed to the correcting character. If more than one correcting character occurs in the same sequence, their associated errors may be confused.

If a message of length $m$ is shorter than the code cycle length, only $m$ steps of the cycle are necessary. By deriving a new set of correcting characters, from the $m^{th}$ character of the sequence beginning with $1000 \cdots$, in all combinations with the previous $b - 1$ characters, a logic network can be constructed that will allow corrections to be made without delays. Further, this approach allows flexibility in exploiting the fact that the redundancy existing in a given code exceeds that required to correct the errors for which it was designed. It is this approach which has been implemented in the system to be described in this paper.

## The computer with error correction

The error correction unit is comprised of encoding, decoding, correcting, and control circuits located between the buffer and file, as shown in Fig. 1. In this location this unit does not affect machine timing but performs its operations during normal data transfers. The encoding cycle takes place while computer data is being recorded in the disk file and the decoding cycle occurs while data from the file is being stored in the buffer. The correction cycle then proceeds under control of the unit during the transfer of data from the buffer to the drum.

The error correction unit is shown in Fig. 2 and the relevant logic diagrams are presented and discussed in the Appendix. The heart of the error correction unit is the feedback shift register implementation[1,4,5,6] shown in

*Figure 1* **Location of error correction unit within RAMAC system.**

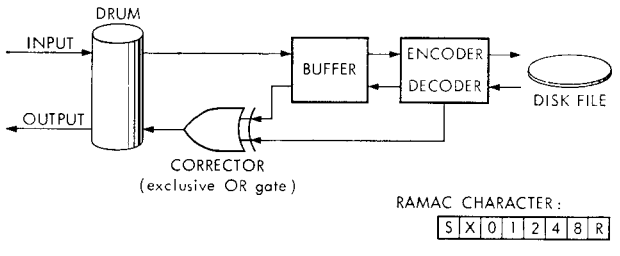


*Figure 2* **Error correction unit.**

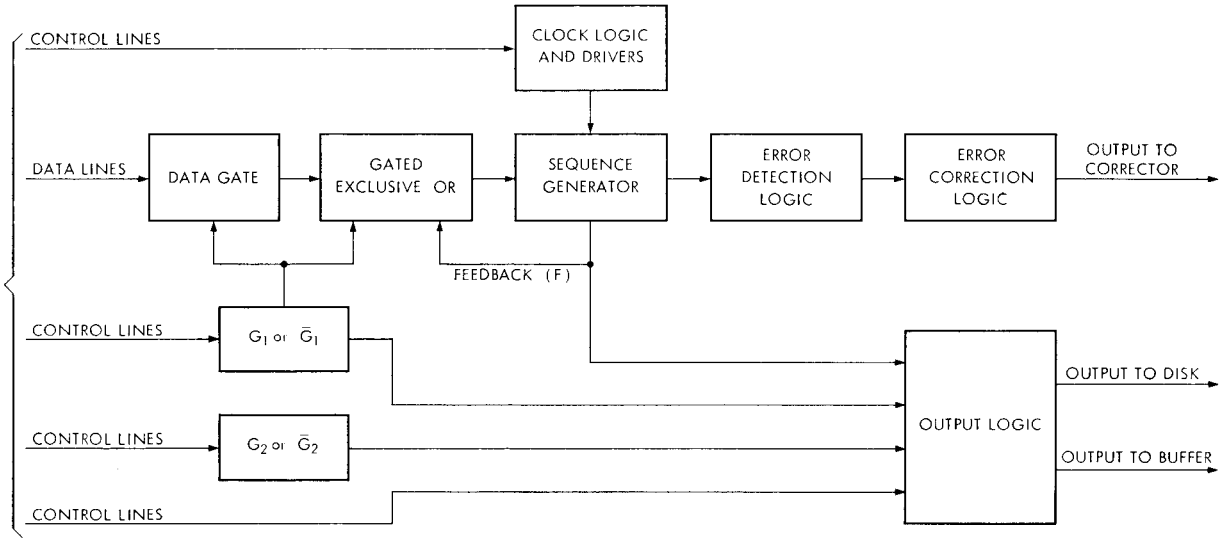
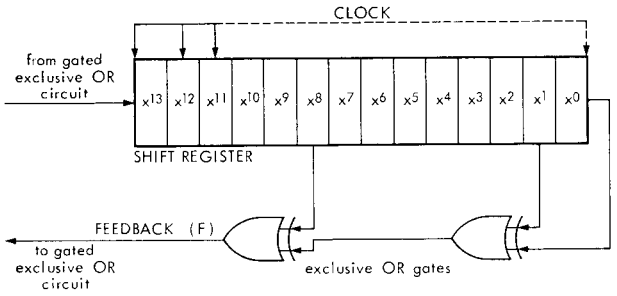



Fig. 3 of a Fire code[5] generated by the polynomial $g(x) = x^{14} + x^8 + x + 1 = 0$, whose length is 889. Although this code was derived for four-bit burst error correction, it was found to provide redundancy sufficient to permit correction of up to 14 of the 16 possible five-bit error bursts; conveniently, it generates 14 redundant bits, which is exactly enough to fill two RAMAC characters. For encoding, the register is first set to the ZERO state. Incoming data being recorded is simultaneously added (except for the start bits) to the feedback by the gated EXCLUSIVE OR circuit until the 98th data character has been encoded. At this time, the circuit is gated off and the feedback line is diverted to provide the correction redundancy that follows the recorded data to complete the 100-character RAMAC record. Similarly, the decoding cycle is begun with the register in the ZERO state. Data being transferred to the buffer is simultaneously added to the feedback and the redundancy is also added. Because the start bits are ignored, only 700 of the 889 available steps of the cycle are needed for the 100-character message.

*Figure 3* **Sequence generator.**




Since the characters containing redundancy will often fail the RAMAC parity checks, the latter are suppressed at appropriate times to prevent shutdowns. However, to allow the computer parity circuits to check the corrected data, two dummy characters are substituted for the redundancy which otherwise would shift into the buffer.

As was indicated in the previous section, when the encoded message length is less than that of the code cycle, additional complexity is required to avoid delays. In this specific case, the code cycle length is 889, but the encoded message contains only 700 bits. Thus, the 700th through the 696th characters of the sequence beginning with 10000000000000 were combined as indicated in Table 1 to derive correcting characters for all five-bit burst errors. This results in a circuit comprising sixteen 14-input AND gates and one 16-input OR gate becoming the error detection logic block of the correction unit. Since the code was derived for four-bit burst correction, it cannot be expected that all five-bit bursts will be correctable. In this particular code, there are 14 different sequences, each of length 889; thus, at least one sequence will contain more than one correcting character. In the present case, each of two sequences contains two correcting characters. This ambiguity may be resolved by excluding gates 2 and 4 (see Table 1). In the case of four-bit burst correction, only the odd-numbered gates are required.

During a correction cycle, whenever a correcting character appears in the register, a signal is generated which causes the bit appearing at the computer buffer output to be inverted. Then the 700th character (11011101001010) is added, modulo 2, before the next shift in order to allow the sequence for the next smaller burst to be generated in the register. An example is given in Table 2.

*Table 1* **Correcting characters for all 5-bit bursts.**

| Gate Number | Error (Burst Pattern) | Correcting Character | | | | | | | | | | | | | | Derivation and Characters Used |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $X^{13}$ | $X^{12}$ | $X^{11}$ | $X^{10}$ | $X^{9}$ | $X^{8}$ | $X^{7}$ | $X^{6}$ | $X^{5}$ | $X^{4}$ | $X^{3}$ | $X^{2}$ | $X$ | 1 | |
| 1 | 1 0 0 0 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 700 |
| 2 | 1 0 0 0 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0[c] | 700 ⊕ 696[b] |
| 3 | 1 0 0 1 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1[c] | 700 ⊕ 697 |
| 4 | 1 0 0 1 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1[d] | 700 ⊕ 697 ⊕ 696 |
| 5 | 1 0 1 0 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 700 ⊕ 698 |
| 6 | 1 0 1 0 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 700 ⊕ 698 ⊕ 696 |
| 7 | 1 0 1 1 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 700 ⊕ 698 ⊕ 697 |
| 8 | 1 0 1 1 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 700 ⊕ 698 ⊕ 697 ⊕ 696 |
| 9 | 1 1 0 0 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 700 ⊕ 699 |
| 10 | 1 1 0 0 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1[d] | 700 ⊕ 699 ⊕ 696 |
| 11 | 1 1 0 1 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 700 ⊕ 699 ⊕ 697 |
| 12 | 1 1 0 1 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 700 ⊕ 699 ⊕ 697 ⊕ 696 |
| 13 | 1 1 1 0 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 700 ⊕ 699 ⊕ 698 |
| 14 | 1 1 1 0 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 700 ⊕ 699 ⊕ 698 ⊕ 696 |
| 15 | 1 1 1 1 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 700 ⊕ 699 ⊕ 698 ⊕ 697 |
| 16 | 1 1 1 1 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 700 ⊕ ····· ⊕ 696 |

[a] Gates 1 + 2 + 3 + . . . + 15 + 16 = Outputs in Figure 2 (implemented with 241 diodes and one transistor).
[b] ⊕ = EXCLUSIVE OR (Mod-2 addition).
[c, d] Correcting characters marked identically occur in the same sequence and thus may be confused, resulting in the attempted correction of the wrong pattern.

*Table 2* **Example of correction* for error burst pattern 10101.**

| Time | Error Burst To Be Corrected | Contents of Shift Register | | | | | | | | | | | | | | | Correction Output, $C_1$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $T_0$ | 1 0 1 0 1 | | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| $T_0{}^+$ | | ⊕ | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| $T_0{}^{++}$ | | = | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| $T_1$ | | | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| $T_2$ | 1 0 1 | | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| $T_2{}^+$ | | ⊕ | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| $T_2{}^{++}$ | | = | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| $T_3$ | | | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| $T_4$ | 1 | | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| $T_4{}^+$ | | ⊕ | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| $T_4{}^{++}$ | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

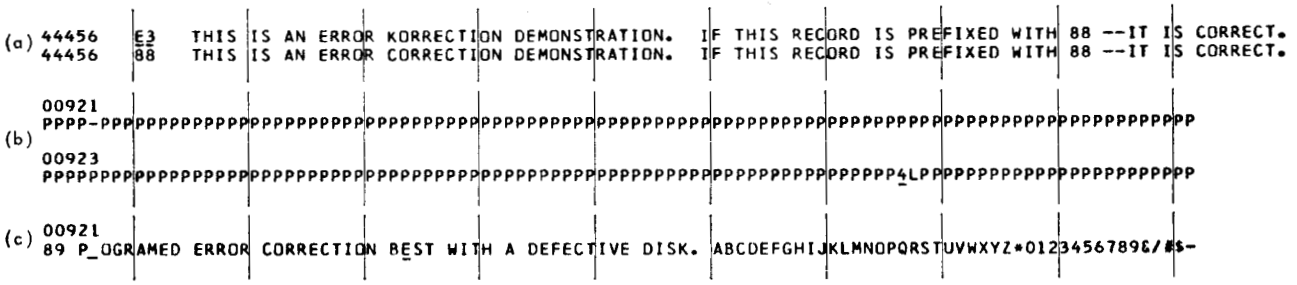
* During a correction cycle, whenever a correcting character appears in the register of Figure 3, the character 11011101001010 is added, modulo 2, to this character. This addition occurs before the next normal shift to allow the sequence for the next smaller burst to be generated in the register.

### Testing the error correction unit

For preliminary testing, the error correction unit was installed so that it could be switched in or out of the computer system. This allowed the computer to serve other users in the normal manner and made it convenient to test the correction of errors deliberately inserted in a recording. Data was recorded on a non-defective disk with the encoder active in the system and subsequently altered with the error correction unit bypassed. With the decoder and correction circuits active in the data line, all inserted errors up to four bits in length and all but four of the 16 possible five-bit error bursts were read correctly. A double adjacent error is corrected as shown in Fig. 4A. The character *K*, 1001001, was substituted for a *C*, 1111001, which appeared in the original message. Since these binary patterns differ in two consecutive bit-positions, substituting one for the other causes a double adjacent burst.

Although the two pairs of ambiguous error patterns were at times read correctly, at other times, when they occurred in certain locations, the unit failed to correct the actual error and caused an additional error by attempting to correct the related pattern. Similarly, additional errors may be caused if bursts longer than five bits occur, since these might be detected by the system and confused with correctable error patterns. During this and later tests with defective disks, machine shutdowns caused by parity or compare failures were suppressed by

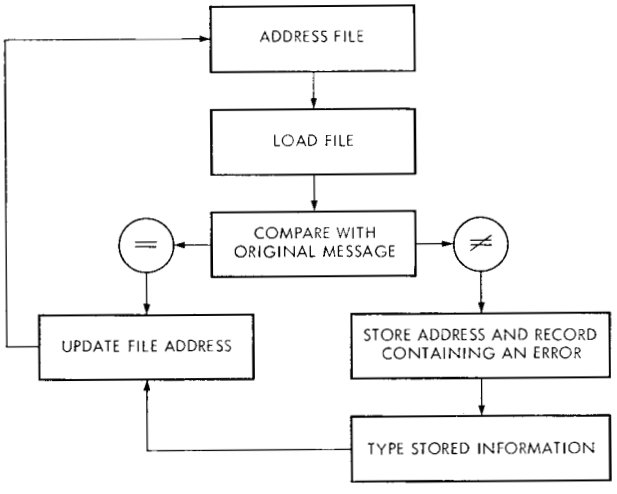
*Figure 4* **Three sample printouts:** *(a) Preliminary test message on nondefective disk, showing inserted error and correction; (b) P test message on inside track, read with error correction circuits bypassed; and (c) Final test message on inside tracks of etched disk read with correction circuits operating.*




a parity switch (normally employed only for special machine testing) to allow printing out errors that fail machine parity checks.

For programmed testing with a defective disk, one side of a disk was defaced by etching four nonradial lines, 40 mils wide, from the outer edge to the inner edge of the recording area. The program was compiled to perform the steps indicated in Fig. 5. First, three test patterns were recorded with the encoder active and were read out with the correction circuits bypassed to allow determination of the size and location of potential error bursts. The first pattern tested was a repetition of the RAMAC space bit which is required for clock synchronization and which consists of one flux reversal at the beginning of each character time. Results of the test with this recorded pattern indicated that very few errors were caused by false flux reversals resulting from defects. Subsequent test patterns were chosen, therefore, to determine the number of errors caused by deletion of flux transitions (ONES) recorded over a surface defect.

*Figure 5* **Simplified flow diagram of test program.**




The next test was made with two patterns: the character "*G*", or binary pattern 1111100; and the character "*P*", or binary pattern 1011101 (unfortunately, no 1111111 pattern was available). The etched lines caused errors ranging from a single bit to seven consecutive bits, depending on the test pattern and, also, on the track location since the recording density was higher at the inner tracks. Results of a five-bit and a seven-bit burst are shown in Fig. 4b. In address 00921, the character *P* (1011101) was replaced by a *hyphen*, (1000000) thus resulting in a 11101 burst. In address 00923, the characters *PP* were replaced by *4L* (*L* is 1011000). Characters that fail the parity check made during a readout are underscored by the output typewriter; thus, the bit-pattern 0000101 is interpreted as a *4*, 0000100, which has failed the parity check. Since 10111011011101 is replaced by 10110000000101, the resulting error pattern is 1011011. Although the error appears as *4L* on the printout sheet, these characters were read from the disk file to the buffer from right to left, thus the actual error pattern results from *L* preceding *4*. (Since the defect affected two adjacent characters at this location, it actually spanned slightly more than eight bits; that is, it included the intervening space and the space bit of the second character).

After the addresses and extent of potential errors had thus been determined, a test message was assembled and the program rerun with the error correction circuits active in the system. All errors were corrected except those in excess of five bits and the four ambiguous error patterns. When the latter occurred, the correction unit often caused additional errors in its attempt to correct the related pattern, as Fig. 4c illustrates. In this figure an *R*, (1010010) became 0000000, and was interpreted as a blank (0000001) which failed the parity check. The character *TE* became *BE* with the resulting error burst being 1101. That is, the pattern 11101010111000 was replaced by 11101001101000.

It may be inferred that the six-bit burst was confused by the correction circuits with a four-bit burst occurring in a different location; by inverting the bits at that location, an additional error was caused. The additional errors due to this ambiguity can be avoided by modifying the error correction logic so that the more probable of two ambiguous patterns is corrected, or so that such patterns are ignored. To correct all five-bit bursts, a different code would be required.

## Conclusion

The described error correction unit demonstrated the application of automatic error correction techniques employing burst-error-correcting cyclic codes (such as those developed by Fire[5] and Melas[7] on principles first investigated by Abramson[8]) to computer memory lines. These are flexible regarding both block length and error-correcting capability, and are susceptible to economical implementation as suggested by Meggitt[6] and Peterson.[4,5] A Fire[5] code designed for burst error correction was employed in this investigation; in applications where errors may occur randomly throughout a data block, different codes (e.g., Bose-Chaudhuri codes[5]) would be required. It has also been suggested that, since a surface defect may cause errors in two dimensions, two-dimensional codes[9,10,11] could be useful. Likewise, it should be remarked that shortened cyclic codes[5] might be advantageous.

Error correction of the type described herein is especially suited to a system, such as RAMAC, in which data is buffered during transfer between the computer memory and an input/output device. In such a system, the unit probably could allow recording on many disks now rejected because of surface defects. Since this investigation was undertaken to examine feasibility and to expose problems that might attend the application of burst error correction techniques, no quantitative measure was developed for the improvement achieved by such a system over the performance of a system having no correction capability.

Fifty percent of all possible errors not corrected by an automatic correction device could be detected by the RAMAC parity check. However, this check requires one redundant bit per character, or a redundancy of 12-1/2 per cent; more reliable detection is possible with a cyclic error detection code requiring only 1 to 2 percent redundancy. A system combining cyclic detection and correction codes[1] could correct error bursts of up to five bits and detect 99 percent of all uncorrected errors with a total redundancy of 3 to 4 percent—on a message basis, rather than a character basis.

Finally, the design of the error correction unit was made more complicated by the need to take into account the difference between message length and code cycle length. Because of this, and the requirement of adapting a transistorized engineering model to an existing tube system, the component count is higher than would be required for such a unit developed as an integral part of a computer. Because of anticipated simplification of control logic and elimination of tube voltage translations, the total count of 137 transistors, 241 diodes, and 12 tube pluggable units should be considerably smaller. In the present case, if the length of the computer record and the cycle length of the code were the same, 28 transistors and 231 diodes would be eliminated because the large AND-OR error detection gate and associated emitter followers would be unnecessary. A saving of another 17 transistors would result if, in addition, no voltage translating were required. It is also expected that the control logic, being simpler, could require about ten fewer transistors.


## Acknowledgments

The code selected and the implementation described were suggested by C. M. Melas. Selection of RAMAC data and control signals was made according to the recommendations of E. H. Scherer. The proposed solution to the variable block length problem is based on an approach originally suggested by J. Appelquist. The advice of F. B. Wood on the implementation and tests and of F. D. Thompson on RAMAC logic and programming is also gratefully acknowledged.


## Appendix: Error correction logic and component requirements

Figure A shows the logic diagrams corresponding to the block diagram of Fig. 2 and Table A lists symbols and standard or equivalent designations for RAMAC data and control signals. In this implementation, it is often necessary to derive a single control signal from several computer signals. Note, for example, in Fig. A4 that four signals are combined to produce $G_1$; in Figs. A1, A4, A5, and A7 that $S_1S_2$ is required for the encoding cycle; and in Figs. A1, A5, and A7 that $S_1R$ is required for the decoding cycle.

The error detection logic (Table 1) produces signal $C_1$ as an input to the correction logic, as shown in Fig. A3. Signal $C_1$ becomes a ONE and flip-flop $C_2$ is set whenever a correcting character is recognized by the detection logic during a correction cycle, Fig. A3. The result is that $Output_3$ becomes a ONE and the incorrect bit is inverted after a one-bit delay provided by flip-flop D. Gate $G_1$, in addition to controlling the data gate and the gated EXCLUSIVE OR CIRCUIT, Fig. A2, supplies logic for inserting redundant bits in the last two character positions of a record during an encoding operation. Gate $G_2$ in Fig. A6 controls the removal of the redundant bits

from a record and the substitution of the dummy characters during a reading operation.

The $Output_1$ logic in Fig. A5 allows data to go from the buffer to the disk, interrupting the flow during the last two character times to allow insertion of the sequence generator feedback. Similarly, the $Output_2$ logic of Fig. A5 allows data to pass from the disk to the buffer except when the two dummy characters are inserted.

The function $C_{RB}C_1W\overline{T}$ was used to derive the extra shift pulse mentioned in Table 2. Signals $\overline{S_1S_2}$ and $\overline{S_1R}$ are used to inhibit machine halts due to parity or file check failure during transfers between the buffer and disk file. This allows the redundancy to be recorded and allows errors to be transferred in order to give the correction circuit an opportunity to perform its function. During transfers from the buffer to the drum, however, a parity failure resulting from an uncorrected error will cause a machine halt.

*Figure A* **Logic diagrams for error correction unit.**

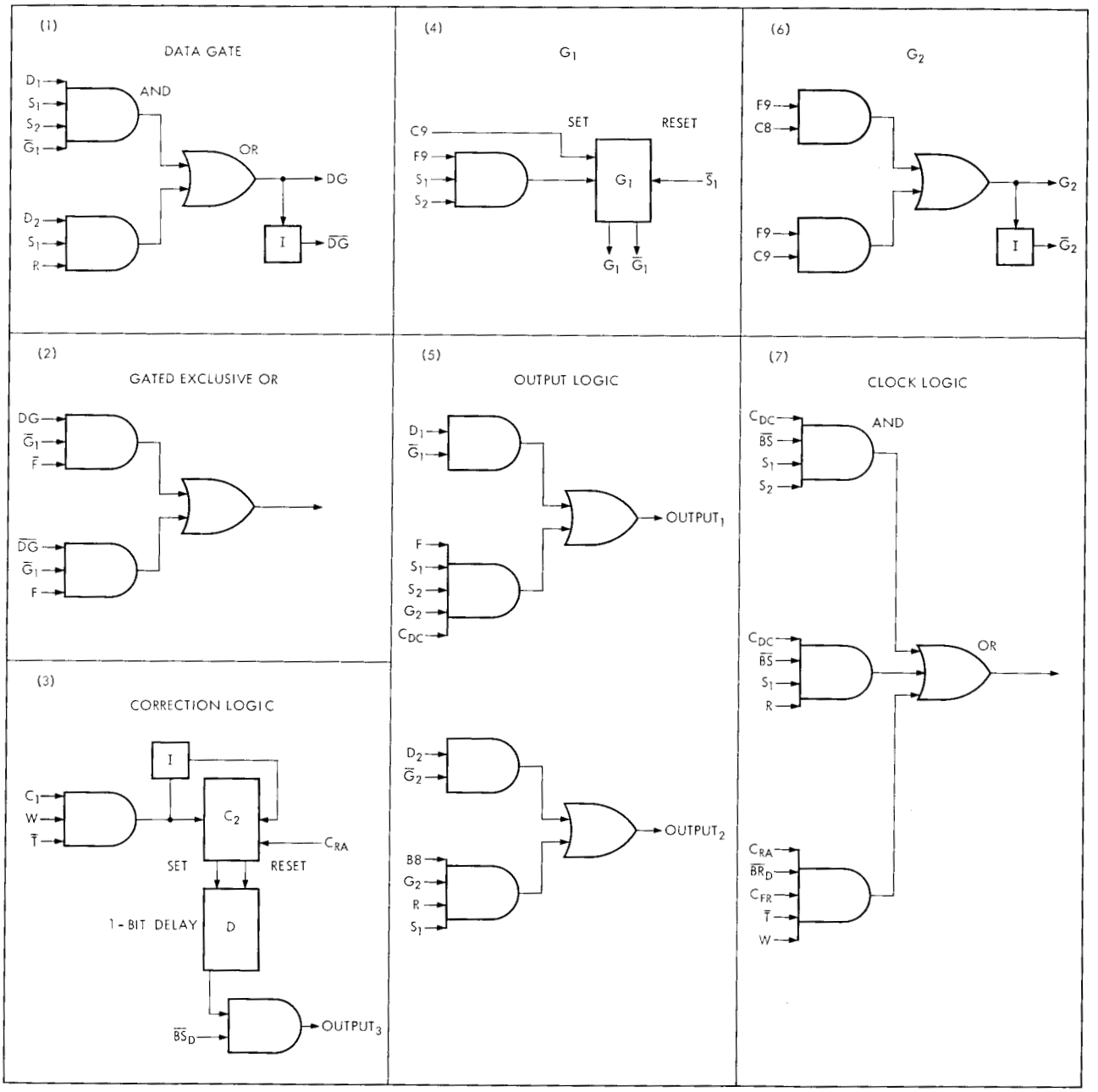


*Table A Standard or equivalent designations for* RAMAC *data and control signals.*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Character and Field Ring Gate (Drum) | $C_{FR}$ | Disk Clock Gate BS | BS | Drum Phase ΦA | $C_{RA}$ | 1 Cycle (to Reset Sequence Generator) | I |
| Data to Encoder | $D_1$ | Disk Clock Gate C8 | C8 | Drum Phase ΦB | $C_{RB}$ | | |
| Data to Decoder | Ds | Disk Clock Gate C9 | C9 | Drum Clock Gate BR | $BR_D$ | Store Check Gate | $S_2$ |
| Disk Clock ΦC | $C_{DC}$ | Disk Clock Gate F9 | F9 | Drum Clock Gate BS | $BS_D$ | T = R Gate | T |
| Disk Clock Gate B8 | B8 | Disk Cycle Gate | $S_1$ | R Cycle Gate | R | W Cycle Gate (Drum) | W |

## References


1. C. M. Melas, "Reliable Data Transmission Through Noisy Media—A Systems Approach," *AIEE Transactions on Communications and Electronics*, **80**, Part 1, 501-504 (1961).
2. *IBM RAMAC 305 Customer Engineering Manual of Instruction*, Form 227-3534.
3. *IBM RAMAC 305 Instructional System Diagrams.*
4. W. W. Peterson, "Binary Codes for Error Control," *Proceedings of National Electronics Conference* **16**, 15 (1960).
5. W. W. Peterson, *Error Correcting Codes*, M.I.T. Press and John Wiley and Sons, Inc., New York, 1961.
6. J. E. Meggitt, "Error Correcting Codes and their Implementation for Data Transmission Systems," *IRE Transactions on Information Theory* **IT-7**, 4, 234-244 (1961).
7. C. M. Melas, "A New Group of Codes for Correction of Dependent Errors in Data Transmission," *IBM Journal of Research and Development* **4**, 58-65 (1960).
8. N. M. Abramson, "A Class of Systematic Codes for Non-Independent Errors," *IRE Transactions on Information Theory*, **IT-5**, No. 4, 150-157 (1959).
9. P. Calingaert, "Two Dimensional Parity Checking," *J. Assoc. Computing Machinery* **8**, 186-200 (1961).
10. M. Rubinoff, "N-Dimensional Codes for Detecting and Correcting Multiple Errors," *Communs. ACM*, Vol. 4, pp. 545-551 (1961).
11. B. Elspas, *Design and Instrumentation of Error-Correcting Codes*, Final Report, Contract AF 30(602)-2327, RADC-TDR-62-511 (October, 1962).


*Received February 14, 1963*