Suffitt
53.17

COMPANY CONFIDENTIAL

August 21, 1957

Exchange Memo #24

BY: J. D. Calvert

A SYMMETRICAL AND FLEXIBLE ERROR CORRECTING CODE

The code described in this memo is symmetrical in eight bit groups, thus lending itself to simple generation on a byte basis as well as a word basis without sacrificing any of the features of the standard Hamming code. This code may be generated very easily in any unit that handles information in eight bit bytes.

A SYMMETRICAL AND FLEXIBLE

ERROR CORRECTING CODE

The main advantage of the proposed error correcting code is that it may be easily generated on a serial by byte basis. The total transistors needed for generation on a byte basis is approximately two hundred and twelve excluding intermediate storage for check bits. It might be reasonable to generate the error correcting code in some input/output devices. It could also be generated in the basic Exchange as shown in figures 2, 3A, and 3B. Parallel generation, checking and correction on a sixty four bit word using the proposed code requires less than seventeen hundred transistors which is substantially the same as the standard Hamming code.

The standard Hamming code does not lend itself to simple generation on a byte basis. Since the Hamming code check bits effectively occupy word positions intermixed with data, the word cannot be divided symmetrically. In other words, corresponding bits of successive bytes of true data do not affect the same check bits. The proposed code removes the check bits from the data portion of the word and is arranged so that each eight bit group has the same general check bit configuration except for two minor exceptions. Eight check bits are used and all of the error detecting and correcting features of the standard Hamming code remain.

Figure 1 shows the format for a full word with error correcting code. For ease of understanding the word positions are numbered starting with zero. The check bits have been assigned the binary value shown by the subscript for ease of mental decoding. The check bits occupy positions 64 through 71 of the word. Every data bit is included in at least three check bits. Data positions zero and thirty-two are special cases and are handled separately. C_T is a total parity bit which includes all other check bits as well as data bits. A study of figure 1 shows that each data position is affected by a unique combination of check bits whose binary sum indicates the position in error, except for positions zero and thirty-two.

Any single bit error in the data portion of the word will change the C_T as well as at least two other check bits which indicate the position in error. Any single bit error in the check bits will change C_T as well as the bit in error. Any double bit error in either data or check bits will leave C_T unchanged. An error in C_T alone will be known since no other check bits will be in error.

Serial ECC Generator (Figures 3A, 3B)

Figure 3A shows the logic necessary to generate all of the check bits except C_T. Figure 3B shows the logic necessary to generate C_T. As each byte comes in, the check bits are generated, as shown, according to which byte is involved. The resulting check bits are combined in exclusive or circuits with the previously generated check bits, which were stored. The new check bits which result, replace the existing check bits in storage. In this way, the check bits are updated as each new byte is assembled into the word. C_T is generated on data bits only until the last byte for a word is received. When the last byte is received the final version of the other check bits are used to help generate the final C_T.

Parallel Checking and Correction (Figures 4, 5A, 5B, 5C, 5D, 5E, 5F, 5G and 5H)

Figure 4 shows the overall logic of parallel error detection and correction. Figure 5A shows the logic necessary to generate C_1. Figure 5B shows the generation of C_2 as well as the partial generation of C_0, C_4, C_8, C_{16}, C_{32} and C_T. Figure 5C completes the generation of all check bits except C_T. Figure 5D completes the generation of C_T. Figure 5E shows the comparison of the generated check bits with the original check bits to determine error conditions. Also figure 5E shows double error indication. Figure 5F whose inputs come from figure 5E shows the decoding to locate which section of a word is in error. Figure 5G shows the decoding of the column within the section that is in error. Figure 5H shows the decoding of the seventy-two bit positions for error conditions. The outputs of figure 5H would feed into seventy-two exclusive or circuits to be matched with the original data for error correction.

ps

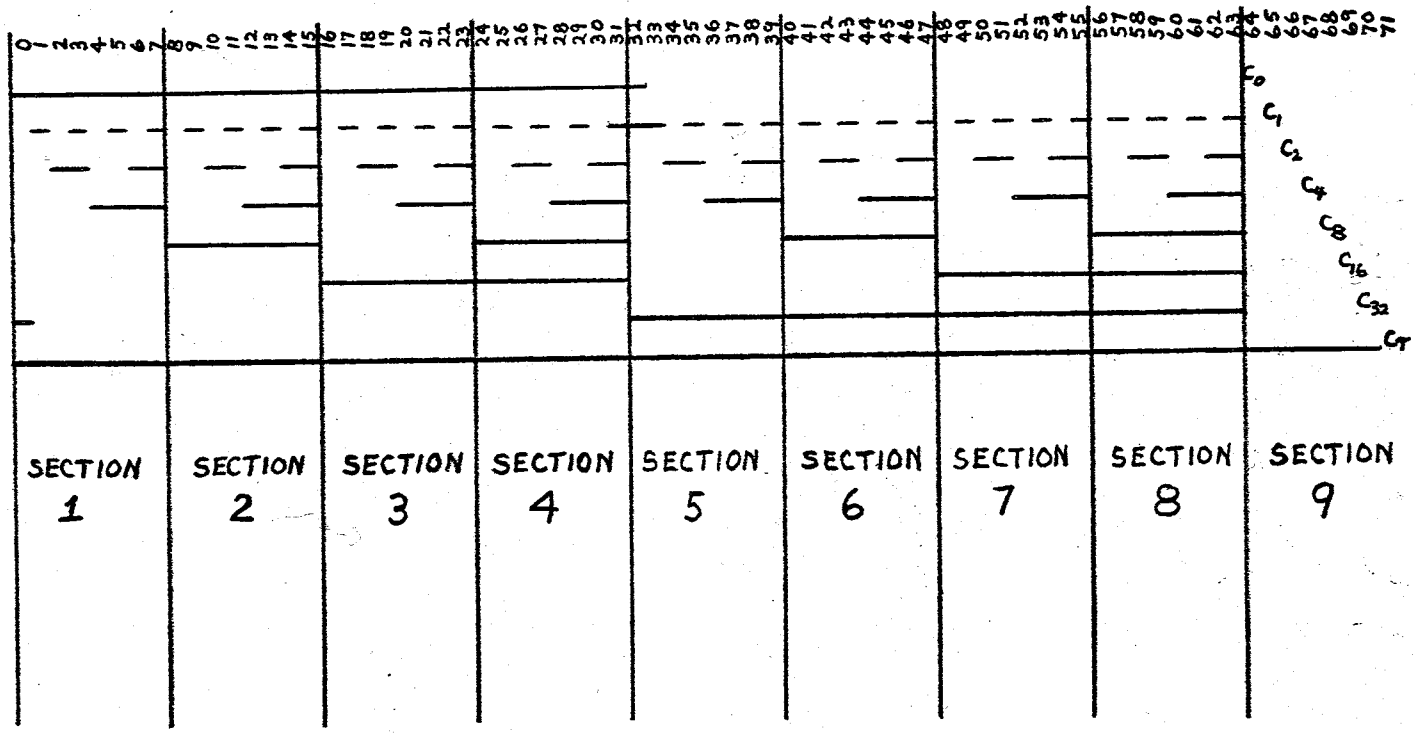

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
C_0
C_1
C_2
C_4
C_8
C_{16}
C_{32}
C_T
SECTION 1
SECTION 2
SECTION 3
SECTION 4
SECTION 5
SECTION 6
SECTION 7
SECTION 8
SECTION 9

FIGURE 1

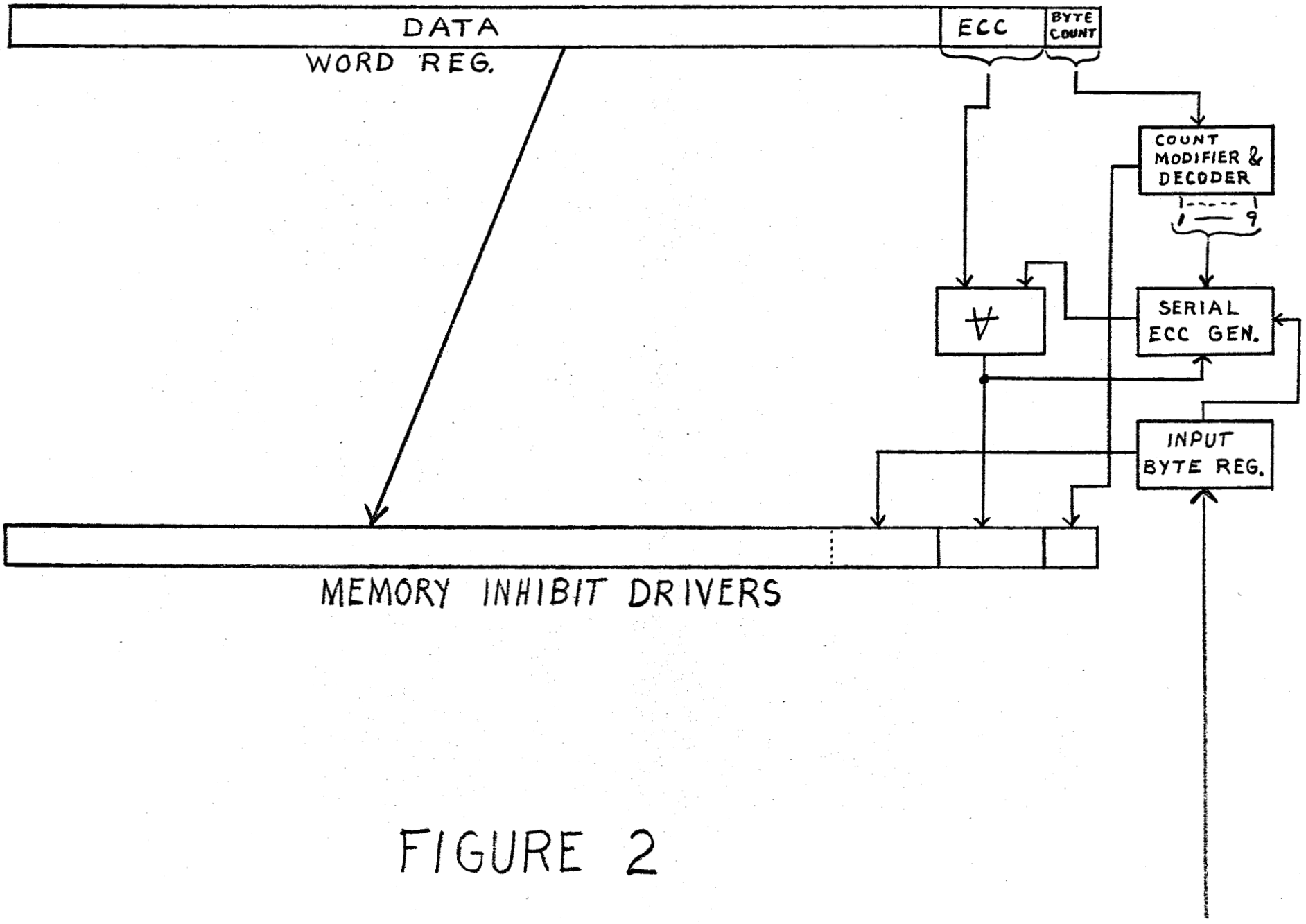

DATA
WORD REG.
ECC
BYTE COUNT
COUNT MODIFIER & DECODER
1 — 9
SERIAL ECC GEN.
INPUT BYTE REG.
MEMORY INHIBIT DRIVERS

FIGURE 2

SERIAL ECC GENERATOR

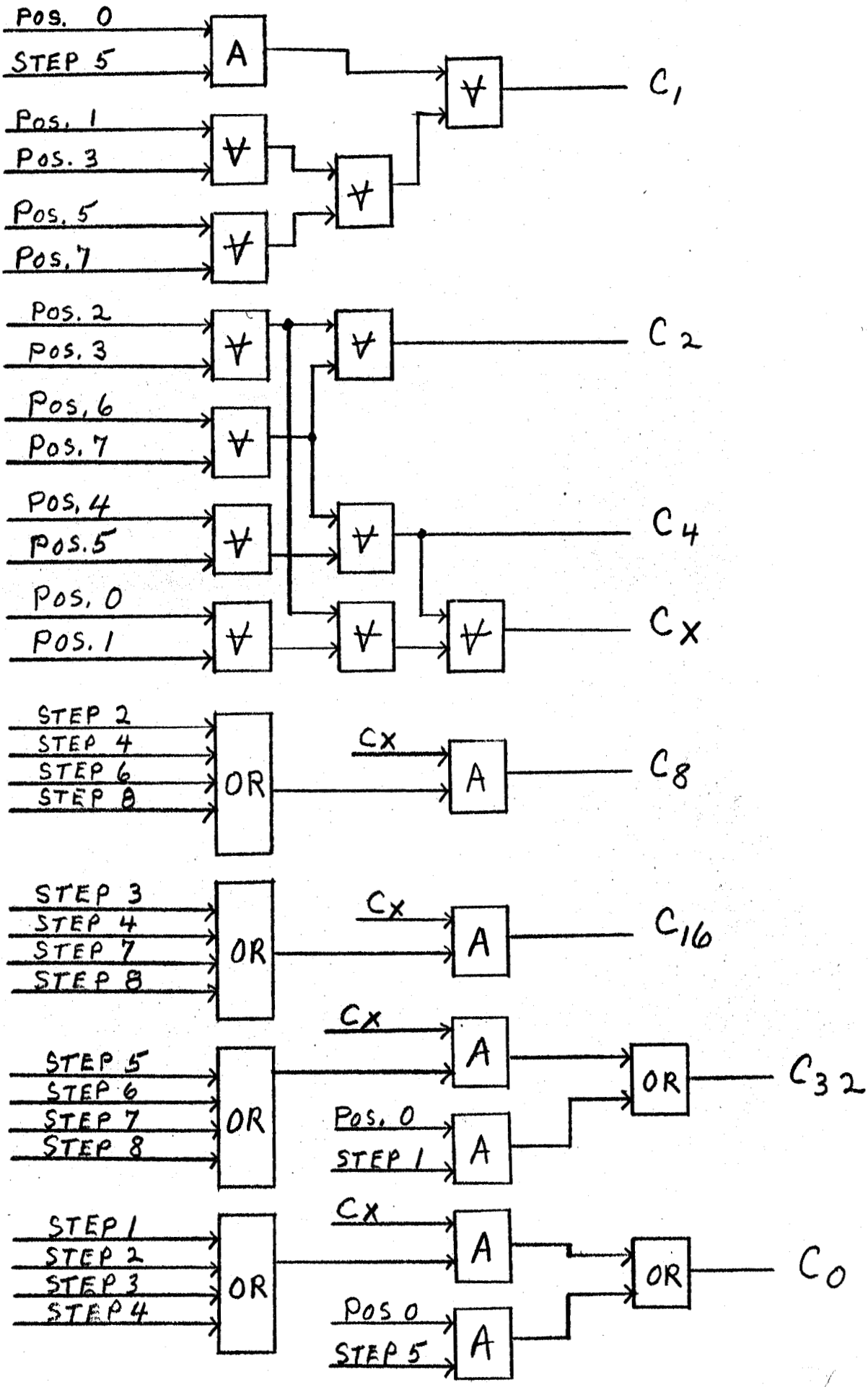
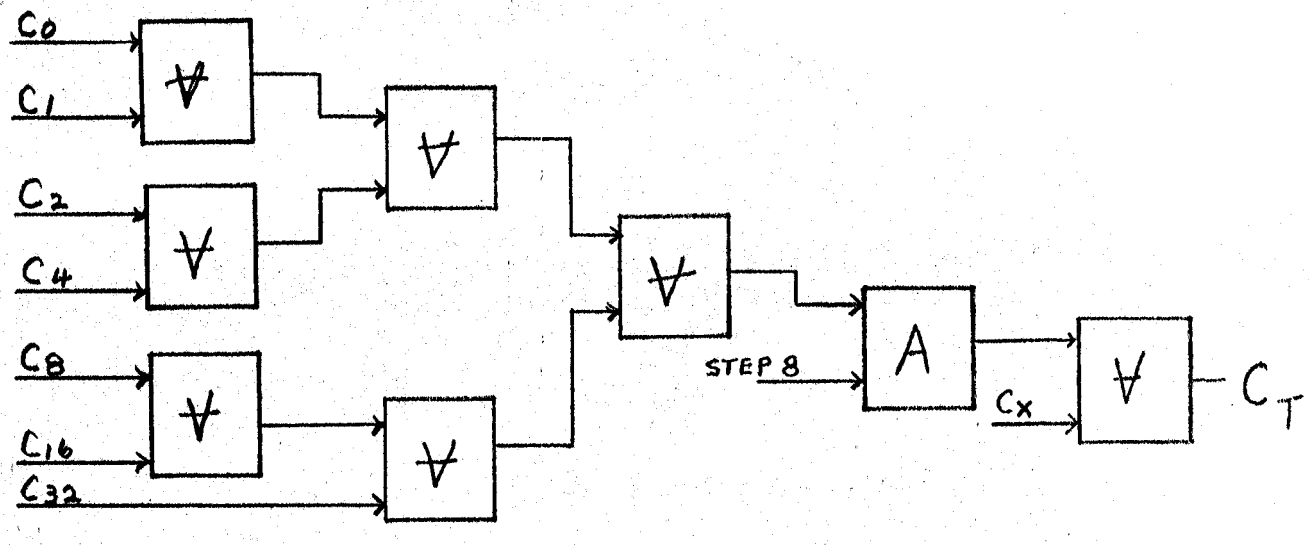


FIGURE 3A



SERIAL ECC GENERATOR

FIGURE 3B

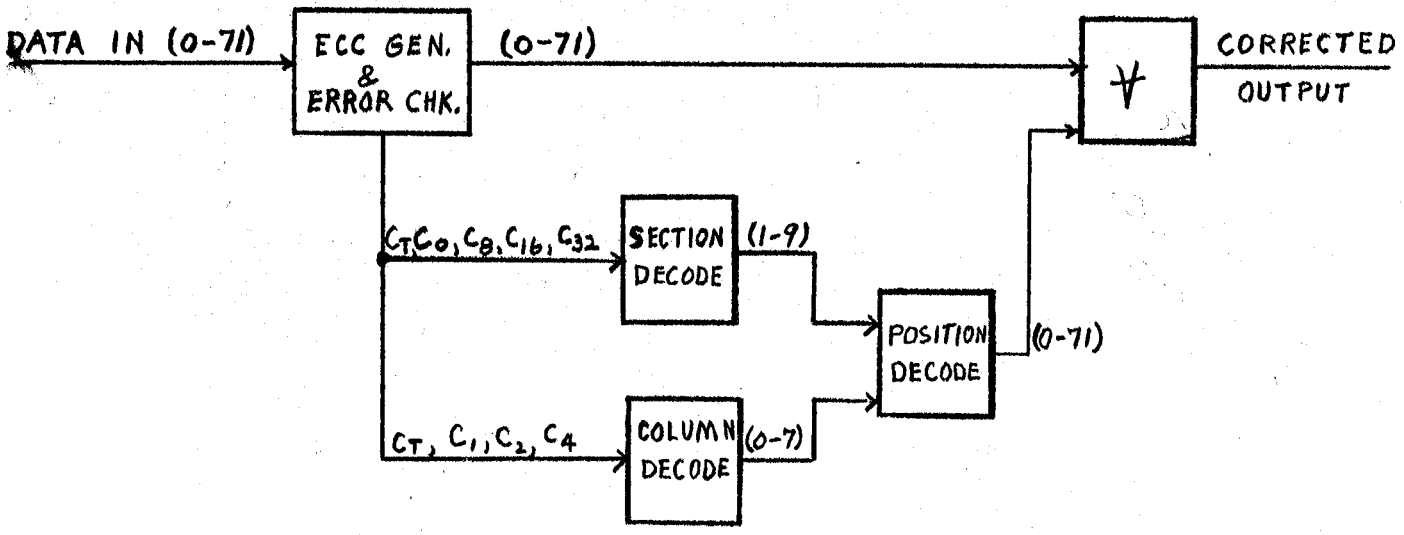
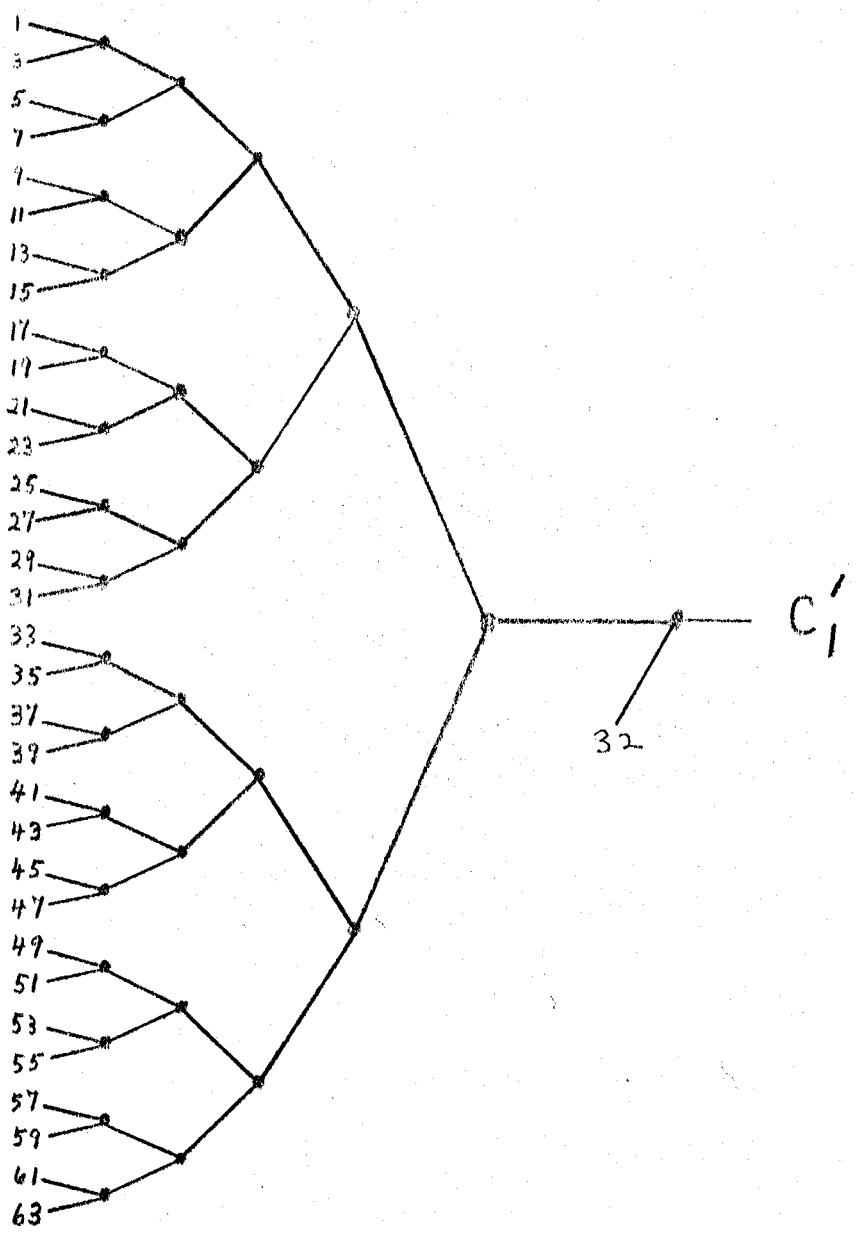

DATA IN (0-71)
ECC GEN. & ERROR CHK.
(0-71)
CORRECTED OUTPUT
$C_T C_0, C_8, C_{16}, C_{32}$
SECTION DECODE
(1-9)
POSITION DECODE
(0-71)
C_T, C_1, C_2, C_4
COLUMN DECODE
(0-7)

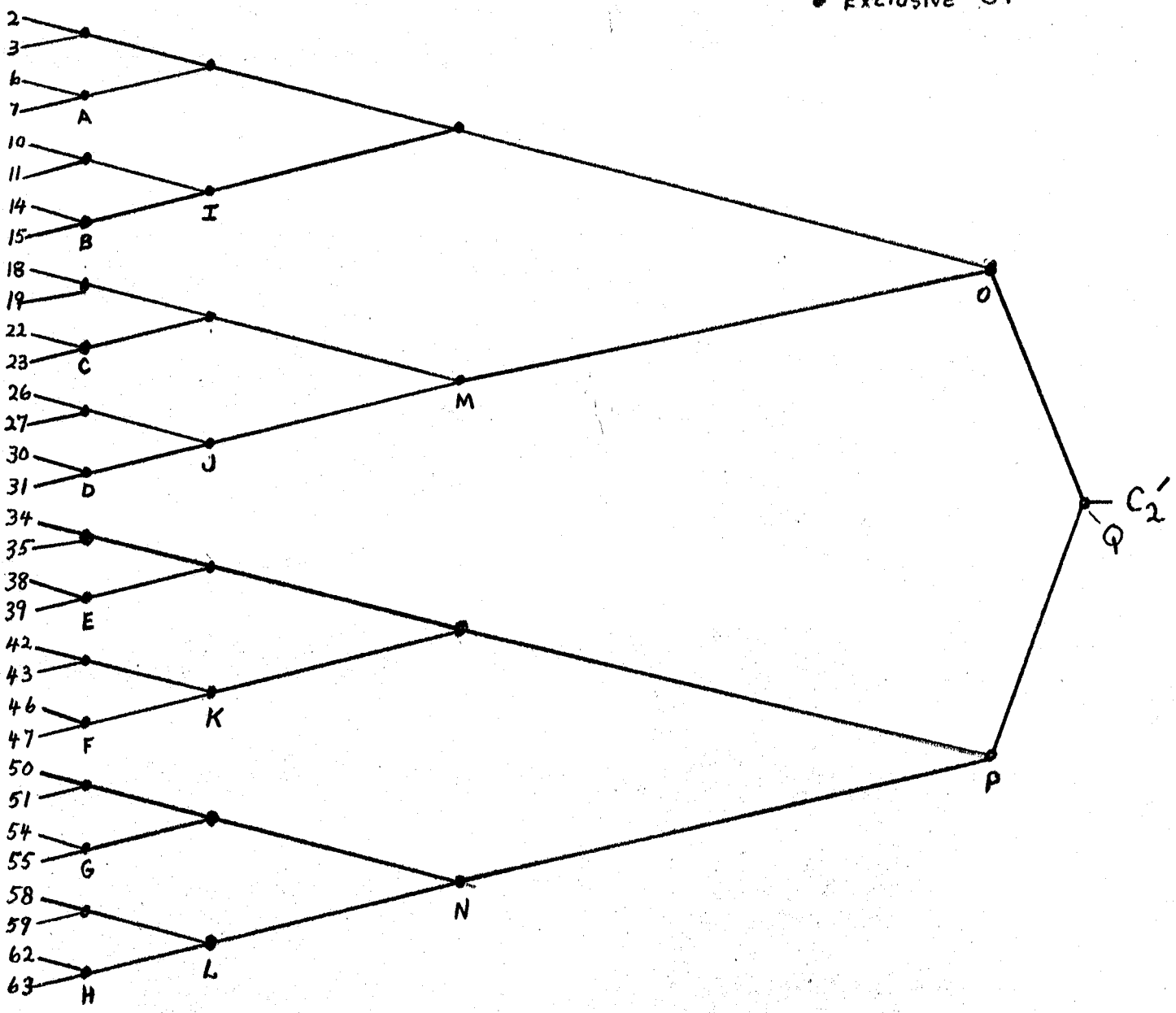
FIGURE 4

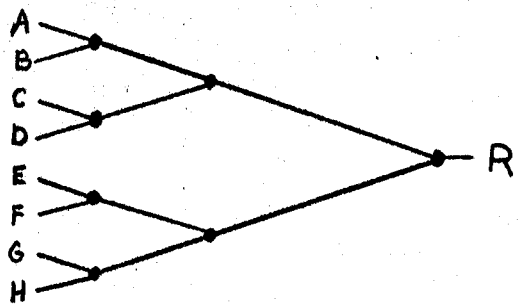


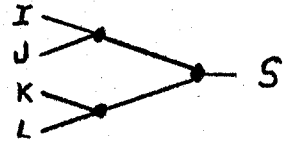
PARALLEL ECC GEN.

FIG. 5A

● Exclusive Or

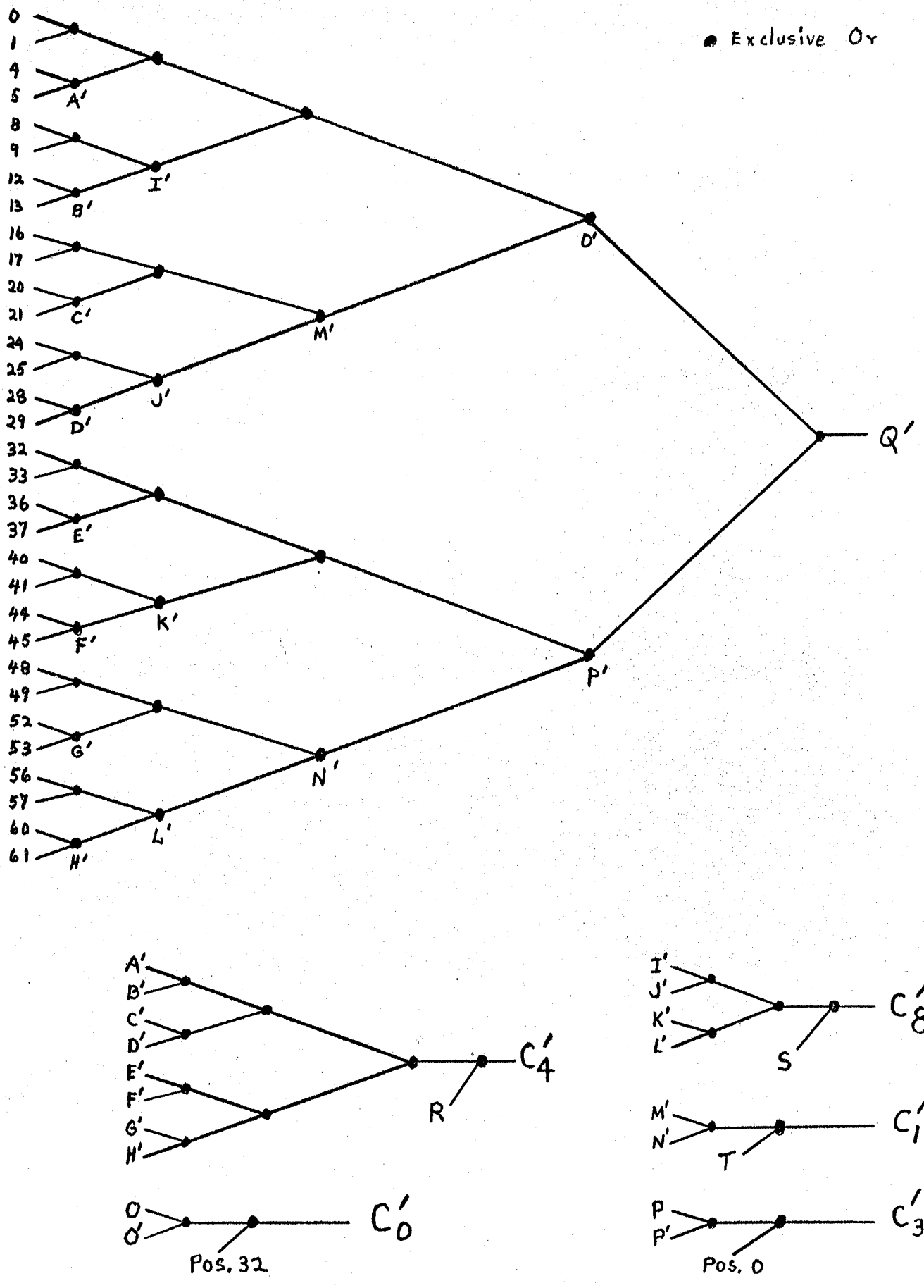






PARALLEL ECC GEN.

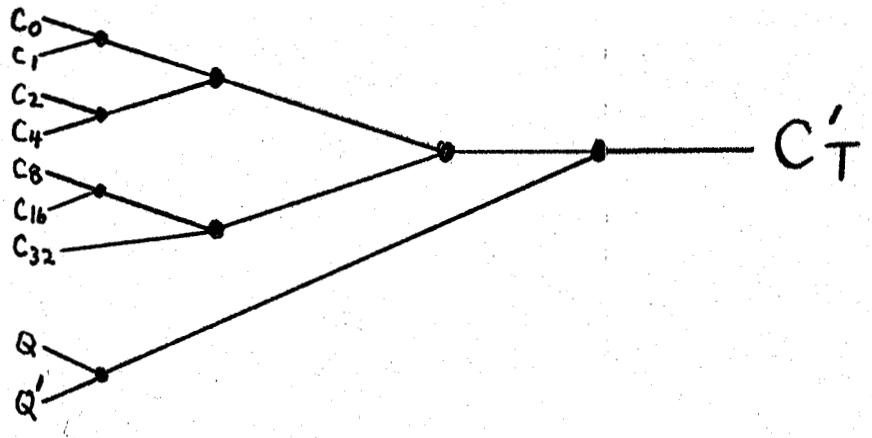
FIG. 5B



PARALLEL ECC GEN.

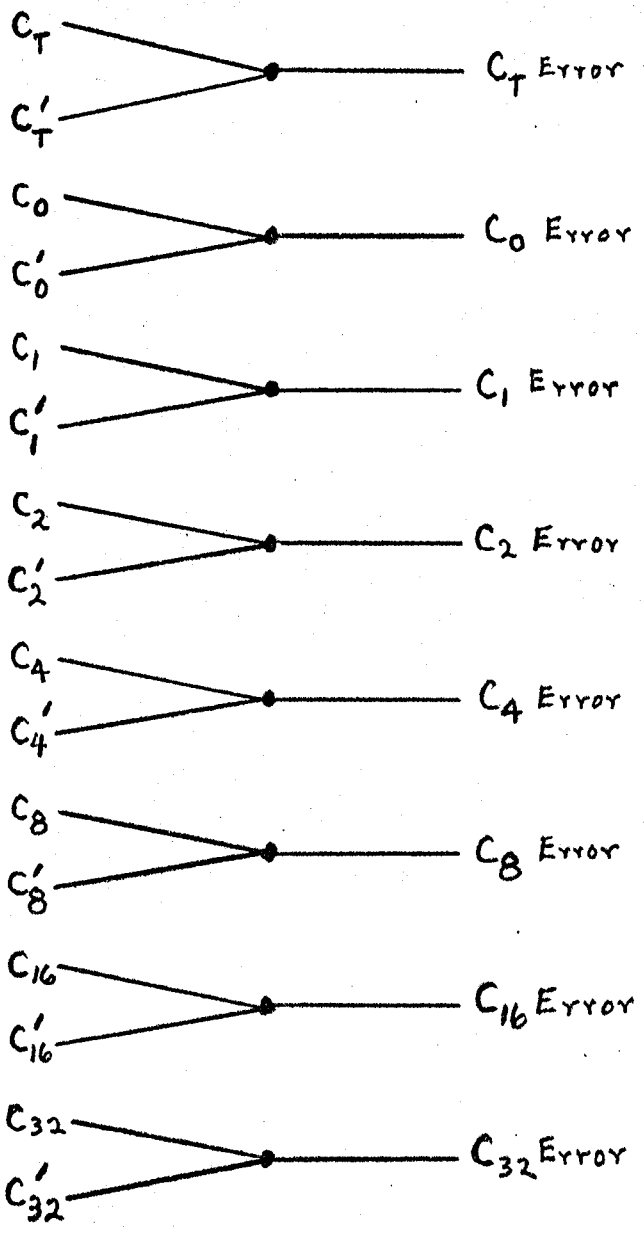
FIG. 5C

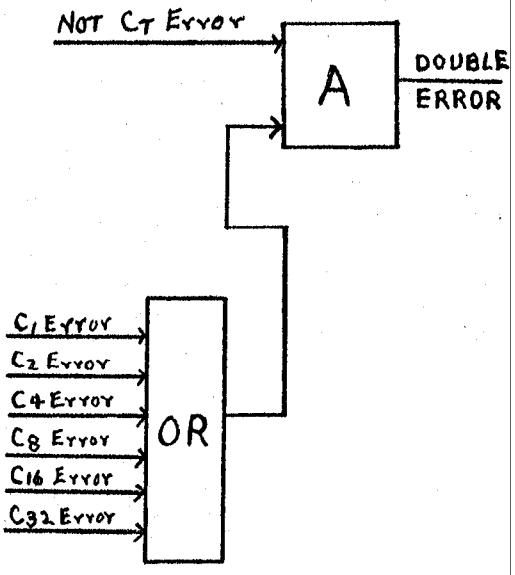
• Exclusive Or



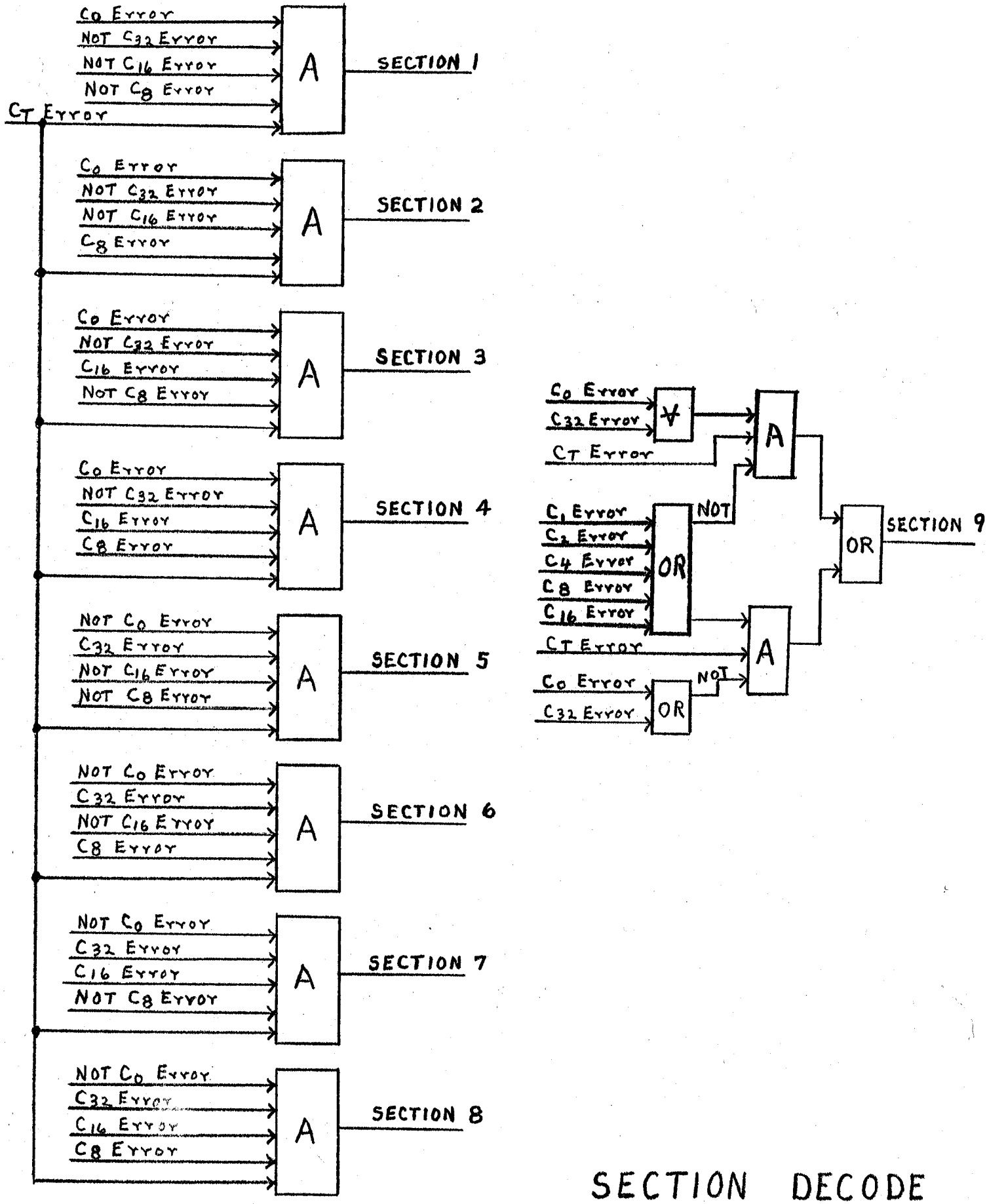
PARALLEL ECC GEN.
FIG. 5D

• Exclusive Or



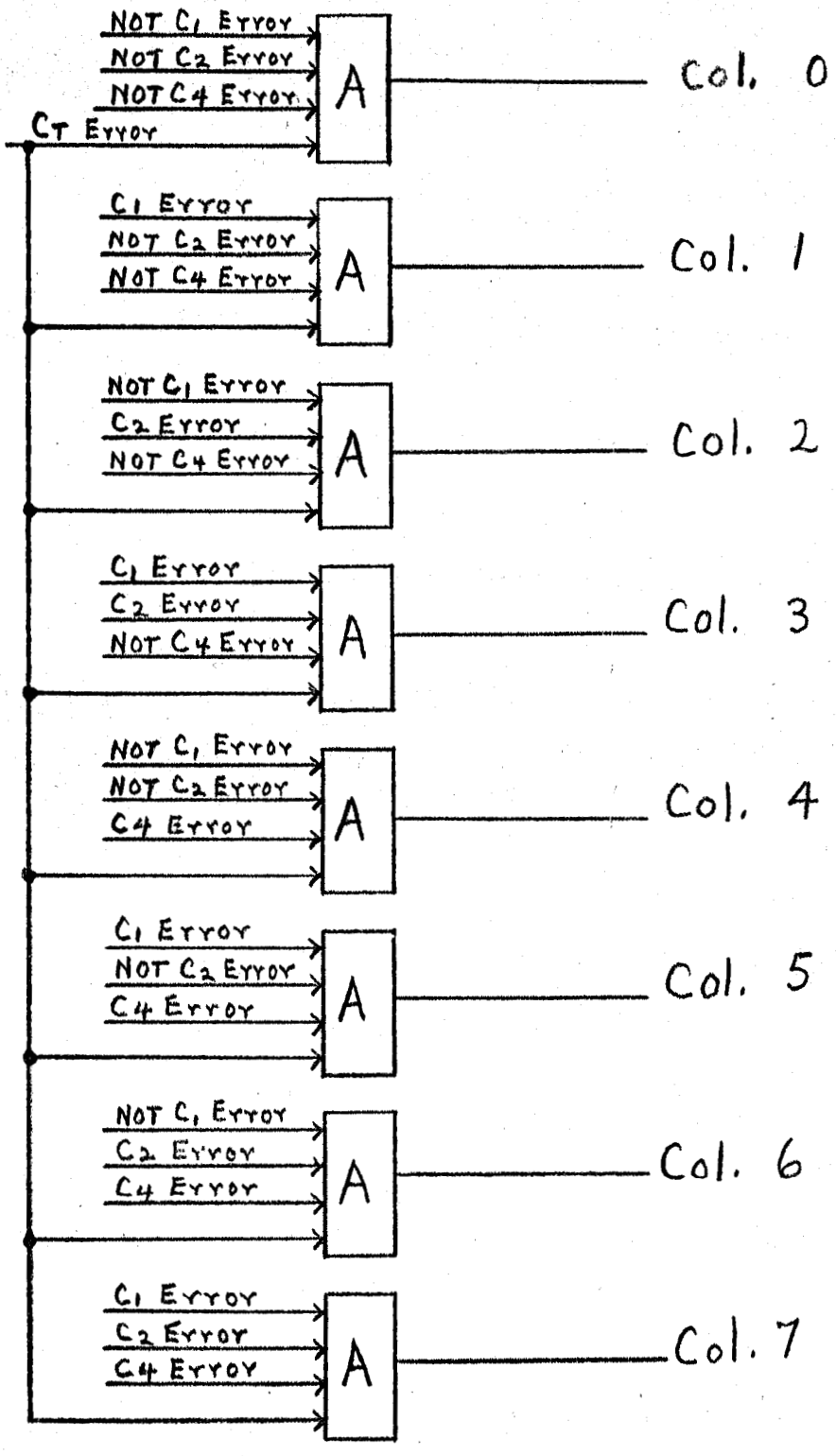


ERROR DETECT
FIG. 5E



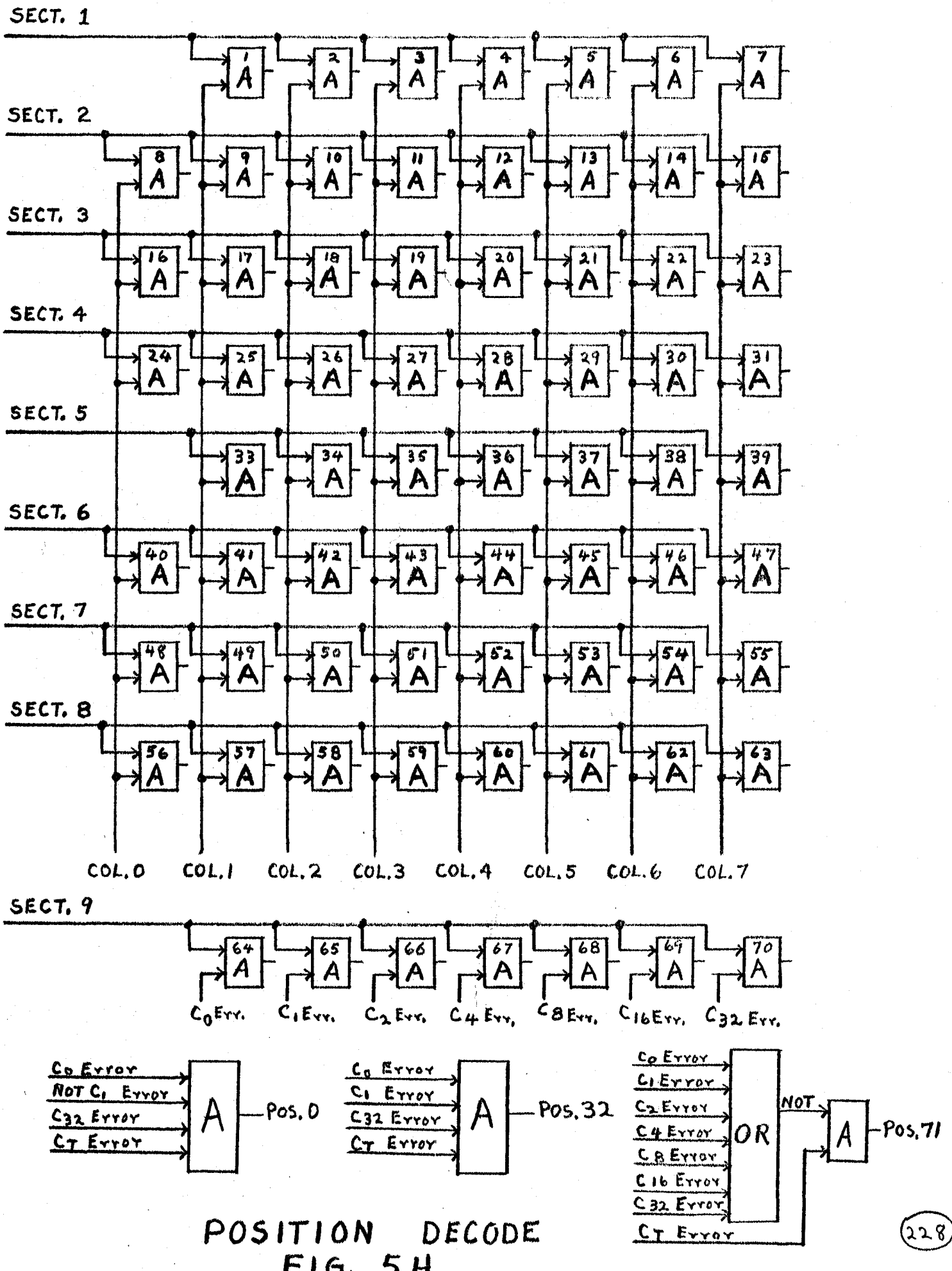
SECTION DECODE

FIG. 5F



COLUMN DECODE

FIG. 5G



POSITION DECODE
FIG. 5H