sum = (x+y) $\bar{x}y$ = $x\bar{y} + \bar{x}y$

carry = xy

Chapter IV

DECIMAL CODES

The considerations involved in choosing a representation for a decimal digit in a calculator are considerably more complex than was the case with binary digits. With the binary system the only significant choices were a signal or no signal, a signal on one of two different lines, or a "positive" or a "negative" signal to represent a 0 or 1, respectively. In the decimal system a single digit may have any one of ten different values, and one of the more obvious ways of representing a decimal digit is through the use of signals having the possibility of ten different amplitudes. A considerable amount of thought has been given to decimal digit representations of this type, but very little progress has been made in the adaptation of multiple-amplitude signals to digital calculators. Consequently it has been necessary to look for other means of digit representations.

Since most calculator components are inherently binary in nature or else work most satisfactorily when employed in binary fashion, it has been almost universal practice to use binary-type signals to represent the decimal digits, and there are a great variety of ways by which this may be accomplished. One way is to use ten separate lines and to place signals on a number of the lines corresponding to the digit value. For example, signals on six of the lines would indicate the digit, 6. It is more common practice to specify that only one signal at a time will be placed on any of the ten lines, and then each of the ten lines is identified with one of the ten digits. This representation is frequently used in transmitting decimal digits from a manual keyboard to a calculator. Also, binary components may be used to transmit a decimal digit on a single line if time is employed as a variable. For example, the duration of a signal may be varied in ten equal steps to represent the ten decimal digits, or the signal may be

repeated a number of times equal to the value of the digit. Another method is to use the timing of a single pulse-type signal.

Although the various "one-out-of-ten" schemes for digit representation which have been mentioned are relatively simple and straightforward, other forms of representation frequently are found to be more desirable in the design of calculators where factors such as speed, savings in components, and reliability are important. It has already been mentioned that it is possible to represent a decimal digit with a minimum of four binary signals by using ten and ignoring six of the possible sixteen different combinations of the four signals. But of the over 29 billion (16!/6!) different ways by which ten of the sixteen combinations can be assigned to the ten decimal digits, only a few of the ways are useful and the problem of selecting the best way for a given application is frequently a formidable one. The problem is further complicated by the fact that the use of more than the minimum of four binary signals may provide means for gaining important advantages in some applications. A particular way of assigning groups of four, five, or more binary signals to the ten decimal digits is called a decimal code, and it is the object of this chapter to point out the distinguishing features and some of the advantages and disadvantages of several of the codes which may be used.

The Weighted Four-Bit Codes. Of the many possible four-bit (where a "bit" refers to a binary digit) codes only a relatively few have the property that values, or weights, can be assigned to each of the four bits with the decimal digit being represented equal to the sum of the weights. All known weighted four-bit codes are listed in Table V, although the listing is more for the record than for any practical value. Since the codes were found by a cut-and-try search process, it cannot be guaranteed

that all such codes have been found.

*5211	8421	631-2	842-3	74-2-1
*4311	531-1	*731-2	621-4	84-2-1
5311	*631-1	441-2	721-4	72-3-1
6311	522-1	541-2	821-4	72-4-1
*4221	*622-1	*641-2	*751-4	84-3-2
5221	432-1	841-2	861-4	*87-4-2
6221	*532-1	*632-2	632-4	
*3321	632-1	*443-2	*832-4	
4321	732-1	543-2	*652-4	
5321	*442-1	643-2	653-4	
6321	542-1	843-2	643-5	
7321	642-1	621-3	*753-6	
4421	742-1	721-3	63-1-1	
5421	842-1	751-3	63-2-1	
6421	621-2	542-3	54-2-1	
7421	531-2	*642-3	64-2-1	

Table V. Weighted Four-Bit Codes

To illustrate the meaning of the listings in Table V, three of the more useful codes and one code involving a negative weight are displayed in detail in Table VI.

	8421	2421	5421	753-6
0	0000	0000	0000	0000
1	0001	0001	0001	1001
2	0010	0010	0010	0111
3	0011	0011	0011	0010
4	0100	0100	0100	1011
5	0101	1011	1000	0100
6	0110	1100	1001	1101
7	0111	1101	1010	1000
8	1000	1110	1011	0110
9	1001	1111	1100	1111

Table VI. Detailed Listing of Some of the Four-Bit Weighted Codes

The 8, 4, 2, 1 code is one of the most straightforward four-bit codes because with it each decimal digit is represented in a conventional binary system. Therefore the code has the advantage that the relatively simple binary techniques may be used, to some degree at least, in the arithmetic manipulations involving

decimal digits. A disadvantage of the code is that the binary representations for ten to fifteen inclusive, have no meaning and steps must be taken to eliminate or correct these binary combinations each time they occur in an arithmetic operation. Another disadvantage of the code is that it is not "self-complementing," where a self-complementing decimal code is one where the 9's complement of each decimal digit may be obtained by changing the 1's to 0's and the 0's to 1's in the coded representation of the digit. Since a simple inversion yields the 15's complement, it is necessary in obtaining the 9's complement to add 10 to the result obtained by inversion or to add 6 to the digit before inversion. In the first of the two methods a carry from the "8's" order will be obtained, but this carry is ignored, which effectually subtracts 16 from the result. That the two processes yield the 9's complement may be illustrated mathematically by the equation,

$$(15 - D) + 10 - 16 = 15 - (D + 6) = 9 - D,$$

where D is the decimal digit.

When the bits of the 8, 4, 2, 1 code are presented in parallel, a somewhat simpler scheme than adding and inverting may be used to generate the 9's complement. By an examination of the code in Table VI it may be observed that the 1-bit should always be inverted in generating the 9's complement, the 2-bit is always the same in the 9's complement as in the original digit; the 4-bit in the 9's complement is a 1 when the 2-bit or the 4-bit, but not both, in the original digit is 1; and the 8-bit in the complement is 0 when 2-bit, 4-bit or the 8-bit in the original digit is 1. In Boolean algebra notation these relationships are

$$1_c = \overline{1}$$

$$2_c = 2$$

$$4_c = (2)(\overline{4}) + (\overline{2})(4)$$

$$8_c = \overline{2 + 4 + 8}$$

where the subscript refers to the complement. The functional arrangement is shown in Fig. 4-1.

Those codes in Table V which are marked with an asterisk do have the property of being self-complementing, that is, the changing of the 0's to 1's and the 1's to 0's in each representation of a decimal digit will yield the 9's complement of that digit. Note that the sum of the weights in each self-complementing code is nine. That a sum of nine is a requirement may be easily understood by observing that 0000 must be a representation of the decimal digit, zero, in any of the weighted codes, and therefore 1111 must be a representation for the decimal digit, nine.

The 4, 2, 2, 1 code, which is self-complementing, is shown in Table VI although the first two columns have been interchanged to indicate a 2, 4, 2, 1 arrangement to correspond to common practice with this particular code. With the 2, 4, 2, 1 code the representations for the decimal digits two to seven, inclusive, are not necessarily unique. For example, either 0011 or 1001 may be used to indicate three, but the self-complementing feature is not effected because 1100 and 0110 are both proper representations of six. The self-complementing codes with only positive weights are useful also when changing from a four-bit code to a single-line decimal code with the true or complement representation of the decimal digit being indicated by the number of pulse-type signals appearing on the line. Fig. 4-2(a) shows an electrical circuit utilizing relay contacts, which may be used to perform the conversion and Fig. 4-2(b) shows the functional arrangement. Although the self-complementing weighted codes, particularly the 2,4,2,1 code, have been used in a few calculators, they create difficulties when attempting simple arithmetic operations such as addition, and their use has therefore not been widespread.

The 5, 4, 2, 1 code is included in Table VI mainly for the purpose of

comparison with the 2, 4, 2, 1 code, although it is not without practical application. Note that the first bit in each code is 1 for the decimal digits five through nine, but in the 5, 4, 2, 1 code the other three bits are the same for the five through nine as for zero through four, which is not the case in the 2, 4, 2, 1 code. The usefulness of the 5, 4, 2, 1 code may be found in certain multiplication and division systems which employ only halving and doubling. The 2 and 4 factors may be obtained by doubling and the 5 factor may be obtained by multiplying by 10 (shifting) and halving.

The study of codes with negative weights can be a source of great fascination; however, no advantages of them deemed worth recording have as yet been found.

Non-Weighted Four-Bit Codes. For some applications rather unusual considerations may assume a role of magnified importance with the effect that one of the non-weighted codes may be the best choice. For example, it may be desired to minimize the amount of power required to store or transmit the digits, and with an objective such as this, a code with as few 1's as possible may be desirable. The 8, 4, 2, 1 code has a total of fifteen 1's and by using, say 1010 instead of 0111 for the digit 7, the number of 1's may be reduced to fourteen, which is minimum possible with a 4-bit code. It so happens that the 7, 4, 2, 1 weighted code also has only fourteen 1's. For another example, it may be desired that all decimal digits including zero be represented by at least one 1 so that the absence of a signal can be detected positively. Again, it is possible to use a weighted code such as the 5, 3, 1, -1 code for the purpose where zero is 0011, but if the weighted properties are not useful one of the non-weighted codes may be better. Still another example is in the use of a four-bit code for storage when the code used for calculations is a 5-bit code. With 5 bits there are ten different combinations with two 1's, and when the ten decimal

digits are represented in this manner it is possible to distinguish any digit without the use of inverters, which is an advantage. However, only four of the five bits are really necessary for unambiguous representation so by dropping one of the five digits, a non-weighted four-bit code for storage is obtained. The fifth bit may be generated when needed. One variation of the scheme is shown in Table VII. This variation is sometimes called the 7, 4, 2, 1, 0 code because, except for zero, the bits have these weights.

	2 out of 5 code 74210	code as stored
0	11000	1100
1	00011	0001
2	00101	0010
3	00110	0011
4	01001	0100
5	01010	0101
6	01100	0110
7	10001	1000
8	10010	1001
9	10100	1010

Table VII. A five-bit code reduced to a four-bit code for storage.

Although all of the ideas presented in this paragraph have been seriously proposed at one time or another and may have actually been used in a few instances, it should be understood that the difficulties encountered when attempting to use the codes in a calculating device usually offset the advantages which have been mentioned.

One non-weighted code of more importance is called the excess-3 code because it may be generated by adding a binary 3 to each digit representation in the conventional 8, 4, 2, 1 code. Table VIII shows the excess-3 code in detail.

0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Table VIII. Excess-3 Code

The excess-3 code has several useful properties. First of all, except for certain corrections which must be applied, straight binary techniques may be used in the performance of many of the arithmetic operations involving the digits. When adding two digits the decimal carry is readily generated by using the carry from the highest order binary digits. That the carry may be obtained in this manner may be understood by observing that when two excess-3 digits are added the sum is excess-6, which automatically eliminates the six unwanted binary configurations. Further advantages of the code are that it is self-complementing and that all decimal digits have at least one 1 in their representation so that zero and the condition of no digit at all may be distinguished. On the other hand, the fact that the excess-3 code is not weighted frequently introduces considerable disadvantages. For example, it is more difficult to learn and remember than the 8, 4, 2, 1 and other weighted codes. Certain forms of arithmetic operations are more difficult with the excess-3 codes than with other codes including the frequently required operation of conversion between a four-bit code and one of the various one-out-of-ten systems of representation. Also, in some calculators a redundancy bit is used for checking purposes, and when this is done the advantage of the excess-3 code with regard to the representation for zero is largely nullified.

Codes Involving Five or More Bits. One reason it might be desirable to use five or more bits in the representation of a decimal digit in spite of the availability of four-bit codes is that it may be possible to effect simplifications in the arithmetic circuits in some cases. The use of a five-bit code with each decimal digit represented by two 1's in the five bits has already been mentioned as providing an improved means for sensing the individual digits. When addition is being performed by switching circuits, the 8, 6, 4, 2, 1 or the 5, 4, 3, 2, 1, 0 weighted codes offer certain advantages. All three of these codes have been employed in calculators built at Harvard University; however, the use of codes with more than four bits for the purpose of circuit simplification has not become widespread.

A second advantage that can be gained by using five or more bits in the representation of decimal digits is the ability to detect errors. In the code involving two 1's out of five bits, for example, the existence of three 1's or only one 1 in the representation of a digit would be recognized as an error. Another code, known as the bi uinary code, has seven bits with the weights of 5, 0, 4, 3, 2, 1, 0. With this code, arithmetic operations may be performed in a moderately straightforward manner, although whether or not there is a net simplification when compared with the 4-bit codes is a debatable point. The main reason for the use of seven bits is the ability to detect errors. From the detailed listing of the code in Table IX it may be observed that one of the first two bits and one of the last four bits is 1 in the representation of each decimal digit.

	5043210
0	0100001
1	0100010
2	0100100
3	0101000
4	0110000
5	1000001
6	1000010
7	1000100
8	1001000
9	1010000

Table IX. Bi quinary Code

Error-Detecting and Error-Correcting Codes in General. In any code composed of binary bits, if a single error in a bit combination can produce another bit combination which is also in the code scheme, the error cannot, in general, be detected. For example, in the 8, 4, 2, 1 code 0110 (decimal digit 6) may appear as the proper representation for six, but if there were an error and it should be 0111 (decimal digit 7) or 0010 (decimal digit 2), there would be no way of detecting from the coded representation itself that the bits were an erroneous representation of some other digit. In order to detect the presence of a single error in the bits of a code it is necessary that the code be such that at least two changes must be made in the bits of the code when changing from the representation of one digit to the representation of any other digit. The 2-out-of-5 code shown in Table VII is an example of a code satisfying this requirement. To change from the representation of decimal 3 to decimal 8, for example, it is necessary to change both the first and the third bits of the code. The changing of any one of the bits in any of the code combinations will result in a combination which can be recognized as an error, and the code is therefore known as "error-detecting." However, the detection of an error does not mean that it can be corrected. For example, the bit combination, 11010,

would be recognized as an error because there are three instead of only two 1's, but there would be no way of knowing from the code itself which of the three 1's should be a 0. The code is therefore not "error-correcting." For similar reasons the bi uinary code in Table IX is error-detecting but not "error-correcting." If two or more errors occur simultaneously with either of the error-detecting codes which have been mentioned, the errors may pass undetected because the result of the errors may be the production of a bit combination which is a proper representation of one of the digits.

To be error-correcting, a code must be such that at least three changes in the bit combination must be made when changing from the representation of one digit to the representation of any other digit. With a code meeting this requirement an error will produce a bit combination which can be recognized to contain an error, as before, but further, the individual bit in error can be determined. The finding and correcting of the incorrect bit can be accomplished through changing the bits one at a time and observing when a bit combination is obtained that is a correct representation of one of the digits. When two errors occur simultaneously the resulting bit combination will be recognized as not corresponding to any digit, but the changing of one bit may produce a bit combination which corresponds to one of the digits, but not the desired digit. Therefore, the error-correcting code will fail with the occurrence of two errors in the representation of a given digit.

By using a code requiring four changes in the bits when changing from the representation of one digit to the representation of any other digit, "double-error-detecting" properties may be obtained. With a code such as this, two simultaneous errors will produce a bit combination which not only may be recognized as not corresponding to any digit, but also may not be changed to any digit-representing combination

by the change of any one bit. Therefore, the code is capable of producing an indication that two errors have occurred in a given digit representation, and the ability to correct a single error is not lost. However, it is still not possible to correct two simultaneous errors because the alteration of two bits can yield a bit combination corresponding to one of the digits but not necessarily the desired one. Also, the occurrence of three or more simultaneous errors may cause a failure in the error detecting and correcting scheme.

"Double-error-correcting," and "triple-error-detecting," and more powerful schemes may be devised through the use of codes requiring still more changes in the changing of the representations of the various digits although the number of bits required for the code soon reaches an impractical value. For the error-detecting, error-correcting, and double-error-detecting codes a minimum of five, seven, and eight bits, respectively, are necessary.

While the concepts presented in this section are of interest in the understanding of the nature of error-detecting and error-correcting codes, these concepts have not been found to be of much value in the devising of useful codes. The concept of redundancy, which is discussed in the next section, appears to be a more useful tool for the development of practical codes for the detection and correction of errors.

Redundancy Checks. Among the more obvious ways of checking for errors in digit transmission is the transmission of each digit twice. If there is a discrepancy in the two transmissions, it may be concluded that an error has occurred. By transmitting the digit three times it is possible not only to detect the error but also to correct it because two of the three transmissions should be the same, unless two or more errors have occurred, in which case the method fails. More powerful

checks may be secured through additional transmissions and comparisons for e digit but the increased complexity which would be required in the equipment causes the scheme to be unattractive.

The above method of checking involves the use of redundant information. Since the purpose of the redundant information is only to check and correct errors in the original information, it is possible to use less redundant information than is necessary for the complete duplication of the transmissions. For example, a coded decimal digit is represented by a set of signals indicating a certain configuration of 0's and 1's, and the configuration has elementary properties which will be changed in the presence of an error. Therefore, in order to check for an error it is sufficient to use one of these elementary properties for the redundant information. The number of 1's is one such property that may be used, but a simpler property and one that is well adapted to the binary nature of the signals which are usually employed is the fact that the number of 1's is either odd or even. Then, if an extra bit is transmitted along with the coded representation of the digit, the extra bit may be used to indicate whether the number of 1's is odd or even and a discrepancy in the indication will signify that an error has occurred. This extra bit is redundant information because it may be derived directly from the coded digit.

In order to use redundancy bits for the correction as well as detection of errors it is necessary to devise a system whereby discrepancies in one or more of the odd-even checks can be identified with individual bits in the code. One arrangement which may be used incorporates the checking of several digits into a single operation. Each digit is provided with a check bit and also each "column" of bits in the several digits is provided with a check bit. If any bit in the digit representing part of the code is in error, two of the checks will fail and the error

may be located in cartesian coordinate fashion. The decimal number, 69,073 in the 8, 4, 2, 1 code is shown with its check bits in Table X. Note that the check bit

		8 4 2 1 C
Decimal Number	6	0 1 1 0 1
	9	1 0 0 1 1
	0	0 0 0 0 1
	7	0 1 1 1 0
	3	0 0 1 1 1
	C	0 1 0 0 0

Table X. Redundancy Bits for Error Correction.

is 1 whenever the number of 1's in the bits being checked is even. The opposite convention could have been used although the digit, zero, would have no 1's in its representation which, as has already been mentioned is sometimes undesirable. If only one check fails, it may be assumed that the check bit itself is in error unless more than one error has occurred. By including a check on the check bits (the bit in the lower right-hand corner of Table X) the checking system becomes double-error detecting as well as single-error-correcting although the complications required to make use of this feature become considerable.

By using three redundancy bits for each digit represented with a code of four bits it is possible to develop a digit-by-digit error correcting code. One variation of the method when applied to the 8, 4, 2, 1 code is to use one redundancy bit (A) to check the 1's in the 4, 2, and 1-bits; a second redundancy bit (B) for the 8, 2, and 1-bits; and a third redundancy bit (C) for the 8, 4, and 1-bits. The pattern and the 8, 4, 2, 1 code with its redundancy bits are shown in detail in Table XI.

8	4	2	1	A	B	C
	A	A	A	A		
B		B	B		B	
C	C		C			C

	8	4	2	1	A	B	C
0	0	0	0	0	1	1	1
1	0	0	0	1	0	0	0
2	0	0	1	0	0	0	1
3	0	0	1	1	1	1	0
4	0	1	0	0	0	1	0
5	0	1	0	1	1	0	1
6	0	1	1	0	1	0	0
7	0	1	1	1	0	1	1
8	1	0	0	0	1	0	0
9	1	0	0	1	0	1	1

Table XI. The 8, 4, 2, 1 Code With Error-Correcting Redundancy Bits.

From Table XI it may be observed that the location of any error may be determined by the combination of checks which fail. If, for example, checks B and C fail, the error must be in the 8-bit. If all three checks fail the error is in the 1-bit, or, if only one check fails, the error is in the redundancy bit itself. The occurrence of two errors in the representation of a single digit will cause a failure in the system although a fourth check bit (D) for indicating the 1's in all four of the original digit positions could be used to make the code double-error detecting. Note that the coded representation for each decimal digit differs from the representation of each of the other digits by at least three bits, which is in line with the discussion in the preceding section.

Four redundancy bits, A, B, C, and D may be used to generate an error-correcting code involving eleven information-carrying bits by following the pattern shown in Table XII.

1024	512	256	128	64	32	16	8	4	2	1	A	B	C	D
	A	A	A	A				A	A	A	A			
B		B	B		B		B		B	B		B		
C	C		C			C	C	C		C			C	
D	D	D	D	D	D	D								D

Table XII. Pattern for an Error-Correcting Code Involving Eleven Information-Carrying Bits.

Again, the bit in error can be ascertained by the combination of checks which fail. For example, if checks A, B, and D fail, the 256-bit is in error. In general, n redundancy bits may be used to form an error-correcting code involving $2^n - n - 1$ information-carrying bits.

In most instances, redundancy bits are useful only in the checking of the transmission of digits. Calculations involving the digits usually must be checked by some other means, although there are a few schemes for calculating the new redundancy bits which are necessary after arithmetic operations have been performed on the digits. For digits which are the results of calculations, the simple generation of the new redundancy bits according to the pattern in use affords no checks at all in the calculations.

Many of the ideas which have been presented on the subject of error-detecting and error-correcting codes may be found in the text "The Design of Switching Circuits" by Keister, Ritchie and Washburn (D. Van Nostrand Co., Inc., New York 1951).

Tape Codes. The various codes used in punched paper tape systems were not usually selected with any consideration being given to the employment of the tapes in calculators. However, in many cases it has been found expedient to use previously established punched paper tape systems for the input and output mechanisms of calculators. For this reason the Bell System 5-hole teletype code and the IBM 8-hole code are presented in Tables XIII and XIV, respectively. A "1" represents a hole and a "0" represents no hole in the tape. In most cases, it is desirable to convert the code to one which is more adaptable to calculations when employed in the calculator and to use the tape codes only in the tape handling mechanisms.

11	000	A		11	101	Q	1
10	011	B		01	010	R	4
01	110	C		10	100	S	
10	010	D		00	001	T	5
10	000	E	3	11	100	U	7
10	110	F		01	111	V	
01	011	G		11	001	W	2
00	101	H		10	111	X	
01	100	I	8	10	101	Y	6
11	010	J		10	001	Z	
11	110	K		00	100	Space	
01	001	L		00	010	Carriage Return	
00	111	M		01	000	Line Feed	
00	110	N		11	111	Shift to Letters	
00	011	O	9	11	011	Shift to Figures	
01	101	P	0				

Table XIII. Bell System Teletype Tape Code

001	00000	0
000	11100	1
000	11000	2
000	10000	3
000	01000	4
000	10010	5
000	01010	6
000	00010	7
000	00100	8
000	01110	9

Table XIV. IBM 8-Hole Code for Decimal Digits (Alphabetic and Special Characters and Various Machine Operations are Included in the Code, but Are Not Presented Here.)

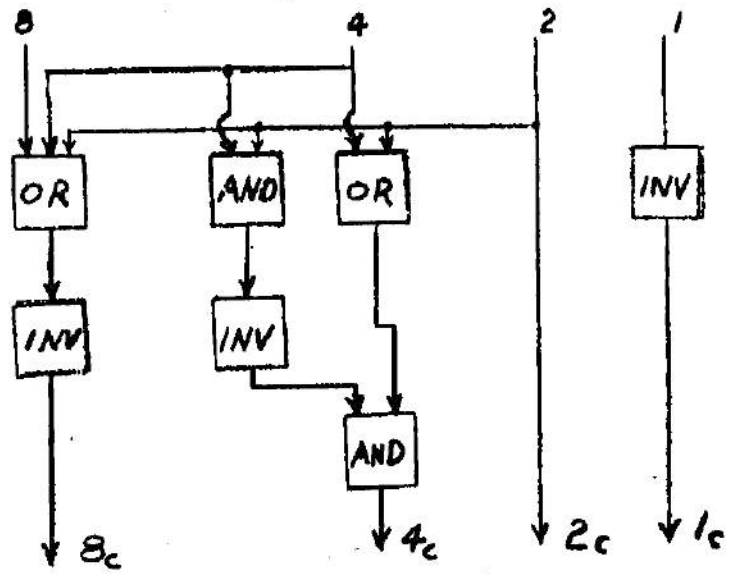

Fig. 4-1. Functional arrangement for generating the 9's complement with the 8,4,2,1 code.

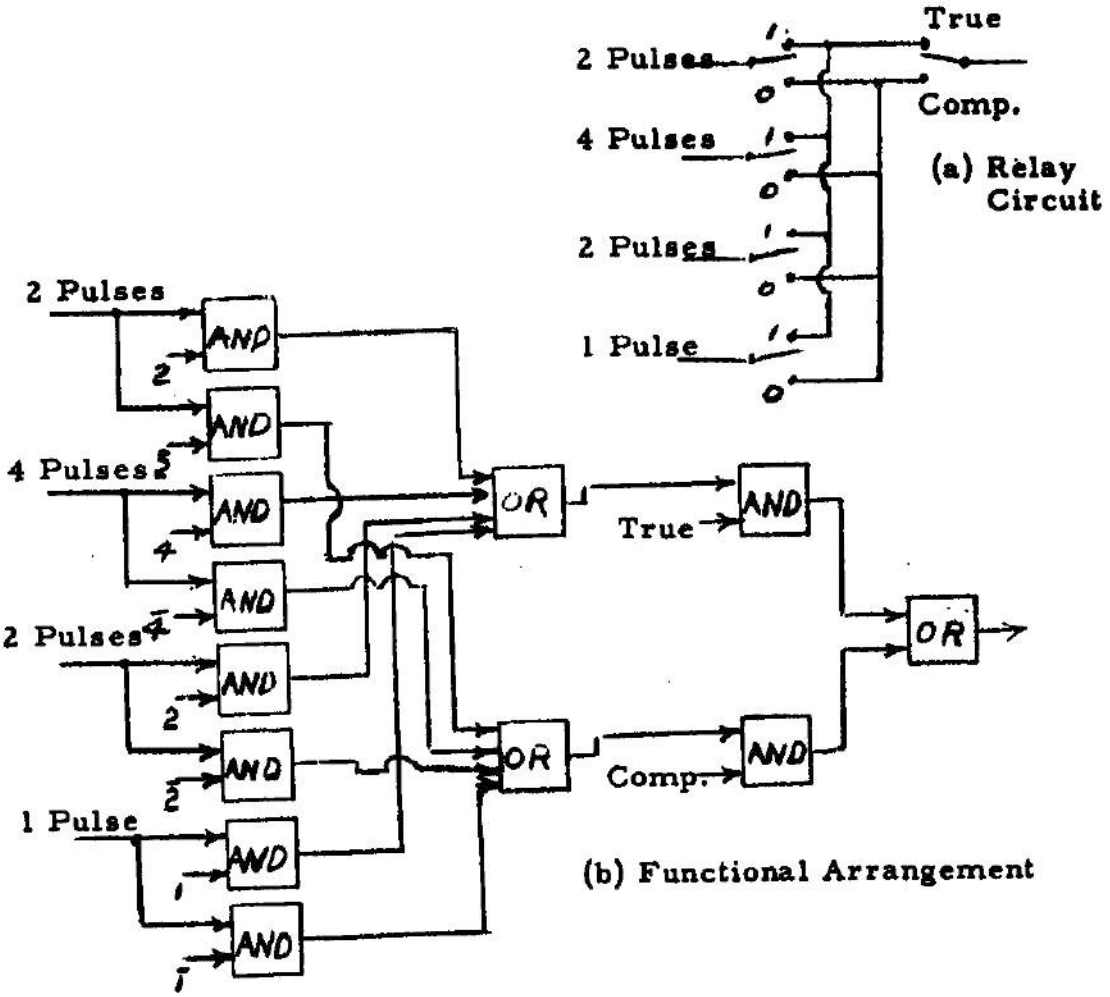

Fig. 4-2. Conversion from parallel 4-bit code to a series of pulses on a line.