CSCI 3080 Binary Encoding Schemes (This is from Switching and Finite Automata Theory by Zvi Kohavi, McGraw Hill, 1978, pp. 10 - 20. There have been a few very slight modifications made by me.)

Although the binary number system has many practical advantages and is widely used in digital computers, in many cases it is convenient to work with the decimal number system, especially when the communication between man and machine is extensive, since most numerical data generated by man are in terms of decimal numbers. To simplify the communication problem between man and machine, a number of codes have been devised so that the decimal digits are represented by sequences of binary digits.

Weighted codes

In order to represent the 10 decimal digits 0,1,...,9 it is necessary to use at least four binary digits. Since there are 16 combinations of four binary digits of which only 10 combinations are used, it is possible to form a very large number of distinct codes.Of particular importance is the class of weighted codes, whose main characteristic is that each binary digit is assigned a weight, and for each group of four bits, the sum of the weights of those binary digits whose value is 1 is equal to the decimal digit which they represent.In other words, if w_1, w_2, w_3, and w_4 are the weights of the binary digits and x_1, x_2, x_3, and x_4 are the corresponding digit values, then the decimal digit $N=w_4x_4 + w_3x_3 + w_2x_2 + w_1x_1$ is represented by the binary sequence $x_4x_3x_2x_1$. A sequence of binary digits which represents a decimal digit is called a code word. Thus the above sequence $x_4x_3x_2x_1$ is the code word for N. A number of weighted, four digit binary codes are shown in the following table.

Decimal digit	8	4	2	1	2	4	2	1	6	4	2	-3
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	1	0	1
2	0	0	1	0	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1	1	0	0	1
4	0	1	0	0	0	1	0	0	0	1	0	0
5	0	1	0	1	1	0	1	1	1	0	1	1
6	0	1	1	0	1	1	0	0	0	1	1	0
7	0	1	1	1	1	1	0	1	1	1	0	1
8	1	0	0	0	1	1	1	0	1	0	1	0
9	1	0	0	1	1	1	1	1	1	1	1	1

The binary digits in the first code in the above table are assigned the weights 8, 4, 2, 1. As a result of this weight assignment, the code word that corresponds to each decimal digit is the binary equivalence of that digit; e.g., 5 is represented by 0101, and so on. This code is known as the BCD (binary-coded-decimal) code. For each of the codes in the table, the decimal digit that corresponds to a given code word is equal to the sum of the weights in those binary positions which are 1s. Thus, in the second code, where the weights are 2, 4, 2, 1, decimal 5 is represented by 1011, corresponding to the sum 2*1 + 4*0 + 2*1 + 1*1 = 5. The weights assigned to the binary digits may also be negative, as is shown by the code (6,4,2,-3). In this code decimal 5 is represented by 1011, since 6*1 + 4*0 + 2*1 - 3*1 = 5.

It is apparent that the representations of some decimal numbers in the (2,4,2,1) and (6,4,2,-3) codes are not unique. For example, in the (2,4,2,1) code, decimal 7 may be represented by 1101 as well as by 0111. Adopting the representations shown in the above table causes the codes to become self-complementing. A code is said to be self-complementing if the code word of the 9s complement of N, i.e., 9-N, can be obtained from the code word of N by interchanging all the 1s and 0s. For example, in the (6,4,2,-3) code, decimal 3 is represented by 1001, while decimal 6 is represented by 0110. In the (2,4,2,1) code, decimal 2 is represented by 0010, while decimal 7 is represented by

1101. Note that the BCD code is not self-complementing. It can be shown that a necessary condition for a weighted code to be self-complementing is that the sum of the weights must equal 9. There exist only four positively weighted self-complementing codes, name, (2,4,2,1), (3,3,2,1), (4,3,1,1), (5,2,1,1). In addition, there exist 13 self-complementing codes with positive and negative weights.

Nonweighted codes

There are many nonweighted binary codes, two of which are shown in the following table. The Excess-3 code is formed by adding 0011 to each BCD code word. Thus, for example, the representation of decimal 7 in Excess-3 is given by 0111 + 0011 = 1010. The Excess-3 code is a self-complementing code, and it possesses a number of properties that made it practical in earlier decimal computers.

Decimal Digit	Excess-3	Cyclic
0	0011	0000
1	0100	0001
2	0101	0011
3	0110	0010
4	0111	0110
5	1000	1110
6	1001	1010
7	1010	1000
8	1011	1100
9	1100	0100

In many practical applications, e.g., analog-to-digital conversion, it is desirable to use codes in which all successive code words differ in only one digit. Codes that have such a property are referred to as cyclic codes. The second code in the above table is an example of such a code. (Note that in this, as in all cyclic codes, the code word representing the decimal digits 0 and 9 differ in only one digit.) A particularly important cyclic code is the Gray code. A four-bit Gray code is shown in the following table.

Decimal Number	Gray	Binary
0	0000	0000
1	0001	0001
2	0011	0010
3	0010	0011
4	0110	0100
5	0111	0101
6	0101	0110
7	0100	0111
8	1100	1000
9	1101	1001
10	1111	1010
11	1110	1011
12	1010	1100
13	1011	1101

14	1001	1110
15	1000	1111

The feature that makes this cyclic code useful is the simplicity of the procedure for converting from the binary number system into the Gray code.

Let $g_n...g_2g_1g_0$ denote a code word in the (n+1)st-bit Gray code, and let $b_n...b_2b_1b_0$ designate the corresponding binary number, where the subscripts 0 and n denote the least significant and most significant digits, respectively. Then, the ith digit g_i can be obtained from the corresponding binary number as follows:

$$g_i = b_i \oplus b_{i+1} \quad 0 \leq I \leq n - 1$$
$$g_n = b_n$$

where the symbol $\oplus$ denotes the modulo-2 sum, which is defined as follows:

$$0 \oplus 0 = 0 \quad 0 \oplus 1 = 1 \quad 1 \oplus 0 = 1 \quad 1 \oplus 1 = 0$$

For example, the Gray code word which corresponds to the binary number 101101 is found to be 111011 in the following manner:

$$1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1$$
$$b_5 \; b_4 \; b_3 \; b_2 \; b_1 \; b_0$$

Using the above rules,

$g_5 = b_5 = 1$ $\quad g_2 = b_3 \oplus b_2 = 0$

$g_4 = b_5 \oplus b_4 = 1$ $\quad g_1 = b_2 \oplus b_1 = 1$

$g_3 = b_4 \oplus b_3 = 1$ $\quad g_0 = b_1 \oplus b_0 = 1$

resulting in the following gray code word.

$$1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1$$
$$g_5 \; g_4 \; g_3 \; g_2 \; g_1 \; g_0$$

To convert from Gray code to binary, start with the leftmost digit and proceed to the least significant digit, making $b_i = g_i$ if the number of 1s preceding g_i is even, and making $b_i = g_i'$ if the number of 1s preceding g_i is odd. (Note that zero 1s is an even number of 1s.) For example, the (Gray) code word 1001011 represents the binary number 1110010. The proofs that the preceding conversion procedures indeed work are left to the reader.

The n-bit Gray code is a member of a class called reflected codes. The term ″reflected″ is used to designate codes which have the property that the n-bit code can be generated by reflecting the (n-1)st-bit code, as illustrated here (left to right is the 1 bit, the two bit, the three bit, and the four bit codes.

0	0 0	0 00	0 000
1	0 1	0 01	0 001
	1 1	0 11	0 011
	1 0	0 10	0 010
		1 10	0 110
		1 11	0 111
		1 01	0 101
		1 00	0 100
			1 100
			1 101
			1 111
			1 110
			1 010
			1 011
			1 001
			1 000

ERROR DETECTION AND CORRECTION

In the codes presented so far, each code word consists of four binary digits, which is the minimum number needed to represent the 10 decimal digits. Such codes, although adequate for the representation of the decimal digits, are very sensitive to transmission errors that may occur because of equipment failure or noise in the transmission channel. In any practical system there is always a finite probability of the occurrence of a single error. The probability that two or more errors will occur simultaneously, although nonzero, is substantially smaller. We therefore restrict our discussion mainly to the detection and correction of single errors..

error-detecting codes

In a four-bit binary code, the occurrence of a single error in one of the binary digits may result in another, incorrect but valid, code word. For example, in the BCD code, if an error occurs in the least significant digit of 0110, the code word 0111 results, and since it is a valid code word, it is incorrectly interpreted by the receiver. If a code possesses the property that the occurrence of any single error transforms a valid code word into an invalid code word, it is said to be a (single)-error-detecting code. Two error-detecting codes are shown in the following table.

Decimal Digit	Even-parity BCD 8421p	2-out-of-5 01247
0	00000	00011
1	00011	11000
2	00101	10100
3	00110	01100
4	01001	10010
5	01010	01010
6	01100	00110

7	01111	10001
8	10001	01001
9	10010	00101

The error detection in either of the codes of the above table is accomplished by a parity check. The basic idea in the parity check is to add an extra digit to each code word of a given code, so as to make the number of 1s in each code word either odd or even. In the codes of the above table we have used even parity. The even-parity BCD code is obtained directly from the BCD code shown previously. The added bit, denoted p, is called parity bit. The 2-out-of-5 code consists of all 10 possible combinations of two 1s in a five-bit code word. With the exception of the code word for decimal 0, the 2-out-of-5 code is a weighted code and can be derived from the (1,2,4,7) code.

In each of the codes in the above table, the number of 1s in a code word is even. Now, if a single error occurs, it transforms the valid code word into an invalid one, thus making the detection of the error straightforward. Although the parity check is intended only for the detection of single errors, it in fact detects any odd number of errors and some even number of errors. For example, if the code word 10100 is received in an even parity BCD message, it is clear that the message is erroneous, although the parity check is satisfied. We cannot determine, however, the original transmitted word.

In general, to obtain an n-bit error-detecting code, no more than half of the possible 2^n combinations of digits can be used. The code words are chosen in such a manner that, in order to change one valid code word into another valid code word, at least two digits must be complemented. In the case of four-bit codes, this constraint means that only 8 valid code words can be formed of the 16 possible combinations. Thus, to obtain an error-detecting code for the 10 decimal digits, at least 5 binary digits are needed. It is useful to define the distance between two code words as the number of digits that must change in one word so that the other word results. For example, the distance between 1010 and 0100 is three, since the two code words differ in three bit positions. The minimum distance of a code is the smallest number of bits in which any two code words differ. Thus the minimum distance of the BCD or the Excess-3 codes is one, while that of the codes in the above table is two. Clearly, a code is an error-detecting code if and only if its minimum distance is two or more.

Error-correcting codes

For a code to be error-correcting, its minimum distance must be further increased. For example, consider the three-bit code which consists of only two valid code words, 000 and 111. If a single error occurs in the first code word, it can be changed to 001, 010, or 100. The second code word can be changed due to a single error to 110, 101, or 011. Note that in each case the invalid code words are different. Clearly, this code is error-detecting, since its minimum distance is three. Moreover, if we assume that only a single error can occur, then this error can be located and corrected, since every error results in an invalid code word that can be associated with only one of the valid code words. Thus the two code words 000 and 111 constitute an error-correcting code whose minimum distance is three. In general, a code is said to be an error-correcting code if the correct code word can always be deduced from the erroneous word. In this section we shall discuss a single type of error-correcting codes, known as the Hamming codes.

If the minimum distance of a code is three, then any single error changes a valid code word into an invalid one, which is a distance one away from the original code word and a distance two from any other valid code word. Therefore, in a code with minimum distance of three, any single error is correctable or any double error detectable. Similarly, a code whose minimum distance is four may be used for either single-error correction and double-error detection or triple-error detection. The key to error correction is that it must be possible to detect and locate erroneous digits. If the location of an error has been determined, then by complementing the erroneous digit the message is corrected.

The basic principles in constructing a Hamming error-correcting code are as follows. To each group of m information, or message, digits, k parity checking digits, denoted $p_1, p_2, \ldots p_k$, are added to form an (m+k)-digit code. The location of each of the m+k digits within a code word is assigned a decimal value, starting by assigning a 1 to the most significant digit and m+k to the least significant digit. k parity checks are performed on selected digits of each code word. The result of each parity check is recorded as 1 or 0, depending, respectively, on whether an error has or has not been detected. These parity checks make possible the development of a binary number, $c_1\, c_2 \ldots c_k$ whose value when an error occurs is equal to the decimal value assigned to the location of the erroneous digit, and is equal to zero if no error occurs. This number is called the position (or location) number.

The number k of digits in the position number must be large enough to describe the location of any of the m+k possible single errors, and must in addition take on the value zero to describe the ″no error″ condition. Consequently, k must satisfy the inequality $2^k \geq m+k+1$. Thus, for example, if the original message is in BCD, where m = 4, then k = 3 and at least three parity checking digits must be added to the BCD code. The resultant error-correcting code thus consists of seven digits. In this case, if the position number is equal to 101, it means that an error has occurred in position 5. If, however, the position number is equal to 000, the message is correct.

In order to be able to specify the checking digits by means of only message digits and independently of each other, they are placed in positions $1,2,4,\ldots,2^{k-1}$. Thus, if m = 4 and k = 3, the checking digits are placed in positions 1,2, and 4, while the remaining positions contain the original (BCD) message bits. For example, in the code word **11**0**0**110 the parity digits (in boldface) are $p_1 = 0$, $p_2 = 1$, $p_3 = 1$, while the message digits are 0,1,1,0, which correspond to decimal 6.

We shall now show how the Hamming code is constructed, by constructing the code for m=4 and k=3. As discussed above, the parity checking digits must be so specified that, when an error occurs, the position number will take on the value assigned to the location of the erroneous digit. The following table lists the seven error positions and the corresponding values of the position number.

Error Position	Position number $c_1\, c_2\, c_3$
0 (no error)	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

It is evident that if an error occurs in position 1, or 3, or 5, or 7, the least significant digit, i.e., c_3, of the position number must be equal to 1. If the code is constructed so that in every code word the digits in positions 1,3,5, and 7 have even parity, then the occurrence of a single error in any one of these positions will cause an odd parity. In such a case the least significant digit of the position number is recorded as 1. If no error occurs among these digits, the parity check will show an even parity, and the least significant digit of the position number is recorded as 0.

From the above table we observe that an error in position 2, or 3, or 6, or 7 should result in the recording of a 1 in the center of the position number. Hence the code must be designed so that the digits in positions 2,3,6, and 7 have even parity. Again, if the parity check of these digits shows an odd parity, the corresponding position number digit, i.e., c_2, is set to 1; otherwise it is set to 0. Finally, if an error occurs in position 4, or 5, or 6, or 7, the most significant digit of the position number, i.e., c_1, should be a 1. Therefore, if digits 4,5,6, and 7 are designed to have even parity,

an error in any one of these digits will be recorded as a 1 in the most significant digit of the position number. To summarize:

p_1 is selected so as to establish even parity in positions 4,5,6,7.
p_2 is selected so as to establish even parity in positions 2,3,6,7.
p_3 is selected so as to establish even parity in positions 1,3,5,7.

The code can now be constructed by adding the appropriate checking digits to the message digits. Consider, for example, the message 0100 (i.e., decimal 4).

Position:	1	2	3	4	5	6	7
	p_3	p_2	m_1	p_1	m_2	m_3	m_4
Original BCD message:			0		1	0	0
Parity check in positions 4,5,6,7 requires $p_1 = 1$			0	**1**	**1**	**0**	**0**
Parity check in positions 2,3,6,7 requires $p_2 = 0$		**0**	**0**	1	1	**0**	**0**
Parity check in positions 1,3,5,7 requires $p_3 = 1$	**1**	0	**0**	1	**1**	0	**0**
Coded message:	1	0	0	1	1	0	0

p_1 is set equal to 1 so as to establish even parity in positions 4, 5, 6, and 7. Similarly, it is evident that p_2 must be a 0 and p_3 a 1, so that even parity is established in positions 2,3,6, and 7 and 1, 3, 5, and 7. The Hamming code for the decimal digits coded in BCD is shown in the following table.

Decimal Digit	Position: 1 p_3	2 p_2	3 m_1	4 p_1	5 m_2	6 m_3	7 m_4
0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	1
2	0	1	0	1	0	1	0
3	1	0	0	0	0	1	1
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	1
6	1	1	0	0	1	1	0
7	0	0	0	1	1	1	1
8	1	1	1	0	0	0	0
9	0	0	1	1	0	0	1

The error location and correction is performed in the following manner. Suppose, for example, that the sequence 1101001 is transmitted but, due to an error in the sixth position, the sequence 1101011 is received. The location of the error can be determined by performing three parity checks as follows:

Position:	1	2	3	4	5	6	7	
Message received:	1	1	0	1	0	1	1	
4567 parity check :				1	0	1	1	$\Rightarrow$ $c_1 = 1$ since parity is odd
2367 parity check :		1	0			1	1	$\Rightarrow$ $c_2 = 1$ since parity is odd
1357 parity check:	1		0		0		1	$\Rightarrow$ $c_3 = 0$ since parity is even

Thus the position number formed of c_1 c_2 c_3 is 110, which means that the location of the error is in position 6. To correct the error, the digit in position 6 is complemented, and the correct message 1101001 is obtained.

It is easy to prove that the Hamming code constructed as shown above is a code whose distance is three. Consider, for example, the case where the two original four-bit code words differ in only one position, e.g., 1001 and 0001. Since each message digit appears in at least two parity checks, the parity checks that involve the digit in which the two code words differ will result in different parities, and hence different checking digits will be added to the two words, making the distance between them equal to three. For example, consider the two code words below.

Position:	1	2	3	4	5	6	7
	p_3	p_2	m_1	p_1	m_2	m_3	m_4
First Word:			1		0	0	1
Second Word :			0		0	0	1
First Word with parity bits:	0	0	1	1	0	0	1
Second word with parity bits:	1	1	0	1	0	0	1

The two words differ in only m_1 (i.e., position 3). Parity checks 1357 and 2367 for these two words will give different results. Therefore the parity-checking digits p_3 and p_2 must be different for these words. Clearly, the foregoing argument is valid in the case where the original code words differ in two of the four positions. Thus the Hamming code has a distance of three.

If the distance is increased to four, by adding a parity bit to the code in the above table, so that all eight digits will have even parity, the code may be used for single-error correction and double-error detection in the following manner. Suppose that two errors occur; then the overall parity check is satisfied but the position number (determined as before from the first seven digits) will indicate an error. Clearly, such a situation indicates the existence of a double error. The error positions, however, cannot be located. If only a single error occurs, the overall parity check will detect it. Now, if the position number is 0, then the error is in the last parity bit; otherwise it is in the position given by the position number. If all four parity checks indicate even parities, then the message is correct.

PROBLEMS

1. Write the entire 4 bit Gray code by reflecting.
2. Using the 4 bit Gray code word for 6, what is the 6 bit Gray code word for 6?
3. Convert the following Gray code word to binary.

   1 0 0 1 1 0 1 0
4. Convert the following Binary code word to Gray.

   1 0 0 1 1 0 1 0

The following hamming code word was received. Use it to answer questions 5 - 10.

0 1 0 1 1 0 1

5. Circle the parity bits.
6. What position number is generated to determine if an error has occurred in transmission?
7. Did an error occur in transmission?
8. What was the transmitted code word?
9. What was the transmitted message?
10. If the message is binary, what is the decimal value?
11. Encode a decimal 13 using each of the following codes.

A. binary

B. BCD

C. 5 bit even parity (4 bit binary with the parity bit on the left).

D. Gray

E. Excess-3

F. 7 bit Hamming

12. For a weighted code to be self-complementing, a necessary--but not sufficient--condition is that the sum of the weights_____.
13. In order for a code to be capable of detecting one transmission error, the minimum distance _____.