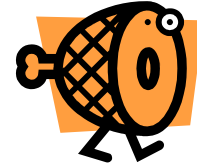


Hamming Code

When data is transmitted from one location to another there is always the possibility that an error may occur. There are a number of reliable codes that can be used to encode data so that the error can be detected and corrected. With this project you will explore a simple error detection-correction technique called a Hamming Code. A Hamming Code can be used to detect and correct one-bit change in an encoded code word. This approach can be useful as a change in a single bit is more probable than a change in two bits or more bits.



Here is an example of how this process works. Consider the table below which has 15 positions. Data is represented (stored) in every position except 1, 2, 4 and 8. These positions (which are powers of 2) are used to store parity (error correction) bits.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		1		2	3	4		5	6	7	8	9	10	11

Using the four parity (error correction bits) positions we can represent 15 values (1- 15). These values and their corresponding binary representation are shown in the table below.

Pos	$2^0 = 1$	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

Using the format given, data is represented by the 11 non-parity bits. Say for example we have the following data item to be encoded:

10101101011

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		1		0	1	0		1	1	0	1	0	1	1

After placing the data in the table we find that in positions 3, 6, 9, 10, 12, 14 and 15 we have a '1'. Using our previous conversion table we obtain the binary representation for each of these values. We then exclusive OR the resulting values (essentially setting the parity bit to 1 if an odd # of 1's else setting it to 0). The results of this activity are shown below:

	1	1	0	0	3
	0	1	1	0	6
	1	0	0	1	9
	0	1	0	1	10
	0	0	1	1	12
	0	1	1	1	14
	1	1	1	1	15
XOR	1	1	0	1	(11)

The parity bits are then put in the proper locations in the table providing the following end result:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	0	1	0	1	1	1	0	1	0	1	1

This is the encoded code word that would be sent. The receiving side would re-compute the parity bits and compare them to the ones received. If they were the same no error occurred – if they were different the location of the flipped bit is determined.

For example, let's say that the bit in position 14 was flipped during transmission. The receiving end would see the following encoded sequence:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	0	1	0	1	1	1	0	1	0	0	1

Below is the re-calculation done at the receiving end (notice the information for position 14 has been left out as it was flipped from 1 to 0).

	1	1	0	0	3
	0	1	1	0	6
	1	0	0	1	9
	0	1	0	1	10
	0	0	1	1	12
	1	1	1	1	15
XOR	1	0	1	0	

The re-calculated parity information is then compared to the parity information sent/received. If they are both the same the result (again using an XOR – even parity) will be all 0's. If a single bit was flipped the resulting number will be the position of the errant bit (check back into table). For example:

	1	1	0	1	sent/received
	1	0	1	0	new calculated
XOR	0	1	1	1	this bit was flipped (14)

Your task is to write a program that will perform two basic functions:

- a) Request an **8 bit word** – generate and display its 11 bit Hamming encoded code word sequence.

- b) Request an 11 bit properly encoded Hamming code word and indicate if an error has occurred (in which case you give the position of the bad bit) or if it is properly encoded.