Error-Correction

Error correcting codes determine the corrupted bits and correct it

Allows the recipient to reconstruct the correct message even after it has been corrupted Hamming code is used to correct single bit error

Reed Solomon code is used to correct burst errors

The use of error-correcting codes is often referred to as forward error correction

Hamming code

Hamming codes are code words formed by adding redundant check bits, or parity bits, to a data word

A frame consists of *m* data (i.e., message) bits and *r* redundant or check bits.

An n-bit unit containing data and check bits is often referred to as an *n*-bit codeword.

The error-detecting and error-correcting properties of a code depend on its Hamming distance.

To design a code with *m* message bits and *r* check bits that will allow all single errors to be corrected then 2^r m + r + 1

The bits of the codeword are numbered consecutively, starting with bit 1 at the left end. The bits that are powers of 2 (1, 2, 4, 8, 16, etc.) are check bits.

The rest (3, 5, 6, 7, 9, etc.) are filled up with the m data bits.

Each check bit forces the parity of some collection of bits, including itself, to be even

_	_	_	_	P	_	_	_	P	_	P	P
12	11	10	9	8	7	6	5	4	3	2	1

Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips.

- Position 1: check 1 bit, skip 1 bit (1, 3, 5, 7, 9, 11, 13, ...)
- Position 2: check 2 bits, skip 2 bits (2,3, 6,7, 10,11, 14,15, ...)
- Position 4: check 4 bits, skip 4 bits (4,5,6,7, 12,13,14,15, 20,21,22,23, ...)
- Position 8: check 8 bits, skip 8 bits, (8-15, 24-31, 40-47,...)
- Position 16: check 16 bits, skip 16 bits (16-31, 48-63, 80-95, ...)

Set a parity bit to 1 if the total number of ones in the positions it checks is odd.
Set a parity bit to 0 if the total number of ones in the positions it checks is even.

Example

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

– Bit 1checks the digits, 3, 5, 7, 9, and 11, so its value is 1.
– Bit 4 checks the digits, 5, 6, 7, and 12, so its value is 1.
– Bit 8 checks the digits, 9, 10, 11, and 12, so its value is also 1.

The completed code word is shown below

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

Suppose the received word is

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

The receiver calculates parit y bits:

- Bit 1 checks digit 3, 5, 7, 9, and 11. *Its value is 1, but should be zero.*
- Bit 2 checks digit 2, 3, 6, 7, 10, and 11. The zero is correct.
- Bit 4 checks digit 5, 6, 7, and 12. *Its value is 1, but should be zero.*
- Bit 8 checks digit 9, 10, 11, and 12. This bit is correct.

By adding the bit positions of the erroneous parity bits (1 + 4), the bad bit (5th bit) is determined. Bit 5 is changed to 1 and the data is restored.

Error Correction

Error Correction can be handled in two ways

1. When an error is discovered, the receiver can have the sender to retransmit the entire data unit.
2. A receiver can use an error correcting code, which automatically correct certain errors.

Error correcting codes are more sophisticated than error-detection codes and require more redundancy bits.

In single bit error detection only two states are sufficient.

1) error

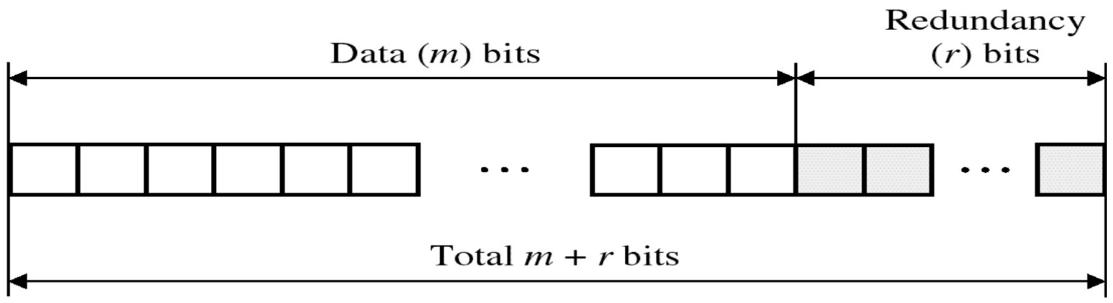
2) no error

Two states are not enough to detect an error but not to correct it.

Redundancy Bits

- To calculate the number of redundancy bit(r) required to correct a given number of data bits (m), we must find a relationship between m and r.
- Add m bits of data with r bits. The length of the resulting code is m+r.

Data and Redundancy bits



- If the total number of bits are m+r, then r must be able to indicate at least m+r+1 different states. r bits can indicate 2^r different states. Therefore, 2^r must be equal to or greater than m+r+1

$$2^r >= m+r+1$$

- For example if the value of m is 7 the smallest r value that can satisfy this equation is 4.

Relationship between data and redundancy bits

Number of Data Bits (m)	**Number of redundancy Bits(r)**	**Total bits (m+r)**
1	2	3
2	3	5
3	3	6

4	3	7
5	4	9
6	4	10
7	4	11

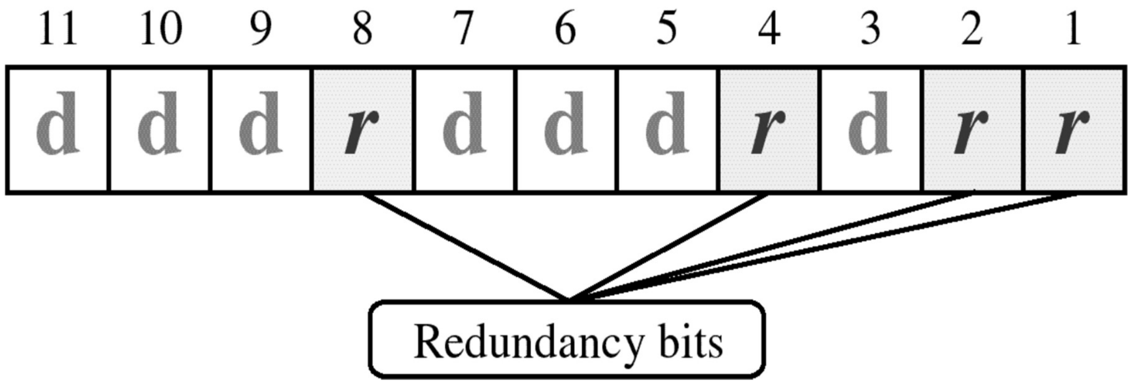
Hamming Code

R.W. Hamming provides a practical solution for the error correction.

Positioning the Redundancy Bits

For example, a seven-bit ASCII code requires four redundancy bits that can be added to the end of the data or intersperse with the original data bits. These redundancy bits are placed in positions 1, 2, 4 and 8. We refer these bits as r1, r2, r3 and r4

Position of redundancy bits in Hamming code



The combination used to calculate each of the four r values for a seven-bit data sequence are as follows

- The r1 bit is calculated using all bits positions whose binary representation include a 1 in the rightmost position
- r2 is calculated using all bit position with a 1 in the second position and so on

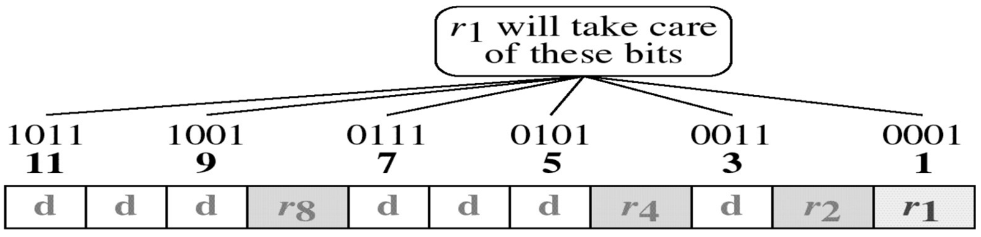
  r1: bits 1,3,5,7,9,11

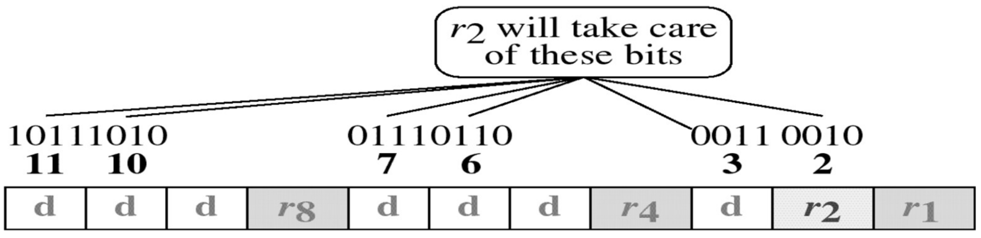
  r2: bits 2, 3, 6, 7, 10, 11

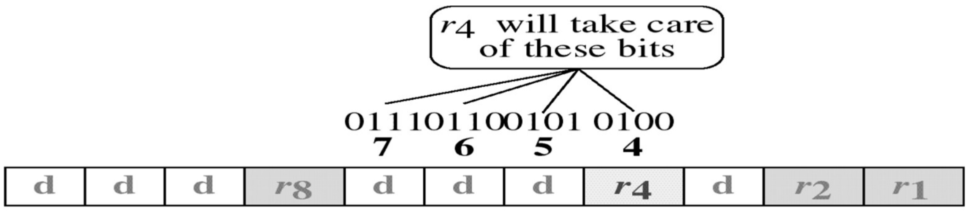
  r3: bits 4, 5, 6, 7

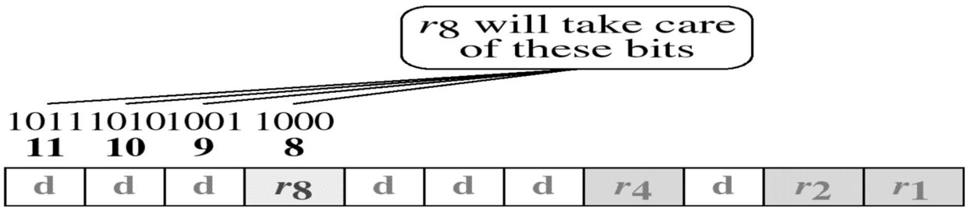
  r4: bits 8, 9, 10, 11

Redundancy bits calculation


r_1 will take care of these bits
1011
11
1001
9
0111
7
0101
5
0011
3
0001
1
d d d r_8 d d d r_4 d r_2 r_1


r_2 will take care of these bits
1011
11
1010
10
0111
7
0110
6
0011
3
0010
2
d d d r_8 d d d r_4 d r_2 r_1





Calculating the r values

- Place each bit of the original character in its appropriate position in the 11-bit unit.
- Calculate the even parities for the various bit combination.
- The parity value for each combination is the value of the corresponding r bit.

For example,

- The value of r1 is calculated to provide even parity for a combination of bits 3,5,7,9 and 11.
- The value of r2 is calculated to provide even parity with bits 3, 6, 7, 10 and 11.
- The value of r3 is calculated to provide even parity with bits 4,5,6 and 7.
- The value of r4 is calculated to provide even parity with bits 8,9,10 and 11.