# Hamming Codes

### Information Theory Lab 11 (2021)

## Objective

Perform encoding and decoding of a data file with the Hamming (8,4) SECDED (Single Error Correction - Double Error Detection) code.

## Theoretical notions

### Encoding

Hamming (8,4) SECDED encoding procedure operates on a block of 4 information bits (denoted as $i_3, i_5, i_6, i_7$) and produces a block of 8 output bits:

$$\mathbf{c} = c_0 c_1 c_2 i_3 c_4 i_5 i_6 i_7$$

The bits denoted with $c$ are parity bits (or *control* bits), and are computed as follows:

$$\begin{cases} c_0 = i_3 \oplus i_5 \oplus i_6 = c_1 \oplus c_2 \oplus i_3 \oplus c_4 \oplus i_5 \oplus i_6 \oplus i_7 \\ c_1 = i_3 \oplus i_5 \oplus i_7 \\ c_2 = i_3 \oplus i_6 \oplus i_7 \\ c_4 = i_5 \oplus i_6 \oplus i_7 \end{cases}$$

The coding is done for every group of 4 bits in the data.

### Decoding

Considering the received 8-bit block

$$\mathbf{r} = r_0 r_1 r_2 r_3 r_4 r_5 r_6 r_7,$$

decoding is done by computing the **syndrome**:

$$\begin{cases} z_0 &= r_0 \oplus r_1 \oplus r_2 \oplus r_3 \oplus r_4 \oplus r_5 \oplus r_6 \oplus r_7 \\ z_1 &= r_4 \oplus r_5 \oplus r_6 \oplus r_7 \\ z_2 &= r_2 \oplus r_3 \oplus r_6 \oplus r_7 \\ z_3 &= r_1 \oplus r_3 \oplus r_5 \oplus r_7 \end{cases}$$

The following cases may take place:

- $z_0 = z_1 = z_2 = z_3 = 0$: no error
- $z_0 = 1$: 1 error on position given by $z_1 z_2 z_{3(10)}$
- $z_0 = 0$, other $z_i \neq 0$: 2 errors on unknown positions

## Exercises

1. Write a C program that performs encoding of a data file with the Hamming (8,4) SECDED code.

   The program shall be called as follows:

   `HammingEncode.exe original.dat encoded.dat`

   - The arguments are:
     - `original.dat`: the original input file
     - `encoded.dat`: the encoded output file
   - The program should consist of the following steps:
     - declare two large vectors of `unsigned char`, for input and output bits
     - open the input file and read everything into the input vector
     - for every group of 4 bits from the input vector:
       * compute the control bits $c_0$, $c_1$, $c_2$, $c_4$
       * write all the 8 bits in the correct order in the output vector
       * advance by 4 and repeat
     - write the output vector to the output data file

   **Note**: Start from the parity-bit encoding program done in a previous lab, and change only what is needed (the overall program architecture is very similar).

2. Write a C program that performs decoding of a data file previously with the encoded Hamming (8,4) SECDED code.

   The program shall be called as follows:

   ```
   HammingDecode.exe encoded.dat decoded.dat
   ```

   - The arguments are:
     - `encoded.dat`: the input file, previously encoded
     - `decoded.dat`: the output decoded file
   - The program should consist of the following steps:
     - declare two large vectors of `unsigned char`, for input and output bits
     - open the input file and read everything into the input vector
     - for every group of 8 bits from the input vector:
       * compute the syndrome bits $z_0$, $z_1$, $z_2$, $z_3$
       * if one error is detected, fix the error (by toggling the erroneous bit) and display a message "Fixed 1 error\n"
       * if two errors are detected, display a message "Detected 2 errors, can't fix\n"
       * write the 4 information bits $(i_3, i_5, i_6, i_7)$ in the output vector
       * advance and repeat
     - write the output vector to the output data file

   **Note**: Start from the parity-bit decoding program done in a previous lab, and change only what is needed (the overall program architecture is very similar).

3. Run the encoding program to produce the encoded file, for some input file. What is the size of the output file compared to the input file?

4. Take the encoded file and use a HEX-editor introduce a few random errors, not more than **1 error per byte**. Save the modified file and run the program on it. Check that the errors have been fixed and the decoded output is identical to the original file.

5. Take the encoded file and use a HEX-editor to also introduce **2 errors** in a single byte. Save the modified file and run the program on it. Check the output against the original file. Have all errors been fixed?

# Final questions

1. TBD