

Error Detection Schemes

PostNET

The scheme used by the US Postal Service to encode zip codes in machine readable form is called PostNET.



To encode a 5-digit zip code, 32 (long and short) bars are used. The first and last bars are always long and are not part of the code. The remaining 30 are divided into 6 groups of 5 bars, each group representing a digit from 0-9. (9-digit zip codes use 52 bars)

1 –	4 –	7 –	0 –
2 –	5 –	8 –	
3 –	6 –	9 –	

Example above is : **56458-2**

PostNET

Notice that a 6th digit has been added to the zip code. This last digit is calculated from the others and is called a *check sum digit*. It is the number needed to make the sum of all the digits a multiple of 10 ($\equiv 0 \pmod{10}$).

So, in our example $5 + 6 + 4 + 5 + 8 = 28$ and we need a 2 to get to 30, so 2 is the check sum digit.

The purpose of the check sum digit is to detect errors in reading the code. So, for instance, if the 4 in the zip code was misread as a 6, then the sum of the digits would be 32, not divisible by 10, and so the postal service's scanner would detect an error and reread the code.

PostNET

If only one digit is in error (including the check sum digit) this code will always detect it, since the error can be no larger than 9. However, the scheme is defeated by any collection of errors which add up to a multiple of 10.

The association of digits to groups of 5 bars is called a *2 out of 5 code* (each group of 5 bars has 2 long bars), a very common coding scheme used when there are only 10 symbols which need to be coded. 2 out of 5 codes were used very heavily in the early days of computers.

PostNET

ZIP (**Z**one **I**mprovement **P**lan) codes were introduced by the U.S. Postal Service in 1963. In 1983, the Postal service added four more digits to the code and called the result a ZIP + 4 code. Recently (probably 2003), two more digits were added to create the Delivery Point Code. All of these codes use the same error detecting scheme check digit as the basic ZIP code.

PostNET (**P**OSTal **N**umeric **E**ncoding **T**echnique) is the bar code scheme used to encode the ZIP code (or any of its derivatives).

ISBN's

The publishing industry uses the International Standard Book Number (ISBN) to identify books. The ISBN for our text is

ISBN 0 – 471 – 19047 – 0

This is a 10 digit number where the first group identifies the language/country/region in which the book was published, the second group identifies the publisher, the third group identifies the book and the last digit is a check sum digit.

In this case the check sum digit is calculated so that

$$10a_1 + 9a_2 + 8a_3 + \dots + 2a_9 + a_{10} \equiv 0 \pmod{11},$$

where the ISBN is $a_1 a_2 a_3 \dots a_9 a_{10}$. In the case where $a_{10} = 10$, an “X” is used as the check sum digit.

ISBN's

So, for our example,

$$10(0) + 9(4) + 8(7) + 7(1) + 6(1) + 5(9) + 4(0) + 3(4) + 2(7) = 176$$
$$= 16(11) \equiv 0 \pmod{11}$$

and the check sum digit is 0.

As with the PostNET code, any single digit error will be detected because the error (no greater than 9) times a coefficient (no greater than 10) can not be a multiple of 11 (since 11 is prime). Also, some pairs of errors will be undetected.

This code however can detect a common type of human error that the PostNET code can not, namely a transposition error – interchanging two digits. In the PostNET code, this type of error does not change the check sum calculation.

ISBN's

Suppose that in the ISBN code, the digits a_i and a_j are interchanged. In the check sum calculation, the terms $(11-i)a_i$ and $(11-j)a_j$ of the correct ISBN are replaced by $(11-i)a_j$ and $(11-j)a_i$.

Thus, the difference in the two calculations is:

$$\begin{aligned}(11-i)a_i + (11-j)a_j - (11-i)a_j - (11-j)a_i &= (11-i)(a_i - a_j) + (11-j)(a_j - a_i) \\ &= (11 - i - 11 + j)(a_i - a_j) = (j-i)(a_i - a_j).\end{aligned}$$

Since $i \neq j$ and $a_i \neq a_j$ (if they are equal there is no change in the ISBN) and both factors are less than 10, this product can not be a multiple of 11. The (not necessarily adjacent) transposition error will therefore be detected.

UPC

The Universal Product Code (UPC) is widely used by supermarkets and mass market retailers for cash register checkout.

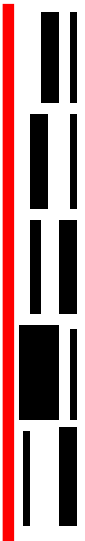










The UPC (actually, **UPC-A** as there is a compressed version called **UPC-E**) is a 12 digit code. The first digit indicates the character set, the next five identify the manufacturer, the following 5 identify the product and the last digit is a check sum digit.

The bar code is divided into a left and right side by long thin bar pairs at the beginning, end and middle. ||···||···||

UPC

The digits are represented by patterns, of width 7, made up of four variable width spaces and bars. Digits on the left side are of the form space-bar-space-bar and those on the right side are of the form bar-space-bar-space. The bars and spaces are between 1 and 4 units wide. The patterns on the right are the black/white (bar/space) reversals of those on the left.

Digit	Left Code	Binary	Digit	Left Code	Binary
0		0001101	5		0110001
1		0011001	6		0101111
2		0010011	7		0111011
3		0111101	8		0110111
4		0100011	9		0001011

UPC

The reversal of the code on the right permits scanning in both directions. For instance, the digit 1 on the left is 0011001 but on the right it would be 1100110 and if this is read from the right we obtain the scan code 0110011 which does not coincide with any of the left codes – so it can be assigned to the digit 1 (from the right.)

The check sum digit for the UPC is calculated by:

$$3a_1 + a_2 + 3a_3 + a_4 + 3a_5 + \dots + 3a_{11} + a_{12} \equiv 0 \pmod{10}$$

where $a_1 a_2 a_3 \dots a_{11} a_{12}$ is the UPC.

With this calculation all single digit errors are detected, and most – but not all – adjacent transposition errors are detected.

UPC

If the digits a_i and a_{i+1} are interchanged then the check sum would change by either:

$$3a_i + a_{i+1} - 3a_{i+1} - a_i = 2(a_i - a_{i+1})$$

or

$$a_i + 3a_{i+1} - a_{i+1} - 3a_i = 2(a_{i+1} - a_i).$$

Thus, if $|a_i - a_{i+1}| = 5$, the change would be ± 10 and so, the error would not be detected.

BookLand (ISBN) Bar Code

The industry standard bar code used to encode the ISBN of a book is EAN-13 (European Article Numbering system), the European version of UPC-A. This system encodes 13 digit numbers, with the last digit being a check sum digit. (Bar codes for non-trade paperbacks do not encode the ISBN and just use a standard UPC).



A BookLand code starts with 978 followed by the first nine digits of the ISBN and the 13th digit is a check sum calculated by:

$$a_1 + 3a_2 + a_3 + 3a_4 + a_5 + \dots + a_{11} + 3a_{12} + a_{13} \equiv 0 \pmod{10}.$$

BookLand (ISBN) Bar Code

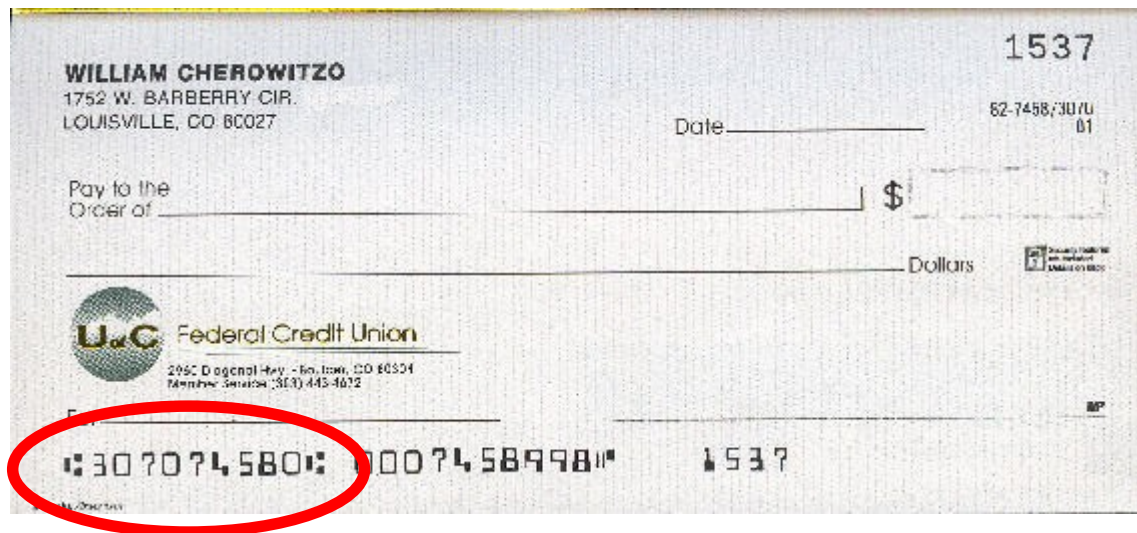
The BookLand code has the same error detecting (and non-detecting) capabilities as the UPC-A code.

It is interesting to note that because the check digit of the ISBN is not encoded by BookLand, this bar code loses the capability of detecting all transposition errors which the ISBN had. Thus, in order to gain optical scanner capability, a fairly good error detecting code was replaced by a mediocre one.

By Jan. 1, 2007, all 10-digit ISBN's will be replaced by the 13-digit BookLand code. When all the 10-digit numbers are used, a new prefix (979) will be started.

Bank Check Code

On the bottom left of your personal checks there appears a nine digit Bank Identification Number (between the two “|:” symbols).



The 9th digit of this code is a check sum digit computed by:

$$7a_1 + 3a_2 + 9a_3 + 7a_4 + 3a_5 + 9a_6 + 7a_7 + 3a_8 + 9a_9 \equiv 0 \pmod{10}.$$

Bank Check Code

This code detects all single digit errors and transposition errors of the forms $\dots ac\dots \rightarrow \dots ca\dots$ and $\dots abc\dots \rightarrow \dots cba\dots$ provided that $|a - c| \neq 5$. The UPC and EAN codes can not detect this second type of transposition error.

Check Sum Schemes

All of the schemes we have looked at have the same basic form.

The coded number will be considered as a vector of length k , i.e., (a_1, a_2, \dots, a_k) , and there is a weighting vector also of length k , (w_1, w_2, \dots, w_k) so that the k^{th} component of the code is calculated so that the standard dot product of the two vectors,

$$(a_1, a_2, \dots, a_k) \cdot (w_1, w_2, \dots, w_k) = a_1 w_1 + a_2 w_2 + \dots + a_k w_k \equiv 0 \pmod{n}$$

for some integer n , greater than any possible a_i .

PostNET : $k = 6, 10$ or 12 , $n = 10$ and $(w_1, w_2, \dots, w_k) = (1, 1, \dots, 1)$

ISBN : $k = 10$, $n = 11$ and $(w_1, w_2, \dots, w_k) = (10, 9, 8, \dots, 2, 1)$

UPC-A : $k = 12$, $n = 10$ and $(w_1, w_2, \dots, w_k) = (3, 1, 3, 1, \dots, 3, 1)$

BookLand: $k = 13$, $n = 10$ and $(w_1, w_2, \dots, w_k) = (1, 3, 1, 3, \dots, 3, 1)$

Bank : $k = 9$, $n = 10$ and $(w_1, w_2, \dots, w_k) = (7, 3, 9, 7, 3, 9, 7, 3, 9)$

Check Sum Schemes

We can make some general statements about schemes of this type.

Proposition 1: The scheme can detect a single error occurring in position i if and only if w_i is relatively prime to n .

Pf: Suppose that the error in position i is x . That is, instead of the digit a_i , we see the digit $a_i + x$. We have that $0 < |x| < n$.

The new check sum differs from the original check sum by xw_i . This error is undetected iff $xw_i \equiv 0 \pmod{n}$. With w_i relatively prime to n , this is equivalent to $x \equiv 0 \pmod{n}$, which is impossible. \square

Check Sum Schemes

Proposition 2: The scheme can detect a transposition error in positions i and j if and only if $|w_i - w_j|$ is relatively prime to n .

Pf: Suppose that the digits a_i and a_j are transposed. The change in the check sum calculation is:

$$a_i w_i + a_j w_j - a_i w_j - a_j w_i = a_i(w_i - w_j) - a_j(w_i - w_j) = (a_i - a_j)(w_i - w_j).$$

This transposition is undetected iff $|a_i - a_j||w_i - w_j| \equiv 0 \pmod n$. With $|w_i - w_j|$ relatively prime to n , this is equivalent to $|a_i - a_j| \equiv 0 \pmod n$.

But this is impossible unless $a_i = a_j$ and in that situation no transposition error has occurred. \square