ऋते ज्ञानान्न मुक्तिः

GLA UNIVERSITY
MATHURA
Recognised by UGC Under Section 2(f)

Accredited with A Grade by NAAC

12-B Status from UGC

Error Detection and Correction

Data can be corrupted during transmission.

Some applications require that errors be detected and corrected.

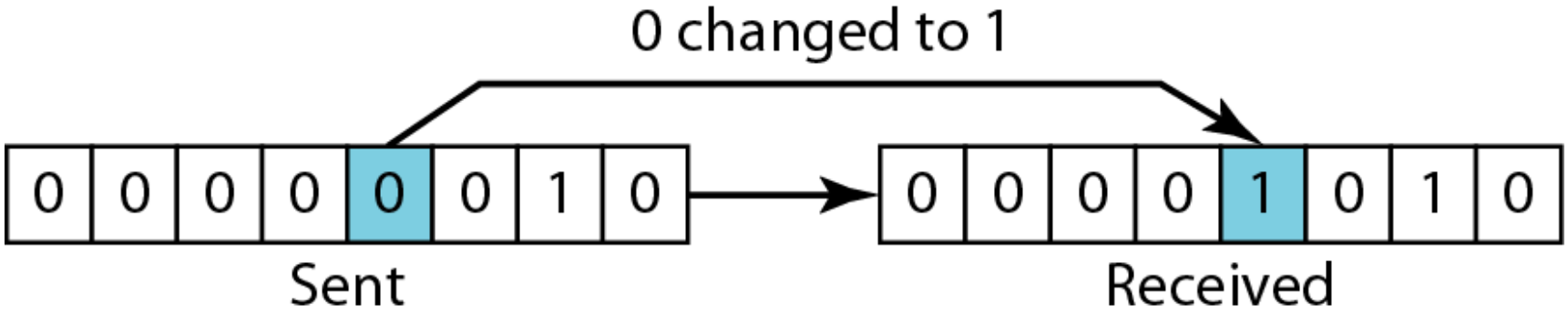
- Networks must be able to transfer data from one device to another with complete accuracy.
- Data can be corrupted during transmission.
- For reliable communication, errors must be detected and corrected.

★**Error detection and correction are implemented either at the data link layer or the transport layer of the OSI model.**

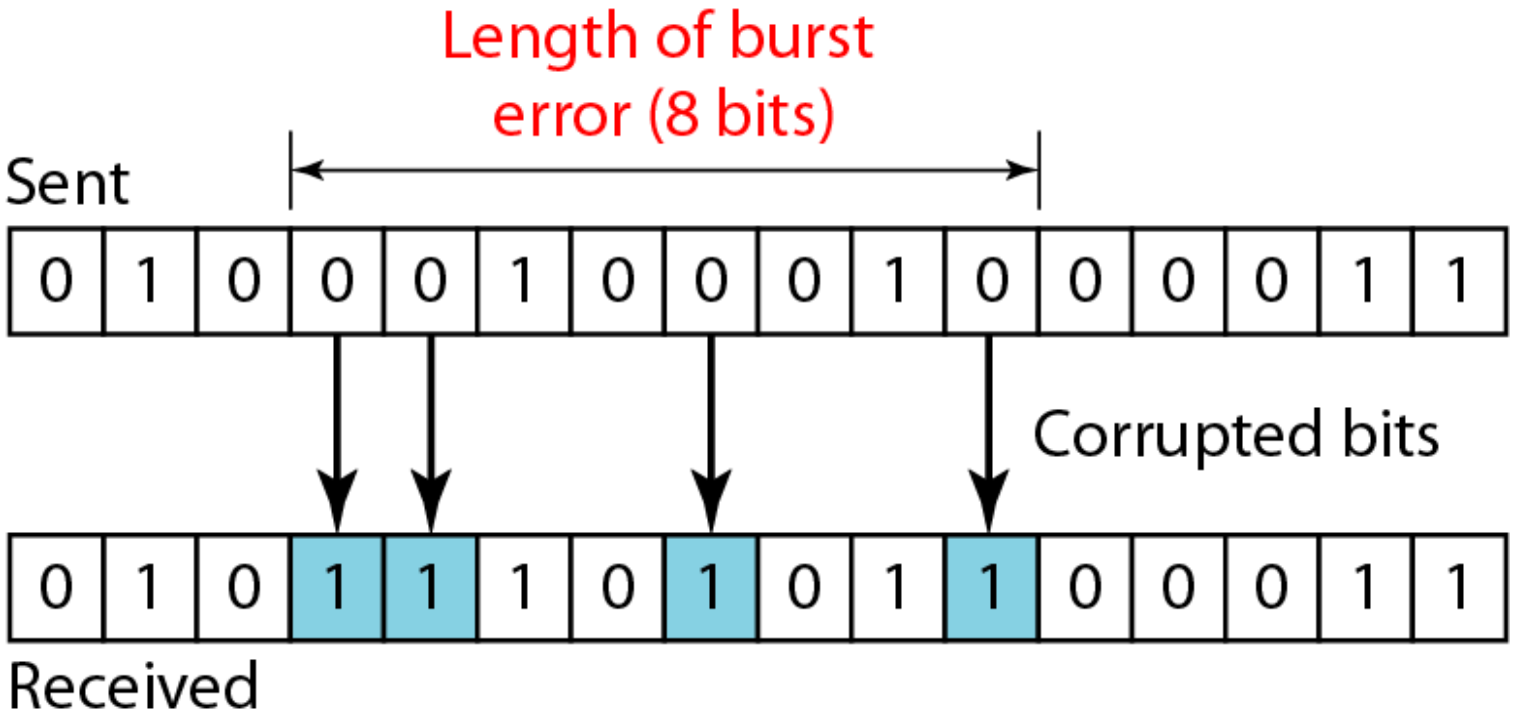
Types of Errors

- Single bit Error
- Burst Error

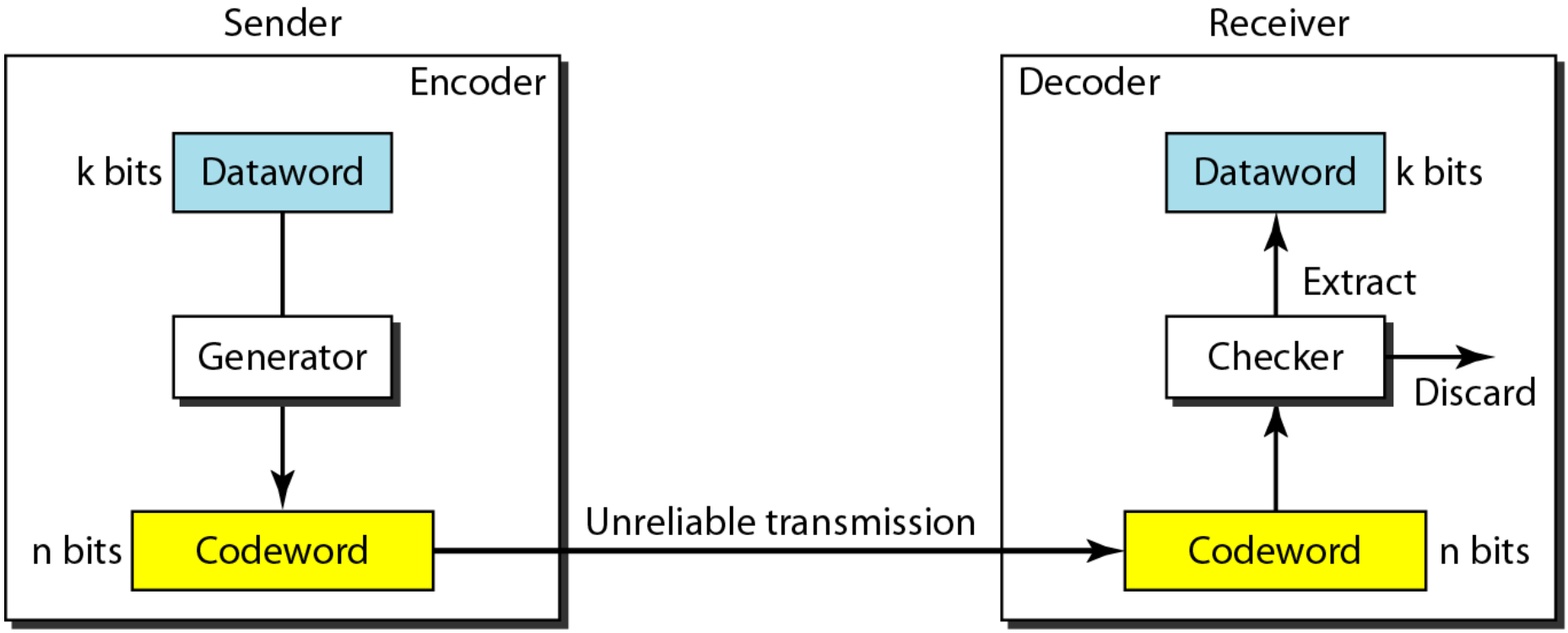
Single-bit error



Burst error of length 8



The structure of encoder and decoder



To detect or correct errors, we need to send redundant bits

BLOCK CODING

In block coding, we divide our message into blocks, each of k bits, called datawords. We add r redundant bits to each block to make the length n = k + r. The resulting n-bit blocks are called codewords.

Topics discussed in this section:

Error Detection
Error Correction
Hamming Distance
Minimum Hamming Distance

Table 10.1 *A code for error detection (Example 10.2)*

Datawords	*Codewords*
00	000
01	011
10	101
11	110

What if we want to send 01? We code it as 011. If 011 is received, no problem.
What if 001 is received? Error detected.
What if 000 is received? Error occurred, but not detected.

Let's add more redundant bits to see if we can correct error.

Table 10.2 *A code for error correction (Example 10.3)*

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

Let's say we want to send 01. We then transmit 01011. What if an error occurs and we receive 01001. If we assume one bit was in error, we can correct.

Note

The Hamming distance between two words is the number of differences between corresponding bits.

Example 10.4

Let us find the Hamming distance between two pairs of words.

1. The Hamming distance d(000, 011) is 2 because

$000 \oplus 011$ is 011 (two 1s)

2. The Hamming distance d(10101, 11110) is 3 because

$10101 \oplus 11110$ is 01011 (three 1s)

Note

The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words.

Example 10.5

Find the minimum Hamming distance of the coding scheme in Table 10.1.

Solution

We first find all Hamming distances.

$d(000, 011) = 2$	$d(000, 101) = 2$	$d(000, 110) = 2$	$d(011, 101) = 2$
$d(011, 110) = 2$	$d(101, 110) = 2$		

The d_{min} in this case is 2.

Example 10.6

Find the minimum Hamming distance of the coding scheme in Table 10.2.

Solution

We first find all the Hamming distances.

$d(00000, 01011) = 3$	$d(00000, 10101) = 3$	$d(00000, 11110) = 4$
$d(01011, 10101) = 4$	$d(01011, 11110) = 3$	$d(10101, 11110) = 3$

The d_{min} in this case is 3.

Note

To guarantee the *detection* of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = s + 1$.

The minimum Hamming distance for our first code scheme (Table) is 2. This code guarantees detection of only a single error. For example, if the third codeword (101) is sent and one error occurs, the received codeword does not match any valid codeword. If two errors occur, however, the received codeword may match a valid codeword and the errors are not detected.

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Example 10.8

Our second block code scheme (Table 10.2) has $d_{min} = 3$. This code can detect up to two errors. Again, we see that when any of the valid codewords is sent, two errors create a codeword which is not in the table of valid codewords. The receiver cannot be fooled.

However, some combinations of three errors change a valid codeword to another valid codeword. The receiver accepts the received codeword and the errors are undetected.

Note

To guarantee correction of up to *t* errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = 2t + 1$.

Example 10.9

A code scheme has a Hamming distance $d_{min} = 4$. What is the error detection and correction capability of this scheme?

Solution

This code guarantees the detection of up to three errors ($s = 3$), but it can correct up to one error. In other words, if this code is used for error correction, part of its capability is wasted. Error correction codes need to have an odd minimum distance (3, 5, 7, . . .).

Consider a binary code that consists only four valid codewords as given below.
00000, 01011, 10101, 11110
Let minimum Hamming distance of code be p and maximum number of erroneous bits that can be corrected by the code be q. The value of p and q are: ***(GATE-2017)***
(A) p = 3 and q = 1
(B) p = 3 and q = 2
(C) p = 4 and q = 1
(D) p = 4 and q = 2

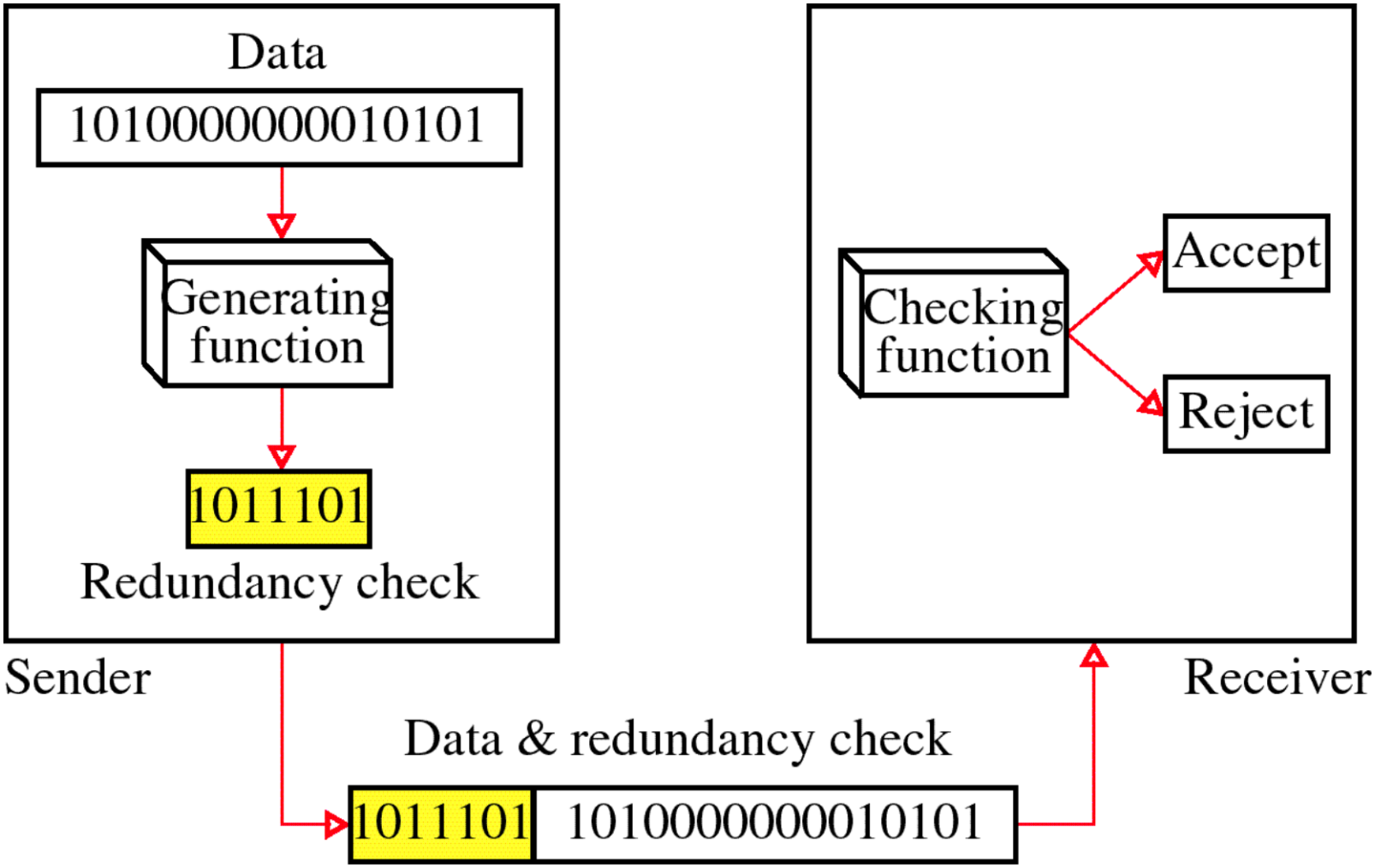
A code with minimum Hamming distance d between its codewords can detect at most d-1 errors and can correct ⌊(d-1)/2⌋ errors. Here d = 3. So number of errors that can be corrected is 1.

Error detection

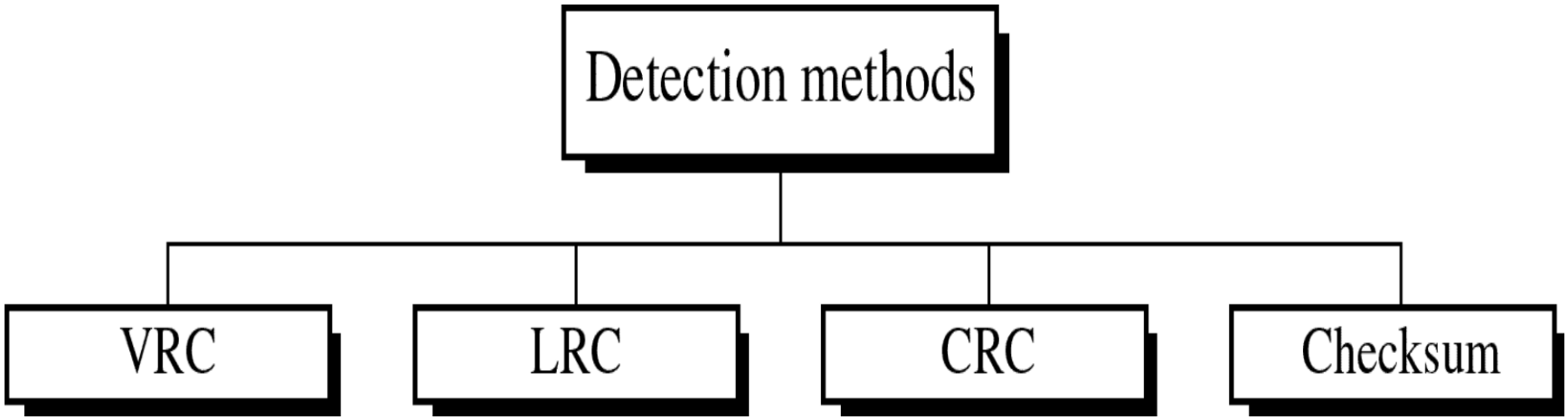
Error detection means to decide whether the received data is correct or not without having a copy of the original message.

Error detection **uses the concept of redundancy**, **which means** adding extra bits for detecting errors at the destination.

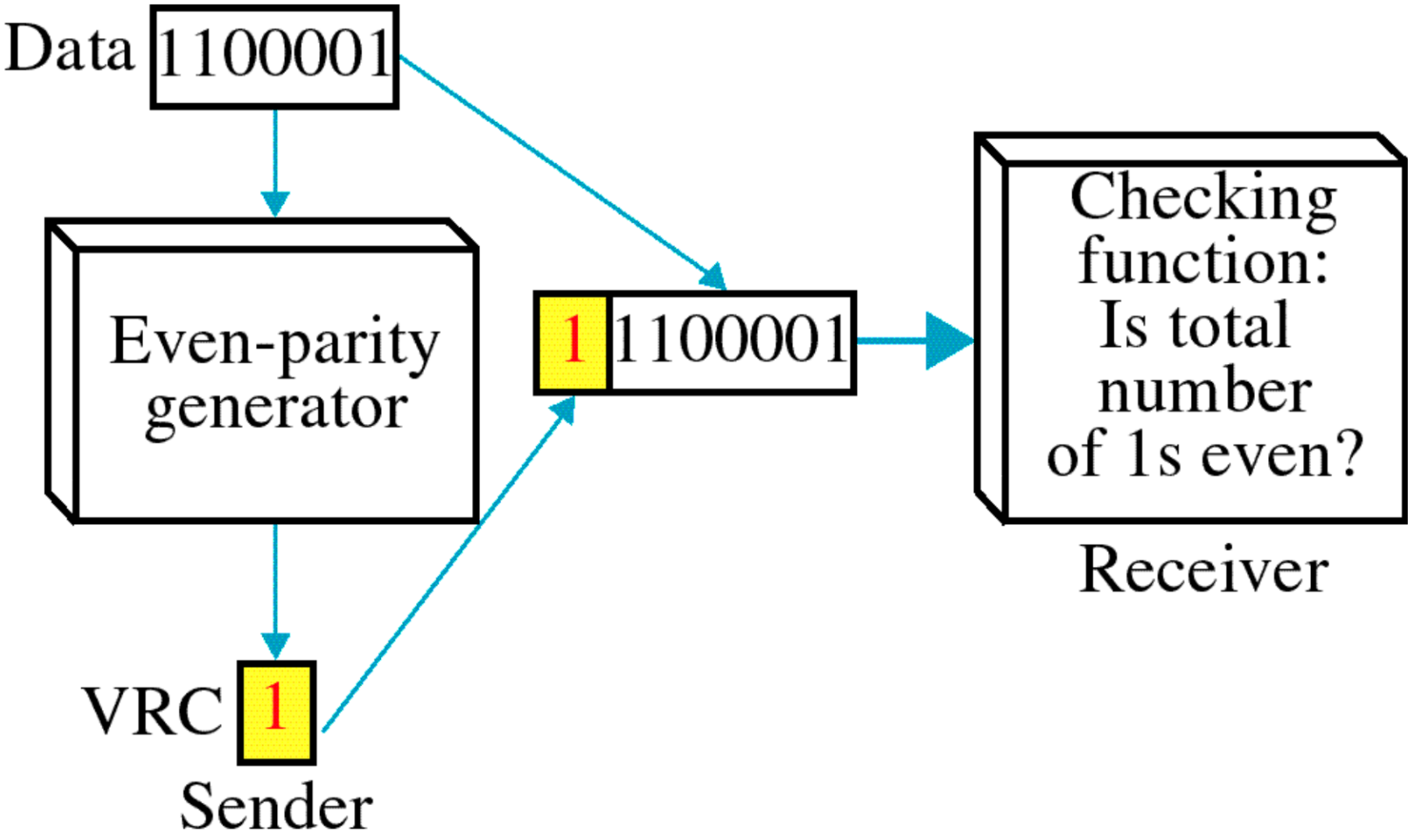
Redundancy


Data
1010000000010101
Generating function
1011101
Redundancy check
Sender
Checking function
Accept
Reject
Receiver
Data & redundancy check
1011101
1010000000010101

Four types of redundancy checks are used in data communications



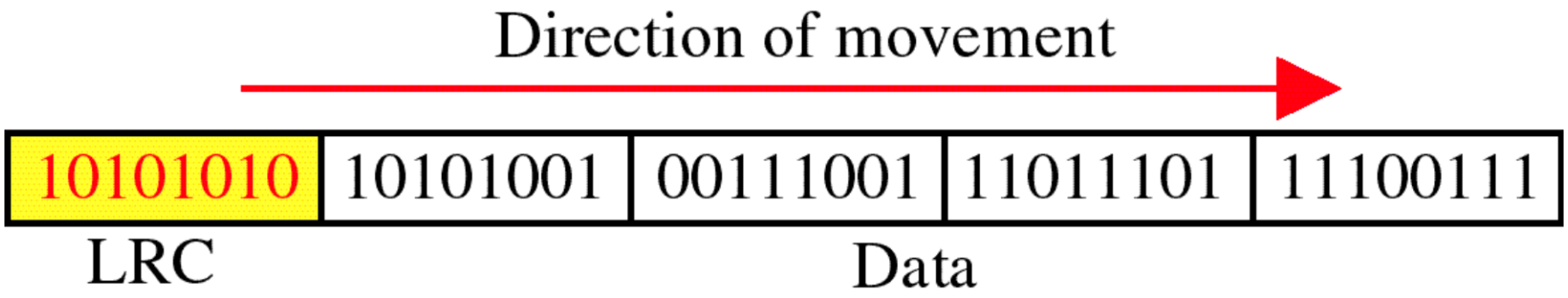
Vertical Redundancy Check
VRC



Performance

- It can detect single bit error
- It can detect burst errors only if the total number of errors is odd.

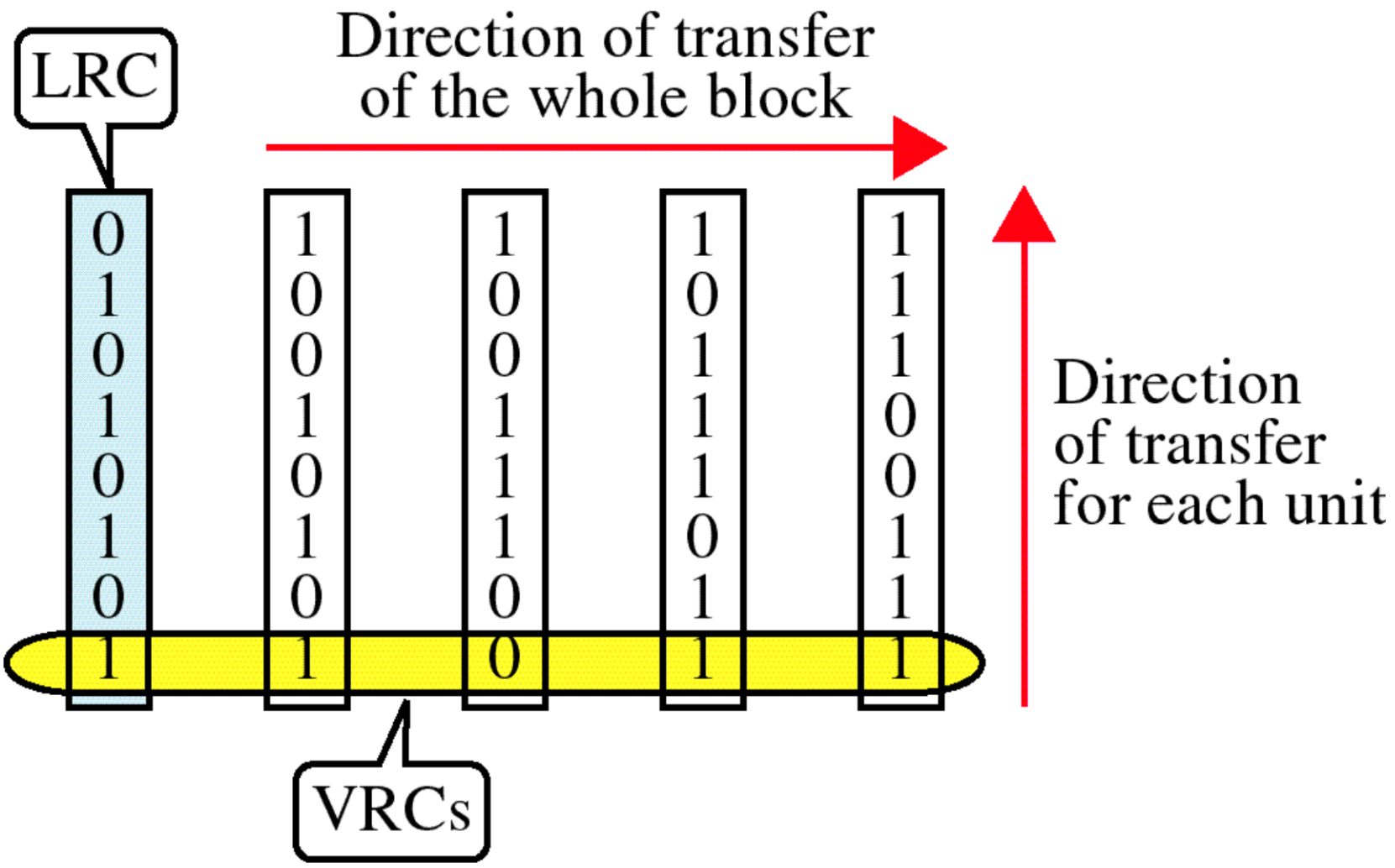
Longitudinal Redundancy Check
LRC



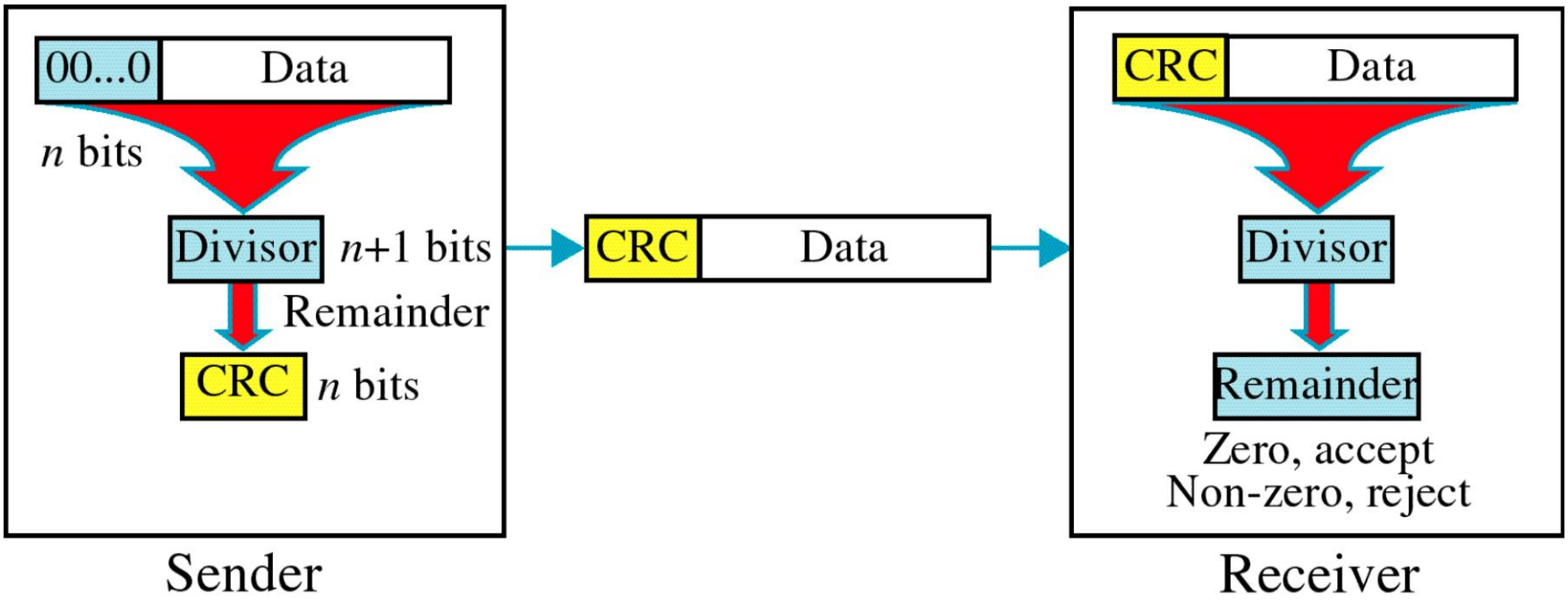
Performance

- **LCR increases the likelihood of detecting burst errors.**
- **If two bits in one data units are damaged and two bits in exactly the same positions in another data unit are also damaged, the LRC checker will not detect an error.**

VRC and LRC


LRC
Direction of transfer
of the whole block
0 1 0 1 0 1 0 1
1 0 0 1 0 1 0 1
1 0 0 1 1 1 1 0
1 0 1 1 1 0 1 1
1 1 1 0 0 1 1 1
Direction
of transfer
for each unit
VRCs

Cyclic Redundancy Check
CRC



Cyclic Redundancy Check

The CRC error detection method treats the packet of data to be transmitted as a large polynomial.

The transmitter takes the message polynomial and using polynomial arithmetic, divides it by a given generating polynomial.

The quotient is discarded but the remainder is “attached” to the end of the message.

Cyclic Redundancy Check

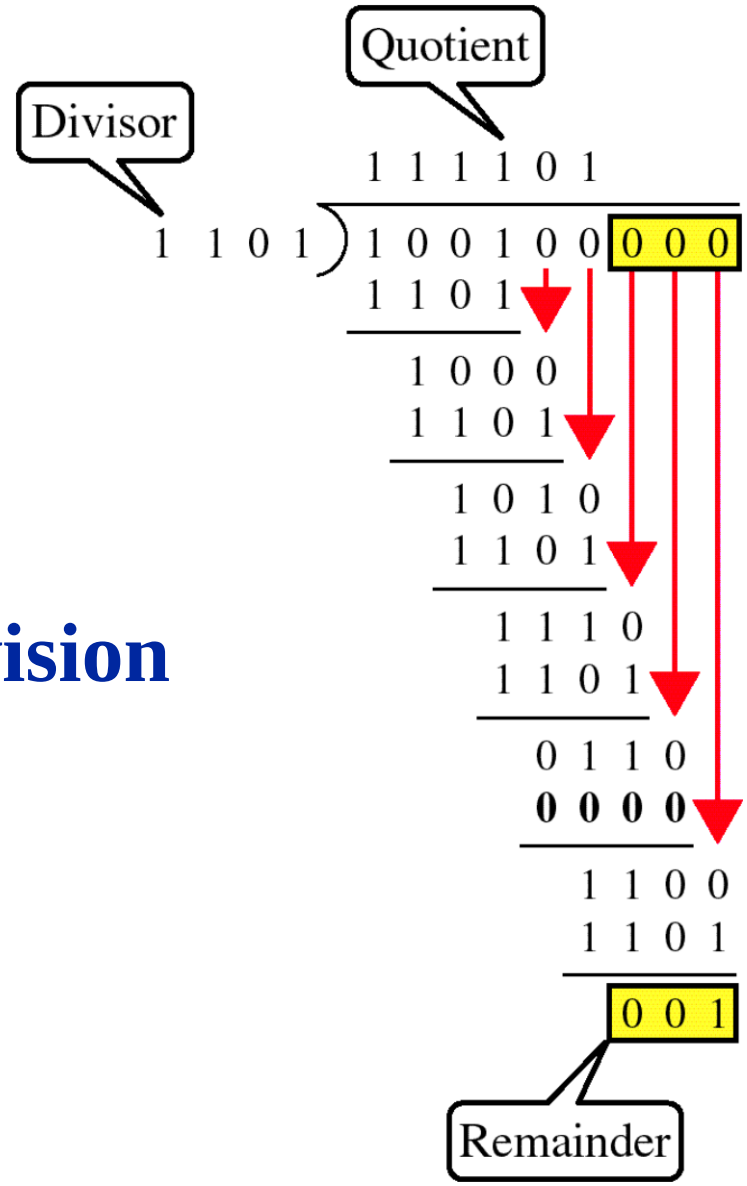
The message (with the remainder) is transmitted to the receiver.

The receiver divides the message and remainder by the same generating polynomial.

If a remainder not equal to zero results, there was an error during transmission.

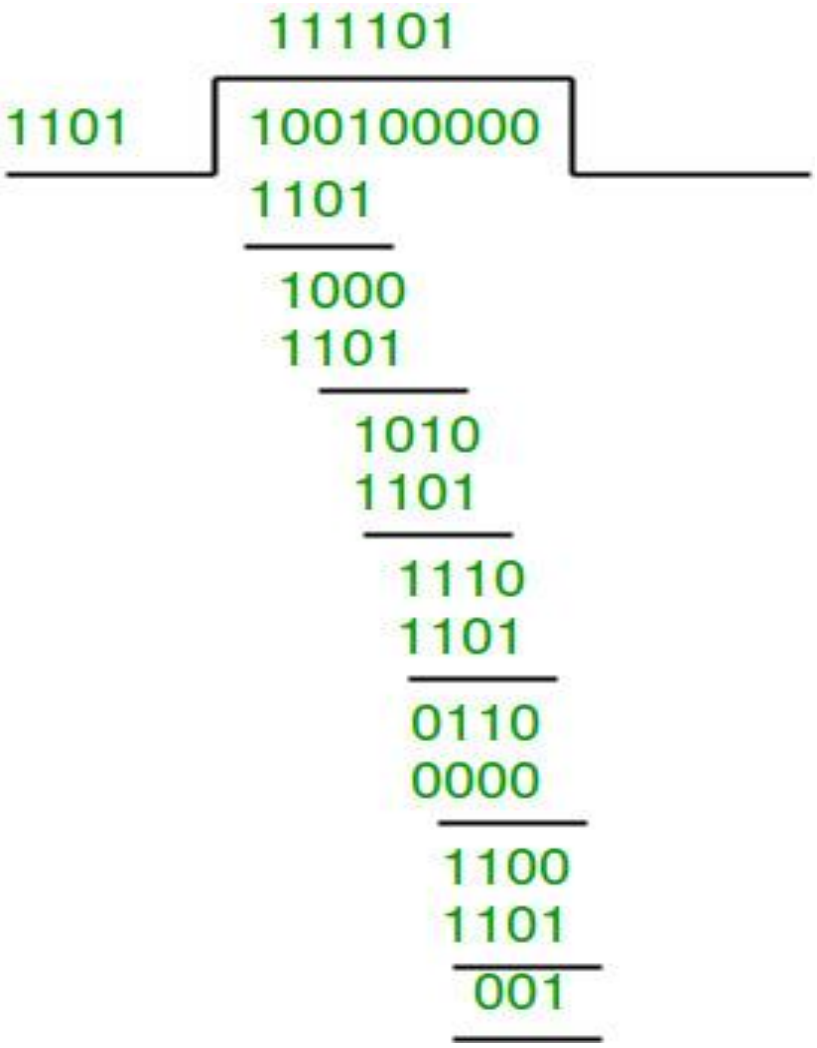
If a remainder of zero results, there was no error during transmission.

Binary Division

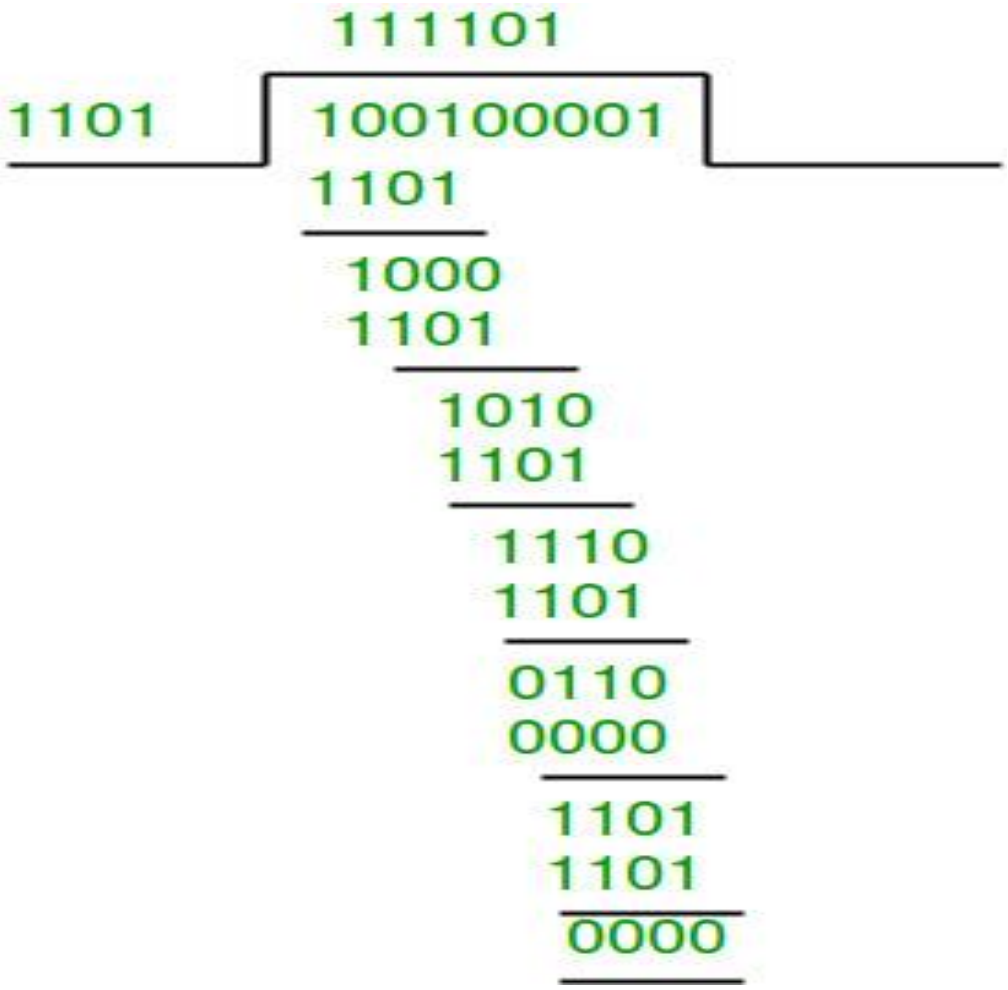


Data word to be sent - 100100 Key - 1101 [Or generator polynomial $x^3 + x^2 + 1$]

Sender Side:



Therefore, the remainder is 001 and hence the encoded data sent is 100100001.
Receiver Side: Code word received at the receiver side 100100001



Polynomial

$$x^7 + x^5 + x^2 + x + 1$$

Consider the following message M = 1010001101. The cyclic redundancy check (CRC) for this message using the divisor polynomial $x^5 + x^4 + x^2 + 1$ is : (GATE-2005)

(A) 01110
(B) 01011
(C) 10101
(D) 10110

Example 10.18

Suppose our data is a list of five 4-bit numbers that we want to send to a destination. In addition to sending these numbers, we send the sum of the numbers. For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, 36), where 36 is the sum of the original numbers. The receiver adds the five numbers and compares the result with the sum. If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum. Otherwise, there is an error somewhere and the data are not accepted.

Example 10.19

We can make the job of the receiver easier if we send the negative (complement) of the sum, called the checksum. In this case, we send (7, 11, 12, 0, 6, −36). The receiver can add all the numbers received (including the checksum). If the result is 0, it assumes no error; otherwise, there is an error.

CHECKSUM

Consider the data unit to be transmitted is-

10011001111000100010010010000100

Consider 8 bit checksum is used.

10011001	11100010	00100100	10000100

Now, all the segments are added and the result is obtained as-

- 10011001 + 11100010 + 00100100 + 10000100 = 1000100011
- Since the result consists of 10 bits, so extra 2 bits are wrapped around.
- 00100011 + 10 = 00100101 (8 bits)
- Now, 1's complement is taken which is 11011010.
- Thus, checksum value = 11011010

The data along with the checksum value is transmitted to the receiver

At receiver side,

- The received data unit is divided into segments of 8 bits.

- All the segments along with the checksum value are added.

- Sum of all segments + Checksum value = 00100101 + 11011010 = 11111111

- Complemented value = 00000000

Example 10.20

How can we represent the number 21 in one's complement arithmetic using only four bits?

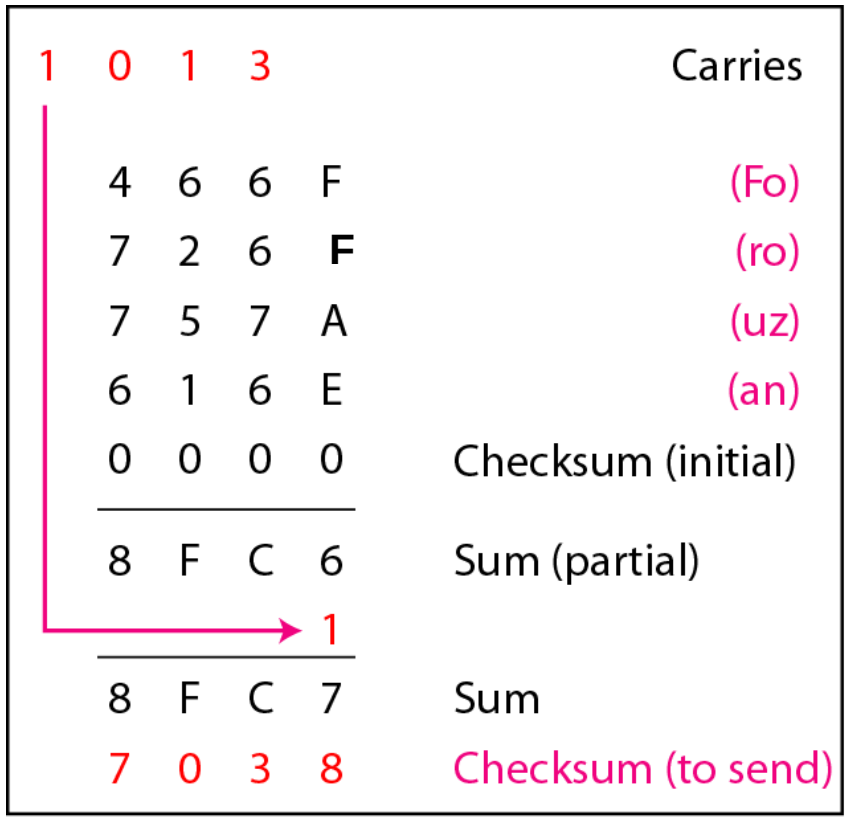
Solution

The number 21 in binary is 10101 (it needs five bits). We can wrap the leftmost bit and add it to the four rightmost bits. We have (0101 + 1) = 0110 or 6.

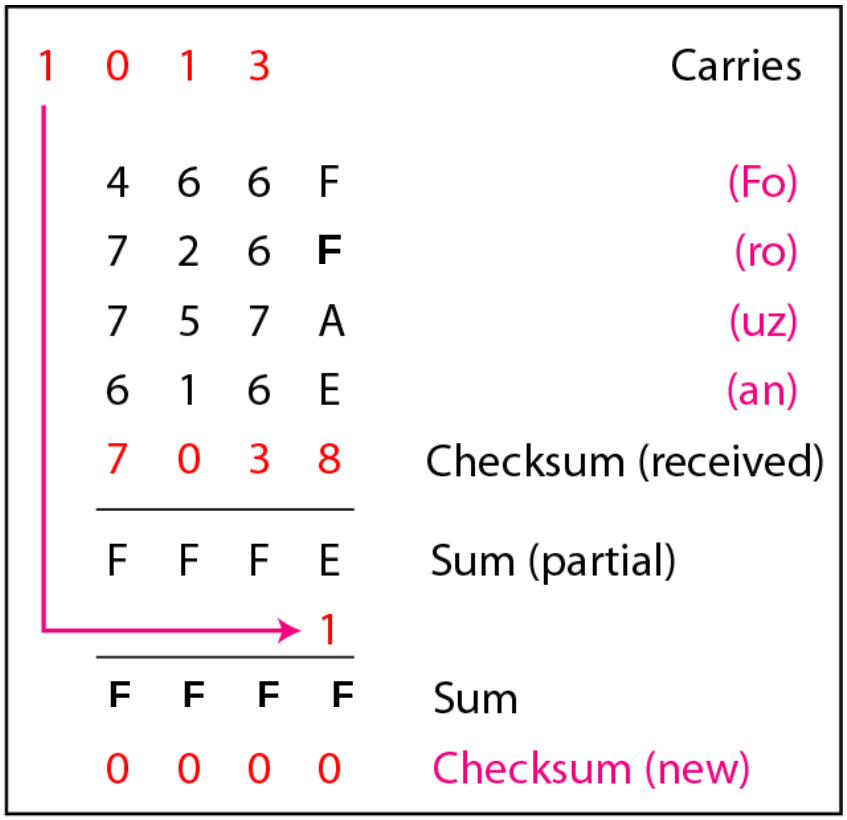
Consider the data unit to be transmitted is-

466F 726F 757A 616E

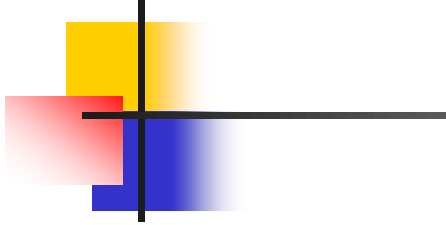
Figure 10.25 *Example 10.23*



a. Checksum at the sender site



a. Checksum at the receiver site

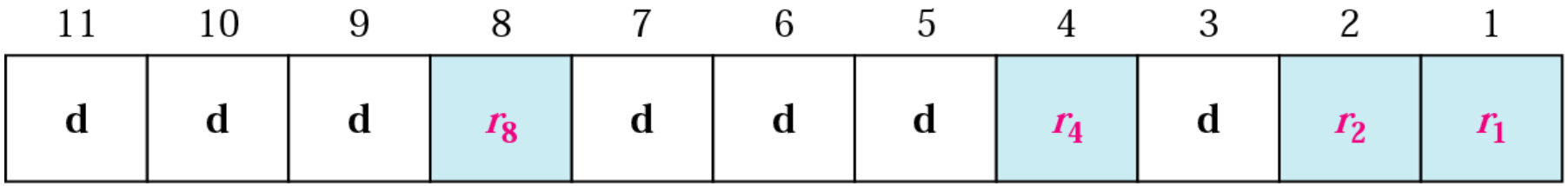


Error Correcting Codes or Forward Error Correction (FEC)

FEC is used in transmission of radio signals, such as those used in transmission of digital television (Reed-Solomon and Trellis encoding) and 4D-PAM5 (Viterbi and Trellis encoding)

Some FEC is based on Hamming Codes

Let's examine a Hamming Code



From previous edition of Forouzan

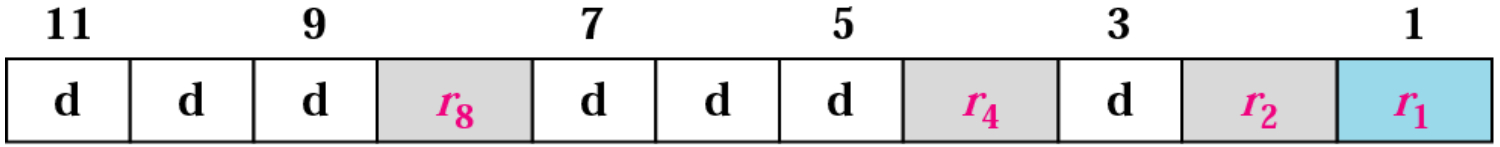
Counting the number of redundant bits

You can use the same formula for encoding, the number of redundant bits

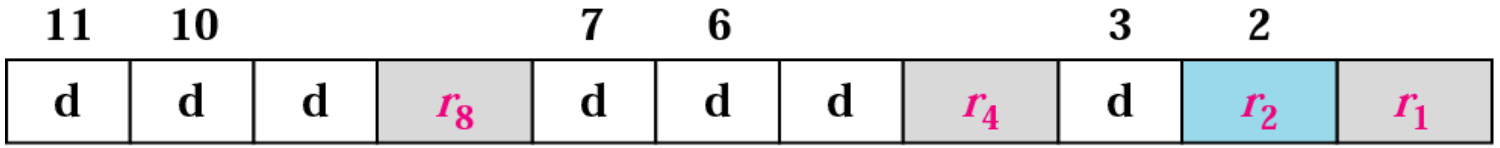
$$\mathbf{2^r \geq n + r + 1}$$

Here, the number of data bits and r is the number of redundant bits.

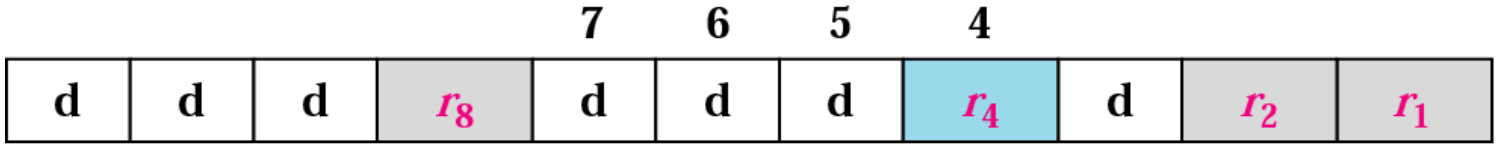
r_1 will take care of these bits.



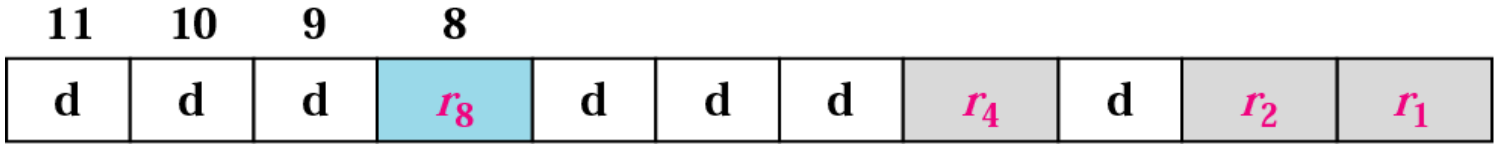
r_2 will take care of these bits.



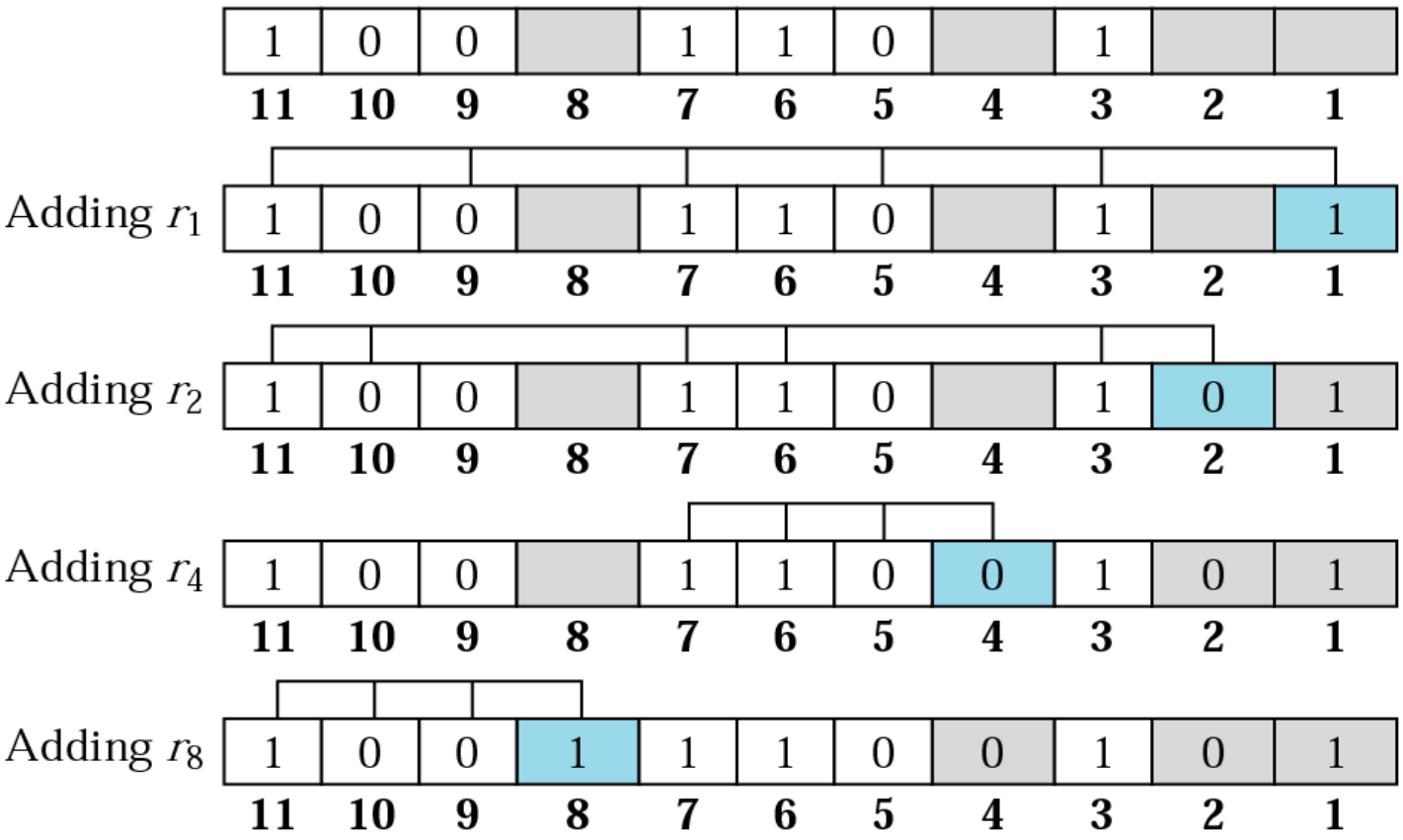
r_4 will take care of these bits.



r_8 will take care of these bits.

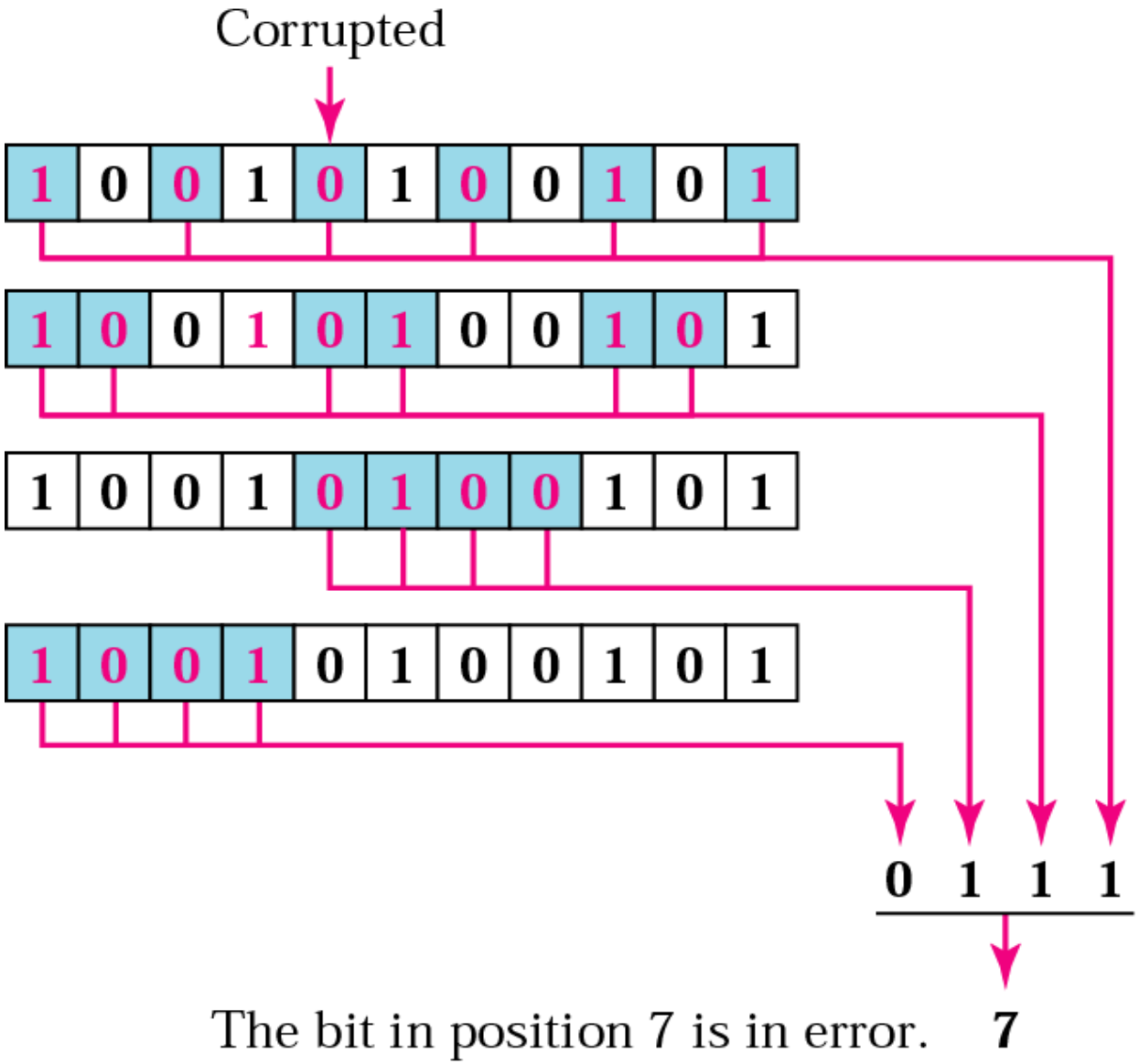


Example of redundancy bit calculation



Data:
1 0 0 1 1 0 1

Code:
1 0 0 1 1 1 0 0 1 0 1


Corrupted
1 0 0 1 0 1 0 0 1 0 1
1 0 0 1 0 1 0 0 1 0 1
1 0 0 1 0 1 0 0 1 0 1
1 0 0 1 0 1 0 0 1 0 1
0 1 1 1
The bit in position 7 is in error. 7