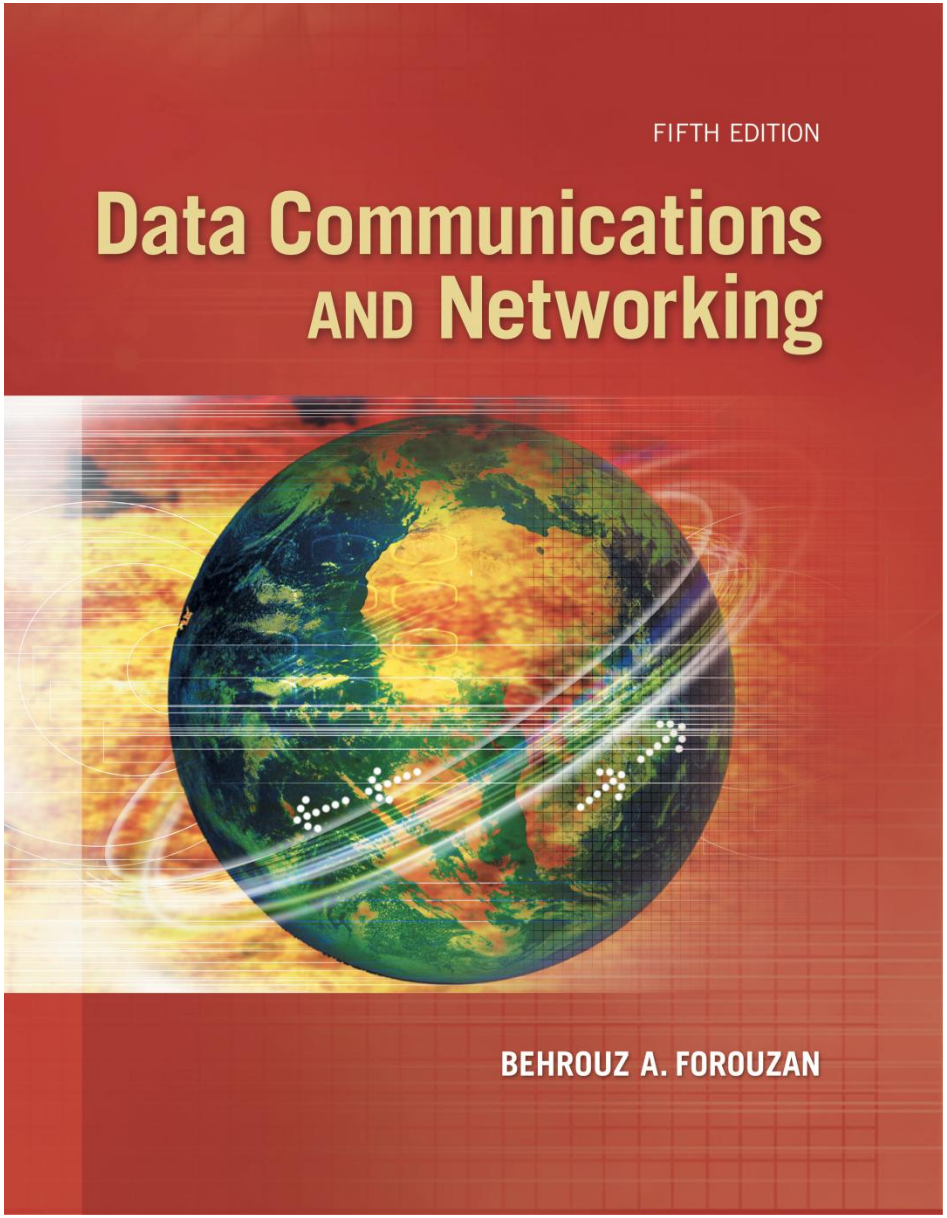
The McGraw-Hill Companies

Chapter 10

Error Detection And Correction



Review of Physical Layer

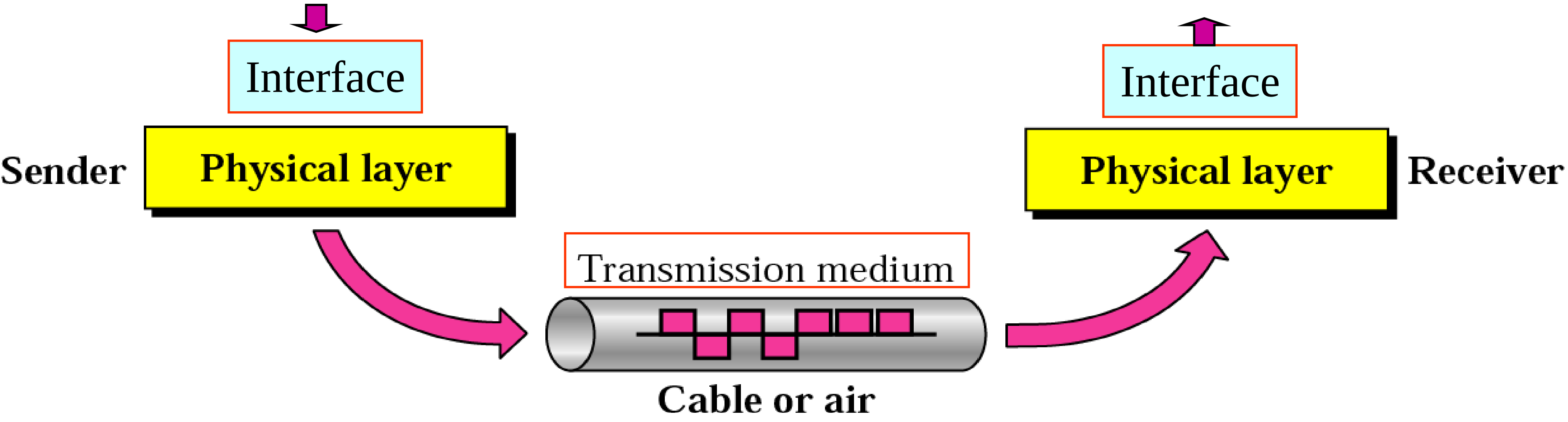
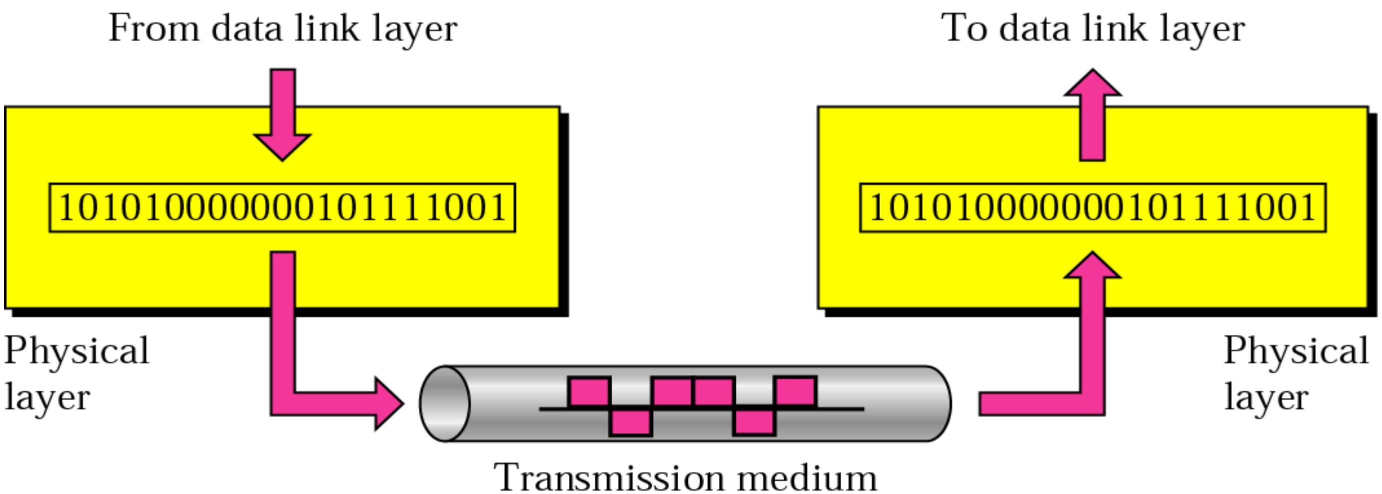
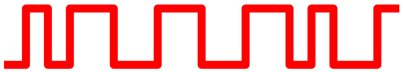


Figure 7.1 **Transmission medium and physical Interface**

Review of Signals

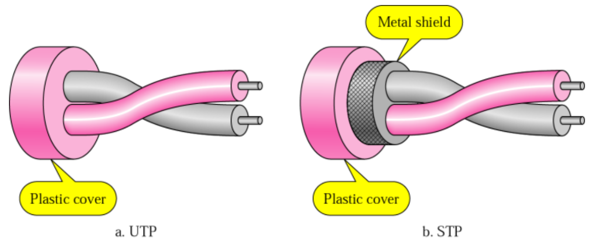




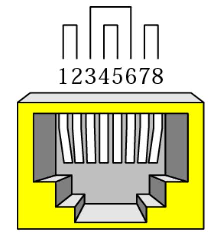
	Analog signal	Digital signal
Analog Data	**AM, FM**	**PCM & Video using codecs**
Digital Data	**ASK, FSK, PSK, QAM**	**LAN Cable Standards (bi-phase, Manchester)**

Review of Ethernet Interface

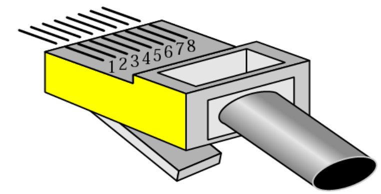
- **Cable: UTP**



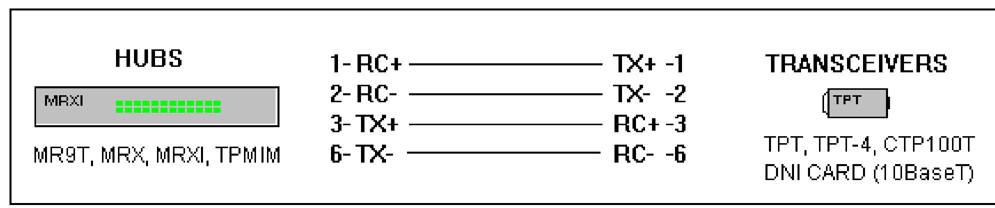
- **Connector: RJ-45**
- **NIC (Network interface card)**



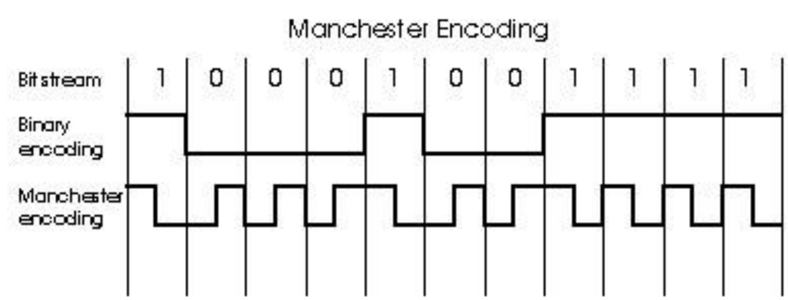
RJ-45 Female



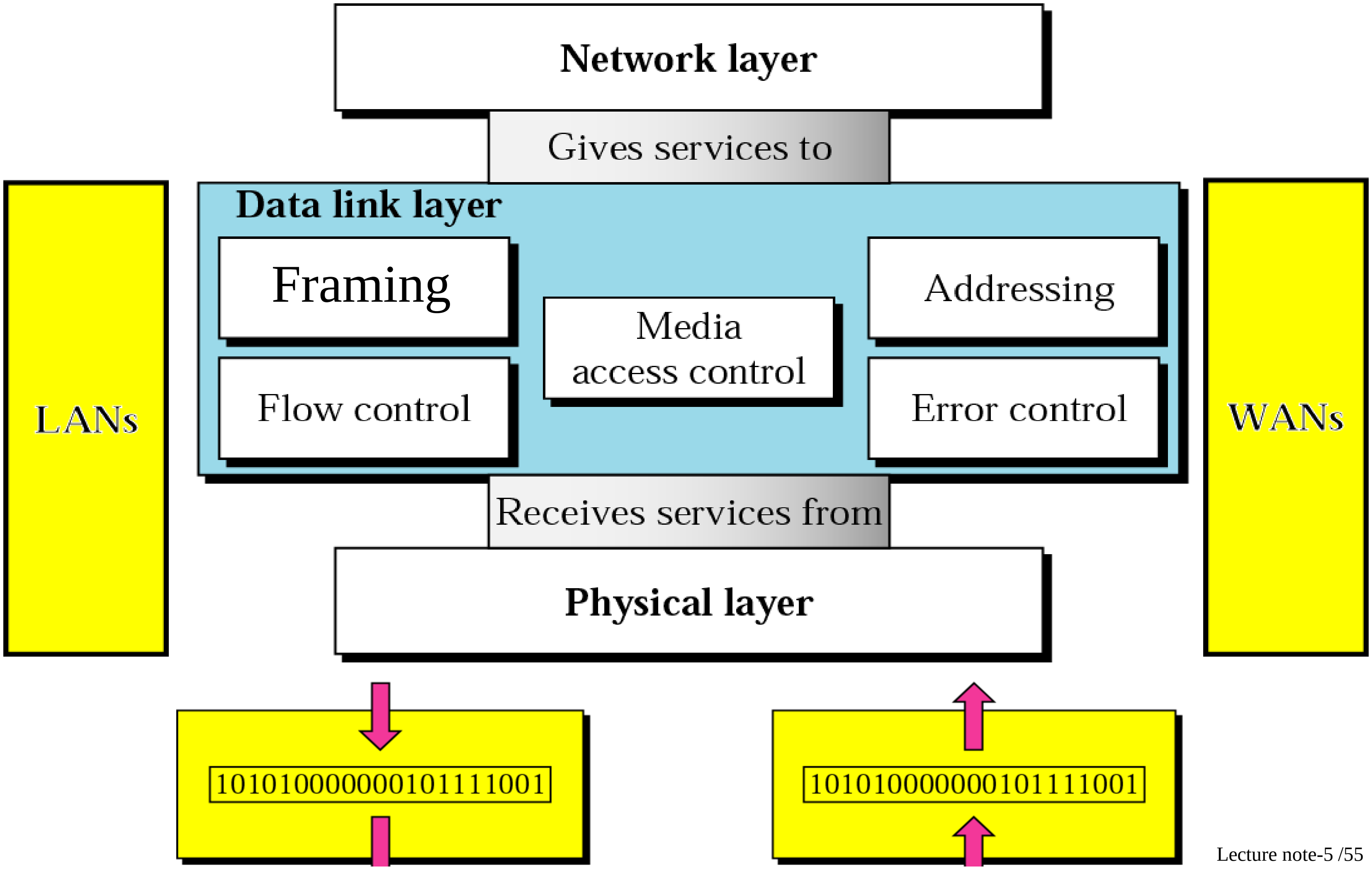
RJ-45 Male



- **LAN encoding is Manchester**

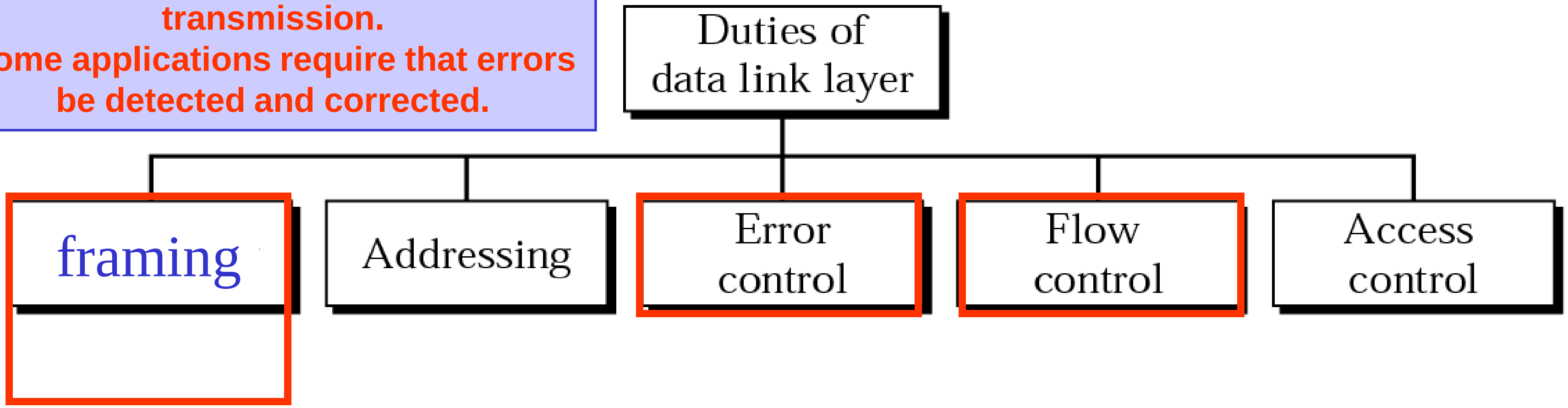


Position of the data-link layer



Data link layer duties

Data can be corrupted during transmission.
Some applications require that errors be detected and corrected.



- **Data link protocols have three functions:**
  – Error Control**: Detecting and correcting transmission errors. (Error & flow)**
  – Media Access Control**: Controlling when computers transmit. Who should send now(Access control)**
  – Message Delineation**: Identifying the beginning and end of a message. (Framing & Addressing)**

Error Detection and Correction

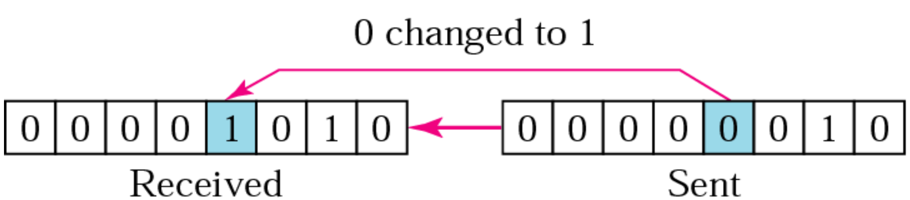
- Error control is handling 1)network errors caused by problems in transmission, including line noise, not human errors.
- 1) Error types include corrupted data and lost data.
- Error control is concerned with:

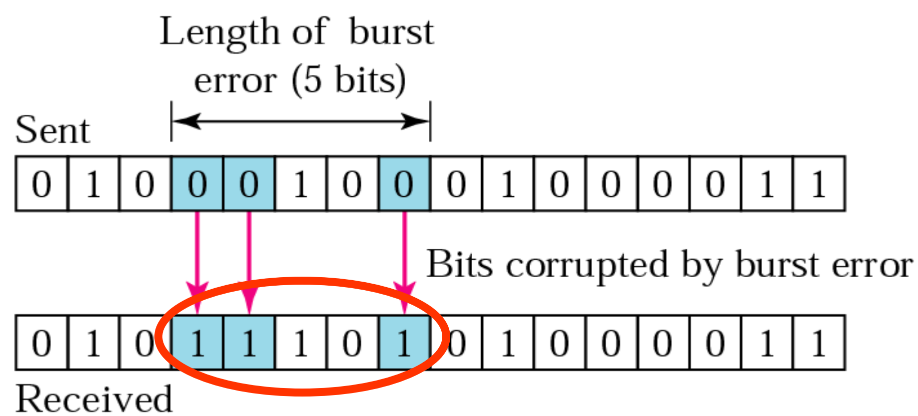
2) detecting and

3) correcting errors.

1) Types of Errors

- In normal transmission environment, the electromagnetic signal flow through the transmission media is subject to unpredictable interference from heat, magnetism, and other forms of electricity
- This interference can change the shape or timing of a signal which will result in altering the meaning of the data
- They are two main types of errors:
  - Single-bit Error
  - Burst Error





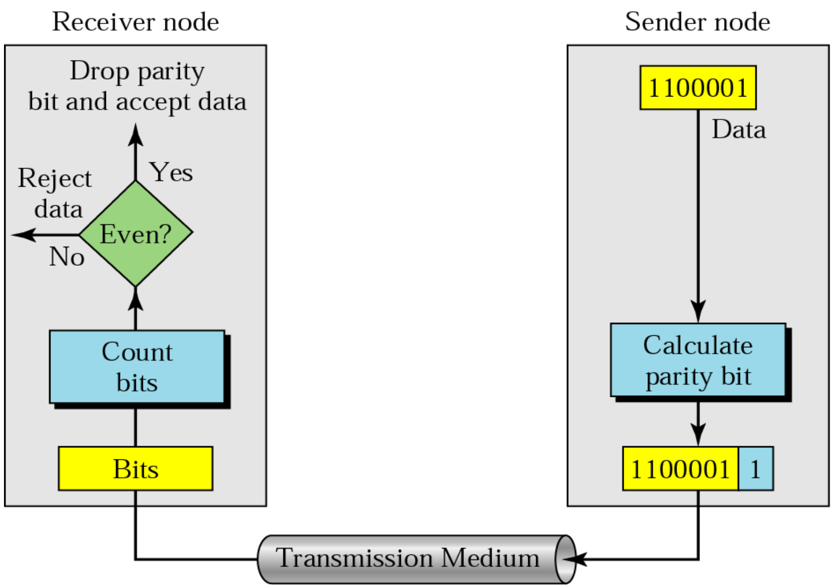
2) Error Detection (Linear Block Code)

- **Four main types of Error Detection mechanisms (redundancy checks) are used:**
  a. Vertical redundancy check (VRC) **(or parity check)**
  b. Longitudinal redundancy check (LRC)
  c. Cyclical redundancy check (CRC)
  d. Checksum

- Others
  - Echo Checking
  - Ignore Parity Checking

a. Vertical Redundancy Check (VRC)

- **Even and Odd Parity**
- **Examples**



Suppose the sender wants to send the word *world*. In ASCII the five characters are coded as

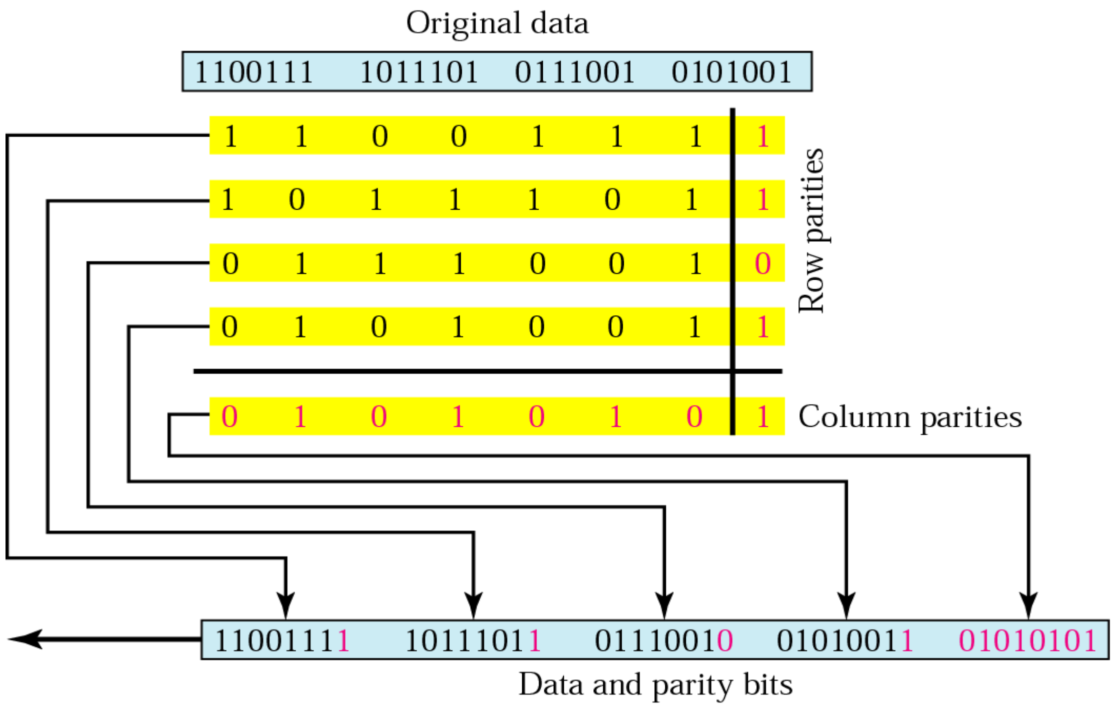
$\boxed{\mathbf{1110111}}$ **1101111 1110010 1101100 1100100**

The following shows the actual bits sent

$\boxed{1110111}$**0** 1101111**0** 1110010**0** 1101100**0** 1100100**1**

b. Longitudinal Redundancy Check (LRC)

- **LRC was developed as an improvement over VRC. LRC adds an additional character of parity checks, called a block control character (BCC) to each block of data.**
- **LRC is a major improvement over VRC, catching over 98% of all errors.**



c. Cyclic Redundancy Check (CRC)

- **Most powerful of redundancy checking techniques**
- **It is based on binary division instead of addition like VRC and LRC (see Figure 10.7)**
- **CRC-16 (99.969% effective) and CRC-32 (99.99%) are in common use today**

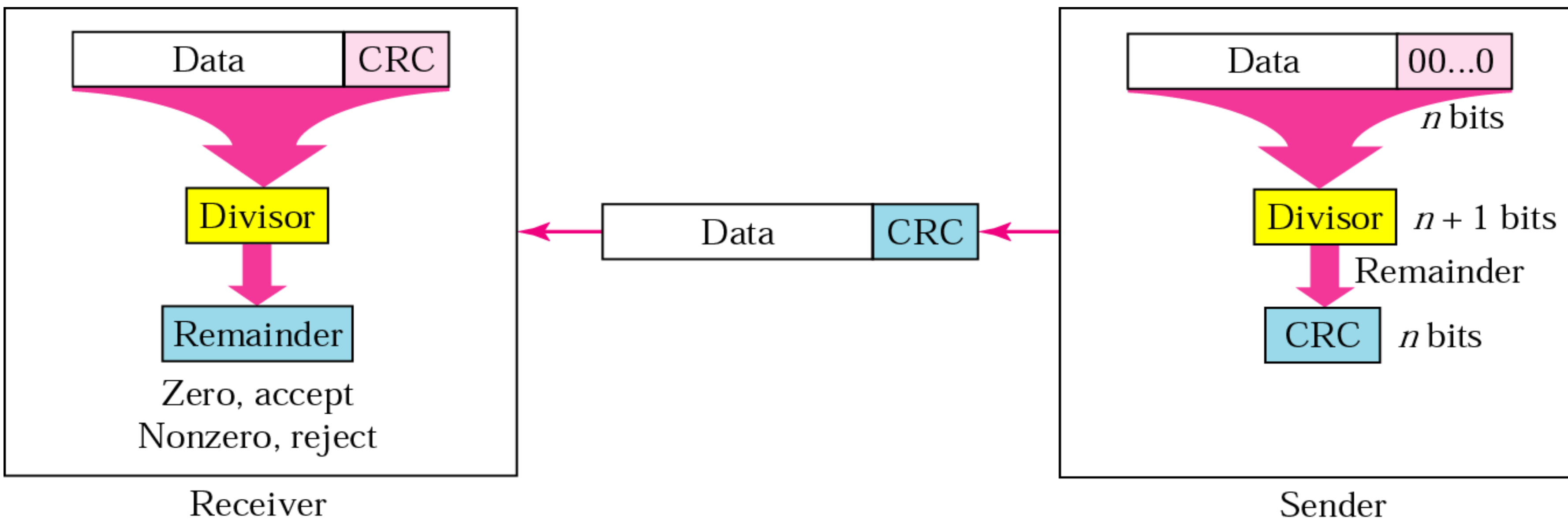
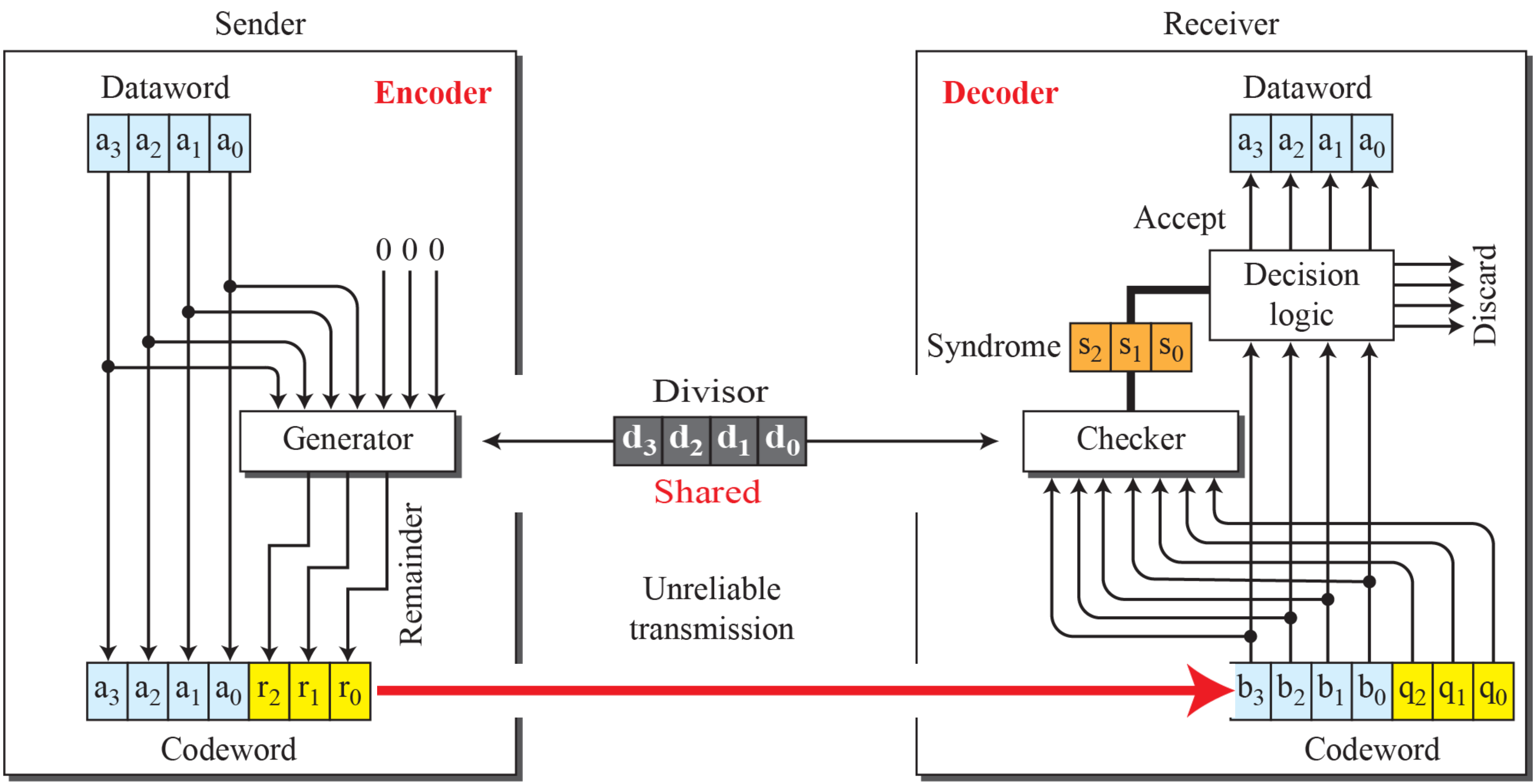
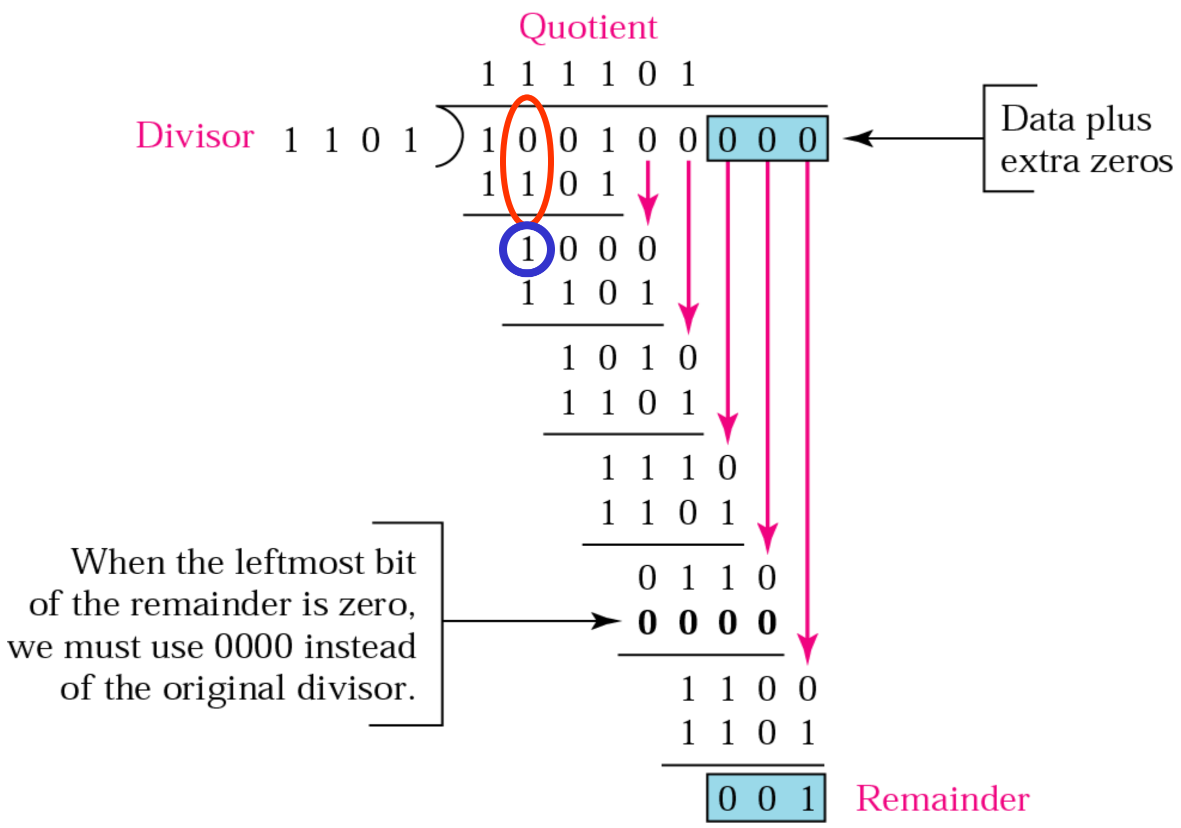


Figure 10.5: CRC encoder and decoder



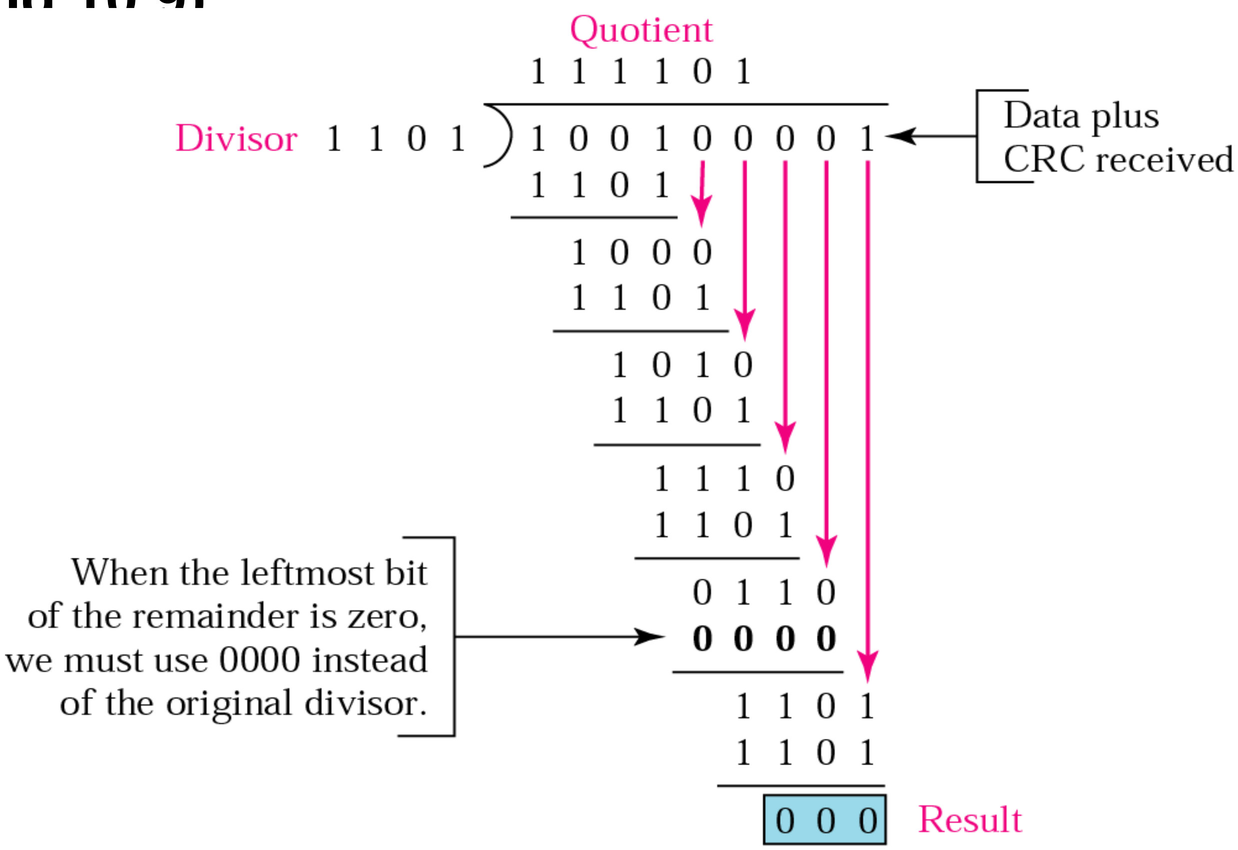
CRC -generation

- **The process of deriving the CRC: (see Figure 10.8)**
  - **Modular 2 division**



CRC -check

- **The process of checking the CRC: (see Figure 10.8 and 10.9)**



CRC-3

Figure 10.15 *Division in CRC encoder*

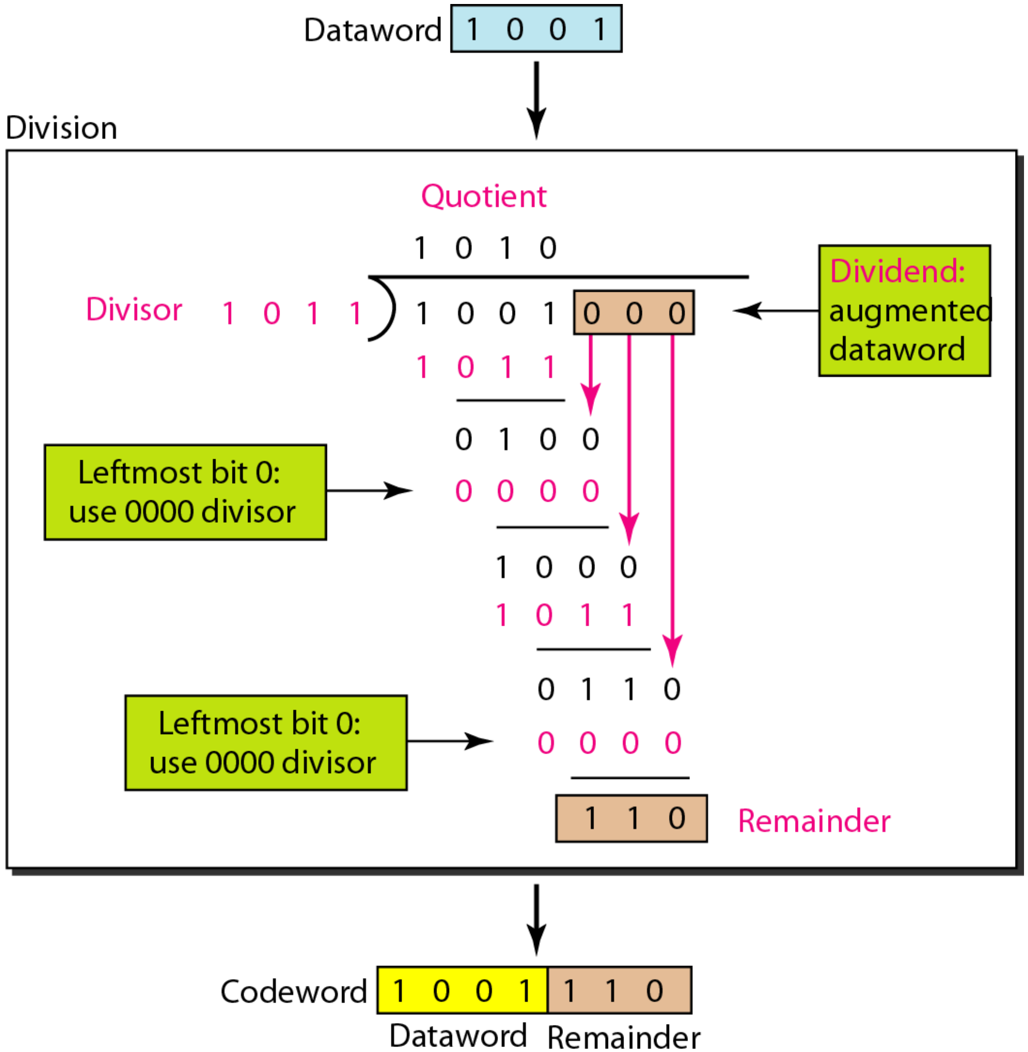
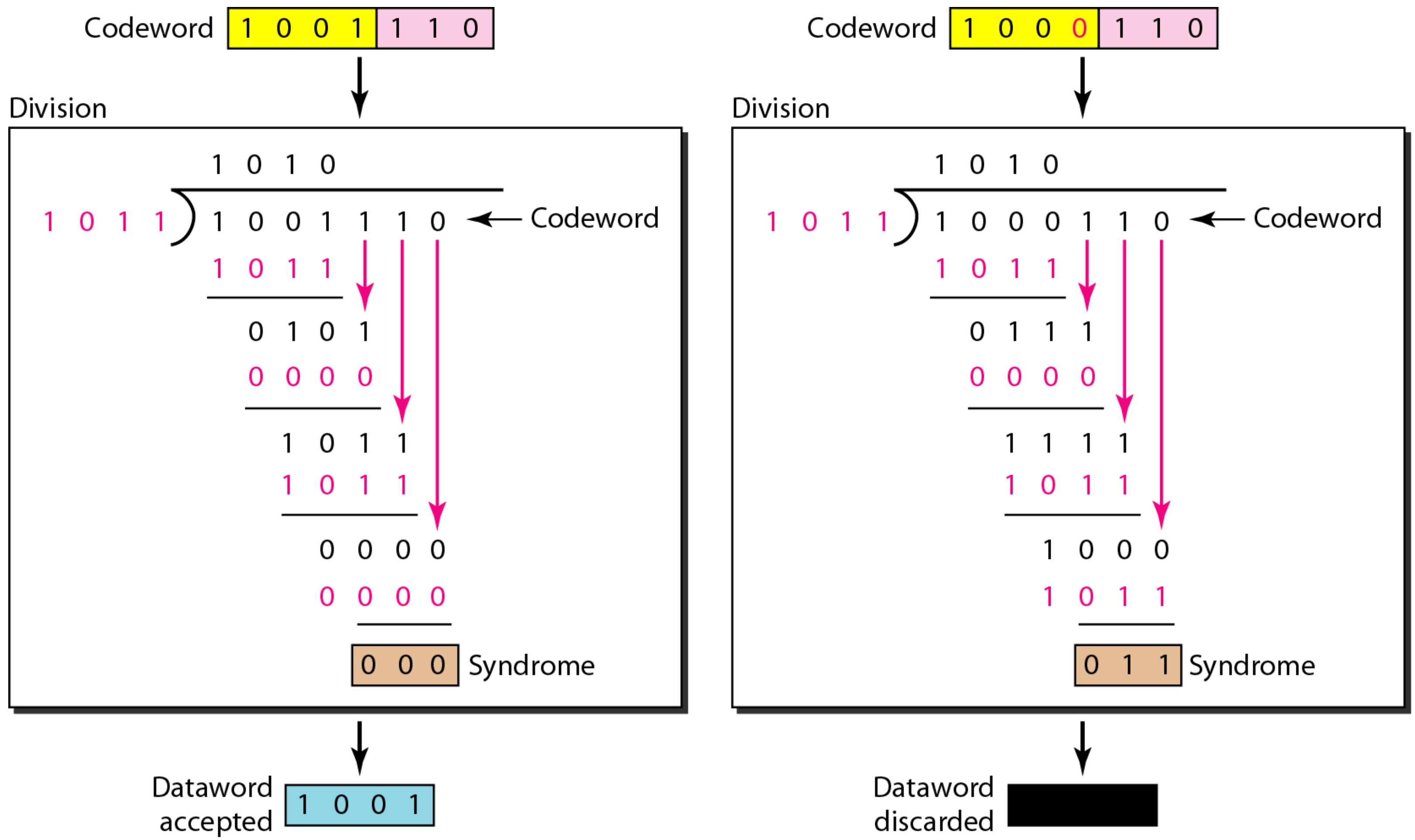


Figure 10.16 *Division in the CRC decoder for two cases*

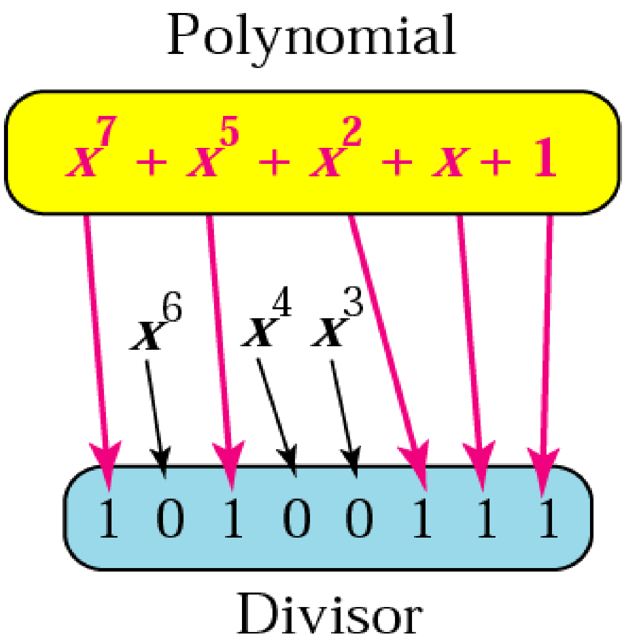


Polynomials

- **The CRC is represented by an algebraic polynomial:**

$$\boxed{x^7 + x^5 + x^2 + x + 1}$$

- **The polynomial format is useful:**
  - **It is short**
  - **It can be used to prove the concept mathematically**
- **A polynomial should be selected to have at least the following properties:**
  - **It should not be divisible by *x*.**
  - **It should be divisible by ($x + 1$)**
    - detect any odd number of errors
  - **At least two terms**
    - Detect all single bit error



Polynomial additions, subtraction, multiplication, division

- **Primitive Polynomial**

$$(x^2 + 1) = (x + 1)(x + 1) = x^2 + x + x + 1$$
$$= x^2 + (1+1)x + 1 \qquad (1+1) \bmod 2 = 0$$

- **Polynomial addition:**

$$(x^7 + x^6 + 1) + (x^6 + x^5) = x^7 + x^6 + x^6 + x^5 + 1$$
$$= x^7 + (1+1)x^6 + x^5 + 1$$
$$= x^7 + x^5 + 1 \qquad (1+1) \bmod 2 = 0$$

- **Polynomial multiplication:**

$$(x^2 + x + 1) + (x + 1) \qquad = x(x^2 + x + 1) + 1(x^2 + x + 1)$$
$$= (x^3 + x^2 + x) + (x^2 + x + 1)$$
$$= x^3 + (1+1)x^2 + (1+1)x + 1$$
$$= x^3 + 1$$

Standards

Table 10.4 Standard polynomials

Name	*Polynomial*	*Used in*
CRC-8	$x^8 + x^2 + x + 1$ 100000111	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$ 11000110101	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$ 10001000000100001	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ 100000100110000010001110110110111	LANs

Examples

- **CRC-16 Polynomial Generator/code:**

$$G(x)= x^{16} + x^{15} + x^2 + 1 = (x + 1)(x^{15} + x + 1)$$

$$= x^{16} + x^{15} + x^2 + (1+1)x + 1 \qquad (1+1) \bmod 2 = 0$$

- **CRC-12 Polynomial Generator/code:**

$$G(x)= x^{12} + x^{11} + x^3 + x^2 + x + 1 = (x + 1)\,(?)$$

Example 10.15

Which of the following g(x) values guarantees that a single-bit error is caught? For each case, what is the error that cannot be caught?

a. $x + 1$ b. x^3 c. 1

Solution

a. ***No x^i can be divisible by $x + 1$. $x^i/(x + 1)$ always has a reminder. Any single-bit error can be caught.***

b. ***If i is equal to or greater than 3, x^i is divisible by g(x). All single-bit errors in positions 1 to 3 are caught.***

c. ***All values of i make x^i divisible by g(x). No single-bit error can be caught. This g(x) is useless.***

Example 10.17

Find the suitability of the following generators in relation to burst errors of different lengths.

a. $x^6 + 1$ b. $x^{18} + x^7 + x + 1$ c. $x^{32} + x^{23} + x^7 + 1$

Solution

a. ***This generator can detect all burst errors with a length less than or equal to 6 bits; 3 out of 100 burst errors with length 7 will slip by; 16 out of 1000 burst errors of length 8 or more will slip by.***

b. ***This generator can detect all burst errors with a length less than or equal to 18 bits; 8 out of 1 million burst errors with length 19 will slip by; 4 out of 1 million burst errors of length 20 or more will slip by.***

c. ***This generator can detect all burst errors with a length less than or equal to 32 bits; 5 out of 10 billion burst errors with length 33 will slip by; 3 out of 10 billion burst errors of length 34 or more will slip by.***

CRC의 에러 검출 능력

- 모든 단일비트 에러
- **G(X)**가 최소한 **3**개의 항을 가지는 경우 모든 두 비트 에러
- **G(X)**가 **(X+1)**의 인수를 가지는 경우 모든 홀수개의 에러
- 길이가 **FCS**보다 짧은 모든 **burst** 에러
- 길이가 **FCS**보다 긴 대부분의 **burst** 에러
- 에러발생 패턴이 **G(X)**로 나누어 떨어지는 경우 에러검출 불가
- 생성다항식 **(Generator Polynomial), G(X),** 의 예

CRC-16	$G(X) = X^{16} + X^{15} + X^{2} + 1$
CRC-CCITT	$G(X) = X^{16} + X^{12} + X^{5} + 1$
CRC-32 LAN	$G(X) = X^{32} + X^{26} + X^{23} + X^{16} + X^{12} + X^{11} + X^{10}$ $+ X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1$

Software Implementation

Java 언어

```
import java.util.zip.CRC32; /* Java class CRC32 */

 private byte[] getCRC(int length)
   {
          byte[] tempCRC = new byte[4];

          CRC32 crc32 = new CRC32();
          crc32.update(frame, 0, length);/* Generate CRC of the frame */
          long temp = crc32.getValue();   /* return CRC value */

      tempCRC[3] = (byte)(int)(temp & 255L);
      tempCRC[2] = (byte)(int)(temp >>> 8 & 255L);
      tempCRC[1] = (byte)(int)(temp >>> 16 & 255L);
      tempCRC[0] = (byte)(int)(temp >>> 24 & 255L);

      return tempCRC;
   }
```

Hardware Implementation

Figure 10.17 *Hardwired design of the divisor in CRC*

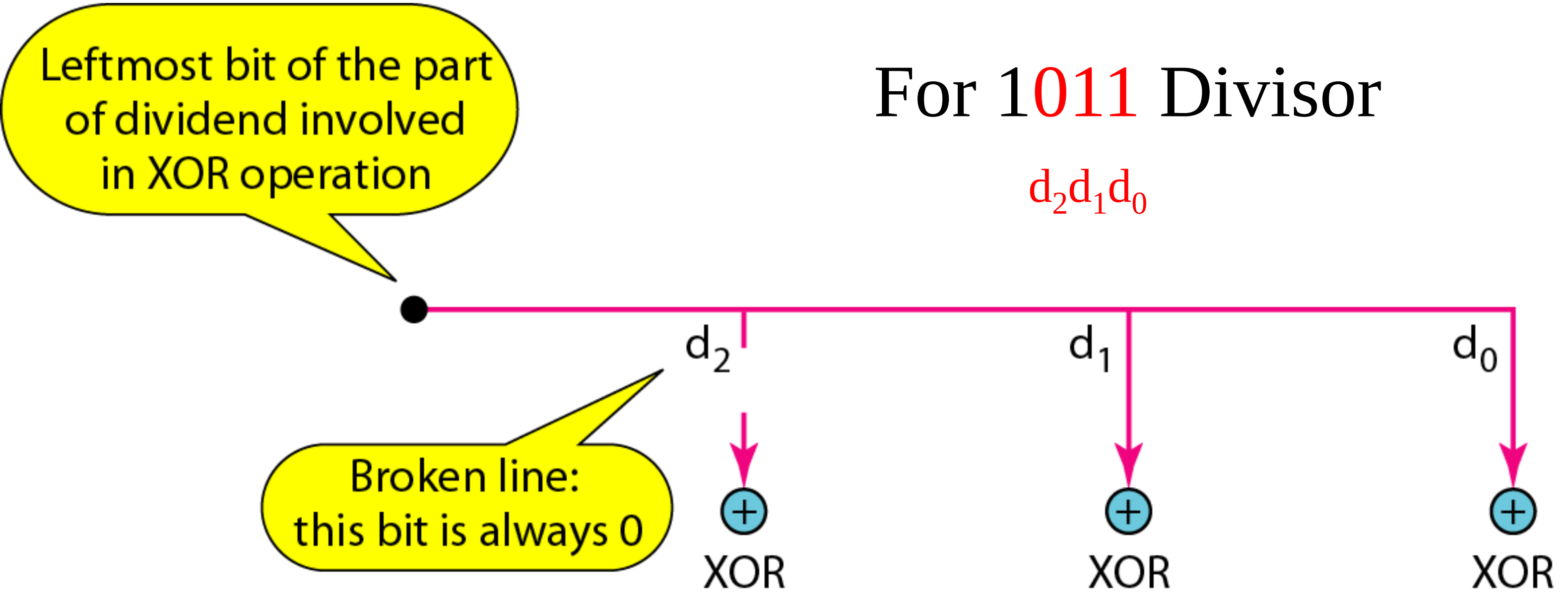


Figure 10.19 *The CRC encoder design using shift registers*

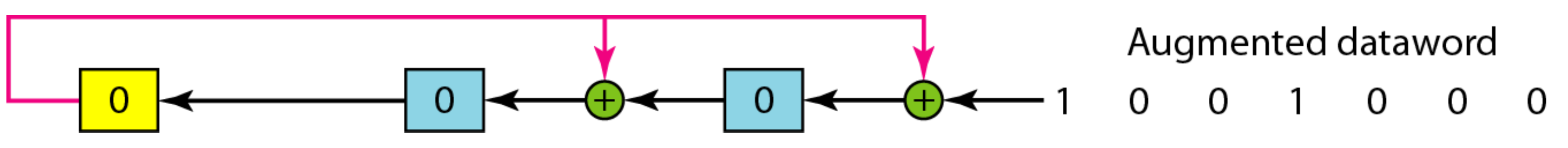


Figure 10.18 *Simulation of division in CRC encoder*

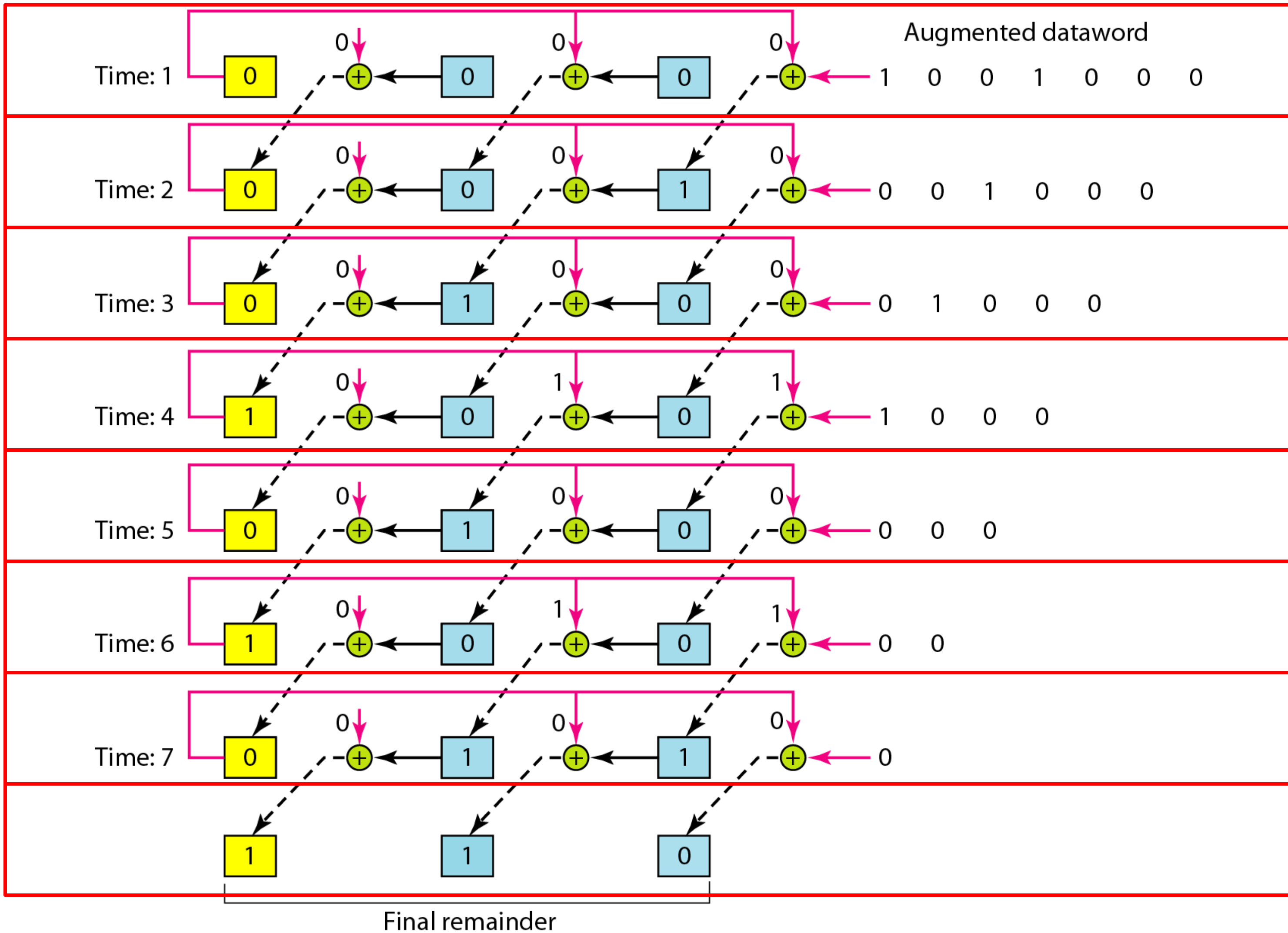
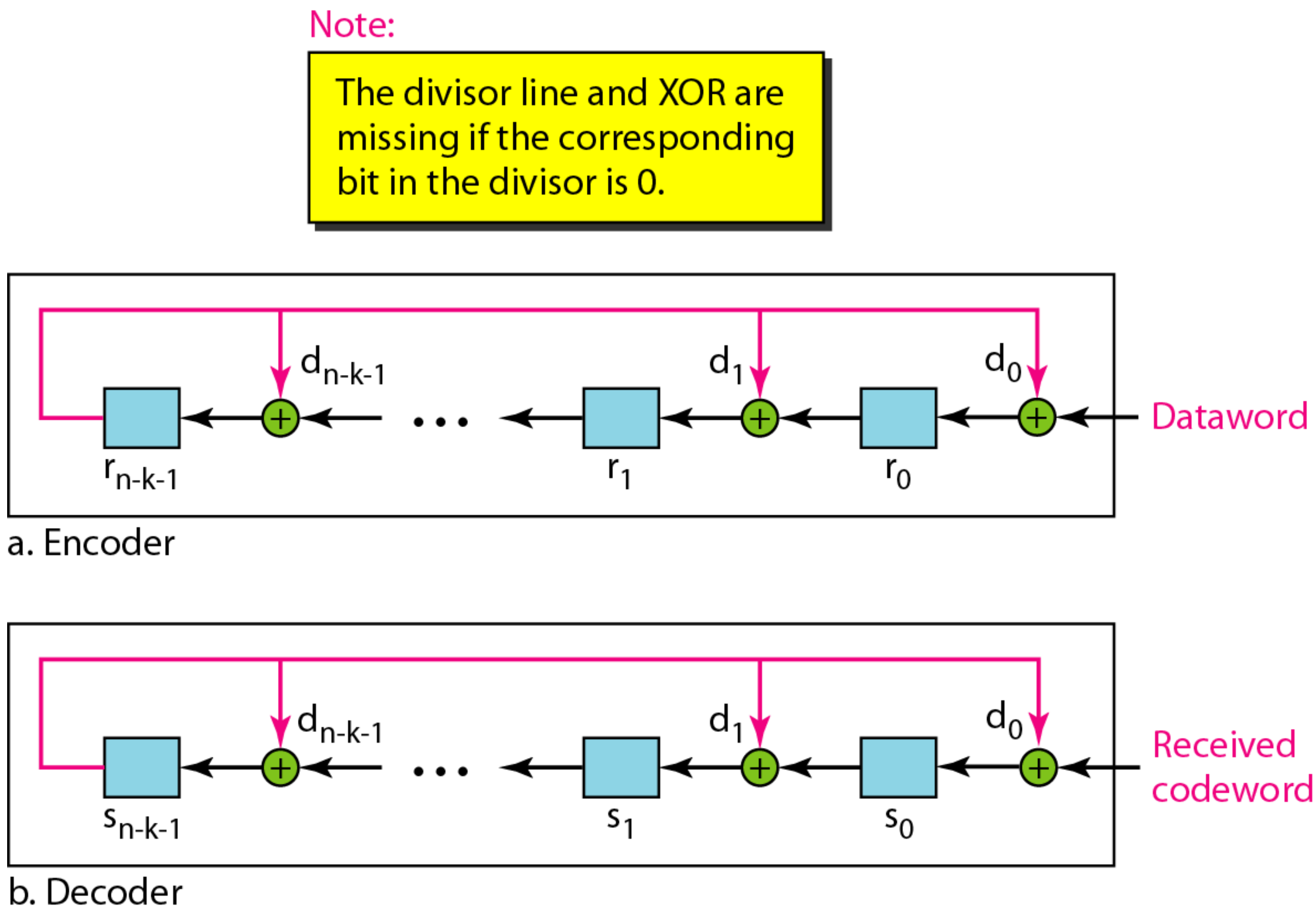


Figure 10.20 *General design of encoder and decoder of a CRC code*



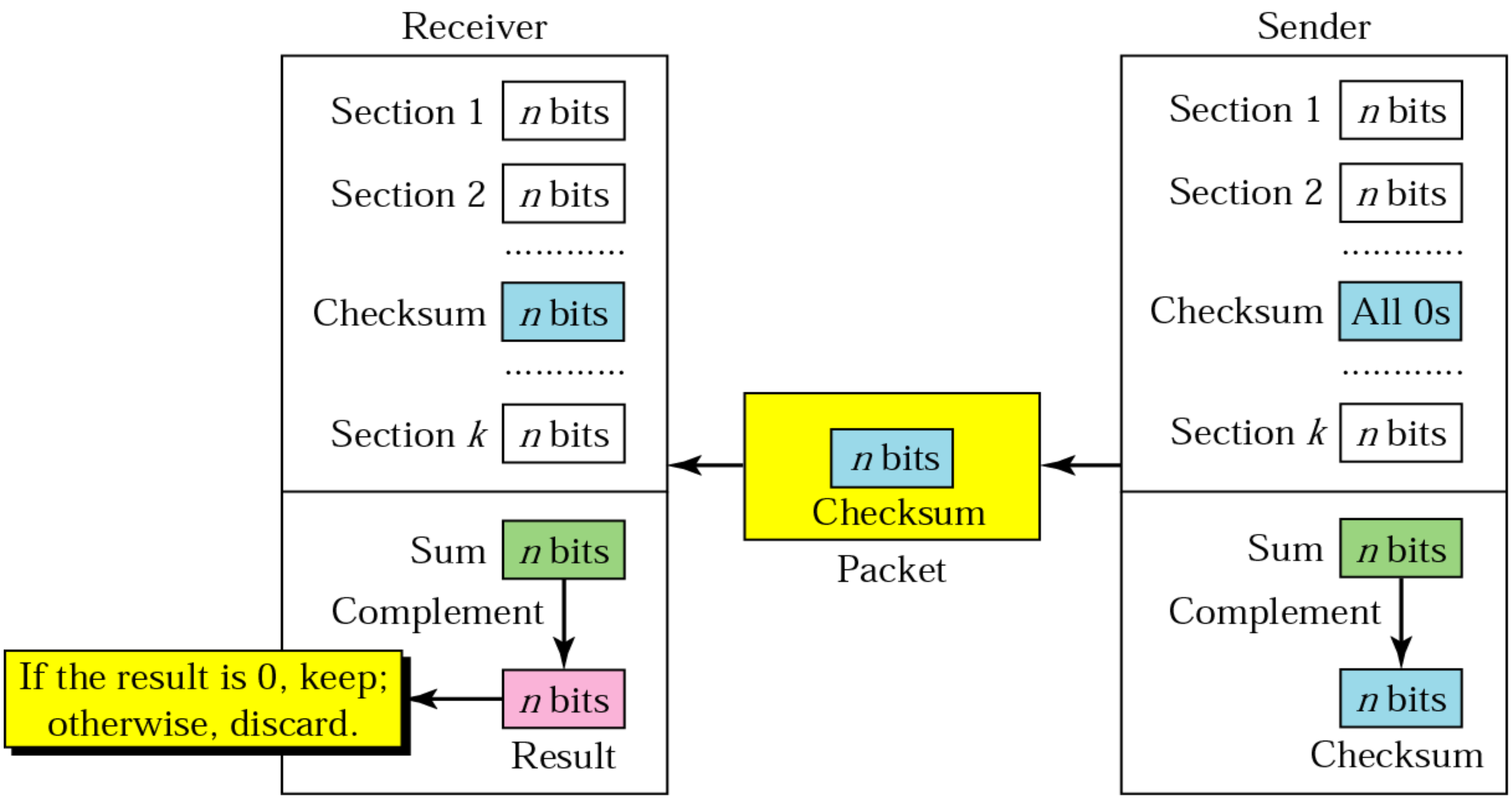
Cyclic Redundancy Checking Implementation with the compact table-driven

```
// calculate a checksum on a buffer -- start address = p, length = bytelength
uint32_t crc32_byte(uint8_t *p, uint32_t bytelength) {
        uint32_t crc = 0xffffffff;
        while (bytelength-- !=0) crc = poly8_lookup[((uint8_t) crc ^ *(p++))] ^ (crc >> 8);
        // return (~crc); also works
        return (crc ^ 0xffffffff); }
```

```
uint32_t poly8_lookup[256] = { 0, 0x77073096, 0xEE0E612C, 0x990951BA, 0x076DC419, 0x706AF48F, 0xE963A535, 0x9E6495A3,
0x0EDB8832, 0x79DCB8A4, 0xE0D5E91E, 0x97D2D988, 0x09B64C2B, 0x7EB17CBD, 0xE7B82D07, 0x90BF1D91, 0x1DB71064,
0x6AB020F2, 0xF3B97148, 0x84BE41DE, 0x1ADAD47D, 0x6DDDE4EB, 0xF4D4B551, 0x83D385C7, 0x136C9856, 0x646BA8C0,
0xFD62F97A, 0x8A65C9EC, 0x14015C4F, 0x63066CD9, 0xFA0F3D63, 0x8D080DF5, 0x3B6E20C8, 0x4C69105E, 0xD56041E4,
0xA2677172, 0x3C03E4D1, 0x4B04D447, 0xD20D85FD, 0xA50AB56B, 0x35B5A8FA, 0x42B2986C, 0xDBBBC9D6, 0xACBCF940,
0x32D86CE3, 0x45DF5C75, 0xDCD60DCF, 0xABD13D59, 0x26D930AC, 0x51DE003A, 0xC8D75180, 0xBFD06116, 0x21B4F4B5,
0x56B3C423, 0xCFBA9599, 0xB8BDA50F, 0x2802B89E, 0x5F058808, 0xC60CD9B2, 0xB10BE924, 0x2F6F7C87, 0x58684C11,
0xC1611DAB, 0xB6662D3D, 0x76DC4190, 0x01DB7106, 0x98D220BC, 0xEFD5102A, 0x71B18589, 0x06B6B51F, 0x9FBFE4A5,
0xE8B8D433, 0x7807C9A2, 0x0F00F934, 0x9609A88E, 0xE10E9818, 0x7F6A0DBB, 0x086D3D2D, 0x91646C97, 0xE6635C01,
0x6B6B51F4, 0x1C6C6162, 0x856530D8, 0xF262004E, 0x6C0695ED, 0x1B01A57B, 0x8208F4C1, 0xF50FC457, 0x65B0D9C6,
0x12B7E950, 0x8BBEB8EA, 0xFCB9887C, 0x62DD1DDF, 0x15DA2D49, 0x8CD37CF3, 0xFBD44C65, 0x4DB26158, 0x3AB551CE,
0xA3BC0074, 0xD4BB30E2, 0x4ADFA541, 0x3DD895D7, 0xA4D1C46D, 0xD3D6F4FB, 0x4369E96A, 0x346ED9FC, 0xAD678846,
0xDA60B8D0, 0x44042D73, 0x33031DE5, 0xAA0A4C5F, 0xDD0D7CC9, 0x5005713C, 0x270241AA, 0xBE0B1010, 0xC90C2086,
0x5768B525, 0x206F85B3, 0xB966D409, 0xCE61E49F, 0x5EDEF90E, 0x29D9C998, 0xB0D09822, 0xC7D7A8B4, 0x59B33D17,
0x2EB40D81, 0xB7BD5C3B, 0xC0BA6CAD, 0xEDB88320, 0x9ABFB3B6, 0x03B6E20C, 0x74B1D29A, 0xEAD54739,
0x9DD277AF, 0x04DB2615, 0x73DC1683, 0xE3630B12, 0x94643B84, 0x0D6D6A3E, 0x7A6A5AA8, 0xE40ECF0B, 0x9309FF9D,
0x0A00AE27, 0x7D079EB1, 0xF00F9344, 0x8708A3D2, 0x1E01F268, 0x6906C2FE, 0xF762575D, 0x806567CB, 0x196C3671,
0x6E6B06E7, 0xFED41B76, 0x89D32BE0, 0x10DA7A5A, 0x67DD4ACC, 0xF9B9DF6F, 0x8EBEEFF9, 0x17B7BE43, 0x60B08ED5,
0xD6D6A3E8, 0xA1D1937E, 0x38D8C2C4, 0x4FDFF252, 0xD1BB67F1, 0xA6BC5767, 0x3FB506DD, 0x48B2364B, 0xD80D2BDA,
0xAF0A1B4C, 0x36034AF6, 0x41047A60, 0xDF60EFC3, 0xA867DF55, 0x316E8EEF, 0x4669BE79, 0xCB61B38C, 0xBC66831A,
0x256FD2A0, 0x5268E236, 0xCC0C7795, 0xBB0B4703, 0x220216B9, 0x5505262F, 0xC5BA3BBE, 0xB2BD0B28, 0x2BB45A92,
0x5CB36A04, 0xC2D7FFA7, 0xB5D0CF31, 0x2CD99E8B, 0x5BDEAE1D, 0x9B64C2B0, 0xEC63F226, 0x756AA39C, 0x026D930A,
0x9C0906A9, 0xEB0E363F, 0x72076785, 0x05005713, 0x95BF4A82, 0xE2B87A14, 0x7BB12BAE, 0x0CB61B38, 0x92D28E9B,
0xE5D5BE0D, 0x7CDCEFB7, 0x0BDBDF21, 0x86D3D2D4, 0xF1D4E242, 0x68DDB3F8, 0x1FDA836E, 0x81BE16CD,
0xF6B9265B, 0x6FB077E1, 0x18B74777, 0x88085AE6, 0xFF0F6A70, 0x66063BCA, 0x11010B5C, 0x8F659EFF, 0xF862AE69,
0x616BFFD3, 0x166CCF45, 0xA00AE278, 0xD70DD2EE, 0x4E048354, 0x3903B3C2, 0xA7672661, 0xD06016F7, 0x4969474D,
0x3E6E77DB, 0xAED16A4A, 0xD9D65ADC, 0x40DF0B66, 0x37D83BF0, 0xA9BCAE53, 0xDEBB9EC5, 0x47B2CF7F,
0x30B5FFE9, 0xBDBDF21C, 0xCABAC28A, 0x53B39330, 0x24B4A3A6, 0xBAD03605, 0xCDD70693, 0x54DE5729, 0x23D967BF,
0xB3667A2E, 0xC4614AB8, 0x5D681B02, 0x2A6F2B94, 0xB40BBE37, 0xC30C8EA1, 0x5A05DF1B, 0x2D02EF8D };
```

d. Checksum

- **Used by higher protocol (TCP/UDP)**



Checksum Sender Example 1.

- **The following block of 16 bits using checksum of 8 bits:**

10101001 00111001

- **The numbers are added using 1's complement:**

10101001

00111001

11100010

(sum)

00011101

(checksum)

- **The pattern send is: 10101001 00111001 00011101**

Checksum Receiver Example

- **If the receiver received with no errors:**

10101001 00111001 00011101

- **Add the three section together:**

10101001

00111001

00011101

11111111

(sum)

00000000

(complement) all 0s means the pattern is OK

Checksum Receiver Example

- **If the receiver received with errors:**

  10101*__11__*1 *__11__*111001 00011101

- **Add the three section together:**

		10101111
		11111001
		00011101

Result	**1**	**11000101**
Carry		**1**

Sum		**11000110**
Complement		**00111001 (means that the pattern is corrupted)**

Checksum Sender Example 2.

- The following block of 16 bits using checksum of 8 bits:

  10101001 00111001 11101001 10111001

- The numbers are added using 1's complement:

```
10_
 1_
110_
__1_
__10_
_____0_
_____ 0_
      10_
10101001
00111001
11101001
10111001
--------------
10000100
      10
10000110
01111001
(checksum)
```

- The pattern send is: 10101001 00111001 11101001 10111001 01111001

Checksum Receiver Example 2

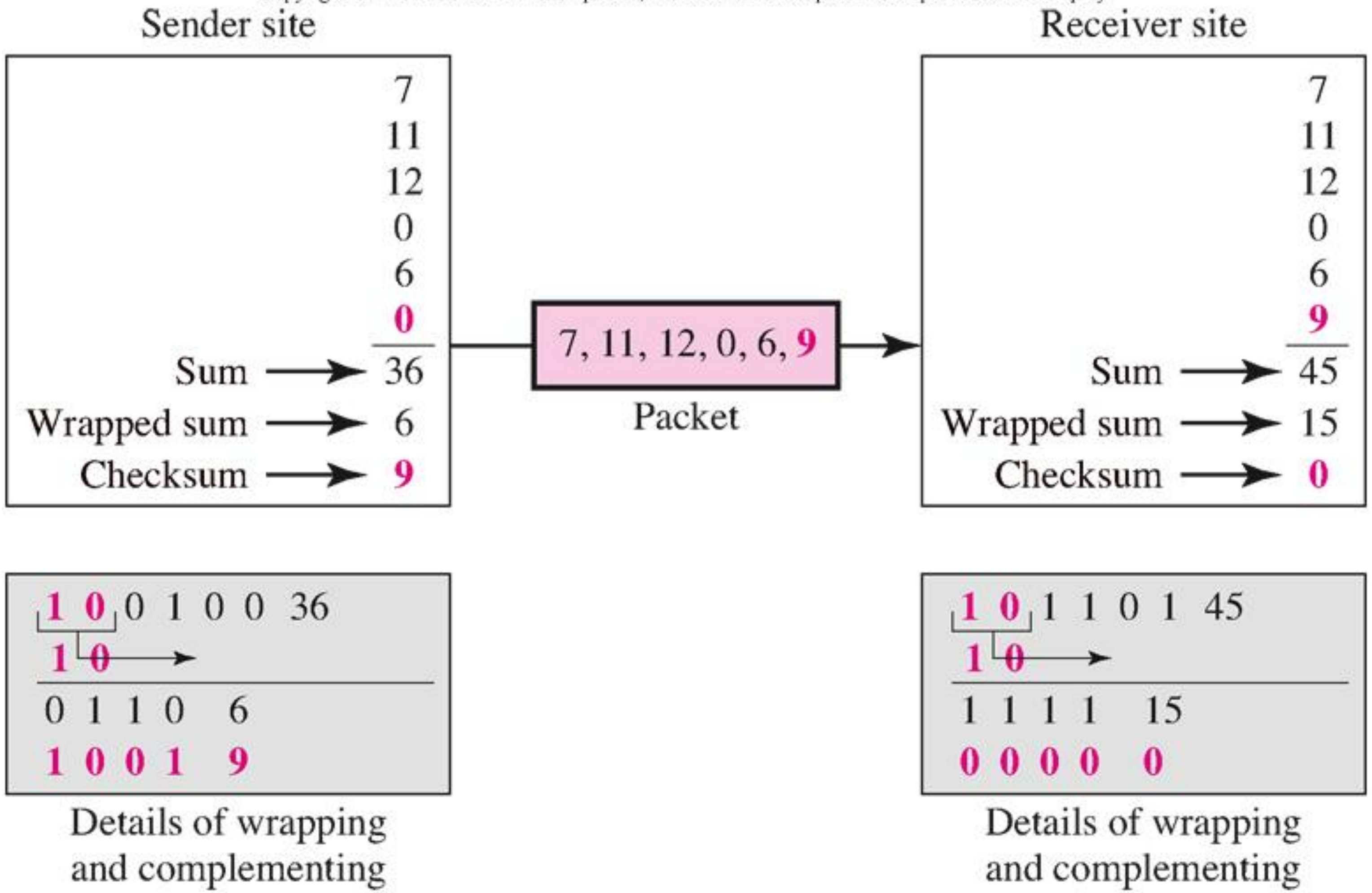
- **If the receiver received with no errors:**
  10101001 00111001 11101001 10111001 01111001
- **Add the three section together:**

```
     10_
      1________
      11_______
      __1______
      __10_____
      ____0____
      _____0___
      ______10_
      10101001
      00111001
      11101001
      10111001
      01111001
     ---------------
      11111101
            10
(sum)---------------
      11111111
     ---------------
      00000000
```

(complement) all 0s means the pattern is OK

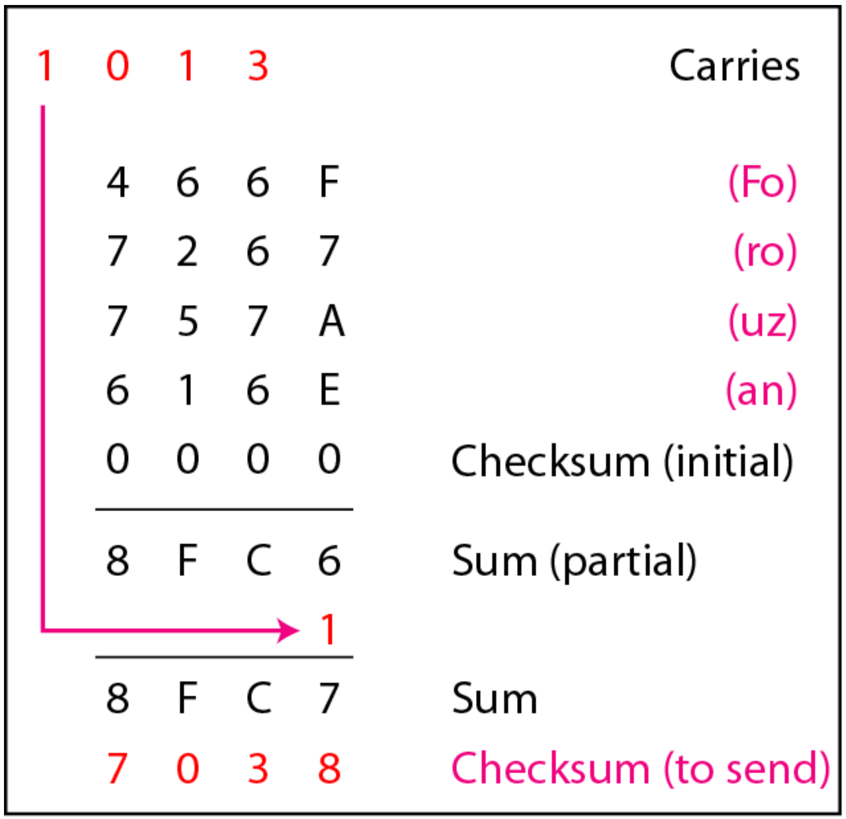
Example of CheckSum

Figure 10.24 *Example 10.22*

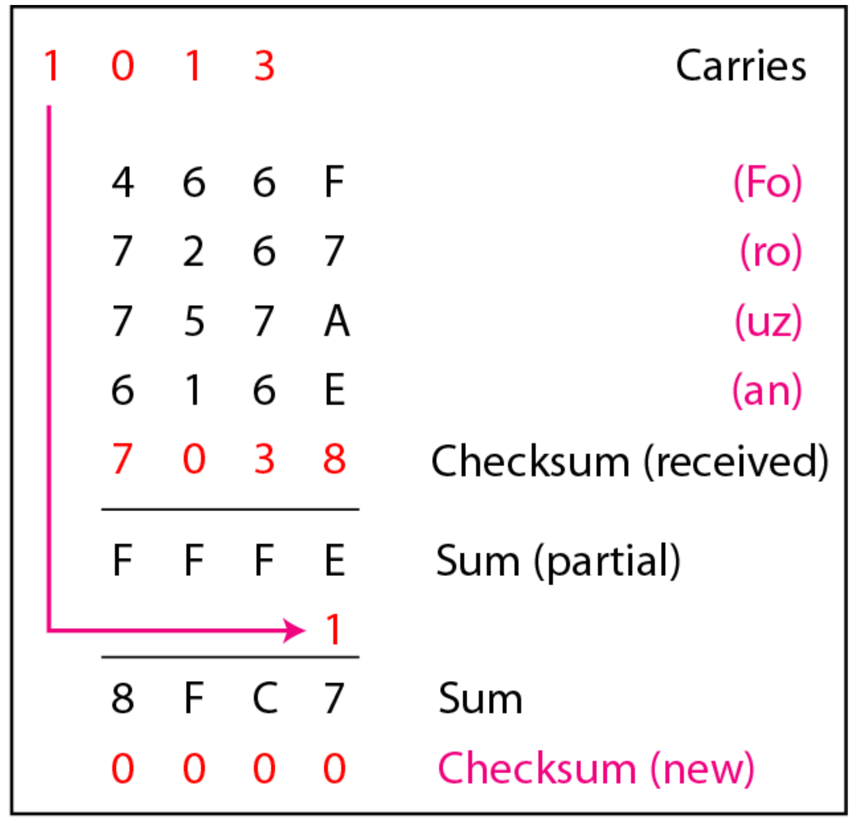


Example of CheckSum

Figure 10.24 ***Example 10.23***



a. Checksum at the sender site



a. Checksum at the receiver site

ISBN checksum: checksum for 10-digit ISBN

- Given 9 digit product code. Starting at leftmost digit:
- multiply corresponding digit by 10, 9, 8, ... down to 2 inclusive
- add the resulting numbers: add digit 10 such that the result is divisible by 11
  the number 10 is written as X

e.g., 0-201-61586-X is valid. The last digit has to be 10 (= X).

$10*0 + 9*2 + 8*0 + 7*1 + 6*6 + 5*1 + 4*5 + 3*8 + 6*2 + 1*10 = 122 + 10 = 132 = 12*11$

- **detects transpositions and single digit errors**
- **an error e at position I gives as result the value eI modulo 11 $\neq$ 0**

- **detection of an transposition error of A and B:**
  $A\ I + B\ (I-1) \rightarrow BI + A(I-1) = AI + B(I-1) - A + B$
  the value $(- A + B) \neq 0$ modulo 11 for $A \neq B$

Bookland EAN Symbol

ISBN Number

10 digit number unique to each publication.
ISBN numbers are issued by the ISBN agency.

Price Add-On

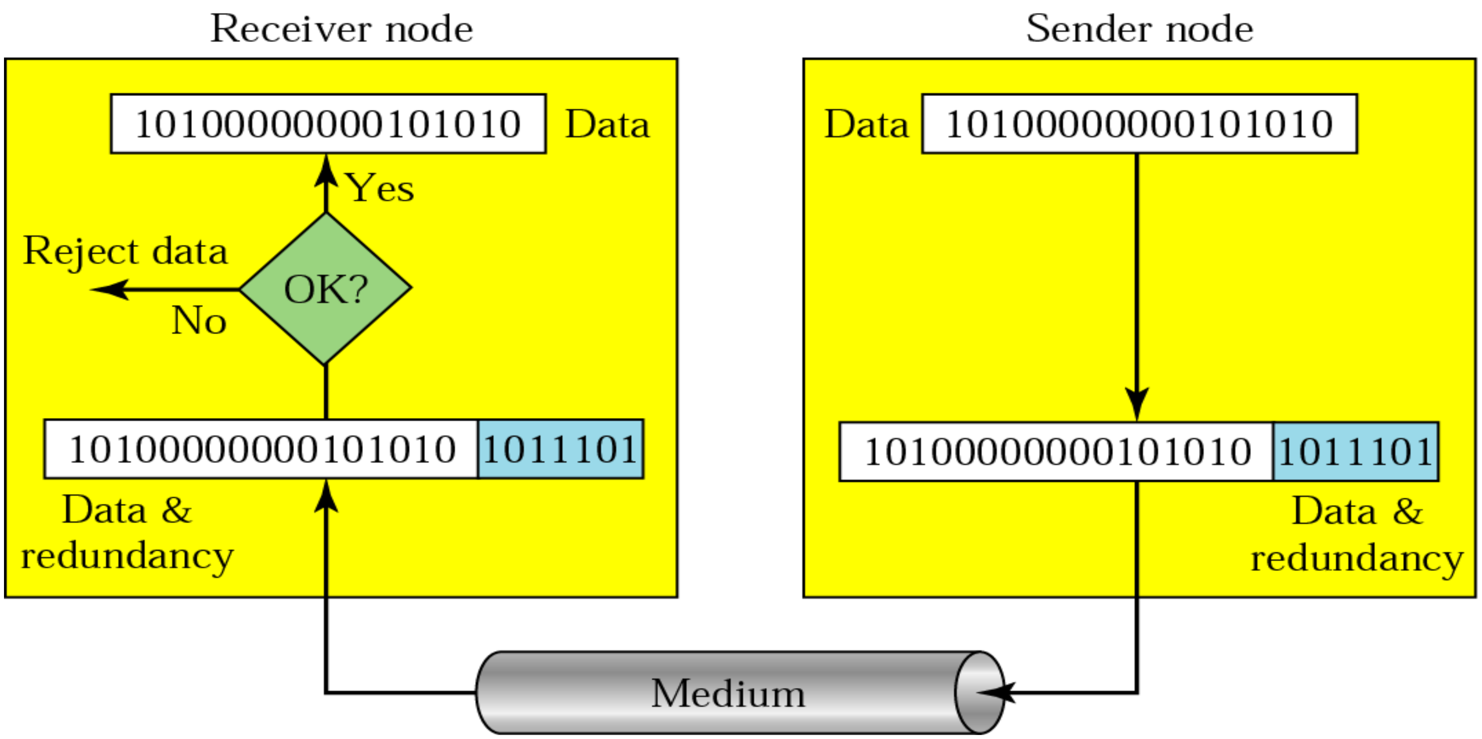
The first digit indicates currency.
The following four numbers encode the suggested retail price.
A null code of "90000" indicates that no price has been specified.

EAN Number

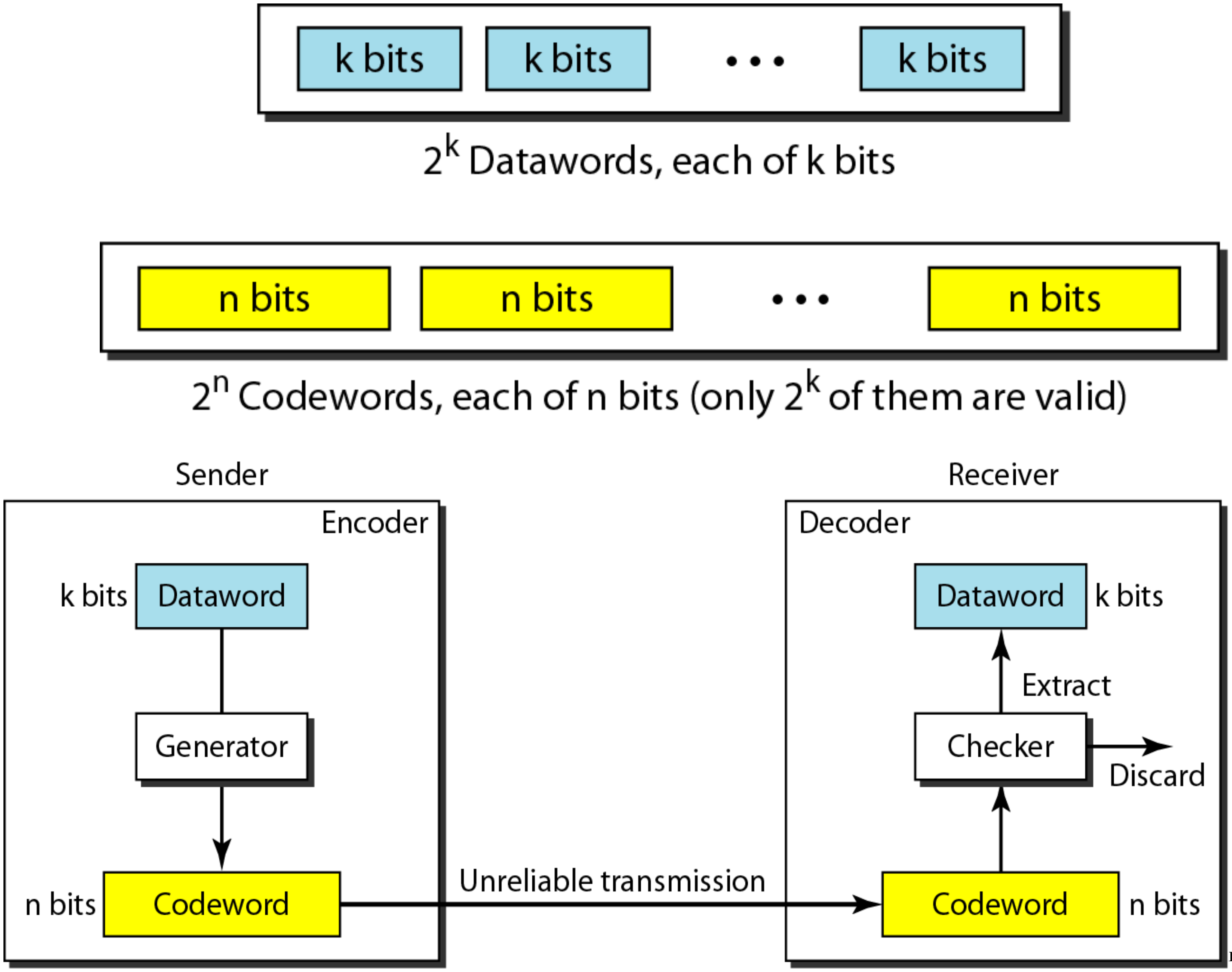
13 digit number derived from the ISBN
Will always begin with "978", followed by
the first 9 digits of the ISBN and a new check digit.

Redundancy

To detect or correct errors, we need to send extra (redundant) bits with data.



BLOCK CODING



Example 10.2 BLOCK CODING

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

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