Data Communication

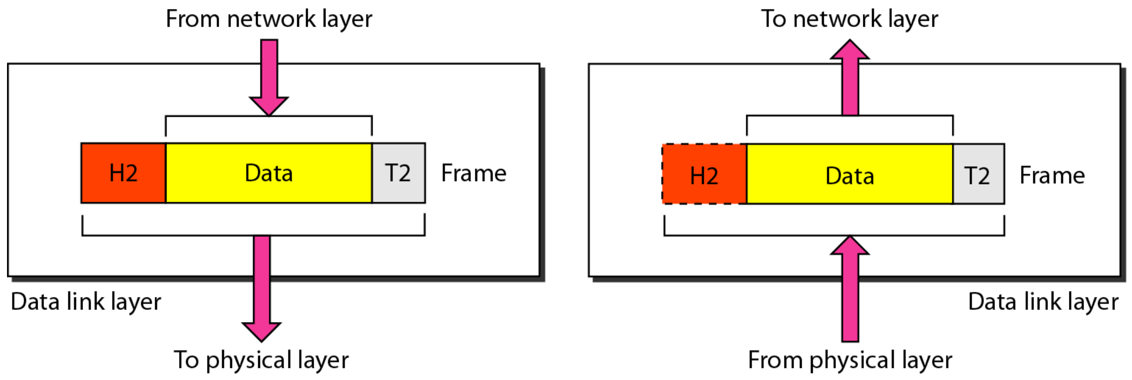
#12 Data Link Layer

Error Control

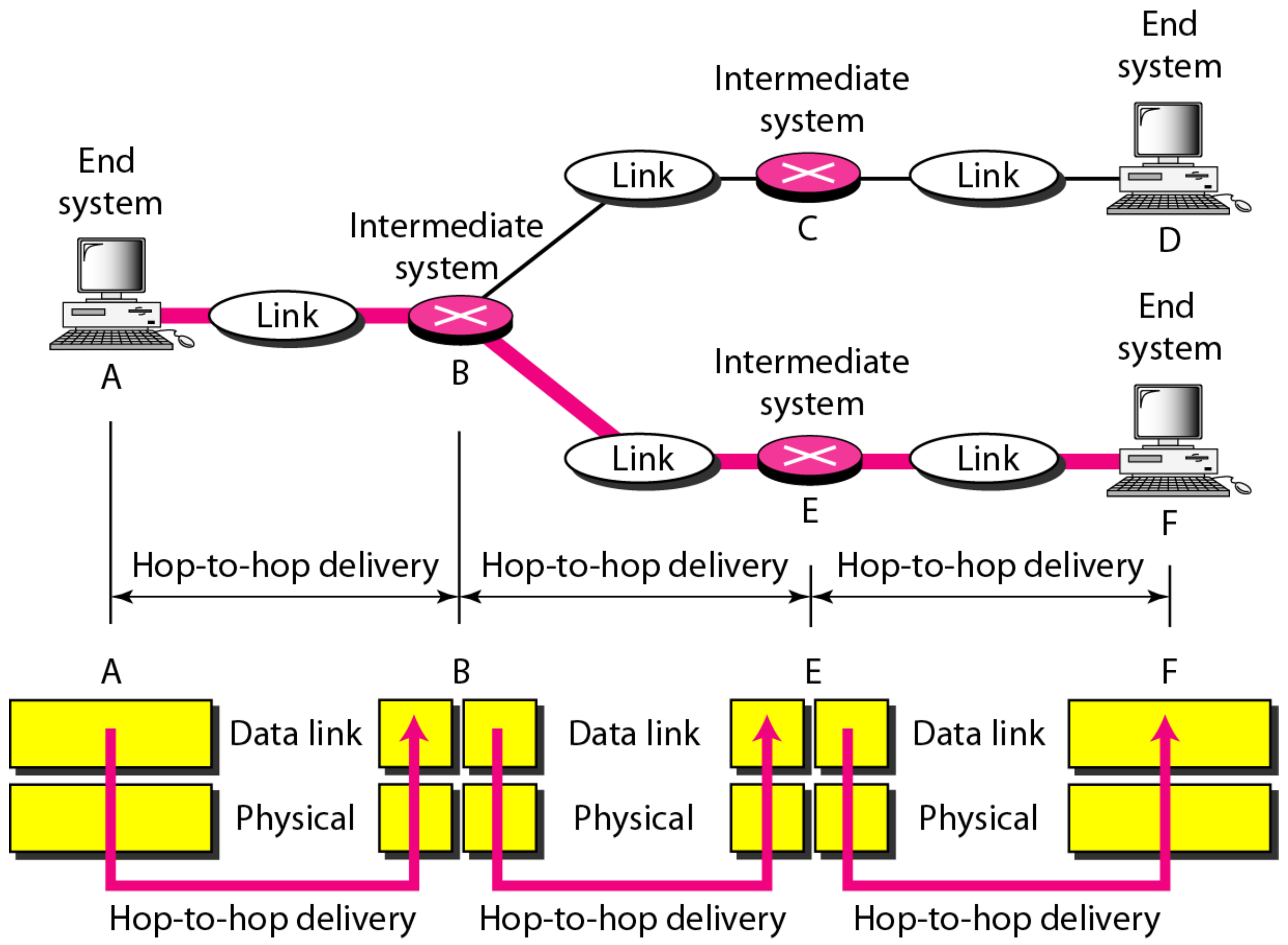
Susmini I. Lestariningati, M.T

Introduction to Data Link Layer

- The data link layer transforms the physical layer, a raw transmission facility, to a link responsible for node-to-node (hop-to-hop) communication.
- Specific responsibilities of the data link layer include
  - framing,
  - addressing,
  - flow control,
  - error control, and
  - media access control.



- The data link layer divides the stream of bits received from the network layer into manageable data units called frames.
- The data link layer adds a header to the frame to define the addresses of the sender and receiver of the frame.

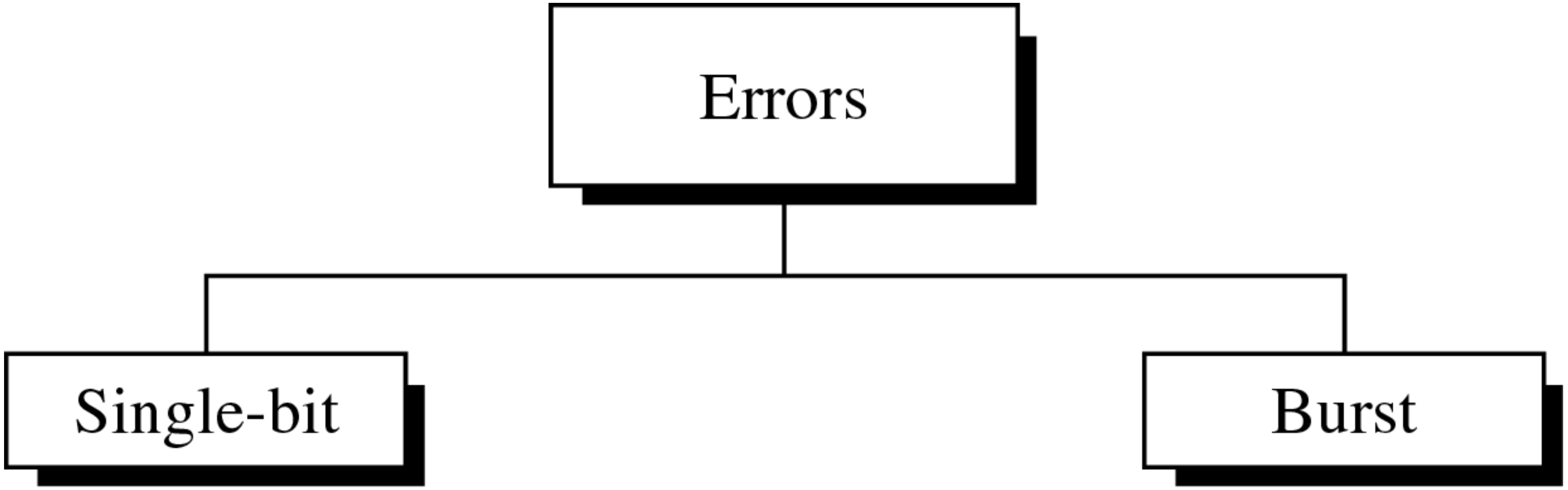


- The data link layer also adds reliability to the physical layer by adding mechanisms to detect and retransmit damaged, duplicate, or lost frames. When two or more devices are connected to the same link, data link layer protocols are necessary to determine which device has control over the link at any given time.

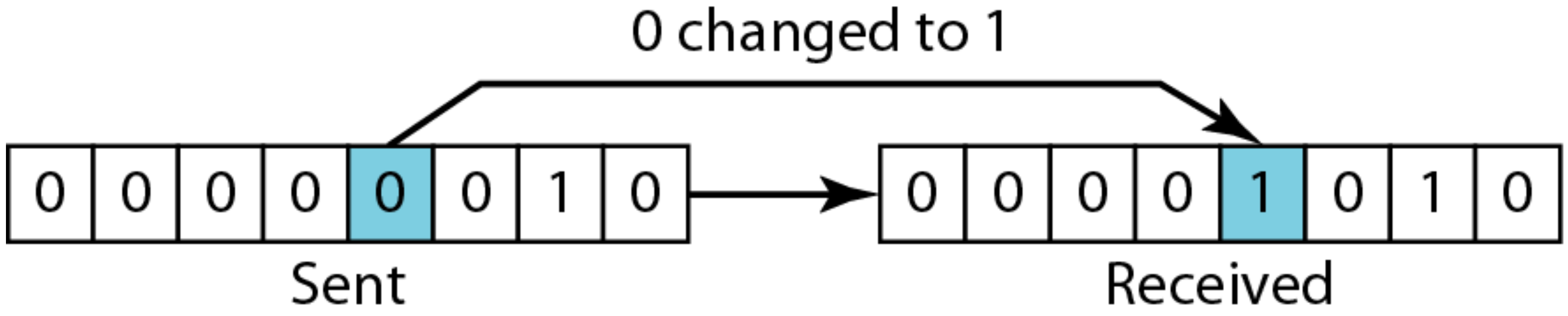
Error Control

- Data can be corrupted during transmission. For reliable communication, error must be detected and corrected are implemented either at the data link layer or the transport layer of the OSI model
- The general definitions of the terms are as follows:
  - **Error detection** is the detection of errors caused by noise or other impairments during transmission from the transmitter to the receiver.
  - **Error correction** is the detection of errors and reconstruction of the original, error-free data.

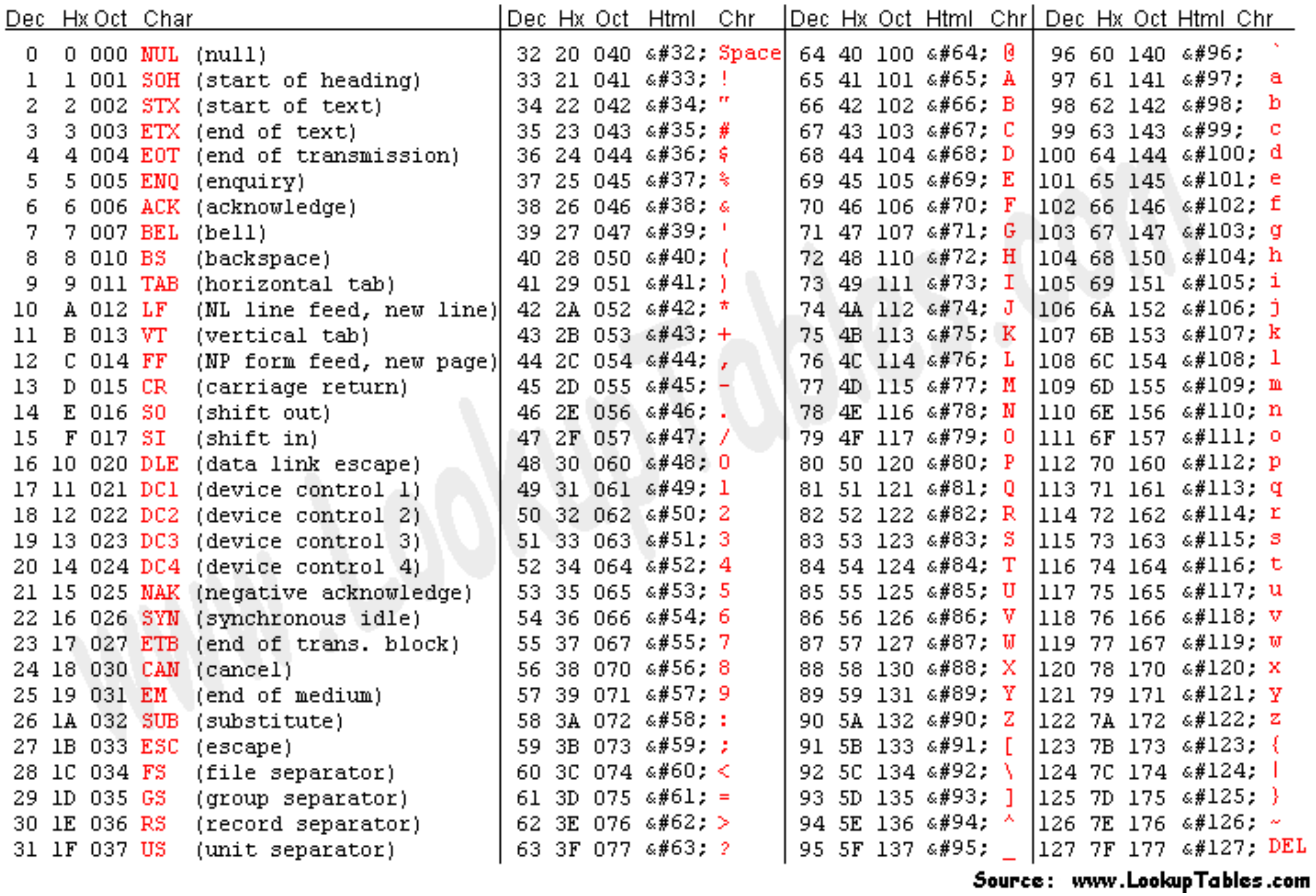
Types of Error



Single Bit Error



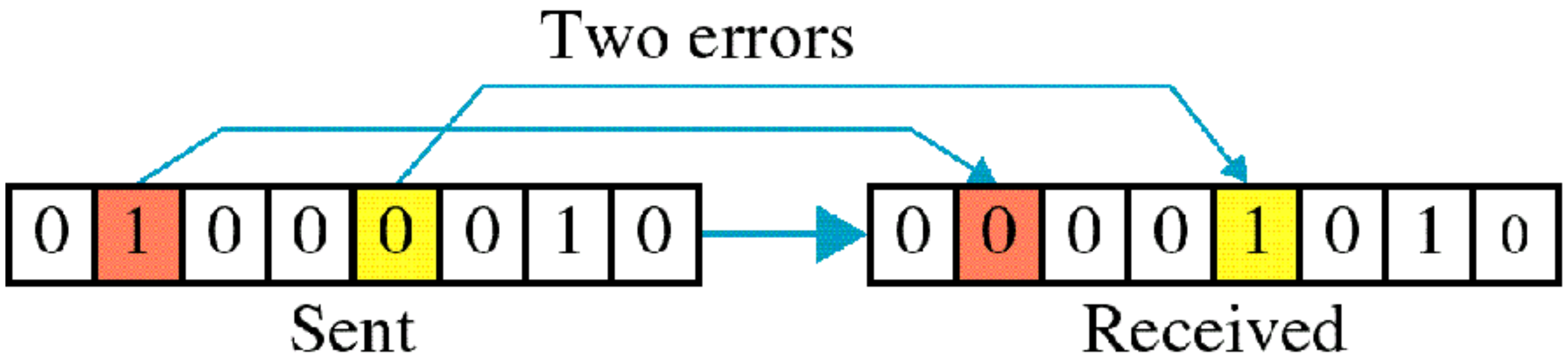
- Single bit error is when only one bit in the data unit has changed
- example : ASCII STX (Hex: 02) change 0 to 1, received ASCII LF (Hex: 0A)



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

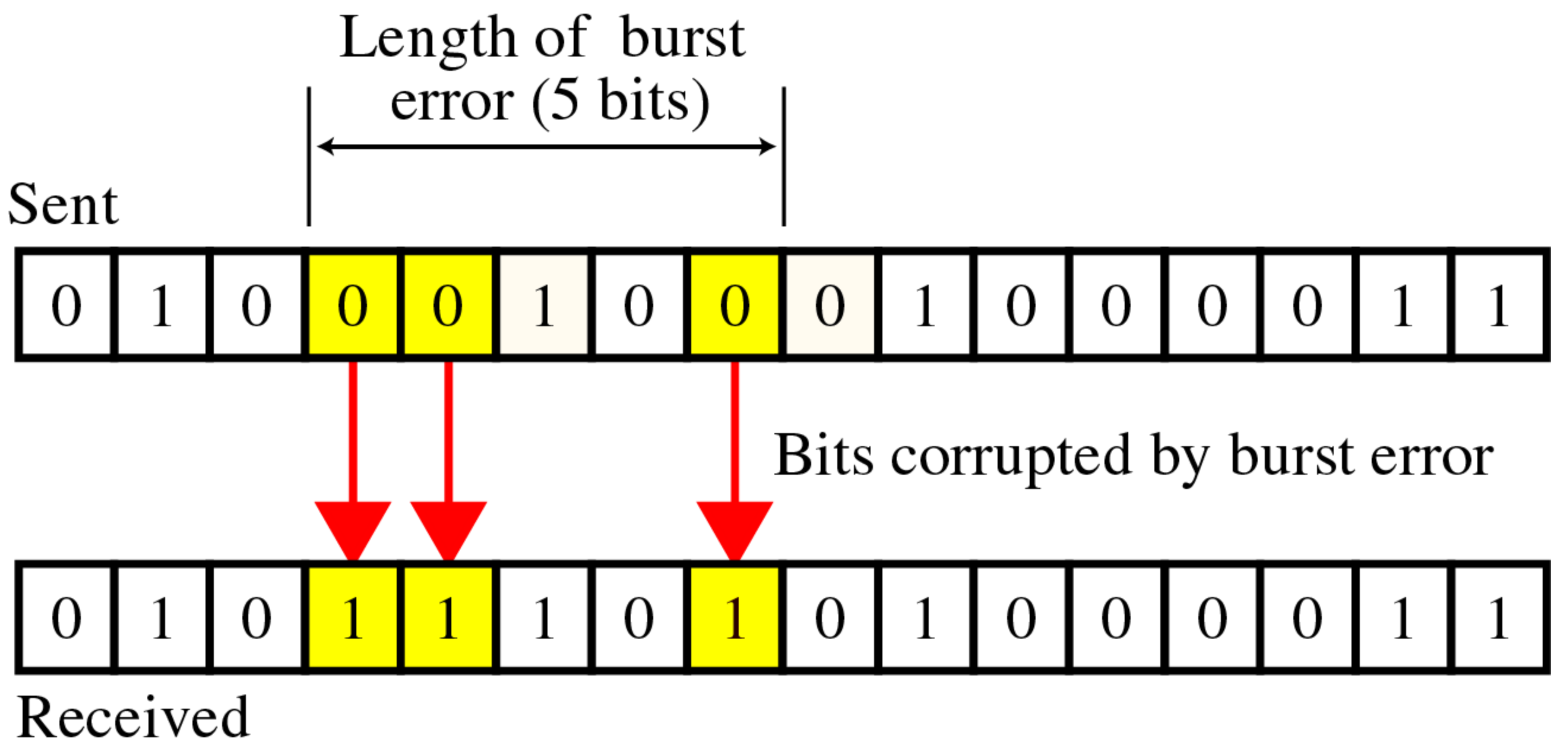
Multiple Bit Error



- Multiple-Bit Error is when two or more nonconsecutive bits in the data unit have changed
- example : ASCII B (Hex: 42) change into ASCII LF (Hex: A)

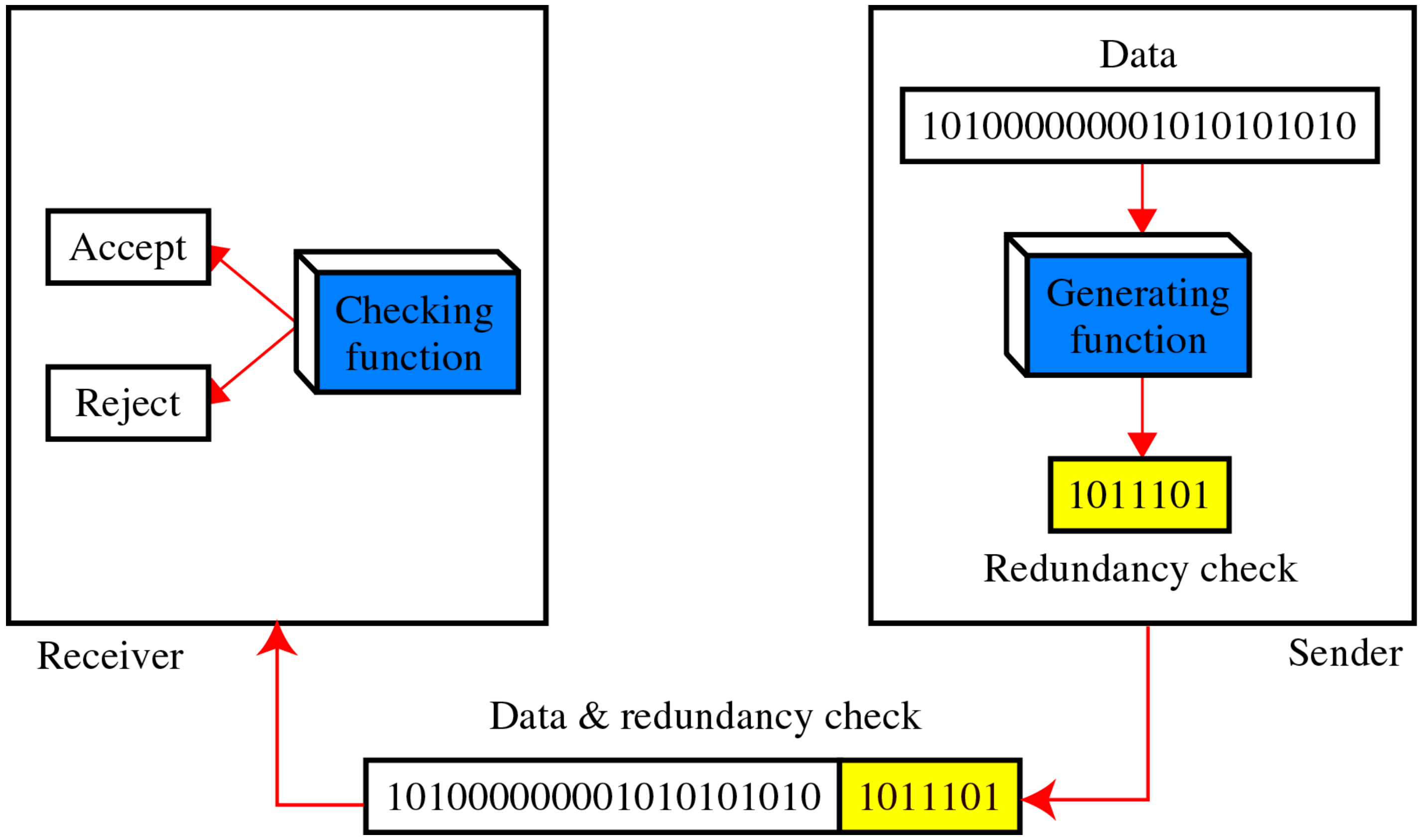
Burst Error

- Burst Error means that two or more consecutive bits in the data unit have changed

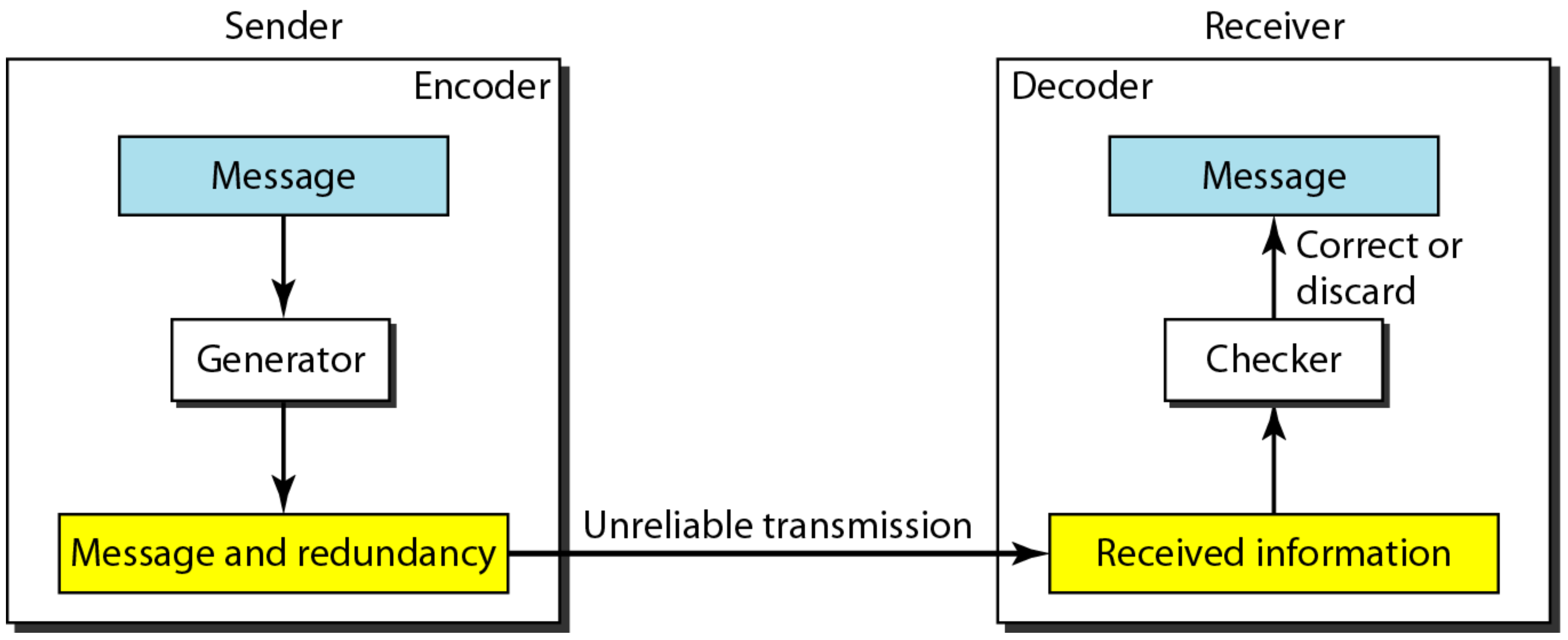


Redundancy Concepts

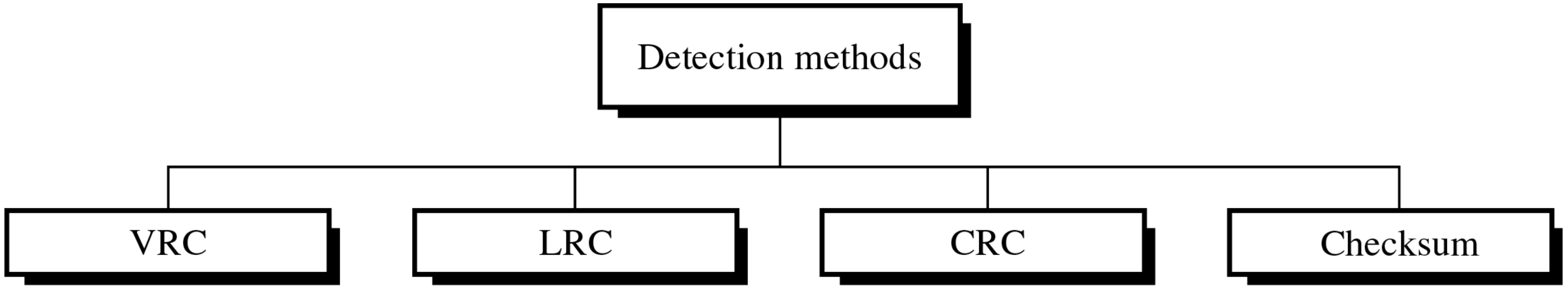
- The general idea for achieving error detection and correction is to add some redundancy (i.e., some extra data) to a message, which receivers can use to check consistency of the delivered message, and to recover data determined to be corrupted.



- Error-detection and correction schemes can be either systematic or non-systematic:
    - In a systematic scheme, the transmitter sends the original data, and attaches a fixed number of check bits (or parity data), which are derived from the data bits by some deterministic algorithm. If only error detection is required, a receiver can simply apply the same algorithm to the received data bits and compare its output with the received check bits; if the values do not match, an error has occurred at some point during the transmission.
    - In a system that uses a non-systematic code, the original message is transformed into an encoded message that has at least as many bits as the original message.


Sender
Encoder
Message
Generator
Message and redundancy
Unreliable transmission
Receiver
Decoder
Message
Correct or discard
Checker
Received information

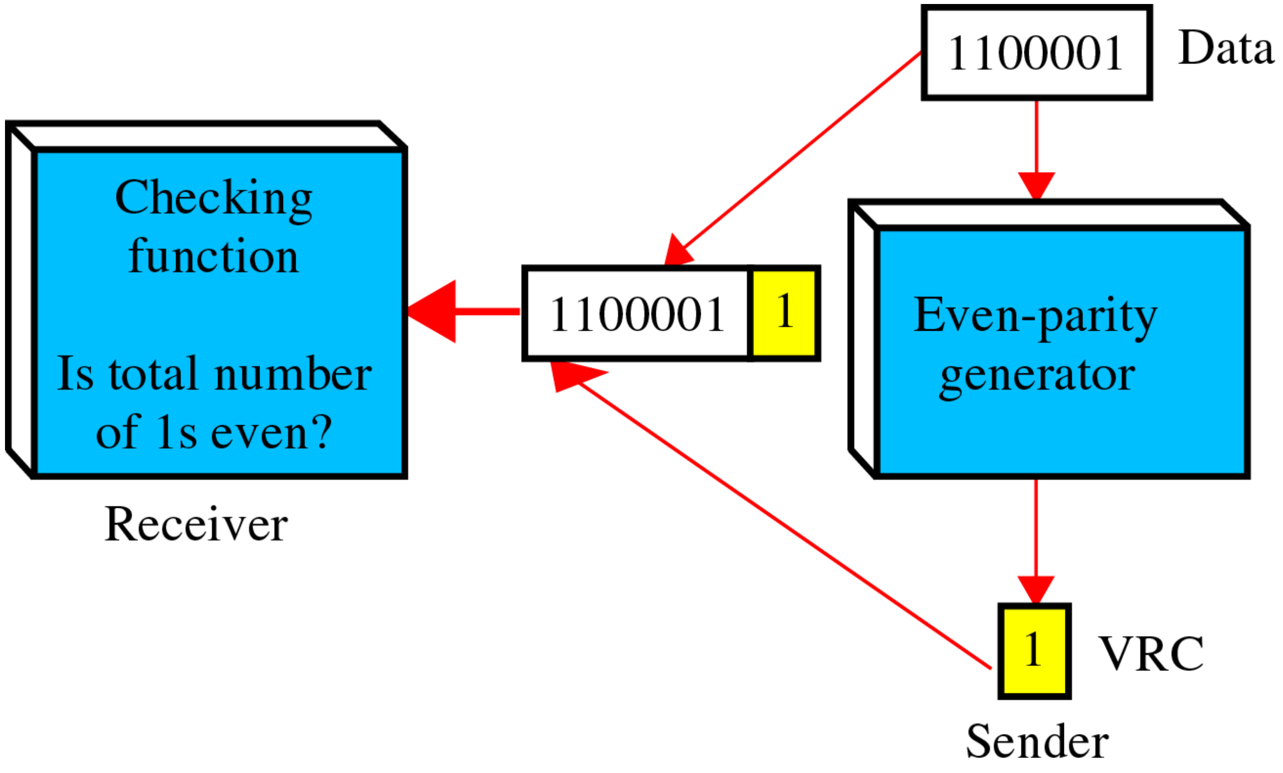
Error Detection Methods



- Vertical Redundancy Check (VRC)
- Longitudinal Redundancy Check (LRC)
- Cyclic Redundancy Check (CRC)
- Checksum

Vertical Redundancy Check (VRC)

- A parity bit is added to every data unit so that the total number of 1s (including the parity bit) becomes even for even-parity check or odd for odd-parity check
- VRC can detect all single-bit errors. It can detect multiple-bit or burst errors only the total number of errors is odd/even



Example of Even Parity

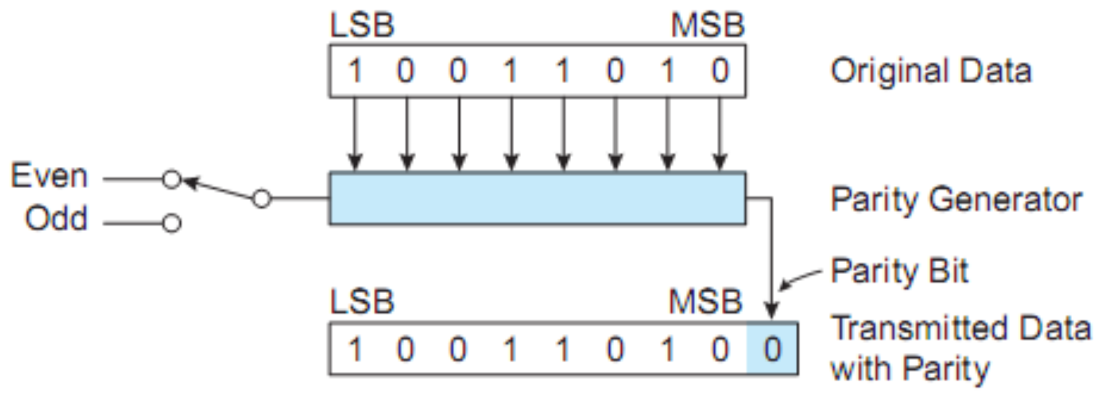
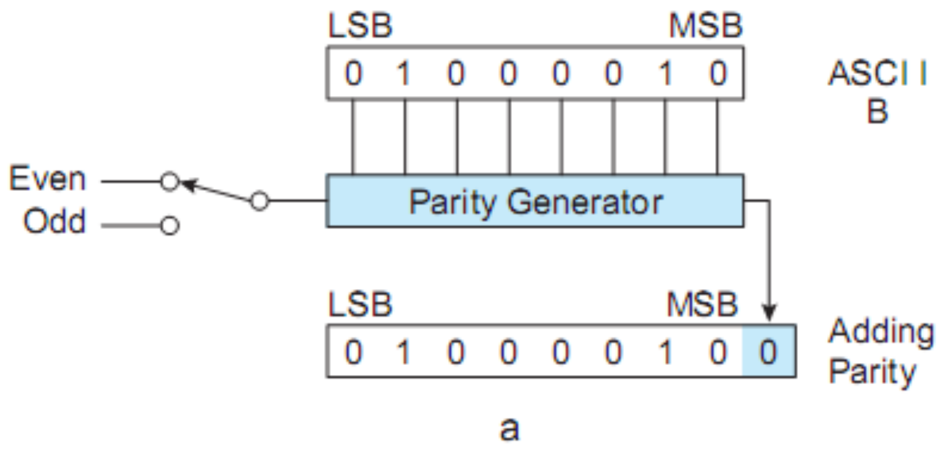
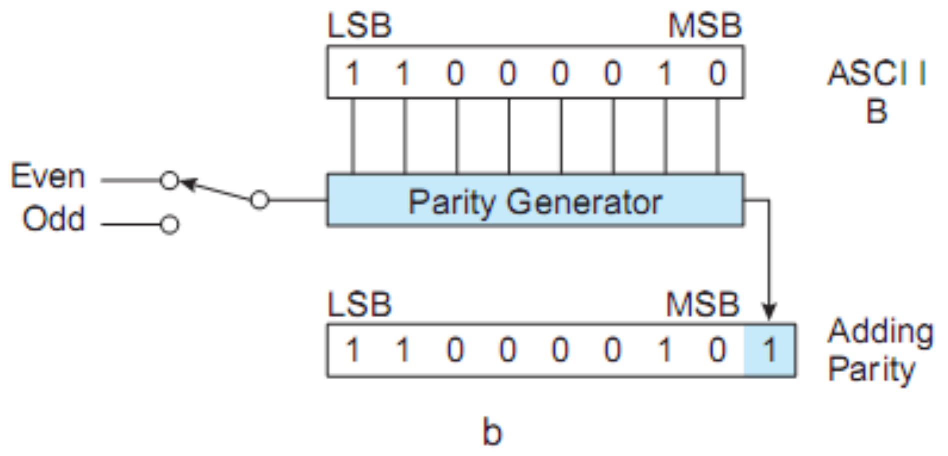


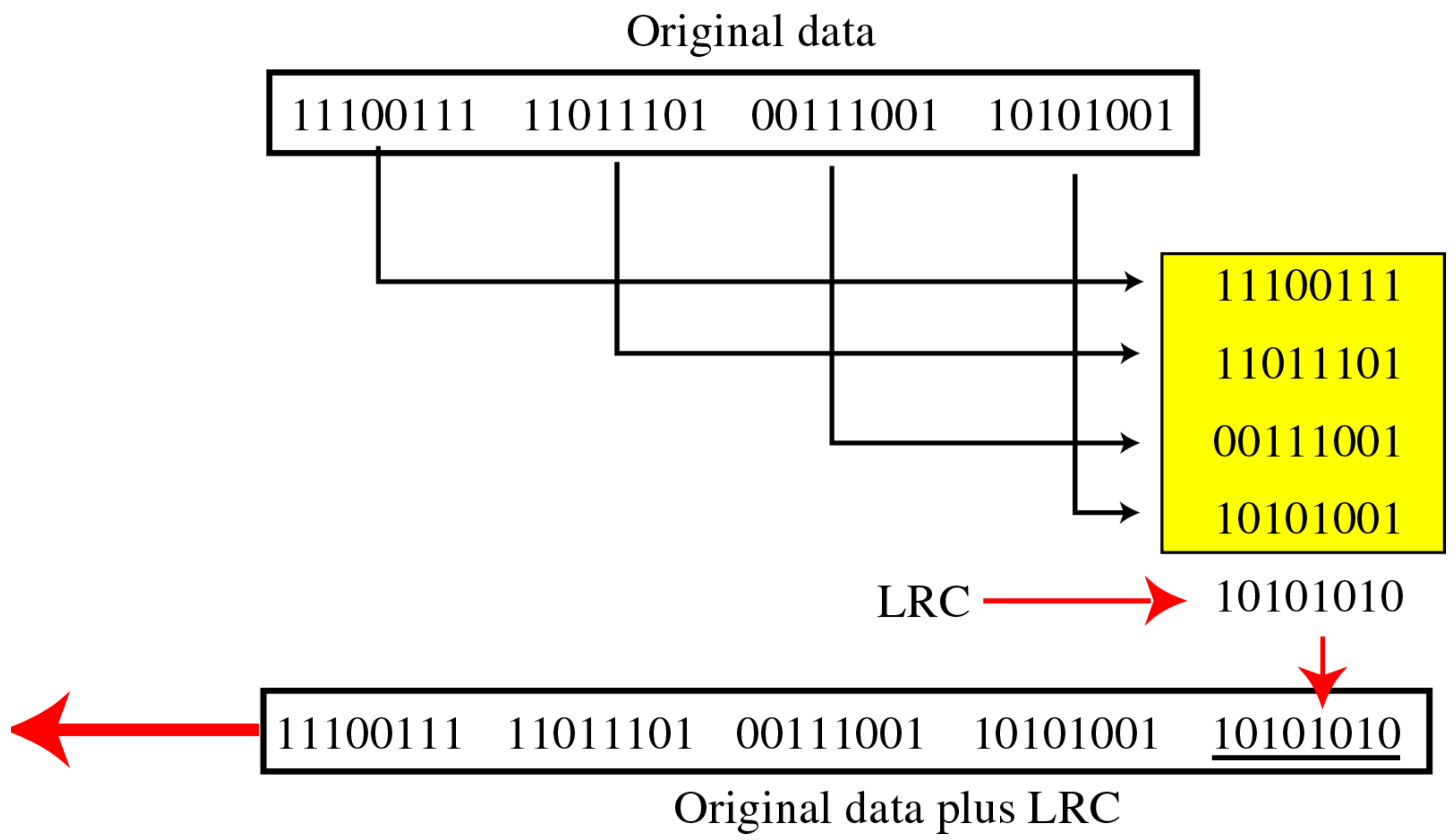
FIGURE 3-1 Appending Parity Bit





Longitudinal Redundancy Check (LRC)

- Parity bits of all the positions are assembled into a new data unit, which is added to the end of the data block



Exclusive OR

$0 \oplus 0 = 0$ $1 \oplus 1 = 0$

a. Two bits are the same, the result is 0.

$0 \oplus 1 = 1$ $1 \oplus 0 = 1$

b. Two bits are different, the result is 1.

$$\begin{array}{cccccc} & 1 & 0 & 1 & 1 & 0 \\ \oplus & 1 & 1 & 1 & 0 & 0 \\ \hline & 0 & 1 & 0 & 1 & 0 \end{array}$$

c. Result of XORing two patterns

Example

EXAMPLE 3-4

Determine the states of the LRC bits for the asynchronous ASCII message "Help!"

SOLUTION

The first step in understanding the process is to list each of the message's characters with their ASCII code and even VRC parity bit:

LSB						MSB	VRC	CHARACTER
0	0	0	1	0	0	1	0	H
1	0	1	0	0	1	1	0	e
0	0	1	1	0	1	1	0	l
0	0	0	0	1	1	1	1	p
1	0	0	0	0	1	0	0	!

Next, for each vertical column, find the LRC bit by applying the exclusive OR function. To make this process easier, you can consider the results of the exclusive OR process as being low or zero (0), if the number of ones (1) are even, and one (1) if the count is odd. For instance, in the LSB column, there are two 1's, so the LRC bit for that column is a 0. And for the rest:

LSB						**MSB**	**VRC**	**CHARACTER**
0	0	0	1	0	0	1	0	H
1	0	1	0	0	1	1	0	e
0	0	1	1	0	1	1	0	l
0	0	0	0	1	1	1	1	p
1	0	0	0	0	1	0	0	!
0	0	0	0	1	0	0	1	LRC

EXAMPLE 3-5

Show how a good message would produce an LRC of 0 at the receiver.

SOLUTION

Repeat the process as before, but include the LRC character this time. Note that the number of 1s in each column are always even if there are no errors present:

LSB						MSB	VRC	CHARACTER
0	0	0	1	0	0	1	0	H
1	0	1	0	0	1	1	0	e
0	0	1	1	0	1	1	0	l
0	0	0	0	1	1	1	1	p
1	0	0	0	0	1	0	0	!
0	0	0	0	1	0	0	1	LRC
0	0	0	0	0	0	0	0	Receiver LRC

EXAMPLE 3-6

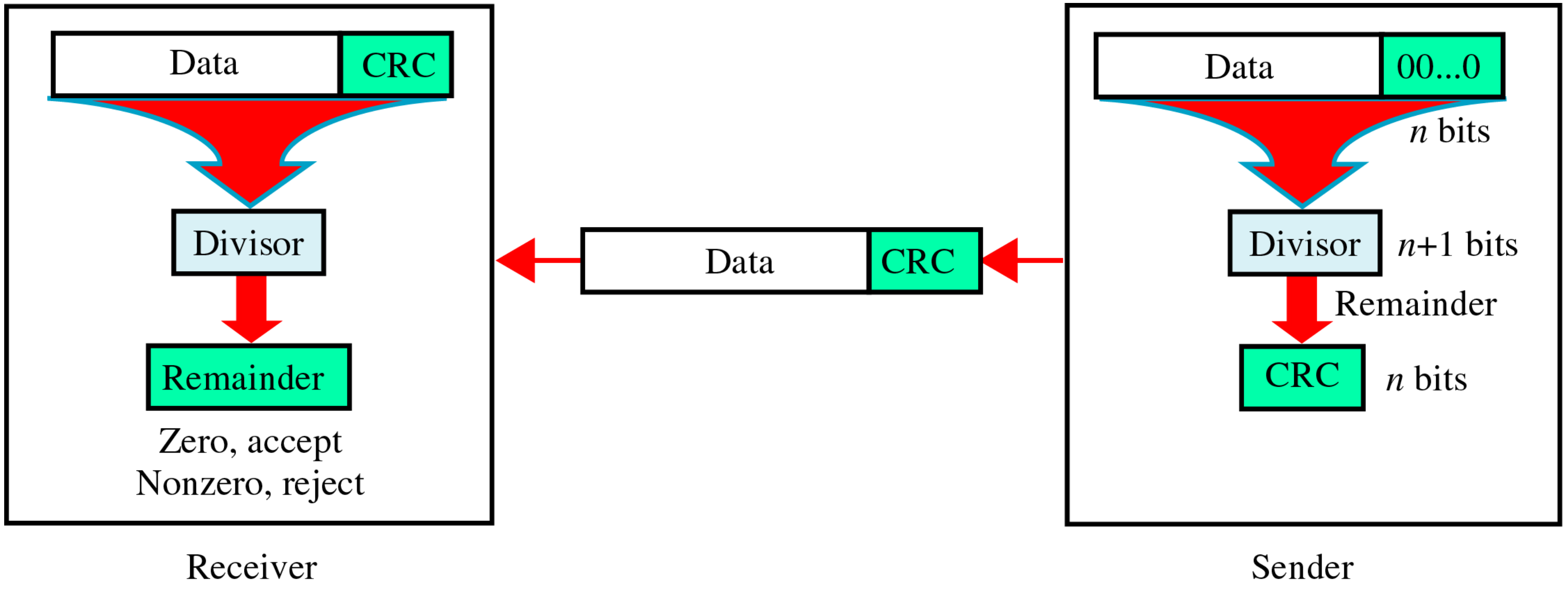
Illustrate how LRC/VRC is used to correct a bad bit.

SOLUTION

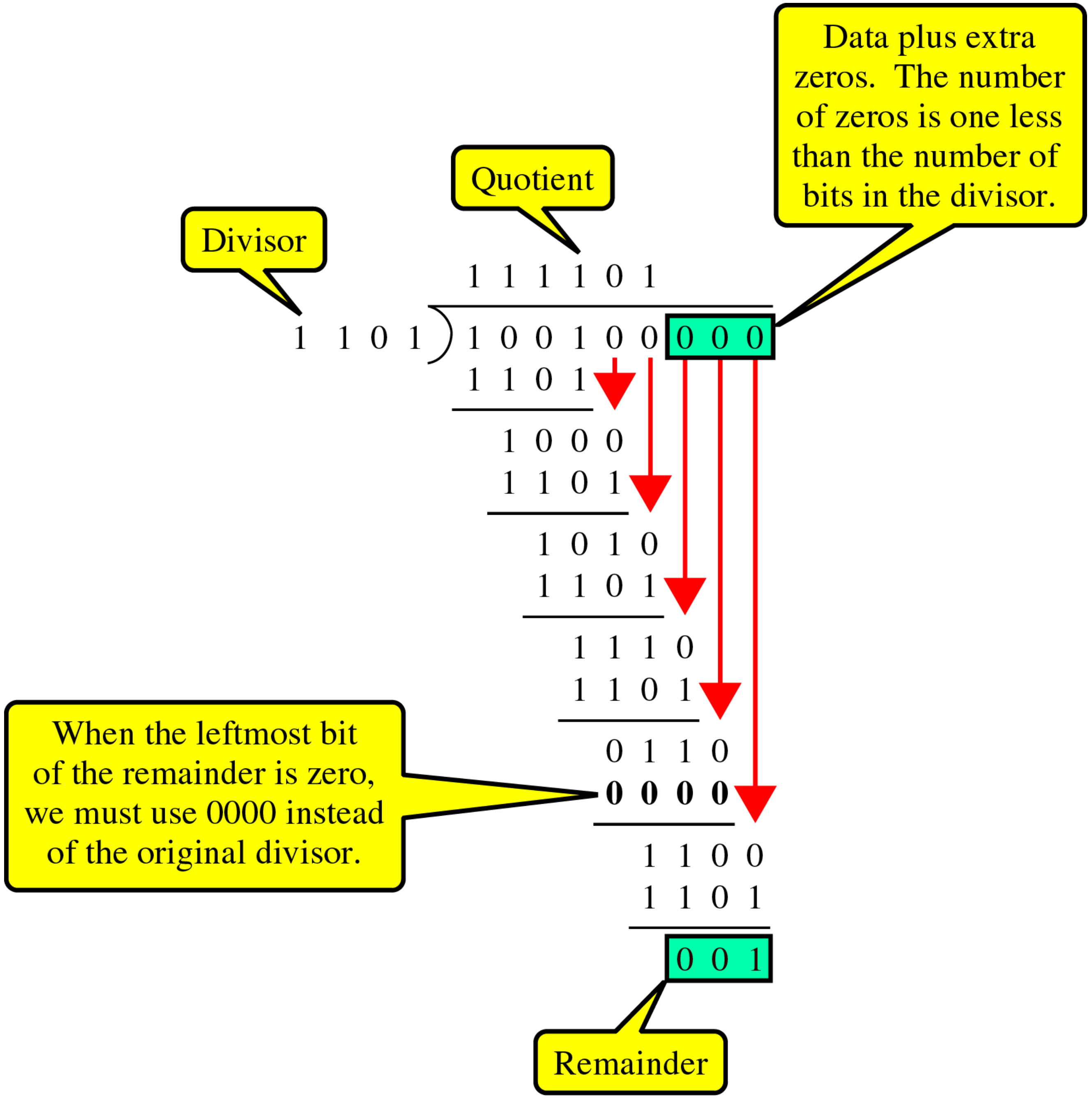
We will use the same message, but by placing an error in the received data would cause the l character to print as an *h*. You can compare the data with the good example to satisfy yourself as to which bit is bad and confirm that the LRC process does indeed pick out the same bit.

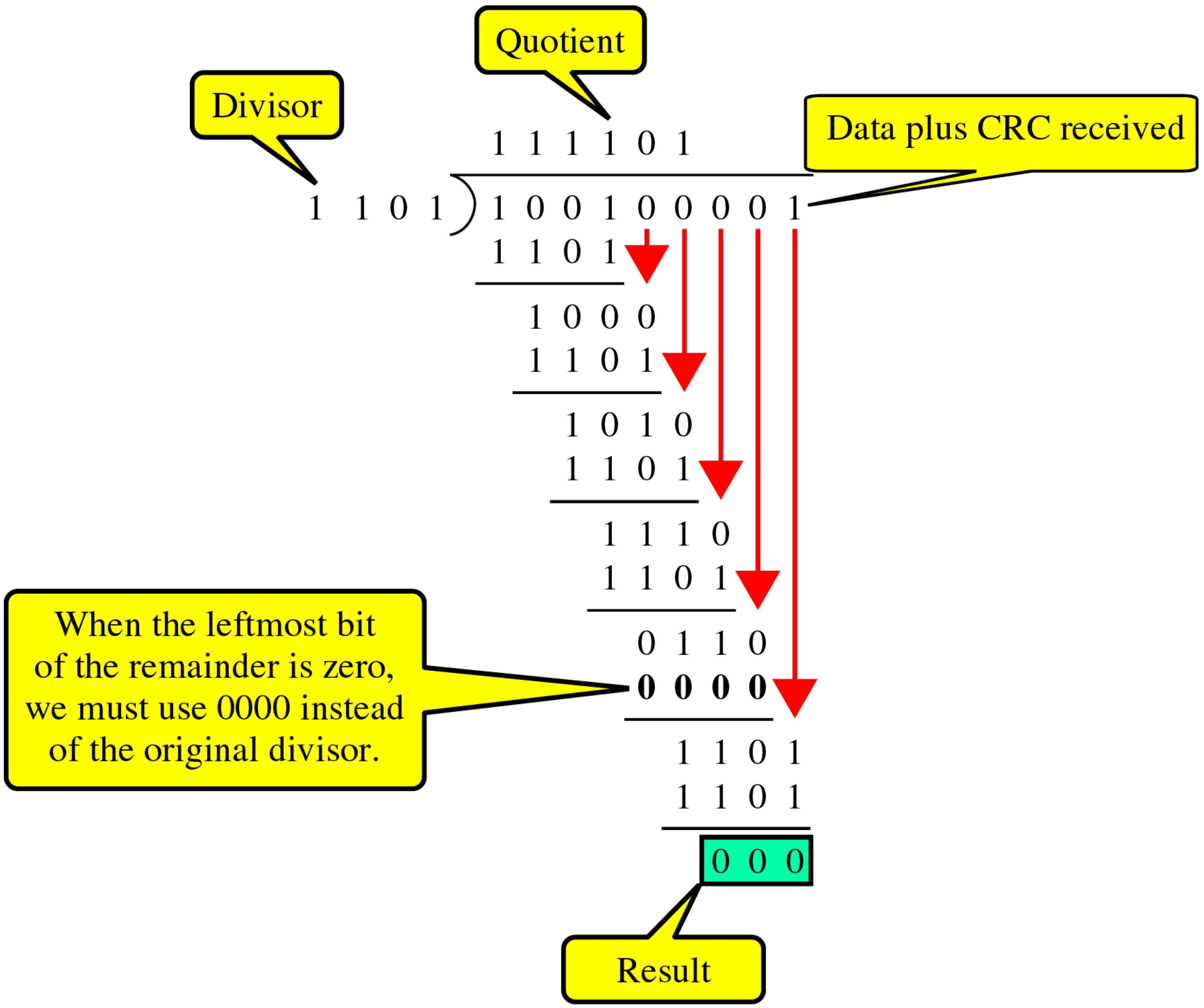
LSB						MSB	VRC	CHARACTER
0	0	0	1	0	0	1	0	H
1	0	1	0	0	1	1	0	e
0	0	0	1	0	1	1	0	h
0	0	0	0	1	1	1	1	p
1	0	0	0	0	1	0	0	!
0	0	0	0	1	0	0	1	LRC
0	0	1	0	0	0	0	1	Received LRC

Cyclic Redundancy Check (CRC)



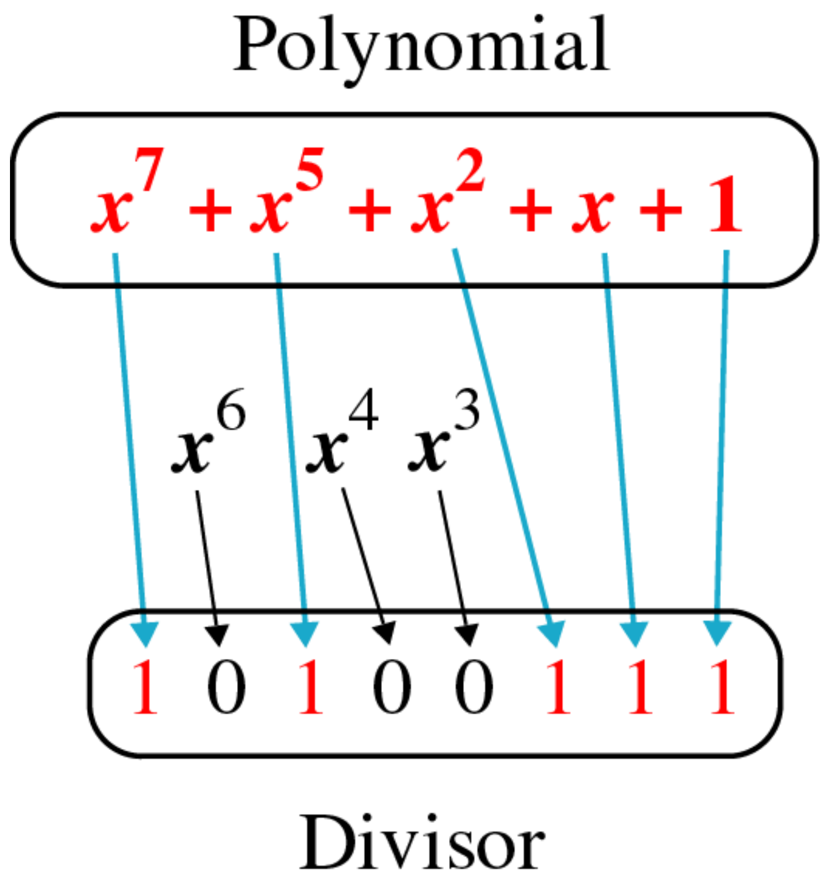
- Using Modular 2 Division
- Data : 1 0 0 1 0 0
- Divisor : 1 1 0 1


Data plus extra zeros. The number of zeros is one less than the number of bits in the divisor.
Quotient
Divisor
1 1 1 1 0 1
1 1 0 1
1 0 0 1 0 0 0 0 0
1 1 0 1
1 0 0 0
1 1 0 1
1 0 1 0
1 1 0 1
1 1 1 0
1 1 0 1
0 1 1 0
0 0 0 0
1 1 0 0
1 1 0 1
0 0 1
When the leftmost bit of the remainder is zero, we must use 0000 instead of the original divisor.
Remainder


Quotient
Divisor
Data plus CRC received
1 1 1 1 0 1
1 1 0 1
1 0 0 1 0 0 0 0 1
1 1 0 1
1 0 0 0
1 1 0 1
1 0 1 0
1 1 0 1
1 1 1 0
1 1 0 1
0 1 1 0
0 0 0 0
1 1 0 1
1 1 0 1
0 0 0
When the leftmost bit of the remainder is zero, we must use 0000 instead of the original divisor.
Result

Polynomials

- CRC generator (divisor) is most often represented not as a string of 1s and 0s, but as an algebraic polynomial.



Standard Polynomials

CRC-12

$$\boxed{x^{12} + x^{11} + x^{3} + x + 1}$$

CRC-16

$$\boxed{x^{16} + x^{15} + x^{2} + 1}$$

CRC-ITU-T

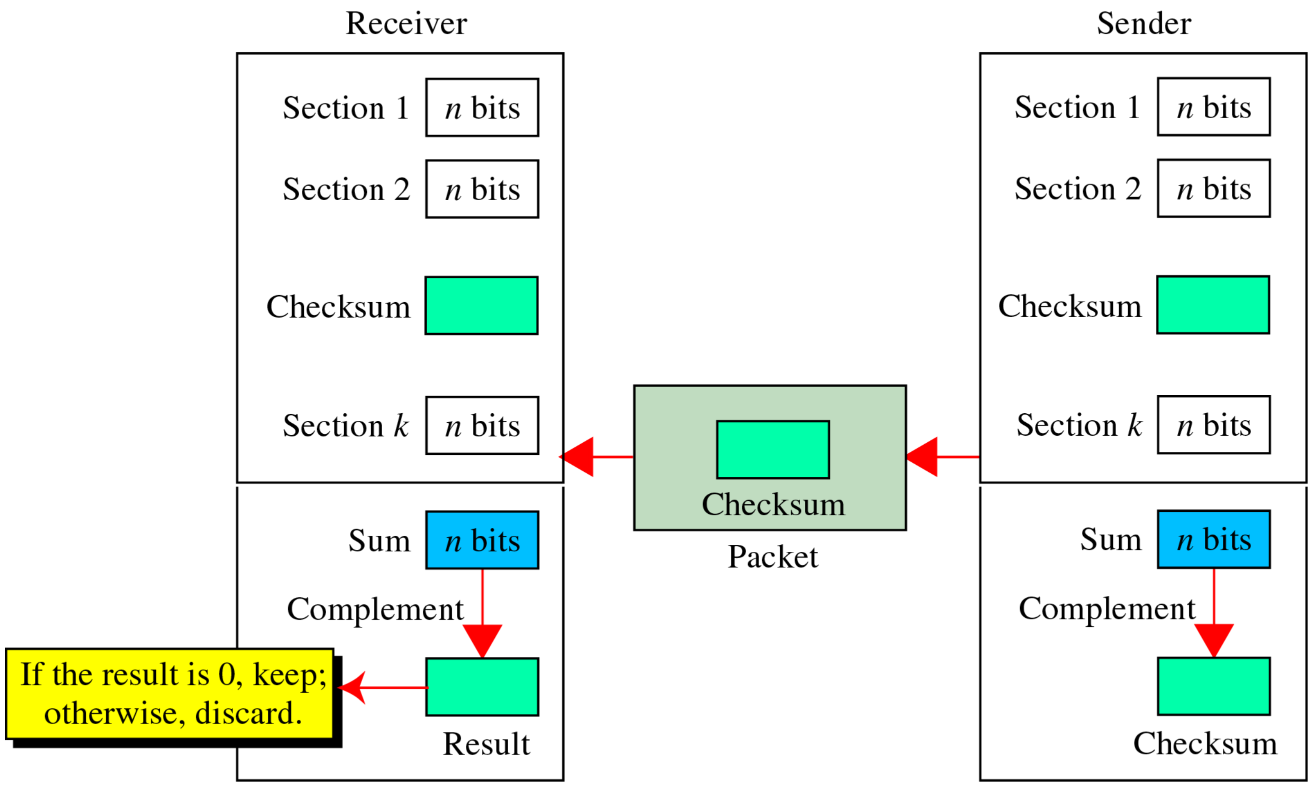
$$\boxed{x^{16} + x^{12} + x^{5} + 1}$$

CRC-32

$$\boxed{x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1}$$

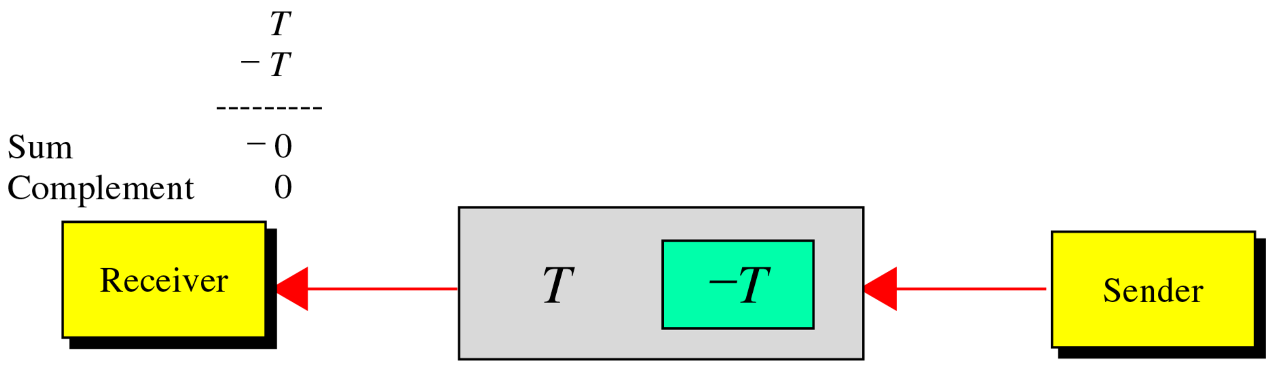
Checksum

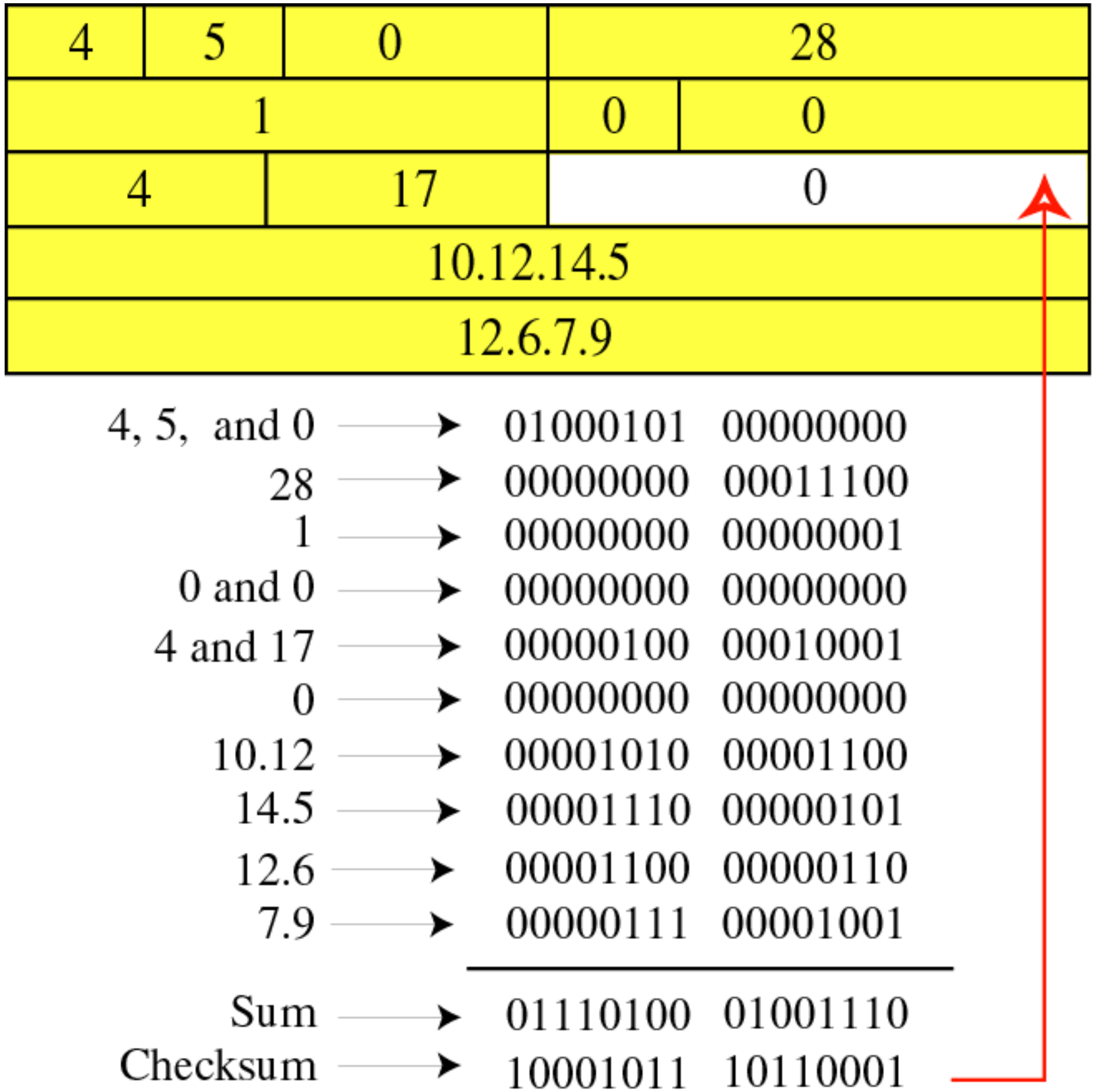
- Used by the higher layer protocols
- based on the concept of redundancy (VRC, LRC and CRC)



- To create the checksum the sender does the following:
  - The unit is divided into K sections, each of n bits.
  - Section 1 and 2 are added together using one's complement.
  - Section 3 is added to the result of the previous step.
  - Section 4 is added to the result of the previous step.
  - The process repeats until section k is added to the result of the previous step.
  - The final result is complemented to make the checksum.

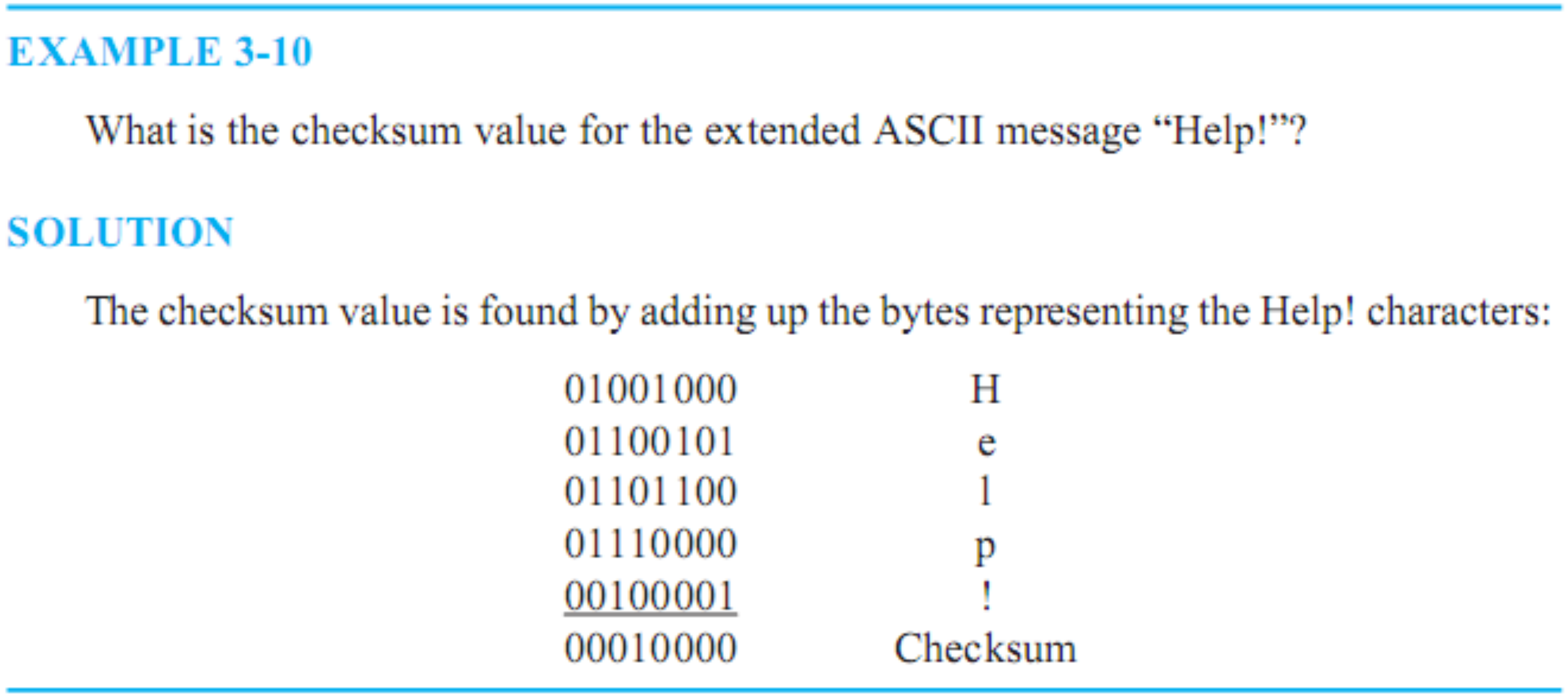
The receiver adds the data unit and the checksum field. If the result is all 1s, the data unit is accepted; otherwise it is discarded.





4, 5, and 0	→	01000101	00000000
28	→	00000000	00011100
1	→	00000000	00000001
0 and 0	→	00000000	00000000
4 and 17	→	00000100	00010001
0	→	00000000	00000000
10.12	→	00001010	00001100
14.5	→	00001110	00000101
12.6	→	00001100	00000110
7.9	→	00000111	00001001
Sum	→	01110100	01001110
Checksum	→	10001011	10110001

Example



EXAMPLE 3-10

What is the checksum value for the extended ASCII message “Help!”?

SOLUTION

The checksum value is found by adding up the bytes representing the Help! characters:

01001000	H
01100101	e
01101100	l
01110000	p
$\underline{00100001}$	!
00010000	Checksum

Error Correction

- Error can be handled in two ways:
  - when an error is discovered, the receiver can have the sender retransmit the entire data unit.
  - a receiver can use an error-correcting code, which automatically corrects certain errors.