identification of the mantissa and the exponent.

The decimal number 79.625 can also be written as follows:

$$79.625_{10} = 1001111.101_2 = 1.001111101_2 \times 2^6$$

– sign bit: $S = 0$;

– biased exponent (8 bits): $E_b = 6_{10} + 127_{10} = 133_{10} = 10000101_2$;

– fractional part of the mantissa (23 bits):

$$f = 00111110100000000000000_2$$

from which:

$$79.625_{10} = 0\ 10000101\ 00111110100000000000000_{IEE754}$$

The decimal number −1000.2 is represented in binary in the form:

$$1000.2_{10} = 1111101000.0011001100110011_2$$

The fractional part corresponds to a continually repeating binary sequence. The closest number to 1000.2 that may be represented is:

$$\begin{aligned} 1000.20001220703125_{10} &= 1111101000.00110011001101_2 \\ &= 1.11110100000110011001101_2 \times 2^9 \end{aligned}$$

– sign bit: $S = 1$;

– biased exponent (8 bits): $E_b = 9_{10} + 127_{10} = 136_{10} = 10001000_2$;

– fractional part of the mantissa (23 bits):

$$f = 11110100000110011001101_2$$

and finally:

$$-1000.2_{10} = 1\ 10001000\ 11110100000110011001101_{IEE754}$$

In the above-cited single-precision IEEE-754 representations, the first bit indicates the sign, the next eight bits allow for the coding of the exponent and the last 23 bits correspond to the fractional part of the mantissa.

The different values taken by the numbers in IEEE-754 representations are recorded in Table 1.5. The IEEE-754 standard uses special symbols (NaN, infinity) to indicate numbers that have an exponent composed entirely of bits set to 0 or 1. The NaN or *not a number* value is used to represent a value that does not correspond to a real number.

Table 1.5. *Values of numbers in IEEE-754 representations*

	Exponent	Fraction	Value
Normalized	$E_{min} \leq E \leq E_{max}$	$f \geq 0$	$\pm(1.f) \times 2^E$
Denormalized	$E = E_{min} - 1$	$f > 0$	$\pm(0.f) \times 2^{E_{min}}$
Zero	$E = E_{min} - 1$	$f = 0$	± 0
Infinite	$E = E_{max} + 1$	$f = 0$	$\pm\infty$
Not a Number	$E = E_{max} + 1$	$f > 0$	NaN

EXAMPLE 1.21.– Find the decimal number corresponding to the following single-precision IEEE-754 representation:

$$1\ 10000111\ 11000000000000000100001_{IEE754}$$

We have:

– sign bit: $S = 1$;

– biased exponent (8 bits): $E_b = 10000111_2 = 135_{10}$;

– fractional part of the mantissa (23 bits):

$$f = 11000000000000000100001_2$$

Applying the formula to the expression of real numbers by starting from the IEEE-754 representation, that is:

$$N_{10} = (-1)^S(1.f) \times 2^{(E_b - 127)}$$

we find:

$$\begin{aligned} N_{10} &= (-1)^1(1.11000000000000000100001_2) \times 2^{(135-127)} \\ &= (-1)(111000000.000000000100001_2) \\ &= (-1)(2^8 + 2^7 + 2^6 + 2^{-10} + 2^{-15}) \\ &= -448.00100708_{10} \end{aligned}$$

from which:

$$1\ 10000111\ 11000000000000000000001_{IEE754} = -448.001_{10}$$

1.11.2.2. *Arithmetic operations on floating-point numbers*

Let $x = M_x \cdot B^{E_x}$ and $y = M_y \cdot B^{E_y}$ be two positive numbers (sign-bit $S = 0$).

Supposing that $E_x \geq E_y$, $y = M_Y \cdot B^{E_x}$ and $M_Y = M_y B^{(E_x - E_y)}$, we have:

$$x + y = (M_x + M_Y) \cdot B^{E_x} \qquad [1.22]$$

and

$$x - y = (M_x - M_Y) \cdot B^{E_x} \qquad [1.23]$$

In a floating-point representation, the numbers to be added or subtracted must, thus, have the same exponent, such as:

$$\begin{aligned} 145.500_{10} &= 10010001.100_2 = 0.10010001100 \times 2^8 \\ 27.625_{10} &= 00011000.101_2 = 0.00011011101 \times 2^8 \end{aligned}$$

In the case of multiplication and division, the results are obtained as follows:

$$x \times y = (M_x \times M_y) \cdot B^{(E_x + E_y)} \qquad [1.24]$$

and

$$x / y = (M_x / M_y) \cdot B^{(E_x - E_y)} \qquad [1.25]$$

It must be noted that because of the overflow effect or round-off errors, the arithmetic operations in a floating-point representation do not have exactly the same properties (associativity, distributivity) as with real numbers.

1.12. Data representation

As the arithmetic unit of a digital system recognizes only the binary states 0 and 1, a code is necessary to manipulate and transfer alphanumeric data (numbers, letters, special characters) between a digital system and its peripheral devices.

1.12.1. *Gray code*

Gray code (or reflected binary code) is a non-weighted code, as it does not ascribe a specific weight to each bit position. It is not used for arithmetic calculations.

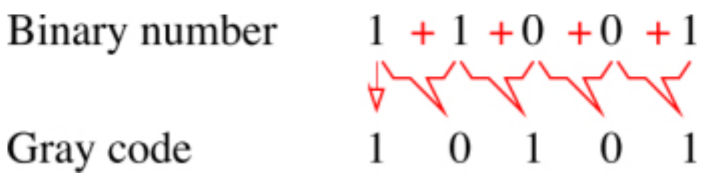
An interesting feature presented by Gray code representation is related to the fact that only a single bit changes value during the transition from one code to the next. Table 1.6 gives the binary and Gray code representation of decimal numbers from 0 to 15.

The conversion of a binary number to Gray code is carried out by making use of the following observations:

– the most significant Gray code bit, situated to the extreme left, is the same as the corresponding MSB for the binary number;

– starting from the left, add, without taking into account the carry-out bit, each pair of adjacent bits to obtain the next bit in Gray code.

EXAMPLE 1.22.– Convert the binary number 11001_2 to Gray code.



For the binary number 11001_2, the corresponding Gray code is 10101.

To convert Gray code to a binary number:

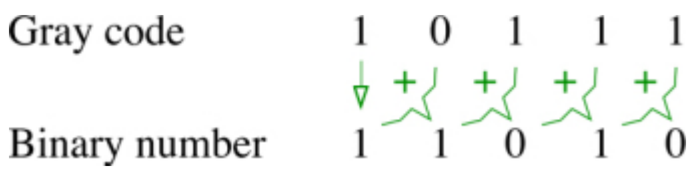
– the MSB of the binary number, located at the extreme left, is identical to the corresponding Gray code bit;

– starting from the left, add each new bit of the binary code to the next bit of the Gray code, without taking into account any carry-out bit, to obtain the next bit of the binary code.

Table 1.6. *Binary and Gray code representation of decimal numbers from* 0 *to* 15

Decimal number	**Binary number**	**Gray code**
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	110
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

EXAMPLE 1.23.– Convert the Gray code 10111 to a binary number.



The binary number corresponding to Gray code 10111 is 11010_2.

Gray code is used in Karnaugh maps and in the design of logic circuits. They also find application in rotary encoders, where the predisposition to errors increases with the number of bits that change logical states between two consecutive positions.

1.12.2. *p-out-of-n code*

A *p-out-of-n code* is an *n*-bit representation that allows only combinations made up of *p* bits at 1 and (*n* – *p*) bits at 0. The number of valid combinations for a *p*-out-of-*n* code is $n!/[(n-p)!p!]$.

The *p*-out-of-*n* code allows for the detection of errors based on the verification of the number of 1s and 0s at the time of reading of each code combination.

Some barcodes use *p*-out-of-*n* encoding, such as 2-out-of-5 encoding. Table 1.7 offers some examples of 2-out-of-5 code. These two codes are weighted only for numbers different from zero and the list of weights appears in each of the denominations.

The 2-out-of-5 code allows for the detection of all errors relating to a single bit, but does not allow for the correction of these errors. As the smallest Hamming distance (or the minimum number of bits that change logic states between two consecutive combinations) is 2, it does not allow for the detection of errors caused by the modification of 2 bits.

Table 1.7. *Examples of 2-out-of-5 code*

	2-out-of-5 code					2-out-of-5 code				
	0	1	2	3	6	7	4	2	1	0
0	0	1	1	0	0	1	1	0	0	0
1	1	1	0	0	0	0	0	0	1	1
2	1	0	1	0	0	0	0	1	0	1
3	1	0	0	1	0	0	0	1	1	0
4	0	1	0	1	0	0	1	0	0	1
5	0	0	1	1	0	0	1	0	1	0
6	1	0	0	0	1	0	1	1	0	0
7	0	1	0	0	1	1	0	0	0	1
8	0	0	1	0	1	1	0	0	1	0
9	0	0	0	1	1	1	0	1	0	0

Barcodes used to sort out letters are represented as shown in Figure 1.8(a), by a series of parallel lines of variable size. The 0 bit corresponds to a small line and the 1 bit to a large line. Figure 1.8 (b) shows another barcode that is used to identify parts and that is composed of parallel lines of variable thickness. The 0 bit is represented by a fine line and the 1 bit by a thick line.

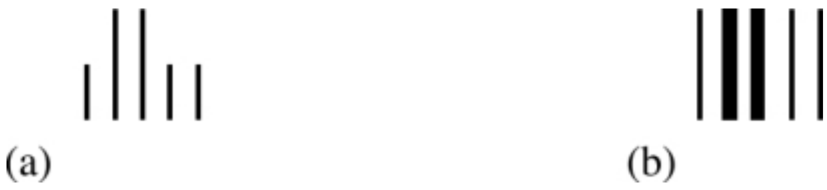


Figure 1.8. *Barcodes corresponding to the binary representation* 01100

A more compact form of the barcode is obtained by using interleaved 2-out-of-5 encoding. The first code is represented by the black lines (three fine lines and two thick lines) of variable thickness, and the second code by the spacing between the black lines (three narrow spaces and two wide spaces). The code shown in Figure 1.9(a) is a representation of the combination 01100 (black lines) followed by 11000 (spaces between the back lines). In general, the odd combinations are represented by black lines and the even combinations are represented by spaces between the black lines. Figure 1.9(b) shows the barcode corresponding to the sequence 01100, 11000, 10001 and 00110.

An appropriate optical reader is necessary to read each kind of barcode.

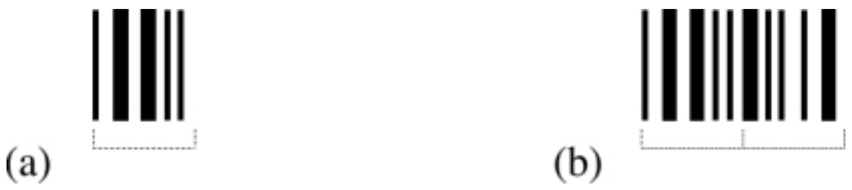


Figure 1.9. *Barcodes based on an interleaved* 2*-out-of-*5 *encoding*

1.12.3. *ASCII code*

ASCII code (or *American standard code for information interchange*) has seven bits allowing for the representation of $2^7 = 128$ symbols.

Table 1.8 gives the correspondence between certain characters and the decimal and hexadecimal numbers of the ASCII code. The letter *N*, for example, is represented in ASCII code by the number 78 in decimal and by 4*E* in hexadecimal. The ASCII code contains 34 characters used to define the format of information and the space between data and to control the transmission and reception of symbols.

1.12.4. *Other codes*

Given the ever-increasing number of characters, other systems of data representation were developed based on the ASCII code:

– EBCDIC (or *extended binary coded decimal interchange code*) is an eight bit code;

– ANSI (or *American national standard institute*) allows for the representation of alphabetical letters from many languages;

– using eight bit words (for UTF-8), 16 bit words (for UTF-16) and 32 bit words (for UTF-32), the universal code, named Unicode (or *Universal code*) represents each character in a unique way by a number. It covers symbols used in most languages.

1.13. Codes to protect against errors

There are different types of codes used to detect and correct errors that come up in digital information during transmission or during storage.

1.13.1. *Parity bit*

To facilitate the detection of errors, a supplementary bit or parity bit is often added at the end of a binary word with a fixed number of bits. It allows for the allocation of an odd or even parity depending on whether the total number of 1 bits in the code is odd or even.

Table 1.8. *ASCII codes table*

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	NUL	32	20	SP	64	40	@	96	60	‘
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	’	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	TAB	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	NP	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SO	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

NUL	Null	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End of transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

EXAMPLE 1.24.– For the word 0101101, the parity bit is 0 (even parity: 4 bits at 1).

For the word 1010001, the parity bit is 1 (odd parity: 3 bits at 1).

Using a single parity bit allows for the detection of all errors that affect only one bit. However, it does not allow for the correction of these errors.

1.13.2. *Error correcting codes*

The reliability of data transmission is generally ensured by using more elaborate codes.

1.13.2.1. *Block codes*

In the block code approach, a certain number of control bits are appended to the message that is structured in blocks of fixed size. In this way, horizontal and vertical parity of data can be verified.

Hamming distance corresponds to the number of bits that vary between two successive words.

EXAMPLE 1.25.– There is a Hamming distance of 3 between the words 111011 and 101010. At least three errors are necessary to make these two words identical.

A possible way of increasing the Hamming distance of a code consists of using several control bits. In this case, a message comprises *m* bits of data and *k* control bits.

EXAMPLE 1.26.– Represent OUI in ASCII code with odd (horizontal and vertical) parity bits and a crossed parity bit allowing for the indication of the integrity of the (horizontal and vertical) parity bits.

The ASCII codes for the characters of the word OUI are as below:

$$\begin{aligned} 79_{10} &= 4F_{16} = 1001111_2 \ \text{for} \ \ \text{O} \\ 85_{10} &= 55_{16} = 1010101_2 \ \text{for} \ \ \text{U} \\ 73_{10} &= 49_{16} = 1001001_2 \ \text{for} \ \ \text{I} \end{aligned}$$

The choice of a two-dimension representation (or a block of bits), as shown in Figure 1.10, allows for the definition of parity bits following the horizontal and vertical direction.

Changing one single bit of the data may bring about a modification of the vertical parity bit, the horizontal parity bit and the crossed parity bit, that is four bits in total. The Hamming distance is, thus, equal to 4.

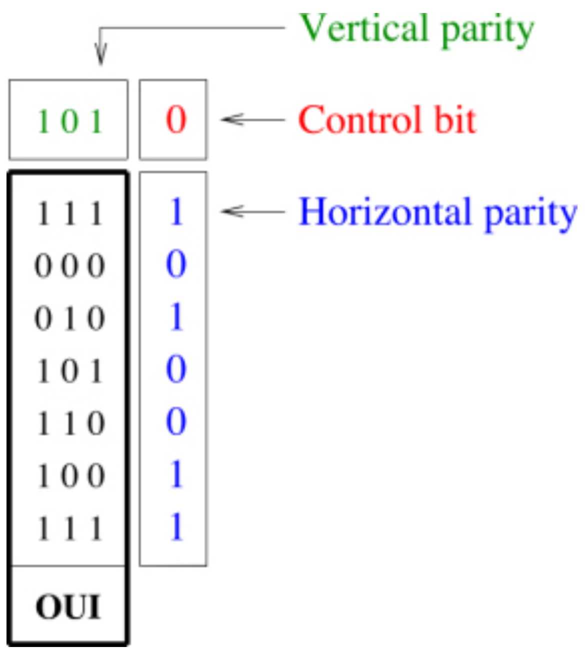


Figure 1.10. *Example of block codes*

Such a block code allows for the detection and correction of all errors affecting one single bit. It allows for the detection of all errors affecting 2 and 3 bits, but it presents the inconvenience of requiring the verification of a large number of bits.

1.13.2.2. *Cyclic codes*

Cyclic codes are based on the transcription of binary numbers in polynomial form and the division of polynomials.

EXAMPLE 1.27.– The binary code $b_{n-1}b_{n-2} \ldots b_1b_0$ corresponds to the polynomial:

$$b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1x^1 + b_0x^0$$

Let $I(x)$ be