Part I:
Basic Elements

Chapter 3:
Representation of information

2024-2025

1. Binary coding

Binary Coded Decimal (BCD)

- BCD is a type of binary code used to represent a given decimal number in an equivalent binary form.
- The BCD equivalent of a decimal number is written by replacing **each decimal digit in the integer and fractional parts with its four-bit binary equivalent.**
- The BCD code described above is more precisely known as the 8421 BCD code.
- As an example, the BCD equivalent of $(53.16)_{10}$ is written as $(0101\ 0011.0001\ 0110)_{BCD}$

BCD code

A given BCD number can be converted into an equivalent binary number by writing its decimal equivalent and then converting it into its binary equivalent.

Example: find the binary equivalent of the BCD number 0010 1001.0111 0101

BCD number: 0010 1001.0111 0101.

- Corresponding decimal number: 29.75.
- The binary equivalent of 29.75 is 11101 for the integer part and .11 for the fractional/decimal part.
- Therefore, $(0010\ 1001.0111\ 0101)_{BCD} = (11101.11)_2$.

BCD code

The table illustrates the difference between straight binary and BCD. BCD represents each decimal digit with a 4-bit code.

Notice that the codes 1010 through 1111 are not used in BCD.

Decimal	Binary	BCD
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010	00010000
11	1011	00010001
12	1100	00010010
13	1101	00010011
14	1110	00010100
15	1111	00010101

Binary Coded Decimal (BCD)

- The BCD code is the 8,4,2,1 code.
- 8, 4, 2, and 1 are weights
- This code is the simplest, most intuitive binary code for decimal digits and uses the same powers of 2 as a binary number, but only encodes the first ten values from 0 to 9.
- Example: 12 (1100 in pure binary, 0001 0010 in BCD)

Do NOT mix the conversion of a decimal number to a binary number with the coding of a decimal number with a BINARY CODE.

$13_{10} = 1101_2$ (This is conversion)

13 $\Leftrightarrow$ 0001|0011 (This is coding)

Gray code

Gray code is an unweighted code that has a single bit change between one code word and the next in a sequence.

Gray code is used to avoid problems in systems where an error can occur if more than one bit changes at a time.

Decimal	Binary	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	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Binary–Gray Code Conversion

1. Begin with the most significant bit (MSB) of the binary number. **The MSB of the Gray code equivalent is the same as the MSB of the given binary number.**

2. The second most significant bit, adjacent to the MSB, in the Gray code number is **obtained by adding the MSB and the second MSB of the binary number and ignoring the carry, if any**. That is, if the MSB and the bit adjacent to it are both '1', then the corresponding Gray code bit would be a '0'.

3. The third most significant bit, adjacent to the second MSB, in the Gray code number is obtained by adding the second MSB and the third MSB in the binary number and ignoring the carry, if any.

4. The process continues until we obtain the LSB of the Gray code number by the addition of the LSB and the next higher adjacent bit of the binary number.

Binary–Gray Code Conversion

The conversion process is further illustrated with the help of an example showing step-by-step conversion of $(1011)_2$ into its Gray code equivalent:

Binary 1011
Gray code 1- - -
Binary 1011
Gray code 11- -
Binary 1011
Gray code 111-
Binary 1011
Gray code 1110

Gray Code- Binary Conversion

A given Gray code number can be converted into its binary equivalent by going through the following steps:

1. Begin with the most significant bit (MSB). The MSB of the binary number is the same as the MSB of the Gray code number.

2. The bit next to the MSB (the second MSB) in the binary number is obtained by adding the MSB in the binary number to the second MSB in the Gray code number and disregarding the carry, if any.

3. The third MSB in the binary number is obtained by adding the second MSB in the binary number to the third MSB in the Gray code number. Again, carry, if any, is to be ignored.

4. The process continues until we obtain the LSB of the binary number.

Gray Code- Binary Conversion

The conversion process is further illustrated with the help of an example showing step-by-step conversion of the Gray code number 1110 into its binary equivalent:

Gray code 1110
Binary 1- - -
Gray code 1110
Binary 10 - -
Gray code 1110
Binary 101
Gray code 1110
Binary 1011

Excess-3 Code

- The excess-3 code is another important BCD code.
- The excess-3 code for a given decimal number is determined by adding ‘3’ to each decimal digit in the given number and then replacing each digit of the newly found decimal number by its four-bit binary equivalent.

Example: find the excess-3 code for the decimal number 597

Solution:

The addition of ‘3’ to each digit yields the three new digits/numbers ‘8’, ‘12’ and ‘10’.

• The corresponding four-bit binary equivalents are 1000, 1100 and 1010 respectively.

• The excess-3 code for 597 is therefore given by:

1000 1100 1010=100011001010.

Excess-3 Code table

Decimal number	Excess-3 code	Decimal number	Excess-3 code
0	0011	5	1000
1	0100	6	1001
2	0101	7	1010
3	0110	8	1011
4	0111	9	1100

2. Characters encoding

- The characters:
  - Alphabetical (A-Z , a-z),
  - Digital (0,1,2,3,4,5,6,7,8,9),
  - Punctuation(;. ? ! ….)
  - Specials (&, $, %,,…)

Character coding is done using a table of correspondence between characters and binary numbers.

Character encoding: ASCII code

- ASCII (American Standard Code for Information Interchange) is a computer standard for character encoding that emerged in the 1960s.
- The basic ASCII code represented 7-bit characters (128 possible characters, from 0 to 127).
  - Codes from 48 to 57: numbers in order (0,1,...,9)
  - Codes from 65 to 90: capital letters (A....Z)
  - Codes from 97 to 122: lowercase letters (a....z).

- This code was developed for the English language, so it does not contain accented characters or language-specific characters.

The ASCII code table (1)

Ctrl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01		SOH
^B	2	02		STX
^C	3	03		ETX
^D	4	04		EOT
^E	5	05		ENQ
^F	6	06		ACK
^G	7	07		BEL
^H	8	08		BS
^I	9	09		HT
^J	10	0A		LF
^K	11	0B		VT
^L	12	0C		FF
^M	13	0D		CR
^N	14	0E		SO
^O	15	0F		SI
^P	16	10		DLE
^Q	17	11		DC1
^R	18	12		DC2
^S	19	13		DC3
^T	20	14		DC4
^U	21	15		NAK
^V	22	16		SYN
^W	23	17		ETB
^X	24	18		CAN
^Y	25	19		EM
^Z	26	1A		SUB
^[	27	1B		ESC
^\	28	1C		FS
^]	29	1D		GS
^^	30	1E	▲	RS
^-	31	1F	▼	US

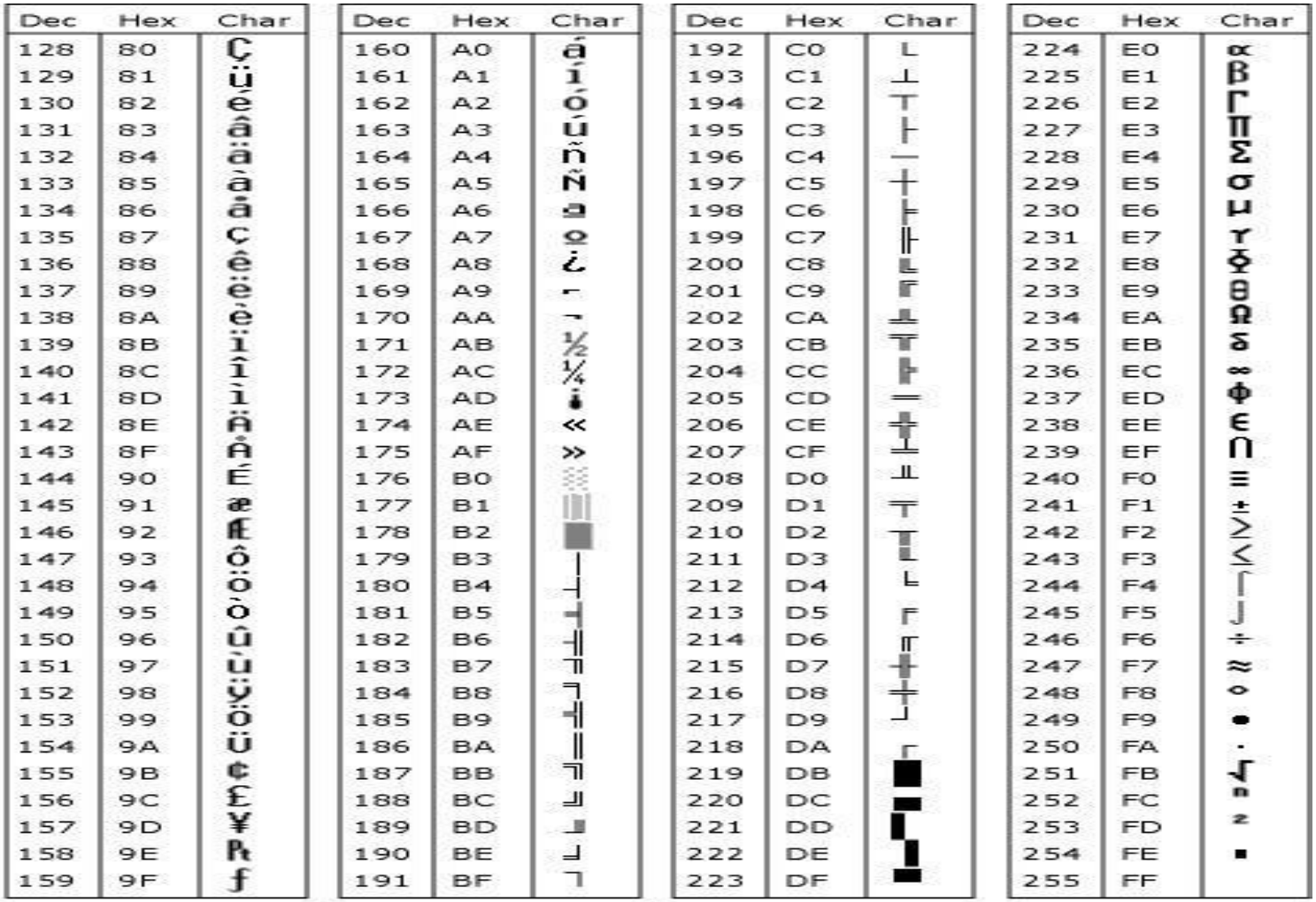
Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂

Extended ASCII code

- The ASCII code has been extended to 8 bits to be able to encode more characters (0 to 255) => extended ASCII code.
- Allows us to code accented characters: à, é, è,...etc.



Dec	Hex	Char
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	ö
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	₧
159	9F	ƒ

Dec	Hex	Char
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	░
177	B1	▒
178	B2	▓
179	B3	│
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐

Dec	Hex	Char
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	─
197	C5	┼
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	═
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	█
220	DC	▄
221	DD	▌
222	DE	▐
223	DF	▀

Dec	Hex	Char
224	E0	α
225	E1	ß
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	µ
231	E7	τ
232	E8	Φ
233	E9	Θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	φ
238	EE	ε
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
245	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	∙
250	FA	·
251	FB	√
252	FC	ⁿ
253	FD	²
254	FE	■
255	FF	

The ASCII code table (2)

USASCII code chart

b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits ↓ / Column → / Row ↓	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

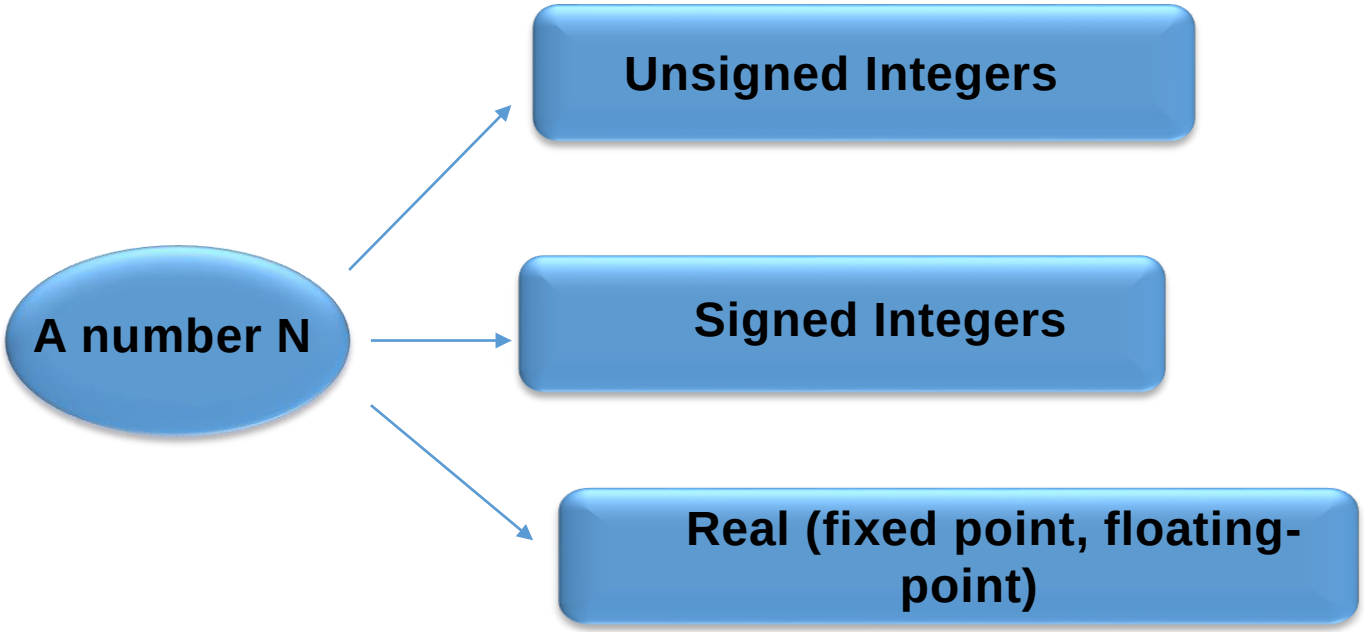
Examples of ASCII encoding

		Binary		Hexadecimal		Decimal
H	=	01001000	=	48	=	72
e	=	01100101	=	65	=	101
l	=	01101100	=	6C	=	108
l	=	01101100	=	6C	=	108
o	=	01101111	=	6F	=	111
,	=	00101100	=	2C	=	44
Espace	=	00100000	=	20	=	32
w	=	01110111	=	77	=	119
o	=	01100111	=	67	=	103
r	=	01110010	=	72	=	114
l	=	01101100	=	6C	=	108
d	=	01100100	=	64	=	100

Unicode code

- Developed in 1991
- It uses 16 bits to represent 65,536 characters (0 to 65,535)
- Unicode defines tens of thousands of codes, but the first 128 remain compatible with ASCII.
- It codes most alphabets: Arabic, Chinese, Turkish, etc.
- We refer to a character by its number written in hexadecimal preceded by “U+”.
- For example, the Latin letter “a” corresponds to U+0061 (in hexadecimal)
- http://www.unicode.org

3. Number representation (internal data representation)



Coding of natural (unsigned) integers -pure binary code-

- A natural number is a positive or zero integer.
- To encode natural numbers, we use pure binary code.
- The natural number is represented in base 2 on n bits.
- The range of numbers on n bits is: [0, 2^n-1]
- With n bits, we can represent 2^n numbers.

Example:

- *On a byte, $(17)_{10}$ is coded in pure binary: 00010001*

- The number of bits to use depends on the range of numbers we want to use
- Using 1 byte (8 bits): we can code 2^8 values: [0 ; 255]
- Using 2 bytes (16 bits): we can encode 2^{16} values: [0 ; 2^{16-1}]
- Using (n bits): we can code 2^n values: [0 ; 2^{n-1}]

Coding of signed integers

- These are numbers with a + or - sign.

  Example: -24, +354,.....

- There are at least three techniques allowing the representation of signed integers:

1. Signed magnitude representation
2. One's complement representation (C1).
3. Two's complement representation (C2)

Signed magnitude representation

The most significant bit is used to represent the sign of the number

- 1: for a negative number
- 0: for a positive number

- The other (n-1) bits encode the magnitude (the absolute value) of the number

- With n bits, we encode all the numbers between $-(2^{n-1}-1)$ and $(2^{n-1}-1)$

- **Example :**
- On 8 bits, we can encode the numbers -13 and +17 in signed magnitude as follows:
  -13 is coded by: 1 0 0 0 1 1 0 1
  +17 is coded by: 0 0 0 1 0 0 0 1

Advantages and disadvantages of signed magnitude

- **Advantages**: Simple
- **Disadvantages (limites):**
  - Two representations of zero :

    On 8 bits : +0 = 00000000

    -0 = 10000000
  - Multiplication and addition are less obvious

For example, we add -3 and -1 on 4 bits

One's complement (C1)

- The first bit is reserved for the sign.
- **If the number is positive then the number keeps its format.**
- If the number is negative then each bit (of the remaining bits) is inverted (0 becomes 1 and 1 becomes 0) (by completing on the left with 0s to obtain an n-bit code).
- The number of possible combinations on n bits is 2^n
- With n bits, we encode all the numbers between $-(2^{n-1}-1)$ and $(2^{n-1}-1)$
- Two combinations for 0
- **Examples:**
  - -5 on 8 bits

    $5 = (00000101)_2$

    $-5 = (11111010)_{C1}$
  - +7 on 8 bits

    $+7 = (00000111)_2 = (00000111)_{C1}$

Addition and subtraction in C1

It is based on the following principle:

- If no carry is generated by the sign bit then the result is correct and it is represented in C1.
- Otherwise, it will be removed and added to the result of the operation, this is represented in C1.

- **Example 1:**

-14+5 on 5 bits

$-14+5=(-1110+0101)_2=(11110)_{SM}+(00101)_{SM}=(10001)_{C1}+(00101)_{C1}$

$=(10110)_{C1}$ no retain, so the result is correct and it is written in C1.

We must transform it into a binary number then decimal:

$(10110)_{C1}=(11001)_{SM}=(-1001)_2=-9$

- **Exemple 2:**

14-6 on 5 bits

$14-6=(1110-0110)_2=(01110+10110)_{SVA}=(01110+11001)_{C1}=(00111)_{C1}$

with a carry (retained) 1, the latter is added to the result obtained; and we obtain $(01000)_{C1}= 8$ since the number is positive

Two's complement (C2)

The representation of a number X in a two's complement on n bits is done as follows:

- if (X>= 0) (number from 0 to (2^{n-1}-1)) then X is coded in the same way as in pure binary,
- if (X<0) (number from -(2^{n-1}-1) to 0) then:
  - Code |X| in binary by completing on the left with 0 to obtain an n-bit code
  - Invert all bits of the binary representation (one's complement);
  - Add 1 to the result (two's complement or C2)
  - The number of possible combinations on n bits is 2^n
  - With n bits, we encode all the numbers between -(2^{n-1}) and (2^{n-1}-1)

Examples:

- -5 on 8 bits

  -5= $(00000101)_2$ = $(11111010)_{C1}$ = $(11111011)_{C2}$

- +7 on 8 bits

  +7= $(00000111)_2$= $(00000111)_{C1}$ = $(00000111)_{C2}$

Representation of some numbers on 4 bits

C2	MS	Decimal
0000	0000	0
0001	0001	+1
0010	0010	+2
0011	0011	+3
0100	0100	+4
0101	0101	+5
0110	0110	+6
0111	0111	+7
1000		-8
1001	1111	-7
1010	1110	-6
1011	1101	-5
1100	1100	-4
1101	1011	-3
1110	1010	-2
1111	1001	-1

Addition and subtraction in C2

It is based on the following principle:

- If there is a carry generated by the sign bit, it is ignored and the result is in C2
- Otherwise the result is correct and in C2

Example 1:

-14+5 on 5 bits

$-14+5=(-1110+0101)_2=(11110)_{SM}+(00101)_{SM}=(10001)_{C1}+(00101)_{C1}$

$=(10010)_{C2}+(00101)_{C2}=(10111)_{C2}$ no carry, so the result is correct and it is written in C2

We must transform it into a binary number then decimal

$(10111)_{C2}=(10110)_{C1}=(11001)_{SM}=(-1001)_2=-9$

Example 2:

14-6 on 5 bits

$14-6=(1110-0110)_2=(01110+10110)_{SM}=(01110+11001)_{C1}=(01110+11010)_{C2}=(01000)_{C2}$ with a carry 1, the latter is ignored and we obtain

$(01000)_{C2}=(00110)_{C1}=(00110)_{SM}=8$ since the number is positive

Representation of real numbers

- **Example**: +15,23, -234,01……
- Two questions arise:

1. How to represent the comma in a machine?

   The designers did not take the comma (or the point) into account, but they offered a place in the representation of numbers.

2. How to tell the machine the position of the decimal point?
   - Fixed point
   - Floating point

Fixed point

- A real number = the integer part + the decimal part.
- The integer part is coded on "p" bits by performing successive divisions by 2.
- The decimal part is encoded on "q" bits by performing successive multiplications by 2 until the decimal part is zero or the number of bits q is reached.

Example : 12,625=(?)$_2$ fixed point format

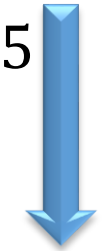
Integer part : 12 = (00001100)$_2$

Decimal part: 0,625 = (?)$_2$

0,625*2=1,25

0,25*2=0,5

0,5*2=1,0



$(12,625)_{10}=(001100,101)_2$

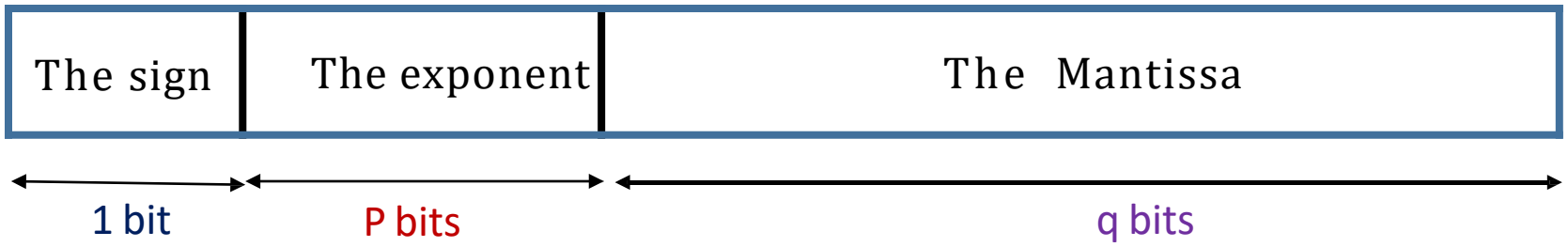
Floating point (IEEE 754 standard)

- In computing, the IEEE 754 standard has become established for the coding of floating numbers.
- In the IEEE 754 standard, a floating point number is always represented by a triple

(S; E ;M)

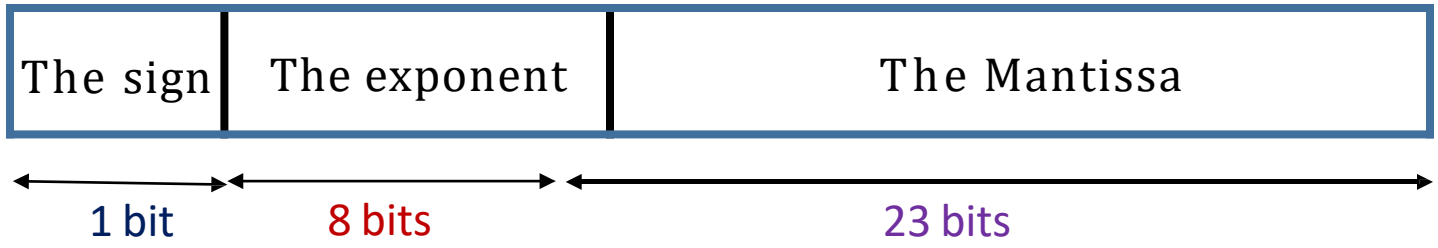
  - S: the sign is coded on 1 most significant bit

    (1: negative; and 0 positive)
  - E: the exponent
  - M: the mantissa



The IEEE 754 standard (single precision)

- **Single precision on 32 bits**
  - 1 bit of the sign
  - 8 bits for the exponent
  - 23 bits for the mantissa



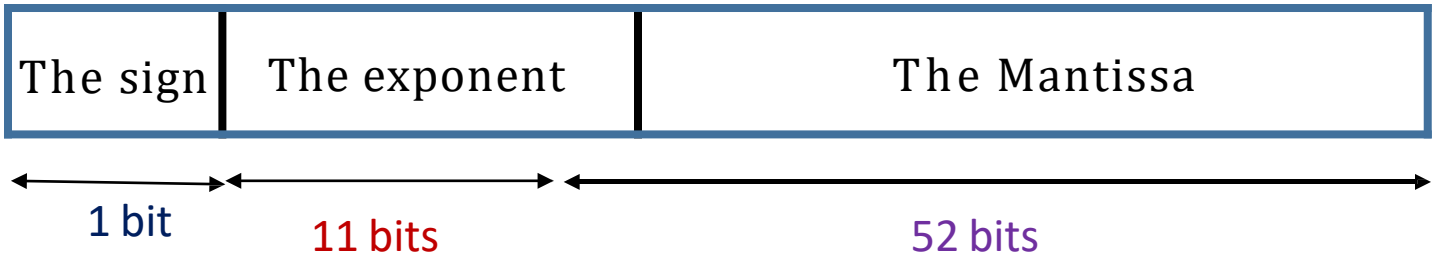
IEEE 754 standard (double precision)

- **Double precision on 64 bits :**

➢1 bit of the sign

➢11 bits for the exponent

➢ 52 bits for the mantissa



The IEEE 754 standard

IEEE 754 Coding Steps:

1. The representation of the number X in floating point format : $X = \pm 1,M \cdot 2^{dec}$

- ***Example:***

$(24,5)_{10} = (11000,1)_2$ (fixe point)

$= + 1,10001 * 2^4$ (floating point)

IEEE 754 Coding Steps

2.Calculation of the exponent E (biased/shifted or normalized)

$$\text{Exponent (E biased)} = \text{dec} + 2^{p-1} - 1$$

- Single precision (32 bits, p=8) : $E = \text{dec} + 127$
- Double precision (64 bits, p=11) : $E = \text{dec} + 1023$

Application exercise

- **Question:**

Convert the decimal number $(12,25)_{10}$ in the floating point format according to the IEEE 754 single precision standard

- ***Solution:***

Converting the number 12.25 to binary

$(12,25)_{10}=(1100,01)_2$

$=1,10001*2^3$

Hence

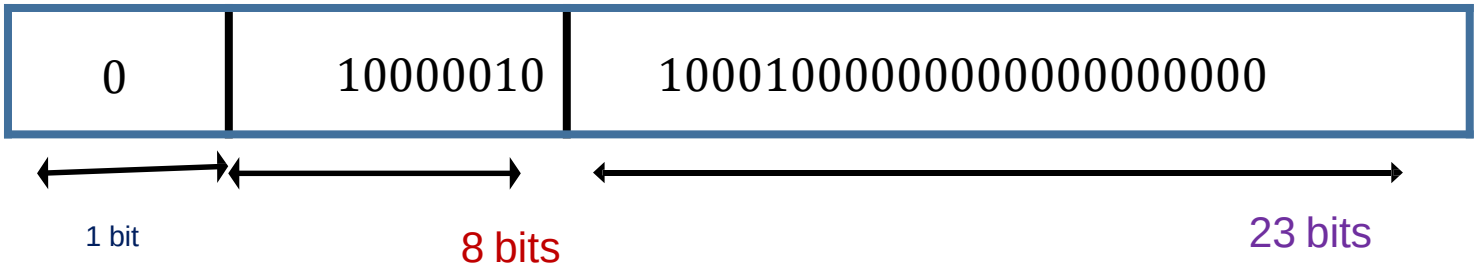
- $12,25=1,\underline{10001}*2^3$

power

The mantissa (M)

Application exercise (continued)

- Estimation of the elements of the number
  - The sign (S) =0 (positive number)
  - The 8-bit exponent : E=dec+127=3+127=130=> $(10000010)_2$
  - The mantissa M on **23** bits => 10001000000000000000000



So, the number $(12,25)_{10}$ in floating point according to the IEEE 754 single precision standard is :

01000001010001000000000000000000

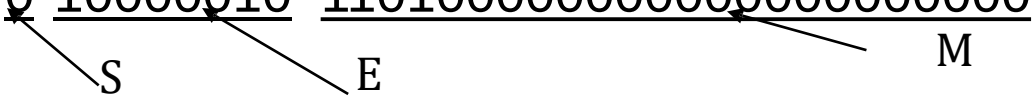
or more readably in hexadecimal representation $(41440000)_{16}$

Converting IEEE 754 to Decimal

Converting a number X from IEEE 754 to decimal means decomposing this number into its elements: S,E;M then estimating its representation in floating point format ($X = \pm 1,M \cdot 2^{dec}$)

***Example*:**

- X=(01000001011010000000000000000000)
- X= 0 10000010 11010000000000000000000



S=0 => positive number

$E=(10000010)_2=130$; E= dec+ 127 => dec=130-127=3

$X = +\ 1,M * 2^3 = 1,11010000000000000000000 * 2^3$ (dec=3)

$X=+\ (1110,10)_2 = (14,5)_{10}$