Unit 2. Digital encoding

Digital Electronic Circuits
E.T.S.I. Informática
Universidad de Sevilla

Jorge Juan-Chico <jjchico@dte.us.es> 2010-2020

Contents

- Digital encoding and digital units
- Positional number systems and binary numbers
- Base conversions
- Octal and hexadecimal
- “Real” numbers
- Binary codes
  - Numeric codes
  - Basic text, image and audio encoding

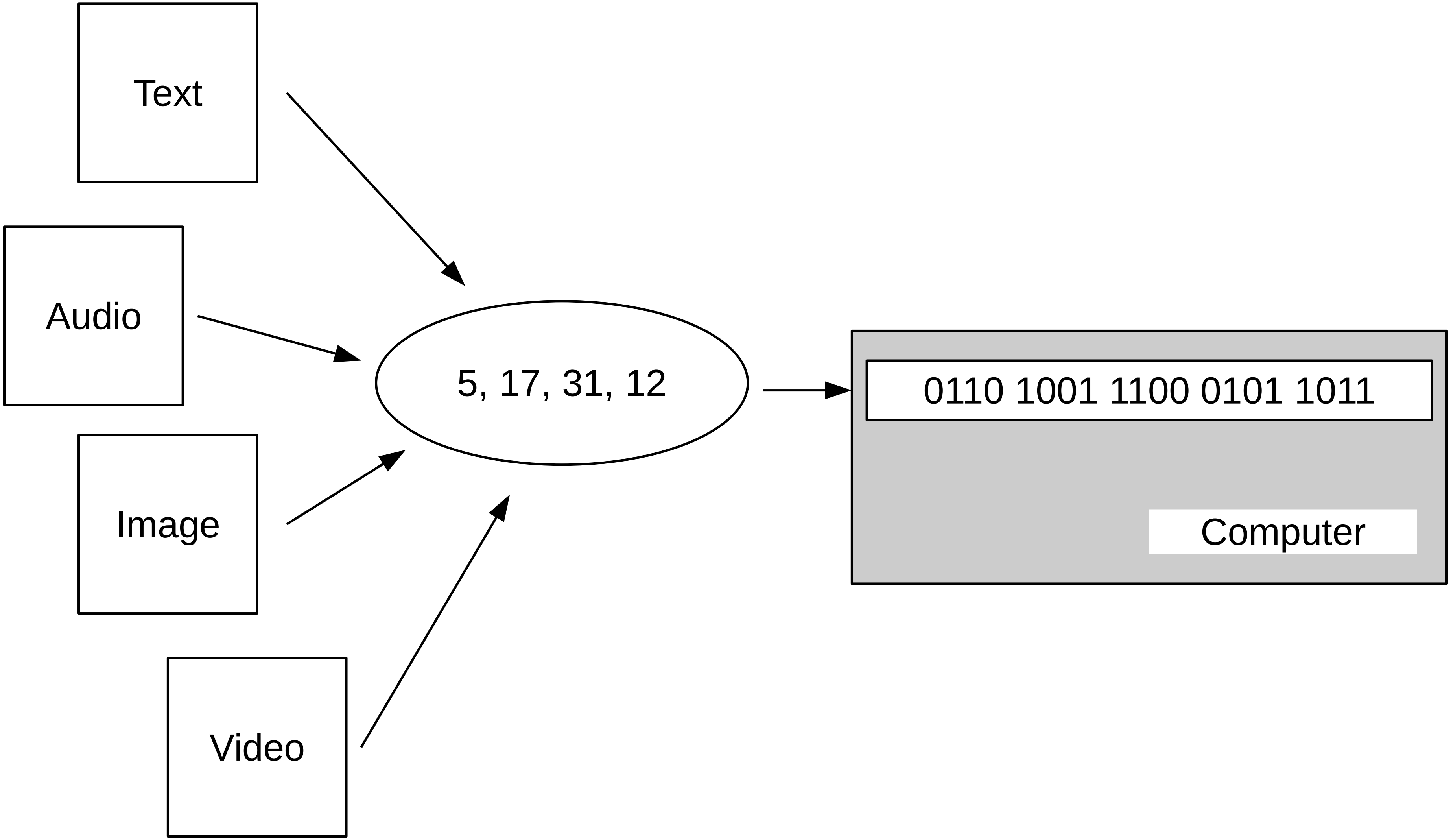
Skills

- Express byte quantities using standards.
- Represent any integer or fractional number in any base.
- Directly transform number representation between base 2 and bases 8 and 16.
- Represent decimal number in BCD encoding and vice-versa.
- Calculate the parity of a binary word and calculate the parity bit to make a define parity.
- Build the Gray code table of any number of bits.
- Calculate the raw size of image and audio data for a known set of codification parameters.

Recommended readings and exercises

- Recommended
  - LaMeres 2.1 and 2.2: Positional number systems and base conversion.
- Reference (to know more)
  - Parity bit (wikipedia)
  - Character encoding (wikipedia)
  - ASCII (wikipedia)
  - Unicode (wikipedia)
  - Raster graphics (Bitmaps) (wikipedia)
  - Pulse-code modulation –PCM– (wikipedia)
- Extra exercises from the course's collection (in Spanish)
  - Unit 1, 1 to 6

Digital encoding



Digital units

- BIT (b) (BInary digiT)
  - Symbol in the set {0,1}.
  - Minimum information unit.
- Word
  - Set of 'n' bits, typically 8, 16, 32 or 64.
  - Computers work with a whole word at a time.
- Nibble (who cares?)
  - 4 bit word.
- Byte (B)
  - 8 bit word.
  - Basic practical information unit in Information Technology (IT).

Digital units

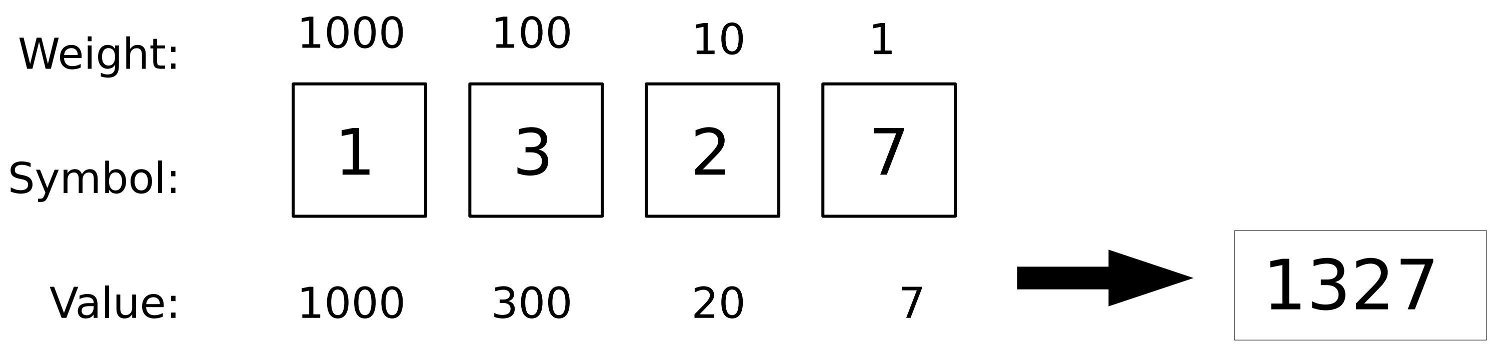
- Tradition: IS units with slightly changed meaning (powers of 2 instead of 10)
- Non-uniform use of digital units:
  - Diskette: 1.44MB = 1000 KB = 1000x1024B
  - Disc 160GB = 160000 MB = 160x1000x1024x1024B
  - DVD 4,7GB = 4700MB = 4,7x1000x1024x1024B
- There is a (not very used) standard for binary units:
  - IEC, IEEE-1541-2002

SI			**Binary**	**IEC**	
kilo	k	10^3	2^{10}	kibi	Ki
mega	M	10^6	2^{20}	mebi	Mi
giga	G	10^9	2^{30}	gibi	Gi
tera	T	10^{12}	2^{40}	tebi	Ti
peta	P	10^{15}	2^{50}	pebi	Pi
exa	E	10^{18}	2^{60}	exbi	Ei
zetta	Z	10^{21}	2^{70}	zebi	Zi

Positional number system and binary numbers

- Decimal system:
  - 10 symbols {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  - Base 10

$$1327 = 1\times10^3 + 3\times10^2 + 2\times10^1 + 7\times10^0$$



Positional number system and binary numbers

- Binary system:
  - 2 symbols {0,1}
  - Base 2

$$1101 = 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0$$

Weight:	8	4	2	1	
Symbol:	1	1	0	1	
Value:	8	4	0	1	→ 13

Positional number system and binary numbers

- In general
  - Magnitude x
  - Base b
  - n figures: $\{x_i\}$

$$x = x_{n-1} \times b^{n-1} + \ldots + x_1 \times b^1 + x_0 \times b^0$$

Maximum representable number: b^n-1

Base 'b' to base '10' conversion

- Simply by applying the formula
  - Ej: $234_{(7}$

$$x = x_{n-1} \times b^{n-1} + \ldots + x_1 \times b^1 + x_0 \times b^0$$

Base '10' to base 'b' conversion

- Successive division by the target base
  - Successive reminders are the base-n figures from less significant to most significant

$$123_{(10} \rightarrow 234_{(7}$$

1 2 3 | 7

5 3 1 7 | 7

4 3 2

Fast conversion using a weights chart
8-bit base 2 example

Starting at 128, place 1's to sum up the desired number.

Copy the bits.

	128	64	32	16	8	4	2	1	
$165_{(10}$ ⇒	1	0	1	0	0	1	0	1	⇒ $10100101_{(2}$

Move the bits under the chart.

Sum up the weights with 1's.

	128	64	32	16	8	4	2	1	
$01001110_{(2}$ ⇒	0	1	0	0	1	1	1	0	⇒ $78_{(10}$

Octal and hexadecimal

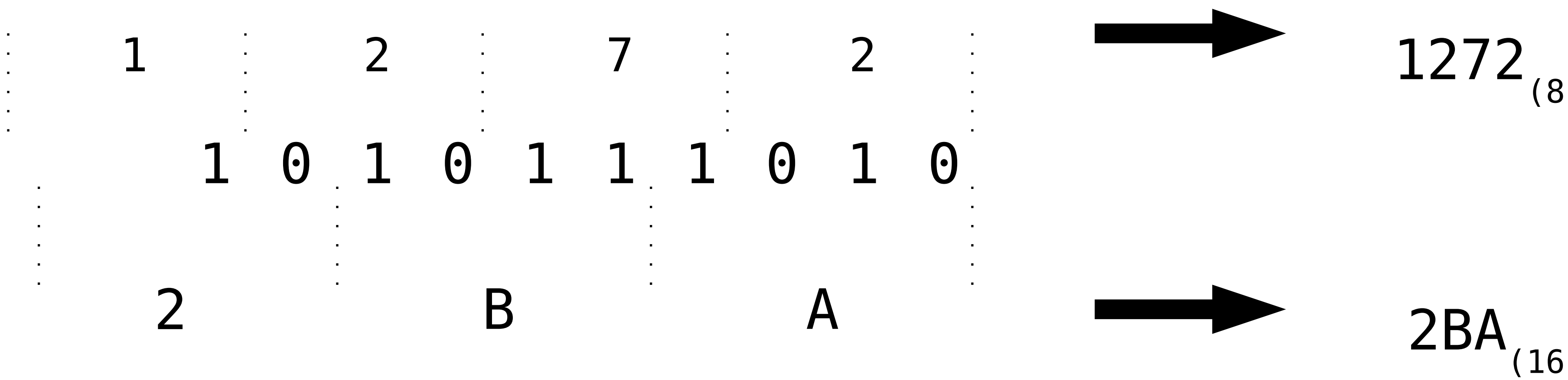
- Base 8 (octal):
  - {0, 1, 2, 3, 4, 5, 6, 7}
- Base 16 (hexadecimal):
  - {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
- Compact way to represent binary numbers
  - 1 octal figure = 3 binary figures
  - 1 hexadecimal figure = 4 binary figures

Octal and hexadecimal

B-8	B-2
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

B-10	B-16	B-2
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Octal and hexadecimal



$2BA_{(16} = 2BA_{h} = 2BA_{hex}$ = #2BA = $2BA = 0x2BA = 10'h2BA

Hexadecimal is very compact and converting to binary is straightforward. That's why you will nor normally see computers printing binary numbers even in low-level routines. E.g. try 'dmesg | less' in a Linux terminal.

Example: Working with number bases in Python

```
$ python3

>>> x = 215
>>> hex(x)          # hexadecimal rep.
'0xd7'
>>> bin(x)          # base 2 rep.
'0b11010111'

>>> y = 0b1011      # binary value
>>> y
11
>>> z = 0x2c        # hexadecimal value
>>> z
44

>>> x + y
226
>>> hex(x+y)
'0xe2'
>>> bin(x+y)
'0b11100010'

>>> x - z
171
>>> hex(x-z)
'0xab'
>>> bin(x-z)
'0b10101011'
```

```
>>> int('321',7)        # base 7
162

# 321(7 + 121(3
>>> x = int('321',7) + int('212',3)

>>> x
185
>>> hex(x)
'0xb9'
>>> bin(x)
'0b10111001'

# base n representation?

>>> from numpy import base_repr
>>> x = 52
>>> base_repr(x, 2)
'110100'
>>> base_repr(x, 3)
'1221'
>>> base_repr(x, 4)
'310'
>>> base_repr(x, 5)
'202'
>>> base_repr(x, 6)
'124'
>>> base_repr(x, 7)
'103'
```

“Real” numbers

$$x = x_{n-1} \times b^{n-1} + \ldots + x_0 \times b^0 + x_{-1} \times b^{-1} + \ldots + x_{-m} \times b^{-m}$$

integer part | fractional part

0 1 1 0 1 • 1 0 1

“Real” numbers base conversion

- Base b to base 10 conversion
  - Directly: just operate in base 10.

$$10{,}101_2 = 1\times 2^{1} + 0\times 2^{0} + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3}$$

$$2 + 1/2 + 1/8 = 2{,}625_{10}$$

- Base 10 to base b:
  - Integer part: like integer numbers
  - Fractional part: successive multiplication by the target base. Take the integer part of the result.

“Real” numbers base conversion

- Example: $12{,}3_{(10}$
  - $12_{(10} = 1100_{(2}$
  - 0,3 x 2 = 0,6 → “0”
  - 0,6 x 2 = 1,2 → “1”
  - 0,2 x 2 = 0,4 → “0”
  - 0,4 x 2 = 0,8 → “0”
  - 0,8 x 2 = 1,6 → “1”
  - 0,6 x 2 = 1,2 → “1” (first repeated bit)
- $12{,}3_{(10} = 1100{,}0100110011001\ldots_{(2} = 1100{,}0\overline{1001}_{(2}$

A number with finite decimal digits in one base may have infinite decimal digits in another base!

“Real” numbers. Some interesting questions (a bit advanced)

- When you get infinite decimals after a conversion, will decimals always repeat periodically?
  - Clue: can a “rational” number become “irrational” after a conversion?
- How do you convert a number with infinite (but periodic) decimals?
  - Clue: starting with an equivalent fraction may be useful here.
- Can you tell in advance if a number will have a finite or infinite number of decimals in a base before converting to it?
  - Clue: equivalent fraction are also useful here.

Fixed point and floating point

- Fixed point real number
  - The number of bits used for the fractional part is fixed.
  - Not good for very big and very small numbers.
- Floating point real numbers
  - The position of the decimal point is determined by an exponent and a significand.
  - Very similar to the scientific notation for numbers.
  - The significand and exponent are stored with a fixed number of bits.
- Most computers use the IEEE 754 standard for floating point
  - 16 to 128 bit formats
  - Can use base 2 or base 10
  - Can represent “infinite” and “zero with sign”.

Wikipedia

$$\text{number} = \text{significand} \times \text{base}^{\text{exponent}}$$

“Real” numbers and digital systems

- In a digital system the number of available bits is always limited, so there are many numbers that cannot be exactly represented in base 2 even if they can be represented in base 10.
- It is a potential source of sever errors even at the software level.
- E.g.: representing 12,3 with an 8-bit limit:

$$12{,}3_{10} = 1100{,}0\,\widehat{1001}_2 \approx 1100{,}0100_2 = 12{,}25_{10}$$

In fact, a binary digital system can only store a subset of real numbers using base 2 representation: rational numbers that can be represented with finite digits in base 2.

“Real” numbers and digital systems

```
$ python3

>>> x = 12.3     # this value is internally stored in b. 2
>>> y = 3 * x    # operations take place in b. 2
>>> z = y / 3

>>> x == z       # !?
False

>>> z - x
1.7763568394002505e-15

>>> from decimal import Decimal

>>> Decimal(x)   # b.10 representation of the value in x
Decimal('12.300000000000000710542735760100185871124267578125')
>>> Decimal(z)
Decimal('12.3000000000000024868995751603506505489349365234375')

>>> if x != z:   # kaboooom!
...     print("Destroy the world!!!")
...
Destroy the world!!!
```

Never check if a real number is exactly the same as another real number: they are probably not even if you think they should.

Binary codes

- Assignment of binary values to a set of symbols
  - Numbers, characters, colors, etc.
- Usually, fixed width words (e.g. 8 bits) but not always.
- Code assignment seeks good properties
  - Similar codes for adjacent numbers.
  - Similar codes for similar colors.
  - Similar codes for lower and upper-case letters.
  - Etc.
- Not all binary words need to have a corresponding symbol assigned.

Natural binary code

- Natural integer coding using base-2 binary numbers

Number	Code
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Gray code

Number	1 bit	2 bits	3 bits	4 bits
0	0	00	000	0000
1	1	01	001	0001
2		11	011	0011
3		10	010	0010
4			110	0110
5			111	0111
6			101	0101
7			100	0100
8				1100
9				1101
10				1111
11				1110
...				...

- Code positive integers.
- Consecutive symbols only differ in one bit (distance 1).
- The n-bit code is built by reflecting the n-1-bit code

One-hot code

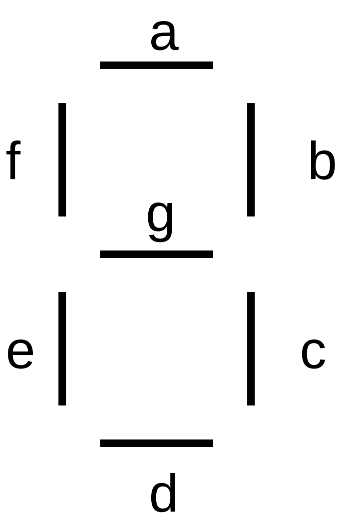
Number	Código
0	00000001
1	00000010
2	00000100
3	00001000
4	00010000
5	00100000
6	01000000
7	10000000

- Code words with only one bit set to '1'
- Pros:
  - Easy to decode.
  - Allow for error detection.
- Cons:
  - Number of bits = number of symbols to code.
  - May use too many bits.

BCD codes (*Binary Coding Decimal*)

- Binary coding of base-10 digits

Digit	natural	excess-3	2-4-2-1	2-out-of-5	7 segm. abcdefg
0	0000	0011	0000	00011	1111110
1	0001	0100	0001	00101	0110000
2	0010	0101	0010	00110	1101101
3	0011	0110	0011	01001	1111001
4	0100	0111	0100	01010	0110011
5	0101	1000	1011	01100	1011011
6	0110	1001	1100	10001	1011111
7	0111	1010	1101	10010	1110000
8	1000	1011	1110	10100	1111111
9	1001	1100	1111	11000	1110011



Parity bit

- The parity of a binary word is “even”/”odd” if the number of bits set to “1” is even/odd.
- Parity bit: additional bit added to a code to achieve a determined parity, even or odd

E.g. natural binary code with “even” leading parity bit.

Number	Code
0	0000
1	1001
2	1010
3	0011
4	1100
5	0101
6	0110
7	1111

Text encoding

- Binary encoding of text symbols or 'characters'.
- Include graphical and control symbols:
  - new line, tab, end of file, etc.
- There are many text “encodings” around for historical and technical reasons.
- Evolution
  - ASCII (1967)
    - One of the first standards. Used in teletypes, etc.
  - ISO/IEC 8859-X (1987-2000)
    - 8-bit extensions to ASCII (compatibility).
    - Different encodings for different language families.
  - Unicode –ISO/IEC 10646– (1991)
    - Universal encoding for all languages

ASCII

0	00	NUL	32	20	SPC	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	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

https://en.wikipedia.org/wiki/ASCII

ISO/IEC-8859-15

ASCII

extension

ISO-8859-15																
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8x	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9x	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
Ax	NBSP	¡	¢	£	€	¥	Š	§	š	©	ª	«	¬	SHY	®	¯
Bx	°	±	²	³	Ž	µ	¶	·	ž	¹	º	»	Œ	œ	Ÿ	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

https://en.wikipedia.org/wiki/ISO/IEC_8859-15

ASCII control codes

- Initially used in teletypes (many of them are not in use anymore).
- Used for:
  - Format: space, tab, end of line, etc.
  - Control: end of transmission, end of text, etc.
- Present in (raw) text documents and in terminals.
  - Terminal: introduced with Ctrl+<letter> (^<letter>)
- Examples
  - ETX (^C): end of text
  - EOT (^D): end of transmission
  - BEL (^G): bel
  - BS (^H): back space
  - HT (^I): horizontal tab
  - LF (^J): line feed
  - CR (^M): carriage return
  - DEL (^?): delete

ISO/IEC 10646 (Unicode)

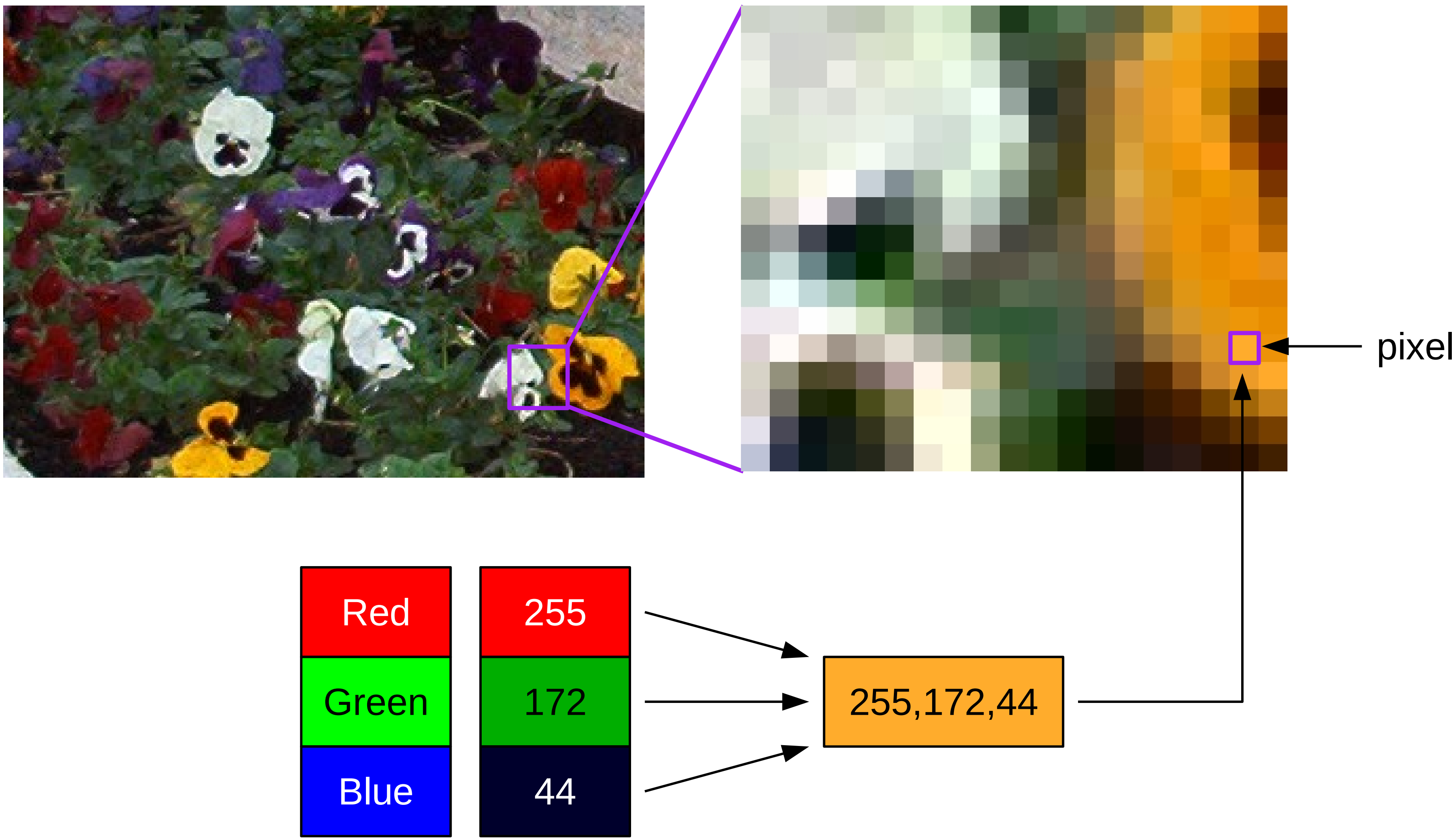
- Universal encoding for all languages
- Around 144000 characters as of 2020
- Can use different encoding formats
- UTF-8
  - Most popular encoding format for Unicode
  - Use 1 to 4 bytes
  - Compatible with 8-bit ASCII

https://en.wikipedia.org/wiki/Unicode

Text encoding. Example

```
$ echo "Muñoz Pérez 5€ debe" > texto.txt

$ cat text.txt
Muñoz Pérez 5€ debe

$ hd text.txt
00000000  4d 75 c3 b1 6f 7a 20 50  c3 a9 72 65 7a 20 35 e2  |Mu..oz P..rez 5.|
00000010  82 ac 20 64 65 62 65 0a                           |.. debe.|

$ iconv -f utf8 -t latin1 text.txt > text_l1.txt
iconv: illegal input sequence at position 15

$ iconv -f utf8 -t latin9 text.txt > text_l9.txt
$ cat text_l9.txt
Mu�oz P�rez 5� debe

$ hd text_l9.txt
00000000  4d 75 f1 6f 7a 20 50 e9  72 65 7a 20 35 a4 20 64  |Mu.oz P.rez 5. d|
00000010  65 62 65 0a                                       |ebe.|

$ file text.txt text_l9.txt
text.txt:     UTF-8 Unicode text
text_l9.txt:  ISO-8859 text

$ ls -l text.txt text_l9.txt
-rw-rw-r-- 1 jjchico jjchico 20 oct  1 16:27 text_l9.txt
-rw-rw-r-- 1 jjchico jjchico 24 oct  1 16:23 text.txt
```

Graphics (image)

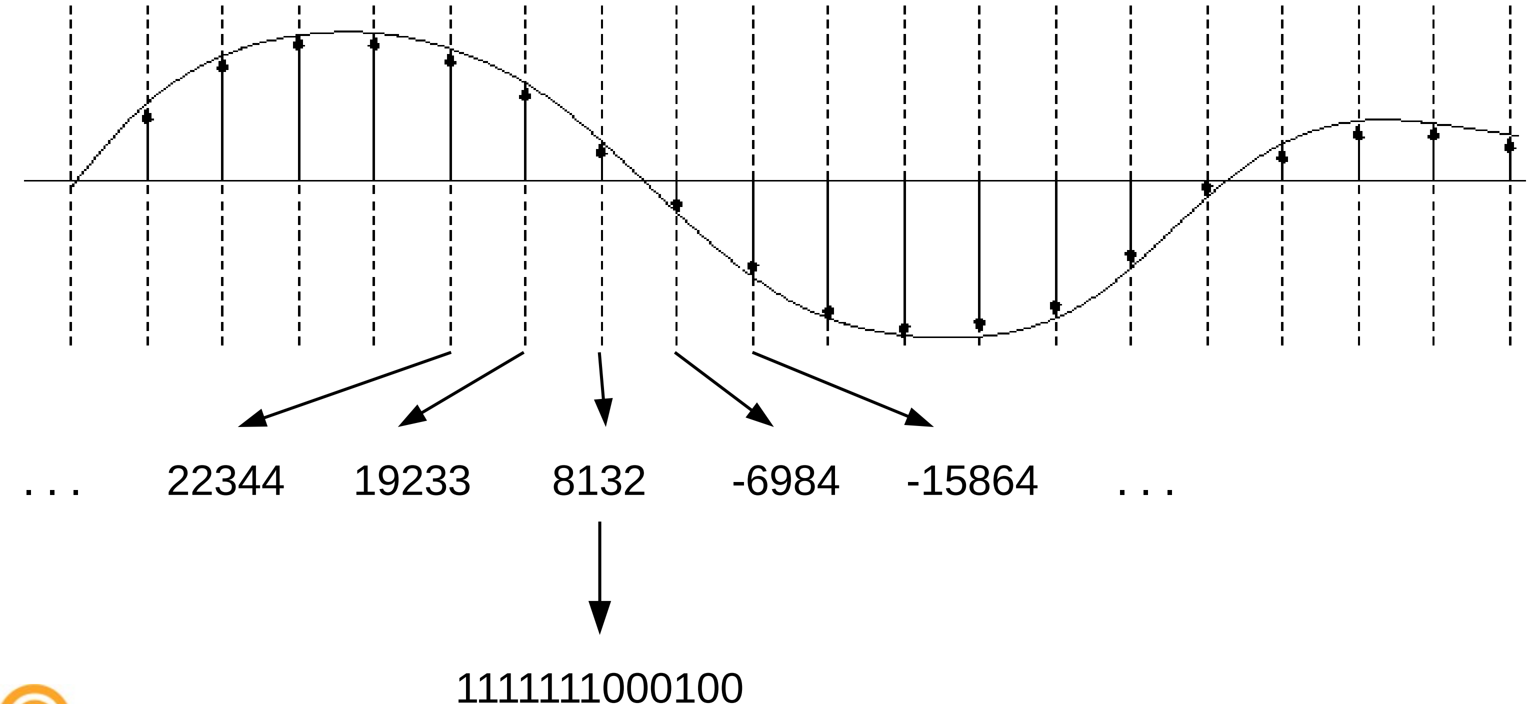


Graphics. Raw calculations

- Color depth (color_depth)
  - Number of bits used to encode the color of each pixel.
- Number of pixels (n_pixel)
  - n_pixel = pixel_width x pixel_highth
- Raw size –without compression– (size)
  - size = n_pixel x color_depth
- Image resolution (res)
  - Can be horizontal or vertical (normally the same)
  - Pixel width or height divided by real distance.

Sonido

- Raw encoding parameters (determine quality)
  - Sampling rate: number of samples per unit time (e.g. 44100Hz)
  - Data resolution: size of data (bits) per sample (e.g. 16 bits)
  - Number of channels (N): number of encoded waves (e.g. 2 -stereo-)



Encoded sound calculation

- Data size (L)
  - Size of encoded data.
- Encoded time (T)
  - Duration of encoded fragment.
- Data rate (R)
  - Encoded data per time unit.
  - Also “bit rate” when in bits per second.
  - L = R x T
- R calculation without compression
  - R = sampling_rate x data_resolution x N
- Compression
  - Lossless: lower R but keeping quality parameters (~1/2)
  - Lossy: even lower R by assuming some quality penalty (~1/10)