Università
Ca' Foscari
Venezia

Computer Science Applications to Cultural Heritage

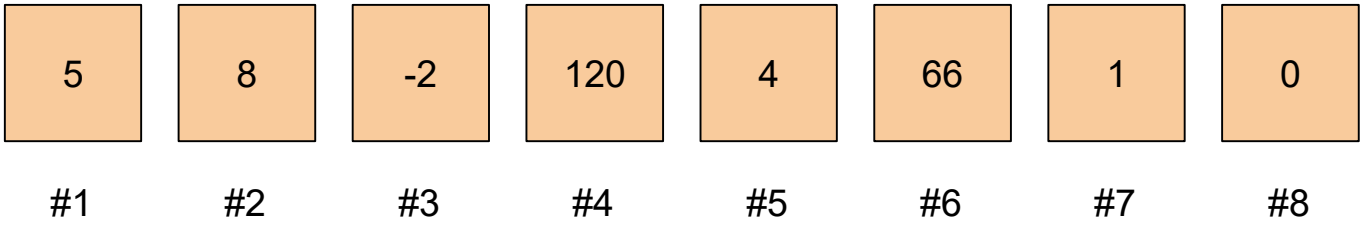
Introduction to computer systems

Filippo Bergamasco (filippo.bergamasco@unive.it)
http://www.dais.unive.it/~bergamasco
DAIS, Ca' Foscari University of Venice
Academic year 2018/2019

Università Ca' Foscari Venezia

A computing game

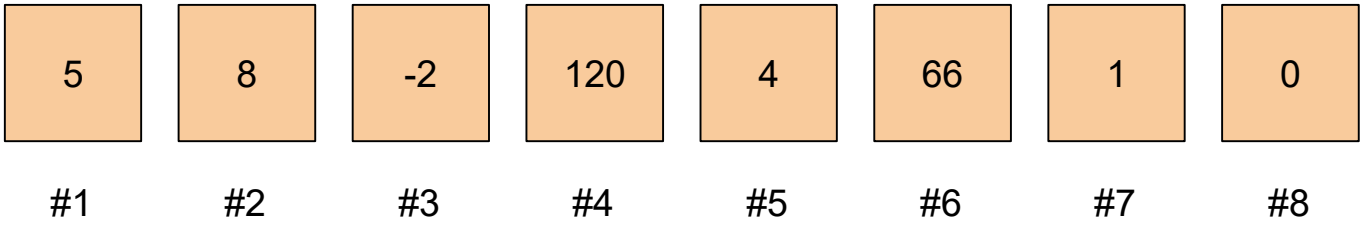
- Tom plays a game in which he pretends to be a computer
- Equipment:
  - A set of boxes
  - Each box is identified with a number (address)
  - Each box contains a piece of paper with a single number written in it



A computing game

Tom must sequentially execute instructions (commands) which operate on the box content.
Example:

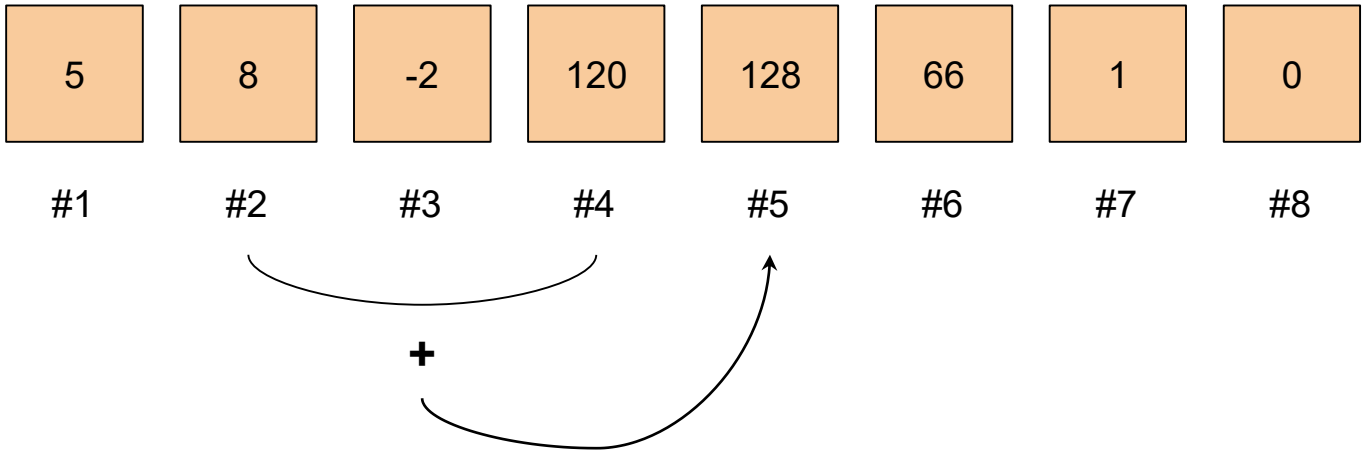
- Add the number contained in box #2 to the number contained in box #4 and place the result in box #5



Università Ca' Foscari Venezia

A computing game

- Add the number contained in box #2 to the number contained in box #4 and place the result in box #5

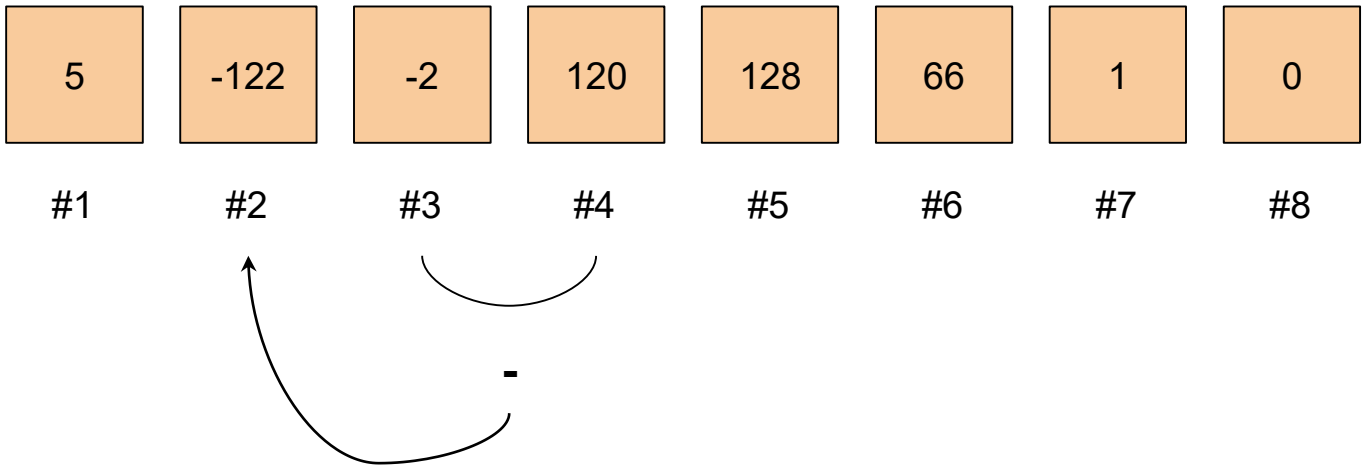


Università Ca' Foscari Venezia

A computing game

Another example:

- Subtract the number contained in box #4 to the number contained in box #3 and place the result in box #2

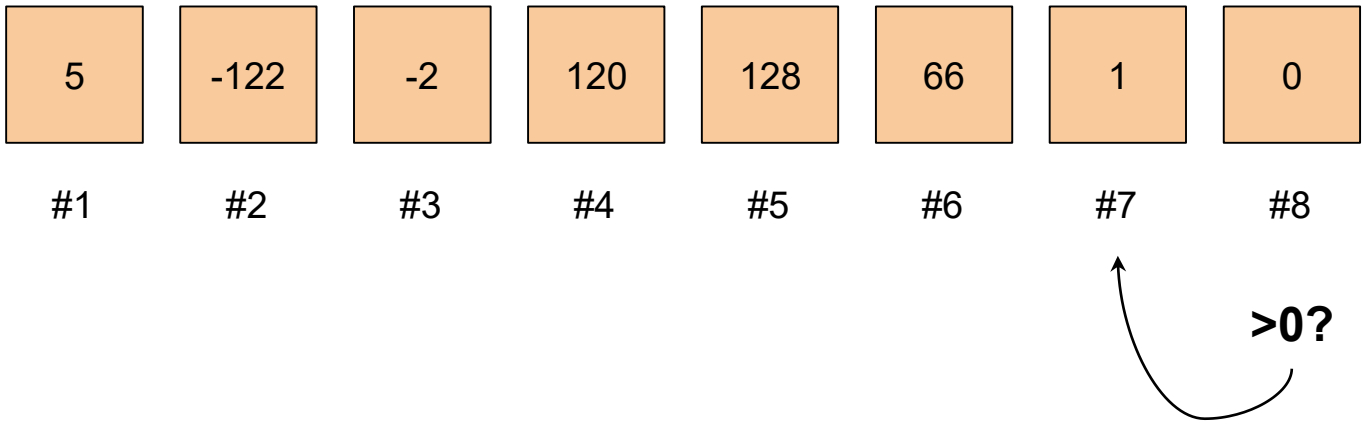


Università Ca' Foscari Venezia

A computing game

Another example

- If box #8 is greater than zero, put number 10 in box #7. Do nothing otherwise



A computing game

This simple game simulate how a real computer works:

A computer consists of a **memory** (the boxes) and a **CPU** (Tom) that can change the content of the boxes by following a sequence of instructions

- A real computer (just like Tom) can only execute a limited set of simple instructions operating on numbers
- A real computer continue indefinitely to execute instructions (Tom never get tired)

More on instructions

To be useful, Tom needs a sequence of instructions that change the content of the boxes to produce a result. Informally we can refer to this sequence as an algorithm.

We use algorithms everyday:

- Cooking recipes
- Instructions describing how to use a device (like the microwave)
- Directions to reach a certain place

Algorithms

An algorithm is an ordered set of **unambiguous executable steps** that defines a terminating process.

To be executed, an algorithm is converted into a sequence of instructions that are valid for a specific type of computer (Tom, Intel processors, ARM processors, etc)

If an algorithm is programmable on one computer, then in principle it is programmable on any computer.

A simple algorithm

Let's define an algorithm that, given two numbers (m,n) in box #1 and #2, puts in box #3 the quotient and in box #4 the remainder of the division m/n

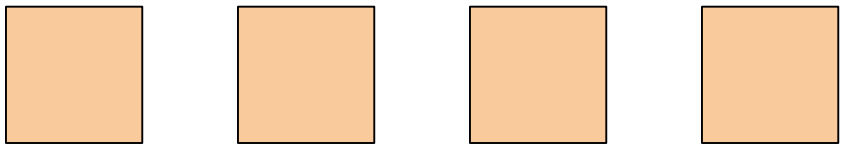
- Content of box #1 and #2 are called **inputs**
- Content of box #3 and #4 are called **outputs**

We also suppose that the only instructions that Tom can perform are:

- Place a number in a box
- Sum/Subtract
- Compare if a number contained in a box is greater than some other

A simple algorithm

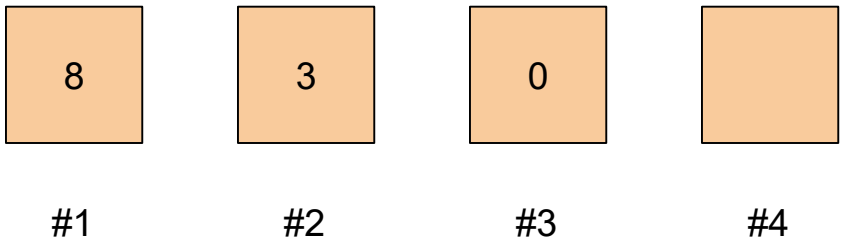
1. Put 0 in box #3
2. Put the content of box #1 in box #4
3. If content of #4 is >= than content of #2
   a. Subtract #2 to #4 and place the result in #4
   b. Add 1 to #3 and place the result in #3
   c. Return to step 3
4. HALT



...let's try to execute it!

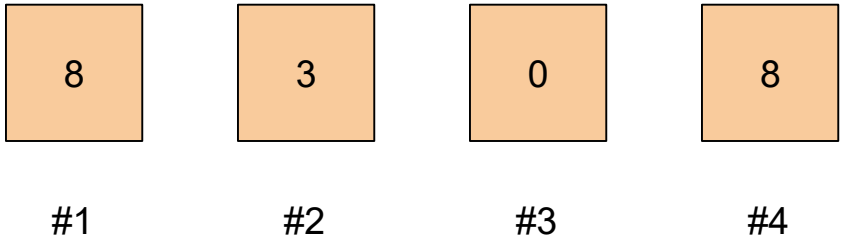
A simple algorithm

1. Put 0 in box #3
2. Put the content of box #1 in box #4
3. If content of #4 is >= than content of #2
   a. Subtract #2 to #4 and place the result in #4
   b. Add 1 to #3 and place the result in #3
   c. Return to step 3
4. HALT



A simple algorithm

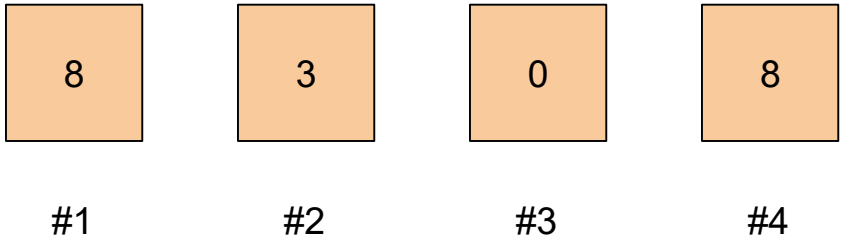
1. Put 0 in box #3
2. Put the content of box #1 in box #4
3. If content of #4 is >= than content of #2
   a. Subtract #2 to #4 and place the result in #4
   b. Add 1 to #3 and place the result in #3
   c. Return to step 3
4. HALT



A simple algorithm

1. Put 0 in box #3
2. Put the content of box #1 in box #4
3. If content of #4 is >= than content of #2
    a. Subtract #2 to #4 and place the result in #4
    b. Add 1 to #3 and place the result in #3
    c. Return to step 3
4. HALT

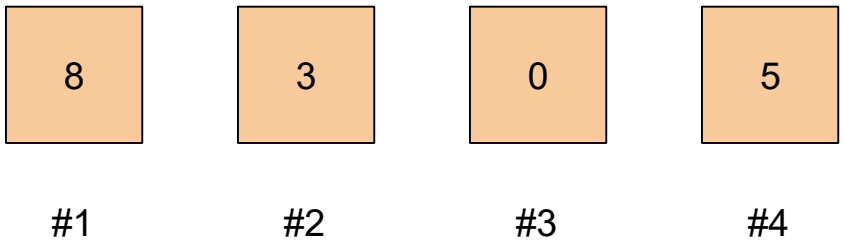
Yes!



Università Ca' Foscari Venezia

A simple algorithm

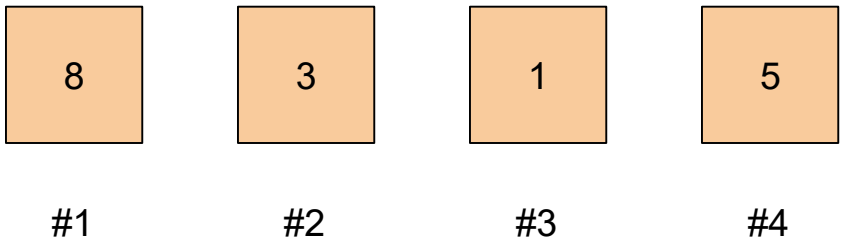
1. Put 0 in box #3
2. Put the content of box #1 in box #4
3. If content of #4 is >= than content of #2
   a. Subtract #2 to #4 and place the result in #4
   b. Add 1 to #3 and place the result in #3
   c. Return to step 3
4. HALT



Università Ca' Foscari Venezia

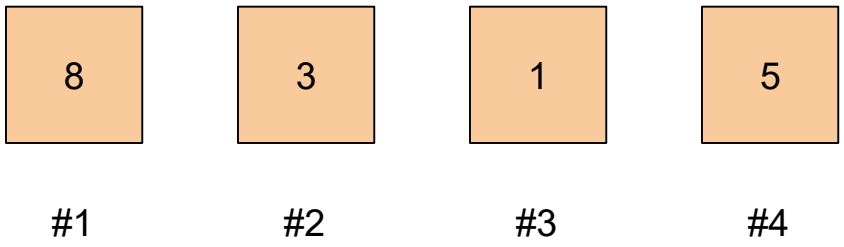
A simple algorithm

1. Put 0 in box #3
2. Put the content of box #1 in box #4
3. If content of #4 is >= than content of #2
   a. Subtract #2 to #4 and place the result in #4
   b. Add 1 to #3 and place the result in #3
   c. Return to step 3
4. HALT



A simple algorithm

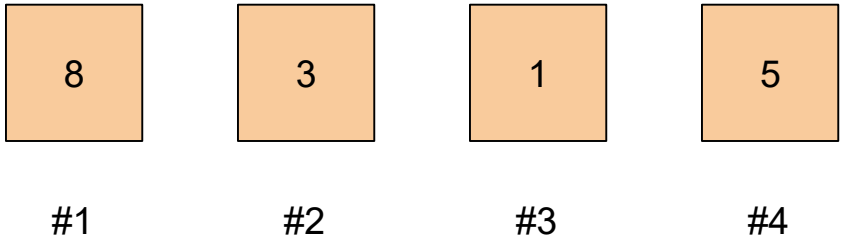
1. Put 0 in box #3
2. Put the content of box #1 in box #4
3. If content of #4 is >= than content of #2
   a. Subtract #2 to #4 and place the result in #4
   b. Add 1 to #3 and place the result in #3
   c. Return to step 3
4. HALT



A simple algorithm

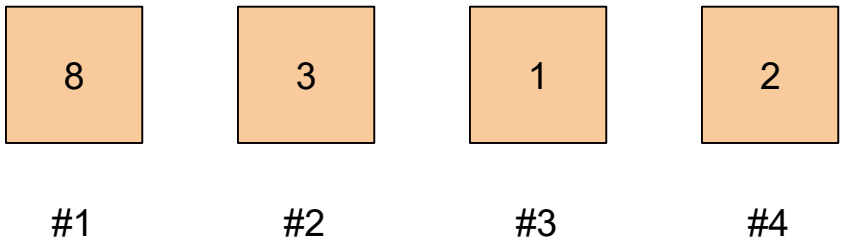
1. Put 0 in box #3
2. Put the content of box #1 in box #4
3. If content of #4 is >= than content of #2
   a. Subtract #2 to #4 and place the result in #4
   b. Add 1 to #3 and place the result in #3
   c. Return to step 3
4. HALT

Yes!



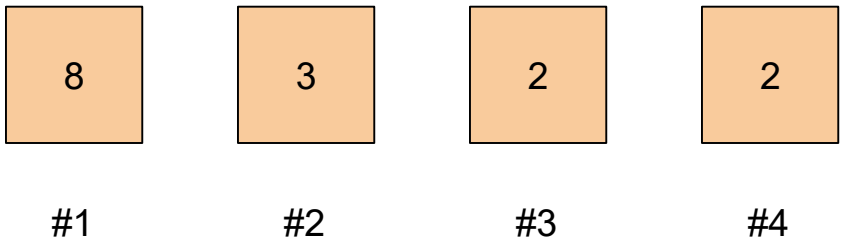
A simple algorithm

1. Put 0 in box #3
2. Put the content of box #1 in box #4
3. If content of #4 is >= than content of #2
   a. Subtract #2 to #4 and place the result in #4
   b. Add 1 to #3 and place the result in #3
   c. Return to step 3
4. HALT



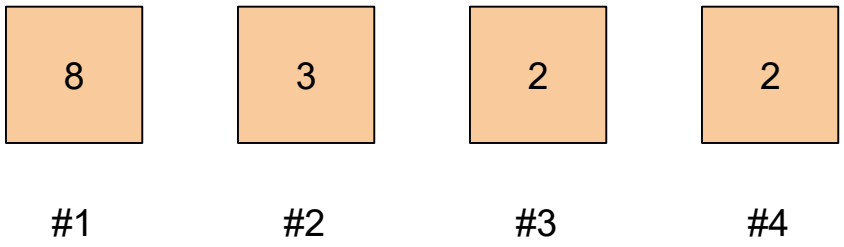
A simple algorithm

1. Put 0 in box #3
2. Put the content of box #1 in box #4
3. If content of #4 is >= than content of #2
   a. Subtract #2 to #4 and place the result in #4
   b. Add 1 to #3 and place the result in #3
   c. Return to step 3
4. HALT



A simple algorithm

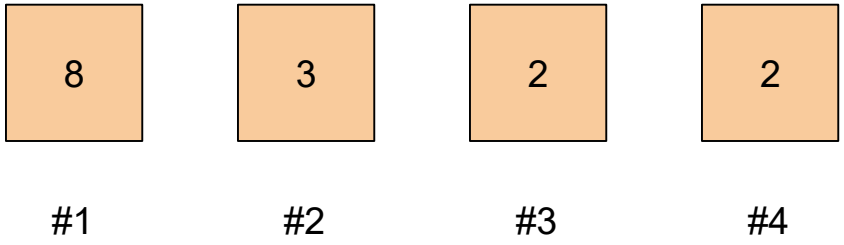
1. Put 0 in box #3
2. Put the content of box #1 in box #4
3. If content of #4 is >= than content of #2
   a. Subtract #2 to #4 and place the result in #4
   b. Add 1 to #3 and place the result in #3
   c. Return to step 3
4. HALT



A simple algorithm

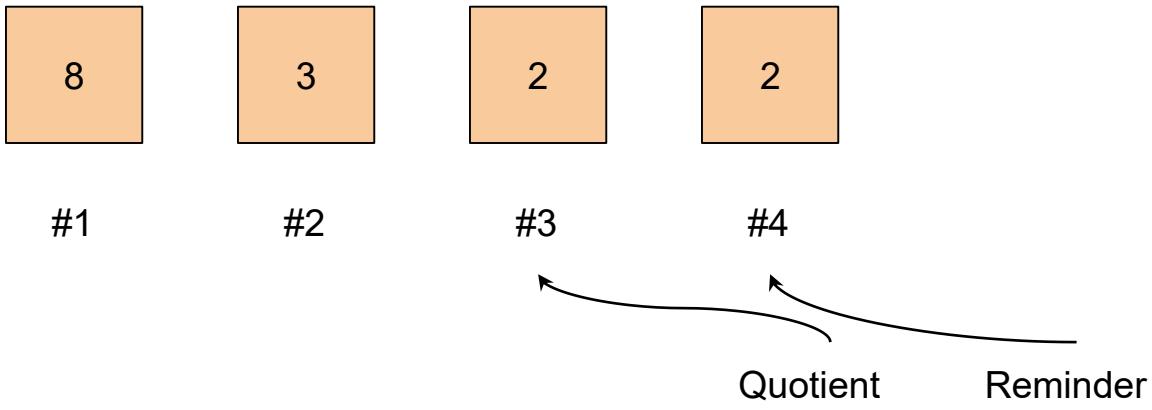
1. Put 0 in box #3
2. Put the content of box #1 in box #4
3. If content of #4 is >= than content of #2
    a. Subtract #2 to #4 and place the result in #4
    b. Add 1 to #3 and place the result in #3
    c. Return to step 3
4. HALT

NO



A simple algorithm

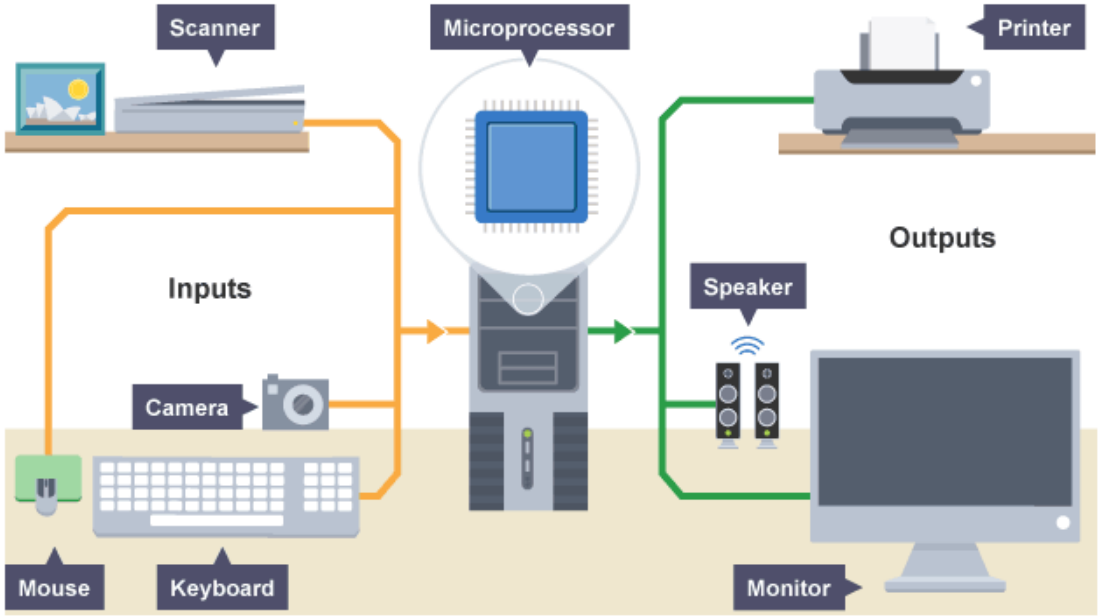
1. Put 0 in box #3
2. Put the content of box #1 in box #4
3. If content of #4 is >= than content of #2
   a. Subtract #2 to #4 and place the result in #4
   b. Add 1 to #3 and place the result in #3
   c. Return to step 3
4. HALT



Università Ca' Foscari Venezia

Modern computers

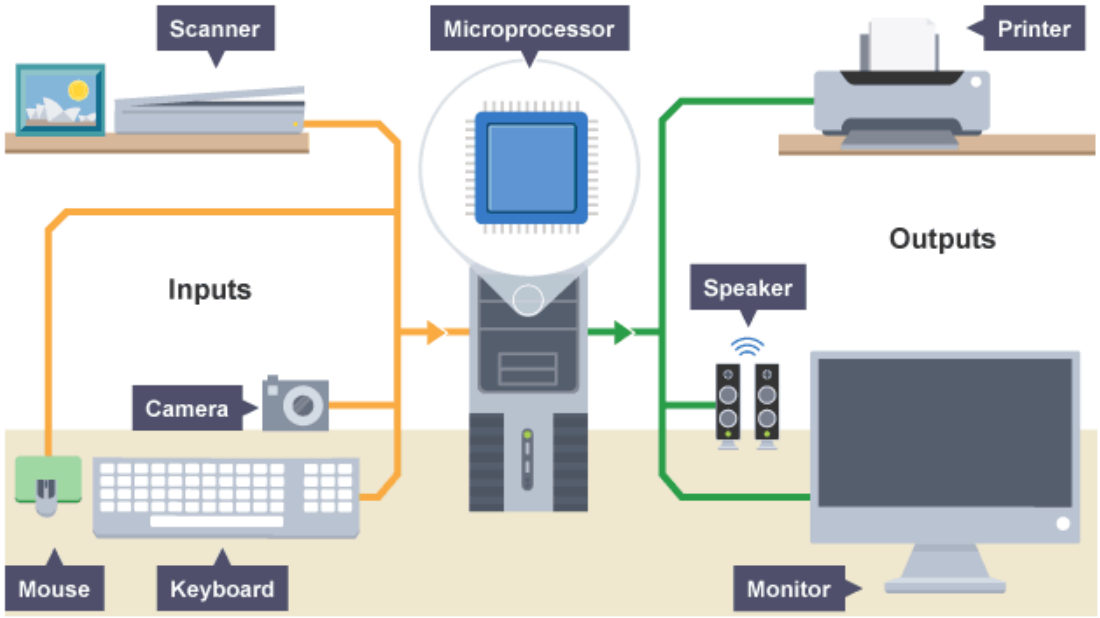
So far we have seen the basic concepts of a basic computing device. As you may imagine, a real-world computer is far more complex than that.



Università Ca' Foscari Venezia

Modern computers

As we have seen in our idealized "computing game", the hardware that characterize a computer is the **CPU** (which executes instructions) and **Memory** (which stores data used by the CPU)



To be useful in practice, computers often control different input-output devices (monitor, printers, keyboard, mouse, cameras, etc.)

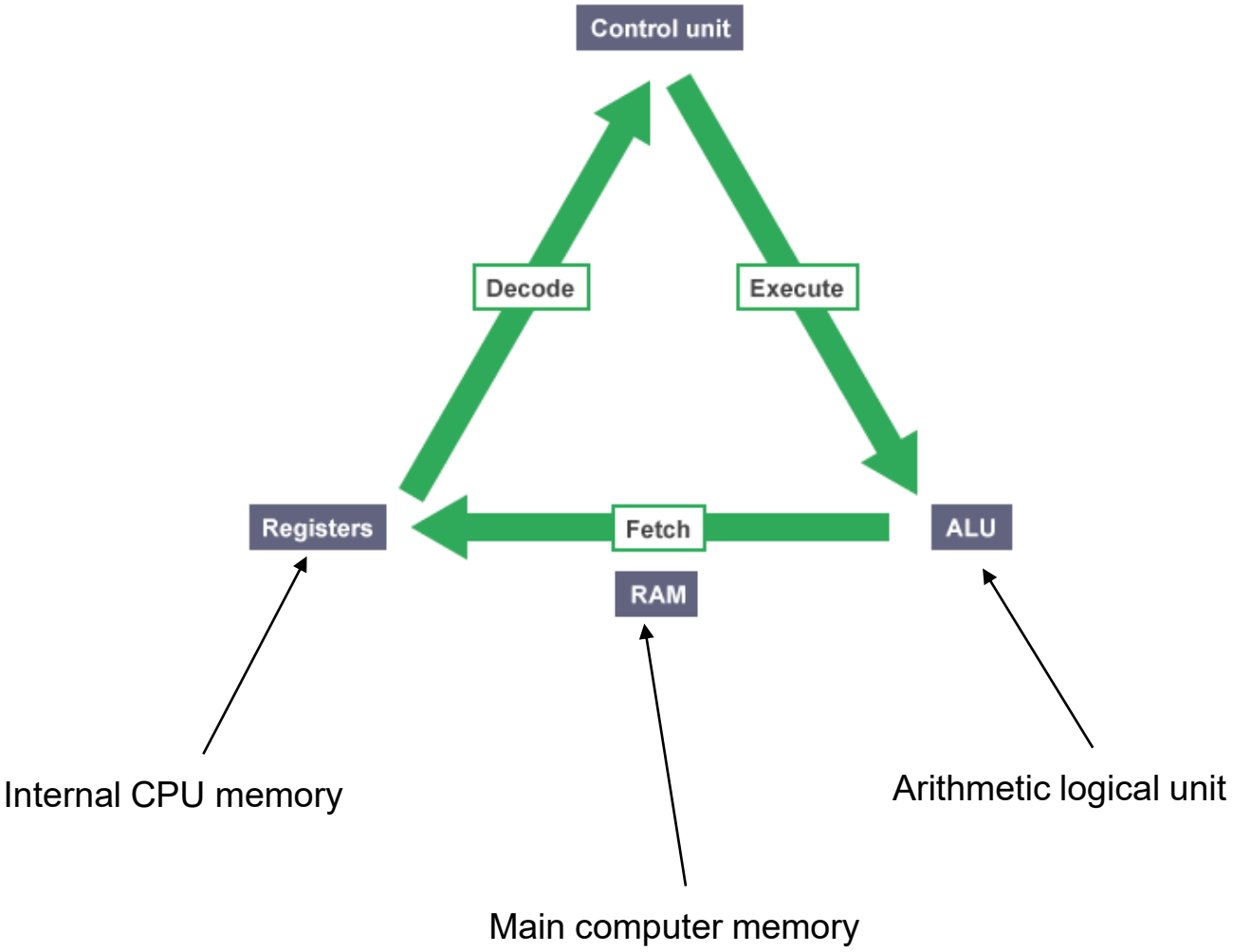
The CPU

CPU (Control Processing Unit) is the core electronic circuitry of a computer. As soon as it is turned on, it executes a series of stored instructions (called a program) in a cycle composed by 3 fundamental steps:

1. **Fetch**: retrieves an instruction from program memory
2. **Decode**: fetched instruction is converted into electrical signals to control/configure other parts of the CPU to the subsequent execution
3. **Execute**: the CPU performs the requested operation. The result often changes the internal registers or main memory directly.

Università Ca' Foscari Venezia

The CPU



Instructions

Where are the instruction stored?

In Von Neumann architecture (Like the computer you typically use) the instructions are stored in the main memory just like any other data (inputs and outputs)

Each specific architecture defines a machine code that maps numbers to specific instructions:

Examples:

13 means sum "1" to the box #3

245 means subtract box #4 to the box #5 and place the result in #4

Compilers

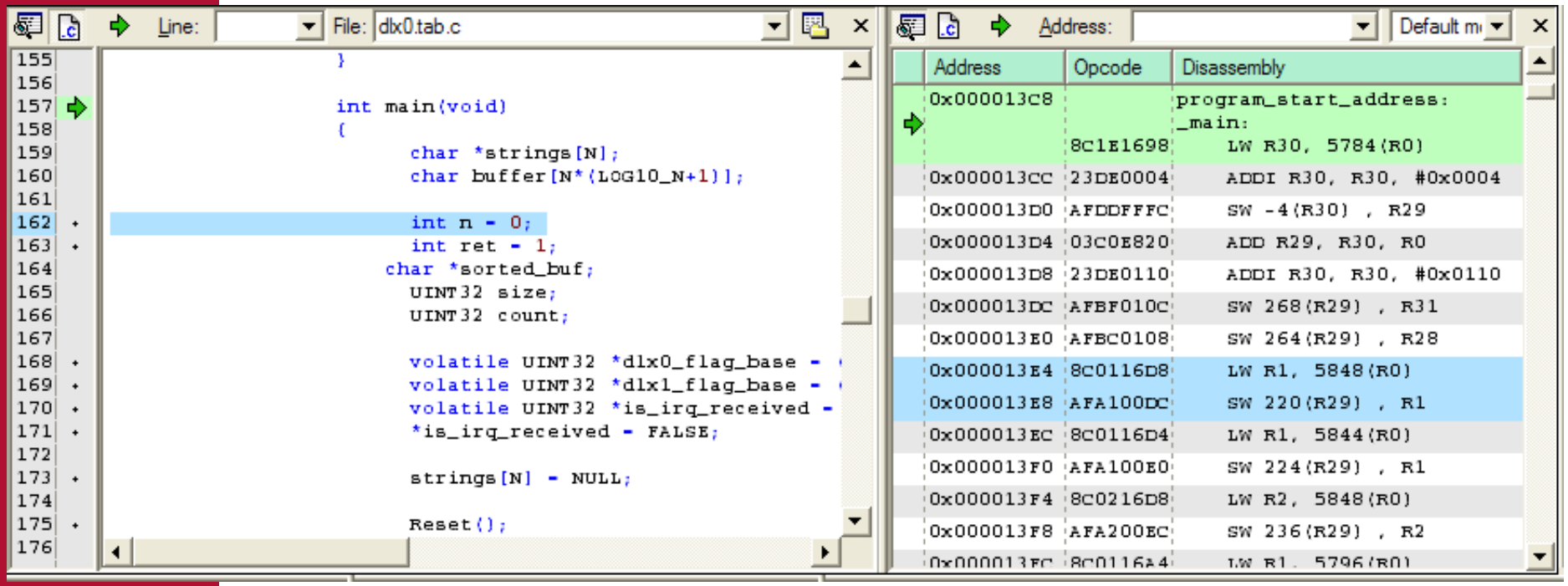
How can we program a computer to execute a specific task?

Usually a complex task is defined by an algorithm written in some "high-level language" (C, C++, C#, Java, Python, Matlab, R, etc).

A compiler (or interpreter) transforms the high level language into machine code (a sequence of numbers) that can be executed by a particular CPU after being loaded in memory

Università Ca' Foscari Venezia

From high level to machine code



Memory address

Instruction code

More about memory

Von Neumann architecture is particularly interesting because of its flexibility:

- Memory contains just numbers. The meaning of those numbers depends on the context. (Can be data used by the program or the instructions itself)
- Since a program is stored in the main memory, we can write a program that modify itself (or other programs) during the execution (not so common nowadays for security reasons..)

In practice, how can the memory store our numbers?

More about memory

Many different memory technologies (SRAM,DRAM,etc)

All based on the same idea: The presence or absence of electric charge defines the memory content.

To increase the signal-to-noise-ratio, it is better to have just two states: presence or absence of charge

That's why, in almost all computers, numbers are expressed in base 2 instead of base 10. (Just 2 states instead of 10)

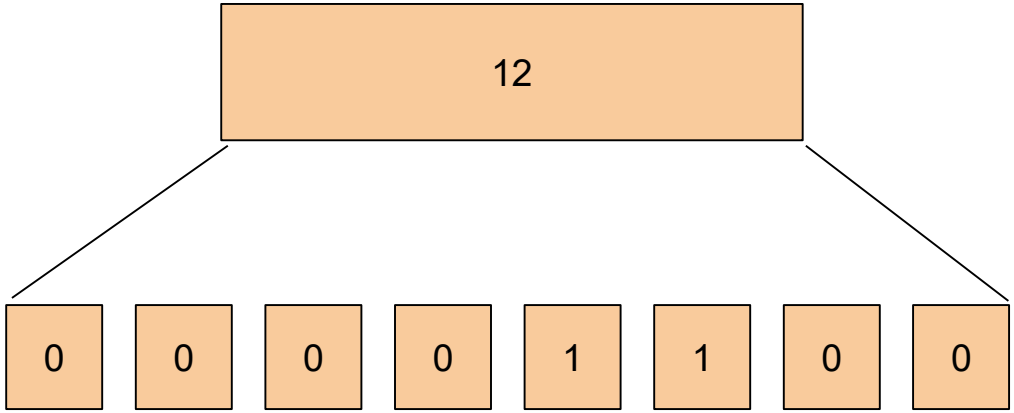
Easy to convert to and from the two bases.

More about memory

Usually, the presence of charge is denoted with number "1" and absence with number "0".

The two basic unit of information (0 and 1) are called bits

Each memory cell is usually grouped in units of 8 bits, called bytes.



Multiples of bytes

Value	Symbol	Name
1000	k	Kilo bytes
1000^2	M	Mega bytes (One million of bytes)
1000^3	G	Giga bytes (One billion of bytes)
1000^4	T	Tera bytes (One trillion of bytes)
1000^5	P	Peta bytes

Volatile/Non volatile memory

The memory discussed so far is called “volatile” since its value get lost when the power is switched off (no charge remain stored in it)

Non-volatile memory is designed to keep its data even without any external power

Advantages:

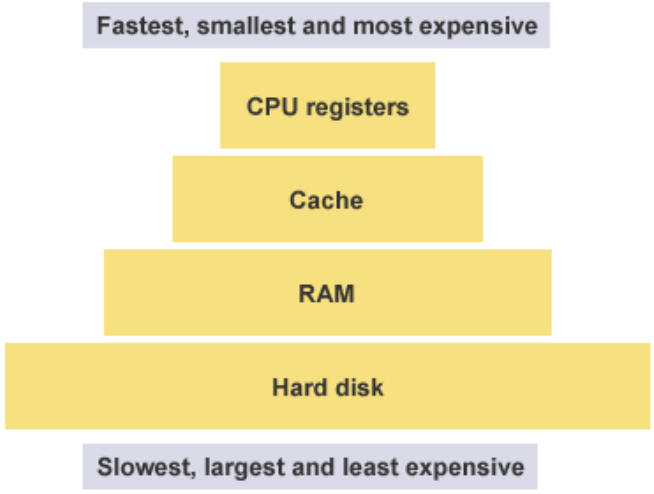
- Data can be stored permanently in it

Disadvantages:

- Slow access time
- Random access may be slower

Volatile/Non volatile memory

The closer a memory type is to the CPU, the quicker the CPU can access the instructions and execute them. However, the closer it is to the CPU the **smaller** and **more expensive** it is. Each type of memory is limited by their **speed**, **size**, **cost** and **position** in relation to the CPU.



Storing complex data

If the memory can only contain numbers (integers), how can we store text, real numbers, images, sounds and videos?

We need some predefined rules (or conventions, or standards) to encode a specific type of data into a sequence of numbers.

Data that are analogue by nature (ex an image, a sound, etc) must first be digitized.

> More on that in the following lessons

Università Ca' Foscari Venezia

Representing text in computers

The simplest type of encoding to represent text in computers is the American Standard Code for Information Interchange: ASCII (Pronounced ASS-kee).

ASCII specifies a correspondence between digital bit patterns and character symbols (1 byte per symbol)

Composed by:

- control characters (which do not represent printable characters but rather control their visualization). Example: line feed, carriage return, ESC, Tab
- 95 printable characters

Università Ca' Foscari Venezia

ASCII character encoding

ASCII control characters

DEC	HEX	Símbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

ASCII printable characters

DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

theASCIIcode.com.ar

ASCII character encoding

ASCII encodes only 95 selected printable characters (94 glyphs and one space), which include the English alphabet, digits, and 31 punctuation marks/symbols

Only the English alphabet is completely represented. If accented letters are replaced by two-character approximations, ascii can partially accommodate also European languages

Not all the possible 8-bit combinations are used. Some ISO standard **extensions** have been proposed to include characters sufficient for most common Western European languages

Università Ca' Foscari Venezia

ISO 8859-1 (Latin 1)

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F
1-	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
2-	0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3-	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4-	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5-	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
6-	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7-	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	007F
8-	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	008A	008B	008C	008D	008E	008F
9-	0090	0091	0092	0093	0094	0095	0096	0097	0098	0099	009A	009B	009C	009D	009E	009F
A-	00A0	¡ 00A1	¢ 00A2	£ 00A3	¤ 00A4	¥ 00A5	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	- 00AD	® 00AE	¯ 00AF
B-	° 00B0	± 00B1	2 00B2	3 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	1 00B9	º 00BA	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	¿ 00BF
C-	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF
D-	Ð 00D0	Ñ 00D1	Ò 00D2	Ó 00D3	Ô 00D4	Õ 00D5	Ö 00D6	× 00D7	Ø 00D8	Ù 00D9	Ú 00DA	Û 00DB	Ü 00DC	Ý 00DD	Þ 00DE	ß 00DF
E-	à 00E0	á 00E1	â 00E2	ã 00E3	ä 00E4	å 00E5	æ 00E6	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF
F-	ð 00F0	ñ 00F1	ò 00F2	ó 00F3	ô 00F4	õ 00F5	ö 00F6	÷ 00F7	ø 00F8	ù 00F9	ú 00FA	û 00FB	ü 00FC	ý 00FD	þ 00FE	ÿ 00FF

Università Ca' Foscari Venezia

ISO 8859-15 (Latin 9)

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F
1-	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
2-	0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3-	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4-	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5-	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
6-	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7-	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	007F
8-	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	008A	008B	008C	008D	008E	008F
9-	0090	0091	0092	0093	0094	0095	0096	0097	0098	0099	009A	009B	009C	009D	009E	009F
A-	00A0	¡ 00A1	¢ 00A2	£ 00A3	€ 20AC	¥ 00A5	Š 0160	§ 00A7	š 0161	© 00A9	ª 00AA	« 00AB	¬ 00AC	- 00AD	® 00AE	¯ 02C9
B-	° 00B0	± 00B1	2 00B2	3 00B3	Ž 017D	µ 00B5	¶ 00B6	· 00B7	ž 017E	1 00B9	º 00BA	» 00BB	Œ 0152	œ 0153	Ÿ 0178	¿ 00BF
C-	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF
D-	ð 00F0	Ñ 00D1	Ò 00D2	Ó 00D3	Ô 00D4	Õ 00D5	Ö 00D6	× 00D7	Ø 00D8	Ù 00D9	Ú 00DA	Û 00DB	Ü 00DC	Ý 00DD	Þ 00DE	ß 00DF
E-	à 00E0	á 00E1	â 00E2	ã 00E3	ä 00E4	å 00E5	æ 00E6	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF
F-	ð 00F0	ñ 00F1	ò 00F2	ó 00F3	ô 00F4	õ 00F5	ö 00F6	÷ 00F7	ø 00F8	ù 00F9	ú 00FA	û 00FB	ü 00FC	ý 00FD	þ 00FE	ÿ 00FF

Character set confusion

In order to correctly interpret and display text data (sequences of characters) that includes extended codes, hardware and software must use the specific extended ASCII encoding that applies to it.

- There is no general way to automatically infer the correct encoding (they are just numbers, right?)

Applying the wrong encoding causes irrational substitution of many or all extended characters in the text.

Since Microsoft Windows (using a superset of ISO 8859-1) is the dominant operating system for desktops today, that encoding is usually assumed

Università Ca' Foscari Venezia

ISO 8859-1 (Latin 1) vs Mac OS Roman

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-		0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F
1-	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
2-	0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3-	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4-	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5-	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
6-	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7-	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	007F
8-	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	008A	008B	008C	008D	008E	008F
9-	0090	0091	0092	0093	0094	0095	0096	0097	0098	0099	009A	009B	009C	009D	009E	009F
A-	00A0	¡ 00A1	¢ 00A2	£ 00A3	¤ 00A4	¥ 00A5	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	- 00AD	® 00AE	¯ 00AF
B-	° 00B0	± 00B1	2 00B2	3 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	1 00B9	º 00BA	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	¿ 00BF
C-	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF
D-	Ð 00D0	Ñ 00D1	Ò 00D2	Ó 00D3	Ô 00D4	Õ 00D5	Ö 00D6	× 00D7	Ø 00D8	Ù 00D9	Ú 00DA	Û 00DB	Ü 00DC	Ý 00DD	Þ 00DE	ß 00DF
E-	à 00E0	á 00E1	â 00E2	ã 00E3	ä 00E4	å 00E5	æ 00E6	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF
F-	ð 00F0	ñ 00F1	ò 00F2	ó 00F3	ô 00F4	õ 00F5	ö 00F6	÷ 00F7	ø 00F8	ù 00F9	ú 00FA	û 00FB	ü 00FC	ý 00FD	þ 00FE	ÿ 00FF

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	032	033 !	034 "	035 #	036 $	037 %	038 ^	039 &	040 *	041 (	042)	043 +	044 ,	045 -	046 .	047 /
3	048 0	049 1	050 2	051 3	052 4	053 5	054 6	055 7	056 8	057 9	058 :	059 ;	060 <	061 =	062 >	063 ?
4	064 @	065 A	066 B	067 C	068 D	069 E	070 F	071 G	072 H	073 I	074 J	075 K	076 L	077 M	078 N	079 O
5	080 P	081 Q	082 R	083 S	084 T	085 U	086 V	087 W	088 X	089 Y	090 Z	091 [	092 \	093]	094 ^	095 _
6	096 `	097 a	098 b	099 c	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
7	112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w	120 x	121 y	122 z	123 {	124 |	125 }	126 ~	127
8	128 Ä	129 Å	130 Ç	131 É	132 Ñ	133 Ö	134 Ü	135 á	136 à	137 â	138 ä	139 ã	140 å	141 ç	142 é	143 è
9	144 ê	145 ë	146 í	147 ì	148 î	149 ï	150 ñ	151 ó	152 ò	153 ô	154 ö	155 õ	156 ú	157 ù	158 û	159 ü
A	160 †	161 °	162 ¢	163 £	164 §	165 •	166 ¶	167 ß	168 ®	169 ©	170 ™	171 ´	172 ¨	173 ≠	174 Æ	175 Ø
B	176 ∞	177 ±	178 ≤	179 ≥	180 ¥	181 µ	182 ∂	183 ∑	184 ∏	185 π	186 ∫	187 º	188 ª	189 Ω	190 æ	191 ø
C	192 ¿	193 ¡	194 ¬	195 √	196 ƒ	197 ≈	198 Δ	199 «	200 »	201 …	202	203 À	204 Ã	205 Õ	206 Œ	207 œ
D	208 –	209 —	210 “	211 ”	212 ‘	213 ’	214 ÷	215 ◊	216 ÿ	217 Ÿ	218 /	219 €	220 ‹	221 ›	222 fi	223 fl
E	224 ‡	225 ·	226 ‚	227 „	228 ‰	229 Â	230 Ê	231 Á	232 Ë	233 È	234 Í	235 Î	236 Ï	237 Ì	238 Ó	239 Ô
F	240	241 Ò	242 Ú	243 Û	244 Ù	245 ı	246 ˆ	247 ˜	248 ¯	249 ˘	250 ˙	251 ˚	252 ¸	253 ˝	254 ˛	255 ˇ

Unicode character set

First attempts to create a universal character set able to represent all the possible symbols used nowadays (and in the past) dates back in 1987 with the work of Joe Becker from Xerox and Lee Collins and Mark Davis from Apple.

Such character set was called **Unicode**. The name was chosen to suggest a “unique, unified, universal encoding”

In its final version, Unicode contains 1.114.112 different elements (called code points)

Unicode character set

In practice, unicode contain every possible symbol in all the known languages, plus additional glyphs like modern emojis

https://unicode-table.com/en/

http://www.unicode.org/emoji/charts/full-emoji-list.html

Unfortunately, to encode all the 1.114.112 different code points at least 3 bytes are needed, thus substantially increasing the size of a written text

Multi-byte character encodings

To optimize space, we can exploit the fact that not all the code points are equally probable in a written text

Ex:

All the characters used in the ASCII encoding are far more common than the ones from ancient languages

To accommodate a vast set of different characters, modern encodings use **a variable number of bytes** to represent a single character.

The most common (at least in the WWW) character encoding used today is the **UTF-8**.

UTF-8 character encodings

UTF-8 encoding can use a variable number of bytes (1..4) for each character:

- The first 128 characters (US-ASCII) need **1 byte**. ie. an ASCII encoded text is also a valid UTF-8 text
- The next 1,920 characters need **2 bytes** to encode, which covers the remainder of almost all Latin-script alphabets, Greek, Cyrillic, Coptic, Armenian, etc.
- **3 bytes** are needed for characters in the rest of the Basic Multilingual Plane, which contains virtually all characters in common use
- **4 bytes** are needed for characters of, various historic scripts, mathematical symbols, and emoji

UTF-8 character encodings

Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Università Ca' Foscari Venezia

Representing real numbers

Similarly to text, real (fractional) numbers need some sort of encoding to be represented

A simple way to express fractional numbers is by using a fixed number of bits (**fixed point numbers**) for both the integer part and the fractional:

Example:

1 byte for the signed integer part, 1 byte for the fractional part

Fixed vs Floating point

- The major advantage of using a fixed-point representation is performance: A fixed-point number can be added/subtracted/multiplied like an integer number
- However… fixed-point numbers have a relatively limited range of values that they can represent.

Floating Point Notation is a way to represent very large or very small numbers precisely using **scientific notation** in binary. It provides a varying degrees of precision depending on the scale of the numbers used.

Scientific notation (in decimal)

When we use Scientific Notation in decimal (the form you're probably most familiar with), we write numbers in the following form:

+/- mantissa x $10^{exponent}$

Example: -12.3 x 10^{5}

Since the same number can be expressed with different powers of 10, we usually use the **normalized scientific notation** in which e choose an exponent so that the absolute value of the mantissa remains greater than or equal to 1 but less than the number base.

Example: -12.3 x 10^{5} -> -1.23 x 10^{6}

IEEE 754

Floating Point Representation is essentially the Normalized Scientific Notation applied to binary numbers

The **IEEE754** standard defines a number of different binary representations that can be used when storing Floating Point Numbers in memory:

- **Half Precision** – Uses 16-bits of storage in total.
- **Single Precision** – Uses 32-bits of storage in total.
- **Double Precision** – Uses 64-bits of storage in total.

Structure: $(-1)^{sign} \times mantissa \times 2^{exponent}$

IEEE 754

Single precision:

max positive value: $\approx 3.402823466 \times 10^{38}$

min positive value: $\approx 1.175494351 \times 10^{-38}$

Double precision:

max positive value: $\approx 1.7976931348623157 \cdot 10^{308}$

min positive value: $\approx 2.2250738585072014 \cdot 10^{-308}$

Floating point arithmetic is more complex than the fixed point counterpart. Operations are usually implemented with a dedicated circuitry in the CPU called **floating-point unit** (FPU)

Accuracy problems

Since we can only store numbers with finite precision (memory is finite) we can encounter erroneous and unexpected situations.

Some examples:

- The number π cannot be represented. The closest representable approximation (in double) is:
  3.14159265358979323846264338327 95

  This implies that:

  $\sin(\pi) = 0.1225 \times 10^{-15}$ (should be zero!)

  $\tan(\pi/2) = 16331239353195370.0$ (should be +inf)

Accuracy problems

Addition and multiplication are **commutative** but **not associative** due to rounding errors

Example using 7-digit significand decimal arithmetic:

a = 1234.567, b = 45.67834, c = 0.000400

(a+b) = 1280.24534 that rounds to: 1280.245

(a+b)+c = 1280.2454 that rounds to: **1280.245**

(b+c) = 45.67874

a+(b+c) = 1280.24574 that rounds to: **1280.246**

This is a typical problem that may rise when summing many small numbers. We can have different result depending on the summation algorithm

Operating System

Using a computer by simply reading and writing numbers on its memory would be extremely cumbersome for a human.

To manage its complexity, a special program called **operating system** controls the general operation of a computer, and provides an easy way for us to interact with computers and run applications.

Popular operating systems:

- Microsoft Windows
- Mac OS
- Linux
- Android

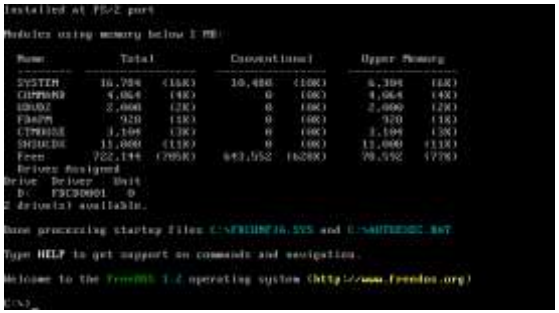
Key functions of an OS

- **Provides a user interface** so it is easy to interact with the computer
- **Manages the CPU**: runs applications and executes and cancels processes
- **Multi-tasks**: allows multiple applications to run at the same time
- **manages memory**: transfers programs into and out of memory, allocates free space between programs, and keeps track of memory usage
- **manages peripherals**: opens, closes and writes to peripheral devices such as storage attached to the computer
- **Organises data:** creates a file system to organise files and directories
- **Security**: provides security through user accounts and passwords. Manages privileges to restrict data access to specific users
- **Utilities**: provides tools for managing and organising hardware

Università Ca' Foscari Venezia

OS: user interface

The os provides an environment for the user to interact with the machine. The user interface (UI) can be either graphical or text-based:



Text based:

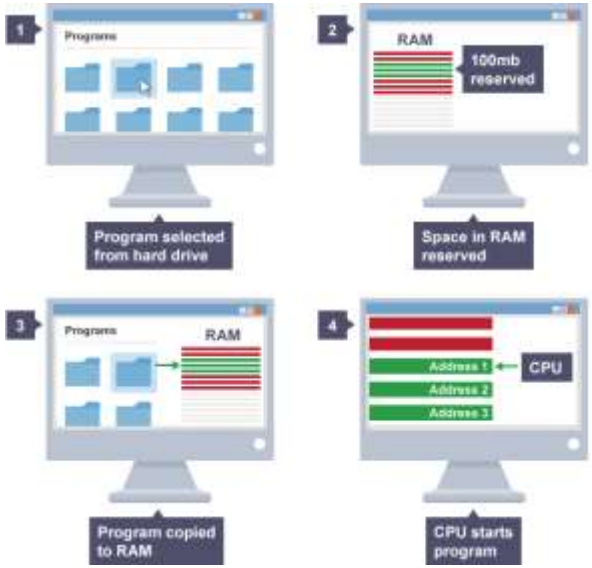
Implemented as a Command Line Interface (CLI). This is a text-only service with feedback from the OS appearing in text. Using a CLI requires knowledge of the commands available on a particular machine.

Graphical user interface (GUI)

provides an environment with tiles, icons and/or menus. Interactions usually happens with a combination of mouse and/or touchscreen

OS: CPU management

To execute a specific program, it must first be copied into main memory at a known location. One of the task of an OS is to load a desired program to memory and run it



Also, the OS manage the simultaneous execution of several programs at once with a component called **scheduler**.

OS: Memory management

Different processes simultaneously running must not interfere: They must use different parts of the main memory or use specific parts of the memory that can be used to communicate.

The OS decides:

- how memory is shared between processes
- what happens when there is not enough main memory to get the job done

In general the OS provides an isolated view of the underlying hardware to each program. **Each program can act as if it were the only program running on the computer with no conflicts on memory, CPU and other resources**

OS: Files

The operating system manages data into **files,** which are structured in specific ways in order to allow:

- faster access
- higher reliability
- better use of the available space.

files are organized into one-dimensional sequence of bytes, whose format (ie. the way the data is written) can be defined by its content and/or by its “extension” (ie. the suffix of the file name)

OS: Files

Together with its content, a file is commonly characterized by:

- A **name**, which provides a way to access its content
- An **extension** (the suffix of the name starting with .)
- A **size**, usually expressed in bytes, defining the amount of data contained in it
- **Attributes** like the creation date, last modification date, details about the owner, etc.

The way the information is stored into a file is up to how it is designed. Usually computer programs assist the user in creating files with specific formats

OS: File systems

The operating system manages the following operations on **files:**

- **Creation**
- **Changing of permissions/attributes**
- **Opening** (which makes the file contents available to a program)
- **Reading data**
- **Writing data**
- **Closing** (So that the content is no more available to the program that has it opened)

The specific way in which files are stored on a non volatile memory (disk,SSD,flash drive,etc) is called a **file system**, and enables files to have names and attributes.

OS: File systems

Most file systems are **hierarchical** and contain **directories** that contain **lists of other files (or other directories)**.

The OS organises **where** and **how** files are stored, deleted, read, found and repaired.

It detects errors such as missing disks or incorrect file names, and informs the user that errors have occurred.