

Computing Fundamentals

Salvatore Filippone

`salvatore.filippone@uniroma2.it`

2012–2013

Computer Science is the Science of Computers

Computer Science is the Science of Computers ?

Computer Science is the Science of Computers ?

Somewhat mistaken; it can be argued that the Science of (building) Computers is Electronics Engineering.

Computer Science is the Science of Computers ?

Somewhat mistaken; it can be argued that the Science of (building) Computers is Electronics Engineering. Better definition:

Computer Science is the discipline dealing with representation, storing, retrieval and processing of information by automated means

Actually Informatics would be a better name !



The Ancestors' Era:

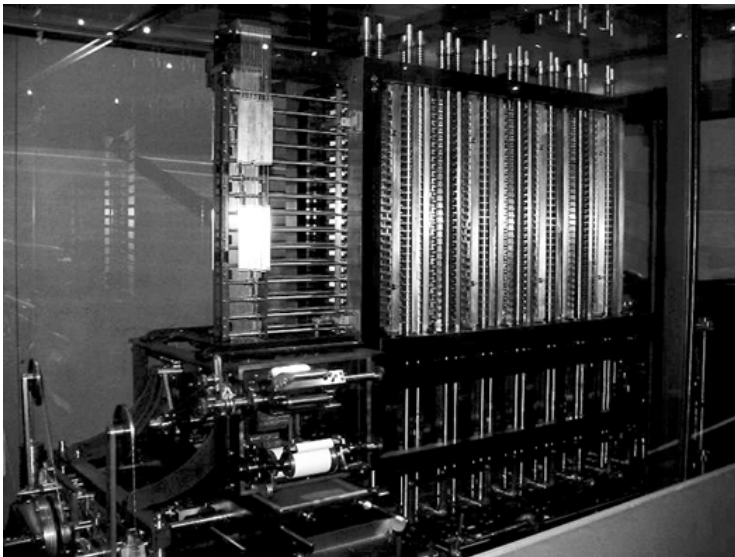
- The Antikythera mechanism
- Charles Babbage: the Analytical Engine;
- Hermann Hollerith: card collation machine for US census;



History of computing: Antikythera



History of computing: Babbage



History of computing: Hollerith

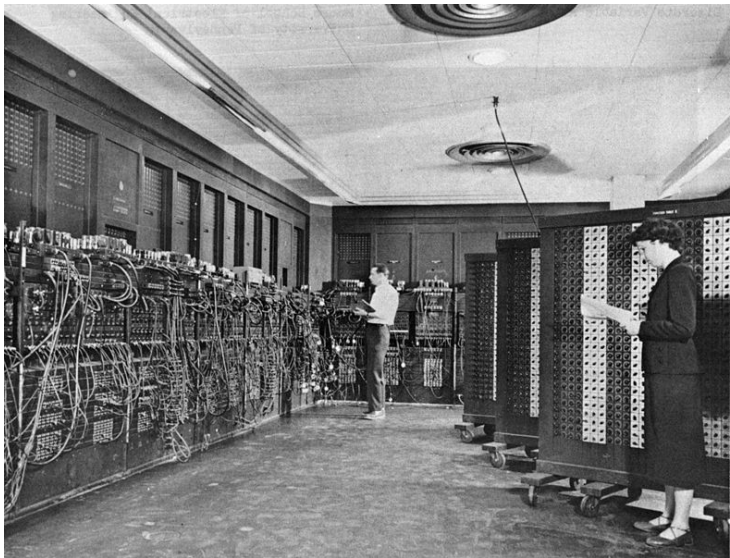




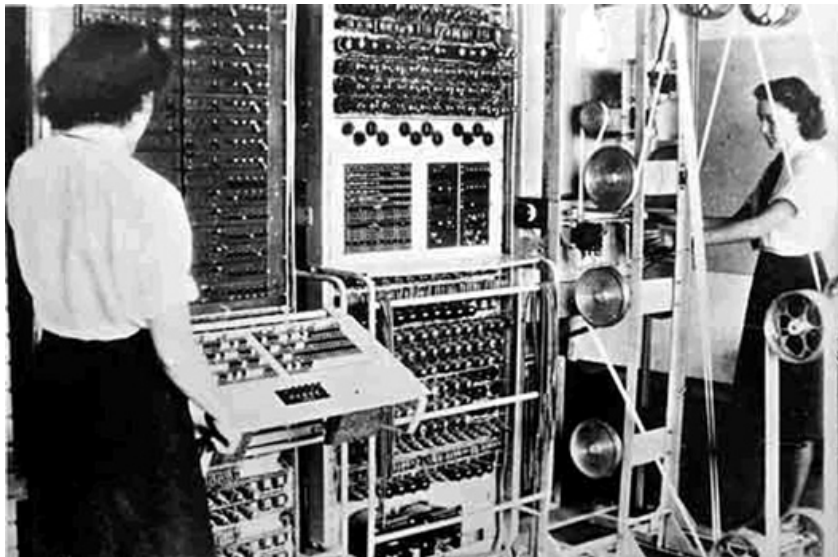
The Pioneer's Era

- ENIAC, used for army ballistic tables;
- Colossus, deciphering Nazi messages;

History of computing: ENIAC



History of computing: Colossus





The Mainframe Era: Computers become an indispensable business tool:

- IBM 360;
- Cray 1;
- Digital VAX;

History of computing: IBM 360



History of computing: Cray 1



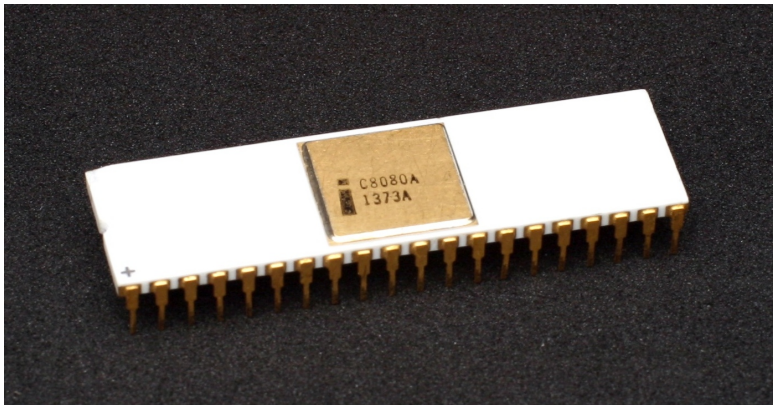




The Microprocessor Era:

- Intel 8080;
- Apple Macintosh;
- IBM PC;

History of computing: Intel 8080





History of computing: Apple Macintosh





Bundesarchiv, B 146 Bild-F077869-0042
Foto: Reineke, Engelbert | 6. April 1988



Today and Tomorrow:

- Supercomputers;
- GPUs;
- Mobile computing;



Titan, Oak Ridge National Laboratory: most powerful computer in the world.

History of computing: Jugene



Jülich Supercomputing Center, today largest in Europe

History of computing: Fermi



Fermi, CINECA: most powerful computer in Italy (for the first time in the top 10).





Information: a rigorous definition

The concept of “information” was defined rigorously by Claude Shannon in 1948.



Information: a rigorous definition

The concept of “information” was defined rigorously by Claude Shannon in 1948. Basic ideas:

- Setting: the transmission of a message;
- The message is composed of items from an *alphabet*;
- The alphabet has an *encoding*;
- Receiving a message changes a probability estimate;
- Independent messages should add their information.

Thus information should be a function of probability (of a symbol):



Information: a rigorous definition

The concept of “information” was defined rigorously by Claude Shannon in 1948. Basic ideas:

- Setting: the transmission of a message;
- The message is composed of items from an *alphabet*;
- The alphabet has an *encoding*;
- Receiving a message changes a probability estimate;
- Independent messages should add their information.

Thus information should be a function of probability (of a symbol):

$$I(\alpha) = -\log_b(p(\alpha))$$

If $b = 2$ the unit is a “bit”. Entropy: Average information per symbol

$$H(\mathcal{A}) = \sum_{\alpha \in \mathcal{A}} -p(\alpha) \log_b(p(\alpha))$$

Key idea: *Representation of data*



An example: the encoding of the Latin alphabet. Encoding one letter out of 26, the bits carried by a letter are:

$$B(\alpha) = \log_2(26) \approx 4.7$$

so we need at least 5 bits. ASCII encoding uses 7 (out of 8) bits:

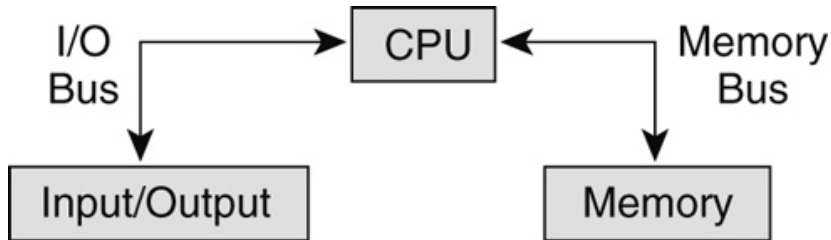
65	A	78	N	97	a	110	n
66	B	79	O	98	b	111	o
67	C	80	P	99	c	112	p
68	D	81	Q	100	d	113	q
69	E	82	R	101	e	114	r
70	F	83	S	102	f	115	s
71	G	84	T	103	g	116	t
72	H	85	U	104	h	117	u
73	I	86	V	105	i	118	v
74	J	87	W	106	j	119	w
75	K	88	X	107	k	120	x
76	L	89	Y	108	l	121	y
77	M	90	Z	109	m	122	z



Organization and interaction of the various components making up a computer

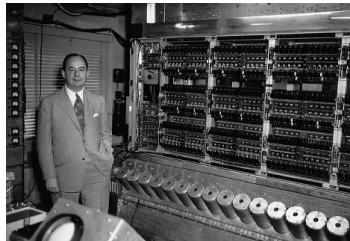
- Nomenclature originally introduced in the 60s (IBM 360)
- Basic idea: Von Neumann architecture;
- Evolution over time:
 - “Traditional” systems;
 - “Pipelined” computers;
 - Vector CPUs ;
 - Microprocessors;
 - RISC (Reduced Instruction Set Computer) CPUs;
 - SMPs (Symmetric MultiProcessor);
 - MPPs (Massively Parallel Processor).
 - Multi-core computers
 - GPU (Graphical Processing Units) accelerators;

CPU: Central Processing Unit.

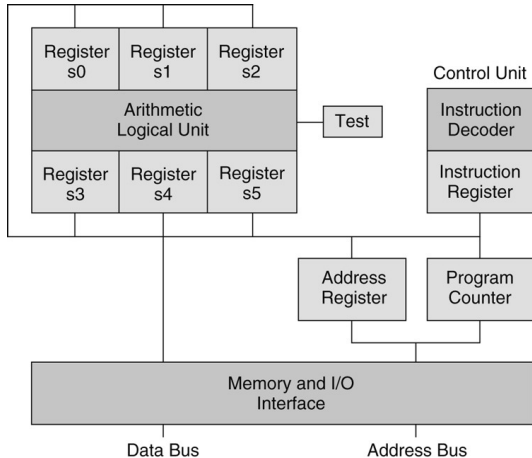


Central
to the Von Neumann architecture:

- Time is discretized;
- Simple operations are executed inside the CPU;
- The list of operations to be executed is stored in memory;
- The data are also in memory;
- The instructions of one program can be the data of another (compilers!).



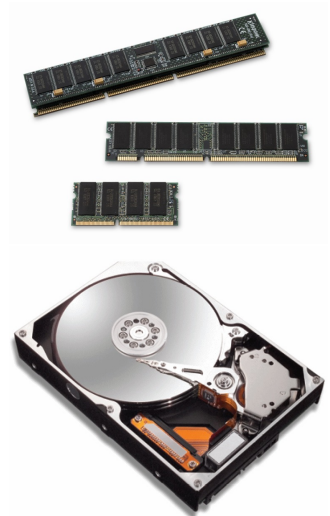
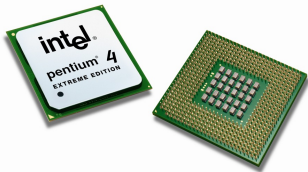
Von Neumann architecture





Classification:

- Handheld/mobile devices;
- Laptops;
- Desktop;
- Workstations;
- Mainframes;
- Supercomputers;





Central to computer science is the notion of **algorithm**:

A procedure for solving a problem, i.e. producing an answer given the data, with the following characteristics:

- ① *Finiteness;*
- ② *Definiteness; (no ambiguities)*
- ③ *Input;*
- ④ *Output;*
- ⑤ *Effectiveness (can be carried out).*

Note: by definition, an algorithm **always** answers in a **finite** time. Otherwise it's a *computational method*.



Oldest algorithm recorded in history is Euclid's algorithm.

Given m and n positive integers:

- 1 Divide m by n and note the remainder r ;
- 2 if $r = 0$, output n and stop;
- 3 Set $m \leftarrow n$, $n \leftarrow r$ and go back to step 1.

This algorithm is the efficient way to compute $GCD(m, n)$.

Exercise: prove that it terminates and works.



For any problem (provided it is specified precisely) there will be an algorithm. Right ?



For any problem (provided it is specified precisely) there will be an algorithm. Right ?

NO!

Related to Gödel's incompleteness theorem, we have the Turing's Halt theorem:

There exist problems for which no algorithm can possibly be devised.

To clarify, there can be procedures that solve some problem instances, but on some instances they will not terminate in finite time.



For any problem (provided it is specified precisely) there will be an algorithm. Right ?

NO!

Related to Gödel's incompleteness theorem, we have the Turing's Halt theorem:

There exist problems for which no algorithm can possibly be devised.

To clarify, there can be procedures that solve some problem instances, but on some instances they will not terminate in finite time. Ironically, one of the impossible problems is:

Given a student's programming project, decide if it will stop on its input or it will go in an endless loop

No algorithm can fully solve this! (But many cases are recognizable)



Going back to the *finiteness* property:

*An algorithm should be VERY finite, not just finite.
(D. Knuth)*

Let us make this precise. Algorithms are characterized by

Time complexity: $T(n)$ is $O(f(n))$ if we can find $f(n)$, C and n_0 such that on an input of size n the time to completion is

$$T(n) \leq C \cdot f(n) \quad \text{for all } n > n_0$$

Space complexity: a similar upper bound on the amount of memory employed

Algorithms are typically considered “tractable” if their complexity is a polynomial in n (but this may still leave room for improvement)



Consider the game of chess; in 1950 Shannon wrote a paper about implementing a computer program to play. In theory the program can just look at all possible table configurations (aka positions, and they are finite). Right ?



Consider the game of chess; in 1950 Shannon wrote a paper about implementing a computer program to play. In theory the program can just look at all possible table configurations (aka positions, and they are finite). Right ?

- ① Average number of legal moves per position: 30;
- ② Thus with a move for White followed by a move for Black we have: 10^3 ;
- ③ Average game length: 40 moves;

As a consequences we need to look at

$$10^{120}$$

different positions.



Consider the game of chess; in 1950 Shannon wrote a paper about implementing a computer program to play. In theory the program can just look at all possible table configurations (aka positions, and they are finite). Right ?

- ① Average number of legal moves per position: 30;
- ② Thus with a move for White followed by a move for Black we have: 10^3 ;
- ③ Average game length: 40 moves;

As a consequences we need to look at

$$10^{120}$$

different positions. To put in perspective:

- Number of atoms in the universe: 10^{80}
- Size of the universe in electron diameters: 10^{39}



The Discrete Fourier Transform (of size N): it is an essential tool of communication technology:

$$F(k) = \sum_{0 \leq j < N} \omega_N^{kj} f(j), \quad \omega_N^{kj} = e^{2\pi i \frac{jk}{N}} \quad 0 \leq k < N$$



The Discrete Fourier Transform (of size N): it is an essential tool of communication technology:

$$F(k) = \sum_{0 \leq j < N} \omega_N^{kj} f(j), \quad \omega_N^{kj} = e^{2\pi i \frac{jk}{N}} \quad 0 \leq k < N$$

This is a (complex) matrix-vector product, so the cost is $8N^2$,



Algorithm

The Discrete Fourier Transform (of size N): it is an essential tool of communication technology:

$$F(k) = \sum_{0 \leq j < N} \omega_N^{kj} f(j), \quad \omega_N^{kj} = e^{2\pi i \frac{jk}{N}} \quad 0 \leq k < N$$

This is a (complex) matrix-vector product, so the cost is $8N^2$, but in 1965 Cooley and Tukey (re)discovered a way to do it in $5N \log(N)$!

Size	DFT	FFT
10	800	166
100	80000	3321.93
1000	8e+06	49828.9
5000	4e+08	307193
10000	8e+08	664386
50000	4e+10	3.90241e+06
100000	8e+10	8.30482e+06
500000	4e+12	4.73289e+07
1000000	8e+12	9.96578e+07

Things that would not exist without the FFT include: Satellite communications, mobile phones, CAT, PET, VOIP, CD, JPEG, MPEG DVD, DVTB...



From the above discussion, we now know what a computer scientist does (most of the time):

- ➊ Study the representation of the problem data;
- ➋ Figure out if for a given problem there is an algorithm;
- ➌ Figure out if there are more algorithms, perhaps with different efficiencies;
- ➍ Find a good implementation of a given algorithm;



To get a computer do what you want you have to *talk* to it.

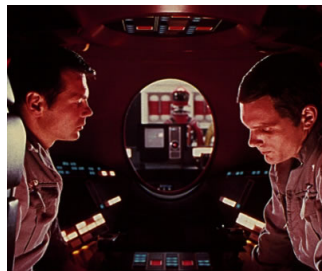




To get a computer do what you want you have to *talk* to it.

Deep down computers are exceedingly stupid, they only understand 0s and 1s. But: with zeros and ones you can build pretty amazing things. Anyway, when you're in front of a computer you have to

- Talk to them in a language that's *precise* and *unambiguous*;
- Employ a *translator* (unless you can talk 011100110100111010111...)
- Every now and then, get down to their level, and see why they are (mis)behaving





Languages have three levels of correctness:

- 1 Lexical (**Ash nazg durbatulûk**);
- 2 Syntactic (**fly airplane an ? does**);
- 3 Semantics (**The airplane is reading a nice book**).

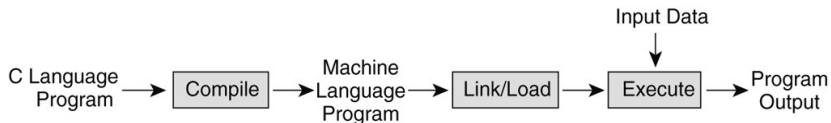
Computer translators can be

- 1 Compilers;
- 2 Interpreters.

They can help you with lexical and syntactic analysis, but semantics is a lot harder.

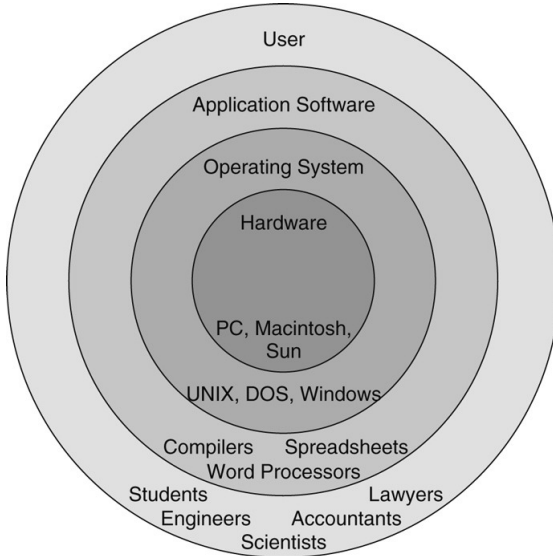


The translation (compile) chain:





What sits between you and the computer?





Fortran:

```
program hello
  write(*,*) 'Hello world'
end program hello
```

C:

```
#include <stdio.h>
void main()
{
    printf("Hello_world\n");
}
```

C++

```
#include <stdio>
main()
{
    cout << "Hello_world_"<<endl;
}
```

Matlab:

```
fprintf('Hello_world\n');
```

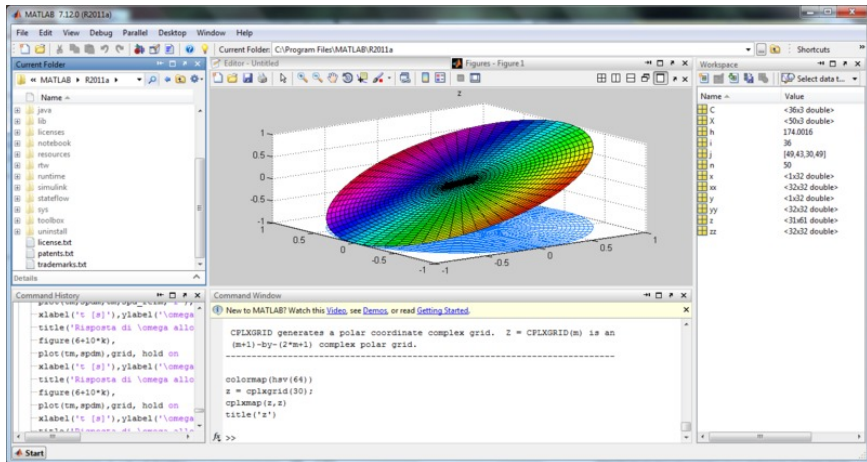


MATLAB (matrix laboratory) is a numerical computing environment.

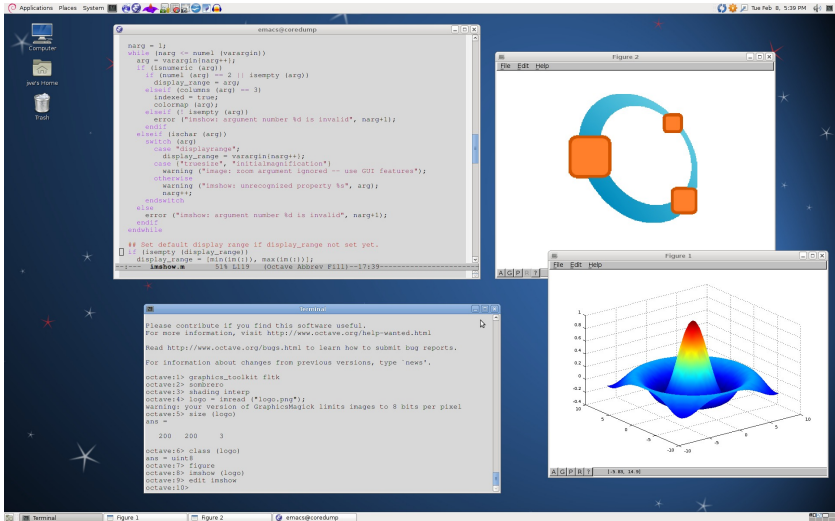
<http://www.mathworks.it/> Comprises:

- An interpreted environment;
- A C-like programming syntax;
- A Fortran-90 like array language;
- Extensive access to widespread libraries;
- Graphical capabilities;
- Interfacing with other languages.

Very important if you are using one the toolboxes (e.g. Control systems).



Octave <http://www.gnu.org/software/octave/>: a free software environment, largely compatible with Matlab





For the purposes of this course, Octave should be sufficient. You may also consider getting a student license of Matlab.

My email address (again):

`salvatore.filippone@uniroma2.it`

Exam:

- Written exam only;
- Set of questions;
- Small programming assignment.

Receiving times: to be announced later.



Textbook:

D. Smith: Engineering Computation with MATLAB, Pearson;

Other material

- ① S. De Marchi: Appunti di Calcolo Numerico, Esculapio;
- ② A. Quarteroni, R. Sacco, F. Salero *Matematica Numerica*, Springer.
- ③ D. Knuth: The Art of Computer Programming, Addison-Wesley;