Unit 1

Introduction

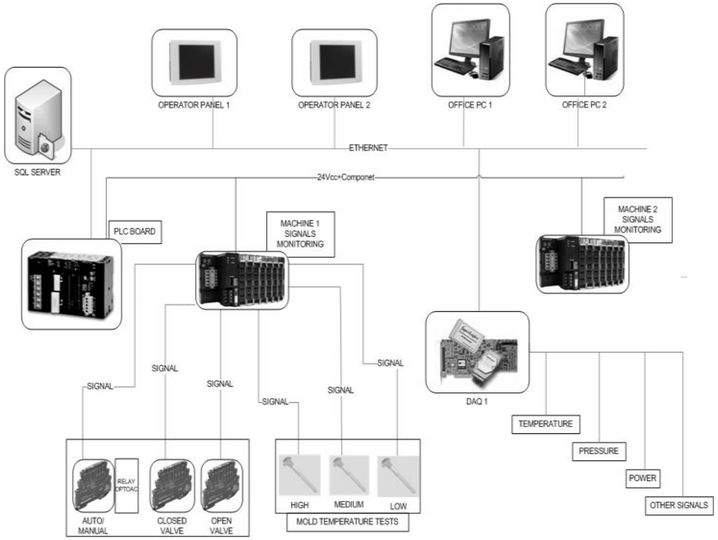
Computing Basics

University of Oviedo

Table of contents

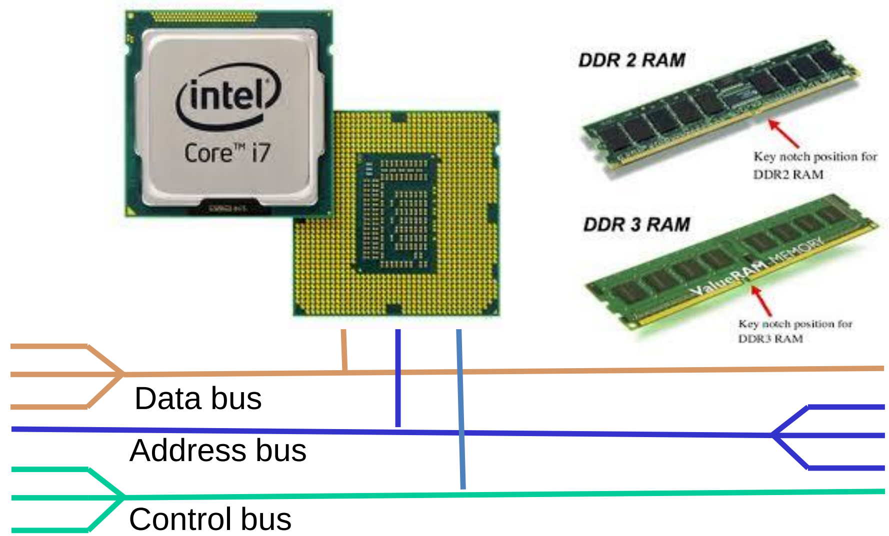
1.1 Overview of computer science

- **Informática**: **Infor**~~mación~~ + ~~auto~~**mática** (French origin)
  - According to the RAE: A set of scientific knowledge and techniques that enables the automatic processing of information by using computers.
    - Science that deals with the acquisition, storage, representation, processing and transmission of information using machines known as computers.
  - Information in companies is managed using different software tools, including
    - Different operating systems
    - Networking
    - Databases
    - User applications
    - Applications for the control and management of the production.



1.2 Structure and operation of a computer

- What is a computer?
  - It is a machine that collects data, processes them, and produces an output.
  - It processes information much quicker than the human brain.
  - It is programmable.
- How are these actions carried out?
  - Architecture and block diagram
    - Unique address space
  - Program
    - Algorithm



Given a second order polynomial p(x)=a·x²+b·x+c, evaluate p(x) for all numbers x other than zero provided by the user.

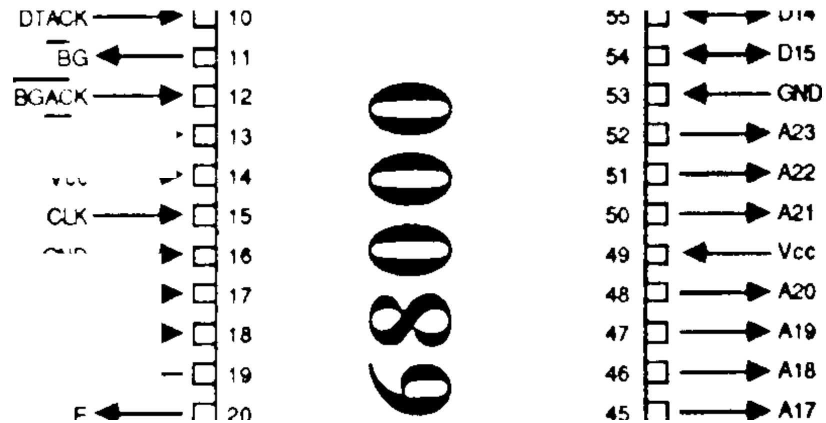
Algorithm:

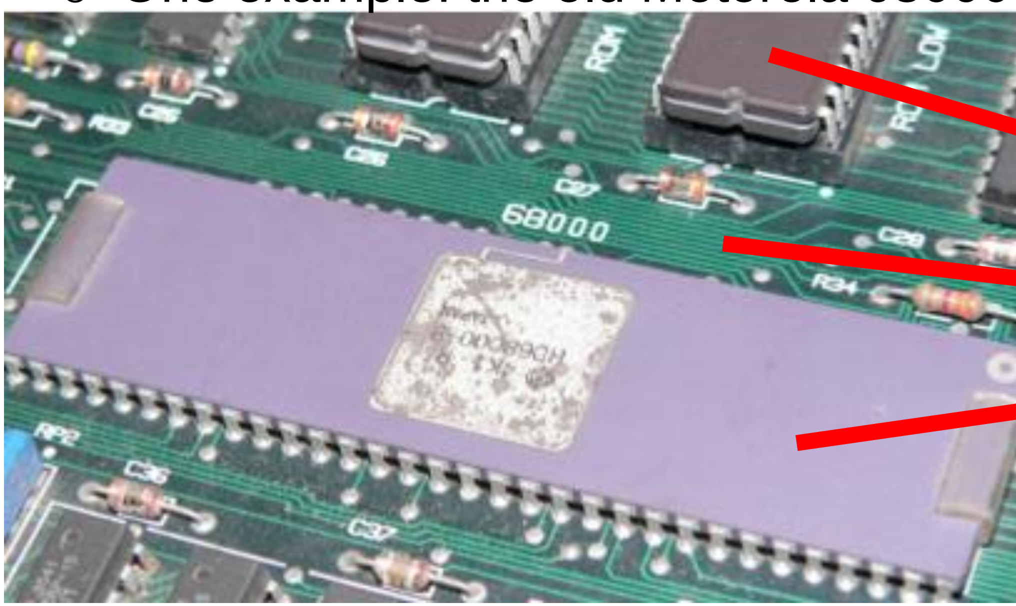
Ask for the coefficients
Ask for x
While x!=0
    Calculate p(x)
    Show p(x)
    Ask for x

```
print("Evaluating polynomials")
print("p(x)=a*x*x+b*x+c")
a = float(input("Type a: "))
b = float(input("Type b: "))
c = float(input("Type c: "))
x = float(input("Type x: "))
while x!=0:
    p = a*x*x+b*x+c
    print("p(",x,")=",p)
    x = float(input("Type x: "))
print("End of the program")
```

1.2 Structure and operation of a computer

- Information on a computer
  - Electrical signals are exchanged through the buses
  - Electrical signals are discrete (ALL / NOTHING)!
  - One example: the old Motorola 68000

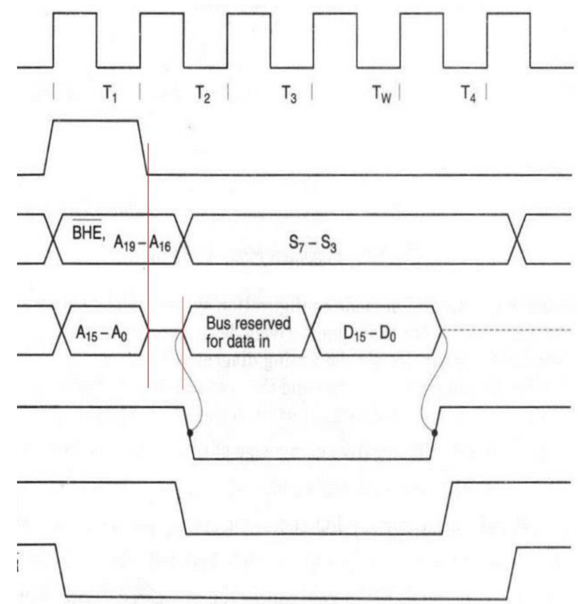




Memory

Buses

CPU



1.3 Representation of information in a computer

- In order to process information in a computer, it must be represented or encoded
  - Computers only work with 1/0 → **Binary information!**
- Binary system
  - Base 2
    - Digits {0, 1}
    - Bit → binary digit (the basic information unit)
  - Groupings
    - 8 bits → 1 **Byte**
    - 10^3 bytes → k**B**
    - 2^{10} bits → k**b**

- Decimal system
  - Base 10
    - Digits {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  - Decomposing a number into digits (and weights):

138.32 ⟷ 2×10^{-2} + 3×10^{-1} + 8×10^{0} + 3×10^{1} + 1×10^{2}

2 1 0 -1 -2

$$
\begin{array}{r}
2\text{x}10^{-2} \\
3\text{x}10^{-1} \\
8\text{x}10^{0} \\
3\text{x}10^{1} \\
1\text{x}10^{2} \\
\hline
138.32
\end{array}
$$

Broadly speaking, given a number **abcd.ef** represented in **base B**, its conversion to the decimal system is done by successive multiplications by powers of the base B, as follows:

abc.d → $a*B^2 + b*B^1 + c*B^0 + d*B^{-1}$

2 1 0 -1

1.3 Representation of information in a computer

- Hexadecimal system
  - Base 16
    - Digits {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
  - Possible values: integers in the range [0, 15]
  - Direct match between hexadecimal and binary:
    - $16=2^4$ → all the hex digits can be represented with 4 bits
  - Conversion between hexadecimal and decimal
    - The same as in the binary case

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

Use this table to easily convert from binary to hexadecimal.

Positive integer number representation

FROM BASE B TO BASE 10	FROM BASE 10 to BASE B
Broadly speaking, given a number **abcd.ef** represented in **base B**, its conversion to the **decimal** system is done by successive multiplications by powers of the base B, as follows:	Broadly speaking, given a number **abcd.ef** represented in **base 10**, its conversion to **base B** is done by successive divisions by the base B, as follows:

$$\underset{2}{a}\underset{1}{b}\underset{0}{c}.\underset{-1}{d} \rightarrow a*B^{2}+b*B^{1}+c*B^{0}+d*B^{-1}$$

25/2=12, remainder 1
12/2=6, remainder 0
6/2=3, remainder 0
3/2=1, remainder 1
1/2=0, remainder 1

$\rightarrow 11001_2$

- We are going to use three different systems:

	Decimal	Binary	Hexadecimal
	Used by humans	Used by the CPU	Intermediate representation
Base	B=10	B=2	B=16
Digits	0 1 2 3 4 5 6 7 8 9	0 1	0 1 2 3 4 5 6 7 8 9 A B C D E F

Positive integer number representation

- From decimal to binary
  - 18 → 18/2=9 r=0; 9/2=4 r=1; 4/2=2 r=0; 2/2=1 r=0; 1/2=0 r=1 → 10010
  - 29 → 29/2=14 r=1; 14/2=7 r=0; 7/2=3 r=1; 3/2=1 r=1; 1/2=0 r=1 → 11101
- From binary to decimal
  - 11010 → $1*2^4+1*2^3+0*2^2+1*2^1+0*2^0=26$ → 26
  - 0110111 → $0*2^6+1*2^5+1*2^4+0*2^3+1*2^2+1*2^1+1*2^0=55$
- From binary to hexadecimal (8 bits)
  - 0011010 → 0001 1010 → 0x1A
  - 0110111 → 0011 0111 → 0x37
- From hexadecimal to decimal
  - 1A2F → $1*16^3+10*16^2+2*16^1+15*16^0=6703$ → 6703
- From hexadecimal to binary
  - 1A2F → 0001 1010 0010 1111

Signed integer and real numbers

- **Signed integer numbers**
  - As in the unsigned case, but using the most significant bit for the sign +(0) o –(1).
  - In two's complement

Example to represent -50:

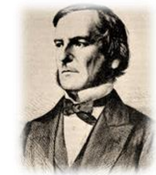
1.- We represent 50 in binary:
  50 → 00110010

2.- We get the two's complement of 50 (we invert the number and add 1)
  00110010 → 11001101 → **11001110**

- **Real numbers (floating point representation)**
  - Standard IEEE 754
  - Use of scientific notation plus some simplification rules
  - Using 32 or 64 bits

Sign	Exponent	Mantissa
1 bit	8 or 11 bits	23 or 52 bits

Characters and boolean data representation



- **Characters**
  - Alphanumeric, punctuation marks, etc.
  - Encoding table
    - ASCII & extended ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space
1	1	001	SOH (start of heading)	33	21	041	!	!
2	2	002	STX (start of text)	34	22	042	"	"
3	3	003	ETX (end of text)	35	23	043	#	#
4	4	004	EOT (end of transmission)	36	24	044	$	$
5	5	005	ENQ (enquiry)	37	25	045	%	%
6	6	006	ACK (acknowledge)	38	26	046	&	&
7	7	007	BEL (bell)	39	27	047	'	'
8	8	010	BS (backspace)	40	28	050	(	(
9	9	011	TAB (horizontal tab)	41	29	051	)	)
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*
11	B	013	VT (vertical tab)	43	2B	053	+	+
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,
13	D	015	CR (carriage return)	45	2D	055	-	-
14	E	016	SO (shift out)	46	2E	056	.	.

- **Boolean data**
  - Possible values **True** or **False** (1 or 0)
  - Named in honor of George Boole (1815-1864), father of the propositional logic.
  - Examples that produce a boolean value:
    - 3 > 5 → False
    - 14 >= 2 → True
    - 3 is an integer → True
  - A boolean value determines the minimum possible amount of information: 1 bit

System of units in computer science

- According to the International System of Units (SI):

Kilobit (kb)	10^3 bits
Megabit (Mb)	10^6 bits = 1000 kb
Gigabit (Gb)	10^9 bits = 1000 Mb
Terabit (Tb)	10^{12} bits = 1000 Gb
Petabit (Pb)	10^{15} bits = 1000 Tb

Multiples of a **bit**

Kilobyte (kB)	10^3 bytes
Megabyte (MB)	10^6 bytes = 1000 kB
Gigabyte (GB)	10^9 bytes = 1000 MB
Terabyte (TB)	10^{12} bytes = 1000 GB
Petabyte (PB)	10^{15} bytes = 1000 TB

Multiples of a **byte**

1.4 Propositional logic

- Logic is what will allow us to take decisions in a computer program
- It models human reasoning in a way suitable to be understood by a machine
- It is vital to handle the main ideas behind logic to be able to write programs correctly
- Real life example: Airplane landing
  - When a plane lands it needs to reduce speed before wheel brakes can be applied
  - Depending on the runway length and other parameters, it is required to use the **reverse thrust**
  - Reverse thrust can only be used when the plane "knows" that all the whole landing gear has touched ground

"**IF** all_wheels_are_on_the_ground **AND** pilot_engages_reverse
THEN engage_reverse_thrust"

Boolean values and operators

- A **truth, logical or boolean value** can only be **True** (T) or **False** (F)
- A logical expression produces a truth value, just like an arithmetic expression returns a number
- Logical expressions are build up using two types of operators
  - Relational operators: ==, !=, <=, >, etc.
  - Boolean connectives: AND, OR, NOT

Semantic rules

- Being P1 and P2 two parts of a logical expression, known as propositions, the following semantic rules for the connectives apply:

P1	P2	P1 ∧ P2
T	T	T
T	F	F
F	T	F
F	F	F

AND

P1	P2	P1 ∨ P2
T	T	T
T	F	T
F	T	T
F	F	F

OR

P1	¬ P1
T	F
F	T

NOT

TRUTH TABLES FOR THE THREE CONNECTIVES

Evaluation of logical expressions

- Evaluation of **(x>3)** ∧ ¬ **(y<=0)** with {x= 4, y= 5}

	(x>3)	∧	¬	(y<=0)
1º	(4>3)	∧	¬	(5<=0)
2º	T			F
3º	T		T	
4º		T		

- Evaluation of **((x>3)** ∧ ¬**(y<=0))** ∨ **A** with {x= 4, y= 5, A=T}

	((x>3)	∧	¬	(y<=0))	∨	A
1º	((4>3)	∧	¬	(5<=0))	∨	A
2º	T			F		T
3º	T		T			T
4º		T				T
5º					T	

Logical laws

Logical laws are very important equivalences between propositions; i.e., propositions with an identical truth table. Being P, Q and R propositions:

- **Associative**
  $(P \vee Q) \vee R = P \vee (Q \vee R)$
  $(P \wedge Q) \wedge R = P \wedge (Q \wedge R)$
- **Conmutative**
  $P \vee Q = Q \vee P$
  $P \wedge Q = Q \wedge P$
- **Distributive**
  $(P \vee Q) \wedge R = (P \wedge R) \vee (Q \wedge R)$
  $(P \wedge Q) \vee R = (P \vee R) \wedge (Q \vee R)$
- **Idempotent**
  $\neg\neg P = P$
- **Complements**
  $P \vee \neg P = \mathbf{T}$,
  $P \wedge \neg P = \mathbf{F}$
- **T** and **F**
  $\mathbf{T} \vee P = \mathbf{T}$ $\quad$ $\mathbf{T} \wedge P = P$
  $\mathbf{F} \vee P = P$ $\quad$ $\mathbf{F} \wedge P = \mathbf{F}$
- ***De Morgan***
  $\neg(P \vee Q) = \neg P \wedge \neg Q$
  $\neg(P \wedge Q) = \neg P \vee \neg Q$

Syntactic simplifications

- Propositions can be simplified using the logical laws. Two examples:

$\neg\ \mathbf{((\neg A \wedge B) \vee \neg(B \vee \neg A))}$ =
$\neg\ (\neg A \wedge B) \wedge \neg \neg (B \vee \neg A)$ =
$(\neg \neg A \vee \neg B) \wedge (B \vee \neg A)$ =
$(A \vee \neg B) \wedge (B \vee \neg A)$

$\mathbf{\neg\ ((x<=3 \vee x>=5) \vee (y<3 \vee y>=9))}$ **=**
$\neg\ (x<=3 \vee x>=5) \wedge \neg (y<3 \vee y>=9)$ =
$(\neg (x<=3) \wedge \neg (x>=5)) \wedge (\neg (y<3) \wedge \neg (y>=9))$ =
$(x>3 \wedge x<5) \wedge (y>=3 \wedge y<9)$

Using interval notation: $x \in (3,5)$ e $y \in [3, 9)$

Translations

- The human language can be translated into logical expressions. Two examples:

- It is white and it is not white

Being A = to be white, the translation is: $A \wedge \neg A = F$

- Being x and y integer numbers, at least one of them is greater or equal to zero and the first one is not greater than the second one

$((x>=0) \vee (y>=0)) \wedge \neg (x>y) = ((x>=0) \vee (y>=0)) \wedge (x<=y)$

Exercises (taken from an exam)

- Consider the expression “either the number *x* is odd, or even and greater or equal to 10”. Write a boolean expression in Python using a variable *x* with the same meaning as in the previous sentence.

  (Note that the expression **a%b** in Python returns the remainder when a is divided by b)

```
(x%2 == 1) or ((x%2 == 0) and (x>=10))
```

- Negate the previous expression and then apply the De Morgan’s laws, writing the result without any negations. What would be the result for x=6?

```
not ((x%2 == 1) or ((x%2 == 0) and (x>=10))) =
not (x%2 == 1) and not ((x%2 == 0) and (x>=10)) =
not (x%2 == 1) and (not(x%2 == 0) or not(x>=10)) =
(x%2 != 1) and ((x%2 != 0) or (x<10))
```

The result for x=6 is True