Introduction to Digital Systems

Part I (4 lectures)

2020/2021

Introduction

Number Systems and Codes

Combinational Logic Design Principles

Arnaldo Oliveira, Augusto Silva, Iouliia Skliarova

Universidade de Aveiro

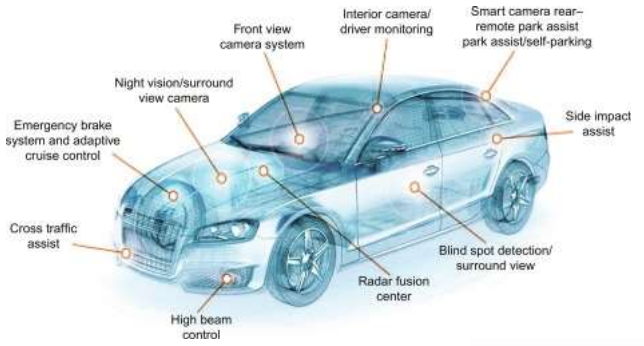
Lecture 1 contents

- About digital design
- Number systems
  - Positional number systems
  - Binary, octal, and hexadecimal numbers
  - General positional-number-system conversions

Universidade de Aveiro

Motivation

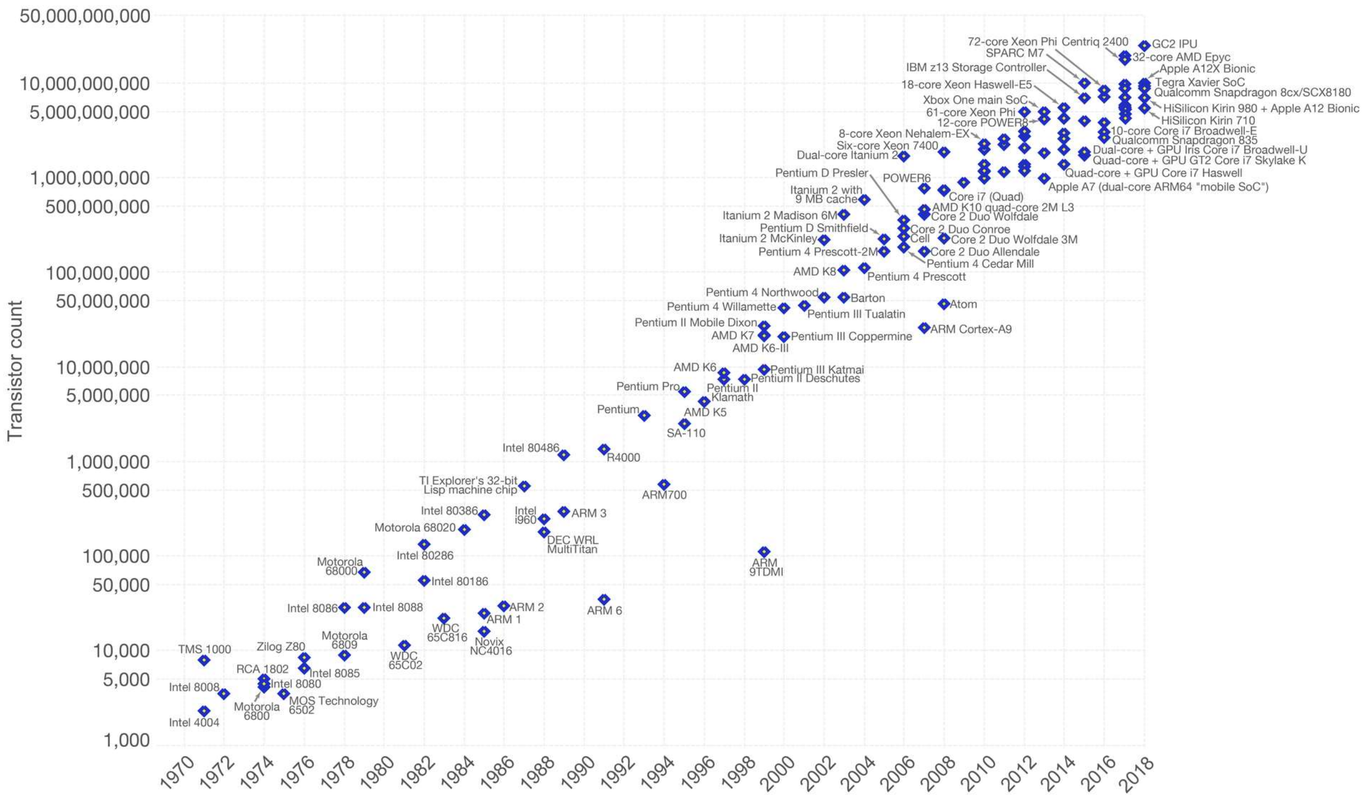
Electronics all around us: automotive, consumer products, communications, military and aerospace, medicine, etc.



https://www.sciencedirect.com/topics/engineering/automotive-electronics
http://www.beadelectronics.com/military-and-aerospace-electronics-applications
https://electronicsforu.com/technology-trends/medical-electronics-key-concerns-opportunities
https://www.ittechnologynews24.com/tag/online-consumer-electronics-market-review/
http://www.indiaeducation.net/engineering/engineering-branch/electronics-engineering-versus-electronics-communication-engineering.html

Exponential Growth

Moore's law (observed in 1965): the number of transistors on a computer chip doubles every 1.5-2 years

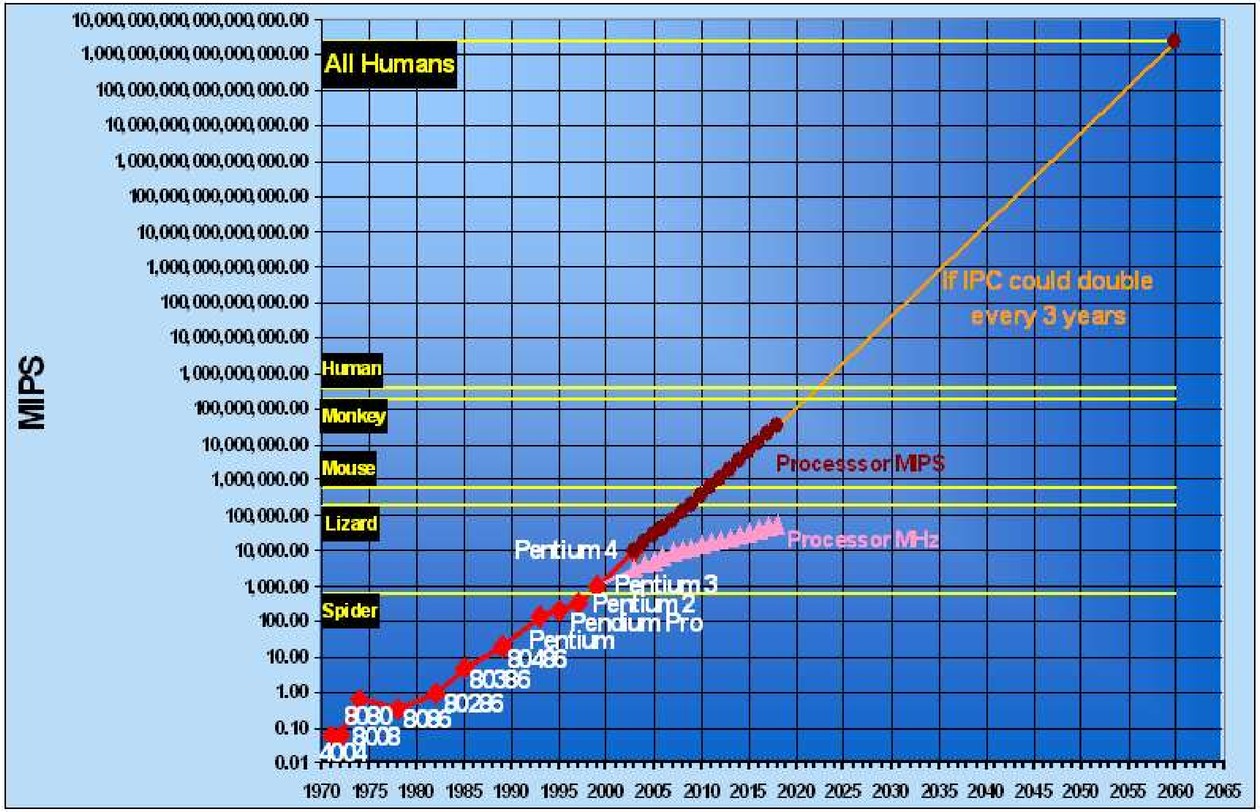


https://en.wikipedia.org/wiki/Moore%27s_law#/media/File:Moore's_Law_Transistor_Count_1971-2018.png

Universidade de Aveiro

Exponential Growth

Processing power increases at about the same rate (~40%)



http://www.captec-group.com/

Course content

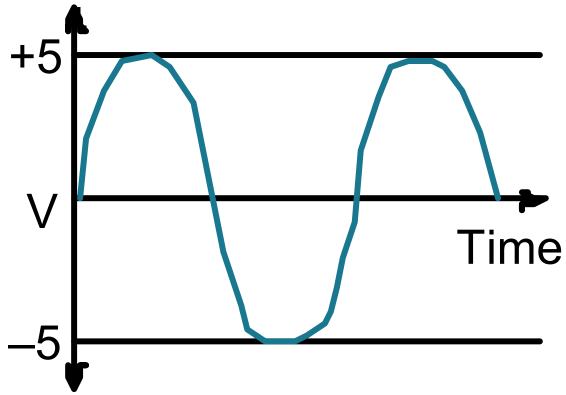
Application software (programs)
Operating systems (device drivers)
Instruction Set Arch (instructions, registers)
Machine organization (datapaths, controllers)
Logic (adders, encoders)
Digital circuits (gates)
Analog circuits (amplifiers)
Devices (transistors)
Physics (electrons)

- Fundamentals of Boolean logic
- Encoding
- Combinational circuits
- Arithmetic units
- Synchronous circuits
- Finite state machines
- Timing and clocking
- Simulation

Universidade de Aveiro

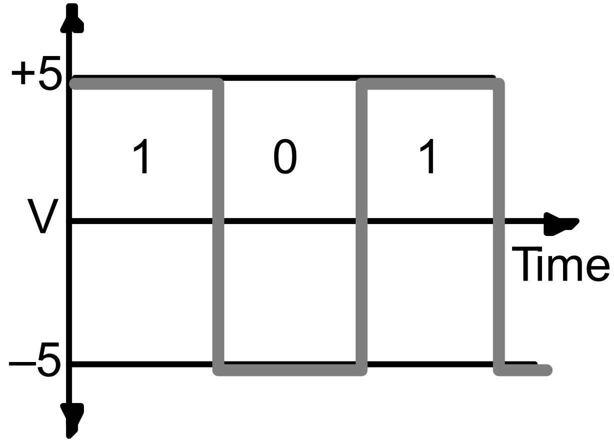
Digital abstraction

Analog: values vary over a broad range continuously



Digital: only assumes discrete values

- reproducibility of results
- ease of design
- programmability
- speed



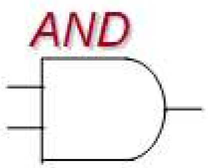
Boolean Algebra

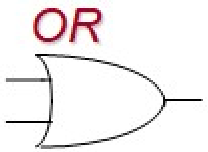
- Inputs and outputs can only have two discrete values:
  - physical domain (usually, voltages) (0V/5V)
  - mathematical domain : Boolean variables (true/ false, 0/1)
- **Boolean algebra** is used to analyze and describe the behavior of digital circuits
  - a symbolic variable, such as *x*, represents the condition of a logic signal
  - algebraic operators, such as AND, OR and NOT, represent **logic gates**
    - a **gate** is the most basic digital device and has one or more input and produces an output that is a function of the current input value(s)

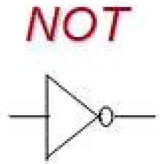
Universidade de Aveiro

Logic Gates

- AND – logical product
- OR – logical sum
- NOT - inversion







x	y	x and y xy $x \cdot y$ $x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x or y $x+y$ $x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

x	not x $\bar{x}$ x'
0	1
1	0

Number Systems

- Digital systems are built from circuits that process binary digits
- Very few real-life problems are based on binary numbers or any numbers at all
- Some correspondence must be established between the binary digits processed by digital circuits and real-life numbers, events, and conditions
  - How to represent familiar numeric quantities?
    - number systems: binary, octal, and hexadecimal
  - How to represent nonnumeric data?

Positional Number Systems

- In a positional number system, a number is represented by a string of digits, where each digit position has an associated weight.
- The value of a number is a weighted sum of the digits. For a decimal system with **radix** *r* = 10:
  - $1734_{10} = 1 \cdot 1000 + 7 \cdot 100 + 3 \cdot 10 + 4 \cdot 1$
  - $5185.68_{10} = 5 \cdot 1000 + 1 \cdot 100 + 8 \cdot 10 + 5 \cdot 1 + 6 \cdot 0.1 + 8 \cdot 0.01$

- In a general positional number system, the radix may be any integer $r \geq 2$, and a digit in position *i* has weight r^i.
- The general form of a number *D* in such a system is

$$D = d_{p-1}d_{p-2}\ldots d_1 d_0 . d_{-1} d_{-2} \ldots d_{-n} = \sum_{i=-n}^{p-1} d_i * r^i$$

, where there are *p* digits to the left of the point and *n* digits to the right of the point, called the **radix point**.

Radices and Sets of Symbols

number system	radix	symbols
binary	2	0,1
octal	8	0, 1,…, 7
decimal	10	0, 1,…, 9
hexadecimal	16	0, 1,…, 9, A, B, C, D, E, F

Examples:

$$D = \sum_{i=-n}^{p-1} d_i * 10^i$$

Decimal

$2007_{10} = 2*1000 + 0*100 + 0*10 + 7*1$

$19.85_{10} = 1*10 + 9*1 + 8*0.1 + 5*0.01$

$$D = \sum_{i=-n}^{p-1} d_i * 2^i$$

Binary

$1100110_2 = 1*2^6 + 1*2^5 + 1*2^2 + 1*2^1 = 64 + 32 + 4 + 2 = 102_{10}$

$101.0011_2 = 1*2^2 + 1*2^0 + 1*2^{-3} + 1*2^{-4}$

Most significant bit

Least significant bit

Universidade de Aveiro

Number Systems Examples

Examples:

Octal

$$D = \sum_{i=-n}^{p-1} d_i * 8^i$$

$3577_8 = 3*8^3 + 5*8^2 + 7*8^1 + 7*8^0 = 1919_{10}$

$35.77_8 = 3*8^1 + 5*8^0 + 7*8^{-1} + 7*8^{-2}$

Hexadecimal

$$D = \sum_{i=-n}^{p-1} d_i * 16^i$$

$2007_{16} = 2*16^3 + 7*16^0 = 8199_{10}$

$7D7_{16} = 7*16^2 + 13*16^1 + 7*16^0 = 2007_{10}$

$A.2C_{16} = 10*16^0 + 2*16^{-1} + 12*16^{-2}$

Universidade de Aveiro

Conversion from any Positional Number System to Decimal

- Radix 10 is important because we use it in everyday life.
- Radix 2 is important because binary numbers can be processed directly by digital circuits.
- Numbers in other radices are not often processed directly but may be important for documentation or other purposes.
- In particular, the radices 8 and 16 provide convenient shorthand representations for multibit numbers in a digital system.
- The value of the number expressed in radix *r* can be found by converting each digit of the number to its radix-10 equivalent and expanding the formula (D= $\sum_{i=-n}^{p-1} d_i \times r^i$) using radix-10 arithmetic.

Binary, Decimal, Octal , and Hexadecimal Numbers

binary	decimal	octal	hexadecimal
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	8	10	8
1001	9	11	9
1010	10	12	A
1011	11	13	B
1100	12	14	C
1101	13	15	D
1110	14	16	E
1111	15	17	F

Conversion from Decimal to any Positional Number System: Integral Part

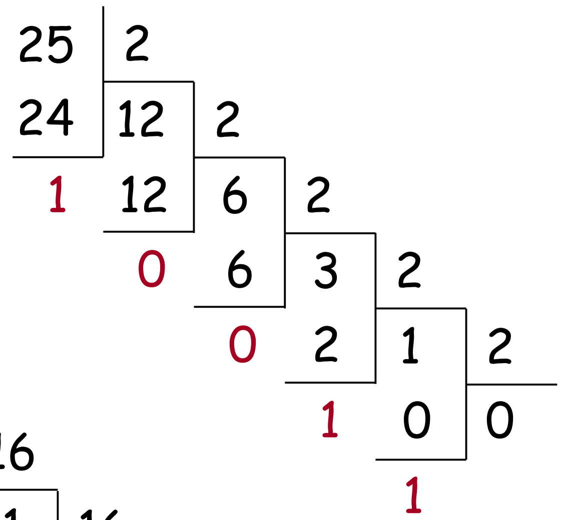
- The **integral part** of a number *D* expressed in decimal can be converted to any radix $r \geq 2$ by **successful division of *D* by *r*** (using radix-10 arithmetic, until the result is 0) with **reverse recording** (in radix *r*) of all the obtained **remainders**.

Examples:

Conversion to binary

$25_{10} = ???_2$

$25_{10} = \quad\quad _2$

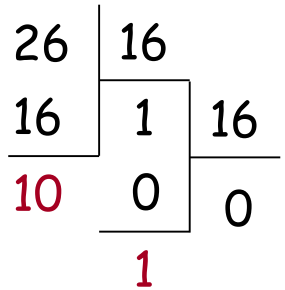


Conversion to hexadecimal

$26_{10} = ???_{16}$

$26_{10} = \quad\quad _{16}$

A



Conversion from Decimal to any Positional Number System: Fractional Part

- The **fractional part** of a number *D* expressed in decimal can be converted to any radix *r* ≥ *2* by **successful multiplication of *D* by *r*** (using radix-10 arithmetic, until the desired precision is reached) with **direct recording** (in radix *r*) of all the obtained **integral parts**.

Examples:

Conversion to binary

$0.6875_{10} = ???_2$

$0.6875_{10} = 0.\qquad_2$

```
 0.6875
      2
---------
1 .3750
      2
---------
0 .750
      2
---------
1 . 50
      2
---------
1 .00
```

Conversion to hexadecimal

$0.25_{10} = ???_{16}$

$0.25_{10} = 0.\quad_{16}$

```
 0.25
   16
---------
4 .00
```

Universidade de Aveiro

Rounding Error

- If the successive multiplication processes does not seem to be heading towards a final zero, the fractional number will have an infinite length. Try representing 0.33_{10} in binary.
- So the number of multiplication steps should be limited depending on the degree of accuracy required.
- In order to keep the accuracy of the original presentation, the following formula is applied:

$$n_2 = \lfloor n_1 * \log_{r_2} r_1 \rfloor$$

r_1 – initial radix
r_2 – final radix
n_1 – number of fractional digits in the original number, expressed in radix r_1
n_2 – number of fractional digits in the converted number, expressed in radix r_2

Examples:

$0.6875_{10} = ???_2$ $\quad$ $0.6875_{10} = 0.1011_2$ $\quad$ $0.6875_{10} = 0.101100000000_2$

$A.2C_{16} = 10*16^0 + 2*16^{-1} + 12*16^{-2} = 10 + 0.125 + 0.046875 = 10.171875 = 10.17_{10}$

$101.0011_2 = 1*2^2 + 1*2^0 + 1*2^{-3} + 1*2^{-4} = 4 + 1 + 0.125 + 0.0625 = 5.1875 = 5.2_{10}$

Universidade de Aveiro

Special Conversion Cases

- When a number is converted from radix r_1 to radix r_2 and $r_1 = r_2^x$, then each digit in radix r_1 can be **substituted directly** with x radix r_2 digits.
- The **octal** and **hexadecimal** number systems are useful for representing multibit numbers because their radices are powers of 2.
- Each **octal digit is represented with 3 binary digits** ($8=2^3$). Each **hexadecimal digit is represented with 4 binary digits** ($16=2^4$).

Examples:

Conversion from octal to binary

$r_1 = 8, r_2 = 2, 8 = 2^3$

$753.6_8 = 111\ 101\ 011\ .\ 110_2$

Conversion from hexadecimal to binary

$r_1 = 16, r_2 = 2, 16 = 2^4$

$A5.E_{16} = 1010\ 0101\ .\ 1110_2$

Universidade de Aveiro

Special Conversion Cases (cont.)

- When a number is converted from radix r_1 to radix r_2 and $r_1^x = r_2$, then a sequence of x digits in radix r_1 can be **substituted directly** with one radix r_2 digit.
- To convert a **binary number to octal**, start at the binary point and work left, separating the bits into groups of three and replacing each group with the corresponding octal digit; then work right. Freely add zeroes on the left or right to make the total number of bits a multiple of 3.
- To convert a **binary number to hexadecimal**, start at the binary point and work left, separating the bits into groups of four and replacing each group with the corresponding hexadecimal digit; then work right. Freely add zeroes on the left or right to make the total number of bits a multiple of 4.

Examples:

Conversion from binary to octal $\quad 1\,101\,.\,010_2 = 15\,.\,2_8$

$r_1 = 8,\ r_2 = 2,\ 8 = 2^3$

Conversion from binary to hexadecimal $\quad 110\ 0101\ 1100_2 = 65C_{16}$

$r_1 = 16,\ r_2 = 2,\ 16 = 2^4$

Summary of Conversion Methods

From	To	Method
Binary	Octal	Substitution (replace each group of 3 bits with the corresponding octal digit)
	Decimal	Summation (D= $\sum_{i=-n}^{p-1} d_i \times 2^i$)
	Hexadecimal	Substitution (replace each group of 4 bits with the corresponding hexadecimal digit)
Octal	Binary	Substitution (each octal digit is represented with 3 bits)
	Decimal	Summation (D= $\sum_{i=-n}^{p-1} d_i \times 8^i$)
	Hexadecimal	Convert to binary, then to hexadecimal
Decimal	Binary	Division by 2 for integral part, multiplication by 2 for fractional part
	Octal	Division by 8 for integral part, multiplication by 8 for fractional part
	Hexadecimal	Division by 16 for integral part, multiplication by 16 for fractional part
Hexadecimal	Binary	Substitution (each hexadecimal digit is represented with 4 bits)
	Octal	Convert to binary, then to octal
	Decimal	Summation (D= $\sum_{i=-n}^{p-1} d_i \times 16^i$)

Exercises

- Explain Moore's Law.
- What are the advantages of digital systems compared to analog systems?
- When an OR gate output is 0?
- When an AND gate output is 1?

Exercises (cont.)

- How to convert an integer octal number into hexadecimal?
- Consider that expression $1234_r < 1234_8$ is true. Is it possible to determine the value of *r*?
- Is it possible to convert the number 39_8 into decimal?
- What is the relationship (>, =, or <) between 34_8 and 34_{16}?
- Is the following affirmation true: $20=14_{16}$?

Exercises (cont.)

- Convert the following numbers into binary, octal, decimal, and hexadecimal:

$10111011001_2 = 2731_8 = 5D9_{16} = 1497_{10}$

$1234_8 = 001010011100_2 = 29C_{16} = 668_{10}$

$C0DE_{16} = 1100000011011110_2 = 140336_8 = 49374_{10}$

$108_{10} = 1101100_2 = 154_8 = 6C_{16}$

$15.46_{10} = 1111.011101_2 = 17.35_8 = F.7_{16}$

Universidade de Aveiro

Lecture 2 contents

- Representation of negative numbers
- Addition and subtraction of nondecimal numbers
- Codes
  - Character codes
  - Binary-coded decimal
  - Gray code

Universidade de Aveiro

Representation of Negative Numbers

- There are many ways to represent negative numbers.
- In everyday business we use the **signed-magnitude system** (i.e. reserve a special symbol to indicate whether a number is negative).
- However, most computers use **two's-complement representation**:
  - The **most significant bit** (**MSB**) of a number in this system serves as the sign bit; a number is negative if and only if its MSB is 1.
  - The weight of the MSB is negative: for an *n*-bit number the weight is -2^{n-1}.
  - The decimal equivalent for a two's-complement binary number is computed the same way as for an unsigned number, except that the weight of the MSB is negative:
    - D= $d_{n-1}d_{n-2}\ldots d_1 d_0 = -2^{n-1} + \sum_{i=0}^{n-2} d_i \times 2^i$

Examples:

$1010_2 = ???_{10}$ $\quad$ $1010_2 = -2^3 + 2^1 = -8 + 2 = -6_{10}$

$1111_2 = ???_{10}$ $\quad$ $1111_2 = -2^3 + 2^2 + 2^1 + 2^0 = -8 + 4 + 2 + 1 = -1_{10}$

$0111_2 = ???_{10}$ $\quad$ $0111_2 = 2^2 + 2^1 + 2^0 = 4 + 2 + 1 = 7_{10}$

Universidade de Aveiro

Two's Complement Representation

- For *n* bits, the range of representable numbers is $[-2^{n-1}, 2^{n-1}-1]$.
- For *n*=4, the range is [-8, 7]:

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
-8	1	0	0	0
-7	1	0	0	1
-6	1	0	1	0
-5	1	0	1	1
-4	1	1	0	0
-3	1	1	0	1
-2	1	1	1	0
-1	1	1	1	1

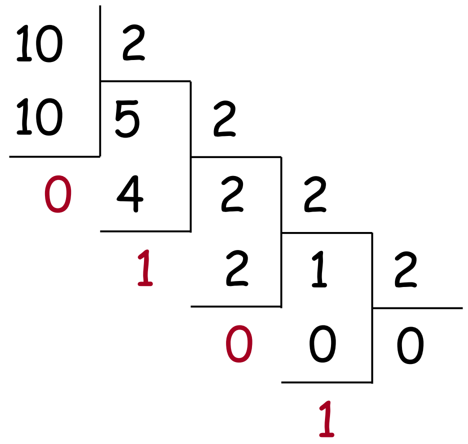
Conversion between Decimal and Two's Complement

- The decimal value of the number expressed in two's complement can be found by expanding the formula (D= $d_{n-1}d_{n-2} \dots d_1 d_0 = -2^{n-1} + \sum_{i=0}^{n-2} d_i \times 2^i$) using radix-10 arithmetic.
- The integer number *D* expressed in decimal can be converted to *n*-bit two's complement by **successful division of *D* by 2** (using radix-10 arithmetic, until the result is 0) with **reverse recording** of all the obtained **remainders**.
  - If there are **empty bit positions** left, **fill** them with **0s**.
  - **Do not exceed** the allowed **range** of representable numbers: [-2^{n-1} , $2^{n-1}-1$].
  - If the number is negative, the result must be **negated**:
    - Invert all the bits individually and add 1 or
    - Copy all the bits starting from the least significant until the first 1 is copied, then invert all the remaining bits.

Examples (with n=8):

$10_{10} = ???_2$ $\qquad$ $10_{10} = 00001010_2$

$-10_{10} = ???_2$ $\qquad$ $-10_{10} = 11110110_2$



Changing the Number of Bits

- We can convert an *n*-bit two's-complement number into an *m*-bit one.
- If *m* > *n*, perform **sign extension**:
  - append *m* - *n* copies of the sign bit to the left
- If *m* < *n*, **discard** *n* – *m* leftmost bits; however, **the result is valid only if all of the discarded bits are the same as the sign bit of the result**.

Examples:

n = 5 m = 8	00101 = **000**00101 11110 = **111**11110
n = 5 m = 3	**00**101 = 101 – result is not valid **11**110 = 110 – result is valid

Universidade de Aveiro

Addition of Binary Numbers

- Addition and subtraction of nondecimal numbers by hand uses the same technique that you know from school for decimal numbers.
- The only catch is that the addition and subtraction tables are different.
- To add two **binary numbers** *X* and *Y*, we add together the least significant bits with an initial carry (c_{in}) of 0, producing carry (c_{out}) and sum (s) bits according to the table. We continue processing bits from right to left, adding the carry out of each column into the next column's sum.

c_{in}	x	y	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Example:

```
     0 1 0 0 0 0 1
     0 0 1 0 1 1 0 1
  +  0 1 1 0 0 0 0 1
  ------------------
     1 0 0 0 1 1 1 0
```

Universidade de Aveiro

Subtraction of Binary Numbers

- Binary subtraction is performed similarly, using borrows (b_{in} and b_{out}) instead of carries between steps, and producing a difference bit d.

b_{in}	x	y	b_{out}	d
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

Examples:

```
    0 1 1 1 1 0 0
    1 1 1 0 0 0 0 1
  - 1 0 1 0 1 1 0 1
  -----------------
    0 0 1 1 0 1 0 0
```

```
    1 1 1
    1 0 0 0
  - 0 0 1 1
  ---------
    0 1 0 1
```

Universidade de Aveiro

Addition of Octal Numbers

- To add two **octal numbers** *X* and *Y,* we add together the least significant digits with an initial carry (c_{in}) of 0. If the *intermediate result* is less than or equal to 7, then c_{out} = 0 and s = *intermediate result*. If the *intermediate result* is greater than 7, then c_{out}= 1 and sum (s) digit = *intermediate result* – 8.
- We continue processing digits from right to left, adding the carry out of each column into the next column's sum.

Examples (radix 8):

	0	0	0	
	3	0	4	1
+	1	7	3	2
	4	7	7	3

	1	1	1	
	3	4	5	6
+	1	7	3	4
	5	4	1	2

Addition of Hexadecimal Numbers

- To add two **hexadecimal numbers** *X* and *Y*, we add together the least significant digits with an initial carry (c_{in}) of 0. If the *intermediate result* is less than or equal to 15, then c_{out} = 0 and s = *intermediate result*. If the *intermediate result* is greater than 15, then c_{out}= 1 and sum (s) digit = *intermediate result* – 16.
- We continue processing digits from right to left, adding the carry out of each column into the next column's sum.

Examples (radix 16):

$$
\begin{array}{cccccc}
 & 1 & 0 & 0 & \\
 & 3 & A & 4 & 1 \\
+ & 1 & 7 & 8 & 2 \\
\hline
 & 5 & 1 & C & 3
\end{array}
\qquad\qquad
\begin{array}{cccccc}
 & 1 & 1 & 0 & \\
 & 3 & F & A & A \\
+ & B & 7 & E & 4 \\
\hline
 & F & 7 & 8 & E
\end{array}
$$

Subtraction of Octal and Hexadecimal Numbers

- When subtracting **octal numbers**, a borrow brings the value 8.
- When subtracting **hexadecimal numbers**, a borrow brings the value 16.

Examples:

radix 8

$$\begin{array}{ccccc} & 1 & 0 & 1 & \\ & 3 & 0 & 4 & 1 \\ - & 1 & 7 & 3 & 2 \\ \hline & 1 & 1 & 0 & 7 \end{array} \qquad \begin{array}{ccccc} & 1 & 1 & 1 & \\ & 6 & 0 & 0 & 0 \\ - & 1 & 5 & 7 & 7 \\ \hline & 4 & 2 & 0 & 1 \end{array}$$

radix 16

$$\begin{array}{ccccc} & 0 & 1 & 1 & \\ & 3 & A & 4 & 1 \\ - & 1 & 7 & 8 & 2 \\ \hline & 2 & 2 & B & F \end{array} \qquad \begin{array}{ccccc} & 1 & 1 & 1 & \\ & B & 0 & 0 & 0 \\ - & A & 7 & E & 4 \\ \hline & 0 & 8 & 1 & C \end{array}$$

Universidade de Aveiro

Two's-Complement Addition

- Addition is performed in the same way as for nonnegative numbers.
- Carries beyond the MSB are **ignored**.
- The result will always be the correct sum as long as the range of the number system is not exceeded.
- If an addition operation produces a result that exceeds the range of the number system, **overflow** is said to occur.
- Addition of two numbers with different signs can never produce overflow.
- Addition of two numbers of like sign can produce overflow if
  - the addends' signs are the same but the sum's sign is different from the addends'.
  - the carry bits c_{in} into and c_{out} out of the sign position are different.

Examples (n=4):

```
  1 1 0 0
    0 1 0 0
  + 1 1 0 1
  ---------
    0 0 0 1
```

```
    0 1 0
    0 0 1 0
  + 0 0 1 1
  ---------
    0 1 0 1
```

```
  1 0 0 0
    1 0 0 1
  + 1 1 1 0
  ---------
    0 1 1 1
```

overflow

Two's-Complement Subtraction

- Two's-complement numbers may be subtracted as if they were ordinary unsigned binary numbers.
- However, **most subtraction circuits for two's-complement numbers do not perform subtraction directly**.
- Rather, they **negate the subtrahend** by taking its two's complement, and then **add** it to the minuend using the normal rules for addition (***X***-***Y***=***X***+(-***Y***)).
- Overflow in subtraction can be detected using the same rule as in addition.
- Negating the subtrahend and adding the minuend can be accomplished with only one addition operation:
  - Perform a bit-by-bit complement of the subtrahend and add the complemented subtrahend to the minuend with an initial carry (c_{in}) of 1 instead of 0.

Examples (n=4):

0010 - 0011:

$$\begin{array}{rcccc} & 0 & 0 & 0 & 1 \\ & 0 & 0 & 1 & 0 \\ + & 1 & 1 & 0 & 0 \\ \hline & 1 & 1 & 1 & 1 \end{array}$$

1011 - 0110:

$$\begin{array}{rccccc} & 1 & 0 & 1 & 1 & 1 \\ & & 1 & 0 & 1 & 1 \\ + & & 1 & 0 & 0 & 1 \\ \hline & & 0 & 1 & 0 & 1 \end{array}$$

overflow

Universidade de Aveiro

Information Encoding

- Digital systems are built from circuits that process binary digits
- Very few real-life problems are based on binary numbers or any numbers at all
- Some correspondence must be established between the binary digits processed by digital circuits and real-life numbers, events, and conditions
  – How to represent familiar numeric quantities? ✓
    - number systems: binary, octal, and hexadecimal
  – How to represent nonnumeric data?

Codes

- A **code** is a set of n-bit strings in which different bit strings represent different numbers or other things.
- A **code word** is a particular combination of n bit-values.
- To code m values, the code length n must respect the following equation: $n \geq \lceil log_2 m \rceil$.

floor	encoding	encoding	encoding
basement	000	000	000001
ground floor	001	001	000010
1st floor	010	011	000100
2nd floor	011	010	001000
3rd floor	100	110	010000
4th floor	101	111	100000

Universidade de Aveiro

Character Codes

- The most common type of nonnumeric data is text, strings of characters from text.
- Each character is represented in the digital system by a bit string according to an established convention.
- The most commonly used character code is **ASCII** (American Standard Code for Information Interchange).
  - ASCII represents each character with a 7-bit string, yielding a total of 128 different characters.

		$b_6b_5b_4$ (column)							
$b_3b_2b_1b_0$	Row (hex)	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
0000	0	NUL	DLE	SP	0	@	P	`	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	'	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[	k	{
1100	C	FF	FS	,	<	L	\	l	|
1101	D	CR	GS	-	=	M	]	m	}
1110	E	SO	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O	_	o	DEL

Universidade de Aveiro

Binary Codes for Decimal Numbers

- Even though binary numbers are the most appropriate for the internal computations of a digital system, most people still prefer to deal with decimal numbers.
- As a result, the external interfaces of a digital system may read or display decimal numbers, and some digital devices actually process decimal numbers directly.
- A decimal number is represented in a digital system by a string of bits, where different combinations of bit values in the string represent different decimal numbers.
- To code *m* = 10 decimal digits, at least $\lceil log_2 10 \rceil = 4$ bits are required.
- Is the maximum number of bits limited?
- Is the number of possible codes limited?

Universidade de Aveiro

Binary-Coded Decimal (BCD)

- Perhaps the most "natural" decimal code is **binary-coded decimal** (BCD), which encodes the digits 0 through 9 by their 4-bit unsigned binary representations, 0000 through 1001.
- The code words 1010 through 1111 are not used.
- Conversions between BCD and decimal representations are trivial, a direct substitution of four bits for each decimal digit.

Example:

$25_{10} = 11001_2$

$25_{10} = 00100101_{BCD}$

decimal digit	BCD (8421)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Gray Code

- Sometimes, it is required to code values so that only **one bit changes** between each pair of successive code words.
- Such a code is called a **Gray code**.
- There are two convenient ways to construct a Gray code with any desired number of bits.

1 bit	2 bits	3 bits	4 bits
0	00	000	0000
1	01	001	0001
	11	011	0011
	10	010	0010
		110	0110
		111	0111
		101	0101
		100	0100
			1100
			1101
			1111
			1110
			1010
			1011
			1001
			1000

Universidade de Aveiro

Constructing Gray Code

- The first method is based on the fact that Gray code is a reflected code; it can be defined (and constructed) recursively using the following rules:
  - A 1-bit Gray code has two code words, 0 and 1.
  - The first 2^n code words of an $(n + 1)$-bit Gray code equal the code words of an n-bit Gray code, written in order with a leading 0 appended.
  - The last 2^n code words of an $(n + 1)$-bit Gray code equal the code words of an n-bit Gray code, but written in reverse order with a leading 1 appended.

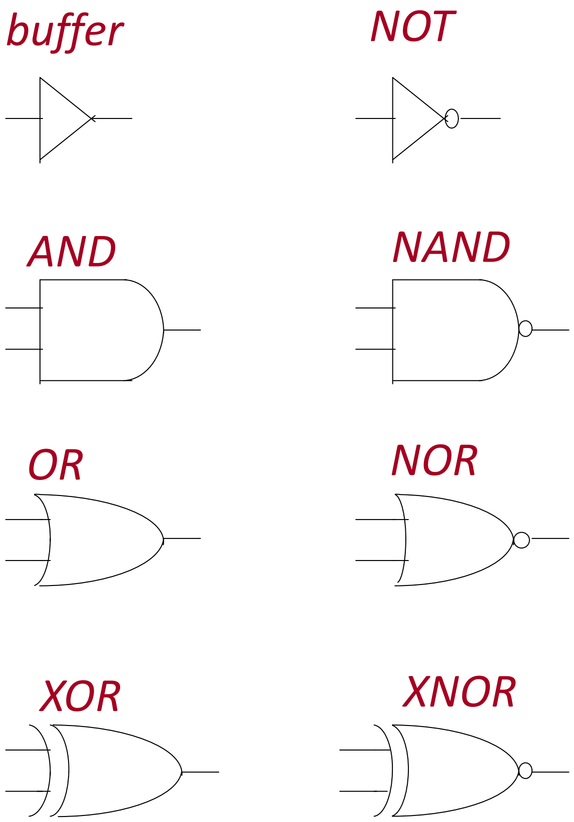
Constructing Gray Code (cont.)

- The second method allows us to derive an *n*-bit Gray-code code word directly from the corresponding *n*-bit binary code word:
  - The bits of an *n*-bit binary or Gray-code code word are numbered from right to left, from 0 to *n* - 1.
  - Bit *i* of a Gray-code code word is 0 if bits *i* and *i* + 1 of the corresponding binary code word are the same, else bit *i* is 1.
  - When *i* + 1 = *n*, bit *n* of the binary code word is considered to be 0
- Similarly, an *n*-bit Gray-code code word can be converted to the corresponding *n*-bit binary code word:
  - The bits of an *n*-bit Gray-code code word are numbered from right to left, from 0 to *n* - 1.
  - Bit *n* – 1 of a binary code word is equal to bit *n* - 1 of a Gray-code code word.
  - Bit *i* (*i* = *n*-2, *n*-3,…, 1, 0) of a binary code word is 0 if bits *i* of the corresponding Gray-code code word and *i* + 1 of the corresponding binary code word are the same, else bit *i* is 1.

Example: $11001_2 = 10101_{GRAY}$

Universidade de Aveiro

Logic Gates



$$x \oplus y$$

x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

$$\overline{x \oplus y}$$

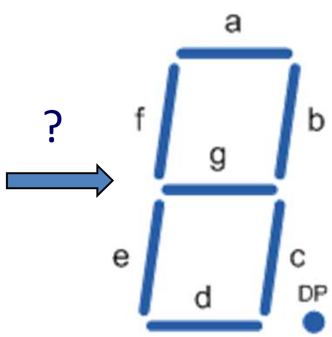
x	y	x XNOR y
0	0	1
0	1	0
1	0	0
1	1	1

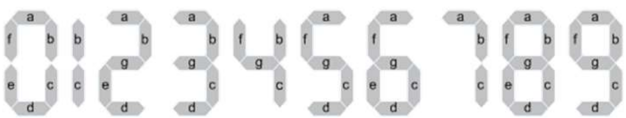
Universidade de Aveiro

7-segment Display Codes

- 7-segment displays are used in watches, calculators, and instruments to display decimal data.
- A digit is displayed by illuminating a subset of the seven line segments.

0,1,…,9





BCD	digit	individual segments						
		a	b	c	d	e	f	g
0000	0	1	1	1	1	1	1	0
0001	1	0	1	1	0	0	0	0
0010	2	1	1	0	1	1	0	1
0011	3	1	1	1	1	0	0	1
0100	4	0	1	1	0	0	1	1
0101	5	1	0	1	1	0	1	1
0110	6	1	0	1	1	1	1	1
0111	7	1	1	1	0	0	0	0
1000	8	1	1	1	1	1	1	1
1001	9	1	1	1	1	0	1	1

Universidade de Aveiro

Hamming Distance

- The **Hamming distance** between two *n*-bit strings is the number of bit positions in which they differ.
- In the Gray code, the Hamming distance between each pair of successive code words is 1.

Example:

1	0	1	0	1	0	1	0
1	1	0	1	1	1	1	0

Hamming distance = 4

Bits, Bytes, Words, etc.

- The prefixes K (kilo-), M (mega-), G (giga-), and T (tera-) mean 10^3, 10^6, 10^9, and 10^{12}, respectively, when referring to bps, hertz, ohms, watts, and most other engineering quantities.
- However, when referring to memory sizes, the prefixes mean 2^{10}, 2^{20}, 2^{30}, and 2^{40}.

Bit	*b*	0 or 1
Byte	*B*	8 bits
Nibble		4 bits
Word		8, 16, 32, 64 ... bits (depends on the context)

1 K/k	$10^3 \approx 2^{10}$		(*kilo*)
1 M	$10^6 \approx 2^{20}$		(*mega*)
1 G	$10^9 \approx 2^{30}$		(*giga*)
1 T	$10^{12} \approx 2^{40}$		(*tera*)

IEEE 1541-2002:

Ki	2^{10} = 1 024	(*kibi*)
Mi	2^{20} = 1 048 576	(*mebi*)
Gi	2^{30} = 1 073 741 824	(*gibi*)
Ti	2^{40} = 1 099 511 627 776	(*tebi*)
Pi	2^{50} = 1 125 899 906 842 624	(*pebi*)
Ei	2^{60} = 1 152 921 504 606 846 976	(*exbi*)

Universidade de Aveiro

Exercises

- Represent the following numbers in two's complement with 8 bits: 39_{10}, -22_{10}.

- Calculate the results of the following operations in two's complement with 8 bits. Detect overflows if any.

	1	0	1	0	1	0	1	0
+	0	1	0	1	0	1	0	1

		0	0	0	0	0	0	0	
		1	0	1	0	1	0	1	0
+		0	1	0	1	0	1	0	1
		1	1	1	1	1	1	1	1

	0	1	1	1	1	1	1	1
+	0	0	1	1	1	1	1	1

	0	1	1	1	1	1	1	1	
		0	1	1	1	1	1	1	1
+		0	0	1	1	1	1	1	1
		1	0	1	1	1	1	1	0

	1	1	0	1	0	0	1	0
-	0	1	1	0	1	1	0	1

	1	0	0	1	0	0	1	0	1
		1	1	0	1	0	0	1	0
+		1	0	0	1	0	0	1	0
		0	1	1	0	0	1	0	1

Exercises (cont.)

- Add the following pairs of octal numbers:

$$\begin{array}{r} 1\ 7\ 7\ 6 \\ +\ 1\ 4\ 3\ 2 \\ \hline \end{array} \qquad \begin{array}{r} 3\ 7\ 7\ 7 \\ +\ 1\ 7\ 7\ 7 \\ \hline \end{array}$$

- Add the following pairs of hexadecimal numbers:

$$\begin{array}{r} 1\ 7\ 7\ 6 \\ +\ 1\ 4\ 3\ 2 \\ \hline \end{array} \qquad \begin{array}{r} 3\ F\ F\ F \\ +\ A\ B\ C\ D \\ \hline \end{array}$$

- Each of the following arithmetic operations is correct in at least one number system. Determine possible radices of the numbers in each operation.
  - 1234 + 5432 = 6666
  - $\sqrt[2]{41} = 5$

Universidade de Aveiro

Exercises (cont.)

- How many bits of information can be stored on a 16 GB pen?
- How many digital photos is it be possible to store on an 8 GB pen assuming that each photo has 4000 x 3000 pixels and each pixel is coded with 24 bits?
- Assuming that the following quantity is represented in two’s complement, indicate its decimal value:
  111111111111111111111111111111111111111111001
- Express in decimal, binary, and hexadecimal systems the value of the largest non-negative integer you can represent in a register with a storage capacity of 2 octal digits.

Exercises (cont.)

- How many bits are required to code in BCD the number 123456_{10}?

- Represent the following values in binary and in BCD and Gray codes.

108_{10} $= 000100001000_{BCD}$
$= 1101100_2$
$= 1011010_{GRAY}$

33_8 $= 00100111_{BCD}$
$= 011011_2$
$= 010110_{GRAY}$

- Prove that a two's-complement number can be converted to a representation with more bits by *sign extension.*

- Determine the Hamming Distance between the following code words:

011010101011
000010101011

= 2

Universidade de Aveiro

Exercises (cont.)

- Airport names are encoded by sequences of three capital letters of English alphabet (having 26 letters).
- How many airports can be coded this way?
- How many bits will be required in ASCII code to binary encode the airport codes?
- And if you use the most efficient code possible to encode only uppercase letters?