Representing natural numbers

Prof. Daniele Gorla

Representing Information

Electronic calculators are machines that can handle *information* and convert it in other information.

In computer science, information is *whatever can be represented through proper sequences of symbols taken from a fixed (finite) alphabeth*.

A *code* C is a set of words formed by the symbols of an alphabeth Σ (called *support* of C).

Coding and decoding

CODING

A *coding function* of a set of info's I in a given code C is

$$f : \mathrm{I} \rightarrow \mathrm{C}$$

Example:

car → *00*   *shuttle* → *01*   *airplane* → *10*

where: I is a set of words

C is a subset of the words made up by 0 and 1

DECODING

A *decoding function* of a previously coded info is

$$g : \mathrm{C} \rightarrow \mathrm{I}$$

(typically, it is the inverse function of *f*)

Example:

00 → *macchina*   *01* → *razzo*   *10* → *aereo*

Valuation Criteria for a coding

Economicity: codes that use less symbols are considered better

Simplicity of coding/decoding: practical performances are desirable

Simplicity of elaboration: we look for codes that provide easier ways of executing operations (e.g., roman numeric system does not provide an easy way of performing sums/subtractions – the mechanisms of carry/borrow fail).

Positional Systems

SAPIENZA Università di Roma Dipartimento di Informatica

A *positional numeric system* in base b, i.e. relying on an **alphabeth** Σ made up of b distinct symbols, allows us to express any natural number N of m digits, through the formula

$$N = \sum_{i=0}^{m-1} c_i b^i, \quad c_i \in \Sigma$$

For example, in the decimal system (b=10, Σ=0,1,..9), the sequence $N_{10} = 254$ expresses the number

$$254_{10} = 200 + 50 + 4$$
$$= 2\cdot10^2 + 5\cdot10^1 + 4\cdot10^0$$

Similarly, in base 7 (b=7, Σ=0,1,..6), the same sequence $N_7 = 254$ expresses the number

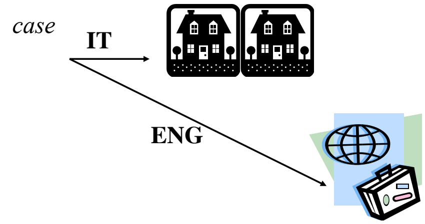
$$254_7 = 2\cdot7^2 + 5\cdot7^1 + 4\cdot7^0 \; (= 98+35+4 = 137)_{10}$$

SAPIENZA Università di Roma Dipartimento di Informatica

Hence, 10_{10} and 10_2 have a **different** meaning, even if they are the same sequence of symbols (1 followed by 0)!!

To understand the meaning of a sequence, we must know the *decoding function* to use

Example:



SAPIENZA Università di Roma Dipartimento di Informatica

	Base 2	Base 3	Base 4	Base 5	…	Base 8	…	Base 10	…	Base 16
	0000	000	00	00		00		00		0
	0001	001	01	01		01		01		1
	0010	002	02	02		02		02		2
	0011	010	03	03		03		03		3
	0100	011	10	04		04		04		4
	0101	012	11	10		05		05		5
	0110	020	12	11		06		06		6
N_b	0111	021	13	12		07		07		7
	1000	022	20	13		10		08		8
	1001	100	21	14		11		09		9
	1010	101	22	20		12		10		A
	1011	102	23	21		13		11		B
	1100	110	30	22		14		12		C
	1101	111	31	23		15		13		D
	1110	112	32	24		16		14		E
	1111	120	33	30		17		15		F

Binary Code

SAPIENZA Università di Roma Dipartimento di Informatica

Binary code: a code made up just from the symbols 0 and 1.

So, b=2 e Σ ={0,1}

0 ed 1 are called *bit*, a contraction for *binary digit*.

Why binary code?

- *George Boole* proved that al logic can be reduced to a simple algebraic system using the binary code.
- Binary code turned out very useful in the *commutation theory* (by Claude Shannon) to describe the behavious of digital circuits (1=on, 0=off).

Every info can be codified in binary

- Let's start with numbers: codes will be **different** according to what kind of numbers we have to represent (naturals, integers, rationals, …)
- We can also encode words (sequences of alphabetical characters, through ASCII and/or UNICODE codes), images, sounds, …

Binary Representation of Naturals

SAPIENZA Università di Roma Dipartimento di Informatica

A **natural number** N in base 2 with n bits is represented by the formula:

$$N_2 = \sum_{i=0}^{n-1} c_i 2^i,\ c_i \in \{0,1\}$$

Bit c_0 is called LSB (*less signifying bit*) whereas c_{n-1} is called MSB (*most signifying bit*).

Example: $111001_2 = 1\cdot2^5+1\cdot2^4+1\cdot2^3+0\cdot2^2+0\cdot2^1+1\cdot2^0$

Changing the base

SAPIENZA Università di Roma Dipartimento di Informatica

Problem: given a number N in base a (N_a),

turn it into a number N' in base b (N'_b)

Base conversion: polynomial method

SAPIENZA Università di Roma Dipartimento di Informatica

We use the formula

$$N_a = \sum_{i=0}^{n-1} c_i a^i,\ c_i \in \{0,\dots,a-1\}$$

That is, we express N_a as a polynomium, by **using digits of the alphabeth b** in the polynomium

Then, we evaluate the polynomium by **using arithmetics in base b**

Example (from base 2 to base 10):

$111001_2 = (1\cdot2^5+1\cdot2^4+1\cdot2^3+0\cdot2^2+0\cdot2^1+1\cdot2^0)_{10}$

$= (32+16+8+0+0+1)_{10} = 57_{10}$

Polynomial Method

SAPIENZA Università di Roma Dipartimento di Informatica

Hard to use if the final base (b) is not 10
(you have to work with the arithmetics modulo b!!)

Example (from base 7 to base 3):
$3602_7 = (10\cdot21^3 + 20\cdot21^2 + 0\cdot21^1 + 2\cdot21^0)_3$

WE NEED ANOTHER METHOD!!!

Theorem of the Euclidean Division

For every $D,d \in \mathbf{N}$, there exists a unique pair $q,r \in \mathbf{N}$ s.t.

$$D = q \cdot d + r \,, \quad \text{with } 0 \leq r < d$$

dividend, quotient, divisor, reminder

Notationally, $q = D \text{ div } d$
$r = D \text{ mod } d$

For example, $7 \text{ div } 3 = 2$
$7 \text{ mod } 3 = 1$

$$N = c_0 + c_1 \cdot b^1 + c_2 \cdot b^2 + c_3 \cdot b^3 + \ldots + c_{n-2} \cdot b^{n-2} + c_{n-1} \cdot b^{n-1}$$

$$= c_0 + b \cdot \underbrace{(c_1 + c_2 \cdot b^1 + c_3 \cdot b^2 + \ldots + c_{n-2} \cdot b^{n-3} + c_{n-1} \cdot b^{n-2})}_{\text{quotient}}$$

dividend, reminder, divisor

From which

$N \text{ mod } b = c_0$

$N \text{ div } b = c_1 + c_2 \cdot b^1 + c_3 \cdot b^2 + \ldots + c_{n-2} \cdot b^{n-3} + c_{n-1} \cdot b^{n-2} = N^{(1)}$

By iterating the reasoning

$N^{(1)} \text{ mod } b = c_1$

$N^{(1)} \text{ div } b = c_2 + c_3 \cdot b^1 + \ldots + c_{n-2} \cdot b^{n-4} + c_{n-1} \cdot b^{n-3} = N^{(2)}$

$N^{(2)} \text{ mod } b = c_2$

$N^{(2)} \text{ div } b = c_3 + \ldots + c_{n-2} \cdot b^{n-5} + c_{n-1} \cdot b^{n-4} = N^{(3)}$

…

Until we arrive at $N^{(n-1)} = c_{n-1}$ and then we have

$N^{(n-1)} \text{ mod } b = c_{n-1}$

$N^{(n-1)} \text{ div } b = 0$

Base conversion: iterated divisions method

To convert N_a in base b

1. Repeatedly divide N_a for b_a
   (IMP.: **b must be expressed in base a and the division is done in base a**)

2. The remainders of the divisions, converted in base b, give us the digits, from the less to the most signifying one, of N_a expressed in base b.

Example: 657_{10} in base 4

657 : 4 = 164 reminder 1
164 : 4 = 41 reminder 0
41 : 4 = 10 reminder 1
10 : 4 = 2 reminder 2
2 : 4 = 0 reminder 2

Hence, $657_{10} = 22101_4$

Example (from base 10 to base 16)

We have to perform the division of 317_{10} by 16_{10}
(in base 16, digits are 0,1,…,9,A,B,…,F)

1) 317 **:** 16
$Q=19_{10}$, $r=13_{10}$
$13_{10}=\mathbf{D}_{16}$ (LSD)

2) 19 **:** 16
$Q=1_{10}$, $r=3_{10}$
$3_{10}=\mathbf{3}_{16}$

3) 1 **:** 16
$Q=0_{10}$, $r=1_{10}$
$1_{10}=\mathbf{1}_{16}$ (MSD)

Hence,

$317_{10}=\mathbf{13D}_{16}$

N.B.: in the previous algorithm, the division **is in base *a* (i.e., in the base of the starting number)**. If $a \neq 10$, this is difficult!

Example (starting and arrival bases are not 10)

Convert 102202_3 in base 5

Three ways:

a) Repeatedly calculate 102202_3 **:** 12_3

division with arithmetics in base 3 → **DIFFICULT!**

b) Use the polynomial method for 102202_3

products and sums with arithmetics in base 5 → **DIFFICULT!**

c) Convert 102202_3 in base 10 (polynomial meth.) and then convert the result in base 5 (iterated divisions)

- $102202_3 = 3^5 + 2 \cdot 3^3 + 2 \cdot 3^2 + 2 = 317_{10}$
- 317:5 = 63 reminder 2
  63:5 = 12 reminder 3
  12:5 = 2 reminder 2
  2:5 = 0 reminder 2

Hence, $102202_3 = 2232_5$

Conversions from base *a* to base a^k

Prop.: in the arithmetics in base *a* we have that

$$c_{n-1} \ldots c_1 c_0 \ \text{mod}\ a^k = c_{k-1} \ldots c_0$$
$$c_{n-1} \ldots c_1 c_0 \ \text{div}\ a^k = c_{n-1} \ldots c_k$$

Example ($a = 10$ and $k = 2$): 453_{10} mod $100 = 53$
453_{10} div $100 = 4$ (since $100 = 10^2$)

So, if $b = a^k$ and $N_a = c_{n-1} \ldots c_1 c_0$, then the number in base b is

$$(c_{n-1} \ldots c_{hk})_b \ldots (c_{3k-1} \ldots c_{2k})_b (c_{2k-1} \ldots c_k)_b (c_{k-1} \ldots c_0)_b$$

Specific case: *Conversions from base 2 to base* 2^k

Take the bits in k-tuples starting from the LSB and convert them in base 2^k

Example: convert 1000111101 from base 2 to base 4 ($= 2^2$)

$(10\ 00\ 11\ 11\ 01)_2 = (2\ 0\ 3\ 3\ 1)_4$

Example

Convert 101001101101 from base 2 to base 8 and 16

OBS: $8=2^3$ and $16=2^4$

First conversion: gruop in triples and convert:

$(101\ 001\ 101\ 101)_2 \rightarrow (5\ 1\ 5\ 5)_8$

Second conversion: group in quadruples and convert

$(1010\ \ 0110\ \ 1101)_2 \rightarrow (A\ 6\ D)_{16}$

OBS: in making the k bits groups, the most signifying group can have less than k bits (if the length of the starting sequence is not a multiple of k); in this case, we add 0's on top

Ex.: 1001101101 from base 2 to base 8: 001 001 101 101

Conversions from base b^k to base *b*

Similarly, if $a = b^k$ and $N_a = c_{n-1} \ldots c_1 c_0$, then the number in base b is

$$(c_{n-1})_b \ldots (c_2)_b (c_1)_b (c_0)_b$$

where every c_i is converted in base b by using k digits.

Example 1: Convert 8315_9 in base 3

Since $9 = 3^2$, we can convert every digit of the given number in base 9 by using two digits in base 3:

$8\ 3\ 1\ 5_9 = 22\ 10\ 01\ 12_3$

Example 2: Convert $8D3A_{16}$ in base 2

Since $16 = 2^4$, we can convert every digit of the given number in base 16 with 4 bits:

$8\ D\ 3\ A_{16} = 1000\ 1101\ 0011\ 1010_2$

To convince yourselves of the correctness of this method, do the conversions in base 10, as explained in the previous slides!

20

Base Conversions: Summing up

- The polynomial method is easy if the arrival base is 10
- Iterated divisions method is easy if the starting base is 10
- If neither the starting nor the arrival base is 10:

General solution:
- Convert N_a in base 10 with the polynomial method
- Convert the obtained result in base b with the iterated divisions method

If the arrival base is a power of the starting base ($b = a^k$):
- Convert in base b k-tuples of digits, from the less to the most signifying, of N_a
- The digits obtained in this way give the digits, from the less to the most signifying one, of the number represented in base b

If the starting base is a power of the arrival base ($a = b^k$):
- Convert in base b every digit of N_a by using k digits (of base b)

Number of binary sequences

If we have n bits, how many different binary sequences can we obtain?

1 bit:	0 ,	1	→ 2 sequences
2 bits:	00,01 ,	10,11	→ 4 sequences
3 bits:	000,001,010,011,	100,101,110,111	→ 8 sequences
4 bits:	…	…	→ 16 sequences

…

At every step, we double the sequences of the previous step:

n bits: $\underbrace{2 \cdot 2 \cdot 2 \ldots \cdot 2}_{n} = 2^n$ sequences

Representation Interval

Hence, with n bits, we can represent 2^n numbers: $\{0,\ldots,2^n-1\}$

Indeed, the smallest number is

$$\underbrace{0 \ldots 0}_{n} = \underbrace{0 + \ldots + 0}_{n} = 0$$

whereas the biggest number is

$$\underbrace{1 \ldots 1}_{n} = \sum_{i=0}^{n-1} 2^i = 2^n - 1$$ (this is the geometrical series!)

Representation Length

How many bits do we need to represent N_2?

We have to find the smallest $n \in \mathbf{N}$ such that

$$N < 2^n$$

OBS: $\log_b k$ is the exponent that must be given to the base (b) to obtain the argument (k), that is

$$k = b^{\log_b k}$$

Hence, by letting $k = N$ and $b = 2$, the solution of $N = 2^n$ is $n = \log_2 N$

OBS.: in general, n is an irrational number. Since we need a natural (and we don't need the exact equality), we take $\lfloor \log_2 N \rfloor + 1$

Examples: $N = 57$: $\log_2 N = 5{,}8328\ldots$

hence, I need 5+1 bits (indeed, $57_{10} = 111001_2$)

$N = 64$: $\log_2 N = 6$

hence, I need 6+1 bits (indeed, $64_{10} = 1000000_2$)

Codewords length

SAPIENZA
Università di Roma
Dipartimento di Informatica

For simplicity, computers work on words of fixed length, that are typically powers of 2:

- 8 bits = *byte*
- 16 bits = *half-word*
- 32 bits = *word*
- 64 bits = *long word*

Let k be the adopted codeword length:

- If a number can be represented with exactly k bits, we're OK
- If a number can be represented with less than k bits (say m), we have to add k-m 0's in the most signifying positions
- If a number needs more than k bits to be represented?
  → error situation, called *overflow*