

A new method for division and factorization based on inverse computing (Prolegomena to a physics of information)

Hans Grassmann¹
Isomorph srl

Since the times of Shannon, information theory and its applications, for instance artificial intelligence, are based on the evaluation of statistical distributions. A physics approach instead would be based on the mathematical structure of information processing. For instance, a computer program for division should be identical to the multiplication program, only evaluated in the inverse direction, from a structural point of view. A natural consequence of this view point is a new computer architecture, equivalent to the mathematical structure.

MSC 01,08,18,68,94 keywords: physics, information, mathematical structures

1) Physics and information

In spite of the fact that computers or brains are objects of the physical world, and are not of a metaphysical nature, information processing is not yet a field of physics. The variables used in information theory or information technology are not physics variables. For instance, many scientific publications are based on the assumption that “information” is identical to the physics variable “entropy”, but this cannot be true since entropy is a state function, while information is not. The first attempt towards a physics of information was made by Szillard in a discussion of the Maxwell demon in 1929 [1].

Szillard was part of a school of physics and mathematics with centers at Berlin (physics) and Goettingen (mathematics), together with colleagues like Schoeninkel, MacLane, Curry, Oppenheimer, Noether and many others [2].

In his ground breaking work Szillard used the term “information” in a purely intuitive sense, without giving a mathematical definition of the term. The next logical step in the development of this new field of science would have been to create a more profound relation to some mathematical structure. This next step was never done, not for scientific reasons, but because the school was destroyed in 1933.

2) The structure of physics

The creation of a new field of physics can be summarized in three steps:

The first step is necessarily intuitive, one starts from an intuitive understanding of the subject to be studied, like for instance of “length” or “position”.

In a second step one defines a fundamental measurement procedure which establishes when objects are identical with respect to the variable under study. For instance one defines, when two bodies are equal in length.

¹ Hans Grassmann, I-34011 Duino-Aurisina, Visogliano 9R1, grassmann@isomorph.it

Third, one chooses a mathematical structure for these variables, for instance one assumes that we live in a three dimensional Euclidean space, described by vector algebra.

As a result one gets what is referred to as “physics theory” [3], giving now a precise mathematical definition of what was necessarily intuitive at the beginning of the procedure.

In order to understand whether the theory is valid or in order to understand the limits of its validity, experiments are performed.

A physics of information would therefore deal with the mathematical structure of information processing.

One might now argue (and it has indeed been argued) that at least for all kinds of information processing performed by computing machines the mathematical structure is already known, and is given by the computing program, and that therefore there is no need and not even room for any physics of information.

3) Program and structure

M. Schoenfinkel has shown, that a function of two input parameters can always be substituted by a different function with one input parameter, which is applied twice [4]. So two computers may perform the same kind of task – meaning that they have both the same mapping of input messages to output messages -, but use different mathematical functions.

Substituting a function of several input parameters with a function of one input parameter is referred to as Currying or Schoenfinkelisation and is frequently done in modern computing [5].

Similar observations can be made also for other fundamental properties of functions. For instance, one can describe the movement of a point on a wheel rotating at constant speed by the sin and cos functions, they are non linear functions. Or instead by the polar angle, which is a linear function.

It is therefore true in more general, that a certain input may be mapped onto a certain output by means of different mathematical functions with different mathematical properties.

Therefore a better understanding of the mathematical structure of information processing systems, and therefore a physics of information, might be useful.

4) The structure of multiplication

The examples in the previous chapter still can reasonably be made based on an intuitive understanding of the term “mathematical structure”. In reality, this term is non trivial and needs consideration, as Saunders Mac Lane has discussed in one of his last articles [6] “ In this and various other aspects many familiar mathematical problems were formulated in terms of an axiomatized “structure”, and the study of such structure provides powerful tools. However, it was never the case that all of mathematics was concerned with such structure. Important classical questions about ... how are the prime numbers distributed ... are hardly structural. “.

It is now true that for instance a conventional multiplication program – calculating for instance $5 \cdot 7 = 35$ – consists of an ordered set of mathematical operations. But if one uses the term “mathematical structure” in the sense of Mac Lane, it is non trivial to assign an axiomatized “mathematical structure” to this program. For instance, the numbers “1” and “0” used by the computer could in principle be elements of F_n^2 . But in a conventional multiplication program we have for instance $1+1=2$, while instead in F_n^2 it is $1+1=0$, so F_n^2 is not the underlying structure.

Figure 1 shows how two 3 bit long numbers are multiplied in present day computing, with $\mathbf{c}=\mathbf{a}\cdot\mathbf{b}$, $\mathbf{a}=(a_3,a_2,a_1)$, $\mathbf{b}=(b_3,b_2,b_1)$, $\mathbf{c}=(c_6,c_5,c_4,c_3,c_2,c_1)$: the pairs a_i,b_j are multiplied in the order shown in the figure, for each column the resulting products are added, together with the carry from the preceding column.

The scheme of figure 1 is referred to as “long multiplication” or “standard algorithm”. There are also more advantageous methods, like the Karazuba algorithm or the Tom-Cook algorithm, they work on larger groups of bits instead of performing single-bit operations, but apart from that they follow the same logical flow as shown in figure 1.

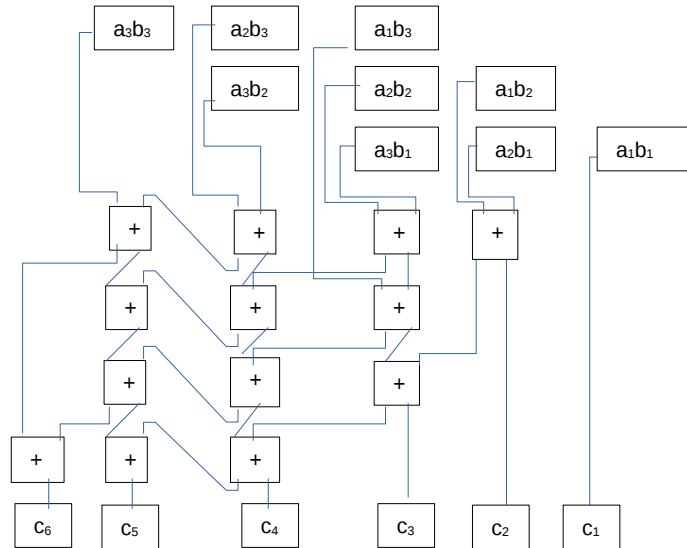


Fig. 1: traditional multiplication algorithm

The algorithm shown in figure 1 dates back to a book written by Adam Ries in late medieval times [7]. Little was known at that time about mathematical structures. Also the division algorithm used nowadays is described in this late medieval book.

One may understand multiplication as a mapping of a pair of input numbers $\mathbf{a},\mathbf{b}$ to an output number $\mathbf{c}=\mathbf{a}\cdot\mathbf{b}$. Then the sets of input and output are not isomorph and multiplication cannot be inverted.

One can instead argue that multiplication has an inverse, which is division, and obviously when one performs a division, not only the dividend is known, but also the divisor. Therefore it is appropriate to investigate the mapping of the pairs $\mathbf{a},\mathbf{b}$ to the pairs $\mathbf{c},\mathbf{a}$ or $\mathbf{c},\mathbf{b}$. These sets of numbers (which in all computer applications are finite) are isomorph so that division is the inverse of multiplication and vice versa.

A comprehensive study of these questions would go beyond the scope of this article. Here we want to limit our investigation to the following example questions:

the structure of figure 1 – be it mathematical or not – seems to correctly reproduce the map $\mathbf{a},\mathbf{b} \rightarrow \mathbf{c}$. If we substitute this structure by a well defined mathematical structure in the sense of Saunders Mac Lane – would this give new properties to multiplication programs? Would for instance multiplication become invertible? And can one build a physical device with an equivalent material structure, different therefore from the structure of figure 1 ? And can one build such a machine without quantum effects?

Again referring to the work of Saunders Mac Lane [8], a multiplication like $\mathbf{c}=\mathbf{a}\cdot\mathbf{b}$, $(c_6,c_5,c_4,c_3,c_2,c_1) = (a_3,a_2,a_1)\cdot(b_3,b_2,b_1)$ can be seen as an associative chain of morphisms, which together with the neutral element form a category:

$$\mathbf{c}=\mathbf{a}\cdot\mathbf{b} = (a_3,a_2,a_1)\cdot(0,0,b_1) + (a_3,a_2,a_1)\cdot(0,b_2,0) + (a_3,a_2,a_1)\cdot(b_3,0,0).$$

A graphical representation of this categorical structure is shown in figure 2: a first morphism starts from $\mathbf{0}$, and adds the number 0 to $\mathbf{a}\cdot(0,0,b_1)$, the result is the sum “s”. A second one adds s to $\mathbf{a}\cdot(0,b_2,0)$ and so forth. The scheme is organized in “cells” with index “n,m”, they are fundamental adders, each one receives 1 bit of input, i_{nm} , adds to it $a_n\cdot b_m$ and the carry from the previous cell, ca_{nm} , and produces an output of 1 bit, s_{nm} , with $s_{nm}=i_{nm}+a_n\cdot b_m+ca_{nm}$, and $ca_{nm}=s_{(n-1)m}a_{n-1}b_m+(s_{(n-1)m}+a_{n-1}b_m)ca_{(n-1)m}$.

(It would in principle be possible to simplify figure 2 by omitting i_{nm} since it is identical to $s_{(n-1)(m+1)}$. Instead we choose to include both the elements “i” and “s” into each cell because this puts more clearly in evidence that each morphism is a stand alone unit, that no mathematical operations are performed in between these morphisms, there is only a transmission of bits between morphisms.)

In order to make the operational structure well visible, also the fundamental adders, which process only 0’s and which do therefore not have any effect, are shown as dotted lines.

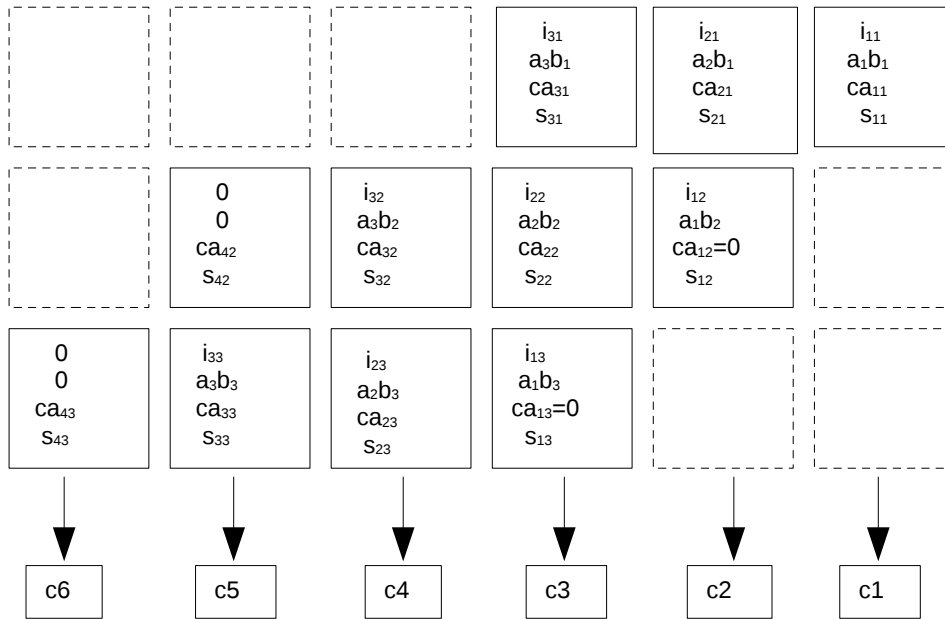


Fig. 2: multiplication represented as a category. Fundamental adders receive an input i_{nm} , they add to this input the product of $a_n\cdot b_m$ and the carry from the previous adder ca_{nm} .

Each line is a stand alone unit, which operates independent of the others, consequently the carry is calculated within each line, or horizontally, not vertically, as in figure 1.

The algebraic evaluation of this scheme is shown in figure 3.

(Since in a next step we will be interested in division, and since the division of a number ending with a “0” by 2 is trivial (shift all bits one position to the right), we assume in figure 3 that $c_1=1$ and therefore $a_1=b_1=1$, in order to keep the algebraic expressions shorter, and without limiting generality.

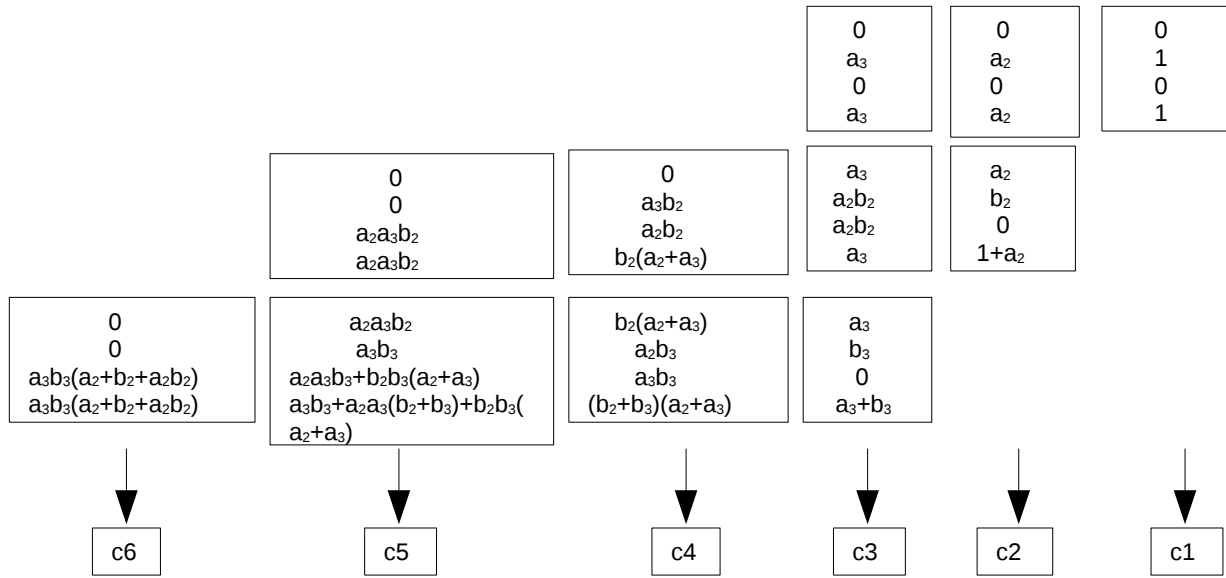


Fig. 3: execution of the categorical multiplication scheme of figure 2.

In the first line of figure 3 the first carry to the right is necessarily 0. Also all inputs “i” are 0, so that as a consequence also all carries are 0. Also in the following lines the first carry to the right is always 0. Figure 3 yields :

- I $c_2 = a_2 + b_2$
- II $c_3 = a_3 + b_3$
- III $c_4 = (b_2 + b_3)(a_2 + a_3)$
- IV $c_5 = a_3b_3 + a_2a_3(b_2 + b_3) + (a_2 + a_3)b_2b_3$
- V $c_6 = a_3b_3(a_2 + b_2 + a_2b_2)$

For instance, with $\mathbf{a} = 7 = (1, 1, 1)$ and $\mathbf{b} = 5 = (1, 0, 1) \Rightarrow \mathbf{c} = (1, 0, 0, 0, 1, 1) = 35$.

The expressions I to V obviously can also be obtained from evaluating figure 1. In that case the evaluation is however more cumbersome:

for example for calculating c_4 , in figure 3 the carry is $a_2b_2 + a_3b_3$. In figure 1 instead the carry results from the addition of a_3 , a_2b_2 , b_3 , and the carry from the previous column a_2b_2 . The carry for c_4 is therefore $a_2a_3b_2 + a_3b_3 + a_2a_3b_2 + a_2b_2b_3 + a_2b_2 + a_2b_2b_3$, instead of $a_2b_2 + a_3b_3$.

(In order to calculate the sum c_4 , the term $a_2a_3b_2 + a_3b_3 + a_2a_3b_2 + a_2b_2b_3 + a_2b_2 + a_2b_2b_3$ can be simplified to $a_2b_2 + a_3b_3$, but this simplification is not allowed for calculating the following carry, since the carry of $a+a$ is not 0 but a .)

So in practice an algebraic study of multiplication based on figure 1 is more cumbersome and this is a first difference between the structures of figure 1 and figure 2.

A more important difference is in the fact, that the operation of a physical device organized as in figure 1 cannot be inverted (public key cryptography is based in this fact). The operation of a device with the structure as in figure 2 can be inverted, as discussed in the next chapter.

5) Division as an inverted multiplication

Above expressions I and II obtain **c** from **a** and **b**, but also **b** from **c** and **a**, or **a** from **c** and **b**:

$$\text{I } c_2 = a_2 + b_2 \Rightarrow a_2 = c_2 + b_2$$

$$\text{II } c_3 = a_3 + b_3 \Rightarrow a_3 = c_3 + b_3$$

which effectively means, that division is now obtained as an inverse of multiplication.

It is also interesting to see, that the higher coefficients c_4, c_5, c_6 are not needed for calculating **a**.

(We discuss here the case of small numbers with only 6 bits in order to keep the expressions and the discussion short. Of course the procedure can be applied also to larger numbers, the next higher coefficients are $a_4 = c_4 + b_4 + c_3 b_3$ and $a_5 = c_5 + b_5 + c_4 b_4 + c_3 b_3 (c_4 + b_4)'$, where “'” indicates negation.)

In difference to figure 1, the categorical structure described in figures 2 and 3 can be inverted because it is not only true that $s = i + ca + a \cdot b$, but it is as well true that $i = s + ca + a \cdot b$. Since this is true for any cell, it is also true for the entire structure: one can start at the top of figure 3 and calculate downwards, obtaining **c** from **a** and **b**. Or instead one can start from the bottom and calculate upwards, obtaining **a** from **c** and **b** or obtaining **b** from **c** and **a**. The operational sequence for obtaining **a** from **c** and **b** is shown in figure 4.

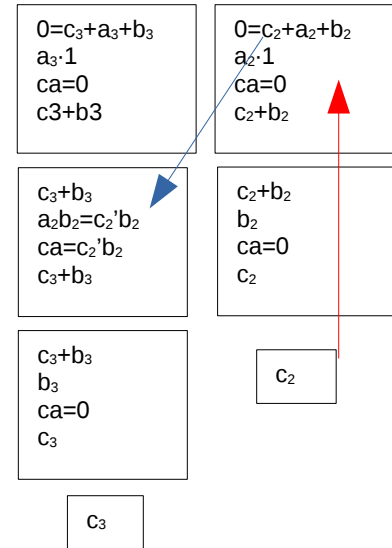


Fig. 4: the calculus shown in figure 3 performed inverted (bottom to top). In order to obtain a_3 and a_2 the first two columns are sufficient, and the columns leading to c_4, c_5, c_6 are therefore omitted in this figure. The negation is written as “'”, so c_2' is the negation of c_2 , $c_2' = 1 + c_2$.

The calculus starts below from c_2 (which is known). What was considered output in figure 2 and 3 becomes now input, so c_2 is input to the last line of the cell above. It is added to b_2 (always $a_1 = 1$), the carry in this cell is 0, and since the first line in this cell must be 0, one obtains $a_2 = c_2 + b_2$. At this point a_2 is known. This now known value of a_2 is used in the following whenever a_2 occurs in the calculation.

The calculus continues from c_3 , which is input to the cell above (again no carry) and added to b_3 , and the result $c_3 + b_3$ is handed to the cell above. This cell adds the term “ $a_2 b_2$ ” to this input and it also adds the carry from the cell to its right. For “ a_2 ” in “ $a_2 b_2$ ” one can use the previously obtained relation $a_2 = c_2 + b_2$, so that we get $a_2 b_2 = c_2' b_2$. And the carry is $b_2 (c_2 + b_2) = c_2 b_2 + b_2 = c_2' b_2$, too. Adding all these terms leads in the uppermost cell to $0 = c_3 + a_3 + b_3$ or $a_3 = c_3 + b_3$.

The procedure can be continued for numbers of any length. In appendix 1 a corresponding program is shown, it can process any number of bits.

Figure 4 can be the graphical description of an algorithm, but it can as well be read as the architecture of a hardware device, for instance built with transistors, which is able to perform division as an inverse multiplication. The device is not a Turing machine, it does not proceed in a sequence of steps. Once an input is provided in the form of electric potentials, an electromagnetic

wave will propagate throughout the circuit until it arrives at the output, “producing” at its arrival the correct output value. The speed of propagation is mainly given by the switching time of the transistors (in a transistor based device), with reference to this speed the calculation occurs on the light cone of the signal.

In order to divide a 1000 bit long number, $O(1000)$ transistors need to be switched in sequence, assuming a switching time of 10^{-11} seconds, one division of a 1000 bit long number would require $O(10^{-8})$ seconds. The device – consisting essentially of a few million transistors - can easily be built with standard microelectronic technology.

What was discussed in this chapter shows, that it can make a great difference, whether a program or a computing device just perform the numerically correct mapping from input to output – in our example $\mathbf{a} \cdot \mathbf{b} \rightarrow \mathbf{c}$. Or whether the program or device has a defined mathematical structure, corresponding to the underlying mathematics of the operation. This demonstrates the necessity of a physics of information.

This inversion of multiplication is also possible, if the result of the division has a remainder. In this case using only half of the bits is not sufficient anymore, one needs to make use of all bits. One very simple method to get $\mathbf{b}$ from $\mathbf{a}$ and $\mathbf{c}$ is as follows:

The calculus starts from below, for each line m , s_{nm} , ca_{nm} and i_{nm} are calculated from the previous line $(m+1)$, assuming first that b_m is “0”. If that choice is correct, the carry produced by the highest element in line m must be equal to the value of “i” in the second highest cell in line $(m+1)$.

If this is not the case, then the choice of “0” is wrong, and the correct value of b_m must be “1”.

It may happen, that both choices for b_m , “1” as well as “0” lead to the same value for the highest carry. The choice for b_m is then arbitrary. In this case one will find an incompatibility in the next higher lines. This allows, to go back and to correct the choice for b_m .

In this way one can proceed up to the first line. Since there is no line above, the check described cannot be applied. Instead, one finds that for the correct assignment of b_1 , in all cells “i” will have the value of “0”.

If $\mathbf{c}$ can be divided by $\mathbf{a}$ without remainder, the algorithm will again find the correct solution for $\mathbf{b}$.

For instance for $\mathbf{c} = 860411 = 11010.01000.00111.11011$ and $\mathbf{a} = 863 = 1101011111$, which is a division without remainder, one obtains correctly $\mathbf{b} = 1111100101 = 997$.

Otherwise one will find an approximation, for instance for a slightly different value of $\mathbf{c}$ like

$\mathbf{c} = 860385 = 11010.01000.00111.00001$ and again $\mathbf{a} = 863 = 1101011111$, the algorithm finds

$\mathbf{b} = 1111100100 = 996$, while the correct result would be $\mathbf{b} = 996,97$ (rounded). (In this case the program cannot find the correct choice for b_1 , so that b_1 is chosen at random).

If one wants to increase the precision of the calculus, one can add “0” bits to the number $\mathbf{c}$. For instance, adding 5 “0”-digits, so $\mathbf{c} = 11010.01000.00111.00001.00000$, one obtains $\mathbf{b} = 111110010011011$, meaning that $\mathbf{b} = 1111100100 + 11011/32 = 996.84$

A copy of the routine can be found in appendix 2, it is able to divide numbers of any length.

Again, this very simple procedure could be performed by an electronics circuit, without the technical necessities of a Turing machine.

7) Factorization as inverted multiplication

Not only the mapping $\mathbf{a}, \mathbf{b} \Rightarrow \mathbf{c}, \mathbf{b}$ is isomorph (for $\mathbf{c} = \mathbf{a} \cdot \mathbf{b}$) and therefore inverse multiplication is possible. But also the mapping $\mathbf{a}, \mathbf{b} \Rightarrow \mathbf{c}$ is isomorph, if $\mathbf{a}$ and $\mathbf{b}$ are prime numbers, and if we remove the ambiguity arising from the symmetry $\mathbf{a} \cdot \mathbf{b} = \mathbf{b} \cdot \mathbf{a}$.

Therefore it should be true, that the expressions I to V can be inverted, so that one can obtain **a** and **b** from **c** – always provided that **a** and **b** are prime numbers.

This is indeed the case, the equations I to V (shown here again for convenience), which give **c** as the result of a multiplication of **a** and **b**,

$$\text{I } c_2 = a_2 + b_2$$

$$\text{II } c_3 = a_3 + b_3$$

$$\text{III } c_4 = (b_2 + b_3)(a_2 + a_3)$$

$$\text{IV } c_5 = a_3 b_3 + a_2 a_3 (b_2 + b_3) + (a_2 + a_3) b_2 b_3$$

$$\text{V } c_6 = a_3 b_3 (a_2 + b_2 + a_2 b_2)$$

allow as well to get **a** and **b**, if **c** is given:

For instance for $c = 35 = 100011$ one gets:

$$c_2 = 1 = a_2 + b_2$$

$$c_3 = 0 = a_3 + b_3$$

$$c_4 = 0 = (c_2 + a_2 + c_3 + a_3)(a_2 + a_3) = (1 + a_2 + a_3)(a_2 + a_3)$$

$$c_5 = 0 = 0 \cdot b_2 + 0 \cdot b_3$$

$$c_6 = 1 = b_3 + (1 + 0 \cdot b_2),$$

From the last equation one gets $b_3 = c_6 = 1$, and from the second one $a_3 = c_3 + b_3 = b_3 = 1$.

The equations give no information about b_2 , since there is the order-ambiguity from the symmetry $\mathbf{a} \cdot \mathbf{b} = \mathbf{b} \cdot \mathbf{a}$. One can break this symmetry by ordering the pair **a,b**. It is not necessary to obtain a numerical ordering with for example $\mathbf{a} > \mathbf{b}$, such an ordering would not be possible, since **a** and **b** are not yet known. Any well defined mathematical ordering will do, for instance one can make use of the lowest bit for which **a** and **b** are different:

In the example $c = 35 = 100011$ the lowest bit, c_1 , is equal to “1” and therefore a_1 and b_1 must be both “1” and are therefore not different. The next higher bit, c_2 , is again “1”, but now this means that a_2 and b_2 must be different, one of them must be “1” and the other one “0”. By arbitrarily setting $b_2 = 0$ the symmetry is broken and the corresponding ambiguity vanishes. Now one obtains from V

$$c_6 = a_3 b_3 a_2, \text{ and therefore } b_3 = a_3 = a_2 = 1.$$

As long as the numbers are not too long, the procedures described here can be performed on the back of an envelope, for instance :

for $c = 10.403 = 10100010100011$, and assuming that **b** and **c** are 7 bits long:

$$c_3 = 0 = a_3 + b_3$$

$$c_4 = 0 = a_4 + b_4$$

$$c_5 = 0 = a_5 + b_5$$

$$c_6 = 1 = a_6 + b_6 + b_4 + b_3$$

$$c_7 = 0 = a_7 + b_7 + b_5 + b_4 b_6' + b_3' b_6$$

$$c_8 = 1 = (b_3 b_4)' b_6 + b_5' b_7 + b_4' b_6' b_7 + b_3 b_6 b_7 + b_4 (b_5 b_6')' + b_3' b_5 b_6'$$

$$c_9 = 0 = (b_3' b_4')' b_6 + (b_3 b_5 b_6)' b_7 + b_4 b_5' b_6 b_7 + b_4' b_5 (b_3 b_7)' + b_3 b_4 (b_6 b_7)'$$

$$c_{10} = 0 = b_4 b_6' + b_3 b_7' + (b_3 b_6)' b_5 b_7' + b_3 b_4' (b_6 b_7)' + b_4 b_5 b_6 b_7 + b_3' b_4 b_5 b_7'$$

$$c_{11} = 0 = b_4' b_5 + b_5 b_6 + b_3 b_4 b_6' + b_3' b_5 b_7 + b_3 b_4' b_5' b_6 b_7'$$

$$c_{12} = 1 = b_5 b_6 + b_3' b_6 + b_4' b_7 + (b_3 b_5)' b_4 b_6 + b_3 b_7 (b_5' b_6)' + b_3' b_5 b_7 (b_4 b_6)'$$

$$c_{13} = 0 = b_4' b_6' b_7 + b_3' b_4 b_5' b_6 + b_3 b_5 b_7 + b_4 b_5 b_7 + b_3 b_5 b_6 b_7'$$

$$c_{14} = 1 = b_3 b_6 b_7 + b_5 b_6 b_7 + b_3 b_4 b_5 b_6' b_7$$

For convenience, in the first 7 lines the trivial substitution $c_n = a_n + b_n + f_n(a_j) \Rightarrow a_n = c_n + b_n + f_n(a_j)$, $j < n$, has already been made and the values a_n obtained have been used in the equations for c_8 to c_{14} .

Directly from c_{14} follows $b_7=1$.

Next, b_3 is obtained from c_8 :

$$b_3(1+b_4b_6+b_5'b_6')=c_8+b_4b_5'b_6'+b_5b_6+b_4$$

With this expression for b_3 one gets from c_9 :

$$b_4b_6 = b_5b_6'$$

and from c_{10} : $b_6' = 0 \Rightarrow b_6=1$

now c_{11} delivers $b_5=0$, and with $b_5=0$ in $b_4b_6 = b_5b_6'$ one obtains $b_4=0$.

With these values, above equation $b_3(1+b_4b_6+b_5'b_6')=(1+b_4b_5'b_6'+b_5b_6+b_4)$ leads to $b_3 \cdot 1 = 1 \Rightarrow b_3=1$

which in summary is **b**= 1100101 = 101 (decimal)

This allows to calculate **a** from the first lines of the equation system, which leads to **a**= 1100111 = 103, and indeed $101 \cdot 103 = 10.403$.

For larger numbers, a computer program is needed.

One can as well get general expressions for **a** and **b** directly as functions of **c**. Again for a 6 bit long number:

From I follows:

$$a_2=c_2+b_2$$

From II :

$$a_3=c_3+b_3$$

One can now substitute a_2 and a_3

in III: $c_4=(b_2+b_3)(c_2+b_2+c_3+b_3)$,

$$c_4=(c_2'+c_3)b_2+(c_2+c_3')b_3$$

in IV: $c_5=(c_3+b_3)b_3+(c_2+b_2)(c_3+b_3)(b_2+b_3)+((c_2+b_2)+(c_3+b_3))b_2b_3$,

$$c_5=c_2'c_3b_2+c_2'c_3'b_3$$

and in V: $c_6=c_6=a_3b_3(a_2+b_2+a_2b_2) = (c_3+b_3)b_3((c_2+b_2)+b_2+(c_2+b_2)b_2$

$$c_6 = c_3'b_3(c_2+c_2'b_2)$$

Using the expression for c_4 , $c_4=(c_2'+c_3)b_2+(c_2+c_3')b_3$ or written as $(c_2'+c_3)b_2=c_4+(c_2+c_3')b_3$,

we can substitute b_2 in c_5 : $c_5(c_2'+c_3)=c_2'c_3(c_4+(c_2+c_3')b_3)+c_2'c_3'(c_2'+c_3)b_3$

or :

$$c_2'c_3'b_3=c_2'c_5+c_3c_5+c_2'c_3c_4 \quad (\text{IV a})$$

and in c_6 : $c_6(1+c_2'+c_3)=c_3'b_3(1+c_2'+c_3)+c_2'(c_4+(c_2+c_3')b_3)b_3$

or:

$$(c_3'+c_2'c_4)b_3 = c_2c_6+c_3c_6 \quad (\text{V a})$$

From these expressions of the type $\alpha \cdot b_n=\beta$, b_3 can be obtained directly:

For $c=35=100011$: from (IV a): $c_2'c_3'b_3=0 = c_2'c_5+c_3c_5+c_2'c_3c_4=0$

from (V a): $(c_3'+c_2'c_4)b_3 = b_3 = c_2c_6+c_3c_6 = 1$

A profound scientific discussion of this situation goes beyond this article and must be left to future research. One might suspect however, that the factor α in above expressions $\alpha \cdot b_n=\beta$ will be always 0 if **c** is not the product of two prime numbers, since in that case an inversion of multiplication cannot be possible. While instead β determines the numerical value of b_n , if the inversion is possible.

Since these equations result from the categorical structure of figure 2, a general algebraic expression for them should exist (but is not known at the time being, see above quote from Mac Lane).

Accordingly, and in analogy to what was said about division, an electronics device can be built, which factorizes the product of two (small) prime numbers in one step of operation.

Again, a computer program is needed in order to apply this procedure to larger numbers.

7) Outlook

Nowadays the most advanced computing applications in artificial intelligence are based on statistical evaluations, not on mathematical structures. Also the hardware – neural networks – does not have any mathematical structure (in the sense of Mac Lane).

A physics based information processing system instead must have the same mathematical structure as the information to be processed, as shown in this article with the example of multiplication and division.

However, as also Saunders Mac Lane has shown in his book “Mathematics Form and Function” [9], all mathematics can be grounded in the facts of the real world. In the chapter “Origins of Formal Structure” he writes “Our concern is ... the very possibility of a development of form from fact. “ Now, if the form or structure of mathematics derives from the facts of the real world, the real world in turn can necessarily be described in the framework of mathematical structure. Or said differently, the real world has a mathematical structure.

The only problem with traditional mathematics (pre-Mac Lane) is, that it is subdivided in many different fields, this makes the creation of a general information processing system at least very difficult. This practical problem is overcome with category theory, which allows to build categories which include morphisms from all fields of mathematics in one uniform structure, and to create maps between the categories of the outer world and the information processing system by means of functors.

An information processing system, for instance a system for image recognition, which is based on the mathematical structures of reality would overcome the difference between “artificial” and “natural” intelligence.

Indeed, some of the most important findings of neural science point into this direction: the mirror neurons of the brain found by G. Rizzolatti and similar systems (like Hale Berry neurons) can not be explained in present day information processing scenarios. While they fit well into a physics scenario of information processing, there they are evidence for the existence of brain structures isomorph to structures of the external world.

The physics company Isomorph srl has successfully written prototype programs based on these principles, some results can be seen on the web page of Isomorph, www.isomorph.it.

8) Conclusion

By taking into account the mathematical structure of multiplication, we are able to obtain a new and non-iterative method for performing division. A corresponding hardware device can be built easily with state of the art microelectronics, it would be able to divide a 1000 bit long number in one step of operation and within about 10^{-8} seconds.

The method can be extended to prime factorization: in this paper it is shown how to do this from an algebraic point of view, but it should also be possible to build a corresponding hardware device – this must be left to future research.

A new field of mathematics is opening up: the mathematical structure of the prime numbers. One should be confident, that this structure exists and can be given explicitly, since we have shown its existence analytically for small numbers, and because an analysis which is valid only for small numbers can hardly be imagined.

These simple examples show, that investigating the mathematical structure of information processing is useful and gives new results. As a consequence, the old field of physics of information, interrupted in 1933, should be continued.

References

- [1] Leo Szilard, "*Über die Entropieverminderung in einem thermodynamischen System bei Eingriffen intelligenter Wesen*". *Zeitschrift für Physik (in German)*. **53** (11–12): 840–856.(1929)
- [2] Saunders MacLane, *Mathematics at Goettingen under the Nazis*, Notices of the AMS, Volume 42, Number 10 (1995)
- [3] Emmy Noether, *Invariante Variationsprobleme*, *Nachr. D. König. Gesellsch. D. Wiss..* **918**. Göttingen: 235–257 (1918)
- [4] Moses Schoenfinkel, *Über die Bausteine der mathematischen Logik*. In: *Mathematische Annalen* 92 S. 305–316, (1924)
- [5] Haskell Brooks Curry, Robert Feys, Roger Hindley, Jonathan P. Seldin: *Combinatory Logic*. North Holland, 2 Bände, (1972)
- [6] Saunders MacLane, *Structure in Mathematics*, *Philosophia Mathematica* (3) Vol. 4 , pp 174-183 (1996)
- [7] Adam Ries, *Rechenung auff der linihen und federn* (1522)
- [8] Saunders MacLane, "Groups, categories and duality," *Proceedings of the Nat. Acad. of Sciences of the USA* 34: 263–67 (1948).
- [9] Saunders MacLane, "Mathematics Form and Function", Springer Verlag, New York (1986)

Appendix 1

Example for a routine which calculates $\mathbf{b}=\mathbf{c}/\mathbf{a}$ for numbers of any length (provided that $\mathbf{c}=\mathbf{a}\cdot\mathbf{b}$). The routine is a scientific toy, not written to industry standards.

```
in=1; do { in=in+1;
  m=0; do { m=m+1;
    n=in+1-m;

    if(n != 1 || m != len) {
      CARRYin[n][m]=CARRYout[n-1][m];
      SUMin[n][m]=SUMout[n+1][m-1];

      SUMout[n][m]=(((A[n]*B[m]) + SUMin[n][m])%2 + CARRYin[n][m])%2 ;
      CARRYout[n][m]=(A[n]*B[m]*SUMin[n][m])+((A[n]*B[m])+SUMin[n][m])%2*CARRYin[n][m] ; }

    if(n==1) {
      SUMin[n][m]=SUMout[n+1][m-1];

    if(m !=2 )    B[m] = (SUMin[n][m] + C[m])%2;
      CARRYout[n][m]=SUMin[n][m] * B[m];

    } } while ( m != in );
    } while ( in != len );

  in=0; do { in=in+1;
    m=in-1; do { m=m+1;
      n=len+1-m+in;

      if(n != 1 || m != len) {
        CARRYin[n][m]=CARRYout[n-1][m];
        SUMin[n][m]=SUMout[n+1][m-1];

        SUMout[n][m]=((A[n]*B[m] + SUMin[n][m])%2 + CARRYin[n][m])%2 ;
        CARRYout[n][m]=A[n]*B[m] * SUMin[n][m] + (A[n]*B[m] + SUMin[n][m])%2 * CARRYin[n]
[m] ;
      }

      if(m==len) {
        SUMin[n][m]=SUMout[n+1][m-1];
        CARRYin[n][m]=CARRYout[n-1][m];
        if( n !=2 )    A[n] = ((SUMin[n][m] + CARRYin[n][m])%2 + C[n+len-1])%2;
        CARRYout[n][m]=(SUMin[n][m]*CARRYin[n][m])+(SUMin[n][m]+CARRYin[n][m] )%2*A[n];
      } } while ( m != len );
    } while ( in != len-1 );
```

Appendix 2

```
int len=10;
C[1]=1; C[2]=1; C[4]=1; C[5]=1; C[6]=1;C[7]=1; C[8]=1; C[14]=1; C[17]=1; C[19]=1; C[20]=1;
A[1]=1; A[2]=1; A[3]=1;A[4]=1; A[5]=1; A[6]=0; A[7]=1; A[8]=0;A[9]=1;A[10]=1;

    m=0; do { m=m+1;
    SUMout[1][m] = C[m] ;
    printf("\n SUMout[1][%d] = %d",m,SUMout[1][m]);
    } while ( m != len );
    n=0; do { n=n+1;
    SUMin[n][len+1] = C[len+n] ;
    } while ( n != len );

nc=0;
m=len+1; do { m=m-1;

    labelloop;;

    n=0; do { n=n+1;
    if(n>1) CARRYin[n][m]=CARRYout[n-1][m];
    if(n>1)    SUMout[n][m] =SUMin[n-1][m+1];
    CARRYout[n][m]=A[n]*B[m] * (SUMout[n][m]+1)%2 + (A[n]*B[m] + (SUMout[n][m]+1)%2)%2
* CARRYin[n][m] ;
    SUMin[n][m]=(((A[n]*B[m]) + SUMout[n][m])%2 + CARRYin[n][m])%2 ;
    A[n]*B[m]= %d SUMin[n][m]= %d",n,m,CARRYout[n][m],CARRYin[n][m],SUMout[n]
[m],A[n]*B[m],SUMin[n][m] );
    } while ( n != len+1 );

    kflag=0;
    if(m ==1) {
    n=0; do { n=n+1;
    if(SUMin[len][m] != 0 ) kflag=1;
    } while ( n != len+1 ); }

if( (B[m] == 0 && (SUMin[len][m+1] != CARRYin[len+1][m])) || kflag ==1 ) {
    B[m] = 1;
    goto labelloop; }

    if( B[m] == 1 && (SUMin[len][m+1] != CARRYin[len+1][m]) ) {
    B[m]=0;
    m=m+1;
    B[m] = (B[m]+1)%2;
    goto labelloop; }

} while ( m !=1 );
```