# The xint bundle: xint and xintgcd

Jean-François Burnol
jfbu (at) free (dot) fr
Package version: 1.02 (2013/04/05)
Documentation generated from the source file
with timestamp "05-04-2013 at 16:49:16 CEST"

**Abstract**

The xint package implements with expandable TEX macros the basic arithmetic operations of addition, subtraction, multiplication and division, as applied to arbitrarily long numbers represented as chains of digits with an optional minus sign.

The xintgcd package provides implementations of the Euclidean algorithm and of its typesetting.

The packages may be used with Plain and with LATEX.

## Contents

# 1 Origins of this package

The package bigintcalc by Heiko Oberdiek already provides expandable arithmetic operations on "big numbers", exceeding the TEX limits (of 2^{31}-1), so why another one?

I got started on this in early March 2013, via a thread on the c.t.tex usenet group, where Ulrich Diez used the previously cited package together with a macro (\ReverseOrder) which I had contributed to another thread. [1] What I had learned in this other thread thanks to interaction with Ulrich Diez and GL on expandable manipulations of tokens motivated me to try my hands at addition and multiplication.

[1] The \ReverseOrder could be avoided in that circumstance, but it does play a crucial rôle here.

I wrote macros \bigMul and \bigAdd which I posted to the newsgroup; they appeared to work comparatively fast. These first versions did not use the ε-TEX \numexpr macro, they worked one digit at a time, having previously stored digit arithmetic in (many) macros.

I noticed that the bigintcalc package used the \numexpr ε-TEX primitive when available, but (as far as I could tell) not to do computations many digits at a time. Using \numexpr for one digit at a time for \bigAdd and \bigMul slowed them a tiny bit but avoided cluttering TEX memory with 1200 macros storing pre-computed arithmetic with 2 or 3 digits. I wondered if some speed could be gained by using \numexpr to do four digits at a time for elementary multiplications (as the maximal admissible number for \numexpr has ten digits).

The present package is the result of this initial questioning.

**xint** requires the ε-TEX \numexpr primitive.

I have aimed at speed wherever I could, and to the extent that I could guess what was more efficient for TEX. After a while though I did opt for more readable coding style in those parts of the code which were not at the heart of repeatedly used loops. In particular I started using \ifnum and \ifcase constructs which I had completely avoided so far, working only with macro expansions.

I wrote a version of addition which does \numexpr operations eight digits at a time, but its additional overhead made it a bit slower for numbers of up to a few hundreds digits and it became faster only for numbers with thousands of digits; for such sizes multiplication starts taking a noticeable time, so I have chosen to retain the addition routine which was most efficient for numbers having a few dozens to a few hundreds digits.

This implementation is thus a TEXnical thing, quite different from what one would do in a structured programming language like C, although the underlying algorithms are just the standard steps applied to hand computations (nothing fancy like Fast Fourier Transform...).

By the way, yes **xint** enjoys working fast and efficiently with 200 digits numbers, but surely any program (even poorly written) in C using the CPU for arithmetic operations on arrays of numbers (not digits!!!) will work thousands of times faster (or more, I don't know) than what can be achieved using TEX to manipulate strings of ASCII representations of digits!

## 2 Expansions

Except otherwise stated all macros are completely expandable. For example, with the following code snippet within myfile.tex

```
\newwrite\outfile
\immediate\openout\outfile \jobname-out\relax
\immediate\write\outfile {\xintQuo{\xintPow{2}{1000}}{\xintFac{100}}}
% \immediate\closeout\outfile
```

the tex run creates a file myfile-out.tex containing the decimal representation of the integer quotient 2^{1000}/100!. Such macros can also be used inside a \csname...\endcsname, and of course in an \edef.

Furthermore the package macros give their final results in two expansion steps. They twice expand their arguments so that they can be arbitrarily chained. Hence

```
\xintLen{\xintQuo{\xintPow{2}{1000}}{\xintFac{100}}}
```

expands in two steps and tells us that [2^{1000}/100!] has 144 digits. This is not so many, let us print them here: 114813249641507505482278393872551066259805517784186172883663478065826541894704737970419535798876630484358265060061503749531707793118627774829601. For the sake of typesetting this documentation and not have big numbers extend into the margin and go beyond the page physical limits, I use this little macro (not provided by the package):

```
\def\allownumbersplit #1{\ifx #1\relax \else #1\hskip 0pt plus 1pt
                         \expandafter\allownumbersplit\fi}%
```

To provoke the double expansion first, it is used as in:

```
\expandafter\expandafter\expandafter\allownumbersplit
  \xintQuo{\xintPow {2}{1000}}{\xintFac{100}}\relax
```

Or, the computation can be done inside an `\edef` and then only one `\expandafter` will be enough before `\allownumbersplit`.

Remarks on the double expansion of arguments:

1. When I say that the macros expand twice their arguments, this means that they expand the first token seen (for each argument), then expand again the first token of the result of the first expansion. For example
   ```
   \def\x{12}\def\y{34}\xintAdd {\x}{\x\y}
   ```
   is *not* a legal construct. It works here by sheer luck as the `\y` gets expanded inside a `\numexpr`. But this would fail in general: if you need a more complete (expandable...) expansion of your initial input, you should use the `\bigintcalcNum` macro from the `bigintcalc` package. Or, outside of an expandable-only context, just massage your inputs through `\edef`'s.

2. Unfortunately, after `\def\x {12}`, one can not use just `-\x` as input to one of the package macros: the rules above explain that the twice expansion will act only on the minus sign, hence do nothing. The only way is to use the `\xintOpp` macro, which replaces a number with its opposite. Example: `\xintAdd {\xintOpp\x}{\x}` = 0.

3. With the definition
   ```
   \def\AplusBC #1#2#3{\xintAdd {#1}{\xintMul {#2}{#3}}}
   ```
   one obtains an expandable macro producing the expected result, not in two, but rather in three steps: a first expansion is consumed by the macro expanding to its definition. As a result `\xintAdd {\AplusBC {1}{2}{3}}{4}` would then miserably fail. The solution is to use the *lowercase* form of `\xintAdd`:
   `\def\AplusBC #1#2#3{\romannumeral0\xintadd {#1}{\xintMul {#2}{#3}}}`
   and then `\AplusBC` will share the same properties as do the other **xint** 'primitive' macros.

   Don't leave any space after the zero, and use the lowercase form *only* for the external highest level of chained commands. All **xint** provided public macros have such a lowercase form precisely to facilitate building-up higher level macros based on them.

# 3 Inputs

After a twice expansion of the arguments, the ensuing numbers have to be strings of digits with one (and not more) optional minus sign (and not a plus sign). The first digit is not zero if there are more than one digit. And `-0` is not legal input. Syntax such as `\xintMul\A\B` is accepted and equivalent to `\xintMul {\A}{\B}`. Or course `\xintAdd\xintMul\A\B\C` does not work, the product operation must be put within braces: `\xintAdd{\xintMul\A\B}\C`.

It would be nice to have a functional form `\add(x,\mul(y,z))` but this is not provided by the package. Arguments must be either within braces or a single control sequence.

For the division (but not for addition, subtraction, or multiplication), the two inputs must have at most 2^{31}-9=2147483639 digits.

I guess anyhow that this is way way way beyond what is possible in terms of memory in any implementation of TEX. But if the situation did arise nevertheless of such a gigantic input, an arithmetic overflow would occur (after some long time I guess) as **xint** first computes the lengths of the inputs by using `\numexpr` with successive additions of the number `8` to itself until the whole input has been parsed [2] (this initial step is only for the division algorithm, the three other arithmetic operations remain unaware of the sizes of their inputs, although they do experience them in a sense, as they initially reverse the order of digits of at least one of the inputs, which means they have to scan it entirely).

Also: the factorial function `\xintFac` will refuse to (start...) compute `N!` if `N` $\geq$ 1000000000, and the power function `\xintPow {A}{B}`, when the absolute value `|A|` is at least two, will refuse to start the computation if `B` $\geq$ 1000000000 (the minimal outcome is 2^{1000000000} which has 301029996 digits...).

In those latter cases, no arithmetic overflow will happen, but rather, copied from package `bigintcalc`, undefined control sequences with names indicating the source of the problem are inserted in the token stream and will appear in the log file in TEX 'undefined macro' error messages. This will not stop the computation, which (most of the time) will output a zero.

No check is done on the format of the inputs after the initial twice expansion. Often, but not always, something starting with a `0` will be assumed to be zero (throwing the rest away, or sometimes not which then will lead to errors). Plus signs are not accepted and will cause errors.

The sole exception is the macro `\xintNum` which accepts numbers starting with an arbitrary long sequence of plus signs, minus signs, followed by zeros and will remove all of them, keeping only the correct sign:

```
\xintNum {+-+-+---++-++--0000000009876543210}=-9876543210
```

But don't insert zeros within the initial signs. An empty string is also acceptable input: `\xintNum {}=0`. As with all other package macros, `\xintNum` expands twice its argument, and obtains its final result in two expansion steps.

[2] It is the macro `\xintLen` (used by `\xintDivision`) which will trigger an arithmetic overflow if it is called with an input of more than 2147483639 digits. I thought it was not worthwile adding to the code of `\xintLen` a safeguard against the arithmetic overflow in a `\numexpr`: this check would have some general impact on speed, whereas the situation can not realistically occur (or even not at all, I admit not having double-checked the intrinsinc TEX memory limitations).

TEX's count registers cannot be directly used but must be prefixed by \the or \number. The same for \numexpr expressions.

## 4 Outputs

The output, when it consists of a single number, is always in the normalized form described in the previous section. Some macros have an output consisting of more than one number, each one is then within braces. For example \xintDivision gives first the quotient and then the remainder, each of them within braces. This is for programming purposes to avoid having to do twice the division, once for the quotient, the other one for the remainder: but of course macros \xintQuo and \xintRem are provided for easier direct access.

The macro \xintDecSplit{x}{N}[3] cuts its second argument N at a location specified by its first argument x, and returns the two pieces one after the other, each within braces. Depending on the value of x and the length of N, the first, or the second, output of \xintDecSplit may be *empty*. Leading zeros in the second string of digits are neither removed. This is the only situation where a package macro may output something which would need to be input to \xintNum before further processing by the other package macros.

When using things such as \ifcase \xintSgn{\A} one has to leave a space after the closing brace for TEX to stop its scanning for a number: once TEX has finished expanding \xintSgn{\A} and has so far obtained either 1, 0, or -1, a space (or something 'unexpandable') must stop it looking for more digits. Using \ifcase\xintSgn\A without the braces is very dangerous, because the blanks (including the end of line) following \A will be skipped and not serve to stop the number which \ifcase is looking for. With \def\A{1}:

```
\ifcase \xintSgn\A   0\or OK\else ERROR\fi   ---> gives ERROR
\ifcase \xintSgn{\A} 0\or OK\else ERROR\fi   ---> gives OK
```

## 5 Assignments

You might not need to maintain at all times complete expandability. For example why not allow oneself the two definitions \edef\A {\xintQuo{100}{3}} and \edef\B {\xintRem {100}{3}}. A special syntax is provided to make these things more efficient, as the package provides \xintDivision which computes both quotient and remainder at the same time:

```
\xintAssign\xintDivision{100}{3}\to\A\B
\xintAssign\xintDivision{\xintPow {2}{1000}}{\xintFac{100}}\to\A\B
```

gives \meaning\A: macro:->1148132496415075054822783938725510662598055177841861728836634780658265418947047379704195357988766304843582650600615037495317077931186277748296 01 and \meaning\B: macro:->5493629452133983225138128786223912807341050049847605059532189961231327664902288388132878702444582075129603152041054804964625083138567652624386837205668069376.

Another example (which uses a macro from the **xintgcd** package):

```
\xintAssign\xintBezout{357}{323}\to\A\B\U\V\D
```

[3] Its behavior has been modified in bundle version 1.01, check its documentation.

is equivalent to setting \A to 357, \B to 323, \U to -9, \V to -10, and \D to 17. And indeed (-9)×357-(-10)×323= 17 is a Bezout Identity.

```
\xintAssign\xintBezout{3570902836026}{200467139463}\to\A\B\U\V\D
```

gives then \U: macro:->5812117166, \V: macro:->103530711951 and \D=3.

When one does not know in advance the number of tokens, one can use \xintAssignArray or its synonym \xintDigitsOf:

```
\xintDigitsOf\xintPow{2}{100}\to\Out
```

This defines \Out to be macro with one parameter, \Out{0} gives the size N of the array and \Out{n}, for n from 1 to N then gives the nth element of the array, here the nth digit of 2^{100}, from the most significant to the least significant. As usual, the generated macro \Out is completely expandable and expands twice its (unique) argument. Consider the following code snippet:

```
\newcount\cnta
\newcount\cntb
\begingroup
\xintDigitsOf\xintPow{2}{100}\to\Out
\cnta = 1
\cntb = 0
\loop
\advance \cntb \xintSqr{\Out{\the\cnta}}
\ifnum \cnta < \Out{0}
\advance\cnta 1
\repeat

|2^{100}| (=\xintPow {2}{100}) has \Out{0} digits and the sum of
their squares is \the\cntb. These digits are, from the least to
the most significant: \cnta = \Out{0}
\loop \Out{\the\cnta}\ifnum \cnta > 1 \advance\cnta -1 , \repeat.
\endgroup
```

2^{100} (=1267650600228229401496703205376) has 31 digits and the sum of their squares is 679. These digits are, from the least to the most significant: 6, 7, 3, 5, 0, 2, 3, 0, 7, 6, 9, 4, 1, 0, 4, 9, 2, 2, 8, 2, 2, 0, 0, 6, 0, 5, 6, 7, 6, 2, 1.

We used a group in order to release the memory taken by the \Out array: indeed internally, besides \Out itself, additional macros are defined which are \Out0, \Out00, \Out1, \Out2, ..., \OutN, where N is the size of the array (which is the value returned by \Out{0}; the digits are parts of the names not arguments).

The command \xintRelaxArray\Out sets all these macros to \relax, but it was simpler to put everything withing a group.

Needless to say \xintAssign, \xintAssignArray and \xintDigitsOf do not do any check on whether the macros they define are already defined.

In the example above, we deliberately broke all rules of complete expandability, but had we wanted to compute the sum of the digits, not the sum of the squares, we could just have written:

```
\xintSum{\xintPow{2}{100}}=115
```

Indeed, \xintSum is usually used as in

```
\xintSum{{123}{-345}{\xintFac{7}}{\xintOpp{\xintRem{3347}{591}}}}=4426
```

but in the example above each digit of 2^{100} is treated as would have been a summand enclosed within braces, due to the rules of TEX for parsing macro arguments.

Note that `{-\xintRem{3347}{591}}` is not a valid input, because the double expansion will apply only to the minus sign and leave unaffected the `\xintRem`. So we used `\xintOpp` which replaces a number with its opposite.

As a last example of use of `\xintAssignArray` here is one line from the source code of the **xintgcd** macro `\xintTypesetEuclideAlgorithm`:

```
\xintAssignArray\xintEuclideAlgorithm {#1}{#2}\to\U
```

This is done inside a group. After this command `\U{1}` contains the number `N` of steps of the algorithm (not to be confused with `\U{0}=2N+4` which is the number of elements in the `\U` array), and the GCD is to be found in `\U{3}`, a convenient location between `\U{2}` and `\U{4}` which are (absolute values of the twice expansion of) the initial inputs. Then follow `N` quotients and remainders from the first to the last step of the algorithm. The `\xintTypesetEuclideAlgorithm` macro organizes this data for typesetting: this is just an example of one way to do it.

## 6 Error messages

We employ the same method as in the `bigintcalc` package. But the error is always thrown *before* the end of the `\romannumeral0` expansion so as to not disturb further processing of the token stream, if the operation was a secondary one whose output is expected by a first one. Here is the list of possible errors:

```
\xintError:ArrayIndexIsNegative
\xintError:ArrayIndexBeyondLimit
\xintError:FactorialOfNegativeNumber
\xintError:FactorialOfTooBigNumber
\xintError:DivisionByZero
\xintError:FractionRoundedToZero
\xintError:ExponentTooBig
\xintError:TooBigDecimalShift
\xintError:TooBigDecimalSplit
\xintError:NoBezoutForZeros
```

## 7 Package namespace

Inner macros of the **xint** and **xintgcd** packages all begin either with `\XINT@` or with `\xint@`. The package public commands all start with `\xint`. The major forms have their initials capitalized, and lowercase forms, prefixed with `\romannumeral0`, allow definitions of further macros expanding in two steps to their full expansion (and can thus be chained with the 'primitive' **xint** macros). Some other control sequence names are used only as delimiters, and left undefined.

The `\xintReverseOrder{`⟨*tokens*⟩`}` macro uses `\xint@UNDEF` and `\xint@undef` as dummy tokens and can be used on arbitrary token strings not containing these control sequence names. Anything within braces is treated as one unit: one level of exterior braces is removed and the contents are not reverted.

# 8 Loading and usage

```
Usage with LaTeX: \usepackage{xint}
                  \usepackage{xintgcd}

Usage with TeX:   \input xint.sty\relax
                  \input xintgcd.sty\relax
```

We have added, directly copied from packages by Heiko Oberdiek, a mecanism of reload and $\varepsilon$-TEX detection, especially for Plain TEX. As $\varepsilon$-TEX is required, the executable `tex` can not be used, `etex` or `pdftex` (version `1.40` or later) or ..., must be invoked.

Furthermore, the package **xintgcd** will check for previous loading of **xint**, and will try to load it if this was not already done.

Also inspired from the Heiko Oberdiek packages we have included a complete catcode protection mecanism. The packages may be loaded in any catcode configuration satisfying these requirements: the percent is comment character, the backslash is escape character, digits have category code other and letters have category code letter. Nothing else is assumed, and the previous configuration is restored after the loading of the packages.

This is for the loading of the packages. For the actual use of the macros, note that when feeding them with negative numbers the minus sign must have category code other, as is standard.

**xint** presupposes that the usual `\space` and `\empty` macros are pre-defined, which is the case in Plain TEX as well as in LATEX.

Lastly, the macros `\xintRelaxArray` (of **xint**) and `\xintTypesetEuclideAlgorithm` and `\xintTypesetBezoutAlgorithm` (of **xintgcd**) use `\loop`, both Plain and LATEX incarnations are compatible. `\xintTypesetBezoutAlgorithm` also uses the `\endgraf` macro.

# 9 Installation

```
Run tex or latex on xint.dtx.

This will extract the style files xint.sty and xintgcd.sty (and
xint.ins). Files with the same names and in the same repertory
will be overwritten. The tex (not latex) run will stop with the
complaint that it does not understand \NeedsTeXFormat, but the
style files will already have been extracted by that time.

Alternatively, run tex or latex on xint.ins if available.

To get xint.pdf run pdflatex thrice on xint.dtx

xint.sty, xintgcd.sty -> TDS:tex/generic/xint/
xint.dtx              -> TDS:source/generic/xint/
xint.pdf              -> TDS:doc/generic/xint/

It may well be necessary to then refresh the TeX installation
filename database.
```

# 10 Commands of the xint package

{N} (resp. {M} or {x}) stands for a normalised number within braces as described in the documentation, or for a control sequence expanding in at most two steps to such a number (without the braces!), or for a control sequence within braces expanding in at most two steps to such a number, of for material within braces which expands in two expansion of the first token to such a number.

## 10.1 \xintRev

\xintRev{N} will revert the order of the digits of the number, keeping the optional sign. Leading zeros resulting from the operation are not removed (see the \xintNum macro for this).

```
\xintRev{-123000}=-000321
\xintNum{\xintRev{-123000}}=-321
```

## 10.2 \xintReverseOrder

\xintReverseOrder{⟨*token_list*⟩} does not do any expansion of its argument and just reverses the order of the tokens. Brace pairs encountered are removed once and the enclosed material does not get reverted.

```
\xintReverseOrder{\xintDigitsOf\xintPow {2}{100}\to\Stuff}
    gives: \Stuff \to 1002\xintPow \xintDigitsOf
```

## 10.3 \xintNum

\xintNum{N} removes chains of plus or minus signs, followed by zeros.

```
\xintNum{+-++--+-000000000367941789479}=-367941789479
```

## 10.4 \xintLen

\xintLen{N} returns the length of the number, not counting the sign.

```
\xintLen{-12345678901234567890123456789}=29
```

## 10.5 \xintLength

\xintLength{⟨*token_list*⟩} does not do any expansion of its argument and just counts how many tokens there are. Things enclosed in braces count as one.

```
\xintLength {\xintPow {2}{100}}=3
 ≠ \xintLen {\xintPow {2}{100}}=31
```

## 10.6 \xintAssign

\xintAssign⟨*braced things*⟩\to⟨*as many cs as they are things*⟩ defines (without checking if something gets overwritten) the control sequences on the right of \to to be the complete expansions of the successive things on the left of \to enclosed within braces.

Important: a double expansion is applied first to the material extending up to \to.

As a special exception, if after this initial double expansion a brace does not immediately follows \xintAssign, it is assumed that there is only one control sequence to define and it is then defined to be the complete expansion of the material between \xintAssign and \to.

```
\xintAssign\xintDivision{1000000000000}{133333333}\to\Q\R
    \meaning\Q: macro:->7500, \meaning\R: macro:->2500
  \xintAssign\xintPow {7}{13}\to\SevenToThePowerThirteen
          \SevenToThePowerThirteen=96889010407
```

Of course this macro and its cousins completely break usage in pure expansion contexts, as assignments are made via the \edef primitive.

## 10.7 \xintAssignArray

\xintAssignArray⟨*braced things*⟩\to\myArray first double expands the first token then defines \myArray to be a macro with one parameter, such that \myArray{N} expands in two steps (which include the twice-expansion of {N}) to give the Nth braced thing, itself completely expanded. \myArray{0} returns the number M of elements of the array so that the successive elements are \myArray{1}, . . . , \myArray{M}.

```
\xintAssignArray\xintBezout {1000}{113}\to\Bez
```

will set \Bez{0} to 5, \Bez{1} to 1000, \Bez{2} to 113, \Bez{3} to -20, \Bez{4} to -177, and \Bez{5} to 1: $(-20) \times 1000 - (-177) \times 113 = 1$.

## 10.8 \xintRelaxArray

\xintRelaxArray\myArray sets to \relax all macros which were defined by the previous \xintAssignArray with \myArray as array name.

## 10.9 \xintDigitsOf

This is a synonym for \xintAssignArray, to be used to define an array giving all the digits of a given number.

```
\xintDigitsOf\xintPow {7}{500}\to\digits
```

7^500 has \digits{0}=423 digits, and the 123rd among them (starting from the most significant) is \digits{123}=3.

## 10.10 \xintSgn

\xintSgn{N} returns 1 if the number is positive, 0 if it is zero and -1 if it is negative.

## 10.11 \xintOpp

\xintOpp{N} returns the opposite -N of the number N.

## 10.12 \xintAbs

\xintAbs{N} returns the absolute value of the number.

### 10.13 \xintAdd

\xintAdd{N}{M} returns the sum of the two numbers. It is more efficient to have the longer of the two be the first argument.

### 10.14 \xintSub

\xintSub{N}{M} returns the difference N-M.

### 10.15 \xintCmp

\xintCmp{N}{M} returns 1 if N>M, 0 if N=M, and -1 if N<M.

### 10.16 \xintGeq

\xintGeq{N}{M} returns 1 if the absolute value of the first number is at least equal to the absolute value of the second number. If |N|<|M| it returns 0.

### 10.17 \xintMax

\xintMax{N}{M} returns the largest of the two in the sense of the order structure on the relative integers (*i.e.* the right-most number if they are put on a line with positive numbers on the right).

### 10.18 \xintMin

\xintMin{N}{M} returns the smallest of the two in the sense of the order structure on the relative integers (*i.e.* the left-most number if they are put on a line with positive numbers on the right).

### 10.19 \xintSum

\xintSum{⟨*braced things*⟩} after expanding its argument twice expects to find a sequence of tokens (or braced material). Each is twice-expanded, and the sum of all these numbers is returned.

```
\xintSum{{123}{-98763450}{\xintFac{7}}{\xintMul{3347}{591}}}=-96780210
                        \xintSum{1234567890}=45
```

An empty sum is no error and returns zero: \xintSum {}=0. A sum with only one term returns that number: \xintSum {{-1234}}=-1234. Attention that \xintSum {-1234} is not legal input and will may the TEX run fail. On the other hand \xintSum {1234}=10.

### 10.20 \xintSumExpr

\xintSum⟨*braced things*⟩\relax is to what \xintSum reduces after its initial double expansion of its argument.

```
\xintSumExpr {123}{-98763450}{\xintFac{7}}{\xintMul{3347}{591}}\relax=-96780210
```

## 10.21 \xintMul

`\xintMul{N}{M}` returns the product of the two numbers. The order of the numbers influences the efficiency of the computation:

1. if the shortest number has at most 4 digits, then it is (always) more efficient if it comes as the second argument,
2. if both numbers have at most 50 digits, then it is generally a bit more efficient to have the longest one be the first argument and the shortest one be the second argument,
3. if one of the number has more than 250 digits, it is always advantageous if the shortest number is the first argument (except if it has only 4 digits or less!). For example, 50 digits × 1000 digits is five times faster than 1000 digits × 50 digits,
4. if both numbers have less than 250 digits, then it is advantageous to have the shortest one be the first argument, as long as it is not too short. The limit depends on the size of the longer number; roughly, when the longer number has 100 digits, this limit on the shorter one is already of 12 digits, and the longer the long number, the lower the limit for the shorter.

So when both numbers have at most 50 digits, put the longer one first and the shorter one second; when both numbers have at least 50 digits, put the shorter one first and the longer one second. The gain will be substantial if the long number is very long.

When one of the number has at most 4 digits, always make it second. But if the shortest has 5 digits or more, it is advantageous to have it in first position when the longer number has 250 digits or more.

## 10.22 \xintSqr

`\xintSqr{N}` returns the square.

## 10.23 \xintPrd

`\xintPrd{`⟨*braced things*⟩`}` after expanding its argument twice expects to find a sequence of tokens (or braced material). Each is twice-expanded, and the product of all these numbers is returned.

```
\xintPrd{{-9876}{\xintFac{7}}{\xintMul{3347}{591}}}=-98458861798080
          \xintPrd{123456789123456789}=131681894400
```

An empty product is no error and returns 1: `\xintPrd {}=1`. A product reduced to a single term returns this number: `\xintPrd {{-1234}}=-1234`. Attention that `\xintPrd {-1234}` is not legal input and will make the TEX compilation fail. On the other hand `\xintPrd {1234}=24`.

```
                                  2^{200}3^{100}7^{100}
 =\xintPrd {{\xintPow {2}{200}}{\xintPow {3}{100}}{\xintPow {7}{100}}}
=267872793166157757576627951700754840232474026637401534897445961481542641296549949000044400724076572713000016531207640654562118014357199401590334353924402821243896682224892786298808438271613337 6
          =\xintPow {\xintMul {\xintPow {42}{9}}{43008}}{10}
```

### 10.24 \xintProductExpr

\xintProductExpr⟨*braced things*⟩\relax is to what \xintPrd reduces after its initial double expansion of its argument.

```
\xintProductExpr 123456789123456789\relax=131681894400
```

### 10.25 \xintFac

\xintFac{N} returns the factorial. It is an error if the argument is negative or at least 10^9. It is not recommended to launch the computation of things such as 100000!, if you need your computer for other tasks. On my laptop 1000! (2568 digits) is computed in a little less than ten seconds, 2000! (5736 digits) is computed in a little less than one hundred seconds, and 3000! (which has 9131 digits) needs close to seven minutes... I have no idea how much time 10000! would need (do rather 9999! if you can, the algorithm has some overhead at the transition from N=9999 to 10000 and higher; 10000! has 35660 digits). Not to mention 100000! which, from the Stirling formula, should have 456574 digits.

### 10.26 \xintPow

\xintPow{N}{M} returns N^M. When M is zero, this is 1. Some cases (N zero and M negative, |N|>1 and M negative, |N|>1 and M at least 10^9) make xint throw errors.

### 10.27 \xintDivision

\xintDivision{N}{M} returns {quotient Q}{remainder R}. This is euclidean division: N = QM + R, 0 ≤ R < |M|. So the remainder is always non-negative and the formula N = QM + R always holds independently of the signs of N or M. Division by zero is of course an error (even if N vanishes) and returns {0}{0}.

### 10.28 \xintQuo

\xintQuo{N}{M} returns the quotient from the euclidean division.

### 10.29 \xintRem

\xintRem{N}{M} returns the remainder from the euclidean division.

### 10.30 \xintFDg

\xintFDg{N} returns the first digit (most significant) of the decimal expansion.

### 10.31 \xintLDg

\xintLDg{N} returns the least significant digit. When the number is positive, this is the same as the remainder in the euclidean division by ten.

### 10.32 \xintOdd

\xintOdd{N} is 1 if the number is odd and 0 otherwise.

## 10.33 \xintDSL

`\xintDSL{N}` is decimal shift left, *i.e.* multiplication by ten.

## 10.34 \xintDSR

`\xintDSR{N}` is decimal shift right, *i.e.* it removes the last digit (keeping the sign). For a positive number, this is the same as the quotient from the euclidean division by ten (of course, done in a more efficient manner than via the general division algorithm). For `N` from `-9` to `-1`, the macro returns `0`.

## 10.35 \xintDSH

`\xintDSH{x}{N}` is parametrized decimal shift. When `x` is negative, it is like iterating `\xintDSL |x|` times (*i.e.* multiplication by `10^{-x}`). When `x` positive, it is like iterating `\DSR x` times (and is more efficient of course), and for a non-negative `N` this is thus the same as the quotient from the euclidean division by `10^x`.

## 10.36 \xintDSHr, \xintDSx

New in bundle version `1.01`.

`\xintDSHr{x}{N}` expects `x` to be zero or positive and it returns then a value `R` which is correlated to the value `Q` returned by `\xintDSH{x}{N}` in the following manner:

- if `N` is positive or zero, `Q` and `R` are the quotient and remainder in the euclidean division by `10^x` (obtained in a more efficient manner than using `\xintDivision`),
- if `N` is negative let `Q1` and `R1` be the quotient and remainder in the euclidean division by `10^x` of the absolute value of `N`. If `Q1` does not vanish, then `Q=-Q1` and `R=R1`. If `Q1` vanishes, then `Q=0` and `R=-R1`.
- for `x=0`, `Q=N` and `R=0`.

So one has `N = 10^x Q + R` if `Q` turns out to be zero or positive, and `N = 10^x Q - R` if `Q` turns out to be negative, which is exactly the case when `N` is at most `-10^x`.

`\xintDSx{x}{N}` for `x` negative is exactly as `\xintDSH{x}{N}`, *i.e.* multiplication by `10^{-x}`. For `x` zero or positive it returns the two numbers `{Q}{R}` described above, each one within braces. So `Q` is `\xintDSH{x}{N}`, and `R` is `\xintDSHr{x}{N}`, but computed simultaneously.

```
\xintAssign\xintDSx {-1}{-123456789}\to\M
\meaning\M: macro:->-1234567890.
\xintAssign\xintDSx {-20}{123456789}\to\M
\meaning\M: macro:->12345676890000000000000000000000.
\xintAssign\xintDSx {0}{-123004321}\to\Q\R
\meaning\Q: macro:->-123004321, \meaning\R: macro:->0.
\xintDSH {0}{-123004321}=-123004321, \xintDSHr {0}{-123004321}=0
\xintAssign\xintDSx {6}{-123004321}\to\Q\R
\meaning\Q: macro:->-123, \meaning\R: macro:->4321.
```

```
\xintDSH {6}{-123004321}=-123, \xintDSHr {6}{-123004321}=4321
\xintAssign\xintDSx {8}{-123004321}\to\Q\R
\meaning\Q: macro:->-1, \meaning\R: macro:->23004321.
\xintDSH {8}{-123004321}=-1, \xintDSHr {8}{-123004321}=23004321
\xintAssign\xintDSx {9}{-123004321}\to\Q\R
\meaning\Q: macro:->0, \meaning\R: macro:->-123004321.
\xintDSH {9}{-123004321}=0, \xintDSHr {9}{-123004321}=-123004321
```

## 10.37 \xintDecSplit

Modified in bundle version 1.01!

\xintDecSplit{x}{N} cuts the number into two pieces (each one within a pair of enclosing braces). First the sign if present is *removed*. Then, for x positive or null, the second piece contains the x least significant digits (*empty* if x=0) and the first piece the remaining digits (*empty* when x equals or exceeds the length of N). Leading zeros in the second piece are not removed. When x is negative the first piece contains the |x| most significant digits and the second piece the remaining digits (*empty* if x equals or exceeds the length of N). Leading zeros in this second piece are not removed. So the absolute value of the original number is always the concatenation of the first and second piece.

This macro is for use in future components of the xint bundle. Its behavior for N non-negative is final and will not change. I am still hesitant about what to do with the sign of a negative N. It is recommended to use the macro only for non-negative N until the definitive version is released.

```
             \xintAssign\xintDecSplit {0}{-123004321}\to\L\R
\meaning\L: macro:->123004321, \meaning\R: macro:->.
             \xintAssign\xintDecSplit {5}{-123004321}\to\L\R
\meaning\L: macro:->1230, \meaning\R: macro:->04321.
             \xintAssign\xintDecSplit {9}{-123004321}\to\L\R
\meaning\L: macro:->, \meaning\R: macro:->123004321.
            \xintAssign\xintDecSplit {10}{-123004321}\to\L\R
\meaning\L: macro:->, \meaning\R: macro:->123004321.
           \xintAssign\xintDecSplit {-5}{-12300004321}\to\L\R
\meaning\L: macro:->12300, \meaning\R: macro:->004321.
          \xintAssign\xintDecSplit {-11}{-12300004321}\to\L\R
\meaning\L: macro:->12300004321, \meaning\R: macro:->.
          \xintAssign\xintDecSplit {-15}{-12300004321}\to\L\R
\meaning\L: macro:->12300004321, \meaning\R: macro:->.
```

## 10.38 \xintDecSplitL

\xintDecSplitL{x}{N} returns the first piece after the action of \xintDecSplit.

## 10.39 \xintDecSplitR

\xintDecSplitR{x}{N} returns the second piece after the action of \xintDecSplit.

# 11 Commands of the xintgcd package

## 11.1 \xintGCD

\xintGCD{N}{M} computes the greatest common divisor. It is positive, except when both N and M vanish, in which case the macro returns zero.

```
\xintGCD{10000}{1113}=1
\xintGCD{123456789012345}{9876543210321}=3
```

## 11.2 \xintBezout

\xintBezout{N}{M} returns five numbers A, B, U, V, D within braces. A is the first (twice-expanded) input number, B the second, D is the GCD, and UA - VB = D.

```
\xintAssign {{\xintBezout {10000}{1113}}}\to\X
\meaning\X: macro:->{10000}{1113}{-131}{-1177}{1}.
\xintAssign {\xintBezout {10000}{1113}}\to\A\B\U\V\D
\A: 10000, \B: 1113, \U: -131, \V: -1177, \D: 1.
\xintAssign {\xintBezout {123456789012345}{9876543210321}}\to\A\B\U\V\D
\A: 123456789012345, \B: 9876543210321, \U: 256654313730, \V: 3208178892607,
\D: 3.
```

## 11.3 \xintEuclideAlgorithm

\xintEuclideAlgorithm{N}{M} applies the Euclide algorithm and keeps a copy of all quotients and remainders.

```
\xintAssign {{\xintEuclideAlgorithm {10000}{1113}}}\to\X
\meaning\X: macro:->{5}{10000}{1}{1113}{8}{1096}{1}{17}{64}{8}{2}{1}
{8}{0}
```

. The first token is the number of steps, the second is N, the third is the GCD, the fourth is M then the first quotient and remainder, the second quotient and remainder, . . . until the final quotient and last (zero) remainder.

## 11.4 \xintBezoutAlgorithm

\xintBezoutAlgorithm{N}{M} applies the Euclide algorithm and keeps a copy of all quotients and remainders. Furthermore it computes the entries of the successive products of the 2 by 2 matrices $\begin{pmatrix} q & 1 \\ 1 & 0 \end{pmatrix}$ formed from the quotients arising in the algorithm.

```
\xintAssign {{\xintEuclideAlgorithm {10000}{1113}}}\to\X
\meaning\X: macro:->{5}{10000}{0}{1}{1}{1113}{1}{0}{8}{1096}{8}{1}{1}
{17}{9}{1}{64}{8}{584}{65}{2}{1}{1177}{131}{8}{0}{10000}{1113}.
```

The first token is the number of steps, the second is N, then 0, 1, the GCD, M, 1, 0, the first quotient, the first remainder, the top left entry of the first matrix, the bottom left entry, and then these four things at each step until the end.

## 11.5 \xintTypesetEuclideAlgorithm

This macro is just an example of how to organize the data returned by \xintEuclideAlgorithm. Copy the source code to a new macro and modify it to what is needed.

```
\xintTypesetEuclideAlgorithm {123456789012345}{9876543210321}
```

$$
\begin{aligned}
123456789012345 &= 12 \times 9876543210321 + 4938270488493\\
9876543210321 &= 2 \times 4938270488493 + 2233335\\
4938270488493 &= 2211164 \times 2233335 + 536553\\
2233335 &= 4 \times 536553 + 87123\\
536553 &= 6 \times 87123 + 13815\\
87123 &= 6 \times 13815 + 4233\\
13815 &= 3 \times 4233 + 1116\\
4233 &= 3 \times 1116 + 885\\
1116 &= 1 \times 885 + 231\\
885 &= 3 \times 231 + 192\\
231 &= 1 \times 192 + 39\\
192 &= 4 \times 39 + 36\\
39 &= 1 \times 36 + 3\\
36 &= 12 \times 3 + 0
\end{aligned}
$$

### 11.6 \xintTypesetBezoutAlgorithm

This macro is just an example of how to organize the data returned by `\xintBezoutAlgorithm`. Copy the source code to a new macro and modify it to what is needed.

```
\xintTypesetBezoutAlgorithm {10000}{1113}
```

$$
\begin{aligned}
10000 &= 8 \times 1113 + 1096\\
8 &= 8 \times 1 + 0\\
1 &= 8 \times 0 + 1\\
1113 &= 1 \times 1096 + 17\\
9 &= 1 \times 8 + 1\\
1 &= 1 \times 1 + 0\\
1096 &= 64 \times 17 + 8\\
584 &= 64 \times 9 + 8\\
65 &= 64 \times 1 + 1\\
17 &= 2 \times 8 + 1\\
1177 &= 2 \times 584 + 9\\
131 &= 2 \times 65 + 1\\
8 &= 8 \times 1 + 0\\
10000 &= 8 \times 1177 + 584\\
1113 &= 8 \times 131 + 65\\
131 \times 10000 &- 1177 \times 1113 = -1
\end{aligned}
$$

## 12 Package xint implementation

The commenting of the macros is currently (2013/04/05) very sparse. Some comments may be left-overs from previous versions of the macro, with parameters in another order for example.

### 12.1 Catcodes, ε-TEX detection, reload detection

The method for package identification and reload detection is copied verbatim from the packages by Heiko Oberdiek.

The method for catcodes was also inspired by these packages, we proceed slightly differently.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode35=6    % #
  \catcode44=12   % ,
  \catcode45=12   % -
  \catcode46=12   % .
  \catcode58=12   % :
  \expandafter\let\expandafter\x\csname ver@xint.sty\endcsname
  \expandafter
    \ifx\csname PackageInfo\endcsname\relax
      \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
    \else
      \def\y#1#2{\PackageInfo{#1}{#2}}%
    \fi
  \expandafter
  \ifx\csname numexpr\endcsname\relax
     \y{xint}{\numexpr not available, aborting input}%
     \aftergroup\endinput
  \else
    \ifx\x\relax % plain-TeX, first loading
    \else
      \def\empty {}%
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
      \else
        \y{xint}{I was already loaded, aborting input}%
        \aftergroup\endinput
      \fi
    \fi
  \fi
  \def\ChangeCatcodesIfInputNotAborted
  {%
      \endgroup
      \edef\XINT@restorecatcodes@endinput
      {%
        \catcode47=\the\catcode47   % /
        \catcode41=\the\catcode41   % )
        \catcode40=\the\catcode40   % (
        \catcode42=\the\catcode42   % *
        \catcode43=\the\catcode43   % +
        \catcode62=\the\catcode62   % >
        \catcode60=\the\catcode60   % <
        \catcode58=\the\catcode58   % :
        \catcode46=\the\catcode46   % .
        \catcode45=\the\catcode45   % -
        \catcode44=\the\catcode44   % ,
```

```
51        \catcode35=\the\catcode35   % #
52        \catcode64=\the\catcode64   % @
53        \catcode125=\the\catcode125 % }
54        \catcode123=\the\catcode123 % {
55        \endlinechar=\the\endlinechar
56        \catcode13=\the\catcode13   % ^^M
57        \catcode32=\the\catcode32   %
58        \catcode61=\the\catcode61   % =
59        \noexpand\endinput
60      }%
61      \def\XINT@setcatcodes
62      {%
63        \catcode61=12   % =
64        \catcode32=10   % space
65        \catcode13=5    % ^^M
66        \endlinechar=13 %
67        \catcode123=1   % {
68        \catcode125=2   % }
69        \catcode64=11   % @
70        \catcode35=6    % #
71        \catcode44=12   % ,
72        \catcode45=12   % -
73        \catcode46=12   % .
74        \catcode58=11   % : (made letter for error cs)
75        \catcode60=12   % <
76        \catcode62=12   % >
77        \catcode43=12   % +
78        \catcode42=12   % *
79        \catcode40=12   % (
80        \catcode41=12   % )
81        \catcode47=12   % /
82      }%
83      \XINT@setcatcodes
84  }%
85 \ChangeCatcodesIfInputNotAborted
```

## 12.2 Package identification

Copied verbatim from Heiko Oberdiek's packages.

```
86 \begingroup
87   \catcode91=12 % [
88   \catcode93=12 % ]
89   \catcode58=12 % : (does not really matter, was letter)
90   \expandafter\ifx\csname ProvidesPackage\endcsname\relax
91     \def\x#1#2#3[#4]{\endgroup
92       \immediate\write-1{Package: #3 #4}%
93       \xdef#1{#4}%
94     }%
95   \else
96     \def\x#1#2[#3]{\endgroup
97       #2[{#3}]%
98       \ifx#1\@undefined
```

```
        \xdef#1{#3}%
      \fi
      \ifx#1\relax
        \xdef#1{#3}%
      \fi
    }%
  \fi
\expandafter\x\csname ver@xint.sty\endcsname
\ProvidesPackage{xint}%
  [2013/04/05 v1.02 Expandable operations on long numbers (jfB)]%
```

## 12.3 Token management macros

```
\def\xint@gobble       #1{}%
\def\xint@gobble@one   #1{}%
\def\xint@gobble@two   #1#2{}%
\def\xint@gobble@three #1#2#3{}%
\def\xint@gobble@four  #1#2#3#4{}%
\def\xint@gobble@five  #1#2#3#4#5{}%
\def\xint@gobble@six   #1#2#3#4#5#6{}%
\def\xint@gobble@seven #1#2#3#4#5#6#7{}%
\def\xint@gobble@eight #1#2#3#4#5#6#7#8{}%
\def\xint@secondoftwo  #1#2{#2}%
\def\xint@firstoftwo@andstop  #1#2{ #1}%
\def\xint@secondoftwo@andstop #1#2{ #2}%
\def\xint@exchangetwo@keepbraces #1#2{{#2}{#1}}%
\def\xint@exchangetwo@keepbraces@andstop #1#2{ {#2}{#1}}%
\def\xint@xpxp@andstop {\expandafter\expandafter\expandafter\space }%
\def\xint@r     #1\R {}%
\def\xint@w     #1\W {}%
\def\xint@z     #1\Z {}%
\def\xint@zero  #10{}%
\def\xint@one   #11{}%
\def\xint@minus #1-{}%
\def\xint@relax #1\relax {}%
\def\xint@quatrezeros #10000{}%
\def\xint@bracedundef {\xint@undef }%
\def\xint@UDzerofork      #10\dummy  #2#3\xint@UDkrof {#2}%
\def\xint@UDsignfork      #1-\dummy  #2#3\xint@UDkrof {#2}%
\def\xint@UDzerosfork     #100\dummy #2#3\xint@UDkrof {#2}%
\def\xint@UDonezerofork   #110\dummy #2#3\xint@UDkrof {#2}%
\def\xint@UDzerominusfork #10-\dummy #2#3\xint@UDkrof {#2}%
\def\xint@UDsignsfork     #1--\dummy #2#3\xint@UDkrof {#2}%
\def\xint@afterfi #1#2\fi {\fi #1}%
```

## 12.4 \xintRev, \xintReverseOrder

```
\xintRev: fait la double expansion, vérifie le signe
\xintReverseOrder: ne fait PAS la double expansion, ne regarde
PAS le signe.
```

```
\def\xintRev {\romannumeral0\xintrev }%
\def\xintrev #1%
{%
```

```
    \expandafter\expandafter\expandafter
      \xint@rev
    \expandafter\expandafter\expandafter
      {#1}%
}%
\def\xint@rev #1%
{%
    \XINT@rev@fork #1\Z
}%
\def\XINT@rev@fork #1#2%
{%
    \xint@UDsignfork
      #1\dummy \XINT@rev@negative
       -\dummy \XINT@rev@nonnegative
    \xint@UDkrof
    #1#2%
}%
\def\XINT@rev@negative #1#2\Z
{%
    \expandafter
        \space
    \expandafter
        -%
    \romannumeral0\XINT@rev {#2}%
}%
\def\XINT@rev@nonnegative #1\Z
{%
    \XINT@rev {#1}%
}%
\def\XINT@Rev {\romannumeral0\XINT@rev }%
\let\xintReverseOrder \XINT@Rev
\def\XINT@rev #1%
{%
    \XINT@rord@main {}#1%
      \xint@UNDEF
        \xint@undef\xint@undef\xint@undef\xint@undef
        \xint@undef\xint@undef\xint@undef\xint@undef
      \xint@UNDEF
}%
\def\XINT@rord@main #1#2#3#4#5#6#7#8#9%
{%
    \XINT@strip@undef #9\XINT@rord@cleanup\xint@undef
    \XINT@rord@main {#9#8#7#6#5#4#3#2#1}%
}%
\def\XINT@rord@cleanup\xint@undef\XINT@rord@main #1#2\xint@UNDEF
{%
    \expandafter\space\XINT@strip@UNDEF #1%
}%
\def\XINT@strip@undef #1\xint@undef {}%
\def\XINT@strip@UNDEF #1\xint@UNDEF {}%
```

## 12.5 \XINT@RQ

```
 cette macro renverse et ajoute le nombre minimal de zéros à
 la fin pour que la longueur soit alors multiple de 4
 \romannumeral0\XINT@RQ {}<le truc à renverser>\R\R\R\R\R\R\R\Z
```

```
\def\XINT@RQ #1#2#3#4#5#6#7#8#9%
{%
    \xint@r  #9\XINT@RQ@end\R
    \XINT@RQ {#9#8#7#6#5#4#3#2#1}%
}%
\def\XINT@RQ@end\R\XINT@RQ #1#2\Z
{%
    \XINT@RQ@end@ #1\Z
}%
\def\XINT@RQ@end@ #1#2#3#4#5#6#7#8%
{%
    \xint@r #8\XINT@RQ@end@viii
            #7\XINT@RQ@end@vii
            #6\XINT@RQ@end@vi
            #5\XINT@RQ@end@v
            #4\XINT@RQ@end@iv
            #3\XINT@RQ@end@iii
            #2\XINT@RQ@end@ii
            \R\XINT@RQ@end@i
            \Z #2#3#4#5#6#7#8%
}%
\def\XINT@RQ@end@viii #1\Z #2#3#4#5#6#7#8#9\Z { #9}%
\def\XINT@RQ@end@vii  #1\Z #2#3#4#5#6#7#8#9\Z { #8#9000}%
\def\XINT@RQ@end@vi   #1\Z #2#3#4#5#6#7#8#9\Z { #7#8#900}%
\def\XINT@RQ@end@v    #1\Z #2#3#4#5#6#7#8#9\Z { #6#7#8#90}%
\def\XINT@RQ@end@iv   #1\Z #2#3#4#5#6#7#8#9\Z { #5#6#7#8#9}%
\def\XINT@RQ@end@iii  #1\Z #2#3#4#5#6#7#8#9\Z { #4#5#6#7#8#9000}%
\def\XINT@RQ@end@ii   #1\Z #2#3#4#5#6#7#8#9\Z { #3#4#5#6#7#8#900}%
\def\XINT@RQ@end@i      \Z #1#2#3#4#5#6#7#8\Z { #1#2#3#4#5#6#7#80}%
```

## 12.6 \XINT@cuz

```
\def\xint@cleanupzeros@andstop #1#2#3#4%
{\expandafter
    \space
    \the\numexpr #1#2#3#4\relax
}%
\def\xint@cleanupzeros@nospace #1#2#3#4%
{%
    \the\numexpr #1#2#3#4\relax
}%
\def\XINT@Rev@andcleanupzeros #1%
{%
    \romannumeral0\expandafter
        \xint@cleanupzeros@andstop
    \romannumeral0\XINT@rord@main {}#1%
      \xint@UNDEF
```

```
        \xint@undef\xint@undef\xint@undef\xint@undef
        \xint@undef\xint@undef\xint@undef\xint@undef
      \xint@UNDEF
}%
 routine CleanUpZeros. Utilisée en particulier par la
 soustraction.
 INPUT:  longueur **multiple de 4**  (<-- ATTENTION)
 OUTPUT: on a retiré tous les leading zéros, on n'est **plus*
 nécessairement de longueur 4n
 Délimiteur pour @main: \W\W\W\W\W\W\W\Z avec SEPT \W

\def\XINT@cuz #1%
{%
    \XINT@cuz@loop #1\W\W\W\W\W\W\W\Z%
}%
\def\XINT@cuz@loop #1#2#3#4#5#6#7#8%
{%
    \xint@w #8\xint@cuz@enda\W
    \xint@z #8\xint@cuz@endb\Z
    \XINT@cuz@checka {#1#2#3#4#5#6#7#8}%
}%
\def\xint@cuz@enda #1\XINT@cuz@checka #2%
{%
    \xint@cuz@endaa #2%
}%
\def\xint@cuz@endaa #1#2#3#4#5\Z
{%
    \expandafter\space\the\numexpr #1#2#3#4\relax
}%
\def\xint@cuz@endb\Z\XINT@cuz@checka #1{ 0}%
\def\XINT@cuz@checka #1%
{%
    \expandafter \XINT@cuz@checkb \the\numexpr #1\relax
}%
\def\XINT@cuz@checkb #1%
{%
    \xint@zero #1\xint@cuz@backtoloop 0\XINT@cuz@Stop #1%
}%
\def\XINT@cuz@Stop #1\W #2\Z{ #1}%
\def\xint@cuz@backtoloop 0\XINT@cuz@Stop 0{\XINT@cuz@loop }%
```

## 12.7 \xintNum

```
 For example \xintNum {----+-+++---+----0000000000000003}


\def\xintNum {\romannumeral0\xintnum }%
\def\xintnum #1%
{%
 \expandafter\expandafter\expandafter
    \XINT@num
 \expandafter\expandafter\expandafter
    {#1}%
```

```
}%
\def\XINT@Num {\romannumeral0\XINT@num }%
\def\XINT@num #1{\XINT@num@loop #1\R\R\R\R\R\R\R\R\Z }%
\def\XINT@num@loop #1#2#3#4#5#6#7#8%
{%
    \xint@r #8\XINT@num@end\R\XINT@num@NumEight #1#2#3#4#5#6#7#8%
}%
\def\XINT@num@end\R\XINT@num@NumEight #1\R #2\Z
{%
    \expandafter\space\the\numexpr #1+0\relax
}%
\def\XINT@num@NumEight #1#2#3#4#5#6#7#8%
{%
    \ifnum \numexpr #1#2#3#4#5#6#7#8+0\relax = 0
      \xint@afterfi {\expandafter\XINT@num@keepsign@a
                     \the\numexpr #1#2#3#4#5#6#7#81\relax}%
    \else
      \xint@afterfi {\expandafter\XINT@num@finish
                     \the\numexpr #1#2#3#4#5#6#7#8\relax}%
    \fi
}%
\def\XINT@num@keepsign@a #1%
{%
    \xint@one#1\XINT@num@gobacktoloop 1\XINT@num@keepsign@b
}%
\def\XINT@num@gobacktoloop 1\XINT@num@keepsign@b {\XINT@num@loop }%
\def\XINT@num@keepsign@b #1{\XINT@num@loop -}%
\def\XINT@num@finish #1\R #2\Z { #1}%
```

## 12.8 \xintLen, \xintLength

```
\xintLen    -> fait la double expansion, ne compte PAS le signe
\xintLength -> ne fait PAS la double expansion, compte le signe
```

```
\def\xintLen {\romannumeral0\xintlen }%
\def\xintlen #1%
{%
    \expandafter\expandafter\expandafter
    \XINT@length@fork #1\R\R\R\R\R\R\R\Z
}%
\def\XINT@Len #1{\romannumeral0\XINT@length@fork #1\R\R\R\R\R\R\R\R\Z }%
\def\XINT@length@fork #1%
{%
    \expandafter\XINT@length@loop
    \xint@UDsignfork
      #1\dummy {{0}}%
       -\dummy {{0}#1}%
    \xint@UDkrof
}%
\def\XINT@Length #1{\romannumeral0\XINT@length@loop {0}#1\R\R\R\R\R\R\R\R\Z }%
\def\XINT@length #1{\XINT@length@loop {0}#1\R\R\R\R\R\R\R\R\Z }%
\let\xintLength\XINT@Length
\def\XINT@length@loop #1#2#3#4#5#6#7#8#9%
```

```
{%
    \xint@r  #9\XINT@length@end {#2#3#4#5#6#7#8#9}\R
    \expandafter\XINT@length@loop\expandafter {\the\numexpr #1+8\relax}%
}%
\def\XINT@length@end #1\R\expandafter\XINT@length@loop\expandafter #2#3\Z
{%
    \XINT@length@end@ #1\W\W\W\W\W\W\W\Z {#2}%
}%
\def\XINT@length@end@ #1\R #2#3#4#5#6#7#8#9\Z
{%
    \xint@w #2\XINT@length@end@i
            #3\XINT@length@end@ii
            #4\XINT@length@end@iii
            #5\XINT@length@end@iv
            #6\XINT@length@end@v
            #7\XINT@length@end@vi
            #8\XINT@length@end@vii
            \W\XINT@length@end@viii
}%
\def\XINT@length@end@viii #1%
    {\expandafter\space\the\numexpr #1-8\relax}%
\def\XINT@length@end@vii  #1\XINT@length@end@viii #2%
    {\expandafter\space\the\numexpr #2-7\relax}%
\def\XINT@length@end@vi   #1\XINT@length@end@viii #2%
    {\expandafter\space\the\numexpr #2-6\relax}%
\def\XINT@length@end@v    #1\XINT@length@end@viii #2%
    {\expandafter\space\the\numexpr #2-5\relax}%
\def\XINT@length@end@iv   #1\XINT@length@end@viii #2%
    {\expandafter\space\the\numexpr #2-4\relax}%
\def\XINT@length@end@iii  #1\XINT@length@end@viii #2%
    {\expandafter\space\the\numexpr #2-3\relax}%
\def\XINT@length@end@ii   #1\XINT@length@end@viii #2%
    {\expandafter\space\the\numexpr #2-2\relax}%
\def\XINT@length@end@i    #1\XINT@length@end@viii #2%
    {\expandafter\space\the\numexpr #2-1\relax}%
```

## 12.9 \xintAssign, \xintAssignArray, \xintDigitsOf

```
\xintAssign {a}{b}..{z}\to\A\B...\Z,
\xintAssignArray {a}{b}..{z}\to\U
version 1.01 corrects an oversight in 1.0 related to the value of
\escapechar at the time of using \xintAssignArray or \xintRelaxArray
```

```
\def\xintAssign #1\to
{%
    \expandafter\expandafter\expandafter
    \XINT@assign@a #1{}\to
}%
\def\XINT@assign@a #1% attention to the # at the beginning of next line
#{%
    \def\xint@temp {#1}%
    \ifx\empty\xint@temp
        \expandafter\XINT@assign@b
```

```
    \else
        \expandafter\XINT@assign@B
    \fi
}%
\def\XINT@assign@b #1#2\to #3%
{%
    \edef #3{#1}\def\xint@temp {#2}%
    \ifx\empty\xint@temp
      \else
      \xint@afterfi{\XINT@assign@a #2\to }%
    \fi
}%
\def\XINT@assign@B #1\to #2%
{%
    \edef #2{\xint@temp}%
}%
\def\xintRelaxArray #1%
{%
    \edef\XINT@restoreescapechar {\escapechar\the\escapechar\relax}%
    \escapechar -1
    \edef\xint@arrayname {\string #1}%
    \XINT@restoreescapechar
    \expandafter\let\expandafter\xint@temp
                \csname\xint@arrayname 0\endcsname
    \count 255 0
    \loop
      \global\expandafter\let
             \csname\xint@arrayname\the\count255\endcsname\relax
      \ifnum \count 255 < \xint@temp
      \advance\count 255 1
    \repeat
    \global\expandafter\let\csname\xint@arrayname 00\endcsname\relax
    \global\let #1\relax
}%
\def\xintAssignArray #1\to #2%
{%
    \edef\XINT@restoreescapechar {\escapechar\the\escapechar\relax}%
    \escapechar -1
    \edef\xint@arrayname {\string #1}%
    \XINT@restoreescapechar
    \count 255 0
        \expandafter\expandafter\expandafter
    \XINT@assignarray@loop #1\xint@undef
    \csname\xint@arrayname 00\endcsname
    \csname\xint@arrayname 0\endcsname
    {\xint@arrayname}%
    #2%
}%
\def\XINT@assignarray@loop #1%
{%
    \def\xint@temp {#1}%
    \ifx\xint@bracedundef\xint@temp
```

```
        \edef\xint@temp{\the\count 255 }%
        \expandafter\let\csname\xint@arrayname0\endcsname\xint@temp
        \expandafter\XINT@assignarray@end
    \else
        \advance\count 255 1
        \expandafter\edef
           \csname\xint@arrayname\the\count 255\endcsname{\xint@temp}%
        \expandafter\XINT@assignarray@loop
    \fi
}%
\def\XINT@assignarray@end {\expandafter\XINT@assignarray@@end  }%
\def\XINT@assignarray@@end #1%
{%
    \expandafter\XINT@assignarray@@@end\expandafter #1%
}%
\def\XINT@assignarray@@@end #1#2#3%
{%
    \expandafter\XINT@assignarray@@@@end
    \expandafter #1\expandafter #2\expandafter{#3}%
}%
\def\XINT@assignarray@@@@end #1#2#3#4%
{%
    \def #4##1%
    {\romannumeral0%
        \expandafter\expandafter\expandafter
          #1%
        \expandafter\expandafter\expandafter
          {##1}%
    }%
    \def #1##1%
    {%
        \ifnum ##1< 0
            \xint@afterfi {\xintError:ArrayIndexIsNegative
                           \expandafter\space 0}%
        \else
            \xint@afterfi {%
              \ifnum ##1> #2
                  \xint@afterfi {\xintError:ArrayIndexBeyondLimit
                                 \expandafter\space 0}%
              \else
                  \xint@afterfi
                  {\expandafter\expandafter\expandafter
                   \space\csname #3##1\endcsname}%
              \fi}%
        \fi
     }%
}%
\let\xintDigitsOf\xintAssignArray
```

## 12.10 \xintSgn

```
\def\xintSgn {\romannumeral0\xintsgn }%
```

```
\def\xintsgn #1%
{%
    \expandafter\expandafter\expandafter
    \XINT@sgn #1\Z%
}%
\def\XINT@Sgn #1{\romannumeral0\XINT@sgn #1\Z }%
\def\XINT@sgn #1%
{%
    \xint@xpxp@andstop
    \xint@UDzerominusfork
      #1-\dummy  {\expandafter0}%  zero
      0#1\dummy  {\expandafter-\expandafter1}% n\'egatif
       0-\dummy  {\expandafter1}%  positif
    \xint@UDkrof
    \xint@z
}%
```

## 12.11 \xintOpp

```
\def\xintOpp {\romannumeral0\xintopp }%
\def\xintopp #1%
{%
    \expandafter\expandafter\expandafter
        \XINT@opp #1%
}%
\def\XINT@Opp #1{\romannumeral0\XINT@opp #1}%
\def\XINT@opp #1%
{%
    \expandafter\space
    \xint@UDzerominusfork
      #1-\dummy  0%      zero
      0#1\dummy  {}%     negative
       0-\dummy  {-#1}%  positive
    \xint@UDkrof
}%
```

## 12.12 \xintAbs

```
\def\xintAbs {\romannumeral0\xintabs }%
\def\xintabs #1%
{%
    \expandafter\expandafter\expandafter
    \XINT@abs #1%
}%
\def\XINT@Abs {\romannumeral0\XINT@abs }%
\def\XINT@abs #1%
{%
    \xint@UDsignfork
      #1\dummy  \space
       -\dummy  { #1}%
    \xint@UDkrof
}%
```

---

---

```
  ARITHMETIC OPERATIONS: ADDITION, SUBTRACTION, SUMS,
  MULTIPLICATION, PRODUCTS, FACTORIAL, POWERS, EUCLIDEAN DIVISION.
```

## 12.13 \xintAdd

```
\def\xintAdd {\romannumeral0\xintadd }%
\def\xintadd #1%
{%
    \expandafter\expandafter\expandafter
           \xint@add
    \expandafter\expandafter\expandafter
           {#1}%
}%
\def\xint@add #1#2%
{%
    \expandafter\expandafter\expandafter
    \XINT@add@fork #2\Z #1\Z
}%
\def\XINT@Add #1#2{\romannumeral0\XINT@add@fork #2\Z #1\Z }%
\def\XINT@add #1#2{\XINT@add@fork #2\Z #1\Z }%
     ADDITION
  Ici #1#2 vient du *deuxième* argument de \xintAdd
  et  #3#4 donc du *premier* [algo plus efficace lorsque
  le premier est plus long que le second]

\def\XINT@add@fork #1#2\Z #3#4\Z
{%
    \xint@UDzerofork
      #1\dummy \XINT@add@secondiszero
      #3\dummy \XINT@add@firstiszero
       0\dummy
        {\xint@UDsignsfork
           #1#3\dummy \XINT@add@minusminus          % #1 = #3 = -
            #1-\dummy \XINT@add@minusplus          % #1 = -
            #3-\dummy \XINT@add@plusminus          % #3 = -
             --\dummy \XINT@add@plusplus
         \xint@UDkrof }%
    \xint@UDkrof
    {#2}{#4}#1#3%
}%
\def\XINT@add@secondiszero #1#2#3#4{ #4#2}%
\def\XINT@add@firstiszero #1#2#3#4{ #3#1}%

  #1 vient du *deuxième* et #2 vient du *premier*

\def\XINT@add@minusminus #1#2#3#4%
{%
    \expandafter\space\expandafter-%
    \romannumeral0\XINT@add@pre {#2}{#1}%
}%
\def\XINT@add@minusplus #1#2#3#4%
{%
    \XINT@sub@pre {#4#2}{#1}%
}%
```

```
\def\XINT@add@plusminus #1#2#3#4%
{%
    \XINT@sub@pre {#3#1}{#2}%
}%
\def\XINT@add@plusplus #1#2#3#4%
{%
    \XINT@add@pre {#4#2}{#3#1}%
}%
\def\XINT@add@pre #1%
{%
  \expandafter\XINT@add@@pre\expandafter{%
  \romannumeral0\XINT@RQ {}#1\R\R\R\R\R\R\R\R\Z
  }%
}%
\def\XINT@add@@pre #1#2%
{%
    \expandafter\XINT@add@A
        \expandafter0\expandafter{\expandafter}%
    \romannumeral0\XINT@RQ {}#2\R\R\R\R\R\R\R\R\Z
        \W\X\Y\Z #1\W\X\Y\Z
}%
```

```
ADDITION \XINT@add@A
INPUT:
\romannumeral0\XINT@add@A <N1>\W\X\Y\Z <N2>\W\X\Y\Z
avec: N1 et N2 sur **4n**, et **renversés**, et le plus long ne
doit pas se terminer par 0000. [Donc on peut avoir 0000 comme
input si l'autre est >0 et ne se termine pas en 0000 bien sûr].
OUTPUT:
La somme N1+N2, *PAS* sur 4n, dans l'ordre *normal*, et *sans
leading zeros*
La procédure est plus rapide lorsque la longueur de N2 est
supérieure à celle de N1
```

```
\def\XINT@add@A #1#2#3#4#5#6%
{%
    \xint@w
    #3\xint@add@az
    \W\XINT@add@AB #1{#3#4#5#6}{#2}%
}%
```

```
1er nombre fini.
```

```
\def\xint@add@az\W\XINT@add@AB #1#2%
{%
    \XINT@add@AC@checkcarry #1%
}%
```

```
ici #2 est prévu pour l'addition, mais attention il devra être renversé pour
\numexpr. #3 = résultat partiel. #4 = chiffres qui restent
```

```
\def\XINT@add@AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
{%
    \xint@w
    #5\xint@add@bz
    \W\XINT@add@ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
}%
\def\XINT@add@ABE #1#2#3#4#5#6%
{\expandafter
    \XINT@add@ABEA\the\numexpr #1+10#5#4#3#2+#6\relax.%
}%
\def\XINT@add@ABEA #1#2#3.#4%
{%
    \XINT@add@A  #2{#3#4}%
}%

 ici le deuxième nombre est fini
 #6 part à la poubelle, #2#3#4#5 est le #2 dans \XINT@add@AB
 on ne vérifie pas la retenue cette fois, mais les fois suivantes

\def\xint@add@bz\W\XINT@add@ABE #1#2#3#4#5#6%
{\expandafter
    \XINT@add@CC\the\numexpr #1+10#5#4#3#2\relax.%
}%
\def\XINT@add@CC #1#2#3.#4%
{%
    \XINT@add@AC@checkcarry  #2{#3#4}% on va examiner et \'eliminer #2
}%

 retenue plus chiffres qui restent de l'un des deux nombres.
 #2 = résultat partiel
 #3#4#5#6 = summand, avec plus significatif à droite

\def\XINT@add@AC@checkcarry #1%
{%
    \xint@zero #1\xint@add@AC@nocarry 0\XINT@add@C
}%
\def\xint@add@AC@nocarry 0\XINT@add@C #1#2\W\X\Y\Z
{%
    \expandafter
    \xint@cleanupzeros@andstop
    \romannumeral0%
    \XINT@rord@main {}#2%
      \xint@UNDEF
        \xint@undef\xint@undef\xint@undef\xint@undef
        \xint@undef\xint@undef\xint@undef\xint@undef
      \xint@UNDEF
    #1%
}%
\def\XINT@add@C #1#2#3#4#5%
{%
    \xint@w
    #2\xint@add@cz
    \W\XINT@add@CD {#5#4#3#2}{#1}%
}%
```

```
\def\XINT@add@CD #1%
{\expandafter
    \XINT@add@CC\the\numexpr 1+10#1\relax.%
}%
\def\xint@add@cz\W\XINT@add@CD #1#2{ 1#2}%
```

## 12.14 \xintSub

```
\def\xintSub {\romannumeral0\xintsub }%
\def\xintsub #1%
{%
    \expandafter\expandafter\expandafter
            \xint@sub
    \expandafter\expandafter\expandafter
            {#1}%
}%
\def\xint@sub #1#2%
{%
    \expandafter\expandafter\expandafter
    \XINT@sub@fork #2\Z #1\Z
}%
\def\XINT@Sub #1#2{\romannumeral0\XINT@sub@fork #2\Z #1\Z }%
\def\XINT@sub #1#2{\XINT@sub@fork #2\Z #1\Z }%
 SOUSTRACTION
 #3#4-#1#2
 #3#4 vient du *premier*
 #1#2 vient du *second*

\def\XINT@sub@fork #1#2\Z #3#4\Z
{%
    \xint@UDsignsfork
           #1#3\dummy \XINT@sub@minusminus
            #1-\dummy \XINT@sub@minusplus   % attention, #3=0 possible
            #3-\dummy \XINT@sub@plusminus   % attention, #1=0 possible
             --\dummy {\xint@UDzerofork
                       #1\dummy \XINT@sub@secondiszero
                       #3\dummy \XINT@sub@firstiszero
                        0\dummy \XINT@sub@plusplus
                       \xint@UDkrof }%
    \xint@UDkrof
    {#2}{#4}#1#3%
}%
\def\XINT@sub@secondiszero #1#2#3#4{ #4#2}%
\def\XINT@sub@firstiszero  #1#2#3#4{ -#3#1}%
\def\XINT@sub@plusplus #1#2#3#4%
{%
    \XINT@sub@pre {#4#2}{#3#1}%
}%
\def\XINT@sub@minusminus #1#2#3#4%
{%
    \XINT@sub@pre {#1}{#2}%
}%
\def\XINT@sub@minusplus #1#2#3#4%
```

```
{%
    \xint@zero #4\xint@sub@mp0\XINT@add@pre {#4#2}{#1}%
}%
\def\xint@sub@mp0\XINT@add@pre #1#2{ #2}%
\def\XINT@sub@plusminus #1#2#3#4%
{%
    \xint@zero #3\xint@sub@pm0\expandafter\space\expandafter-%
    \romannumeral0\XINT@add@pre {#2}{#3#1}%
}%
\def\xint@sub@pm #1\XINT@add@pre #2#3{ -#2}%
\def\XINT@sub@pre #1%
{%
  \expandafter\XINT@sub@@pre\expandafter{%
  \romannumeral0\XINT@RQ {}#1\R\R\R\R\R\R\R\R\Z
  }%
}%
\def\XINT@sub@@pre #1#2%
{%
    \expandafter\XINT@sub@A
        \expandafter1\expandafter{\expandafter}%
    \romannumeral0\XINT@RQ {}#2\R\R\R\R\R\R\R\R\Z
        \W\X\Y\Z #1 \W\X\Y\Z
}%
```

```
\romannumeral0\XINT@subA 1{}<N1>\W\X\Y\Z<N2>\W\X\Y\Z
N1 et N2 sont présentés à l'envers ET ON A RAJOUTÉ DES ZÉROS
POUR QUE LEURS LONGUEURS À CHACUN SOIENT MULTIPLES DE 4, MAIS
AUCUN NE SE TERMINE EN 0000
output: N2 - N1
Elle donne le résultat dans le **bon ordre**, avec le bon signe,
et sans zéros superflus.
```

```
\def\XINT@sub@A #1#2#3\W\X\Y\Z #4#5#6#7%
{%
    \xint@w
    #4\xint@sub@az
    \W\XINT@sub@B #1{#4#5#6#7}{#2}#3\W\X\Y\Z
}%
\def\XINT@sub@B #1#2#3#4#5#6#7%
{%
    \xint@w
    #4\xint@sub@bz
    \W\XINT@sub@onestep #1#2{#7#6#5#4}{#3}%
}%
```

```
d'abord la branche principale
#6 = 4 chiffres de N1, plus significatif en *premier*,
#2#3#4#5 chiffres de N2, plus significatif en *dernier*
On veut N2 - N1.
```

```
\def\XINT@sub@onestep #1#2#3#4#5#6%
{\expandafter
    \XINT@sub@backtoA\the\numexpr 11#5#4#3#2-#6+#1-1\relax.%
}%
```

```
  ON PRODUIT LE RÉSULTAT DANS LE BON ORDRE
\def\XINT@sub@backtoA #1#2#3.#4%
{%
    \XINT@sub@A #2{#3#4}%
}%
\def\xint@sub@bz
    \W\XINT@sub@onestep #1#2#3#4#5#6#7%
{%
    \xint@UDzerofork
      #1\dummy  \XINT@sub@C   % une retenue
       0\dummy  \XINT@sub@D   % pas de retenue
    \xint@UDkrof
    {#7}#2#3#4#5%
}%
\def\XINT@sub@D #1#2\W\X\Y\Z
{%
    \expandafter
    \xint@cleanupzeros@andstop
    \romannumeral0%
    \XINT@rord@main {}#2%
      \xint@UNDEF
        \xint@undef\xint@undef\xint@undef\xint@undef
        \xint@undef\xint@undef\xint@undef\xint@undef
      \xint@UNDEF
    #1%
}%
\def\XINT@sub@C #1#2#3#4#5%
{%
    \xint@w
    #2\xint@sub@cz
    \W\XINT@sub@AC@onestep {#5#4#3#2}{#1}%
}%
\def\XINT@sub@AC@onestep #1%
{\expandafter
    \XINT@sub@backtoC\the\numexpr 11#1-1\relax.%
}%
\def\XINT@sub@backtoC #1#2#3.#4%
{%
    \XINT@sub@AC@checkcarry #2{#3#4}% la retenue va \^etre examin\'ee
}%
\def\XINT@sub@AC@checkcarry #1%
{%
    \xint@one #1\xint@sub@AC@nocarry 1\XINT@sub@C
}%
\def\xint@sub@AC@nocarry 1\XINT@sub@C #1#2\W\X\Y\Z
{%
    \expandafter
    \XINT@cuz@loop
    \romannumeral0%
    \XINT@rord@main {}#2%
      \xint@UNDEF
```

```
        \xint@undef\xint@undef\xint@undef\xint@undef
        \xint@undef\xint@undef\xint@undef\xint@undef
      \xint@UNDEF
    #1\W\W\W\W\W\W\W\Z
}%
\def\xint@sub@cz\W\XINT@sub@AC@onestep #1%
{%
    \XINT@cuz
}%
\def\xint@sub@az\W\XINT@sub@B #1#2#3#4#5#6#7%
{%
    \xint@w
    #4\xint@sub@ez
    \W\XINT@sub@Eenter #1{#3}#4#5#6#7%
}%
```

le premier nombre continue, le résultat sera < 0.

```
\def\XINT@sub@Eenter #1#2%
{%
    \expandafter
    \XINT@sub@E\expandafter1\expandafter{\expandafter}%
    \romannumeral0%
    \XINT@rord@main {}#2%
      \xint@UNDEF
        \xint@undef\xint@undef\xint@undef\xint@undef
        \xint@undef\xint@undef\xint@undef\xint@undef
      \xint@UNDEF
    \W\X\Y\Z #1%
}%
\def\XINT@sub@E #1#2#3#4#5#6%
{%
    \xint@w #3\xint@sub@F\W\XINT@sub@Eonestep
    #1{#6#5#4#3}{#2}%
}%
\def\XINT@sub@Eonestep #1#2%
{\expandafter
    \XINT@sub@backtoE\the\numexpr 110000-#2+#1-1\relax.%
}%
\def\XINT@sub@backtoE #1#2#3.#4%
{%
    \XINT@sub@E #2{#3#4}%
}%
\def\xint@sub@F\W\XINT@sub@Eonestep #1#2#3#4%
{%
    \xint@UDonezerofork
      #4#1\dummy {\XINT@sub@Fdec 0}% soustraire  1. Et faire signe -
      #1#4\dummy {\XINT@sub@Finc 1}% additionner 1. Et faire signe -
        10\dummy  \XINT@sub@DD   % terminer. Mais avec signe -
    \xint@UDkrof
    {#3}%
}%
\def\XINT@sub@DD
```

```
{\expandafter\space\expandafter-\romannumeral0\XINT@sub@D }%
\def\XINT@sub@Fdec #1#2#3#4#5#6%
{%
    \xint@w
    #3\xint@sub@Fdec@finish\W\XINT@sub@Fdec@onestep
    #1{#6#5#4#3}{#2}%
}%
\def\XINT@sub@Fdec@onestep #1#2%
{\expandafter
    \XINT@sub@backtoFdec\the\numexpr 11#2+#1-1\relax.%
}%
\def\XINT@sub@backtoFdec #1#2#3.#4%
{%
    \XINT@sub@Fdec #2{#3#4}%
}%
\def\xint@sub@Fdec@finish\W\XINT@sub@Fdec@onestep #1#2%
{%
    \expandafter\space\expandafter-\romannumeral0\XINT@cuz
}%
\def\XINT@sub@Finc #1#2#3#4#5#6%
{%
    \xint@w
    #3\xint@sub@Finc@finish\W\XINT@sub@Finc@onestep
    #1{#6#5#4#3}{#2}%
}%
\def\XINT@sub@Finc@onestep #1#2%
{\expandafter
    \XINT@sub@backtoFinc\the\numexpr 10#2+#1\relax.%
}%
\def\XINT@sub@backtoFinc #1#2#3.#4%
{%
    \XINT@sub@Finc #2{#3#4}%
}%
\def\xint@sub@Finc@finish\W\XINT@sub@Finc@onestep #1#2#3%
{%
    \xint@UDzerofork
     #1\dummy {\expandafter\space\expandafter-%
               \xint@cleanupzeros@nospace}%
      0\dummy { -1}%
    \xint@UDkrof
    #3%
}%
\def\xint@sub@ez\W\XINT@sub@Eenter #1%
{%
    \xint@UDzerofork
      #1\dummy \XINT@sub@K             %     il y a une retenue
       0\dummy \XINT@sub@L             %     pas de retenue
    \xint@UDkrof
}%
\def\XINT@sub@L #1\W\X\Y\Z
   {\XINT@cuz@loop #1\W\W\W\W\W\W\W\Z }%
\def\XINT@sub@K #1%
```

```
{%
    \expandafter
    \XINT@sub@KK\expandafter1\expandafter{\expandafter}%
    \romannumeral0%
    \XINT@rord@main {}#1%
      \xint@UNDEF
        \xint@undef\xint@undef\xint@undef\xint@undef
        \xint@undef\xint@undef\xint@undef\xint@undef
      \xint@UNDEF
}%
\def\XINT@sub@KK #1#2#3#4#5#6%
{%
    \xint@w
    #3\xint@sub@KK@finish\W\XINT@sub@KK@onestep
    #1{#6#5#4#3}{#2}%
}%
\def\XINT@sub@KK@onestep #1#2%
{\expandafter
    \XINT@sub@backtoKK\the\numexpr 110000-#2+#1-1\relax.%
}%
\def\XINT@sub@backtoKK #1#2#3.#4%
{%
    \XINT@sub@KK #2{#3#4}%
}%
\def\xint@sub@KK@finish\W\XINT@sub@KK@onestep #1#2#3%
{%
    \expandafter\space\expandafter-\romannumeral
    0\XINT@cuz@loop #3\W\W\W\W\W\W\W\Z
}%
```

## 12.15 \xintCmp

```
\def\xintCmp {\romannumeral0\xintcmp }%
\def\xintcmp #1%
{%
    \expandafter\expandafter\expandafter
        \xint@cmp
    \expandafter\expandafter\expandafter
        {#1}%
}%
\def\xint@cmp #1#2%
{%
    \expandafter\expandafter\expandafter
    \XINT@cmp@fork #2\Z #1\Z
}%
\def\XINT@Cmp #1#2{\romannumeral0\XINT@cmp@fork #2\Z #1\Z }%
    COMPARAISON
  1 si #3#4>#1#2, 0 si #3#4=#1#2, -1 si #3#4<#1#2
  #3#4 vient du *premier*
  #1#2 vient du *second*

\def\XINT@cmp@fork #1#2\Z #3#4\Z
{%
```

```
    \xint@UDsignsfork
          #1#3\dummy \XINT@cmp@minusminus
           #1-\dummy \XINT@cmp@minusplus
           #3-\dummy \XINT@cmp@plusminus
            --\dummy {\xint@UDzerosfork
                      #1#3\dummy \XINT@cmp@zerozero
                       #10\dummy \XINT@cmp@zeroplus
                       #30\dummy \XINT@cmp@pluszero
                        00\dummy \XINT@cmp@plusplus
                      \xint@UDkrof }%
    \xint@UDkrof
    {#2}{#4}#1#3%
}%
\def\XINT@cmp@minusplus #1#2#3#4{ 1}%
\def\XINT@cmp@plusminus #1#2#3#4{ -1}%
\def\XINT@cmp@zerozero  #1#2#3#4{ 0}%
\def\XINT@cmp@zeroplus  #1#2#3#4{ 1}%
\def\XINT@cmp@pluszero  #1#2#3#4{ -1}%
\def\XINT@cmp@plusplus #1#2#3#4%
{%
    \XINT@cmp@pre {#4#2}{#3#1}%
}%
\def\XINT@cmp@minusminus #1#2#3#4%
{%
    \XINT@cmp@pre {#1}{#2}%
}%
\def\XINT@cmp@pre #1%
{%
  \expandafter\XINT@cmp@@pre\expandafter{%
  \romannumeral0\XINT@RQ {}#1\R\R\R\R\R\R\R\R\Z
  }%
}%
\def\XINT@cmp@@pre #1#2%
{%
    \expandafter\XINT@cmp@A
    \expandafter1\expandafter{\expandafter}%
    \romannumeral0\XINT@RQ {}#2\R\R\R\R\R\R\R\R\Z
        \W\X\Y\Z #1\W\X\Y\Z
}%

  COMPARAISON
  N1 et N2 sont présentés à l'envers ET ON A RAJOUTÉ DES ZÉROS
  POUR QUE LEURS LONGUEURS À CHACUN SOIENT MULTIPLES DE 4, MAIS
  AUCUN NE SE TERMINE EN 0000
  routine appelée via \XINT@cmp@A 1{}<N1>\W\X\Y\Z<N2>\W\X\Y\Z
  ATTENTION RENVOIE 1 SI N1 < N2, 0 si N1 = N2, -1 si N1 > N2

\def\XINT@cmp@A #1#2#3\W\X\Y\Z #4#5#6#7%
{%
    \xint@w
    #4\xint@cmp@az
    \W\XINT@cmp@B #1{#4#5#6#7}{#2}#3\W\X\Y\Z
}%
```

```
\def\XINT@cmp@B #1#2#3#4#5#6#7%
{%
    \xint@w
    #4\xint@cmp@bz
    \W\XINT@cmp@onestep #1#2{#7#6#5#4}{#3}%
}%
\def\XINT@cmp@onestep #1#2#3#4#5#6%
{\expandafter
    \XINT@cmp@backtoA\the\numexpr 11#5#4#3#2-#6+#1-1\relax.%
}%
\def\XINT@cmp@backtoA #1#2#3.#4%
{%
    \XINT@cmp@A #2{#3#4}%
}%
\def\xint@cmp@bz
    \W\XINT@cmp@onestep #1\Z { 1}%
\def\xint@cmp@az\W\XINT@cmp@B #1#2#3#4#5#6#7%
{%
    \xint@w
    #4\xint@cmp@ez
    \W\XINT@cmp@Eenter #1{#3}#4#5#6#7%
}%
\def\XINT@cmp@Eenter #1\Z { -1}%
\def\xint@cmp@ez\W\XINT@cmp@Eenter #1%
{%
    \xint@UDzerofork
      #1\dummy \XINT@cmp@K             %     il y a une retenue
       0\dummy \XINT@cmp@L             %     pas de retenue
    \xint@UDkrof
}%
\def\XINT@cmp@K #1\Z { -1}%
\def\XINT@cmp@L #1{\XINT@OneIfPositive@main #1}%
\def\XINT@OneIfPositive #1%
{%
    \XINT@OneIfPositive@main #1\W\X\Y\Z%
}%
\def\XINT@OneIfPositive@main #1#2#3#4%
{%
    \xint@z #4\xint@OneIfPositive@terminated\Z\XINT@OneIfPositive@onestep
    #1#2#3#4%
}%
\def\xint@OneIfPositive@terminated\Z\XINT@OneIfPositive@onestep\W\X\Y\Z { 0}%
\def\XINT@OneIfPositive@onestep #1#2#3#4%
{%
    \expandafter
    \XINT@OneIfPositive@check
    \the\numexpr #1#2#3#4\relax
}%
\def\XINT@OneIfPositive@check #1%
{%
    \xint@zero
    #1\xint@OneIfPositive@backtomain 0\XINT@OneIfPositive@finish #1%
```

```
}%
\def\XINT@OneIfPositive@finish #1\W\X\Y\Z{ 1}%
\def\xint@OneIfPositive@backtomain 0\XINT@OneIfPositive@finish 0%
                    {\XINT@OneIfPositive@main }%
```

## 12.16 \xintGeq

```
    PLUS GRAND OU ÉGAL
  attention compare les **valeurs absolues**
```

```
\def\xintGeq {\romannumeral0\xintgeq }%
\def\xintgeq #1%
{%
    \expandafter\expandafter\expandafter
      \xint@geq
    \expandafter\expandafter\expandafter
      {#1}%
}%
\def\xint@geq #1#2%
{\expandafter\expandafter\expandafter
       \XINT@geq@fork #2\Z #1\Z
}%
\def\XINT@Geq #1#2{\romannumeral0\XINT@geq@fork #2\Z #1\Z }%
```

```
    PLUS GRAND OU ÉGAL
  ATTENTION, TESTE les VALEURS ABSOLUES
```

```
\def\XINT@geq@fork #1#2\Z #3#4\Z
{%
    \xint@UDzerofork
      #1\dummy \XINT@geq@secondiszero % |#1#2|=0
      #3\dummy \XINT@geq@firstiszero % |#1#2|>0
       0\dummy {\xint@UDsignsfork
                  #1#3\dummy \XINT@geq@minusminus
                   #1-\dummy \XINT@geq@minusplus
                   #3-\dummy \XINT@geq@plusminus
                    --\dummy \XINT@geq@plusplus
                \xint@UDkrof }%
    \xint@UDkrof
    {#2}{#4}#1#3%
}%
\def\XINT@geq@secondiszero     #1#2#3#4{ 1}%
\def\XINT@geq@firstiszero      #1#2#3#4{ 0}%
\def\XINT@geq@plusplus  #1#2#3#4%
         {\XINT@geq@pre {#4#2}{#3#1}}%
\def\XINT@geq@minusminus  #1#2#3#4%
         {\XINT@geq@pre {#2}{#1}}%
\def\XINT@geq@minusplus #1#2#3#4%
         {\XINT@geq@pre {#4#2}{#1}}%
\def\XINT@geq@plusminus #1#2#3#4%
         {\XINT@geq@pre {#2}{#3#1}}%
\def\XINT@geq@pre #1%
{%
```

```
  \expandafter\XINT@geq@@pre\expandafter{%
  \romannumeral0\XINT@RQ {}#1\R\R\R\R\R\R\R\Z
  }%
}%
\def\XINT@geq@@pre #1#2%
{%
    \expandafter\XINT@geq@A
    \expandafter1\expandafter{\expandafter}%
    \romannumeral0\XINT@RQ {}#2\R\R\R\R\R\R\R\Z
        \W\X\Y\Z #1 \W\X\Y\Z
}%

  PLUS GRAND OU ÉGAL
  N1 et N2 sont présentés à l'envers ET ON A RAJOUTÉ DES ZÉROS
  POUR QUE LEURS LONGUEURS À CHACUN SOIENT MULTIPLES DE 4, MAIS
  AUCUN NE SE TERMINE EN 0000
  routine appelée via
  \romannumeral0\XINT@geq@A 1{}<N1>\W\X\Y\Z<N2>\W\X\Y\Z
  ATTENTION RENVOIE 1 SI N1 < N2 ou N1 = N2 et 0 si N1 > N2

\def\XINT@geq@A #1#2#3\W\X\Y\Z #4#5#6#7%
{%
    \xint@w
    #4\xint@geq@az
    \W\XINT@geq@B #1{#4#5#6#7}{#2}#3\W\X\Y\Z
}%
\def\XINT@geq@B #1#2#3#4#5#6#7%
{%
    \xint@w
    #4\xint@geq@bz
    \W\XINT@geq@onestep #1#2{#7#6#5#4}{#3}%
}%
\def\XINT@geq@onestep #1#2#3#4#5#6%
{\expandafter
    \XINT@geq@backtoA\the\numexpr 11#5#4#3#2-#6+#1-1\relax.%
}%
\def\XINT@geq@backtoA #1#2#3.#4%
{%
    \XINT@geq@A #2{#3#4}%
}%
\def\xint@geq@bz\W\XINT@geq@onestep #1\W\X\Y\Z { 1}%
\def\xint@geq@az\W\XINT@geq@B #1#2#3#4#5#6#7%
{%
    \xint@w
    #4\xint@geq@ez
    \W\XINT@geq@Eenter #1%
}%
\def\XINT@geq@Eenter #1\W\X\Y\Z { 0}%
\def\xint@geq@ez\W\XINT@geq@Eenter #1%
{%
    \xint@UDzerofork
      #1\dummy { 0}             %     il y a une retenue
       0\dummy { 1}             %     pas de retenue
```

```
    \xint@UDkrof
}%
```

## 12.17 \xintMax

```
\def\xintMax {\romannumeral0\xintmax }%
\def\xintmax #1%
{%
    \expandafter\expandafter\expandafter
        \xint@max
    \expandafter\expandafter\expandafter
        {#1}%
}%
\def\xint@max #1#2%
{%
    \expandafter\expandafter\expandafter
      \XINT@max@fork #2\Z #1\Z
}%
\def\XINT@Max #1#2{\romannumeral0\XINT@max@fork #2\Z #1\Z }%
  #3#4 vient du *premier*
  #1#2 vient du *second*

\def\XINT@max@fork #1#2\Z #3#4\Z
{%
    \xint@UDsignsfork
          #1#3\dummy \XINT@max@minusminus  % A < 0, B < 0
           #1-\dummy \XINT@max@minusplus   % B < 0, A >= 0
           #3-\dummy \XINT@max@plusminus   % A < 0, B >= 0
            --\dummy {\xint@UDzerosfork
                      #1#3\dummy \XINT@max@zerozero % A = B = 0
                       #10\dummy \XINT@max@zeroplus % B = 0, A > 0
                       #30\dummy \XINT@max@pluszero % A = 0, B > 0
                        00\dummy \XINT@max@plusplus % A, B > 0
                      \xint@UDkrof }%
    \xint@UDkrof
    {#2}{#4}#1#3%
}%

  A = #4#2, B = #3#1

\def\XINT@max@zerozero #1#2#3#4{ 0}%
\def\XINT@max@zeroplus #1#2#3#4{ #4#2}%
\def\XINT@max@pluszero #1#2#3#4{ #3#1}%
\def\XINT@max@minusplus #1#2#3#4{ #4#2}%
\def\XINT@max@plusminus #1#2#3#4{ #3#1}%
\def\XINT@max@plusplus #1#2#3#4%
{%
    \ifodd\XINT@Geq {#4#2}{#3#1}
      \xint@afterfi { #4#2}%
    \else
      \xint@afterfi { #3#1}%
    \fi
}%
```

```
  #3=-, #4=-, #1 = |B| = -B, #2 = |A| = -A
1141 \def\XINT@max@minusminus #1#2#3#4%
1142 {%
1143     \ifodd\XINT@Geq {#1}{#2}
1144       \xint@afterfi { -#2}%
1145     \else
1146       \xint@afterfi { -#1}%
1147     \fi
1148 }%
```

## 12.18 \xintMin

```
1149 \def\xintMin {\romannumeral0\xintmin }%
1150 \def\xintmin #1%
1151 {%
1152     \expandafter\expandafter\expandafter
1153         \xint@min
1154     \expandafter\expandafter\expandafter
1155         {#1}%
1156 }%
1157 \def\xint@min #1#2%
1158 {%
1159     \expandafter\expandafter\expandafter
1160       \XINT@min@fork #2\Z #1\Z
1161 }%
1162 \def\XINT@Min #1#2{\romannumeral0\XINT@min@fork #2\Z #1\Z }%
  #3#4 vient du *premier*
  #1#2 vient du *second*
1163 \def\XINT@min@fork #1#2\Z #3#4\Z
1164 {%
1165     \xint@UDsignsfork
1166           #1#3\dummy \XINT@min@minusminus  % A < 0, B < 0
1167            #1-\dummy \XINT@min@minusplus   % B < 0, A >= 0
1168            #3-\dummy \XINT@min@plusminus   % A < 0, B >= 0
1169             --\dummy {\xint@UDzerosfork
1170                       #1#3\dummy \XINT@min@zerozero % A = B = 0
1171                        #10\dummy \XINT@min@zeroplus % B = 0, A > 0
1172                        #30\dummy \XINT@min@pluszero % A = 0, B > 0
1173                         00\dummy \XINT@min@plusplus % A, B > 0
1174                       \xint@UDkrof }%
1175     \xint@UDkrof
1176     {#2}{#4}#1#3%
1177 }%
  A = #4#2, B = #3#1
1178 \def\XINT@min@zerozero #1#2#3#4{ 0}%
1179 \def\XINT@min@zeroplus #1#2#3#4{ 0}%
1180 \def\XINT@min@pluszero #1#2#3#4{ 0}%
1181 \def\XINT@min@minusplus #1#2#3#4{ #3#1}%
```

```
\def\XINT@min@plusminus #1#2#3#4{ #4#2}%
\def\XINT@min@plusplus #1#2#3#4%
{%
    \ifodd\XINT@Geq {#4#2}{#3#1}
      \xint@afterfi { #3#1}%
    \else
      \xint@afterfi { #4#2}%
    \fi
}%
```

```
  #3=-, #4=-, #1 = |B| = -B, #2 = |A| = -A
```

```
\def\XINT@min@minusminus #1#2#3#4%
{%
    \ifodd\XINT@Geq {#1}{#2}
      \xint@afterfi { -#1}%
    \else
      \xint@afterfi { -#2}%
    \fi
}%
```

## 12.19 \xintSum, \xintSumExpr

```
  \xintSum {{a}{b}...{z}}
  \xintSumExpr {a}{b}...{z}\relax
```

```
\def\XINT@psum #1%
{%
    \romannumeral0\XINT@psum@checkifemptysum #1\Z
}%
\def\XINT@psum@checkifemptysum #1%
{%
    \xint@relax #1\XINT@psum@returnzero\relax \XINT@psum@RQfirst #1%
}%
\def\XINT@psum@returnzero #1\Z { 0}%
\def\XINT@psum@RQfirst #1\Z
{%
    \expandafter\XINT@psum@loop\expandafter
    {\romannumeral0\XINT@RQ {}#1\R\R\R\R\R\R\R\R\Z}%   avant: #1\Z
}%
\def\XINT@psum@loop #1#2%
{%
    \xint@relax #2\XINT@psum@end\relax
    \expandafter
    \XINT@psum@loop\expandafter
    {\romannumeral0\expandafter\XINT@sum@A
        \expandafter0\expandafter{\expandafter}%
    \romannumeral0\XINT@RQ {}#2\R\R\R\R\R\R\R\R\Z
    \W\X\Y\Z #1\W\X\Y\Z }%
}%
\def\XINT@psum@end\relax\expandafter
                   \XINT@psum@loop\expandafter #1%
         {\XINT@psum@end@ #1}%
```

```
\def\XINT@psum@end@ #1\W\X\Y\Z #2\W\X\Y\Z
{%
    \expandafter
    \xint@cleanupzeros@andstop\romannumeral0\XINT@rev {#2}%
}%
\def\xintSumExpr {\romannumeral0\xintsumexpr }%
\def\xintSum {\romannumeral0\xintsum }%
\def\xintsum #1%
{%
    \expandafter\expandafter\expandafter
    \xintsumexpr #1\relax
}%
\def\xintsumexpr #1%
{%
    \expandafter\expandafter\expandafter
    \XINT@sum@checkifempty #1\Z {\XINT@psum }{\XINT@psum }%
}%
\def\XINT@sum@checkifempty #1%
{%
    \xint@relax #1\XINT@sum@returnzero\relax
    \XINT@sum@checksign #1%
}%
\def\XINT@sum@returnzero #1\Z #2#3{ 0}%
\def\XINT@sum@checksign #1%
{%
    \xint@zero #1\XINT@sum@skipzeroinput0%
    \xint@UDsignfork
      #1\dummy \XINT@sum@pushneg
       -\dummy \XINT@sum@pushpos
    \xint@UDkrof
    #1%
}%
\def\XINT@sum@skipzeroinput #1\xint@UDkrof #2\Z #3#4%
{%
    \XINT@sum@xpxpnext {#3}{#4}%
}%
\def\XINT@sum@pushpos #1#2\Z #3#4%
{%
    \XINT@sum@xpxpnext {#3{#1#2}}{#4}%
}%
\def\XINT@sum@pushneg #1#2\Z #3#4%
{%
    \XINT@sum@xpxpnext  {#3}{#4{#2}}%
}%
\def\XINT@sum@xpxpnext #1#2#3%
{%
    \expandafter\expandafter\expandafter
    \XINT@sum@checkiffinished #3\Z {#1}{#2}%
}%
\def\XINT@sum@checkiffinished #1%
{%
    \xint@relax #1\XINT@sum@end\relax
```

```
    \XINT@sum@checksign #1%
}%
\def\XINT@sum@end\relax\XINT@sum@checksign\relax #1\Z #2#3%
      {\xintsub{#2\relax}{#3\relax}}%
\def\XINT@sum@A #1#2#3#4#5#6%
{%
    \xint@w
    #3\xint@sum@az
    \W\XINT@sum@B #1{#3#4#5#6}{#2}%
}%
\def\xint@sum@az\W\XINT@sum@B #1#2%
{%
    \XINT@sum@AC@checkcarry #1%
}%
\def\XINT@sum@B #1#2#3#4\W\X\Y\Z #5#6#7#8%
{%
    \xint@w
    #5\xint@sum@bz
    \W\XINT@sum@E #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
}%
\def\XINT@sum@E #1#2#3#4#5#6%
{\expandafter
    \XINT@sum@ABEA\the\numexpr #1+10#5#4#3#2+#6\relax
}%
\def\XINT@sum@ABEA #1#2#3#4#5#6#7%
{%
    \XINT@sum@A  #2{#7#6#5#4#3}%
}%
\def\xint@sum@bz\W\XINT@sum@E #1#2#3#4#5#6%
{\expandafter
    \XINT@sum@CC\the\numexpr #1+10#5#4#3#2\relax
}%
\def\XINT@sum@CC #1#2#3#4#5#6#7%
{%
    \XINT@sum@AC@checkcarry  #2{#7#6#5#4#3}%
}%
\def\XINT@sum@AC@checkcarry #1%
{%
    \xint@zero #1\xint@sum@AC@nocarry 0\XINT@sum@C
}%
\def\xint@sum@AC@nocarry 0\XINT@sum@C #1#2\W\X\Y\Z { #1#2}%
\def\XINT@sum@C #1#2#3#4#5%
{%
    \xint@w
    #2\xint@sum@cz
    \W\XINT@sum@D {#5#4#3#2}{#1}%
}%
\def\XINT@sum@D #1%
{\expandafter
    \XINT@sum@CC\the\numexpr 1+10#1\relax
}%
\def\xint@sum@cz\W\XINT@sum@D #1#2{ #21000}%
```

## 12.20 \xintMul

```
\def\xintMul {\romannumeral0\xintmul }%
\def\xintmul #1%
{%
    \expandafter\expandafter\expandafter
        \xint@mul
    \expandafter\expandafter\expandafter
        {#1}%
}%
\def\xint@mul #1#2%
{\expandafter\expandafter\expandafter
       \XINT@mul@fork #2\Z #1\Z
}%
\def\XINT@Mul #1#2{\romannumeral0\XINT@mul@fork #2\Z #1\Z }%
    MULTIPLICATION
  Ici #1#2 = 2e input et #3#4 = 1er input
  Quelques précisions apportées à l'occasion de 1.02 (qui modifie le
  fonctionnement de \xintPrd pour tenir compte des règles ci-dessous, et aussi
  améliore au passage très légèrement la vitesse de calcul de la factorielle):
  La multiplication est plus rapide sous les conditions suivantes:
    - si le nombre le plus court a au plus 4 chiffres, il DOIT être en 2e
    - si les deux ont au plus 50 chiffres, le plus court en *second*
    - si les deux ont au moins 50 chiffres, le plus court en *premier*
    - si le plus long a au moins 250 chiffres, mettre ce plus long
    toujours en *second* (sauf si l'autre a au plus 4 chiffres)
    - si le plus long a au moins 100 chiffres, en second si le premier a au
    moins 12 chiffres.
  La règle générale est donc à peu près: pour les 'gros calculs' mettre le
  plus court en premier (SAUF pour multiplication par un nombre < 10000), et
  pour les 'petits calculs' le plus long en premier.

\def\XINT@mul@fork #1#2\Z #3#4\Z
{%
    \xint@UDzerofork
      #1\dummy \XINT@mul@zero
      #3\dummy \XINT@mul@zero
       0\dummy
        {\xint@UDsignsfork
          #1#3\dummy \XINT@mul@minusminus          % #1 = #3 = -
           #1-\dummy \XINT@mul@minusplus           % #1 = -
           #3-\dummy \XINT@mul@plusminus           % #3 = -
            --\dummy \XINT@mul@plusplus
         \xint@UDkrof }%
    \xint@UDkrof
    {#2}{#4}#1#3%
}%
\def\XINT@mul@zero #1#2#3#4{ 0}%

  Dans ce qui suit #3#1 vient du #1#2 initial correspondant au
  ** 2e ** input.

\def\XINT@mul@minusminus #1#2#3#4%
{%
```

```
      \expandafter
          \XINT@mul@enter\romannumeral0%
          \XINT@RQ {}#2\R\R\R\R\R\R\R\Z
          \W\X\Y\Z #1\W\X\Y\Z
}%
\def\XINT@mul@minusplus #1#2#3#4%
{%
      \expandafter\space\expandafter-%
      \romannumeral0\expandafter
          \XINT@mul@enter\romannumeral0%
          \XINT@RQ {}#4#2\R\R\R\R\R\R\R\Z
          \W\X\Y\Z #1\W\X\Y\Z
}%
\def\XINT@mul@plusminus #1#2#3#4%
{%
      \expandafter\space\expandafter-%
      \romannumeral0\expandafter
          \XINT@mul@enter\romannumeral0%
          \XINT@RQ {}#2\R\R\R\R\R\R\R\Z
          \W\X\Y\Z #3#1\W\X\Y\Z
}%

  Ici #3#1 correspond au **2e input**

\def\XINT@mul@plusplus #1#2#3#4%
{%
      \expandafter
          \XINT@mul@enter\romannumeral0%
          \XINT@RQ {}#4#2\R\R\R\R\R\R\R\Z
          \W\X\Y\Z #3#1\W\X\Y\Z
}%
\def\XINT@mul@add@A #1#2#3#4#5#6%
{%
    \xint@w
    #3\xint@mul@add@az
    \W\XINT@mul@add@AB #1{#3#4#5#6}{#2}%
}%
\def\xint@mul@add@az\W\XINT@mul@add@AB #1#2%
{%
    \XINT@mul@add@AC@checkcarry #1%
}%
\def\XINT@mul@add@AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
{%
    \XINT@mul@add@ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
}%
\def\XINT@mul@add@ABE #1#2#3#4#5#6%
{\expandafter
    \XINT@mul@add@ABEA\the\numexpr #1+10#5#4#3#2+#6\relax.%
}%
\def\XINT@mul@add@ABEA #1#2#3.#4%
{%
    \XINT@mul@add@A  #2{#3#4}%
}%
```

```
\def\XINT@mul@add@AC@checkcarry #1%
{%
    \xint@zero #1\xint@mul@add@AC@nocarry 0\XINT@mul@add@C
}%
\def\xint@mul@add@AC@nocarry 0\XINT@mul@add@C #1#2\W\X\Y\Z
{%
    \expandafter
    \xint@cleanupzeros@andstop
    \romannumeral0%
    \XINT@rord@main {}#2%
      \xint@UNDEF
        \xint@undef\xint@undef\xint@undef\xint@undef
        \xint@undef\xint@undef\xint@undef\xint@undef
      \xint@UNDEF
    #1%
}%
\def\XINT@mul@add@C #1#2#3#4#5%
{%
    \xint@w
    #5\xint@mul@add@cw
    #4\xint@mul@add@cx
    #3\xint@mul@add@cy
    #2\xint@mul@add@cz
    \W\XINT@mul@add@CD {#5#4#3#2}{#1}%
}%
\def\XINT@mul@add@CD #1%
{\expandafter
    \XINT@mul@add@CC\the\numexpr 1+10#1\relax.%
}%
\def\XINT@mul@add@CC #1#2#3.#4%
{%
    \XINT@mul@add@AC@checkcarry  #2{#3#4}%
}%
\def\xint@mul@add@cw
    #1\xint@mul@add@cx
    #2\xint@mul@add@cy
    #3\xint@mul@add@cz
    \W\XINT@mul@add@CD
{\expandafter
    \XINT@mul@add@CDw\the\numexpr 1+#1#2#3\relax.%
}%
\def\XINT@mul@add@CDw #1.#2#3\X\Y\Z
{%
    \XINT@mul@add@end #1#3%
}%
\def\xint@mul@add@cx
    #1\xint@mul@add@cy
    #2\xint@mul@add@cz
    \W\XINT@mul@add@CD
{\expandafter
    \XINT@mul@add@CDx\the\numexpr 1+#1#2\relax.%
}%
```

```
\def\XINT@mul@add@CDx   #1.#2#3\Y\Z
{%
    \XINT@mul@add@end #1#3%
}%
\def\xint@mul@add@cy
    #1\xint@mul@add@cz
    \W\XINT@mul@add@CD
{\expandafter
    \XINT@mul@add@CDy\the\numexpr 1+#1\relax.%
}%
\def\XINT@mul@add@CDy  #1.#2#3\Z
{%
    \XINT@mul@add@end #1#3%
}%
\def\xint@mul@add@cz\W\XINT@mul@add@CD #1#2#3{\XINT@mul@add@end #1#3}%
\def\XINT@mul@add@end #1#2#3#4#5%
{\expandafter\space
  \the\numexpr #1#2#3#4#5\relax
}%
\def\XINT@mul@Ar #1#2#3#4#5#6%
{%
    \xint@z #6\xint@mul@br\Z\XINT@mul@Br #1{#6#5#4#3}{#2}%
}%
\def\xint@mul@br\Z\XINT@mul@Br #1#2%
{%
     \XINT@sum@AC@checkcarry #1%
}%
\def\XINT@mul@Br #1#2#3#4\W\X\Y\Z #5#6#7#8%
{\expandafter
    \XINT@mul@ABEAr\the\numexpr #1+10#2+#8#7#6#5\relax.{#3}#4\W\X\Y\Z
}%
\def\XINT@mul@ABEAr #1#2#3#4#5#6.#7%
{%
    \XINT@mul@Ar #2{#7#6#5#4#3}%
}%
```

```
  Mr renvoie le résultat ***à l'envers***, sur ***4n chiffres***
```

```
\def\XINT@mul@Mr #1%
{%
    \expandafter
    \XINT@mul@Mr@checkifzeroorone
    \expandafter{\the\numexpr #1\relax}%
}%
\def\XINT@mul@Mr@checkifzeroorone #1%
{%
    \ifcase #1
      \expandafter\XINT@mul@Mr@zero
    \or
      \expandafter\XINT@mul@Mr@one
    \else
      \expandafter\XINT@mul@Nr
    \fi
```

```
    {0000}{}{#1}%
}%
\def\XINT@mul@Mr@zero #1\Z\Z\Z\Z { 0000}%
\def\XINT@mul@Mr@one #1#2#3#4\Z\Z\Z\Z { #4}%
\def\XINT@mul@Nr #1#2#3#4#5#6#7%
{%
    \xint@z #4\xint@mul@pr\Z\XINT@mul@Pr {#1}{#3}{#7#6#5#4}{#2}{#3}%
}%
\def\XINT@mul@Pr #1#2#3%
{\expandafter
    \XINT@mul@Lr\the\numexpr 10000#1+#2*#3\relax
}%
\def\XINT@mul@Lr 1#1#2#3#4#5#6#7#8#9%
{%
    \XINT@mul@Nr {#1#2#3#4}{#9#8#7#6#5}%
}%
\def\xint@mul@pr\Z\XINT@mul@Pr #1#2#3#4#5%
{%
    \xint@quatrezeros #1\XINT@mul@Mr@end@nocarry 0000\XINT@mul@Mr@end@carry
    #1{#4}%
}%
\def\XINT@mul@Mr@end@nocarry 0000\XINT@mul@Mr@end@carry 0000#1{ #1}%
\def\XINT@mul@Mr@end@carry #1#2#3#4#5{ #5#4#3#2#1}%
\def\XINT@mul@M #1%
{\expandafter
    \XINT@mul@M@checkifzeroorone
    \expandafter{\the\numexpr #1\relax}%
}%
\def\XINT@mul@M@checkifzeroorone #1%
{%
    \ifcase #1
      \expandafter\XINT@mul@M@zero
    \or
      \expandafter\XINT@mul@M@one
    \else
      \expandafter\XINT@mul@N
    \fi
    {0000}{}{#1}%
}%
\def\XINT@mul@M@zero #1\Z\Z\Z\Z { 0}%
\def\XINT@mul@M@one #1#2#3#4\Z\Z\Z\Z {%
    \expandafter
       \xint@cleanupzeros@andstop
    \romannumeral0\XINT@rev{#4}%
}%
\def\XINT@mul@N #1#2#3#4#5#6#7%
{%
    \xint@z #4\xint@mul@p\Z\XINT@mul@P {#1}{#3}{#7#6#5#4}{#2}{#3}%
}%
\def\XINT@mul@P #1#2#3%
{\expandafter
    \XINT@mul@L\the\numexpr 10000#1+#2*#3\relax
```

```
}%
\def\XINT@mul@L 1#1#2#3#4#5#6#7#8#9%
{%
    \XINT@mul@N {#1#2#3#4}{#5#6#7#8#9}%
}%
\def\xint@mul@p\Z\XINT@mul@P #1#2#3#4#5%
{%
    \XINT@mul@M@end #1#4%
}%
\def\XINT@mul@M@end #1#2#3#4#5#6#7#8%
{\expandafter\space
   \the\numexpr #1#2#3#4#5#6#7#8\relax
}%

  Routine de multiplication principale
  délimiteur \W\X\Y\Z
  Le résultat partiel est toujours maintenu avec significatif à
  droite et il a un nombre multiple de 4 de chiffres
  \romannumeral0\XINT@mul@enter <N1>\W\X\Y\Z <N2>\W\X\Y\Z
  avec N1: *renversé*, *longueur 4n* (zéros éventuellement ajoutés
  au-delà du chiffre le plus significatif)
  et N2 = dans l'ordre *normal*, et pas forcément longueur 4n,
  et N2 est *non nul*.
  pas de signes

\def\XINT@mul@enter #1\W\X\Y\Z #2#3#4#5%
{%
    \xint@w
    #5\xint@mul@enterw
    #4\xint@mul@enterx
    #3\xint@mul@entery
    #2\xint@mul@enterz
    \W\XINT@mul@start {#2#3#4#5}#1\W\X\Y\Z
}%
\def\xint@mul@enterw
    #1\xint@mul@enterx
    #2\xint@mul@entery
    #3\xint@mul@enterz
    \W\XINT@mul@start  #4#5\W\X\Y\Z \X\Y\Z
{%
    \XINT@mul@M {#3#2#1}#5\Z\Z\Z\Z
}%
\def\xint@mul@enterx
    #1\xint@mul@entery
    #2\xint@mul@enterz
    \W\XINT@mul@start  #3#4\W\X\Y\Z \Y\Z
{%
    \XINT@mul@M {#2#1}#4\Z\Z\Z\Z
}%
\def\xint@mul@entery
    #1\xint@mul@enterz
    \W\XINT@mul@start  #2#3\W\X\Y\Z \Z
{%
```

```
    \XINT@mul@M {#1}#3\Z\Z\Z\Z
}%
\def\XINT@mul@start #1#2\W\X\Y\Z
{\expandafter
    \XINT@mul@main \expandafter
    {\romannumeral0\XINT@mul@Mr {#1}#2\Z\Z\Z\Z}#2\W\X\Y\Z
}%
\def\XINT@mul@main #1#2\W\X\Y\Z #3#4#5#6%
{%
    \xint@w
    #6\xint@mul@mainw
    #5\xint@mul@mainx
    #4\xint@mul@mainy
    #3\xint@mul@mainz
    \W\XINT@mul@compute {#1}{#3#4#5#6}#2\W\X\Y\Z
}%
\def\XINT@mul@compute #1#2#3\W\X\Y\Z
{\expandafter
    \XINT@mul@main \expandafter
           {\romannumeral0\expandafter
    \XINT@mul@Ar \expandafter0\expandafter{\expandafter}%
    \romannumeral0\XINT@mul@Mr {#2}#3\Z\Z\Z\Z \W\X\Y\Z 0000#1\W\X\Y\Z
            }#3\W\X\Y\Z
}%
\def\xint@mul@mainw
    #1\xint@mul@mainx
    #2\xint@mul@mainy
    #3\xint@mul@mainz
    \W\XINT@mul@compute  #4#5#6\W\X\Y\Z \X\Y\Z
{%
    \expandafter
    \XINT@mul@add@A \expandafter0\expandafter{\expandafter}%
          \romannumeral0%
    \XINT@mul@Mr {#3#2#1}#6\Z\Z\Z\Z
                 \W\X\Y\Z 000#4\W\X\Y\Z
}%
\def\xint@mul@mainx
    #1\xint@mul@mainy
    #2\xint@mul@mainz
    \W\XINT@mul@compute  #3#4#5\W\X\Y\Z \Y\Z
{%
    \expandafter
    \XINT@mul@add@A \expandafter0\expandafter{\expandafter}%
          \romannumeral0%
    \XINT@mul@Mr {#2#1}#5\Z\Z\Z\Z
               \W\X\Y\Z 00#3\W\X\Y\Z
}%
\def\xint@mul@mainy
    #1\xint@mul@mainz
    \W\XINT@mul@compute  #2#3#4\W\X\Y\Z \Z
{%
    \expandafter
```

```
    \XINT@mul@add@A \expandafter0\expandafter{\expandafter}%
          \romannumeral0%
    \XINT@mul@Mr {#1}#4\Z\Z\Z\Z
             \W\X\Y\Z 0#2\W\X\Y\Z
}%
\def\xint@mul@mainz\W\XINT@mul@compute #1#2#3\W\X\Y\Z
{%
    \expandafter
    \xint@cleanupzeros@andstop\romannumeral0\XINT@rev{#1}%
}%
```

## 12.21 \xintSqr

```
\def\xintSqr {\romannumeral0\xintsqr }%
\def\xintsqr #1%
{%
    \expandafter\expandafter\expandafter
    \XINT@sqr
    \expandafter\expandafter\expandafter
    {\xintAbs{#1}}% fait l'expansion de #1 et se d\'ebarrasse du signe
}%
\def\XINT@sqr #1%
{\expandafter
          \XINT@mul@enter
          \romannumeral0%
          \XINT@RQ {}#1\R\R\R\R\R\R\R\R\Z
          \W\X\Y\Z #1\W\X\Y\Z
}%
```

## 12.22 \xintPrd, \xintProductExpr

```
\xintPrd {{a}...{z}}
\xintProductExpr {a}...{z}\relax
Release 1.02 modifies \XINT@posprod. The new version takes into account that
multiplication on long numbers is more efficient with the shorter one first;
the earlier version was on the premise that it was more efficient to give
the longer number first and used a special version of multiplication to
produce its output in reversed order to serve as next first number. This was
optimal for numbers of at most 50 digits, but the bad choice for long
numbers. As \xintPrd should be fast when it is used to produce long numbers
(producing short numbers means few factors means computation does not take
much time anyhow), I revert this choice here. On the other hand this was the
correct choice (and tested so) for use in the \xintPow recursion which each
times multiplies by something even bigger than what has been obtained so
far. So the original version is renamed and moved to serve for \xintPow.
The factorial receives separate special revision.
```

```
\def\XINT@posprod #1%
{%
    \XINT@pprod@checkifempty #1\Z
}%
\def\XINT@pprod@checkifempty #1%
{%
    \xint@relax #1\XINT@pprod@emptyproduct\relax
```

```
    \XINT@pprod@first #1%
}%
\def\XINT@pprod@emptyproduct #1\Z { 1}%
\def\XINT@pprod@first #1\Z
{%
    \XINT@pprod@getnext {#1}%
}%
\def\XINT@pprod@getnext #1#2%
{%
    \XINT@pprod@checkiffinished #2\Z {#1}%
}%
\def\XINT@pprod@checkiffinished #1%
{%
    \xint@relax #1\XINT@pprod@end\relax
    \XINT@pprod@RQnew #1%
}%
\def\XINT@pprod@RQnew #1\Z
{%
    \expandafter\XINT@pprod@compute
    \expandafter
    {\romannumeral0\XINT@RQ {}#1\R\R\R\R\R\R\R\Z }%
}%
\def\XINT@pprod@compute #1#2%
{%
    \expandafter
        \XINT@pprod@getnext
    \expandafter
    {\romannumeral0\XINT@mul@enter #1\W\X\Y\Z #2\W\X\Y\Z }%
}%
\def\XINT@pprod@end\relax\XINT@pprod@RQnew #1\Z #2{ #2}%
\def\xintProductExpr {\romannumeral0\xintproductexpr }%
\def\xintPrd {\romannumeral0\xintprd }%
\def\xintprd #1%
{%
    \expandafter\expandafter\expandafter
    \xintproductexpr #1\relax
}%
\def\xintproductexpr #1%
{%
    \expandafter\expandafter\expandafter
    \XINT@prod@checkifempty #1\Z
}%
\def\XINT@prod@checkifempty #1%
{%
    \xint@relax #1\XINT@prod@emptyproduct\relax
    \XINT@prod@checkfirstsign #1%
}%
\def\XINT@prod@emptyproduct #1\Z { 1}%
\def\XINT@prod@checkfirstsign #1%
{%
    \xint@zero #1\XINT@prod@returnzero0%
    \xint@UDsignfork
```

```
      #1\dummy \XINT@prod@firstisneg
       -\dummy \XINT@prod@firstispos
    \xint@UDkrof
    #1%
}%
\def\XINT@prod@returnzero #1\relax { 0}%
\def\XINT@prod@firstisneg #1#2\Z
{%
    \XINT@prod@xpxpnext 0{#2}%
}%
\def\XINT@prod@firstispos #1\Z
{%
    \XINT@prod@xpxpnext 1{#1}%
}%
\def\XINT@prod@xpxpnext #1#2#3%
{%
    \expandafter\expandafter\expandafter
    \XINT@prod@checkiffinished #3\Z {#2}#1%
}%
\def\XINT@prod@checkiffinished #1%
{%
    \xint@relax #1\XINT@prod@end\relax
    \XINT@prod@checksign #1%
}%
\def\XINT@prod@checksign #1%
{%
    \xint@zero #1\XINT@prod@returnzero0%
    \xint@UDsignfork
      #1\dummy \XINT@prod@neg@RQnew
       -\dummy \XINT@prod@pos@RQnew
    \xint@UDkrof
    #1%
}%
\def\XINT@prod@pos@RQnew #1\Z
{%
    \expandafter
      \XINT@prod@pos
    \expandafter
    {\romannumeral0\XINT@RQ {}#1\R\R\R\R\R\R\R\R\Z }%
}%
\def\XINT@prod@pos #1#2#3%
{%
    \expandafter
       \XINT@prod@xpxpnext
    \expandafter
       #3%
    \expandafter
       {\romannumeral0\XINT@mul@enter #1\W\X\Y\Z #2\W\X\Y\Z }%
}%
\def\XINT@prod@neg@RQnew #1#2\Z
{%
    \expandafter
```

```
      \XINT@prod@neg
    \expandafter
    {\romannumeral0\XINT@RQ {}#2\R\R\R\R\R\R\R\Z }%
}%
\def\XINT@prod@neg #1#2#3%
{%
    \expandafter
      \XINT@prod@xpxpnext
    \expandafter
      {\the\numexpr 1-#3\expandafter}%
    \expandafter
    {\romannumeral0\XINT@mul@enter #1\W\X\Y\Z #2\W\X\Y\Z }%
}%
\def\XINT@prod@end\relax\XINT@prod@checksign #1\Z #2#3%
{%
    \xint@prod@cleanupzeros #3#2%
}%
\def\xint@prod@cleanupzeros #1#2#3#4#5%
{%
    \expandafter\space\the\numexpr (2*#1-1)*#2#3#4#5\relax
}%
```

## 12.23 \xintFac

```
Modified a bit with 1.02, following changes to \XINT@posprod
I am tempted, here and elsewhere, to use \ifcase\XINT@Geq {#1}{1000000000}
rather than \ifnum\XINT@Length {#1}>9 but for the time being I leave things
as they stand.
```

```
\def\xintFac {\romannumeral0\xintfac }%
\def\xintfac #1%
{%
    \expandafter\expandafter\expandafter
        \XINT@fac@fork
    \expandafter\expandafter\expandafter
        {#1}%
}%
\def\XINT@Fac {\romannumeral0\XINT@fac@fork }%
\def\XINT@fac@fork #1%
{%
    \ifcase\xintSgn {#1}
       \xint@afterfi{\expandafter\space\expandafter 1\xint@gobble }%
    \or
       \expandafter\XINT@fac@checklength
    \else
       \xint@afterfi{\xintError:FactorialOfNegativeNumber
                \expandafter\space\expandafter 1\xint@gobble }%
    \fi
    {#1}%
}%
\def\XINT@fac@checklength #1%
{%
    \ifnum \XINT@Length {#1}> 9
```

```
          \xint@afterfi{\xintError:FactorialOfTooBigNumber
                        \expandafter\space\expandafter 1\xint@gobble }%
    \else
          \xint@afterfi{\ifnum #1>9999
                           \expandafter\XINT@fac@big@loop
                        \else
                           \expandafter\XINT@fac@loop
                        \fi }%
    \fi
    {#1}%
}%
\def\XINT@fac@big@loop #1{\XINT@fac@big@loop@main {10000}{#1}{}}%
\def\XINT@fac@big@loop@main #1#2#3%
{%
    \ifnum #1<#2
        \expandafter
            \XINT@fac@big@loop@main
        \expandafter
           {\the\numexpr #1+1\expandafter }%
    \else
        \expandafter\XINT@fac@big@docomputation
    \fi
    {#2}{#3{#1}}%
}%
\def\XINT@fac@big@docomputation #1#2%
{%
    \expandafter
       \XINT@pprod@getnext
    \expandafter
    {\romannumeral0\XINT@fac@loop {9999}}#2\relax
}%
\def\XINT@fac@loop #1{\XINT@fac@loop@main 1{#1}{}}%
\def\XINT@fac@loop@main #1#2#3%
{%
    \ifnum #1<#2
        \expandafter
            \XINT@fac@loop@main
        \expandafter
           {\the\numexpr #1+1\expandafter }%
    \else
        \expandafter\XINT@fac@docomputation
    \fi
    {#2}{#3{#1}}%
}%
\def\XINT@fac@docomputation #1#2%
{%
    \XINT@fprod@getnext {1000}#2\relax
}%
\def\XINT@fprod@getnext #1#2%
{%
    \XINT@fprod@checkiffinished #2\Z {#1}%
}%
```

```
\def\XINT@fprod@checkiffinished #1%
{%
    \xint@relax #1\XINT@fprod@end\relax
    \XINT@fprod@compute #1%
}%
\def\XINT@fprod@compute #1\Z #2%
{%
    \expandafter
        \XINT@fprod@getnext
    \expandafter
    {\romannumeral0\XINT@mul@Mr {#1}#2\Z\Z\Z\Z }%
}%
\def\XINT@fprod@end\relax\XINT@fprod@compute #1\Z #2%
{%
    \expandafter
    \xint@cleanupzeros@andstop
    \romannumeral0\XINT@rev {#2}%
}%
```

## 12.24 \xintPow

1.02 this is the same as in earlier versions, but I had to move here the special routine of Product as it was previously done, with its accompanying special multiplication and addition (to maintain the intermediate results in a special reversed form), now that I have modified the \XINT@posprod routine.

```
\def\xintPow {\romannumeral0\xintpow }%
\def\xintpow #1%
{%
    \expandafter\expandafter\expandafter
       \xint@pow
       #1\Z%
}%
```

#1#2 = A

```
\def\xint@pow #1#2\Z
{%
    \xint@UDsignfork
      #1\dummy \XINT@pow@Aneg
       -\dummy \XINT@pow@Anonneg
    \xint@UDkrof
       #1{#2}%
}%
\def\XINT@pow@Aneg #1#2#3%
{%
   \expandafter\expandafter\expandafter
      \XINT@pow@Aneg@
   \expandafter\expandafter\expandafter
      {#3}{#2}%
}%
```

B = #1, xpxp déjà fait

```
\def\XINT@pow@Aneg@ #1%
{%
   \ifcase\XINT@Odd{#1}
   \or \expandafter\XINT@pow@Aneg@Bodd
   \fi
   \XINT@pow@Anonneg@ {#1}%
}%
\def\XINT@pow@Aneg@Bodd #1%
{%
    \expandafter\XINT@opp\romannumeral0\XINT@pow@Anonneg@
}%
```

B = #3, faire le xpxp

```
\def\XINT@pow@Anonneg #1#2#3%
{%
   \expandafter\expandafter\expandafter
      \XINT@pow@Anonneg@
   \expandafter\expandafter\expandafter
   {#3}{#1#2}%
}%
```

#1 = B, #2 = |A|

```
\def\XINT@pow@Anonneg@ #1#2%
{%
    \ifcase\XINT@Cmp {#2}{1}
        \expandafter\XINT@pow@AisOne
    \or
        \expandafter\XINT@pow@AatleastTwo
    \else
        \expandafter\XINT@pow@AisZero
    \fi
    {#1}{#2}%
}%
\def\XINT@pow@AisOne #1#2{ 1}%
```

#1 = B

```
\def\XINT@pow@AisZero #1#2%
{%
     \ifcase\XINT@Sgn {#1}
         \xint@afterfi { 1}%
     \or
         \xint@afterfi { 0}%
     \else
         \xint@afterfi {\xintError:DivisionByZero\space 0}%
     \fi
}%
\def\XINT@pow@AatleastTwo #1%
{%
    \ifcase\XINT@Sgn {#1}
        \expandafter\XINT@pow@BisZero
    \or
```

```
        \expandafter\XINT@pow@checkBlength
    \else
        \expandafter\XINT@pow@BisNegative
    \fi
    {#1}%
}%
\def\XINT@pow@BisNegative #1#2{\xintError:FractionRoundedToZero\space 0}%
\def\XINT@pow@BisZero #1#2{ 1}%
```

B = #1 > 0, A = #2 > 1

```
\def\XINT@pow@checkBlength #1#2%
{%
    \ifnum\xintLen{#1} >9
        \expandafter\XINT@pow@BtooBig
    \else
        \expandafter\XINT@pow@loop
    \fi
    {#1}{#2}\XINT@pow@posprod
      \xint@UNDEF
        \xint@undef\xint@undef\xint@undef\xint@undef
        \xint@undef\xint@undef\xint@undef\xint@undef
      \xint@UNDEF
}%
\def\XINT@pow@BtooBig #1\xint@UNDEF #2\xint@UNDEF
                {\xintError:ExponentTooBig\space 0}%
\def\XINT@pow@loop #1#2%
{%
    \ifnum #1 = 1
        \expandafter\XINT@pow@loop@end
    \else
        \xint@afterfi{\expandafter\XINT@pow@loop@a
            \expandafter{\the\numexpr 2*(#1/2)-#1\expandafter }% b mod 2
            \expandafter{\the\numexpr #1-#1/2\expandafter }%     [b/2]
            \expandafter{\romannumeral0\xintsqr{#2}}}%
    \fi
    {{#2}}%
}%
\def\XINT@pow@loop@end {\romannumeral0\XINT@rord@main {}\relax }%
\def\XINT@pow@loop@a #1%
{%
    \ifnum #1 = 1
        \expandafter\XINT@pow@loop
    \else
        \expandafter\XINT@pow@loop@throwaway
    \fi
}%
\def\XINT@pow@loop@throwaway #1#2#3%
{%
   \XINT@pow@loop {#1}{#2}%
}%
```

Routine de produit servant pour le calcul des puissances.

```
\def\XINT@pow@posprod #1%
{%
    \XINT@pow@pprod@checkifempty #1\Z
}%
\def\XINT@pow@pprod@checkifempty #1%
{%
    \xint@relax #1\XINT@pow@pprod@emptyproduct\relax
    \XINT@pow@pprod@RQfirst #1%
}%
\def\XINT@pow@pprod@emptyproduct #1\Z { 1}%
\def\XINT@pow@pprod@RQfirst #1\Z
{%
    \expandafter\XINT@pow@pprod@getnext\expandafter
    {\romannumeral0\XINT@RQ {}#1\R\R\R\R\R\R\R\Z}%
}%
\def\XINT@pow@pprod@getnext #1#2%
{%
    \XINT@pow@pprod@checkiffinished #2\Z {#1}%
}%
\def\XINT@pow@pprod@checkiffinished #1%
{%
    \xint@relax #1\XINT@pow@pprod@end\relax
    \XINT@pow@pprod@compute #1%
}%
\def\XINT@pow@pprod@compute #1\Z #2%
{%
    \expandafter
        \XINT@pow@pprod@getnext
    \expandafter
    {\romannumeral0\XINT@pow@mul@enter #2\W\X\Y\Z #1\W\X\Y\Z}%
}%
\def\XINT@pow@pprod@end\relax\XINT@pow@pprod@compute #1\Z #2%
{%
    \expandafter
    \xint@cleanupzeros@andstop
    \romannumeral0\XINT@rev {#2}%
}%
```

Multiplication spéciale pour emploi par le Produit servant pour le calcul des Puissances (sic)

```
\def\XINT@pow@mul@enter #1\W\X\Y\Z #2#3#4#5%
{%
    \xint@w
    #5\xint@pow@mul@enterw
    #4\xint@pow@mul@enterx
    #3\xint@pow@mul@entery
    #2\xint@pow@mul@enterz
    \W\XINT@pow@mul@start {#2#3#4#5}#1\W\X\Y\Z
}%
\def\xint@pow@mul@enterw
    #1\xint@pow@mul@enterx
    #2\xint@pow@mul@entery
```

```
    #3\xint@pow@mul@enterz
    \W\XINT@pow@mul@start  #4#5\W\X\Y\Z \X\Y\Z
{%
    \XINT@mul@Mr {#3#2#1}#5\Z\Z\Z\Z
}%
\def\xint@pow@mul@enterx
    #1\xint@pow@mul@entery
    #2\xint@pow@mul@enterz
    \W\XINT@pow@mul@start  #3#4\W\X\Y\Z \Y\Z
{%
    \XINT@mul@Mr {#2#1}#4\Z\Z\Z\Z
}%
\def\xint@pow@mul@entery
    #1\xint@pow@mul@enterz
    \W\XINT@pow@mul@start  #2#3\W\X\Y\Z \Z
{%
    \XINT@mul@Mr {#1}#3\Z\Z\Z\Z
}%
\def\XINT@pow@mul@start #1#2\W\X\Y\Z
{\expandafter
    \XINT@pow@mul@main \expandafter
          {\romannumeral0%
           \XINT@mul@Mr {#1}#2\Z\Z\Z\Z
          }#2\W\X\Y\Z
}%
\def\XINT@pow@mul@main #1#2\W\X\Y\Z #3#4#5#6%
{%
    \xint@w
    #6\xint@pow@mul@mainw
    #5\xint@pow@mul@mainx
    #4\xint@pow@mul@mainy
    #3\xint@pow@mul@mainz
    \W\XINT@pow@mul@compute {#1}{#3#4#5#6}#2\W\X\Y\Z
}%
\def\XINT@pow@mul@compute #1#2#3\W\X\Y\Z
{\expandafter
    \XINT@pow@mul@main \expandafter
           {\romannumeral0\expandafter
    \XINT@mul@Ar \expandafter0\expandafter{\expandafter}%
    \romannumeral0\XINT@mul@Mr {#2}#3\Z\Z\Z\Z \W\X\Y\Z 0000#1\W\X\Y\Z
            }#3\W\X\Y\Z
}%
\def\xint@pow@mul@mainw
    #1\xint@pow@mul@mainx
    #2\xint@pow@mul@mainy
    #3\xint@pow@mul@mainz
    \W\XINT@pow@mul@compute  #4#5#6\W\X\Y\Z \X\Y\Z
{%
    \expandafter
    \XINT@pow@add@A \expandafter0\expandafter{\expandafter}%
          \romannumeral0%
    \XINT@mul@Mr  {#3#2#1}#6\Z\Z\Z\Z
```

```
                              \W\X\Y\Z 000#4\W\X\Y\Z
}%
\def\xint@pow@mul@mainx
    #1\xint@pow@mul@mainy
    #2\xint@pow@mul@mainz
    \W\XINT@pow@mul@compute  #3#4#5\W\X\Y\Z \Y\Z
{%
    \expandafter
    \XINT@pow@add@A \expandafter0\expandafter{\expandafter}%
          \romannumeral0%
    \XINT@mul@Mr   {#2#1}#5\Z\Z\Z\Z
                             \W\X\Y\Z  00#3\W\X\Y\Z
}%
\def\xint@pow@mul@mainy
    #1\xint@pow@mul@mainz
    \W\XINT@pow@mul@compute  #2#3#4\W\X\Y\Z \Z
{%
    \expandafter
    \XINT@pow@add@A \expandafter0\expandafter{\expandafter}%
          \romannumeral0%
    \XINT@mul@Mr    {#1}#4\Z\Z\Z\Z
                           \W\X\Y\Z 0#2\W\X\Y\Z
}%
\def\xint@pow@mul@mainz\W\XINT@pow@mul@compute #1#2#3\W\X\Y\Z
{ #1}%
```

ADDITION spéciale pour emploi dans la routine de Multiplication utilisée dans la routine de Produit servant pour le calcul des puissances (sic).

```
\def\XINT@pow@add@A #1#2#3#4#5#6%
{%
    \xint@w
    #3\xint@pow@add@az
    \W\XINT@pow@add@AB #1{#3#4#5#6}{#2}%
}%
\def\xint@pow@add@az\W\XINT@pow@add@AB #1#2%
{%
    \XINT@pow@add@AC@checkcarry #1%
}%
\def\XINT@pow@add@AC@checkcarry #1%
{%
    \xint@zero #1\xint@pow@add@AC@nocarry 0\XINT@pow@add@C
}%
\def\xint@pow@add@AC@nocarry 0\XINT@pow@add@C
{%
    \XINT@pow@add@F
}%
\def\XINT@pow@add@AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
{%
    \XINT@pow@add@ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
}%
\def\XINT@pow@add@ABE #1#2#3#4#5#6%
{\expandafter
```

```
2174     \XINT@pow@add@ABEA\the\numexpr #1+10#5#4#3#2+#6\relax
2175 }%
2176 \def\XINT@pow@add@ABEA #1#2#3#4#5#6#7%
2177 {%
2178    \XINT@pow@add@A  #2{#7#6#5#4#3}%<-- attention on met donc \`a droite
2179 }%
2180 \def\XINT@pow@add@C #1#2#3#4#5%
2181 {%
2182     \xint@w
2183     #5\xint@pow@add@cw
2184     #4\xint@pow@add@cx
2185     #3\xint@pow@add@cy
2186     #2\xint@pow@add@cz
2187     \W\XINT@pow@add@CD   {#5#4#3#2}{#1}%
2188 }%
2189 \def\XINT@pow@add@CD #1%
2190 {\expandafter
2191     \XINT@pow@add@CC\the\numexpr 1+10#1\relax
2192 }%
2193 \def\XINT@pow@add@CC #1#2#3#4#5#6#7%
2194 {%
2195     \XINT@pow@add@AC@checkcarry  #2{#7#6#5#4#3}%
2196 }%
2197 \def\xint@pow@add@cw
2198     #1\xint@pow@add@cx
2199     #2\xint@pow@add@cy
2200     #3\xint@pow@add@cz
2201     \W\XINT@pow@add@CD
2202 {\expandafter
2203     \XINT@pow@add@CDw\the\numexpr 1+10#1#2#3\relax
2204 }%
2205 \def\XINT@pow@add@CDw #1#2#3#4#5#6%
2206 {%
2207     \xint@quatrezeros #2#3#4#5\XINT@pow@add@endDw@zeros
2208                             0000\XINT@pow@add@endDw #2#3#4#5%
2209 }%
2210 \def\XINT@pow@add@endDw@zeros 0000\XINT@pow@add@endDw 0000#1\X\Y\Z{ #1}%
2211 \def\XINT@pow@add@endDw #1#2#3#4#5\X\Y\Z{ #5#4#3#2#1}%
2212 \def\xint@pow@add@cx
2213     #1\xint@pow@add@cy
2214     #2\xint@pow@add@cz
2215     \W\XINT@pow@add@CD
2216 {\expandafter
2217     \XINT@pow@add@CDx\the\numexpr 1+100#1#2\relax
2218 }%
2219 \def\XINT@pow@add@CDx #1#2#3#4#5#6%
2220 {%
2221     \xint@quatrezeros #2#3#4#5\XINT@pow@add@endDx@zeros
2222                             0000\XINT@pow@add@endDx #2#3#4#5%
2223 }%
2224 \def\XINT@pow@add@endDx@zeros 0000\XINT@pow@add@endDx 0000#1\Y\Z{ #1}%
2225 \def\XINT@pow@add@endDx #1#2#3#4#5\Y\Z{ #5#4#3#2#1}%
```

```
\def\xint@pow@add@cy
    #1\xint@pow@add@cz
    \W\XINT@pow@add@CD
{\expandafter
    \XINT@pow@add@CDy\the\numexpr 1+1000#1\relax
}%
\def\XINT@pow@add@CDy #1#2#3#4#5#6%
{%
    \xint@quatrezeros #2#3#4#5\XINT@pow@add@endDy@zeros
                           0000\XINT@pow@add@endDy #2#3#4#5%
}%
\def\XINT@pow@add@endDy@zeros 0000\XINT@pow@add@endDy 0000#1\Z{ #1}%
\def\XINT@pow@add@endDy #1#2#3#4#5\Z{ #5#4#3#2#1}%
\def\xint@pow@add@cz\W\XINT@pow@add@CD #1#2{ #21000}%
\def\XINT@pow@add@F #1#2#3#4#5%
{%
    \xint@w
    #5\xint@pow@add@Gw
    #4\xint@pow@add@Gx
    #3\xint@pow@add@Gy
    #2\xint@pow@add@Gz
    \W\XINT@pow@add@G   {#2#3#4#5}{#1}%
}%
\def\XINT@pow@add@G #1#2%
{%
    \XINT@pow@add@F {#2#1}%
}%
\def\xint@pow@add@Gw
    #1\xint@pow@add@Gx
    #2\xint@pow@add@Gy
    #3\xint@pow@add@Gz
    \W\XINT@pow@add@G #4%
{%
    \xint@quatrezeros  #3#2#10\XINT@pow@add@endGw@zeros
                           0000\XINT@pow@add@endGw #3#2#10%
}%
\def\XINT@pow@add@endGw@zeros 0000\XINT@pow@add@endGw 0000#1\X\Y\Z{ #1}%
\def\XINT@pow@add@endGw #1#2#3#4#5\X\Y\Z{ #5#1#2#3#4}%
\def\xint@pow@add@Gx
    #1\xint@pow@add@Gy
    #2\xint@pow@add@Gz
    \W\XINT@pow@add@G #3%
{%
    \xint@quatrezeros  #2#100\XINT@pow@add@endGx@zeros
                         0000\XINT@pow@add@endGx #2#100%
}%
\def\XINT@pow@add@endGx@zeros 0000\XINT@pow@add@endGx 0000#1\Y\Z{ #1}%
\def\XINT@pow@add@endGx #1#2#3#4#5\Y\Z{ #5#1#2#3#4}%
\def\xint@pow@add@Gy
    #1\xint@pow@add@Gz
    \W\XINT@pow@add@G #2%
{%
```

```
    \xint@quatrezeros   #1000\XINT@pow@add@endGy@zeros
                         0000\XINT@pow@add@endGy #1000%
}%
\def\XINT@pow@add@endGy@zeros 0000\XINT@pow@add@endGy 0000#1\Z{ #1}%
\def\XINT@pow@add@endGy #1#2#3#4#5\Z{ #5#1#2#3#4}%
\def\xint@pow@add@Gz\W\XINT@pow@add@G #1#2{ #2}%
```

## 12.25 \xintDivision, \xintQuo, \xintRem

```
\def\xintQuo {\romannumeral0\xintquo }%
\def\xintRem {\romannumeral0\xintrem }%
\def\xintquo {\expandafter
                  \xint@firstoftwo@andstop
               \romannumeral0\xintdivision }%
\def\xintrem {\expandafter
                  \xint@secondoftwo@andstop
               \romannumeral0\xintdivision }%
  #1 = A, #2 = B. On calcule le quotient de A par B
\def\xintDivision {\romannumeral0\xintdivision }%
\def\xintdivision #1%
{%
    \expandafter\expandafter\expandafter
        \xint@division
    \expandafter\expandafter\expandafter
        {#1}%
}%
\def\xint@division #1#2%
{%
    \expandafter\expandafter\expandafter
    \XINT@div@fork #2\Z #1\Z
}%
\def\XINT@Division #1#2{\romannumeral0\XINT@div@fork #2\Z #1\Z }%

  #1#2 = 2e input = diviseur = B
  #3#4 = 1er input = divisé = A
\def\XINT@div@fork #1#2\Z #3#4\Z
{%
    \xint@UDzerofork
      #1\dummy \XINT@div@BisZero
      #3\dummy \XINT@div@AisZero
       0\dummy
        {\xint@UDsignfork
           #1\dummy \XINT@div@BisNegative  % B < 0
           #3\dummy \XINT@div@AisNegative  % A < 0, B > 0
            -\dummy \XINT@div@plusplus     % B > 0, A > 0
         \xint@UDkrof }%
    \xint@UDkrof
    {#2}{#4}#1#3% #1#2=B, #3#4=A
}%
\def\XINT@div@BisZero #1#2#3#4%
    {\xintError:DivisionByZero\space {0}{0}}%
\def\XINT@div@AisZero #1#2#3#4{ {0}{0}}%
```

```
  jusqu'à présent c'est facile.
  minusplus signifie B < 0, A > 0
  plusminus signifie B > 0, A < 0
  Ici #3#1 correspond au diviseur B et #4#2 au divisé A
\def\XINT@div@plusplus #1#2#3#4%
{%
    \XINT@div@prepare {#3#1}{#4#2}%
}%

  B = #3#1 < 0, A non nul positif ou négatif

\def\XINT@div@BisNegative #1#2#3#4%
{%
    \expandafter\XINT@div@BisNegative@post
    \romannumeral0\XINT@div@fork #1\Z #4#2\Z
}%
\def\XINT@div@BisNegative@post #1#2%
{%
    \expandafter\space\expandafter
        {\romannumeral0\XINT@opp #1}{#2}%
}%

  B = #3#1 > 0, A =-#2< 0

\def\XINT@div@AisNegative #1#2#3#4%
{%
    \expandafter\XINT@div@AisNegative@post
    \romannumeral0\XINT@div@prepare {#3#1}{#2}{#3#1}%
}%
\def\XINT@div@AisNegative@post #1#2%
{%
    \ifcase\xintSgn {#2}
       \expandafter \XINT@div@AisNegative@zerorem
    \or
       \expandafter \XINT@div@AisNegative@posrem
    \fi
    {#1}{#2}%
}%

  en #3 on a une copie de B (à l'endroit)

\def\XINT@div@AisNegative@zerorem #1#2#3%
{%
    \expandafter\space\expandafter
       {\romannumeral0\XINT@opp #1}{0}%
}%

  #1 = quotient, #2 = reste, #3 = diviseur initial (à l'endroit)
```

```
\def\XINT@div@AisNegative@posrem #1%
{%
    \expandafter
        \XINT@div@AisNegative@posrem@b
    \expandafter
       {\romannumeral0\xintopp {\XINT@Add{#1}{1}}}%
}%
```

```
  remplace Reste par B - Reste, après avoir remplacé Q par -(Q+1)
  de sorte que la formule a = qb + r, 0<= r < |b| est valable
```

```
\def\XINT@div@AisNegative@posrem@b #1#2#3%
{%
    \expandafter
        \xint@exchangetwo@keepbraces@andstop
    \expandafter
    {\romannumeral0\XINT@sub {#3}{#2}}{#1}%
}%
```

```
  par la suite A et B sont > 0.
  #1 = B. Pour le moment à l'endroit.
```

```
\def\XINT@div@prepare #1%
{%
    \expandafter
      \XINT@div@prepareB@a
    \expandafter
    {\romannumeral0\XINT@length {#1}}{#1}% B > 0 ici
}%
```

```
  Calcul du plus petit K = 4n >= longueur de B
```

```
\def\XINT@div@prepareB@a #1%
{%
    \expandafter\XINT@div@prepareB@b\expandafter
    {\the\numexpr 4*((#1+1)/4)\relax}{#1}%
}%
```

```
  #1 = K
```

```
\def\XINT@div@prepareB@b #1#2%
{%
    \expandafter\XINT@div@prepareB@c \expandafter
       {\the\numexpr #1-#2\relax}{#1}%
}%
```

```
  #1 = c
```

```
\def\XINT@div@prepareB@c #1%
{%
    \ifcase #1
          \expandafter\XINT@div@prepareB@di
    \or   \expandafter\XINT@div@prepareB@dii
    \or   \expandafter\XINT@div@prepareB@diii
    \else \expandafter\XINT@div@prepareB@div
```

```
    \fi
}%
\def\XINT@div@prepareB@di   {\XINT@div@prepareB@e {}{0}}%
\def\XINT@div@prepareB@dii  {\XINT@div@prepareB@e {0}{1}}%
\def\XINT@div@prepareB@diii {\XINT@div@prepareB@e {00}{2}}%
\def\XINT@div@prepareB@div  {\XINT@div@prepareB@e {000}{3}}%

  #1 = zéros à rajouter à B, #2=c, #3=K, #4 = B

\def\XINT@div@prepareB@e #1#2#3#4%
{%
    \XINT@div@prepareB@f #4#1\Z {#3}{#2}{#1}%
}%

  x = #1#2#3#4 = 4 premiers chiffres de B. #1 est non nul.
  Ensuite on renverse B pour calculs plus rapides par la suite.

\def\XINT@div@prepareB@f #1#2#3#4#5\Z
{%
    \expandafter
      \XINT@div@prepareB@g
    \expandafter
    {\romannumeral0\XINT@rev {#1#2#3#4#5}}{#1#2#3#4}%
}%

  #3= K, #4 = c, #5= {} ou {0} ou {00} ou {000}, #6 = A initial
  #1 = B préparé et renversé, #2 = x = quatre premiers chiffres
  On multiplie aussi A par 10^c.
  B, x, K, c, {} ou {0} ou {00} ou {000}, A initial

\def\XINT@div@prepareB@g #1#2#3#4#5#6%
{%
    \XINT@div@prepareA@a {#6#5}{#2}{#3}{#1}{#4}%
}%

  A, x, K, B, c,

\def\XINT@div@prepareA@a #1%
{%
    \expandafter
      \XINT@div@prepareA@b
    \expandafter
    {\romannumeral0\XINT@length {#1}}{#1}% A >0 ici
}%

  L0, A, x, K, B, ...

\def\XINT@div@prepareA@b #1%
{%
    \expandafter\XINT@div@prepareA@c\expandafter
    {\the\numexpr 4*((#1+1)/4)\relax}{#1}%
}%

  L, L0, A, x, K, B,...
```

```
\def\XINT@div@prepareA@c #1#2%
{%
    \expandafter\XINT@div@prepareA@d \expandafter
       {\the\numexpr #1-#2\relax}{#1}%
}%
\def\XINT@div@prepareA@d #1%
{%
    \ifcase #1
          \expandafter\XINT@div@prepareA@di
    \or   \expandafter\XINT@div@prepareA@dii
    \or   \expandafter\XINT@div@prepareA@diii
    \else \expandafter\XINT@div@prepareA@div
    \fi
}%
\def\XINT@div@prepareA@di   {\XINT@div@prepareA@e {}}%
\def\XINT@div@prepareA@dii  {\XINT@div@prepareA@e {0}}%
\def\XINT@div@prepareA@diii {\XINT@div@prepareA@e {00}}%
\def\XINT@div@prepareA@div  {\XINT@div@prepareA@e {000}}%
```

```
  #1#3 = A préparé, #2 = longueur de ce A préparé,
```

```
\def\XINT@div@prepareA@e #1#2#3%
{%
    \XINT@div@startswitch {#1#3}{#2}%
}%
```

```
  A, L, x, K, B, c
```

```
\def\XINT@div@startswitch #1#2#3#4%
{%
    \ifnum #2 > #4
      \expandafter\XINT@div@body@a
    \else
    \ifnum #2 = #4
      \expandafter\expandafter\expandafter
          \XINT@div@final@a
    \else
      \expandafter\expandafter\expandafter
          \XINT@div@finished@a
    \fi\fi {#1}{#4}{#3}{0000}{#2}%
}%
```

```
  A, K, x, Q, L, B, c
  ---- "Finished"
```

```
\def\XINT@div@finished@a #1#2#3%
{%
    \expandafter
      \XINT@div@finished@b
    \expandafter
      {\romannumeral0\XINT@cuz {#1}}%
}%
```

```
  A, Q, L, B, c
  no leading zeros in A at this stage
```

```
\def\XINT@div@finished@b #1#2#3#4#5%
{%
    \ifcase \XINT@Sgn {#1}
       \xint@afterfi {\XINT@div@finished@c {0}}%
    \or
       \xint@afterfi {\expandafter\XINT@div@finished@c
                      \expandafter
         {\romannumeral0\XINT@dsh@checksignx #5\Z {#1}}}%
    \fi
    {#2}%
}%
```

```
  Reste Final, Q à renverser
  #2 = Quotient, #1 = Reste.
```

```
\def\XINT@div@finished@c #1#2%
{%
    \expandafter
         \space
    \expandafter
        {\romannumeral0\expandafter\xint@cleanupzeros@andstop
         \romannumeral0\XINT@rev {#2}}{#1}%
}%
```

```
  ---- "Final"
  A, K, x, Q, L, B, c
```

```
\def\XINT@div@final@a #1%
{%
    \XINT@div@final@b #1\Z
}%
\def\XINT@div@final@b #1#2#3#4#5\Z
{%
    \xint@quatrezeros #1#2#3#4\xint@div@final@c0000%
    \XINT@div@final@c {#1#2#3#4}{#1#2#3#4#5}%
}%
\def\xint@div@final@c0000\XINT@div@final@c #1%
                              {\XINT@div@finished@a }%
```

```
  a, A, K, x, Q, L, B ,c
```

```
\def\XINT@div@final@c #1#2#3#4%
{%
    \expandafter
    \XINT@div@final@d
    \expandafter
    {\the\numexpr #1/#4\relax}{#2}%
}%
```

```
  q, A, Q, L, B à l'envers sur 4n, c
  1.01 code ré-écrit pour optimisations diverses
```

```
\def\XINT@div@final@d #1#2#3#4#5% q,A,Q,L,B puis c
{%
    \expandafter
       \XINT@div@final@da
    \expandafter
    {\romannumeral0\XINT@mul@M {#1}#5\Z\Z\Z\Z }%
    {\romannumeral0\xint@cleanupzeros@andstop #2}%
    {#1}{#3}{#5}%
}%
\def\XINT@div@final@da #1#2%
{%
    \expandafter\XINT@div@final@db\expandafter {#2}{#1}%
}%
\def\XINT@div@final@db #1#2% A,qB, puis q,Q,B,c
{%
    \ifcase\XINT@Geq {#1}{#2}
       \expandafter\XINT@div@final@dc  % A < qB
    \or\expandafter\XINT@div@final@e   % A au moins qB
    \fi
    {#1}{#2}%
}%
\def\XINT@div@final@e #1#2#3#4#5% A,qB,q,Q,B,puis c
{%
    \expandafter\XINT@div@final@f
    \expandafter{\romannumeral0\xintsub {#1}{#2}}%
    {\romannumeral0\xintadd {\XINT@Rev@andcleanupzeros{#4}}{#3}}%
}%
\def\XINT@div@final@dc #1#2#3% A sans leading zeros,trash,q,Q,B,c
{%
    \expandafter\XINT@div@final@dd
    \expandafter{\the\numexpr #3-1\relax}{#1}%
}%
\def\XINT@div@final@dd #1#2#3#4% q,A,Q,B puis c
{%
    \expandafter\XINT@div@final@f
    \expandafter{\romannumeral0\xintsub
                 {#2}{\romannumeral0\XINT@mul@M {#1}#4\Z\Z\Z\Z }}%
    {\romannumeral0\xintadd {\XINT@Rev@andcleanupzeros{#3}}{#1}}%
}%
\def\XINT@div@final@f #1#2#3% R,Q à développer,c
{%
    \ifcase \XINT@Sgn {#1}
       \xint@afterfi {\XINT@div@final@end {0}}%
    \or
       \xint@afterfi {\expandafter\XINT@div@final@end
                      \expandafter % pas de leading zeros dans #1=R
                      {\romannumeral0\XINT@dsh@checksignx #3\Z {#1}}}%
    \fi
    {#2}%
}%
\def\XINT@div@final@end #1#2%
{%
    \expandafter\space\expandafter {#2}{#1}%
```

```
}%

  Boucle Principale
  A, K, x, Q, L, B, c

\def\XINT@div@body@a #1%
{%
    \XINT@div@body@b #1\Z
}%
\def\XINT@div@body@b #1#2#3#4#5#6#7#8#9\Z
{%
    \XINT@div@body@c
    {#1#2#3#4#5#6#7#8#9}%
    {#1#2#3#4#5#6#7#8}%
}%

  A, a, K, x, Q, L, B, c

\def\XINT@div@body@c #1#2#3%
{%
    \XINT@div@body@d {#3}{}#1\Z {#2}{#3}%
}%
\def\XINT@div@body@d #1#2#3#4#5#6%
{%
    \ifnum #1 > 0
        \expandafter
        \XINT@div@body@d
        \expandafter
        {\the\numexpr #1-4\expandafter }%
    \else
        \expandafter
        \XINT@div@body@e
    \fi
    {#6#5#4#3#2}%
}%
\def\XINT@div@body@e #1#2\Z #3%
{%
    \XINT@div@body@f {#3}{#1}{#2}%
}%

  a, alpha, alpha', K, x, Q, L, B, c

\def\XINT@div@body@f #1#2#3#4#5#6#7#8%
{%
    \expandafter\XINT@div@body@g
    \expandafter
    {\the\numexpr (#1+(#5+1)/2)/(#5+1)-1\relax }%
    {#2}{#8}{#4}{#5}{#3}{#6}{#7}{#8}%
}%

  q1, alpha, B, K, x, alpha', Q, L, B, c
```

```
\def\XINT@div@body@g #1#2#3%
{%
    \expandafter
       \XINT@div@body@h
    \romannumeral0\XINT@div@sub@xpxp
       {\romannumeral0\XINT@pow@mul@enter #3\W\X\Y\Z #1\W\X\Y\Z }%
       {#2}\Z
    {#3}{#1}%
}%
```

```
alpha1 = alpha-q1 B, \Z, B, q1, K, x, alpha', Q, L, B, c
```

```
\def\XINT@div@body@h #1#2#3#4#5#6#7#8#9\Z
{%
    \ifnum #1#2#3#4>0
        \xint@afterfi{\XINT@div@body@i {#1#2#3#4#5#6#7#8}}%
    \else
        \expandafter\XINT@div@body@k
    \fi
    {#1#2#3#4#5#6#7#8#9}%
}%
```

```
a1, alpha1, B, q1, K, x, alpha', Q, L, B, c
```

```
\def\XINT@div@body@i #1#2#3#4#5#6%
{%
    \expandafter\XINT@div@body@j
    \expandafter{\the\numexpr (#1+(#6+1)/2)/(#6+1)-1\relax }%
       {#2}{#3}{#4}{#5}{#6}%
}%
```

```
q2, alpha1, B, q1, K, x, alpha', Q, L, B, c
```

```
\def\XINT@div@body@j #1#2#3#4%
{%
    \expandafter
       \XINT@div@body@l
    \expandafter{\romannumeral0\XINT@div@sub@xpxp
       {\romannumeral0\XINT@pow@mul@enter #3\W\X\Y\Z #1\W\X\Y\Z }%
       {\XINT@Rev{#2}}}%
    {#4+#1}%
}%
```

```
alpha2, q1+q2, K, x, alpha', Q, L, B, c
attention body@j -> body@l
alpha1, B, q=q1, K, x, alpha', Q, L, B, c
```

```
\def\XINT@div@body@k #1#2%
{%
    \XINT@div@body@l {#1}%
}%
```

```
alpha2, q= q1+q2, K, x, alpha', Q, L, B, c
```

```
\def\XINT@div@body@l #1#2#3#4#5#6#7%
{%
   \expandafter
       \XINT@div@body@m
   \the\numexpr 100000000+#2\relax
       {#6}{#3}{#7}{#1#5}{#4}%
}%
```

chiffres de q, Q, K, L, A', x, B, c

```
\def\XINT@div@body@m #1#2#3#4#5#6#7#8#9%
{%
    \ifnum #2#3#4#5>0
       \xint@afterfi {\XINT@div@body@n {#9#8#7#6#5#4#3#2}}%
    \else
       \xint@afterfi {\XINT@div@body@n {#9#8#7#6}}%
    \fi
}%
```

q renversé, Q, K, L, A', x, B, c

```
\def\XINT@div@body@n #1#2%
{%
    \expandafter\XINT@div@body@o\expandafter
    {\romannumeral0\XINT@sum@A 0{}#1\W\X\Y\Z #2\W\X\Y\Z }%
}%
```

q+Q, K, L, A', x, B, c

```
\def\XINT@div@body@o #1#2#3#4%
{%
    \XINT@div@body@p {#3}{#2}{}#4\Z {#1}%
}%
```

L, K, {}, A'\Z, q+Q, x, B, c

```
\def\XINT@div@body@p #1#2#3#4#5#6#7%
{%
    \ifnum #1 > #2
        \xint@afterfi
        {\ifnum #4#5#6#7 > 0
           \expandafter\XINT@div@body@q
         \else
           \expandafter\XINT@div@body@repeatp
         \fi }%
    \else
        \expandafter\XINT@div@gotofinal@a
    \fi
    {#1}{#2}{#3}#4#5#6#7%
}%
```

L, K, zeros,  A' avec moins de zéros\Z, q+Q, x, B, c

```
\def\XINT@div@body@repeatp #1#2#3#4#5#6#7%
{%
    \expandafter
        \XINT@div@body@p
    \expandafter
        {\the\numexpr #1-4\relax}{#2}{0000#3}%
}%
```

```
L -> L-4, zeros->zeros+0000, répéter jusqu'à ce que soit L=K
soit on ne trouve plus 0000
nouveau L, K, zeros, nouveau A=#4, Q+q, x, B, c
```

```
\def\XINT@div@body@q #1#2#3#4\Z #5#6%
{%
    \XINT@div@body@a {#4}{#2}{#6}{#3#5}{#1}%
}%
```

```
A, K, x, Q, L, B, c --> iterate
----
Boucle Principale achevée
ATTENTION IL FAUT AJOUTER 4 ZEROS DE MOINS QUE CEUX
QUI ONT ÉTÉ PRÉPARÉS DANS #3!!
L, K (L=K), zeros, A\Z, Q, x, B, c
```

```
\def\XINT@div@gotofinal@a #1#2#3#4\Z %
{%
    \XINT@div@gotofinal@b #3\Z {#4}{#1}%
}%
```

```
zeros\Z, A, L=K, Q, x, B, c
```

```
\def\XINT@div@gotofinal@b 0000#1\Z #2#3#4#5%
{%
    \XINT@div@final@a {#2}{#3}{#5}{#1#4}{#3}%
}%
```

```
A, L=K, x, Q avec zéros, L, B, c
La soustraction spéciale. Étendre deux fois les arguments
pour \XINT@div@sub@enter longueur multiple de 4 on sait que #2>#1,
```

```
\def\XINT@div@sub@xpxp #1%
{%
    \expandafter
        \XINT@div@sub@xpxp@
    \expandafter
    {#1}%
}%
\def\XINT@div@sub@xpxp@ #1#2%
{%
    \expandafter\expandafter\expandafter
        \XINT@div@sub@xpxp@@
     #2\W\X\Y\Z #1\W\X\Y\Z
}%
\def\XINT@div@sub@xpxp@@
{%
```

```
    \XINT@div@sub@A 1{}%
}%
\def\XINT@div@sub@A #1#2#3#4#5#6%
{%
    \xint@w
    #3\xint@div@sub@az
    \W\XINT@div@sub@B #1{#3#4#5#6}{#2}%
}%
\def\XINT@div@sub@B #1#2#3#4\W\X\Y\Z #5#6#7#8%
{%
    \xint@w
    #5\xint@div@sub@bz
    \W\XINT@div@sub@onestep #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
}%
\def\XINT@div@sub@onestep #1#2#3#4#5#6%
{\expandafter
    \XINT@div@sub@backtoA\the\numexpr 11#5#4#3#2-#6+#1-1\relax.%
}%
\def\XINT@div@sub@backtoA #1#2#3.#4%
{%
    \XINT@div@sub@A #2{#3#4}%
}%
\def\xint@div@sub@bz
    \W\XINT@div@sub@onestep #1#2#3#4#5#6#7%
{%
    \xint@UDzerofork
      #1\dummy  \XINT@div@sub@C   %
       0\dummy  \XINT@div@sub@D   % pas de retenue
    \xint@UDkrof
    {#7}#2#3#4#5%
}%
\def\XINT@div@sub@D #1#2\W\X\Y\Z
{%
    \expandafter\space
    \romannumeral0%
    \XINT@rord@main {}#2%
      \xint@UNDEF
        \xint@undef\xint@undef\xint@undef\xint@undef
        \xint@undef\xint@undef\xint@undef\xint@undef
      \xint@UNDEF
    #1%
}%
\def\XINT@div@sub@C #1#2#3#4#5%
{%
    \xint@w
    #2\xint@div@sub@cz
    \W\XINT@div@sub@AC@onestep {#5#4#3#2}{#1}%
}%
\def\XINT@div@sub@AC@onestep #1%
{\expandafter
    \XINT@div@sub@backtoC\the\numexpr 11#1-1\relax.%
}%
```

```
\def\XINT@div@sub@backtoC #1#2#3.#4%
{%
    \XINT@div@sub@AC@checkcarry #2{#3#4}% la retenue va \^etre examin\'ee
}%
\def\XINT@div@sub@AC@checkcarry #1%
{%
    \xint@one #1\xint@div@sub@AC@nocarry 1\XINT@div@sub@C
}%
\def\xint@div@sub@AC@nocarry 1\XINT@div@sub@C #1#2\W\X\Y\Z
{%
    \expandafter\space
    \romannumeral0%
    \XINT@rord@main {}#2%
      \xint@UNDEF
        \xint@undef\xint@undef\xint@undef\xint@undef
        \xint@undef\xint@undef\xint@undef\xint@undef
      \xint@UNDEF
    #1%
}%
\def\xint@div@sub@cz\W\XINT@div@sub@AC@onestep #1#2{ #2}%
\def\xint@div@sub@az\W\XINT@div@sub@B #1#2#3#4\Z { #3}%
```

```
 ------------------------------------------------------------------
 ------------------------------------------------------------------
  DECIMAL OPERATIONS: FIRST DIGIT, LASTDIGIT, ODDNESS,
  MULTIPLICATION BY TEN, QUOTIENT BY TEN, QUOTIENT OR
  MULTIPLICATION BY POWER OF TEN, SPLIT OPERATION.
```

## 12.26 \xintFDg

```
    FIRST DIGIT
```

```
\def\xintFDg {\romannumeral0\xintfdg }%
\def\xintfdg #1%
{%
    \expandafter\expandafter\expandafter
    \XINT@fdg #1\W\Z
}%
\def\XINT@FDg #1{\romannumeral0\XINT@fdg #1\W\Z }%
\def\XINT@fdg #1#2%
{%
    \xint@xpxp@andstop
    \xint@UDzerominusfork
      #1-\dummy  {\expandafter 0}%      zero
      0#1\dummy  {\expandafter #2}%     negative
       0-\dummy  {\expandafter #1}%  positive
    \xint@UDkrof
    \xint@z
}%
```

## 12.27 \xintLDg

```
    LAST DIGIT
```

```
\def\xintLDg {\romannumeral0\xintldg }%
\def\xintldg #1%
{%
    \expandafter\expandafter\expandafter
        \XINT@ldg
    \expandafter\expandafter\expandafter
    {#1}%
}%
\def\XINT@LDg #1{\romannumeral0\XINT@ldg {#1}}%
\def\XINT@ldg #1%
{%
    \expandafter
    \XINT@ldg@
    \romannumeral0\XINT@rev {#1}\Z
}%
\def\XINT@ldg@ #1%
{%
    \expandafter\space\expandafter #1\xint@z
}%
```

## 12.28 \xintOdd

ODDNESS

```
\def\xintOdd {\romannumeral0\xintodd }%
\def\xintodd #1%
{%
    \ifodd\xintLDg{#1}
        \xint@afterfi{ 1}%
    \else
        \xint@afterfi{ 0}%
    \fi
}%
\def\XINT@Odd #1%
{\romannumeral0%
    \ifodd\XINT@LDg{#1}
        \xint@afterfi{ 1}%
    \else
        \xint@afterfi{ 0}%
    \fi
}%
```

## 12.29 \xintDSL

DECIMAL SHIFT LEFT (=MULTIPLICATION PAR 10)

```
\def\xintDSL {\romannumeral0\xintdsl }%
\def\xintdsl #1%
{%
    \expandafter\expandafter\expandafter
      \XINT@dsl #1\Z
}%
\def\XINT@DSL #1{\romannumeral0\XINT@dsl #1\Z }%
```

```
\def\XINT@dsl #1%
{%
    \xint@zero #1\xint@dsl@zero 0\XINT@dsl@ #1%
}%
\def\xint@dsl@zero 0\XINT@dsl@ 0#1\Z { 0}%
\def\XINT@dsl@ #1\Z { #10}%
```

## 12.30 \xintDSR

```
    DECIMAL SHIFT RIGHT (=DIVISION PAR 10)
```

```
\def\xintDSR {\romannumeral0\xintdsr }%
\def\xintdsr #1%
{%
    \expandafter\expandafter\expandafter
       \XINT@dsr@a
    \expandafter\expandafter\expandafter
    {#1}\W\Z
}%
\def\XINT@DSR #1{\romannumeral0\XINT@dsr@a {#1}\W\Z }%
\def\XINT@dsr@a
{%
    \expandafter
       \XINT@dsr@b
    \romannumeral0\XINT@rev
}%
\def\XINT@dsr@b #1#2#3\Z
{%
    \xint@w #2\xint@dsr@onedigit\W
    \xint@minus #2\xint@dsr@onedigit-%
    \expandafter
      \XINT@dsr@removew
    \romannumeral0\XINT@rev {#2#3}%
}%
\def\xint@dsr@onedigit #1\XINT@rev #2{ 0}%
\def\XINT@dsr@removew #1\W { }%
```

## 12.31 \xintDSH, \xintDSHr

```
     DECIMAL SHIFTS
    \xintDSH {x}{A}
  si x <= 0, fait A -> A.10^(|x|)
  si x >  0, et A >=0, fait A -> quo(A,10^(x))
  si x >  0, et A < 0, fait A -> -quo(-A,10^(x))
  (donc pour x > 0 c'est comme DSR itéré x fois)
  \xintDSHr donne le 'reste'.
```

```
\def\xintDSHr {\romannumeral0\xintdshr }%
\def\xintdshr #1%
{\expandafter\expandafter\expandafter
                  \XINT@dshr@checkxpositive #1\Z
}%
\def\XINT@dshr@checkxpositive #1%
```

```
{%
    \xint@UDzerominusfork
      0#1\dummy \XINT@dshr@xzeroorneg
      #1-\dummy \XINT@dshr@xzeroorneg
       0-\dummy \XINT@dshr@xpositive
    \xint@UDkrof #1%
}%
\def\XINT@dshr@xzeroorneg #1\Z #2{ 0}%
\def\XINT@dshr@xpositive #1\Z
{%
    \expandafter
    \xint@secondoftwo@andstop
    \romannumeral0\xintdsx {#1}%
}%
\def\xintDSH {\romannumeral0\xintdsh }%
\def\xintdsh #1#2%
{%
    \expandafter\expandafter\expandafter
      \xint@dsh
    \expandafter\expandafter\expandafter
      {#2}{#1}%
}%
\def\xint@dsh #1#2%
{%
    \expandafter\expandafter\expandafter
        \XINT@dsh@checksignx
     #2\Z {#1}%
}%
\def\XINT@dsh@checksignx #1%
{%
    \xint@UDzerominusfork
      #1-\dummy  \XINT@dsh@xiszero
      0#1\dummy  \XINT@dsx@xisNeg
       0-\dummy  {\XINT@dsh@xisPos #1}%
    \xint@UDkrof
}%
\def\XINT@dsh@xiszero #1\Z #2{ #2}%
\def\XINT@dsh@xisPos #1\Z #2%
{%
    \expandafter
    \xint@firstoftwo@andstop
    \romannumeral0\XINT@dsx@checksignA #2\Z {#1}%
}%
```

## 12.32 \xintDSx

Je fais cette routine pour la version 1.01, après modification de \xintDecSplit. Dorénavant \xintDSx fera appel à \xintDecSplit et de même \xintDSH fera appel à \xintDSx. J'ai donc supprimé entièrement l'ancien code de \xintDSH et re-écrit entièrement celui de \xintDecSplit pour x positif.
si x <= 0, fait A -> A.10^(|x|)
si x > 0, et A >=0, fait A -> {quo(A,10^(x))}{rem(A,10^(x))}

```
   si x >  0, et A < 0, d'abord on calcule {quo(-A,10^(x))}{rem(-A,10^(x))}
      puis, si le premier n'est pas nul on lui donne le signe -
            si le premier est nul on donne le signe - au second.
   On peut donc toujours reconstituer l'original A par 10^x Q \pm R
   où il faut prendre le signe plus si Q est positif ou nul et le signe moins si
   Q est strictement négatif.
2920 \def\xintDSx {\romannumeral0\xintdsx }%
2921 \def\xintdsx #1#2%
2922 {%
2923     \expandafter\expandafter\expandafter
2924         \xint@dsx
2925     \expandafter\expandafter\expandafter
2926     {#2}{#1}%
2927 }%
2928 \def\xint@dsx #1#2%
2929 {%
2930     \expandafter\expandafter\expandafter
2931         \XINT@dsx@checksignx
2932      #2\Z {#1}%
2933 }%
2934 \def\XINT@DSx #1#2{\romannumeral0\XINT@dsx@checksignx #1\Z {#2}}%
2935 \def\XINT@dsx #1#2{\XINT@dsx@checksignx #1\Z {#2}}%
2936 \def\XINT@dsx@checksignx #1%
2937 {%
2938     \xint@UDzerominusfork
2939       #1-\dummy  \XINT@dsx@xisZero
2940       0#1\dummy  \XINT@dsx@xisNeg
2941        0-\dummy  {\XINT@dsx@xisPos #1}%
2942     \xint@UDkrof
2943 }%
2944 \def\XINT@dsx@xisZero #1\Z #2{ {#2}{0}}%
2945 \def\XINT@dsx@xisNeg #1\Z
2946 {%
2947     \ifnum\XINT@Len {#1} > 9
2948        \xint@afterfi {\xintError:TooBigDecimalShift
2949                       \XINT@dsx@toobigx }%
2950     \else
2951        \expandafter \XINT@dsx@zeroloop
2952     \fi
2953     {#1}\Z
2954 }%
2955 \def\XINT@dsx@toobigx #1\Z #2{ #2}%
2956 \def\XINT@dsx@zeroloop #1%
2957 {%
2958     \ifcase #1
2959         \expandafter \XINT@dsx@exit
2960     \or
2961         \expandafter \XINT@dsx@exiti
2962     \or
2963         \expandafter \XINT@dsx@exitii
2964     \or
2965         \expandafter \XINT@dsx@exitiii
```

```
    \or
        \expandafter \XINT@dsx@exitiv
    \or
        \expandafter \XINT@dsx@exitv
    \or
        \expandafter \XINT@dsx@exitvi
    \or
        \expandafter \XINT@dsx@exitvii
    \else
        \xint@afterfi
         {\expandafter
          \XINT@dsx@zeroloop
          \expandafter {\the\numexpr #1-8}00000000%
         }%
    \fi
}%
\def\XINT@dsx@exit #1\Z
                {\XINT@dsx@addzeros {#1}}%
\def\XINT@dsx@exiti #1\Z
                {\XINT@dsx@addzeros {0#1}}%
\def\XINT@dsx@exitii #1\Z
                {\XINT@dsx@addzeros {00#1}}%
\def\XINT@dsx@exitiii #1\Z
                {\XINT@dsx@addzeros {000#1}}%
\def\XINT@dsx@exitiv #1\Z
                {\XINT@dsx@addzeros {0000#1}}%
\def\XINT@dsx@exitv #1\Z
                {\XINT@dsx@addzeros {00000#1}}%
\def\XINT@dsx@exitvi #1\Z
                {\XINT@dsx@addzeros {000000#1}}%
\def\XINT@dsx@exitvii #1\Z
                {\XINT@dsx@addzeros {0000000#1}}%
\def\XINT@dsx@addzeros #1#2{ #2#1}%
\def\XINT@dsx@xisPos #1\Z #2%
{%
    \XINT@dsx@checksignA #2\Z {#1}%
}%
\def\XINT@dsx@checksignA #1%
{%
    \xint@UDzerominusfork
      #1-\dummy  \XINT@dsx@AisZero
      0#1\dummy  \XINT@dsx@AisNeg
       0-\dummy  {\XINT@dsx@AisPos #1}%
    \xint@UDkrof
}%
\def\XINT@dsx@AisZero #1\Z #2{ {0}{0}}%
\def\XINT@dsx@AisNeg #1\Z #2%
{%
    \expandafter
      \XINT@dsx@AisNeg@dosplit@andcheckfirst
    \romannumeral0\XINT@split@checksizex {#2}{#1}%
}%
```

```
\def\XINT@dsx@AisNeg@dosplit@andcheckfirst #1%
{%
    \XINT@dsx@AisNeg@checkiffirstempty #1\Z
}%
\def\XINT@dsx@AisNeg@checkiffirstempty #1%
{%
    \xint@z #1\XINT@dsx@AisNeg@finish@zero\Z
    \XINT@dsx@AisNeg@finish@notzero #1%
}%
\def\XINT@dsx@AisNeg@finish@zero\Z
    \XINT@dsx@AisNeg@finish@notzero\Z #1%
{%
    \expandafter
        \XINT@dsx@end
    \expandafter {\romannumeral0\XINT@num {-#1}}{0}%
}%
\def\XINT@dsx@AisNeg@finish@notzero #1\Z #2%
{%
    \expandafter
        \XINT@dsx@end
    \expandafter {\romannumeral0\XINT@num {#2}}{-#1}%
}%
\def\XINT@dsx@AisPos #1\Z #2%
{%
    \expandafter
      \XINT@dsx@AisPos@finish
    \romannumeral0\XINT@split@checksizex {#2}{#1}%
}%
\def\XINT@dsx@AisPos@finish #1#2%
{%
    \expandafter
        \XINT@dsx@end
    \expandafter {\romannumeral0\XINT@num {#2}}%
                 {\romannumeral0\XINT@num {#1}}%
}%
\def\XINT@dsx@end #1#2%
{%
    \expandafter\space\expandafter{#2}{#1}%
}%
```

## 12.33 \xintDecSplit, \xintDecSplitL, \xintDecSplitR

```
    DECIMAL SPLIT
v1.01: **New** behavior, for use in future extensions of the xint bundle:
The macro \xintDecSplit {x}{A} first replaces A with |A| (*)
This macro cuts the number into two pieces L and R. The concatenation LR
always reproduces |A|, and R may be empty or have leading zeros. The
position of the cut is specified by the first argument x. If x is zero or
positive the cut location is x slots to the left of the right end of the
number. If x becomes equal to or larger than the length of the number then L
becomes empty. If x is negative the location of the cut is x slots to the
right of the left end of the number.
```

```
   (*) warning: this may change in a future version. Only the behavior
   for A non-negative is guaranteed to remain the same.
3057 \def\xintDecSplitL {\romannumeral0\xintdecsplitl }%
3058 \def\xintDecSplitR {\romannumeral0\xintdecsplitr }%
3059 \def\xintdecsplitl
3060 {%
3061     \expandafter
3062        \xint@firstoftwo@andstop
3063     \romannumeral0\xintdecsplit
3064 }%
3065 \def\xintdecsplitr
3066 {%
3067     \expandafter
3068        \xint@secondoftwo@andstop
3069     \romannumeral0\xintdecsplit
3070 }%
3071 \def\xintDecSplit {\romannumeral0\xintdecsplit }%
3072 \def\xintdecsplit #1#2%
3073 {%
3074     \expandafter
3075         \xint@split
3076     \expandafter
3077     {\romannumeral0\xintabs {#2}}{#1}%  fait expansion de A
3078 }%
3079 \def\xint@split #1#2%
3080 {%
3081     \expandafter\expandafter\expandafter
3082         \XINT@split@checksizex
3083     \expandafter\expandafter\expandafter
3084     {#2}{#1}%
3085 }%
3086 \def\XINT@split@checksizex #1%
3087 {%
3088     \ifnum\XINT@Len {#1} > 9
3089        \xint@afterfi {\xintError:TooBigDecimalSplit
3090                       \XINT@split@bigx }%
3091     \else
3092        \expandafter\XINT@split@xfork
3093     \fi
3094     #1\Z
3095 }%
3096 \def\XINT@split@bigx  #1\Z #2%
3097 {%
3098     \ifcase\XINT@Sgn {#1}
3099     \or \xint@afterfi { {}{#2}}% positive big x
3100     \else
3101         \xint@afterfi { {#2}{}}% negative big x
3102     \fi
3103 }%
3104 \def\XINT@split@xfork #1%
3105 {%
3106     \xint@UDzerominusfork
```

```
      #1-\dummy  \XINT@split@zerosplit
      0#1\dummy  \XINT@split@fromleft
       0-\dummy  {\XINT@split@fromright #1}%
    \xint@UDkrof
}%
\def\XINT@split@zerosplit #1\Z #2{ {#2}{}}%
\def\XINT@split@fromleft #1\Z #2%
{%
    \XINT@split@fromleft@loop {#1}{}#2\W\W\W\W\W\W\W\W\Z
}%
\def\XINT@split@fromleft@loop #1%
{%
    \ifcase #1
        \expandafter\XINT@split@fromleft@endsplit
    \or
        \expandafter\XINT@split@fromleft@one@andend
    \or
        \expandafter\XINT@split@fromleft@two@andend
    \or
        \expandafter\XINT@split@fromleft@three@andend
    \or
        \expandafter\XINT@split@fromleft@four@andend
    \or
        \expandafter\XINT@split@fromleft@five@andend
    \or
        \expandafter\XINT@split@fromleft@six@andend
    \or
        \expandafter\XINT@split@fromleft@seven@andend
    \else
        \expandafter \XINT@split@fromleft@loop@perhaps
        \expandafter
            {\the\numexpr
            #1-8\expandafter\expandafter\expandafter }%
        \expandafter
        \XINT@split@fromleft@eight
    \fi
}%
\def\XINT@split@fromleft@endsplit #1#2\W #3\Z
    { {#1}{#2}}%
\def\XINT@split@fromleft@eight #1#2#3#4#5#6#7#8#9%
{%
    #9{#1#2#3#4#5#6#7#8#9}%
}%
\def\XINT@split@fromleft@loop@perhaps #1#2%
{%
    \xint@w #2\XINT@split@fromleft@toofar\W \XINT@split@fromleft@loop
    {#1}%
}%
\def\XINT@split@fromleft@toofar\W \XINT@split@fromleft@loop #1#2#3\Z
{%
    \XINT@split@fromleft@toofar@b #2\Z
}%
```

```
\def\XINT@split@fromleft@toofar@b #1\W #2\Z
{%
    \space {#1}{}%
}%
\def\XINT@split@fromleft@one@andend
   {\expandafter\XINT@split@fromleft@checkiftoofar\XINT@split@fromleft@one }%
\def\XINT@split@fromleft@one #1#2{#2{#1#2}}%
\def\XINT@split@fromleft@two@andend
   {\expandafter\XINT@split@fromleft@checkiftoofar\XINT@split@fromleft@two }%
\def\XINT@split@fromleft@two #1#2#3{#3{#1#2#3}}%
\def\XINT@split@fromleft@three@andend
   {\expandafter\XINT@split@fromleft@checkiftoofar\XINT@split@fromleft@three }%
\def\XINT@split@fromleft@three #1#2#3#4{#4{#1#2#3#4}}%
\def\XINT@split@fromleft@four@andend
   {\expandafter\XINT@split@fromleft@checkiftoofar\XINT@split@fromleft@four }%
\def\XINT@split@fromleft@four #1#2#3#4#5{#5{#1#2#3#4#5}}%
\def\XINT@split@fromleft@five@andend
   {\expandafter\XINT@split@fromleft@checkiftoofar\XINT@split@fromleft@five }%
\def\XINT@split@fromleft@five #1#2#3#4#5#6{#6{#1#2#3#4#5#6}}%
\def\XINT@split@fromleft@six@andend
   {\expandafter\XINT@split@fromleft@checkiftoofar\XINT@split@fromleft@six }%
\def\XINT@split@fromleft@six #1#2#3#4#5#6#7{#7{#1#2#3#4#5#6#7}}%
\def\XINT@split@fromleft@seven@andend
   {\expandafter\XINT@split@fromleft@checkiftoofar\XINT@split@fromleft@seven }%
\def\XINT@split@fromleft@seven #1#2#3#4#5#6#7#8{#8{#1#2#3#4#5#6#7#8}}%
\def\XINT@split@fromleft@checkiftoofar #1#2#3\W #4\Z
{%
    \xint@w #1\XINT@split@fromleft@wenttoofar\W
    \space {#2}{#3}%
}%
\def\XINT@split@fromleft@wenttoofar\W\space #1%
{%
    \XINT@split@fromleft@wenttoofar@b #1\Z
}%
\def\XINT@split@fromleft@wenttoofar@b #1\W #2\Z
{%
    \space {#1}%
}%
\def\XINT@split@fromright #1\Z #2%
{%
    \expandafter
        \XINT@split@fromright@a
    \expandafter
    {\romannumeral0\XINT@rev {#2}}{#1}{#2}%
}%
\def\XINT@split@fromright@a #1#2%
{%
    \XINT@split@fromright@loop {#2}{}#1\W\W\W\W\W\W\W\W\Z
}%
\def\XINT@split@fromright@loop #1%
{%
    \ifcase #1
```

```
        \expandafter\XINT@split@fromright@endsplit
    \or
        \expandafter\XINT@split@fromright@one@andend
    \or
        \expandafter\XINT@split@fromright@two@andend
    \or
        \expandafter\XINT@split@fromright@three@andend
    \or
        \expandafter\XINT@split@fromright@four@andend
    \or
        \expandafter\XINT@split@fromright@five@andend
    \or
        \expandafter\XINT@split@fromright@six@andend
    \or
        \expandafter\XINT@split@fromright@seven@andend
    \else
        \expandafter \XINT@split@fromright@loop@perhaps
        \expandafter
            {\the\numexpr
            #1-8\expandafter\expandafter\expandafter }%
        \expandafter
        \XINT@split@fromright@eight
    \fi
}%
\def\XINT@split@fromright@endsplit #1#2\W #3\Z #4%
{%
    \expandafter
    \space
    \expandafter {\romannumeral0\XINT@rev{#2}}{#1}%
}%
\def\XINT@split@fromright@eight #1#2#3#4#5#6#7#8#9%
{%
    #9{#9#8#7#6#5#4#3#2#1}%
}%
\def\XINT@split@fromright@loop@perhaps #1#2%
{%
    \xint@w #2\XINT@split@fromright@toofar\W\XINT@split@fromright@loop
    {#1}%
}%
\def\XINT@split@fromright@toofar\W\XINT@split@fromright@loop #1#2#3\Z
    { {}}%
\def\XINT@split@fromright@one@andend
 {\expandafter\XINT@split@fromright@checkiftoofar\XINT@split@fromright@one }%
\def\XINT@split@fromright@one #1#2{#2{#2#1}}%
\def\XINT@split@fromright@two@andend
 {\expandafter\XINT@split@fromright@checkiftoofar\XINT@split@fromright@two }%
\def\XINT@split@fromright@two #1#2#3{#3{#3#2#1}}%
\def\XINT@split@fromright@three@andend
 {\expandafter\XINT@split@fromright@checkiftoofar\XINT@split@fromright@three }%
\def\XINT@split@fromright@three #1#2#3#4{#4{#4#3#2#1}}%
\def\XINT@split@fromright@four@andend
 {\expandafter\XINT@split@fromright@checkiftoofar\XINT@split@fromright@four }%
```

```
\def\XINT@split@fromright@four #1#2#3#4#5{#5{#5#4#3#2#1}}%
\def\XINT@split@fromright@five@andend
 {\expandafter\XINT@split@fromright@checkiftoofar\XINT@split@fromright@five }%
\def\XINT@split@fromright@five #1#2#3#4#5#6{#6{#6#5#4#3#2#1}}%
\def\XINT@split@fromright@six@andend
 {\expandafter\XINT@split@fromright@checkiftoofar\XINT@split@fromright@six }%
\def\XINT@split@fromright@six #1#2#3#4#5#6#7{#7{#7#6#5#4#3#2#1}}%
\def\XINT@split@fromright@seven@andend
 {\expandafter\XINT@split@fromright@checkiftoofar\XINT@split@fromright@seven }%
\def\XINT@split@fromright@seven #1#2#3#4#5#6#7#8{#8{#8#7#6#5#4#3#2#1}}%
\def\XINT@split@fromright@checkiftoofar #1%
{%
    \xint@w #1\XINT@split@fromright@wenttoofar\W
    \XINT@split@fromright@endsplit
}%
\def\XINT@split@fromright@wenttoofar\W\XINT@split@fromright@endsplit #1\Z #2%
    { {}{#2}}%
\XINT@restorecatcodes@endinput%
```

# 13 Package xintgcd implementation

The commenting is currently (2013/04/05) very sparse.

## 13.1 Catcodes, ε-TEX detection, reload detection

The code for reload detection is copied from Heiko Oberdiek's packages, and adapted here to check for previous loading of the master **xint** package.

The method for catcodes is slightly different, but still directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode35=6    % #
  \catcode44=12   % ,
  \catcode45=12   % -
  \catcode46=12   % .
  \catcode58=12   % :
  \def\space { }%
  \let\z\endgroup
  \expandafter\let\expandafter\x\csname ver@xintgcd.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xint.sty\endcsname
  \expandafter
    \ifx\csname PackageInfo\endcsname\relax
      \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
    \else
      \def\y#1#2{\PackageInfo{#1}{#2}}%
    \fi
  \expandafter
  \ifx\csname numexpr\endcsname\relax
```

```
    \y{xintgcd}{\numexpr not available, aborting input}%
    \aftergroup\endinput
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintgcd.sty
      \ifx\w\relax % but xint.sty not yet loaded.
         \y{xintgcd}{Package xint is required}%
         \y{xintgcd}{Will try \string\input\space xint.sty}%
         \def\z{\endgroup\input xint.sty\relax}%
      \fi
    \else
      \def\empty {}%
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xint.sty not yet loaded.
            \y{xintgcd}{Package xint is required}%
            \y{xintgcd}{Will try \string\RequirePackage{xint}}%
            \def\z{\endgroup\RequirePackage{xint}}%
          \fi
      \else
        \y{xintgcd}{I was already loaded, aborting input}%
        \aftergroup\endinput
      \fi
    \fi
  \fi
\z%
```

## 13.2 Validation of xint loading

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode35=6    % #
  \catcode44=12   % ,
  \catcode45=12   % -
  \catcode46=12   % .
  \catcode58=12   % :
  \expandafter
    \ifx\csname PackageInfo\endcsname\relax
      \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
    \else
      \def\y#1#2{\PackageInfo{#1}{#2}}%
    \fi
  \def\empty {}%
  \expandafter\let\expandafter\w\csname ver@xint.sty\endcsname
  \ifx\w\relax % Plain TeX, user gave a file name at the prompt
     \y{xintgcd}{Loading of package xint failed, aborting input}%
     \aftergroup\endinput
  \fi
  \ifx\w\empty % LaTeX, user gave a file name at the prompt
```

```
      \y{xintgcd}{Loading of package xint failed, aborting input}%
      \aftergroup\endinput
  \fi
\endgroup%
```

## 13.3 Catcodes

Perhaps catcodes have changed after the loading of **xint** and prior to the current loading of **xintgcd**, so we can not employ the `\XINT@restorecatcodes@endinput` in this style file. But there is no problem using `\XINT@setcatcodes`.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \def\x
  {%
      \endgroup
      \edef\XINT@gcd@restorecatcodes@endinput
      {%
        \catcode36=\the\catcode36   % $
        \catcode47=\the\catcode47   % /
        \catcode41=\the\catcode41   % )
        \catcode40=\the\catcode40   % (
        \catcode42=\the\catcode42   % *
        \catcode43=\the\catcode43   % +
        \catcode62=\the\catcode62   % >
        \catcode60=\the\catcode60   % <
        \catcode58=\the\catcode58   % :
        \catcode46=\the\catcode46   % .
        \catcode45=\the\catcode45   % -
        \catcode44=\the\catcode44   % ,
        \catcode35=\the\catcode35   % #
        \catcode64=\the\catcode64   % @
        \catcode125=\the\catcode125 % }
        \catcode123=\the\catcode123 % {
        \endlinechar=\the\endlinechar
        \catcode13=\the\catcode13   % ^^M
        \catcode32=\the\catcode32   %
        \catcode61=\the\catcode61   % =
        \noexpand\endinput
      }%
      \XINT@setcatcodes
      \catcode36=3  % $
  }%
\x
```

## 13.4 Package identification

```
\begingroup
  \catcode91=12 % [
```

```
  \catcode93=12 % ]
  \catcode58=12 % :
  \expandafter\ifx\csname ProvidesPackage\endcsname\relax
    \def\x#1#2#3[#4]{\endgroup
      \immediate\write-1{Package: #3 #4}%
      \xdef#1{#4}%
    }%
  \else
    \def\x#1#2[#3]{\endgroup
      #2[{#3}]%
      \ifx#1\@undefined
        \xdef#1{#3}%
      \fi
      \ifx#1\relax
        \xdef#1{#3}%
      \fi
    }%
  \fi
\expandafter\x\csname ver@xintgcd.sty\endcsname
\ProvidesPackage{xintgcd}%
  [2013/04/05 v1.02 Euclide algorithm with xint package (jfB)]%
```

## 13.5 \xintGCD

```
\def\xintGCD {\romannumeral0\xintgcd }%
\def\xintgcd #1%
{%
    \expandafter
      \XINT@gcd
    \expandafter
      {\romannumeral0\xintabs {#1}}%
}%
\def\XINT@gcd #1#2%
{%
    \expandafter
      \XINT@gcd@fork
    \romannumeral0\xintabs {#2}\Z #1\Z
}%
  Ici #3#4=A, #1#2=B
\def\XINT@gcd@fork #1#2\Z #3#4\Z
{%
    \xint@UDzerofork
      #1\dummy \XINT@gcd@BisZero
      #3\dummy \XINT@gcd@AisZero
       0\dummy \XINT@gcd@loop
    \xint@UDkrof
    {#1#2}{#3#4}%
}%
\def\XINT@gcd@AisZero #1#2{ #1}%
\def\XINT@gcd@BisZero #1#2{ #2}%
\def\XINT@gcd@CheckRem #1#2\Z
{%
    \xint@zero #1\xint@gcd@end0\XINT@gcd@loop {#1#2}%
```

```
}%
\def\xint@gcd@end0\XINT@gcd@loop #1#2{ #2}%
```

#1=B, #2=A

```
\def\XINT@gcd@loop #1#2%
{%
    \expandafter\expandafter\expandafter
        \XINT@gcd@CheckRem
    \expandafter\xint@secondoftwo
    \romannumeral0\XINT@div@prepare {#1}{#2}\Z
    {#1}%
}%
```

## 13.6 \xintBezout

```
\def\xintBezout {\romannumeral0\xintbezout }%
\def\xintbezout #1%
{%
    \expandafter\expandafter\expandafter
      \xint@bezout
    \expandafter\expandafter\expandafter
      {#1}%
}%
\def\xint@bezout #1#2%
{\expandafter\expandafter\expandafter
    \XINT@bezout@fork #2\Z #1\Z
}%
```

#3#4 = A, #1#2=B

```
\def\XINT@bezout@fork #1#2\Z #3#4\Z
{%
    \xint@UDzerosfork
     #1#3\dummy \XINT@bezout@botharezero
      #10\dummy \XINT@bezout@secondiszero
      #30\dummy \XINT@bezout@firstiszero
       00\dummy
        {\xint@UDsignsfork
           #1#3\dummy \XINT@bezout@minusminus % A < 0, B < 0
            #1-\dummy \XINT@bezout@minusplus  % A > 0, B < 0
            #3-\dummy \XINT@bezout@plusminus  % A < 0, B > 0
             --\dummy \XINT@bezout@plusplus   % A > 0, B > 0
         \xint@UDkrof }%
    \xint@UDkrof
    {#2}{#4}#1#3{#3#4}{#1#2}% #1#2=B, #3#4=A
}%
\def\XINT@bezout@botharezero #1#2#3#4#5#6%
{%
    \xintError:NoBezoutForZeros
    \space {0}{0}{0}{0}{0}%
}%
```

attention première entrée doit être ici (-1)^n donc 1
#4#2=0 = A, B = #3#1

```
\def\XINT@bezout@firstiszero #1#2#3#4#5#6%
{%
    \xint@UDsignfork
      #3\dummy { {0}{#3#1}{0}{1}{#1}}%
       -\dummy { {0}{#3#1}{0}{-1}{#1}}%
    \xint@UDkrof
}%
```

`#4#2= A, B = #3#1 = 0`

```
\def\XINT@bezout@secondiszero #1#2#3#4#5#6%
{%
    \xint@UDsignfork
       #4\dummy{ {#4#2}{0}{-1}{0}{#2}}%
        -\dummy{ {#4#2}{0}{1}{0}{#2}}%
    \xint@UDkrof
}%
```

`#4#2= A < 0, #3#1 = B < 0`

```
\def\XINT@bezout@minusminus #1#2#3#4%
{%
    \expandafter\XINT@bezout@mm@post
    \romannumeral0\XINT@bezout@loop@a 1{#1}{#2}1001%
}%
\def\XINT@bezout@mm@post #1#2%
{%
    \expandafter
        \XINT@bezout@mm@postb
    \expandafter
        {\romannumeral0\xintopp{#2}}{\romannumeral0\xintopp{#1}}%
}%
\def\XINT@bezout@mm@postb #1#2%
{%
    \expandafter
        \XINT@bezout@mm@postc
    \expandafter {#2}{#1}%
}%
\def\XINT@bezout@mm@postc #1#2#3#4#5%
{%
    \space {#4}{#5}{#1}{#2}{#3}%
}%
```

`minusplus  #4#2= A > 0, B < 0`

```
\def\XINT@bezout@minusplus #1#2#3#4%
{%
    \expandafter\XINT@bezout@mp@post
    \romannumeral0\XINT@bezout@loop@a 1{#1}{#4#2}1001%
}%
\def\XINT@bezout@mp@post #1#2%
{%
    \expandafter
      \XINT@bezout@mp@postb
```

```
3533     \expandafter
3534       {\romannumeral0\xintopp {#2}}{#1}%
3535 }%
3536 \def\XINT@bezout@mp@postb #1#2#3#4#5%
3537 {%
3538     \space {#4}{#5}{#2}{#1}{#3}%
3539 }%
```

plusminus  A < 0, B > 0

```
3540 \def\XINT@bezout@plusminus #1#2#3#4%
3541 {%
3542     \expandafter\XINT@bezout@pm@post
3543     \romannumeral0\XINT@bezout@loop@a 1{#3#1}{#2}1001%
3544 }%
3545 \def\XINT@bezout@pm@post #1%
3546 {%
3547     \expandafter
3548         \XINT@bezout@pm@postb
3549     \expandafter
3550         {\romannumeral0\xintopp{#1}}%
3551 }%
3552 \def\XINT@bezout@pm@postb #1#2#3#4#5%
3553 {%
3554     \space {#4}{#5}{#1}{#2}{#3}%
3555 }%
```

plusplus

```
3556 \def\XINT@bezout@plusplus #1#2#3#4%
3557 {%
3558     \expandafter\XINT@bezout@pp@post
3559     \romannumeral0\XINT@bezout@loop@a 1{#3#1}{#4#2}1001%
3560 }%
```

la parité (-1)^N est en #1, et on la jette ici.

```
3561 \def\XINT@bezout@pp@post #1#2#3#4#5%
3562 {%
3563     \space {#4}{#5}{#1}{#2}{#3}%
3564 }%
```

n = 0: 1BAalpha(0)beta(0)alpha(-1)beta(-1)
n général:
{(-1)^n}{r(n-1)}{r(n-2)}{alpha(n-1)}{beta(n-1)}{alpha(n-2)}{beta(n-2)}
#2 = B, #3 = A

```
3565 \def\XINT@bezout@loop@a #1#2#3%
3566 {%
3567     \expandafter\XINT@bezout@loop@b
3568     \expandafter{\the\numexpr -#1\expandafter }%
3569     \romannumeral0\XINT@div@prepare {#2}{#3}{#2}%
3570 }%
```

```
  Le q(n) a ici une existence éphémère, dans le version Bezout Algorithm
  il faudra le conserver. On voudra à la fin
  {{q(n)}{r(n)}{alpha(n)}{beta(n)}}
  De plus ce n'est plus (-1)^n que l'on veut mais n. (ou dans un autre ordre)
  {-(-1)^n}{q(n)}{r(n)}{r(n-1)}{alpha(n-1)}{beta(n-1)}{alpha(n-2)}{beta(n-2)}
```

```
\def\XINT@bezout@loop@b #1#2#3#4#5#6#7#8%
{%
    \expandafter
        \XINT@bezout@loop@c
    \expandafter
        {\romannumeral0\xintadd{\XINT@Mul{#5}{#2}}{#7}}%
        {\romannumeral0\xintadd{\XINT@Mul{#6}{#2}}{#8}}%
    {#1}{#3}{#4}{#5}{#6}%
}%
```

```
  {alpha(n)}{->beta(n)}{-(-1)^n}{r(n)}{r(n-1)}{alpha(n-1)}{beta(n-1)}
```

```
\def\XINT@bezout@loop@c #1#2%
{%
    \expandafter
        \XINT@bezout@loop@d
    \expandafter
        {#2}{#1}%
}%
```

```
  {beta(n)}{alpha(n)}{(-1)^(n+1)}{r(n)}{r(n-1)}{alpha(n-1)}{beta(n-1)}
```

```
\def\XINT@bezout@loop@d #1#2#3#4#5%
{%
    \XINT@bezout@loop@e #4\Z {#3}{#5}{#2}{#1}%
}%
```

```
  r(n)\Z {(-1)^(n+1)}{r(n-1)}{alpha(n)}{beta(n)}{alpha(n-1)}{beta(n-1)}
```

```
\def\XINT@bezout@loop@e #1#2\Z
{%
    \xint@zero #1\xint@bezout@loop@exit0\XINT@bezout@loop@f
    {#1#2}%
}%
```

```
  {r(n)}{(-1)^(n+1)}{r(n-1)}{alpha(n)}{beta(n)}{alpha(n-1)}{beta(n-1)}
```

```
\def\XINT@bezout@loop@f #1#2%
{%
    \XINT@bezout@loop@a {#2}{#1}%
}%
```

```
  {(-1)^(n+1)}{r(n)}{r(n-1)}{alpha(n)}{beta(n)}{alpha(n-1)}{beta(n-1)}
  et itération
```

```
\def\xint@bezout@loop@exit0\XINT@bezout@loop@f #1#2%
{%
    \ifcase #2
    \or  \expandafter\XINT@bezout@exiteven
    \else\expandafter\XINT@bezout@exitodd
    \fi
}%
\def\XINT@bezout@exiteven #1#2#3#4#5%
{%
    \space {#5}{#4}{#1}%
}%
\def\XINT@bezout@exitodd #1#2#3#4#5%
{%
    \space {-#5}{-#4}{#1}%
}%
```

## 13.7 \xintEuclideAlgorithm

```
  Pour Euclide:
  {N}{A}{D=r(n)}{B}{q1}{r1}{q2}{r2}{q3}{r3}....{qN}{rN=0}
  u<2n> = u<2n+3>u<2n+2> + u<2n+4> à la n ième étape
```

```
\def\xintEuclideAlgorithm {\romannumeral0\xinteuclidealgorithm }%
\def\xinteuclidealgorithm #1%
{%
    \expandafter
      \XINT@euc
    \expandafter
      {\romannumeral0\xintabs {#1}}%
}%
\def\XINT@euc #1#2%
{%
    \expandafter
      \XINT@euc@fork
    \romannumeral0\xintabs {#2}\Z #1\Z
}%
```

```
  Ici #3#4=A, #1#2=B
```

```
\def\XINT@euc@fork #1#2\Z #3#4\Z
{%
    \xint@UDzerofork
      #1\dummy \XINT@euc@BisZero
      #3\dummy \XINT@euc@AisZero
       0\dummy \XINT@euc@a
    \xint@UDkrof
    {0}{#1#2}{#3#4}{{#3#4}{#1#2}}{}\Z
}%
```

```
  Le {} pour protéger {{A}{B}} si on s'arrête après une étape (B divise A)
  On va renvoyer:
  {N}{A}{D=r(n)}{B}{q1}{r1}{q2}{r2}{q3}{r3}....{qN}{rN=0}
```

```
\def\XINT@euc@AisZero #1#2#3#4#5#6{ {1}{0}{#2}{#2}{0}{0}}%
\def\XINT@euc@BisZero #1#2#3#4#5#6{ {1}{0}{#3}{#3}{0}{0}}%

  {n}{rn}{an}{{qn}{rn}}...{{A}{B}}{}\Z
   an = r(n-1)
  Pour n=0 on a juste {0}{B}{A}{{A}{B}}{}\Z
  \XINT@div@prepare {u}{v} divise v par u

\def\XINT@euc@a #1#2#3%
{%
    \expandafter
        \XINT@euc@b
    \expandafter {\the\numexpr #1+1\expandafter }%
    \romannumeral0\XINT@div@prepare {#2}{#3}{#2}%
}%

  {n+1}{q(n+1)}{r(n+1)}{rn}{{qn}{rn}}...

\def\XINT@euc@b #1#2#3#4%
{%
    \XINT@euc@c #3\Z {#1}{#3}{#4}{{#2}{#3}}%
}%

  r(n+1)\Z {n+1}{r(n+1)}{r(n)}{{q(n+1)}{r(n+1)}}{{qn}{rn}}...
  Test si r(n+1) est nul.

\def\XINT@euc@c #1#2\Z
{%
    \xint@zero #1\xint@euc@end0\XINT@euc@a
}%

  {n+1}{r(n+1)}{r(n)}{{q(n+1)}{r(n+1)}}...{}\Z
  Ici r(n+1) = 0. On arrête on se prépare à inverser.
  {n+1}{0}{r(n)}{{q(n+1)}{r(n+1)}}.....{{q1}{r1}}{{A}{B}}{}\Z
  On veut renvoyer:
  {N=n+1}{A}{D=r(n)}{B}{q1}{r1}{q2}{r2}{q3}{r3}....{qN}{rN=0}

\def\xint@euc@end0\XINT@euc@a #1#2#3#4\Z%
{%
      \expandafter\xint@euc@end@
      \romannumeral0%
      \XINT@rord@main {}#4{{#1}{#3}}%
      \xint@UNDEF
        \xint@undef\xint@undef\xint@undef\xint@undef
        \xint@undef\xint@undef\xint@undef\xint@undef
      \xint@UNDEF
}%
\def\xint@euc@end@ #1#2#3%
{%
    \space {#1}{#3}{#2}%
}%
```

## 13.8 \xintBezoutAlgorithm

```
  Pour Bezout: objectif, renvoyer
  alpha0=1, beta0=0
  alpha(-1)=0, beta(-1)=1
  {N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
        {q2}{r2}{alpha2}{beta2}....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}
```

```
\def\xintBezoutAlgorithm {\romannumeral0\xintbezoutalgorithm }%
\def\xintbezoutalgorithm #1%
{%
    \expandafter
      \XINT@bezalg
    \expandafter
      {\romannumeral0\xintabs {#1}}%
}%
\def\XINT@bezalg #1#2%
{%
    \expandafter
      \XINT@bezalg@fork
    \romannumeral0\xintabs {#2}\Z #1\Z
}%
```

```
  Ici #3#4=A, #1#2=B
```

```
\def\XINT@bezalg@fork #1#2\Z #3#4\Z
{%
    \xint@UDzerofork
      #1\dummy \XINT@bezalg@BisZero
      #3\dummy \XINT@bezalg@AisZero
       0\dummy \XINT@bezalg@a
    \xint@UDkrof
    0{#1#2}{#3#4}1001{{#3#4}{#1#2}}{}\Z
}%
\def\XINT@bezalg@AisZero #1#2#3\Z{ {1}{0}{0}{1}{#2}{#2}{1}{0}{0}{0}{0}{1}}%
\def\XINT@bezalg@BisZero #1#2#3#4\Z{ {1}{0}{0}{1}{#3}{#3}{1}{0}{0}{0}{0}{1}}%
```

```
  pour préparer l'étape n+1 il faut
  {n}{r(n)}{r(n-1)}{alpha(n)}{beta(n)}{alpha(n-1)}{beta(n-1)}
                    {{q(n)}{r(n)}{alpha(n)}{beta(n)}}...
  division de #3 par #2
```

```
\def\XINT@bezalg@a #1#2#3%
{%
    \expandafter
        \XINT@bezalg@b
    \expandafter {\the\numexpr #1+1\expandafter }%
    \romannumeral0\XINT@div@prepare {#2}{#3}{#2}%
}%
```

```
  {n+1}{q(n+1)}{r(n+1)}{r(n)}{alpha(n)}{beta(n)}{alpha(n-1)}{beta(n-1)}...
```

```
\def\XINT@bezalg@b #1#2#3#4#5#6#7#8%
{%
    \expandafter\XINT@bezalg@c\expandafter
     {\romannumeral0\xintadd {\xintMul {#6}{#2}}{#8}}%
     {\romannumeral0\xintadd {\xintMul {#5}{#2}}{#7}}%
     {#1}{#2}{#3}{#4}{#5}{#6}%
}%
  {beta(n+1)}{alpha(n+1)}{n+1}{q(n+1)}{r(n+1)}{r(n)}{alpha(n)}{beta(n}}%
\def\XINT@bezalg@c #1#2#3#4#5#6%
{%
    \expandafter\XINT@bezalg@d\expandafter
     {#2}{#3}{#4}{#5}{#6}{#1}%
}%
  {alpha(n+1)}{n+1}{q(n+1)}{r(n+1)}{r(n)}{beta(n+1)}
\def\XINT@bezalg@d #1#2#3#4#5#6#7#8%
{%
    \XINT@bezalg@e #4\Z {#2}{#4}{#5}{#1}{#6}{#7}{#8}{{#3}{#4}{#1}{#6}}%
}%
  r(n+1)\Z {n+1}{r(n+1)}{r(n)}{alpha(n+1)}{beta(n+1)}
                              {alpha(n)}{beta(n)}{q,r,alpha,beta(n+1)}
  Test si r(n+1) est nul.
\def\XINT@bezalg@e #1#2\Z
{%
    \xint@zero #1\xint@bezalg@end0\XINT@bezalg@a
}%
  Ici r(n+1) = 0. On arrête on se prépare à inverser.
  {n+1}{r(n+1)}{r(n)}{alpha(n+1)}{beta(n+1)}%
                    {alpha(n)}{beta(n)}%
                     {q,r,alpha,beta(n+1)}...{{A}{B}}{}\Z
  On veut renvoyer
  {N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
       {q2}{r2}{alpha2}{beta2}....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}
\def\xint@bezalg@end0\XINT@bezalg@a #1#2#3#4#5#6#7#8\Z
{%
      \expandafter\xint@bezalg@end@
      \romannumeral0%
      \XINT@rord@main {}#8{{#1}{#3}}%
      \xint@UNDEF
        \xint@undef\xint@undef\xint@undef\xint@undef
        \xint@undef\xint@undef\xint@undef\xint@undef
      \xint@UNDEF
}%
  {N}{D}{A}{B}{q1}{r1}{alpha1=q1}{beta1=1}{q2}{r2}{alpha2}{beta2}
       ....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}
  On veut renvoyer
  {N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
        {q2}{r2}{alpha2}{beta2}....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}
\def\xint@bezalg@end@ #1#2#3#4%
{%
    \space {#1}{#3}{0}{1}{#2}{#4}{1}{0}%
}%
```

## 13.9 \xintTypesetEuclideAlgorithm

```
TYPESETTING
Organisation:
{N}{A}{D}{B}{q1}{r1}{q2}{r2}{q3}{r3}....{qN}{rN=0}
\U1 = N = nombre d'étapes, \U3 = PGCD, \U2 = A, \U4=B
q1 = \U5, q2 = \U7 --> qn = \U<2n+3>, rn = \U<2n+4>
bn = rn. B = r0. A=r(-1)
r(n-2) = q(n)r(n-1)+r(n) (n e étape) (n au moins 1)
\U{2n} = \U{2n+3} \times \U{2n+2} + \U{2n+4}, n e étape.
avec n entre 1 et N.
```

```
\def\xintTypesetEuclideAlgorithm #1#2%
{% l'algo remplace #1 et #2 par |#1| et |#2|
  \par
  \begingroup
    \xintAssignArray\xintEuclideAlgorithm {#1}{#2}\to\U
    \edef\A{\U2}\edef\B{\U4}\edef\N{\U1}%
    \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
    \noindent
    \count 255 1
    \loop
      \hbox to \wd 0 {\hfil$\U{\the\numexpr 2*\count 255\relax}$}%
      ${} =  \U{\the\numexpr 2*\count 255 + 3\relax}
      \times \U{\the\numexpr 2*\count 255 + 2\relax}
          +  \U{\the\numexpr 2*\count 255 + 4\relax}$%
    \ifnum \count 255 < \N
      \hfill\break
      \advance \count 255 1
    \repeat
  \par
  \endgroup
}%
```

## 13.10 \xintTypesetBezoutAlgorithm

```
Pour Bezout on a:
{N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
            {q2}{r2}{alpha2}{beta2}....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}%
Donc 4N+8 termes
U1 = N, U2= A, U5=D, U6=B,
q1 = U9, qn = U{4n+5}, n au moins 1
rn = U{4n+6}    , n au moins -1
alpha(n) = U{4n+7}, n au moins -1
beta(n)  = U{4n+8}, n au moins -1
```

```
\def\xintTypesetBezoutAlgorithm #1#2%
{%
  \par
  \begingroup
    \parindent0pt
    \xintAssignArray\xintBezoutAlgorithm {#1}{#2}\to\BEZ
    \edef\A{\BEZ2}\edef\B{\BEZ6}\edef\N{\BEZ1}% A = |#1|, B = |#2|
```

```
    \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
    \count 255 1
    \loop
      \noindent
      \hbox to \wd 0 {\hfil$\BEZ{\the\numexpr 4*\count 255 - 2\relax}$}%
      ${} =  \BEZ{\the\numexpr 4*\count 255 + 5\relax}
      \times \BEZ{\the\numexpr 4*\count 255 + 2\relax}
          +  \BEZ{\the\numexpr 4*\count 255 + 6\relax}$\hfill\break
      \hbox to \wd 0 {\hfil$\BEZ{\the\numexpr 4*\count 255 +7\relax}$}%
      ${} = \BEZ{\the\numexpr 4*\count 255 + 5\relax}
      \times \BEZ{\the\numexpr 4*\count 255 + 3\relax}
          +  \BEZ{\the\numexpr 4*\count 255 - 1\relax}$\hfill\break
      \hbox to \wd 0 {\hfil$\BEZ{\the\numexpr 4*\count 255 +8\relax}$}%
      ${} =  \BEZ{\the\numexpr 4*\count 255 + 5\relax}
      \times \BEZ{\the\numexpr 4*\count 255 + 4\relax}
          +  \BEZ{\the\numexpr 4*\count 255 \relax}$
      \endgraf
    \ifnum \count 255 < \N
    \advance \count 255 1
  \repeat
  \par
    \edef\U{\BEZ{\the\numexpr 4*\N + 4\relax}}%
    \edef\V{\BEZ{\the\numexpr 4*\N + 3\relax}}%
    \edef\D{\BEZ5}%
    \ifodd\N\relax
       $\U\times\A  - \V\times \B = -\D$%
    \else
       $\U\times\A - \V\times\B = \D$%
    \fi
  \par
  \endgroup
}%
\XINT@gcd@restorecatcodes@endinput%
```