

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	3
2	Expansions	4
3	Inputs	6
4	Outputs	7
5	Assignments	7
6	Error messages	9
7	Package namespace	9
8	Loading and usage	10
9	Installation	10
10	Commands of the xint package	11
10.1	<code>\xintRev</code>	11
10.2	<code>\xintReverseOrder</code>	11
10.3	<code>\xintNum</code>	11
10.4	<code>\xintLen</code>	11
10.5	<code>\xintLength</code>	11
10.6	<code>\xintAssign</code>	11
10.7	<code>\xintAssignArray</code>	12
10.8	<code>\xintRelaxArray</code>	12
10.9	<code>\xintDigitsOf</code>	12
10.10	<code>\xintSgn</code>	12
10.11	<code>\xintOpp</code>	12
10.12	<code>\xintAbs</code>	12

Contents

10.13	<code>\xintAdd</code>	13
10.14	<code>\xintSub</code>	13
10.15	<code>\xintCmp</code>	13
10.16	<code>\xintGeq</code>	13
10.17	<code>\xintMax</code>	13
10.18	<code>\xintMin</code>	13
10.19	<code>\xintSum</code>	13
10.20	<code>\xintSumExpr</code>	13
10.21	<code>\xintMul</code>	14
10.22	<code>\xintSqr</code>	14
10.23	<code>\xintPrd</code>	14
10.24	<code>\xintProductExpr</code>	15
10.25	<code>\xintFac</code>	15
10.26	<code>\xintPow</code>	15
10.27	<code>\xintDivision</code>	15
10.28	<code>\xintQuo</code>	15
10.29	<code>\xintRem</code>	15
10.30	<code>\xintFDg</code>	15
10.31	<code>\xintLDg</code>	15
10.32	<code>\xintOdd</code>	15
10.33	<code>\xintDSL</code>	16
10.34	<code>\xintDSR</code>	16
10.35	<code>\xintDSH</code>	16
10.36	<code>\xintDSHr</code> , <code>\xintDSx</code>	16
10.37	<code>\xintDecSplit</code>	17
10.38	<code>\xintDecSplitL</code>	17
10.39	<code>\xintDecSplitR</code>	17
11	Commands of the <code>xintgcd</code> package	18
11.1	<code>\xintGCD</code>	18
11.2	<code>\xintBezout</code>	18
11.3	<code>\xintEuclideanAlgorithm</code>	18
11.4	<code>\xintBezoutAlgorithm</code>	18
11.5	<code>\xintTypesetEuclideanAlgorithm</code>	18
11.6	<code>\xintTypesetBezoutAlgorithm</code>	19
12	Package <code>xint</code> implementation	19
12.1	Catcodes, ε - \TeX detection, reload detection	19
12.2	Package identification	21
12.3	Token management macros	22
12.4	<code>\xintRev</code> , <code>\xintReverseOrder</code>	22
12.5	<code>\XINT@RQ</code>	24
12.6	<code>\XINT@cuz</code>	24
12.7	<code>\xintNum</code>	25
12.8	<code>\xintLen</code> , <code>\xintLength</code>	26
12.9	<code>\xintAssign</code> , <code>\xintAssignArray</code> , <code>\xintDigitsOf</code>	27
12.10	<code>\xintSgn</code>	29

12.11	<code>\xintOpp</code>	30
12.12	<code>\xintAbs</code>	30
12.13	<code>\xintAdd</code>	31
12.14	<code>\xintSub</code>	34
12.15	<code>\xintCmp</code>	39
12.16	<code>\xintGeq</code>	42
12.17	<code>\xintMax</code>	44
12.18	<code>\xintMin</code>	45
12.19	<code>\xintSum</code> , <code>\xintSumExpr</code>	46
12.20	<code>\xintMul</code>	49
12.21	<code>\xintSqr</code>	56
12.22	<code>\xintPrd</code> , <code>\xintProductExpr</code>	56
12.23	<code>\xintFac</code>	59
12.24	<code>\xintPow</code>	61
12.25	<code>\xintDivision</code> , <code>\xintQuo</code> , <code>\xintRem</code>	69
12.26	<code>\xintFDg</code>	81
12.27	<code>\xintLDg</code>	81
12.28	<code>\xintOdd</code>	82
12.29	<code>\xintDSL</code>	82
12.30	<code>\xintDSR</code>	83
12.31	<code>\xintDSH</code> , <code>\xintDSHr</code>	83
12.32	<code>\xintDSx</code>	84
12.33	<code>\xintDecSplit</code> , <code>\xintDecSplitL</code> , <code>\xintDecSplitR</code>	87
13	Package <code>xintgcd</code> implementation	92
13.1	Catcodes, ε - \TeX detection, reload detection	92
13.2	Validation of <code>xint</code> loading	93
13.3	Catcodes	94
13.4	Package identification	94
13.5	<code>\xintGCD</code>	95
13.6	<code>\xintBezout</code>	96
13.7	<code>\xintEuclideanAlgorithm</code>	100
13.8	<code>\xintBezoutAlgorithm</code>	102
13.9	<code>\xintTypesetEuclideanAlgorithm</code>	104
13.10	<code>\xintTypesetBezoutAlgorithm</code>	104

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.

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

2 Expansions

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`.

2 Expansions

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 {+--+---+---+---000000000009876543210}=-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.

²It 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 worthwhile 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 intrinsic \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 T_EX to stop its scanning for a number: once T_EX 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:

$\backslash \text{xintAssign} \backslash \text{xintDivision} \{100\} \{3\} \backslash \text{to} \backslash \text{A} \backslash \text{B}$
 $\backslash \text{xintAssign} \backslash \text{xintDivision} \{ \backslash \text{xintPow} \{2\} \{1000\} \} \{ \backslash \text{xintFac} \{100\} \} \backslash \text{to} \backslash \text{A} \backslash \text{B}$
 gives $\backslash \text{meaning} \backslash \text{A}$: macro:->1148132496415075054822783938725510662598055177
 84186172883663478065826541894704737970419535798876630484358265060061503
 749531707793118627774829601 and $\backslash \text{meaning} \backslash \text{B}$: macro:->54936294521339832251
 38128786223912807341050049847605059532189961231327664902288388132878702
 444582075129603152041054804964625083138567652624386837205668069376.

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

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

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

5 Assignments

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

`\xintAssign\xintBezout{3570902836026}{200467139463}\to\A\B\U\V\D`
gives then $\backslash U$: macro:->5812117166, $\backslash V$: macro:->103530711951 and $\backslash D=3$.

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 $\backslash Out$ to be macro with one parameter, $\backslash Out\{0\}$ gives the size N of the array and $\backslash Out\{n\}$, for n from 1 to N then gives the n th element of the array, here the n th digit of 2^{100} , from the most significant to the least significant. As usual, the generated macro $\backslash Out$ 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 $\backslash Out\{0\}$ digits and the sum of their squares is $\backslash the\cntb$. These digits are, from the least to the most significant: $\backslash cnta = \backslash Out\{0\}$
 $\backslash loop \backslash Out\{\backslash the\cnta\} \backslash ifnum \cnta > 1 \backslash advance\cnta -1 , \backslash repeat.$
 $\backslash endgroup$

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 $\backslash Out$ array: indeed internally, besides $\backslash Out$ itself, additional macros are defined which are $\backslash Out0$, $\backslash Out00$, $\backslash Out1$, $\backslash Out2$, ..., $\backslash OutN$, where N is the size of the array (which is the value returned by $\backslash Out\{0\}$; the digits are parts of the names not arguments).

The command `\xintRelaxArray` $\backslash Out$ sets all these macros to $\backslash relax$, but it was simpler to put everything within 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, $\backslash xintSum$ 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.

6 Error messages

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 `\xintTypesetEuclideanAlgorithm`:

```
\xintAssignArray\xintEuclideanAlgorithm {#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 `\xintTypesetEuclideanAlgorithm` 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 mechanism of reload and ε -TeX detection, especially for Plain TeX. As ε -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 mechanism. 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 `\xintTypesetEuclideanAlgorithm` 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, or 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{10000000000000}{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 *N*th 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 \times 1000 digits is five times faster than 1000 digits \times 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}}}
=2678727931661577575766279517007548402324740266374015348974459614815426
412965499490000444007240765727130000165312076406545621180143571994015903
343539244028212438966822248927862988084382716133376
=\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: $N = QM + R$, $0 \leq R < |M|$. 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:->12345678900000000000000000000000.
\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 **\xintEuclideanAlgorithm**

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

```
\xintAssign {\xintEuclideanAlgorithm {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 Euclidean 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 {\xintEuclideanAlgorithm {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 **\xintTypesetEuclideanAlgorithm**

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

```
\xintTypesetEuclideanAlgorithm {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.

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

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.

12 Package *xint* implementation

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \expandafter\let\expandafter\x\csname ver@xint.sty\endcsname
13 \expandafter
14 \ifx\csname PackageInfo\endcsname\relax
15 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
16 \else
17 \def\y#1#2{\PackageInfo{#1}{#2}}%
18 \fi
19 \expandafter
20 \ifx\csname numexpr\endcsname\relax
21 \y{xint}{\numexpr not available, aborting input}%
22 \aftergroup\endinput
23 \else
24 \ifx\x\relax % plain-TeX, first loading
25 \else
26 \def\empty {}%
27 \ifx\x\empty % LaTeX, first loading,
28 % variable is initialized, but \ProvidesPackage not yet seen
29 \else
30 \y{xint}{I was already loaded, aborting input}%
31 \aftergroup\endinput
32 \fi
33 \fi
34 \fi
35 \def\ChangeCatcodesIfInputNotAborted
36 {%
37 \endgroup
38 \edef\XINT@restorecatcodes@endinput
39 {%
40 \catcode47=\the\catcode47 % /
41 \catcode41=\the\catcode41 % )
42 \catcode40=\the\catcode40 % (
43 \catcode42=\the\catcode42 % *
44 \catcode43=\the\catcode43 % +
45 \catcode62=\the\catcode62 % >
46 \catcode60=\the\catcode60 % <
47 \catcode58=\the\catcode58 % :
48 \catcode46=\the\catcode46 % .
49 \catcode45=\the\catcode45 % -
50 \catcode44=\the\catcode44 % ,

```

12 Package *xint* implementation

```
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
```

```

99      \xdef#1{#3}%
100     \fi
101     \ifx#1\relax
102       \xdef#1{#3}%
103     \fi
104   }%
105 \fi
106 \expandafter\x\csname ver@xint.sty\endcsname
107 \ProvidesPackage{xint}%
108 [2013/04/05 v1.02 Expandable operations on long numbers (jfb)]%

```

12.3 Token management macros

```

109 \def\xint@gobble      #1{}%
110 \def\xint@gobble@one #1{}%
111 \def\xint@gobble@two  #1#2{}%
112 \def\xint@gobble@three #1#2#3{}%
113 \def\xint@gobble@four  #1#2#3#4{}%
114 \def\xint@gobble@five  #1#2#3#4#5{}%
115 \def\xint@gobble@six   #1#2#3#4#5#6{}%
116 \def\xint@gobble@seven #1#2#3#4#5#6#7{}%
117 \def\xint@gobble@eight #1#2#3#4#5#6#7#8{}%
118 \def\xint@secondoftwo  #1#2{#2}%
119 \def\xint@firstoftwo@andstop #1#2{ #1}%
120 \def\xint@secondoftwo@andstop #1#2{ #2}%
121 \def\xint@exchangetwo@keepbraces #1#2{{#2}{#1}}%
122 \def\xint@exchangetwo@keepbraces@andstop #1#2{{#2}{#1}}%
123 \def\xint@xpxp@andstop {\expandafter\expandafter\expandafter\space}%
124 \def\xint@r      #1\R {}%
125 \def\xint@w      #1\W {}%
126 \def\xint@z      #1\Z {}%
127 \def\xint@zero   #10{}%
128 \def\xint@one    #11{}%
129 \def\xint@minus  #1-{}%
130 \def\xint@relax  #1\relax {}%
131 \def\xint@quatrezeros #10000{}%
132 \def\xint@bracedundef {\xint@undef}%
133 \def\xint@UDzerofork    #10\dummy #2#3\xint@UDkrof {#2}%
134 \def\xint@UDsignfork    #1-\dummy #2#3\xint@UDkrof {#2}%
135 \def\xint@UDzerosfork    #100\dummy #2#3\xint@UDkrof {#2}%
136 \def\xint@UDonezerofork  #110\dummy #2#3\xint@UDkrof {#2}%
137 \def\xint@UDzerominusfork #10-\dummy #2#3\xint@UDkrof {#2}%
138 \def\xint@UDsignsfork    #1--\dummy #2#3\xint@UDkrof {#2}%
139 \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.

```

140 \def\xintRev {\romannumeral0\xintrev}%
141 \def\xintrev #1%
142 {%

```

12 Package *xint* implementation

```

143 \expandafter\expandafter\expandafter
144 \xint@rev
145 \expandafter\expandafter\expandafter
146 {#1}%
147 }%
148 \def\xint@rev #1%
149 {%
150 \XINT@rev@fork #1\Z
151 }%
152 \def\XINT@rev@fork #1#2%
153 {%
154 \xint@UDsignfork
155 #1\dummy \XINT@rev@negative
156 -\dummy \XINT@rev@nonnegative
157 \xint@UDkrof
158 #1#2%
159 }%
160 \def\XINT@rev@negative #1#2\Z
161 {%
162 \expandafter
163 \space
164 \expandafter
165 -%
166 \romannumeral0\XINT@rev {#2}%
167 }%
168 \def\XINT@rev@nonnegative #1\Z
169 {%
170 \XINT@rev {#1}%
171 }%
172 \def\XINT@Rev {\romannumeral0\XINT@rev }%
173 \let\xintReverseOrder \XINT@Rev
174 \def\XINT@rev #1%
175 {%
176 \XINT@rord@main {}#1%
177 \xint@UNDEF
178 \xint@undef\xint@undef\xint@undef\xint@undef
179 \xint@undef\xint@undef\xint@undef\xint@undef
180 \xint@UNDEF
181 }%
182 \def\XINT@rord@main #1#2#3#4#5#6#7#8#9%
183 {%
184 \XINT@strip@undef #9\XINT@rord@cleanup\xint@undef
185 \XINT@rord@main {#9#8#7#6#5#4#3#2#1}%
186 }%
187 \def\XINT@rord@cleanup\xint@undef\XINT@rord@main #1#2\xint@UNDEF
188 {%
189 \expandafter\space\XINT@strip@UNDEF #1%
190 }%
191 \def\XINT@strip@undef #1\xint@undef {}%
192 \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

```

193 \def\XINT@RQ #1#2#3#4#5#6#7#8#9%
194 {%
195   \xint@r #9\XINT@RQ@end\R
196   \XINT@RQ {#9#8#7#6#5#4#3#2#1}%
197 }%
198 \def\XINT@RQ@end\R\XINT@RQ #1#2\Z
199 {%
200   \XINT@RQ@end@ #1\Z
201 }%
202 \def\XINT@RQ@end@ #1#2#3#4#5#6#7#8%
203 {%
204   \xint@r #8\XINT@RQ@end@viii
205           #7\XINT@RQ@end@vii
206           #6\XINT@RQ@end@vi
207           #5\XINT@RQ@end@v
208           #4\XINT@RQ@end@iv
209           #3\XINT@RQ@end@iii
210           #2\XINT@RQ@end@ii
211           \R\XINT@RQ@end@i
212           \Z #2#3#4#5#6#7#8%
213 }%
214 \def\XINT@RQ@end@viii #1\Z #2#3#4#5#6#7#8#9\Z { #9}%
215 \def\XINT@RQ@end@vii #1\Z #2#3#4#5#6#7#8#9\Z { #8#9000}%
216 \def\XINT@RQ@end@vi #1\Z #2#3#4#5#6#7#8#9\Z { #7#8#900}%
217 \def\XINT@RQ@end@v #1\Z #2#3#4#5#6#7#8#9\Z { #6#7#8#90}%
218 \def\XINT@RQ@end@iv #1\Z #2#3#4#5#6#7#8#9\Z { #5#6#7#8#9}%
219 \def\XINT@RQ@end@iii #1\Z #2#3#4#5#6#7#8#9\Z { #4#5#6#7#8#9000}%
220 \def\XINT@RQ@end@ii #1\Z #2#3#4#5#6#7#8#9\Z { #3#4#5#6#7#8#900}%
221 \def\XINT@RQ@end@i #1\Z #2#3#4#5#6#7#8\Z { #1#2#3#4#5#6#7#80}%

```

12.6 \XINT@cuz

```

222 \def\xint@cleanupzeros@andstop #1#2#3#4%
223 {\expandafter
224   \space
225   \the\numexpr #1#2#3#4\relax
226 }%
227 \def\xint@cleanupzeros@nospace #1#2#3#4%
228 {%
229   \the\numexpr #1#2#3#4\relax
230 }%
231 \def\XINT@Rev@andcleanupzeros #1%
232 {%
233   \romannumeral0\expandafter
234     \xint@cleanupzeros@andstop
235   \romannumeral0\XINT@rord@main {}#1%
236   \xint@UNDEF

```


12 Package **xint** implementation

```

237      \xint@undef\xint@undef\xint@undef\xint@undef
238      \xint@undef\xint@undef\xint@undef\xint@undef
239      \xint@UNDEF
240 }%
  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

241 \def\xINT@cuz #1%
242 {%
243   \XINT@cuz@loop #1\W\W\W\W\W\W\W\Z%
244 }%
245 \def\xINT@cuz@loop #1#2#3#4#5#6#7#8%
246 {%
247   \xint@w #8\xint@cuz@enda\W
248   \xint@z #8\xint@cuz@endb\Z
249   \XINT@cuz@checka {#1#2#3#4#5#6#7#8}%
250 }%
251 \def\xint@cuz@enda #1\xINT@cuz@checka #2%
252 {%
253   \xint@cuz@endaa #2%
254 }%
255 \def\xint@cuz@endaa #1#2#3#4#5\Z
256 {%
257   \expandafter\space\the\numexpr #1#2#3#4\relax
258 }%
259 \def\xint@cuz@endb\Z\xINT@cuz@checka #1{ 0}%
260 \def\xINT@cuz@checka #1%
261 {%
262   \expandafter \XINT@cuz@checkb \the\numexpr #1\relax
263 }%
264 \def\xINT@cuz@checkb #1%
265 {%
266   \xint@zero #1\xint@cuz@backto loop 0\xINT@cuz@Stop #1%
267 }%
268 \def\xINT@cuz@Stop #1\W #2\Z{ #1}%
269 \def\xint@cuz@backto loop 0\xINT@cuz@Stop 0{\XINT@cuz@loop }%

```

12.7 \xintNum

For example \xintNum {-----+-----000000000000003}

```

270 \def\xintNum {\romannumeral0\xintnum }%
271 \def\xintnum #1%
272 {%
273   \expandafter\expandafter\expandafter
274   \XINT@num
275   \expandafter\expandafter\expandafter
276   {#1}%

```

```

277 }%
278 \def\XINT@Num {\romannumeral0\XINT@num }%
279 \def\XINT@num #1{\XINT@num@loop #1\R\R\R\R\R\R\R\R\Z }%
280 \def\XINT@num@loop #1#2#3#4#5#6#7#8%
281 {%
282   \xint@r #8\XINT@num@end\R\XINT@num@NumEight #1#2#3#4#5#6#7#8%
283 }%
284 \def\XINT@num@end\R\XINT@num@NumEight #1\R #2\Z
285 {%
286   \expandafter\space\the\numexpr #1+0\relax
287 }%
288 \def\XINT@num@NumEight #1#2#3#4#5#6#7#8%
289 {%
290   \ifnum \numexpr #1#2#3#4#5#6#7#8+0\relax = 0
291     \xint@afterfi {\expandafter\XINT@num@keepsign@a
292                   \the\numexpr #1#2#3#4#5#6#7#81\relax}%
293   \else
294     \xint@afterfi {\expandafter\XINT@num@finish
295                   \the\numexpr #1#2#3#4#5#6#7#8\relax}%
296   \fi
297 }%
298 \def\XINT@num@keepsign@a #1%
299 {%
300   \xint@one#1\XINT@num@gobackto loop 1\XINT@num@keepsign@b
301 }%
302 \def\XINT@num@gobackto loop 1\XINT@num@keepsign@b {\XINT@num@loop }%
303 \def\XINT@num@keepsign@b #1{\XINT@num@loop -}%
304 \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

```

305 \def\xintLen {\romannumeral0\xintlen }%
306 \def\xintlen #1%
307 {%
308   \expandafter\expandafter\expandafter
309   \XINT@length@fork #1\R\R\R\R\R\R\R\R\Z
310 }%
311 \def\XINT@Len #1{\romannumeral0\XINT@length@fork #1\R\R\R\R\R\R\R\R\Z }%
312 \def\XINT@length@fork #1%
313 {%
314   \expandafter\XINT@length@loop
315   \xint@UDsignfork
316   #1\dummy {{0}}%
317   -\dummy {{0}}#1}%
318   \xint@UDkrof
319 }%
320 \def\XINT@Length #1{\romannumeral0\XINT@length@loop {0}#1\R\R\R\R\R\R\R\R\Z }%
321 \def\XINT@length #1{\XINT@length@loop {0}#1\R\R\R\R\R\R\R\R\Z }%
322 \let\xintLength\XINT@Length
323 \def\XINT@length@loop #1#2#3#4#5#6#7#8#9%

```

```

324 {%
325   \xint@r #9\XINT@length@end {#2#3#4#5#6#7#8#9}\R
326   \expandafter\XINT@length@loop\expandafter {\the\numexpr #1+8\relax}%
327 }%
328 \def\XINT@length@end #1\R\expandafter\XINT@length@loop\expandafter #2#3\Z
329 {%
330   \XINT@length@end@ #1\W\W\W\W\W\W\W\Z {#2}%
331 }%
332 \def\XINT@length@end@ #1\R #2#3#4#5#6#7#8#9\Z
333 {%
334   \xint@w #2\XINT@length@end@i
335           #3\XINT@length@end@ii
336           #4\XINT@length@end@iii
337           #5\XINT@length@end@iv
338           #6\XINT@length@end@v
339           #7\XINT@length@end@vi
340           #8\XINT@length@end@vii
341           \W\XINT@length@end@viii
342 }%
343 \def\XINT@length@end@viii #1%
344   {\expandafter\space\the\numexpr #1-8\relax}%
345 \def\XINT@length@end@vii #1\XINT@length@end@viii #2%
346   {\expandafter\space\the\numexpr #2-7\relax}%
347 \def\XINT@length@end@vi #1\XINT@length@end@viii #2%
348   {\expandafter\space\the\numexpr #2-6\relax}%
349 \def\XINT@length@end@v #1\XINT@length@end@viii #2%
350   {\expandafter\space\the\numexpr #2-5\relax}%
351 \def\XINT@length@end@iv #1\XINT@length@end@viii #2%
352   {\expandafter\space\the\numexpr #2-4\relax}%
353 \def\XINT@length@end@iii #1\XINT@length@end@viii #2%
354   {\expandafter\space\the\numexpr #2-3\relax}%
355 \def\XINT@length@end@ii #1\XINT@length@end@viii #2%
356   {\expandafter\space\the\numexpr #2-2\relax}%
357 \def\XINT@length@end@i #1\XINT@length@end@viii #2%
358   {\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

```

359 \def\xintAssign #1\to
360 {%
361   \expandafter\expandafter\expandafter
362   \XINT@assign@a #1{}\to
363 }%
364 \def\XINT@assign@a #1% attention to the # at the beginning of next line
365 #{%
366   \def\xint@temp {#1}%
367   \ifx\empty\xint@temp
368     \expandafter\XINT@assign@b

```

12 Package *xint* implementation

```

369     \else
370         \expandafter\XINT@assign@B
371     \fi
372 }%
373 \def\XINT@assign@b #1#2\to #3%
374 {%
375     \edef #3{#1}\def\xint@temp {#2}%
376     \ifx\empty\xint@temp
377         \else
378         \xint@afterfi{\XINT@assign@a #2\to }%
379     \fi
380 }%
381 \def\XINT@assign@B #1\to #2%
382 {%
383     \edef #2{\xint@temp}%
384 }%
385 \def\xintRelaxArray #1%
386 {%
387     \edef\XINT@restoreescapechar {\escapechar\the\escapechar\relax}%
388     \escapechar -1
389     \edef\xint@arrayname {\string #1}%
390     \XINT@restoreescapechar
391     \expandafter\let\expandafter\xint@temp
392         \csname\xint@arrayname 0\endcsname
393     \count 255 0
394     \loop
395         \global\expandafter\let
396             \csname\xint@arrayname\the\count255\endcsname\relax
397         \ifnum \count 255 < \xint@temp
398             \advance\count 255 1
399         \repeat
400     \global\expandafter\let\csname\xint@arrayname 00\endcsname\relax
401     \global\let #1\relax
402 }%
403 \def\xintAssignArray #1\to #2%
404 {%
405     \edef\XINT@restoreescapechar {\escapechar\the\escapechar\relax}%
406     \escapechar -1
407     \edef\xint@arrayname {\string #1}%
408     \XINT@restoreescapechar
409     \count 255 0
410     \expandafter\expandafter\expandafter
411     \XINT@assignarray@loop #1\xint@undef
412     \csname\xint@arrayname 00\endcsname
413     \csname\xint@arrayname 0\endcsname
414     {\xint@arrayname}%
415     #2%
416 }%
417 \def\XINT@assignarray@loop #1%
418 {%
419     \def\xint@temp {#1}%
420     \ifx\xint@bracedundef\xint@temp

```

```

421     \edef\xint@temp{\the\count 255 }%
422     \expandafter\let\csname\xint@arrayname0\endcsname\xint@temp
423     \expandafter\XINT@assignarray@end
424   \else
425     \advance\count 255 1
426     \expandafter\edef
427       \csname\xint@arrayname\the\count 255\endcsname{\xint@temp}%
428     \expandafter\XINT@assignarray@loop
429   \fi
430 }%
431 \def\XINT@assignarray@end {\expandafter\XINT@assignarray@@end }%
432 \def\XINT@assignarray@@end #1%
433 {%
434   \expandafter\XINT@assignarray@@@end\expandafter #1%
435 }%
436 \def\XINT@assignarray@@@end #1#2#3%
437 {%
438   \expandafter\XINT@assignarray@@@@end
439   \expandafter #1\expandafter #2\expandafter{#3}%
440 }%
441 \def\XINT@assignarray@@@@end #1#2#3#4%
442 {%
443   \def #4##1%
444   {\romannumeral0%
445     \expandafter\expandafter\expandafter
446     #1%
447     \expandafter\expandafter\expandafter
448     {##1}%
449   }%
450   \def #1##1%
451   {%
452     \ifnum ##1< 0
453       \xint@afterfi {\xintError:ArrayIndexIsNegative
454         \expandafter\space 0}%
455     \else
456       \xint@afterfi {%
457         \ifnum ##1> #2
458           \xint@afterfi {\xintError:ArrayIndexBeyondLimit
459             \expandafter\space 0}%
460         \else
461           \xint@afterfi
462           {\expandafter\expandafter\expandafter
463             \space\csname #3##1\endcsname}%
464         \fi}%
465     \fi
466   }%
467 }%
468 \let\xintDigitsOf\xintAssignArray

```

12.10 \xintSgn

```

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

```

```

470 \def\xintsgn #1%
471 {%
472   \expandafter\expandafter\expandafter
473   \XINT@sgn #1\Z%
474 }%
475 \def\XINT@Sgn #1{\romannumeral0\XINT@sgn #1\Z}%
476 \def\XINT@sgn #1%
477 {%
478   \xint@xpxp@andstop
479   \xint@UDzerominusfork
480   #1-\dummy {\expandafter0}% zero
481   0#1\dummy {\expandafter-\expandafter1}% n\'egatif
482   0-\dummy {\expandafter1}% positif
483   \xint@UDkrof
484   \xint@z
485 }%

```

12.11 \xintOpp

```

486 \def\xintOpp {\romannumeral0\xintopp}%
487 \def\xintopp #1%
488 {%
489   \expandafter\expandafter\expandafter
490   \XINT@opp #1%
491 }%
492 \def\XINT@Opp #1{\romannumeral0\XINT@opp #1}%
493 \def\XINT@opp #1%
494 {%
495   \expandafter\space
496   \xint@UDzerominusfork
497   #1-\dummy 0% zero
498   0#1\dummy {}% negative
499   0-\dummy {-#1}% positive
500   \xint@UDkrof
501 }%

```

12.12 \xintAbs

```

502 \def\xintAbs {\romannumeral0\xintabs}%
503 \def\xintabs #1%
504 {%
505   \expandafter\expandafter\expandafter
506   \XINT@abs #1%
507 }%
508 \def\XINT@Abs {\romannumeral0\XINT@abs}%
509 \def\XINT@abs #1%
510 {%
511   \xint@UDsignfork
512   #1\dummy \space
513   -\dummy { #1}%
514   \xint@UDkrof
515 }%

```

```

-----
-----

```

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

12.13 \xintAdd

```

516 \def\xintAdd {\romannumeral0\xintadd }%
517 \def\xintadd #1%
518 {%
519     \expandafter\expandafter\expandafter
520         \xint@add
521     \expandafter\expandafter\expandafter
522         {#1}%
523 }%
524 \def\xint@add #1#2%
525 {%
526     \expandafter\expandafter\expandafter
527     \XINT@add@fork #2\Z #1\Z
528 }%
529 \def\XINT@Add #1#2{\romannumeral0\XINT@add@fork #2\Z #1\Z }%
530 \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]

531 \def\XINT@add@fork #1#2\Z #3#4\Z
532 {%
533     \xint@UDzerofork
534     #1\dummy \XINT@add@secondiszero
535     #3\dummy \XINT@add@firstiszero
536     0\dummy
537     {\xint@UDsignsfork
538         #1#3\dummy \XINT@add@minusminus           % #1 = #3 = -
539         #1-\dummy \XINT@add@minusplus             % #1 = -
540         #3-\dummy \XINT@add@plusminus             % #3 = -
541         --\dummy \XINT@add@plusplus
542     \xint@UDkrof }%
543     \xint@UDkrof
544     {#2}{#4}#1#3%
545 }%
546 \def\XINT@add@secondiszero #1#2#3#4{ #4#2}%
547 \def\XINT@add@firstiszero #1#2#3#4{ #3#1}%

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

548 \def\XINT@add@minusminus #1#2#3#4%
549 {%
550     \expandafter\space\expandafter-%
551     \romannumeral0\XINT@add@pre {#2}{#1}%
552 }%
553 \def\XINT@add@minusplus #1#2#3#4%
554 {%
555     \XINT@sub@pre {#4#2}{#1}%
556 }%
```

12 Package *xint* implementation

```

557 \def\XINT@add@plusminus #1#2#3#4%
558 {%
559   \XINT@sub@pre {#3#1}{#2}%
560 }%
561 \def\XINT@add@plusplus #1#2#3#4%
562 {%
563   \XINT@add@pre {#4#2}{#3#1}%
564 }%
565 \def\XINT@add@pre #1%
566 {%
567   \expandafter\XINT@add@@pre\expandafter{%
568     \romannumeral0\XINT@RQ {}#1\R\R\R\R\R\R\R\R\Z
569   }%
570 }%
571 \def\XINT@add@@pre #1#2%
572 {%
573   \expandafter\XINT@add@A
574     \expandafter0\expandafter{\expandafter}%
575     \romannumeral0\XINT@RQ {}#2\R\R\R\R\R\R\R\R\Z
576     \W\X\Y\Z #1\W\X\Y\Z
577 }%

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

578 \def\XINT@add@A #1#2#3#4#5#6%
579 {%
580   \xint@w
581   #3\xint@add@az
582   \W\XINT@add@AB #1{#3#4#5#6}{#2}%
583 }%

1er nombre fini.

584 \def\xint@add@az\W\XINT@add@AB #1#2%
585 {%
586   \XINT@add@AC@checkcarry #1%
587 }%
```

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

12 Package *xint* implementation

```

588 \def\XINT@add@AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
589 {%
590     \xint@w
591     #5\xint@add@bz
592     \W\XINT@add@ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
593 }%
594 \def\XINT@add@ABE #1#2#3#4#5#6%
595 {\expandafter
596     \XINT@add@ABEA\the\numexpr #1+10#5#4#3#2+#6\relax.%
597 }%
598 \def\XINT@add@ABEA #1#2#3.#4%
599 {%
600     \XINT@add@A #2{#3#4}%
601 }%

    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

602 \def\xint@add@bz\W\XINT@add@ABE #1#2#3#4#5#6%
603 {\expandafter
604     \XINT@add@CC\the\numexpr #1+10#5#4#3#2\relax.%
605 }%
606 \def\XINT@add@CC #1#2#3.#4%
607 {%
608     \XINT@add@AC@checkcarry #2{#3#4}% on va examiner et \'eliminer #2
609 }%

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

610 \def\XINT@add@AC@checkcarry #1%
611 {%
612     \xint@zero #1\xint@add@AC@nocarry 0\XINT@add@C
613 }%
614 \def\xint@add@AC@nocarry 0\XINT@add@C #1#2\W\X\Y\Z
615 {%
616     \expandafter
617     \xint@cleanupzeros@andstop
618     \romannumeral0%
619     \XINT@rord@main {}#2%
620     \xint@UNDEF
621     \xint@undef\xint@undef\xint@undef\xint@undef
622     \xint@undef\xint@undef\xint@undef\xint@undef
623     \xint@UNDEF
624     #1%
625 }%
626 \def\XINT@add@C #1#2#3#4#5%
627 {%
628     \xint@w
629     #2\xint@add@cz
630     \W\XINT@add@CD {#5#4#3#2}{#1}%
631 }%

```

```

632 \def\XINT@add@CD #1%
633 {\expandafter
634   \XINT@add@CC\the\numexpr 1+10#1\relax.%
635 }%
636 \def\xint@add@cz\W\XINT@add@CD #1#2{ 1#2}%

```

12.14 \xintSub

```

637 \def\xintSub {\romannumeral0\xintsub }%
638 \def\xintsub #1%
639 {%
640   \expandafter\expandafter\expandafter
641     \xint@sub
642   \expandafter\expandafter\expandafter
643     {#1}%
644 }%
645 \def\xint@sub #1#2%
646 {%
647   \expandafter\expandafter\expandafter
648     \XINT@sub@fork #2\Z #1\Z
649 }%
650 \def\XINT@Sub #1#2{\romannumeral0\XINT@sub@fork #2\Z #1\Z }%
651 \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*

652 \def\XINT@sub@fork #1#2\Z #3#4\Z
653 {%
654   \xint@UDsignsfork
655     #1#3\dummy \XINT@sub@minusminus
656     #1-\dummy \XINT@sub@minusplus % attention, #3=0 possible
657     #3-\dummy \XINT@sub@plusminus % attention, #1=0 possible
658     --\dummy {\xint@UDzerofork
659       #1\dummy \XINT@sub@secondiszero
660       #3\dummy \XINT@sub@firstiszero
661       0\dummy \XINT@sub@plusplus
662       \xint@UDkrof }%
663   \xint@UDkrof
664   {#2}{#4}#1#3%
665 }%
666 \def\XINT@sub@secondiszero #1#2#3#4{ #4#2}%
667 \def\XINT@sub@firstiszero #1#2#3#4{ -#3#1}%
668 \def\XINT@sub@plusplus #1#2#3#4%
669 {%
670   \XINT@sub@pre {#4#2}{#3#1}%
671 }%
672 \def\XINT@sub@minusminus #1#2#3#4%
673 {%
674   \XINT@sub@pre {#1}{#2}%
675 }%
676 \def\XINT@sub@minusplus #1#2#3#4%

```

12 Package *xint* implementation

```

677 {%
678   \xint@zero #4\xint@sub@mp0\XINT@add@pre {#4#2}{#1}%
679 }%
680 \def\xint@sub@mp0\XINT@add@pre #1#2{ #2}%
681 \def\XINT@sub@plusminus #1#2#3#4%
682 {%
683   \xint@zero #3\xint@sub@pm0\expandafter\space\expandafter-%
684   \romannumeral0\XINT@add@pre {#2}{#3#1}%
685 }%
686 \def\xint@sub@pm #1\XINT@add@pre #2#3{ -#2}%
687 \def\XINT@sub@pre #1%
688 {%
689   \expandafter\XINT@sub@@pre\expandafter{%
690     \romannumeral0\XINT@RQ {}#1\R\R\R\R\R\R\R\R\Z
691   }%
692 }%
693 \def\XINT@sub@@pre #1#2%
694 {%
695   \expandafter\XINT@sub@A
696   \expandafter1\expandafter{\expandafter}%
697   \romannumeral0\XINT@RQ {}#2\R\R\R\R\R\R\R\R\Z
698   \W\X\Y\Z #1 \W\X\Y\Z
699 }%

\romannumeral0\XINT@sub@A 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.

700 \def\XINT@sub@A #1#2#3\W\X\Y\Z #4#5#6#7%
701 {%
702   \xint@w
703   #4\xint@sub@az
704   \W\XINT@sub@B #1{#4#5#6#7}{#2}#3\W\X\Y\Z
705 }%
706 \def\XINT@sub@B #1#2#3#4#5#6#7%
707 {%
708   \xint@w
709   #4\xint@sub@bz
710   \W\XINT@sub@onestep #1#2{#7#6#5#4}{#3}%
711 }%

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.

712 \def\XINT@sub@onestep #1#2#3#4#5#6%
713 {\expandafter
714   \XINT@sub@backtoA\the\numexpr 11#5#4#3#2-#6+#1-1\relax.%
715 }%

```

ON PRODUIT LE RÉSULTAT DANS LE BON ORDRE

```

716 \def\XINT@sub@backtoA #1#2#3.#4%
717 {%
718   \XINT@sub@A #2{#3#4}%
719 }%
720 \def\xint@sub@bz
721   \W\XINT@sub@onestep #1#2#3#4#5#6#7%
722 {%
723   \xint@UDzerofork
724   #1\dummy \XINT@sub@C % une retenue
725   0\dummy \XINT@sub@D % pas de retenue
726   \xint@UDkrof
727   {#7}#2#3#4#5%
728 }%
729 \def\XINT@sub@D #1#2\W\X\Y\Z
730 {%
731   \expandafter
732   \xint@cleanupzeros@andstop
733   \romannumeral0%
734   \XINT@rord@main {}#2%
735   \xint@UNDEF
736   \xint@undef\xint@undef\xint@undef\xint@undef
737   \xint@undef\xint@undef\xint@undef\xint@undef
738   \xint@UNDEF
739   #1%
740 }%
741 \def\XINT@sub@C #1#2#3#4#5%
742 {%
743   \xint@w
744   #2\xint@sub@cz
745   \W\XINT@sub@AC@onestep {#5#4#3#2}{#1}%
746 }%
747 \def\XINT@sub@AC@onestep #1%
748 {\expandafter
749   \XINT@sub@backtoC\the\numexpr 11#1-1\relax.%
750 }%
751 \def\XINT@sub@backtoC #1#2#3.#4%
752 {%
753   \XINT@sub@AC@checkcarry #2{#3#4}% la retenue va \^etre examin\’ee
754 }%
755 \def\XINT@sub@AC@checkcarry #1%
756 {%
757   \xint@one #1\xint@sub@AC@nocarry 1\XINT@sub@C
758 }%
759 \def\xint@sub@AC@nocarry 1\XINT@sub@C #1#2\W\X\Y\Z
760 {%
761   \expandafter
762   \XINT@cz@loop
763   \romannumeral0%
764   \XINT@rord@main {}#2%
765   \xint@UNDEF

```

12 Package *xint* implementation

```

766      \xint@undef\xint@undef\xint@undef\xint@undef
767      \xint@undef\xint@undef\xint@undef\xint@undef
768      \xint@UNDEF
769      #1\W\W\W\W\W\W\W\Z
770 }%
771 \def\xint@sub@cz\W\XINT@sub@AC@onestep #1%
772 {%
773     \XINT@cuz
774 }%
775 \def\xint@sub@az\W\XINT@sub@B #1#2#3#4#5#6#7%
776 {%
777     \xint@w
778     #4\xint@sub@ez
779     \W\XINT@sub@Eenter #1{#3}#4#5#6#7%
780 }%

    le premier nombre continue, le résultat sera < 0.

781 \def\XINT@sub@Eenter #1#2%
782 {%
783     \expandafter
784     \XINT@sub@E\expandafter1\expandafter{\expandafter}%
785     \romannumeral0%
786     \XINT@rord@main {}#2%
787     \xint@UNDEF
788     \xint@undef\xint@undef\xint@undef\xint@undef
789     \xint@undef\xint@undef\xint@undef\xint@undef
790     \xint@UNDEF
791     \W\X\Y\Z #1%
792 }%
793 \def\XINT@sub@E #1#2#3#4#5#6%
794 {%
795     \xint@w #3\xint@sub@F\W\XINT@sub@Eonestep
796     #1{#6#5#4#3}{#2}%
797 }%
798 \def\XINT@sub@Eonestep #1#2%
799 {\expandafter
800     \XINT@sub@backtoE\the\numexpr 110000-#2+#1-1\relax.%
801 }%
802 \def\XINT@sub@backtoE #1#2#3.#4%
803 {%
804     \XINT@sub@E #2{#3#4}%
805 }%
806 \def\xint@sub@F\W\XINT@sub@Eonestep #1#2#3#4%
807 {%
808     \xint@UDonezerofork
809     #4#1\dummy {\XINT@sub@Fdec 0}% soustraire 1. Et faire signe -
810     #1#4\dummy {\XINT@sub@Finc 1}% additionner 1. Et faire signe -
811     10\dummy \XINT@sub@DD % terminer. Mais avec signe -
812     \xint@UDkrof
813     {#3}%
814 }%
815 \def\XINT@sub@DD

```

12 Package **xint** implementation

```

816 {\expandafter\space\expandafter-\romannumeral0\XINT@sub@D }%
817 \def\XINT@sub@Fdec #1#2#3#4#5#6%
818 {%
819     \xint@w
820     #3\xint@sub@Fdec@finish\W\XINT@sub@Fdec@onestep
821     #1{#6#5#4#3}{#2}%
822 }%
823 \def\XINT@sub@Fdec@onestep #1#2%
824 {\expandafter
825     \XINT@sub@backtoFdec\the\numexpr 11#2+#1-1\relax.%
826 }%
827 \def\XINT@sub@backtoFdec #1#2#3.#4%
828 {%
829     \XINT@sub@Fdec #2{#3#4}%
830 }%
831 \def\xint@sub@Fdec@finish\W\XINT@sub@Fdec@onestep #1#2%
832 {%
833     \expandafter\space\expandafter-\romannumeral0\XINT@cuz
834 }%
835 \def\XINT@sub@Finc #1#2#3#4#5#6%
836 {%
837     \xint@w
838     #3\xint@sub@Finc@finish\W\XINT@sub@Finc@onestep
839     #1{#6#5#4#3}{#2}%
840 }%
841 \def\XINT@sub@Finc@onestep #1#2%
842 {\expandafter
843     \XINT@sub@backtoFinc\the\numexpr 10#2+#1\relax.%
844 }%
845 \def\XINT@sub@backtoFinc #1#2#3.#4%
846 {%
847     \XINT@sub@Finc #2{#3#4}%
848 }%
849 \def\xint@sub@Finc@finish\W\XINT@sub@Finc@onestep #1#2#3%
850 {%
851     \xint@UDzerofork
852     #1\dummy {\expandafter\space\expandafter-
853               \xint@cleanupzeros@nospace}%
854     0\dummy { -1}%
855     \xint@UDkrof
856     #3%
857 }%
858 \def\xint@sub@ez\W\XINT@sub@Eenter #1%
859 {%
860     \xint@UDzerofork
861     #1\dummy \XINT@sub@K           %      il y a une retenue
862     0\dummy \XINT@sub@L           %      pas de retenue
863     \xint@UDkrof
864 }%
865 \def\XINT@sub@L #1\W\X\Y\Z
866     {\XINT@cuz@loop #1\W\W\W\W\W\W\W\Z }%
867 \def\XINT@sub@K #1%

```

```

868 {%
869   \expandafter
870   \XINT@sub@KK\expandafter1\expandafter{\expandafter}%
871   \romannumeral0%
872   \XINT@rord@main {}#1%
873   \xint@UNDEF
874   \xint@undef\xint@undef\xint@undef\xint@undef
875   \xint@undef\xint@undef\xint@undef\xint@undef
876   \xint@UNDEF
877 }%
878 \def\XINT@sub@KK #1#2#3#4#5#6%
879 {%
880   \xint@w
881   #3\xint@sub@KK@finish\W\XINT@sub@KK@onestep
882   #1{#6#5#4#3}{#2}%
883 }%
884 \def\XINT@sub@KK@onestep #1#2%
885 {\expandafter
886   \XINT@sub@backtoKK\the\numexpr 110000-#2+#1-1\relax.%
887 }%
888 \def\XINT@sub@backtoKK #1#2#3.#4%
889 {%
890   \XINT@sub@KK #2{#3#4}%
891 }%
892 \def\xint@sub@KK@finish\W\XINT@sub@KK@onestep #1#2#3%
893 {%
894   \expandafter\space\expandafter-\romannumeral
895   0\XINT@cuz@loop #3\W\W\W\W\W\W\W\W\Z
896 }%

```

12.15 \xintCmp

```

897 \def\xintCmp {\romannumeral0\xintcmp }%
898 \def\xintcmp #1%
899 {%
900   \expandafter\expandafter\expandafter
901   \xint@cmp
902   \expandafter\expandafter\expandafter
903   {#1}%
904 }%
905 \def\xint@cmp #1#2%
906 {%
907   \expandafter\expandafter\expandafter
908   \XINT@cmp@fork #2\Z #1\Z
909 }%
910 \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*

911 \def\XINT@cmp@fork #1#2\Z #3#4\Z
912 {%

```

12 Package *xint* implementation

```

913 \xint@UDsignsfork
914     #1#3\dummy \XINT@cmp@minusminus
915     #1-\dummy \XINT@cmp@minusplus
916     #3-\dummy \XINT@cmp@plusminus
917     --\dummy {\xint@UDzerosfork
918         #1#3\dummy \XINT@cmp@zerozero
919         #10\dummy \XINT@cmp@zeroplus
920         #30\dummy \XINT@cmp@pluszero
921         00\dummy \XINT@cmp@plusplus
922         \xint@UDkrof }%
923 \xint@UDkrof
924 {#2}{#4}#1#3%
925 }%
926 \def\XINT@cmp@minusplus #1#2#3#4{ 1}%
927 \def\XINT@cmp@plusminus #1#2#3#4{ -1}%
928 \def\XINT@cmp@zerozero #1#2#3#4{ 0}%
929 \def\XINT@cmp@zeroplus #1#2#3#4{ 1}%
930 \def\XINT@cmp@pluszero #1#2#3#4{ -1}%
931 \def\XINT@cmp@plusplus #1#2#3#4%
932 {%
933     \XINT@cmp@pre {#4#2}{#3#1}%
934 }%
935 \def\XINT@cmp@minusminus #1#2#3#4%
936 {%
937     \XINT@cmp@pre {#1}{#2}%
938 }%
939 \def\XINT@cmp@pre #1%
940 {%
941     \expandafter\XINT@cmp@@pre\expandafter{%
942     \romannumeral0\XINT@RQ {}#1\R\R\R\R\R\R\R\R\Z
943     }%
944 }%
945 \def\XINT@cmp@@pre #1#2%
946 {%
947     \expandafter\XINT@cmp@A
948     \expandafter1\expandafter{\expandafter}%
949     \romannumeral0\XINT@RQ {}#2\R\R\R\R\R\R\R\R\Z
950     \W\X\Y\Z #1\W\X\Y\Z
951 }%

```

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

```

952 \def\XINT@cmp@A #1#2#3\W\X\Y\Z #4#5#6#7%
953 {%
954     \xint@w
955     #4\xint@cmp@az
956     \W\XINT@cmp@B #1{#4#5#6#7}{#2}#3\W\X\Y\Z
957 }%

```


12 Package *xint* implementation

```

958 \def\XINT@cmp@B #1#2#3#4#5#6#7%
959 {%
960     \xint@w
961     #4\xint@cmp@bz
962     \W\XINT@cmp@onestep #1#2{#7#6#5#4}{#3}%
963 }%
964 \def\XINT@cmp@onestep #1#2#3#4#5#6%
965 {\expandafter
966     \XINT@cmp@backtoA\the\numexpr 11#5#4#3#2-#6+#1-1\relax.%
967 }%
968 \def\XINT@cmp@backtoA #1#2#3.#4%
969 {%
970     \XINT@cmp@A #2{#3#4}%
971 }%
972 \def\xint@cmp@bz
973     \W\XINT@cmp@onestep #1\Z { 1}%
974 \def\xint@cmp@az\W\XINT@cmp@B #1#2#3#4#5#6#7%
975 {%
976     \xint@w
977     #4\xint@cmp@ez
978     \W\XINT@cmp@Eenter #1{#3}#4#5#6#7%
979 }%
980 \def\XINT@cmp@Eenter #1\Z { -1}%
981 \def\xint@cmp@ez\W\XINT@cmp@Eenter #1%
982 {%
983     \xint@UDzerofork
984     #1\dummy \XINT@cmp@K                %      il y a une retenue
985     0\dummy \XINT@cmp@L                %      pas de retenue
986     \xint@UDkrof
987 }%
988 \def\XINT@cmp@K #1\Z { -1}%
989 \def\XINT@cmp@L #1{\XINT@OneIfPositive@main #1}%
990 \def\XINT@OneIfPositive #1%
991 {%
992     \XINT@OneIfPositive@main #1\W\X\Y\Z%
993 }%
994 \def\XINT@OneIfPositive@main #1#2#3#4%
995 {%
996     \xint@z #4\xint@OneIfPositive@terminated\Z\XINT@OneIfPositive@onestep
997     #1#2#3#4%
998 }%
999 \def\xint@OneIfPositive@terminated\Z\XINT@OneIfPositive@onestep\W\X\Y\Z { 0}%
1000 \def\XINT@OneIfPositive@onestep #1#2#3#4%
1001 {%
1002     \expandafter
1003     \XINT@OneIfPositive@check
1004     \the\numexpr #1#2#3#4\relax
1005 }%
1006 \def\XINT@OneIfPositive@check #1%
1007 {%
1008     \xint@zero
1009     #1\xint@OneIfPositive@backtomain 0\XINT@OneIfPositive@finish #1%

```

```

1010 }%
1011 \def\XINT@OneIfPositive@finish #1\W\X\Y\Z{ 1}%
1012 \def\xint@OneIfPositive@backtomain 0\XINT@OneIfPositive@finish 0%
1013          {\XINT@OneIfPositive@main }%

```

12.16 \xintGeq

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

```

1014 \def\xintGeq {\romannumeral0\xintgeq }%
1015 \def\xintgeq #1%
1016 {%
1017   \expandafter\expandafter\expandafter
1018   \xint@geq
1019   \expandafter\expandafter\expandafter
1020   {#1}%
1021 }%
1022 \def\xint@geq #1#2%
1023 {\expandafter\expandafter\expandafter
1024   \XINT@geq@fork #2\Z #1\Z
1025 }%
1026 \def\XINT@Geq #1#2{\romannumeral0\XINT@geq@fork #2\Z #1\Z }%

```

PLUS GRAND OU ÉGAL
ATTENTION, TESTE les VALEURS ABSOLUES

```

1027 \def\XINT@geq@fork #1#2\Z #3#4\Z
1028 {%
1029   \xint@UDzerofork
1030   #1\dummy \XINT@geq@secondiszero % |#1#2|=0
1031   #3\dummy \XINT@geq@firstiszero % |#1#2|>0
1032   0\dummy {\xint@UDsignsfork
1033             #1#3\dummy \XINT@geq@minusminus
1034             #1-\dummy \XINT@geq@minusplus
1035             #3-\dummy \XINT@geq@plusminus
1036             --\dummy \XINT@geq@plusplus
1037             \xint@UDkrof }%
1038   \xint@UDkrof
1039   {#2}{#4}#1#3%
1040 }%
1041 \def\XINT@geq@secondiszero #1#2#3#4{ 1}%
1042 \def\XINT@geq@firstiszero #1#2#3#4{ 0}%
1043 \def\XINT@geq@plusplus #1#2#3#4%
1044   {\XINT@geq@pre {#4#2}{#3#1}}%
1045 \def\XINT@geq@minusminus #1#2#3#4%
1046   {\XINT@geq@pre {#2}{#1}}%
1047 \def\XINT@geq@minusplus #1#2#3#4%
1048   {\XINT@geq@pre {#4#2}{#1}}%
1049 \def\XINT@geq@plusminus #1#2#3#4%
1050   {\XINT@geq@pre {#2}{#3#1}}%
1051 \def\XINT@geq@pre #1%
1052 {%

```

12 Package **xint** implementation

```

1053 \expandafter\XINT@geq@@pre\expandafter{%
1054 \romannumeral0\XINT@RQ {}#1\R\R\R\R\R\R\R\R\Z
1055 }%
1056 }%
1057 \def\XINT@geq@@pre #1#2%
1058 {%
1059 \expandafter\XINT@geq@A
1060 \expandafter1\expandafter{\expandafter}%
1061 \romannumeral0\XINT@RQ {}#2\R\R\R\R\R\R\R\R\Z
1062 \W\X\Y\Z #1 \W\X\Y\Z
1063 }%

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

1064 \def\XINT@geq@A #1#2#3\W\X\Y\Z #4#5#6#7%
1065 {%
1066 \xint@w
1067 #4\xint@geq@az
1068 \W\XINT@geq@B #1{#4#5#6#7}{#2}#3\W\X\Y\Z
1069 }%
1070 \def\XINT@geq@B #1#2#3#4#5#6#7%
1071 {%
1072 \xint@w
1073 #4\xint@geq@bz
1074 \W\XINT@geq@onestep #1#2{#7#6#5#4}{#3}%
1075 }%
1076 \def\XINT@geq@onestep #1#2#3#4#5#6%
1077 {\expandafter
1078 \XINT@geq@backtoA\the\numexpr 11#5#4#3#2-#6+#1-1\relax.%
1079 }%
1080 \def\XINT@geq@backtoA #1#2#3.#4%
1081 {%
1082 \XINT@geq@A #2{#3#4}%
1083 }%
1084 \def\xint@geq@bz\W\XINT@geq@onestep #1\W\X\Y\Z { 1}%
1085 \def\xint@geq@az\W\XINT@geq@B #1#2#3#4#5#6#7%
1086 {%
1087 \xint@w
1088 #4\xint@geq@ez
1089 \W\XINT@geq@Eenter #1%
1090 }%
1091 \def\XINT@geq@Eenter #1\W\X\Y\Z { 0}%
1092 \def\xint@geq@ez\W\XINT@geq@Eenter #1%
1093 {%
1094 \xint@UDzerofork
1095 #1\dummy { 0} % il y a une retenue
1096 0\dummy { 1} % pas de retenue

```

```

1097 \xint@UDkrof
1098 }%

```

12.17 \xintMax

```

1099 \def\xintMax {\romannumeral0\xintmax }%
1100 \def\xintmax #1%
1101 {%
1102   \expandafter\expandafter\expandafter
1103     \xint@max
1104   \expandafter\expandafter\expandafter
1105     {#1}%
1106 }%
1107 \def\xint@max #1#2%
1108 {%
1109   \expandafter\expandafter\expandafter
1110     \XINT@max@fork #2\Z #1\Z
1111 }%
1112 \def\XINT@Max #1#2{\romannumeral0\XINT@max@fork #2\Z #1\Z }%
1113   #3#4 vient du *premier*
1114   #1#2 vient du *second*
1115 \def\XINT@max@fork #1#2\Z #3#4\Z
1116 {%
1117   \xint@UDsignsfork
1118     #1#3\dummy \XINT@max@minusminus % A < 0, B < 0
1119     #1-\dummy \XINT@max@minusplus % B < 0, A >= 0
1120     #3-\dummy \XINT@max@plusminus % A < 0, B >= 0
1121     --\dummy {\xint@UDzerosfork
1122       #1#3\dummy \XINT@max@zerozero % A = B = 0
1123       #10\dummy \XINT@max@zeroplus % B = 0, A > 0
1124       #30\dummy \XINT@max@pluszero % A = 0, B > 0
1125       00\dummy \XINT@max@plusplus % A, B > 0
1126     }%
1127   \xint@UDkrof
1128   {#2}{#4}#1#3%
1129 }%

```

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

```

1128 \def\XINT@max@zerozero #1#2#3#4{ 0}%
1129 \def\XINT@max@zeroplus #1#2#3#4{ #4#2}%
1130 \def\XINT@max@pluszero #1#2#3#4{ #3#1}%
1131 \def\XINT@max@minusplus #1#2#3#4{ #4#2}%
1132 \def\XINT@max@plusminus #1#2#3#4{ #3#1}%
1133 \def\XINT@max@plusplus #1#2#3#4%
1134 {%
1135   \ifodd\XINT@Geq {#4#2}{#3#1}
1136   \xint@afterfi { #4#2}%
1137   \else
1138     \xint@afterfi { #3#1}%
1139   \fi
1140 }%

```

```

#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}%

```

```

1182 \def\XINT@min@plusminus #1#2#3#4{ #4#2}%
1183 \def\XINT@min@plusplus #1#2#3#4%
1184 {%
1185   \ifodd\XINT@Geq {#4#2}{#3#1}
1186   \xint@afterfi { #3#1}%
1187   \else
1188   \xint@afterfi { #4#2}%
1189   \fi
1190 }%

```

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

```

1191 \def\XINT@min@minusminus #1#2#3#4%
1192 {%
1193   \ifodd\XINT@Geq {#1}{#2}
1194   \xint@afterfi { -#1}%
1195   \else
1196   \xint@afterfi { -#2}%
1197   \fi
1198 }%

```

12.19 \xintSum, \xintSumExpr

```

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

```

```

1199 \def\XINT@psum #1%
1200 {%
1201   \romannumeral0\XINT@psum@checkifemptysum #1\Z
1202 }%
1203 \def\XINT@psum@checkifemptysum #1%
1204 {%
1205   \xint@relax #1\XINT@psum@returnzero\relax \XINT@psum@RQfirst #1%
1206 }%
1207 \def\XINT@psum@returnzero #1\Z { 0}%
1208 \def\XINT@psum@RQfirst #1\Z
1209 {%
1210   \expandafter\XINT@psum@loop\expandafter
1211   {\romannumeral0\XINT@RQ {}}#1\R\R\R\R\R\R\R\R\Z}%   avant: #1\Z
1212 }%
1213 \def\XINT@psum@loop #1#2%
1214 {%
1215   \xint@relax #2\XINT@psum@end\relax
1216   \expandafter
1217   \XINT@psum@loop\expandafter
1218   {\romannumeral0\expandafter\XINT@sum@A
1219   \expandafter0\expandafter{\expandafter}%
1220   \romannumeral0\XINT@RQ {}}#2\R\R\R\R\R\R\R\R\Z
1221   \W\X\Y\Z #1\W\X\Y\Z }%
1222 }%
1223 \def\XINT@psum@end\relax\expandafter
1224   \XINT@psum@loop\expandafter #1%
1225   {\XINT@psum@end@ #1}%

```

12 Package *xint* implementation

```

1226 \def\XINT@psum@end@ #1\W\X\Y\Z #2\W\X\Y\Z
1227 {%
1228     \expandafter
1229     \xint@cleanupzeros@andstop\romannumeral0\XINT@rev {#2}%
1230 }%
1231 \def\xintSumExpr {\romannumeral0\xintsumexpr }%
1232 \def\xintSum {\romannumeral0\xintsum }%
1233 \def\xintsum #1%
1234 {%
1235     \expandafter\expandafter\expandafter
1236     \xintsumexpr #1\relax
1237 }%
1238 \def\xintsumexpr #1%
1239 {%
1240     \expandafter\expandafter\expandafter
1241     \XINT@sum@checkifempty #1\Z {\XINT@psum }{\XINT@psum }%
1242 }%
1243 \def\XINT@sum@checkifempty #1%
1244 {%
1245     \xint@relax #1\XINT@sum@returnzero\relax
1246     \XINT@sum@checksign #1%
1247 }%
1248 \def\XINT@sum@returnzero #1\Z #2#3{ 0}%
1249 \def\XINT@sum@checksign #1%
1250 {%
1251     \xint@zero #1\XINT@sum@skipzeroinput0%
1252     \xint@UDsignfork
1253     #1\dummy \XINT@sum@pushneg
1254     -\dummy \XINT@sum@pushpos
1255     \xint@UDkrof
1256     #1%
1257 }%
1258 \def\XINT@sum@skipzeroinput #1\xint@UDkrof #2\Z #3#4%
1259 {%
1260     \XINT@sum@xpxpnext {#3}{#4}%
1261 }%
1262 \def\XINT@sum@pushpos #1#2\Z #3#4%
1263 {%
1264     \XINT@sum@xpxpnext {#3}{#1#2}}{#4}%
1265 }%
1266 \def\XINT@sum@pushneg #1#2\Z #3#4%
1267 {%
1268     \XINT@sum@xpxpnext {#3}{#4{#2}}}%
1269 }%
1270 \def\XINT@sum@xpxpnext #1#2#3%
1271 {%
1272     \expandafter\expandafter\expandafter
1273     \XINT@sum@checkiffinished #3\Z {#1}{#2}%
1274 }%
1275 \def\XINT@sum@checkiffinished #1%
1276 {%
1277     \xint@relax #1\XINT@sum@end\relax

```

12 Package **xint** implementation

```

1278 \XINT@sum@checksign #1%
1279 }%
1280 \def\XINT@sum@end\relax\XINT@sum@checksign\relax #1\Z #2#3%
1281 {\xintsub{#2\relax}{#3\relax}}%
1282 \def\XINT@sum@A #1#2#3#4#5#6%
1283 {%
1284 \xint@w
1285 #3\xint@sum@az
1286 \W\XINT@sum@B #1{#3#4#5#6}{#2}%
1287 }%
1288 \def\xint@sum@az\W\XINT@sum@B #1#2%
1289 {%
1290 \XINT@sum@AC@checkcarry #1%
1291 }%
1292 \def\XINT@sum@B #1#2#3#4\W\X\Y\Z #5#6#7#8%
1293 {%
1294 \xint@w
1295 #5\xint@sum@bz
1296 \W\XINT@sum@E #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
1297 }%
1298 \def\XINT@sum@E #1#2#3#4#5#6%
1299 {\expandafter
1300 \XINT@sum@ABEA\the\numexpr #1+10#5#4#3#2+#6\relax
1301 }%
1302 \def\XINT@sum@ABEA #1#2#3#4#5#6#7%
1303 {%
1304 \XINT@sum@A #2{#7#6#5#4#3}%
1305 }%
1306 \def\xint@sum@bz\W\XINT@sum@E #1#2#3#4#5#6%
1307 {\expandafter
1308 \XINT@sum@CC\the\numexpr #1+10#5#4#3#2\relax
1309 }%
1310 \def\XINT@sum@CC #1#2#3#4#5#6#7%
1311 {%
1312 \XINT@sum@AC@checkcarry #2{#7#6#5#4#3}%
1313 }%
1314 \def\XINT@sum@AC@checkcarry #1%
1315 {%
1316 \xint@zero #1\xint@sum@AC@nocarry 0\XINT@sum@C
1317 }%
1318 \def\xint@sum@AC@nocarry 0\XINT@sum@C #1#2\W\X\Y\Z { #1#2}%
1319 \def\XINT@sum@C #1#2#3#4#5%
1320 {%
1321 \xint@w
1322 #2\xint@sum@cz
1323 \W\XINT@sum@D {#5#4#3#2}{#1}%
1324 }%
1325 \def\XINT@sum@D #1%
1326 {\expandafter
1327 \XINT@sum@CC\the\numexpr 1+10#1\relax
1328 }%
1329 \def\xint@sum@cz\W\XINT@sum@D #1#2{ #21000}%

```


12.20 \xintMul

```

1330 \def\xintMul {\romannumeral0\xintmul }%
1331 \def\xintmul #1%
1332 {%
1333     \expandafter\expandafter\expandafter
1334         \xint@mul
1335     \expandafter\expandafter\expandafter
1336         {#1}%
1337 }%
1338 \def\xint@mul #1#2%
1339 {\expandafter\expandafter\expandafter
1340     \XINT@mul@fork #2\Z #1\Z
1341 }%
1342 \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.

1343 \def\XINT@mul@fork #1#2\Z #3#4\Z
1344 {%
1345     \xint@UDzerofork
1346     #1\dummy \XINT@mul@zero
1347     #3\dummy \XINT@mul@zero
1348     0\dummy
1349     {\xint@UDsignsfork
1350         #1#3\dummy \XINT@mul@minusminus          % #1 = #3 = -
1351         #1-\dummy \XINT@mul@minusplus             % #1 = -
1352         #3-\dummy \XINT@mul@plusminus             % #3 = -
1353         --\dummy \XINT@mul@plusplus
1354     \xint@UDkrof }%
1355     \xint@UDkrof
1356     {#2}{#4}#1#3%
1357 }%
1358 \def\XINT@mul@zero #1#2#3#4{ 0}%

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

1359 \def\XINT@mul@minusminus #1#2#3#4%
1360 {%

```

12 Package **xint** implementation

```

1361      \expandafter
1362      \XINT@mul@enter\romannumeral0%
1363      \XINT@RQ {}#2\R\R\R\R\R\R\R\R\Z
1364      \W\X\Y\Z #1\W\X\Y\Z
1365 }%
1366 \def\XINT@mul@minusplus #1#2#3#4%
1367 {%
1368     \expandafter\space\expandafter-%
1369     \romannumeral0\expandafter
1370     \XINT@mul@enter\romannumeral0%
1371     \XINT@RQ {}#4#2\R\R\R\R\R\R\R\R\Z
1372     \W\X\Y\Z #1\W\X\Y\Z
1373 }%
1374 \def\XINT@mul@plusminus #1#2#3#4%
1375 {%
1376     \expandafter\space\expandafter-%
1377     \romannumeral0\expandafter
1378     \XINT@mul@enter\romannumeral0%
1379     \XINT@RQ {}#2\R\R\R\R\R\R\R\R\Z
1380     \W\X\Y\Z #3#1\W\X\Y\Z
1381 }%

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

1382 \def\XINT@mul@plusplus #1#2#3#4%
1383 {%
1384     \expandafter
1385     \XINT@mul@enter\romannumeral0%
1386     \XINT@RQ {}#4#2\R\R\R\R\R\R\R\R\Z
1387     \W\X\Y\Z #3#1\W\X\Y\Z
1388 }%
1389 \def\XINT@mul@add@A #1#2#3#4#5#6%
1390 {%
1391     \xint@w
1392     #3\xint@mul@add@az
1393     \W\XINT@mul@add@AB #1{#3#4#5#6}{#2}%
1394 }%
1395 \def\xint@mul@add@az\W\XINT@mul@add@AB #1#2%
1396 {%
1397     \XINT@mul@add@AC@checkcarry #1%
1398 }%
1399 \def\XINT@mul@add@AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
1400 {%
1401     \XINT@mul@add@ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
1402 }%
1403 \def\XINT@mul@add@ABE #1#2#3#4#5#6%
1404 {\expandafter
1405     \XINT@mul@add@ABEA\the\numexpr #1+10#5#4#3#2+#6\relax.%
1406 }%
1407 \def\XINT@mul@add@ABEA #1#2#3.#4%
1408 {%
1409     \XINT@mul@add@A #2{#3#4}%
1410 }%

```

12 Package *xint* implementation

```

1411 \def\XINT@mul@add@AC@checkcarry #1%
1412 {%
1413   \xint@zero #1\xint@mul@add@AC@nocarry 0\XINT@mul@add@C
1414 }%
1415 \def\xint@mul@add@AC@nocarry 0\XINT@mul@add@C #1#2\W\X\Y\Z
1416 {%
1417   \expandafter
1418   \xint@cleanupzeros@andstop
1419   \romannumeral0%
1420   \XINT@rord@main {}#2%
1421   \xint@UNDEF
1422   \xint@undef\xint@undef\xint@undef\xint@undef
1423   \xint@undef\xint@undef\xint@undef\xint@undef
1424   \xint@UNDEF
1425   #1%
1426 }%
1427 \def\XINT@mul@add@C #1#2#3#4#5%
1428 {%
1429   \xint@w
1430   #5\xint@mul@add@cw
1431   #4\xint@mul@add@cx
1432   #3\xint@mul@add@cy
1433   #2\xint@mul@add@cz
1434   \W\XINT@mul@add@CD {#5#4#3#2}{#1}%
1435 }%
1436 \def\XINT@mul@add@CD #1%
1437 {\expandafter
1438   \XINT@mul@add@CC\the\numexpr 1+10#1\relax.%
1439 }%
1440 \def\XINT@mul@add@CC #1#2#3.#4%
1441 {%
1442   \XINT@mul@add@AC@checkcarry #2{#3#4}%
1443 }%
1444 \def\xint@mul@add@cw
1445   #1\xint@mul@add@cx
1446   #2\xint@mul@add@cy
1447   #3\xint@mul@add@cz
1448   \W\XINT@mul@add@CD
1449 {\expandafter
1450   \XINT@mul@add@CDw\the\numexpr 1+#1#2#3\relax.%
1451 }%
1452 \def\XINT@mul@add@CDw #1.#2#3\X\Y\Z
1453 {%
1454   \XINT@mul@add@end #1#3%
1455 }%
1456 \def\xint@mul@add@cx
1457   #1\xint@mul@add@cy
1458   #2\xint@mul@add@cz
1459   \W\XINT@mul@add@CD
1460 {\expandafter
1461   \XINT@mul@add@CDx\the\numexpr 1+#1#2\relax.%
1462 }%

```

12 Package **xint** implementation

```

1463 \def\XINT@mul@add@CDx #1.#2#3\Y\Z
1464 {%
1465   \XINT@mul@add@end #1#3%
1466 }%
1467 \def\xint@mul@add@cy
1468   #1\xint@mul@add@cz
1469   \W\XINT@mul@add@CD
1470 {\expandafter
1471   \XINT@mul@add@CDy\the\numexpr 1+#1\relax.%
1472 }%
1473 \def\XINT@mul@add@CDy #1.#2#3\Z
1474 {%
1475   \XINT@mul@add@end #1#3%
1476 }%
1477 \def\xint@mul@add@cz\W\XINT@mul@add@CD #1#2#3{\XINT@mul@add@end #1#3}%
1478 \def\XINT@mul@add@end #1#2#3#4#5%
1479 {\expandafter\space
1480   \the\numexpr #1#2#3#4#5\relax
1481 }%
1482 \def\XINT@mul@Ar #1#2#3#4#5#6%
1483 {%
1484   \xint@z #6\xint@mul@br\Z\XINT@mul@Br #1{#6#5#4#3}{#2}%
1485 }%
1486 \def\xint@mul@br\Z\XINT@mul@Br #1#2%
1487 {%
1488   \XINT@sum@AC@checkcarry #1%
1489 }%
1490 \def\XINT@mul@Br #1#2#3#4\W\X\Y\Z #5#6#7#8%
1491 {\expandafter
1492   \XINT@mul@ABEAR\the\numexpr #1+10#2+#8#7#6#5\relax.{#3}#4\W\X\Y\Z
1493 }%
1494 \def\XINT@mul@ABEAR #1#2#3#4#5#6.#7%
1495 {%
1496   \XINT@mul@Ar #2{#7#6#5#4#3}%
1497 }%

```

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

```

1498 \def\XINT@mul@Mr #1%
1499 {%
1500   \expandafter
1501   \XINT@mul@Mr@checkifzeroorone
1502   \expandafter{\the\numexpr #1\relax}%
1503 }%
1504 \def\XINT@mul@Mr@checkifzeroorone #1%
1505 {%
1506   \ifcase #1
1507     \expandafter\XINT@mul@Mr@zero
1508   \or
1509     \expandafter\XINT@mul@Mr@one
1510   \else
1511     \expandafter\XINT@mul@Nr
1512   \fi

```

12 Package *xint* implementation

```

1513 {0000}{#1}%
1514 }%
1515 \def\XINT@mul@Mr@zero #1\Z\Z\Z\Z { 0000}%
1516 \def\XINT@mul@Mr@one #1#2#3#4\Z\Z\Z\Z { #4}%
1517 \def\XINT@mul@Nr #1#2#3#4#5#6#7%
1518 {%
1519 \xint@z #4\xint@mul@pr\Z\XINT@mul@Pr {#1}{#3}{#7#6#5#4}{#2}{#3}%
1520 }%
1521 \def\XINT@mul@Pr #1#2#3%
1522 {\expandafter
1523 \XINT@mul@Lr\the\numexpr 10000#1+#2*#3\relax
1524 }%
1525 \def\XINT@mul@Lr #1#2#3#4#5#6#7#8#9%
1526 {%
1527 \XINT@mul@Nr {#1#2#3#4}{#9#8#7#6#5}%
1528 }%
1529 \def\xint@mul@pr\Z\XINT@mul@Pr #1#2#3#4#5%
1530 {%
1531 \xint@quatrezeros #1\XINT@mul@Mr@end@nocarry 0000\XINT@mul@Mr@end@carry
1532 #1{#4}%
1533 }%
1534 \def\XINT@mul@Mr@end@nocarry 0000\XINT@mul@Mr@end@carry 0000#1{ #1}%
1535 \def\XINT@mul@Mr@end@carry #1#2#3#4#5{ #5#4#3#2#1}%
1536 \def\XINT@mul@M #1%
1537 {\expandafter
1538 \XINT@mul@M@checkifzeroorone
1539 \expandafter{\the\numexpr #1\relax}%
1540 }%
1541 \def\XINT@mul@M@checkifzeroorone #1%
1542 {%
1543 \ifcase #1
1544 \expandafter\XINT@mul@M@zero
1545 \or
1546 \expandafter\XINT@mul@M@one
1547 \else
1548 \expandafter\XINT@mul@N
1549 \fi
1550 {0000}{#1}%
1551 }%
1552 \def\XINT@mul@M@zero #1\Z\Z\Z\Z { 0}%
1553 \def\XINT@mul@M@one #1#2#3#4\Z\Z\Z\Z {%
1554 \expandafter
1555 \xint@cleanupzeros@andstop
1556 \romannumeral0\XINT@rev{#4}%
1557 }%
1558 \def\XINT@mul@N #1#2#3#4#5#6#7%
1559 {%
1560 \xint@z #4\xint@mul@p\Z\XINT@mul@P {#1}{#3}{#7#6#5#4}{#2}{#3}%
1561 }%
1562 \def\XINT@mul@P #1#2#3%
1563 {\expandafter
1564 \XINT@mul@L\the\numexpr 10000#1+#2*#3\relax

```

12 Package *xint* implementation

```

1565 }%
1566 \def\XINT@mul@L #1#2#3#4#5#6#7#8#9%
1567 {%
1568     \XINT@mul@N {#1#2#3#4}{#5#6#7#8#9}%
1569 }%
1570 \def\xint@mul@p\Z\XINT@mul@P #1#2#3#4#5%
1571 {%
1572     \XINT@mul@M@end #1#4%
1573 }%
1574 \def\XINT@mul@M@end #1#2#3#4#5#6#7#8%
1575 {\expandafter\space
1576     \the\numexpr #1#2#3#4#5#6#7#8\relax
1577 }%

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

1578 \def\XINT@mul@enter #1\W\X\Y\Z #2#3#4#5%
1579 {%
1580     \xint@w
1581     #5\xint@mul@enterw
1582     #4\xint@mul@enterx
1583     #3\xint@mul@entery
1584     #2\xint@mul@enterz
1585     \W\XINT@mul@start {#2#3#4#5}#1\W\X\Y\Z
1586 }%
1587 \def\xint@mul@enterw
1588     #1\xint@mul@enterx
1589     #2\xint@mul@entery
1590     #3\xint@mul@enterz
1591     \W\XINT@mul@start #4#5\W\X\Y\Z \X\Y\Z
1592 {%
1593     \XINT@mul@M {#3#2#1}#5\Z\Z\Z\Z
1594 }%
1595 \def\xint@mul@enterx
1596     #1\xint@mul@entery
1597     #2\xint@mul@enterz
1598     \W\XINT@mul@start #3#4\W\X\Y\Z \Y\Z
1599 {%
1600     \XINT@mul@M {#2#1}#4\Z\Z\Z\Z
1601 }%
1602 \def\xint@mul@entery
1603     #1\xint@mul@enterz
1604     \W\XINT@mul@start #2#3\W\X\Y\Z \Z
1605 }%

```

12 Package *xint* implementation

```

1606 \XINT@mul@M {#1}#3\Z\Z\Z\Z
1607 }%
1608 \def\XINT@mul@start #1#2\W\X\Y\Z
1609 {\expandafter
1610 \XINT@mul@main \expandafter
1611 {\romannumeral0\XINT@mul@Mr {#1}#2\Z\Z\Z\Z}#2\W\X\Y\Z
1612 }%
1613 \def\XINT@mul@main #1#2\W\X\Y\Z #3#4#5#6%
1614 {%
1615 \xint@w
1616 #6\xint@mul@mainw
1617 #5\xint@mul@mainx
1618 #4\xint@mul@mainy
1619 #3\xint@mul@mainz
1620 \W\XINT@mul@compute {#1}{#3#4#5#6}#2\W\X\Y\Z
1621 }%
1622 \def\XINT@mul@compute #1#2#3\W\X\Y\Z
1623 {\expandafter
1624 \XINT@mul@main \expandafter
1625 {\romannumeral0\expandafter
1626 \XINT@mul@Ar \expandafter0\expandafter{\expandafter}%
1627 \romannumeral0\XINT@mul@Mr {#2}#3\Z\Z\Z\Z \W\X\Y\Z 0000#1\W\X\Y\Z
1628 }#3\W\X\Y\Z
1629 }%
1630 \def\xint@mul@mainw
1631 #1\xint@mul@mainx
1632 #2\xint@mul@mainy
1633 #3\xint@mul@mainz
1634 \W\XINT@mul@compute #4#5#6\W\X\Y\Z \X\Y\Z
1635 {%
1636 \expandafter
1637 \XINT@mul@add@A \expandafter0\expandafter{\expandafter}%
1638 \romannumeral0%
1639 \XINT@mul@Mr {#3#2#1}#6\Z\Z\Z\Z
1640 \W\X\Y\Z 000#4\W\X\Y\Z
1641 }%
1642 \def\xint@mul@mainx
1643 #1\xint@mul@mainy
1644 #2\xint@mul@mainz
1645 \W\XINT@mul@compute #3#4#5\W\X\Y\Z \Y\Z
1646 {%
1647 \expandafter
1648 \XINT@mul@add@A \expandafter0\expandafter{\expandafter}%
1649 \romannumeral0%
1650 \XINT@mul@Mr {#2#1}#5\Z\Z\Z\Z
1651 \W\X\Y\Z 00#3\W\X\Y\Z
1652 }%
1653 \def\xint@mul@mainy
1654 #1\xint@mul@mainz
1655 \W\XINT@mul@compute #2#3#4\W\X\Y\Z \Z
1656 {%
1657 \expandafter

```

```

1658 \XINT@mul@add@A \expandafter\expandafter\expandafter}%
1659 \romannumeral0%
1660 \XINT@mul@Mr {#1}#4\Z\Z\Z\Z
1661 \W\X\Y\Z 0#2\W\X\Y\Z
1662 }%
1663 \def\xint@mul@mainz\W\XINT@mul@compute #1#2#3\W\X\Y\Z
1664 {%
1665 \expandafter
1666 \xint@cleanupzeros@andstop\romannumeral0\XINT@rev{#1}%
1667 }%

```

12.21 \xintSqr

```

1668 \def\xintSqr {\romannumeral0\xintsqr }%
1669 \def\xintsqr #1%
1670 {%
1671 \expandafter\expandafter\expandafter
1672 \XINT@sqr
1673 \expandafter\expandafter\expandafter
1674 {\xintAbs{#1}}% fait l'expansion de #1 et se d'ebarrasse du signe
1675 }%
1676 \def\XINT@sqr #1%
1677 {\expandafter
1678 \XINT@mul@enter
1679 \romannumeral0%
1680 \XINT@RQ {}#1\R\R\R\R\R\R\R\R\Z
1681 \W\X\Y\Z #1\W\X\Y\Z
1682 }%

```

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.

```

1683 \def\XINT@posprod #1%
1684 {%
1685 \XINT@pprod@checkifempty #1\Z
1686 }%
1687 \def\XINT@pprod@checkifempty #1%
1688 {%
1689 \xint@relax #1\XINT@pprod@emptyproduct\relax

```



```

1690 \XINT@pprod@first #1%
1691 }%
1692 \def\XINT@pprod@emptyproduct #1\Z { 1}%
1693 \def\XINT@pprod@first #1\Z
1694 {%
1695 \XINT@pprod@getnext {#1}%
1696 }%
1697 \def\XINT@pprod@getnext #1#2%
1698 {%
1699 \XINT@pprod@checkiffinished #2\Z {#1}%
1700 }%
1701 \def\XINT@pprod@checkiffinished #1%
1702 {%
1703 \xint@relax #1\XINT@pprod@end\relax
1704 \XINT@pprod@RQnew #1%
1705 }%
1706 \def\XINT@pprod@RQnew #1\Z
1707 {%
1708 \expandafter\XINT@pprod@compute
1709 \expandafter
1710 {\romannumeral0\XINT@RQ {#1\R\R\R\R\R\R\R\R\Z }%
1711 }%
1712 \def\XINT@pprod@compute #1#2%
1713 {%
1714 \expandafter
1715 \XINT@pprod@getnext
1716 \expandafter
1717 {\romannumeral0\XINT@mul@enter #1\W\X\Y\Z #2\W\X\Y\Z }%
1718 }%
1719 \def\XINT@pprod@end\relax\XINT@pprod@RQnew #1\Z #2{ #2}%
1720 \def\xintProductExpr {\romannumeral0\xintproductexpr }%
1721 \def\xintPrd {\romannumeral0\xintprd }%
1722 \def\xintprd #1%
1723 {%
1724 \expandafter\expandafter\expandafter
1725 \xintproductexpr #1\relax
1726 }%
1727 \def\xintproductexpr #1%
1728 {%
1729 \expandafter\expandafter\expandafter
1730 \XINT@prod@checkifempty #1\Z
1731 }%
1732 \def\XINT@prod@checkifempty #1%
1733 {%
1734 \xint@relax #1\XINT@prod@emptyproduct\relax
1735 \XINT@prod@checkfirstsign #1%
1736 }%
1737 \def\XINT@prod@emptyproduct #1\Z { 1}%
1738 \def\XINT@prod@checkfirstsign #1%
1739 {%
1740 \xint@zero #1\XINT@prod@returnzero0%
1741 \xint@UDsignfork

```

12 Package *xint* implementation

```

1742      #1\dummy \XINT@prod@firstisneg
1743      -\dummy \XINT@prod@firstispos
1744      \xint@UDkrof
1745      #1%
1746 }%
1747 \def\XINT@prod@returnzero #1\relax { 0}%
1748 \def\XINT@prod@firstisneg #1#2\Z
1749 {%
1750     \XINT@prod@xpxpnext 0{#2}%
1751 }%
1752 \def\XINT@prod@firstispos #1\Z
1753 {%
1754     \XINT@prod@xpxpnext 1{#1}%
1755 }%
1756 \def\XINT@prod@xpxpnext #1#2#3%
1757 {%
1758     \expandafter\expandafter\expandafter
1759     \XINT@prod@checkiffinished #3\Z {#2}#1%
1760 }%
1761 \def\XINT@prod@checkiffinished #1%
1762 {%
1763     \xint@relax #1\XINT@prod@end\relax
1764     \XINT@prod@checksign #1%
1765 }%
1766 \def\XINT@prod@checksign #1%
1767 {%
1768     \xint@zero #1\XINT@prod@returnzero0%
1769     \xint@UDsignfork
1770     #1\dummy \XINT@prod@neg@RQnew
1771     -\dummy \XINT@prod@pos@RQnew
1772     \xint@UDkrof
1773     #1%
1774 }%
1775 \def\XINT@prod@pos@RQnew #1\Z
1776 {%
1777     \expandafter
1778     \XINT@prod@pos
1779     \expandafter
1780     {\romannumeral0\XINT@RQ {#1\R\R\R\R\R\R\R\R\Z }%
1781 }%
1782 \def\XINT@prod@pos #1#2#3%
1783 {%
1784     \expandafter
1785     \XINT@prod@xpxpnext
1786     \expandafter
1787     #3%
1788     \expandafter
1789     {\romannumeral0\XINT@mul@enter #1\W\X\Y\Z #2\W\X\Y\Z }%
1790 }%
1791 \def\XINT@prod@neg@RQnew #1#2\Z
1792 {%
1793     \expandafter

```

```

1794 \XINT@prod@neg
1795 \expandafter
1796 {\romannumeral0\XINT@RQ {}#2\R\R\R\R\R\R\R\R\Z }%
1797 }%
1798 \def\XINT@prod@neg #1#2#3%
1799 {%
1800 \expandafter
1801 \XINT@prod@xpxpnext
1802 \expandafter
1803 {\the\numexpr 1-#3\expandafter}%
1804 \expandafter
1805 {\romannumeral0\XINT@mul@enter #1\W\X\Y\Z #2\W\X\Y\Z }%
1806 }%
1807 \def\XINT@prod@end\relax\XINT@prod@checksign #1\Z #2#3%
1808 {%
1809 \xint@prod@cleanupzeros #3#2%
1810 }%
1811 \def\xint@prod@cleanupzeros #1#2#3#4#5%
1812 {%
1813 \expandafter\space\the\numexpr (2*#1-1)*#2#3#4#5\relax
1814 }%

```

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}{10000000000}
rather than \ifnum\XINT@Length {#1}>9 but for the time being I leave things
as they stand.

```

1815 \def\xintFac {\romannumeral0\xintfac }%
1816 \def\xintfac #1%
1817 {%
1818 \expandafter\expandafter\expandafter
1819 \XINT@fac@fork
1820 \expandafter\expandafter\expandafter
1821 {#1}%
1822 }%
1823 \def\XINT@Fac {\romannumeral0\XINT@fac@fork }%
1824 \def\XINT@fac@fork #1%
1825 {%
1826 \ifcase\xintSgn {#1}
1827 \xint@afterfi{\expandafter\space\expandafter 1\xint@gobble }%
1828 \or
1829 \expandafter\XINT@fac@checklength
1830 \else
1831 \xint@afterfi{\xintError:FactorialOfNegativeNumber
1832 \expandafter\space\expandafter 1\xint@gobble }%
1833 \fi
1834 {#1}%
1835 }%
1836 \def\XINT@fac@checklength #1%
1837 {%
1838 \ifnum \XINT@Length {#1}> 9

```

12 Package *xint* implementation

```

1839      \xint@afterfi{\xintError:FactorialOfTooBigNumber
1840                  \expandafter\space\expandafter 1\xint@gobble }%
1841  \else
1842      \xint@afterfi{\ifnum #1>9999
1843                  \expandafter\XINT@fac@big@loop
1844                  \else
1845                  \expandafter\XINT@fac@loop
1846                  \fi }%
1847  \fi
1848  {#1}%
1849 }%
1850 \def\XINT@fac@big@loop #1{\XINT@fac@big@loop@main {10000}{#1}{}}%
1851 \def\XINT@fac@big@loop@main #1#2#3%
1852 {%
1853     \ifnum #1<#2
1854         \expandafter
1855             \XINT@fac@big@loop@main
1856         \expandafter
1857             {\the\numexpr #1+1\expandafter }%
1858     \else
1859         \expandafter\XINT@fac@big@docomputation
1860     \fi
1861     {#2}{#3{#1}}%
1862 }%
1863 \def\XINT@fac@big@docomputation #1#2%
1864 {%
1865     \expandafter
1866         \XINT@pprod@getnext
1867     \expandafter
1868     {\romannumeral0\XINT@fac@loop {9999}}#2\relax
1869 }%
1870 \def\XINT@fac@loop #1{\XINT@fac@loop@main 1{#1}{}}%
1871 \def\XINT@fac@loop@main #1#2#3%
1872 {%
1873     \ifnum #1<#2
1874         \expandafter
1875             \XINT@fac@loop@main
1876         \expandafter
1877             {\the\numexpr #1+1\expandafter }%
1878     \else
1879         \expandafter\XINT@fac@docomputation
1880     \fi
1881     {#2}{#3{#1}}%
1882 }%
1883 \def\XINT@fac@docomputation #1#2%
1884 {%
1885     \XINT@fprod@getnext {1000}#2\relax
1886 }%
1887 \def\XINT@fprod@getnext #1#2%
1888 {%
1889     \XINT@fprod@checkiffinished #2\Z {#1}%
1890 }%

```

```

1891 \def\XINT@fprod@checkiffinished #1%
1892 {%
1893     \xint@relax #1\XINT@fprod@end\relax
1894     \XINT@fprod@compute #1%
1895 }%
1896 \def\XINT@fprod@compute #1\Z #2%
1897 {%
1898     \expandafter
1899         \XINT@fprod@getnext
1900     \expandafter
1901     {\romannumeral0\XINT@mul@Mr {#1}#2\Z\Z\Z\Z }%
1902 }%
1903 \def\XINT@fprod@end\relax\XINT@fprod@compute #1\Z #2%
1904 {%
1905     \expandafter
1906     \xint@cleanupzeros@andstop
1907     \romannumeral0\XINT@rev {#2}%
1908 }%

```

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.

```

1909 \def\xintPow {\romannumeral0\xintpow }%
1910 \def\xintpow #1%
1911 {%
1912     \expandafter\expandafter\expandafter
1913         \xint@pow
1914         #1\Z%
1915 }%

    #1#2 = A

1916 \def\xint@pow #1#2\Z
1917 {%
1918     \xint@UDsignfork
1919     #1\dummy \XINT@pow@Aneg
1920     -\dummy \XINT@pow@Anonneg
1921     \xint@UDkrof
1922     #1{#2}%
1923 }%
1924 \def\XINT@pow@Aneg #1#2#3%
1925 {%
1926     \expandafter\expandafter\expandafter
1927         \XINT@pow@Aneg@
1928     \expandafter\expandafter\expandafter
1929     {#3}{#2}%
1930 }%

    B = #1, xpxp déjà fait

```

12 Package *xint* implementation

```

1931 \def\XINT@pow@Aneg@ #1%
1932 {%
1933   \ifcase\XINT@Odd{#1}
1934   \or \expandafter\XINT@pow@Aneg@Bodd
1935   \fi
1936   \XINT@pow@Anonneg@ {#1}%
1937 }%
1938 \def\XINT@pow@Aneg@Bodd #1%
1939 {%
1940   \expandafter\XINT@opp\romannumeral0\XINT@pow@Anonneg@
1941 }%

    B = #3, faire le xpxp

1942 \def\XINT@pow@Anonneg #1#2#3%
1943 {%
1944   \expandafter\expandafter\expandafter
1945   \XINT@pow@Anonneg@
1946   \expandafter\expandafter\expandafter
1947   {#3}{#1#2}%
1948 }%

    #1 = B, #2 = |A|

1949 \def\XINT@pow@Anonneg@ #1#2%
1950 {%
1951   \ifcase\XINT@Cmp {#2}{1}
1952   \expandafter\XINT@pow@AisOne
1953   \or
1954   \expandafter\XINT@pow@AatleastTwo
1955   \else
1956   \expandafter\XINT@pow@AisZero
1957   \fi
1958   {#1}{#2}%
1959 }%
1960 \def\XINT@pow@AisOne #1#2{ 1}%

    #1 = B

1961 \def\XINT@pow@AisZero #1#2%
1962 {%
1963   \ifcase\XINT@Sgn {#1}
1964   \xint@afterfi { 1}%
1965   \or
1966   \xint@afterfi { 0}%
1967   \else
1968   \xint@afterfi {\xintError:DivisionByZero\space 0}%
1969   \fi
1970 }%
1971 \def\XINT@pow@AatleastTwo #1%
1972 {%
1973   \ifcase\XINT@Sgn {#1}
1974   \expandafter\XINT@pow@BisZero
1975   \or

```

12 Package *xint* implementation

```

1976     \expandafter\XINT@pow@checkBlength
1977   \else
1978     \expandafter\XINT@pow@BisNegative
1979   \fi
1980   {#1}%
1981 }%
1982 \def\XINT@pow@BisNegative #1#2{\xintError:FractionRoundedToZero\space 0}%
1983 \def\XINT@pow@BisZero #1#2{ 1}%

    B = #1 > 0, A = #2 > 1
1984 \def\XINT@pow@checkBlength #1#2%
1985 {%
1986   \ifnum\xintLen{#1} >9
1987     \expandafter\XINT@pow@BtooBig
1988   \else
1989     \expandafter\XINT@pow@loop
1990   \fi
1991   {#1}{#2}\XINT@pow@posprod
1992   \xint@UNDEF
1993   \xint@undef\xint@undef\xint@undef\xint@undef
1994   \xint@undef\xint@undef\xint@undef\xint@undef
1995   \xint@UNDEF
1996 }%
1997 \def\XINT@pow@BtooBig #1\xint@UNDEF #2\xint@UNDEF
1998     {\xintError:ExponentTooBig\space 0}%
1999 \def\XINT@pow@loop #1#2%
2000 {%
2001   \ifnum #1 = 1
2002     \expandafter\XINT@pow@loop@end
2003   \else
2004     \xint@afterfi{\expandafter\XINT@pow@loop@a
2005       \expandafter{\the\numexpr 2*(#1/2)-#1\expandafter}% b mod 2
2006       \expandafter{\the\numexpr #1-#1/2\expandafter}% [b/2]
2007       \expandafter{\romannumeral0\xintsqr{#2}}}%
2008   \fi
2009   {{#2}}%
2010 }%
2011 \def\XINT@pow@loop@end {\romannumeral0\XINT@rord@main {} \relax}%
2012 \def\XINT@pow@loop@a #1%
2013 {%
2014   \ifnum #1 = 1
2015     \expandafter\XINT@pow@loop
2016   \else
2017     \expandafter\XINT@pow@loop@throwaway
2018   \fi
2019 }%
2020 \def\XINT@pow@loop@throwaway #1#2#3%
2021 {%
2022   \XINT@pow@loop {#1}{#2}%
2023 }%

```

Routine de produit servant pour le calcul des puissances.

12 Package **xint** implementation

```

2024 \def\XINT@pow@posprod #1%
2025 {%
2026     \XINT@pow@pprod@checkifempty #1\Z
2027 }%
2028 \def\XINT@pow@pprod@checkifempty #1%
2029 {%
2030     \xint@relax #1\XINT@pow@pprod@emptyproduct\relax
2031     \XINT@pow@pprod@RQfirst #1%
2032 }%
2033 \def\XINT@pow@pprod@emptyproduct #1\Z { 1}%
2034 \def\XINT@pow@pprod@RQfirst #1\Z
2035 {%
2036     \expandafter\XINT@pow@pprod@getnext\expandafter
2037     {\romannumeral0\XINT@RQ { }#1\R\R\R\R\R\R\R\R\Z}%
2038 }%
2039 \def\XINT@pow@pprod@getnext #1#2%
2040 {%
2041     \XINT@pow@pprod@checkiffinished #2\Z {#1}%
2042 }%
2043 \def\XINT@pow@pprod@checkiffinished #1%
2044 {%
2045     \xint@relax #1\XINT@pow@pprod@end\relax
2046     \XINT@pow@pprod@compute #1%
2047 }%
2048 \def\XINT@pow@pprod@compute #1\Z #2%
2049 {%
2050     \expandafter
2051         \XINT@pow@pprod@getnext
2052     \expandafter
2053     {\romannumeral0\XINT@pow@mul@enter #2\W\X\Y\Z #1\W\X\Y\Z}%
2054 }%
2055 \def\XINT@pow@pprod@end\relax\XINT@pow@pprod@compute #1\Z #2%
2056 {%
2057     \expandafter
2058     \xint@cleanupzeros@andstop
2059     \romannumeral0\XINT@rev {#2}%
2060 }%

```

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

```

2061 \def\XINT@pow@mul@enter #1\W\X\Y\Z #2#3#4#5%
2062 {%
2063     \xint@w
2064     #5\xint@pow@mul@enterw
2065     #4\xint@pow@mul@enterx
2066     #3\xint@pow@mul@entery
2067     #2\xint@pow@mul@enterz
2068     \W\XINT@pow@mul@start {#2#3#4#5}#1\W\X\Y\Z
2069 }%
2070 \def\xint@pow@mul@enterw
2071     #1\xint@pow@mul@enterx
2072     #2\xint@pow@mul@entery

```


12 Package *xint* implementation

```

2073 #3\xint@pow@mul@enterz
2074 \W\XINT@pow@mul@start #4#5\W\X\Y\Z \X\Y\Z
2075 {%
2076 \XINT@mul@Mr {#3#2#1}#5\Z\Z\Z\Z
2077}%
2078 \def\xint@pow@mul@enterx
2079 #1\xint@pow@mul@entery
2080 #2\xint@pow@mul@enterz
2081 \W\XINT@pow@mul@start #3#4\W\X\Y\Z \Y\Z
2082 {%
2083 \XINT@mul@Mr {#2#1}#4\Z\Z\Z\Z
2084}%
2085 \def\xint@pow@mul@entery
2086 #1\xint@pow@mul@enterz
2087 \W\XINT@pow@mul@start #2#3\W\X\Y\Z \Z
2088 {%
2089 \XINT@mul@Mr {#1}#3\Z\Z\Z\Z
2090}%
2091 \def\XINT@pow@mul@start #1#2\W\X\Y\Z
2092 {\expandafter
2093 \XINT@pow@mul@main \expandafter
2094 {\romannumeral0%
2095 \XINT@mul@Mr {#1}#2\Z\Z\Z\Z
2096 }#2\W\X\Y\Z
2097}%
2098 \def\XINT@pow@mul@main #1#2\W\X\Y\Z #3#4#5#6%
2099 {%
2100 \xint@w
2101 #6\xint@pow@mul@mainw
2102 #5\xint@pow@mul@mainx
2103 #4\xint@pow@mul@mainy
2104 #3\xint@pow@mul@mainz
2105 \W\XINT@pow@mul@compute {#1}{#3#4#5#6}#2\W\X\Y\Z
2106}%
2107 \def\XINT@pow@mul@compute #1#2#3\W\X\Y\Z
2108 {\expandafter
2109 \XINT@pow@mul@main \expandafter
2110 {\romannumeral0\expandafter
2111 \XINT@mul@Ar \expandafter0\expandafter{\expandafter}%
2112 \romannumeral0\XINT@mul@Mr {#2}#3\Z\Z\Z\Z \W\X\Y\Z 0000#1\W\X\Y\Z
2113 }#3\W\X\Y\Z
2114}%
2115 \def\xint@pow@mul@mainw
2116 #1\xint@pow@mul@mainx
2117 #2\xint@pow@mul@mainy
2118 #3\xint@pow@mul@mainz
2119 \W\XINT@pow@mul@compute #4#5#6\W\X\Y\Z \X\Y\Z
2120 {%
2121 \expandafter
2122 \XINT@pow@add@A \expandafter0\expandafter{\expandafter}%
2123 \romannumeral0%
2124 \XINT@mul@Mr {#3#2#1}#6\Z\Z\Z\Z

```

12 Package *xint* implementation

```

2125                                \W\X\Y\Z 000#4\W\X\Y\Z
2126 }%
2127 \def\xint@pow@mul@mainx
2128     #1\xint@pow@mul@mainy
2129     #2\xint@pow@mul@mainz
2130     \W\XINT@pow@mul@compute #3#4#5\W\X\Y\Z \Y\Z
2131 {%
2132     \expandafter
2133     \XINT@pow@add@A \expandafter0\expandafter{\expandafter}%
2134         \romannumeral0%
2135     \XINT@mul@Mr    {#2#1}#5\Z\Z\Z\Z
2136                     \W\X\Y\Z 00#3\W\X\Y\Z
2137 }%
2138 \def\xint@pow@mul@mainy
2139     #1\xint@pow@mul@mainz
2140     \W\XINT@pow@mul@compute #2#3#4\W\X\Y\Z \Z
2141 {%
2142     \expandafter
2143     \XINT@pow@add@A \expandafter0\expandafter{\expandafter}%
2144         \romannumeral0%
2145     \XINT@mul@Mr    {#1}#4\Z\Z\Z\Z
2146                     \W\X\Y\Z 0#2\W\X\Y\Z
2147 }%
2148 \def\xint@pow@mul@mainz\W\XINT@pow@mul@compute #1#2#3\W\X\Y\Z
2149 { #1}%

```

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

```

2150 \def\XINT@pow@add@A #1#2#3#4#5#6%
2151 {%
2152     \xint@w
2153     #3\xint@pow@add@az
2154     \W\XINT@pow@add@AB #1{#3#4#5#6}{#2}%
2155 }%
2156 \def\xint@pow@add@az\W\XINT@pow@add@AB #1#2%
2157 {%
2158     \XINT@pow@add@AC@checkcarry #1%
2159 }%
2160 \def\XINT@pow@add@AC@checkcarry #1%
2161 {%
2162     \xint@zero #1\xint@pow@add@AC@nocarry 0\XINT@pow@add@C
2163 }%
2164 \def\xint@pow@add@AC@nocarry 0\XINT@pow@add@C
2165 {%
2166     \XINT@pow@add@F
2167 }%
2168 \def\XINT@pow@add@AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
2169 {%
2170     \XINT@pow@add@ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
2171 }%
2172 \def\XINT@pow@add@ABE #1#2#3#4#5#6%
2173 {\expandafter

```

12 Package **xint** implementation

```

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}%

```

12 Package **xint** implementation

```

2226 \def\xint@pow@add@cy
2227     #1\xint@pow@add@cz
2228     \W\XINT@pow@add@CD
2229 {\expandafter
2230     \XINT@pow@add@CDy\the\numexpr 1+1000#1\relax
2231 }%
2232 \def\XINT@pow@add@CDy #1#2#3#4#5#6%
2233 {%
2234     \xint@quatrezeros #2#3#4#5\XINT@pow@add@endDy@zeros
2235     0000\XINT@pow@add@endDy #2#3#4#5%
2236 }%
2237 \def\XINT@pow@add@endDy@zeros 0000\XINT@pow@add@endDy 0000#1\Z{ #1}%
2238 \def\XINT@pow@add@endDy #1#2#3#4#5\Z{ #5#4#3#2#1}%
2239 \def\xint@pow@add@cz\W\XINT@pow@add@CD #1#2{ #21000}%
2240 \def\XINT@pow@add@F #1#2#3#4#5%
2241 {%
2242     \xint@w
2243     #5\xint@pow@add@Gw
2244     #4\xint@pow@add@Gx
2245     #3\xint@pow@add@Gy
2246     #2\xint@pow@add@Gz
2247     \W\XINT@pow@add@G {#2#3#4#5}{#1}%
2248 }%
2249 \def\XINT@pow@add@G #1#2%
2250 {%
2251     \XINT@pow@add@F {#2#1}%
2252 }%
2253 \def\xint@pow@add@Gw
2254     #1\xint@pow@add@Gx
2255     #2\xint@pow@add@Gy
2256     #3\xint@pow@add@Gz
2257     \W\XINT@pow@add@G #4%
2258 {%
2259     \xint@quatrezeros #3#2#10\XINT@pow@add@endGw@zeros
2260     0000\XINT@pow@add@endGw #3#2#10%
2261 }%
2262 \def\XINT@pow@add@endGw@zeros 0000\XINT@pow@add@endGw 0000#1\X\Y\Z{ #1}%
2263 \def\XINT@pow@add@endGw #1#2#3#4#5\X\Y\Z{ #5#1#2#3#4}%
2264 \def\xint@pow@add@Gx
2265     #1\xint@pow@add@Gy
2266     #2\xint@pow@add@Gz
2267     \W\XINT@pow@add@G #3%
2268 {%
2269     \xint@quatrezeros #2#100\XINT@pow@add@endGx@zeros
2270     0000\XINT@pow@add@endGx #2#100%
2271 }%
2272 \def\XINT@pow@add@endGx@zeros 0000\XINT@pow@add@endGx 0000#1\Y\Z{ #1}%
2273 \def\XINT@pow@add@endGx #1#2#3#4#5\Y\Z{ #5#1#2#3#4}%
2274 \def\xint@pow@add@Gy
2275     #1\xint@pow@add@Gz
2276     \W\XINT@pow@add@G #2%
2277 {%

```

```

2278 \xint@quatrezeros #1000\XINT@pow@add@endGy@zeros
2279 0000\XINT@pow@add@endGy #1000%
2280 }%
2281 \def\XINT@pow@add@endGy@zeros 0000\XINT@pow@add@endGy 0000#1\Z{ #1}%
2282 \def\XINT@pow@add@endGy #1#2#3#4#5\Z{ #5#1#2#3#4}%
2283 \def\xint@pow@add@Gz\W\XINT@pow@add@G #1#2{ #2}%

```

12.25 \xintDivision, \xintQuo, \xintRem

```

2284 \def\xintQuo {\romannumeral0\xintquo }%
2285 \def\xintRem {\romannumeral0\xintrem }%
2286 \def\xintquo {\expandafter
2287 \xint@firstoftwo@andstop
2288 \romannumeral0\xintdivision }%
2289 \def\xintrem {\expandafter
2290 \xint@secondoftwo@andstop
2291 \romannumeral0\xintdivision }%
#1 = A, #2 = B. On calcule le quotient de A par B

2292 \def\xintDivision {\romannumeral0\xintdivision }%
2293 \def\xintdivision #1%
2294 {%
2295 \expandafter\expandafter\expandafter
2296 \xint@division
2297 \expandafter\expandafter\expandafter
2298 {#1}%
2299 }%
2300 \def\xint@division #1#2%
2301 {%
2302 \expandafter\expandafter\expandafter
2303 \XINT@div@fork #2\Z #1\Z
2304 }%
2305 \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

2306 \def\XINT@div@fork #1#2\Z #3#4\Z
2307 {%
2308 \xint@UDzerofork
2309 #1\dummy \XINT@div@BisZero
2310 #3\dummy \XINT@div@AisZero
2311 0\dummy
2312 {\xint@UDsignfork
2313 #1\dummy \XINT@div@BisNegative % B < 0
2314 #3\dummy \XINT@div@AisNegative % A < 0, B > 0
2315 -\dummy \XINT@div@plusplus % B > 0, A > 0
2316 \xint@UDkrof }%
2317 \xint@UDkrof
2318 {#2}{#4}#1#3% #1#2=B, #3#4=A
2319 }%
2320 \def\XINT@div@BisZero #1#2#3#4%
2321 {\xintError:DivisionByZero\space {0}{0}}%
2322 \def\XINT@div@AisZero #1#2#3#4{ {0}{0}}%

```

12 Package **xint** implementation

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

```

2323 \def\XINT@div@plusplus #1#2#3#4%
2324 {%
2325   \XINT@div@prepare {#3#1}{#4#2}%
2326 }%

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

2327 \def\XINT@div@BisNegative #1#2#3#4%
2328 {%
2329   \expandafter\XINT@div@BisNegative@post
2330   \romannumeral0\XINT@div@fork #1\Z #4#2\Z
2331 }%
2332 \def\XINT@div@BisNegative@post #1#2%
2333 {%
2334   \expandafter\space\expandafter
2335   {\romannumeral0\XINT@opp #1}{#2}%
2336 }%

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

2337 \def\XINT@div@AisNegative #1#2#3#4%
2338 {%
2339   \expandafter\XINT@div@AisNegative@post
2340   \romannumeral0\XINT@div@prepare {#3#1}{#2}{#3#1}%
2341 }%
2342 \def\XINT@div@AisNegative@post #1#2%
2343 {%
2344   \ifcase\xintSgn {#2}
2345     \expandafter \XINT@div@AisNegative@zerorem
2346   \or
2347     \expandafter \XINT@div@AisNegative@posrem
2348   \fi
2349   {#1}{#2}%
2350 }%

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

2351 \def\XINT@div@AisNegative@zerorem #1#2#3%
2352 {%
2353   \expandafter\space\expandafter
2354   {\romannumeral0\XINT@opp #1}{0}%
2355 }%

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

```

12 Package *xint* implementation

```

2356 \def\XINT@div@AisNegative@posrem #1%
2357 {%
2358     \expandafter
2359         \XINT@div@AisNegative@posrem@b
2360     \expandafter
2361         {\romannumeral0\xintopp {\XINT@Add{#1}{1}}}%
2362 }%

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

2363 \def\XINT@div@AisNegative@posrem@b #1#2#3%
2364 {%
2365     \expandafter
2366         \xint@exchangetwo@keepbraces@andstop
2367     \expandafter
2368         {\romannumeral0\XINT@sub {#3}{#2}}{#1}%
2369 }%

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

2370 \def\XINT@div@prepare #1%
2371 {%
2372     \expandafter
2373         \XINT@div@prepareB@a
2374     \expandafter
2375         {\romannumeral0\XINT@length {#1}}{#1}% B  $> 0$  ici
2376 }%

    Calcul du plus petit  $K = 4n \geq$  longueur de B

2377 \def\XINT@div@prepareB@a #1%
2378 {%
2379     \expandafter\XINT@div@prepareB@b\expandafter
2380     {\the\numexpr 4*((#1+1)/4)\relax}{#1}%
2381 }%

    #1 = K

2382 \def\XINT@div@prepareB@b #1#2%
2383 {%
2384     \expandafter\XINT@div@prepareB@c \expandafter
2385     {\the\numexpr #1-#2\relax}{#1}%
2386 }%

    #1 = c

2387 \def\XINT@div@prepareB@c #1%
2388 {%
2389     \ifcase #1
2390         \expandafter\XINT@div@prepareB@di
2391     \or \expandafter\XINT@div@prepareB@dii
2392     \or \expandafter\XINT@div@prepareB@diii
2393     \else \expandafter\XINT@div@prepareB@div

```

12 Package *xint* implementation

```

2394 \fi
2395 }%
2396 \def\XINT@div@prepareB@di {\XINT@div@prepareB@e {}{0}}%
2397 \def\XINT@div@prepareB@dii {\XINT@div@prepareB@e {0}{1}}%
2398 \def\XINT@div@prepareB@diii {\XINT@div@prepareB@e {00}{2}}%
2399 \def\XINT@div@prepareB@div {\XINT@div@prepareB@e {000}{3}}%

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

2400 \def\XINT@div@prepareB@e #1#2#3#4%
2401 {%
2402 \XINT@div@prepareB@f #4#1\Z {#3}{#2}{#1}%
2403 }%

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.

2404 \def\XINT@div@prepareB@f #1#2#3#4#5\Z
2405 {%
2406 \expandafter
2407 \XINT@div@prepareB@g
2408 \expandafter
2409 {\romannumeral0\XINT@rev {#1#2#3#4#5}}{#1#2#3#4}%
2410 }%

#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

2411 \def\XINT@div@prepareB@g #1#2#3#4#5#6%
2412 {%
2413 \XINT@div@prepareA@a {#6#5}{#2}{#3}{#1}{#4}%
2414 }%

A, x, K, B, c,

2415 \def\XINT@div@prepareA@a #1%
2416 {%
2417 \expandafter
2418 \XINT@div@prepareA@b
2419 \expandafter
2420 {\romannumeral0\XINT@length {#1}}{#1}% A >0 ici
2421 }%

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

2422 \def\XINT@div@prepareA@b #1%
2423 {%
2424 \expandafter\XINT@div@prepareA@c\expandafter
2425 {\the\numexpr 4*((#1+1)/4)\relax}{#1}%
2426 }%

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

```


12 Package *xint* implementation

```

2427 \def\XINT@div@prepareA@c #1#2%
2428 {%
2429     \expandafter\XINT@div@prepareA@d \expandafter
2430     {\the\numexpr #1-#2\relax}{#1}%
2431 }%
2432 \def\XINT@div@prepareA@d #1%
2433 {%
2434     \ifcase #1
2435         \expandafter\XINT@div@prepareA@di
2436     \or \expandafter\XINT@div@prepareA@dii
2437     \or \expandafter\XINT@div@prepareA@diii
2438     \else \expandafter\XINT@div@prepareA@div
2439     \fi
2440 }%
2441 \def\XINT@div@prepareA@di {\XINT@div@prepareA@e {}}%
2442 \def\XINT@div@prepareA@dii {\XINT@div@prepareA@e {0}}%
2443 \def\XINT@div@prepareA@diii {\XINT@div@prepareA@e {00}}%
2444 \def\XINT@div@prepareA@div {\XINT@div@prepareA@e {000}}%

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

2445 \def\XINT@div@prepareA@e #1#2#3%
2446 {%
2447     \XINT@div@startswitch {#1#3}{#2}%
2448 }%

    A, L, x, K, B, c

2449 \def\XINT@div@startswitch #1#2#3#4%
2450 {%
2451     \ifnum #2 > #4
2452         \expandafter\XINT@div@body@a
2453     \else
2454         \ifnum #2 = #4
2455             \expandafter\expandafter\expandafter
2456             \XINT@div@final@a
2457         \else
2458             \expandafter\expandafter\expandafter
2459             \XINT@div@finished@a
2460         \fi\fi {#1}{#4}{#3}{0000}{#2}%
2461 }%

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

2462 \def\XINT@div@finished@a #1#2#3%
2463 {%
2464     \expandafter
2465     \XINT@div@finished@b
2466     \expandafter
2467     {\romannumeral0\XINT@cuz {#1}}%
2468 }%

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

```

12 Package **xint** implementation

```

2469 \def\XINT@div@finished@b #1#2#3#4#5%
2470 {%
2471   \ifcase \XINT@Sgn {#1}
2472     \xint@afterfi {\XINT@div@finished@c {0}}%
2473   \or
2474     \xint@afterfi {\expandafter\XINT@div@finished@c
2475                   \expandafter
2476                     {\romannumeral0\XINT@dsh@checksignx #5\Z {#1}}}%
2477   \fi
2478   {#2}%
2479 }%

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

2480 \def\XINT@div@finished@c #1#2%
2481 {%
2482   \expandafter
2483     \space
2484   \expandafter
2485     {\romannumeral0\expandafter\xint@cleanupzeros@andstop
2486     \romannumeral0\XINT@rev {#2}}{#1}%
2487 }%

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

2488 \def\XINT@div@final@a #1%
2489 {%
2490   \XINT@div@final@b #1\Z
2491 }%
2492 \def\XINT@div@final@b #1#2#3#4#5\Z
2493 {%
2494   \xint@quatrezeros #1#2#3#4\xint@div@final@c0000%
2495   \XINT@div@final@c {#1#2#3#4}{#1#2#3#4#5}%
2496 }%
2497 \def\xint@div@final@c0000\XINT@div@final@c #1%
2498   {\XINT@div@finished@a }%

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

2499 \def\XINT@div@final@c #1#2#3#4%
2500 {%
2501   \expandafter
2502   \XINT@div@final@d
2503   \expandafter
2504   {\the\numexpr #1/#4\relax}{#2}%
2505 }%

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

```

12 Package *xint* implementation

```

2506 \def\XINT@div@final@d #1#2#3#4#5% q,A,Q,L,B puis c
2507 {%
2508     \expandafter
2509     \XINT@div@final@da
2510     \expandafter
2511     {\romannumeral0\XINT@mul@M {#1}#5\Z\Z\Z\Z }%
2512     {\romannumeral0\xint@cleanupzeros@andstop #2}%
2513     {#1}{#3}{#5}%
2514 }%
2515 \def\XINT@div@final@da #1#2%
2516 {%
2517     \expandafter\XINT@div@final@db\expandafter {#2}{#1}%
2518 }%
2519 \def\XINT@div@final@db #1#2% A,qB, puis q,Q,B,c
2520 {%
2521     \ifcase\XINT@Geq {#1}{#2}
2522     \expandafter\XINT@div@final@dc % A < qB
2523     \or\expandafter\XINT@div@final@e % A au moins qB
2524     \fi
2525     {#1}{#2}%
2526 }%
2527 \def\XINT@div@final@e #1#2#3#4#5% A,qB,q,Q,B,puis c
2528 {%
2529     \expandafter\XINT@div@final@f
2530     \expandafter{\romannumeral0\xintsub {#1}{#2}}%
2531     {\romannumeral0\xintadd {\XINT@Rev@andcleanupzeros{#4}}{#3}}%
2532 }%
2533 \def\XINT@div@final@dc #1#2#3% A sans leading zeros,trash,q,Q,B,c
2534 {%
2535     \expandafter\XINT@div@final@dd
2536     \expandafter{\the\numexpr #3-1\relax}{#1}%
2537 }%
2538 \def\XINT@div@final@dd #1#2#3#4% q,A,Q,B puis c
2539 {%
2540     \expandafter\XINT@div@final@f
2541     \expandafter{\romannumeral0\xintsub
2542         {#2}{\romannumeral0\XINT@mul@M {#1}#4\Z\Z\Z\Z }}%
2543     {\romannumeral0\xintadd {\XINT@Rev@andcleanupzeros{#3}}{#1}}%
2544 }%
2545 \def\XINT@div@final@f #1#2#3% R,Q à développer,c
2546 {%
2547     \ifcase \XINT@Sgn {#1}
2548     \xint@afterfi {\XINT@div@final@end {0}}%
2549     \or
2550     \xint@afterfi {\expandafter\XINT@div@final@end
2551         \expandafter % pas de leading zeros dans #1=R
2552         {\romannumeral0\XINT@dsh@checksignx #3\Z {#1}}}%
2553     \fi
2554     {#2}%
2555 }%
2556 \def\XINT@div@final@end #1#2%
2557 {%
2558     \expandafter\space\expandafter {#2}{#1}%

```

12 Package **xint** implementation

```

2559 }%

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

2560 \def\XINT@div@body@a #1%
2561 {%
2562     \XINT@div@body@b #1\Z
2563 }%
2564 \def\XINT@div@body@b #1#2#3#4#5#6#7#8#9\Z
2565 {%
2566     \XINT@div@body@c
2567     {#1#2#3#4#5#6#7#8#9}%
2568     {#1#2#3#4#5#6#7#8}%
2569 }%

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

2570 \def\XINT@div@body@c #1#2#3%
2571 {%
2572     \XINT@div@body@d {#3}{#1}\Z {#2}{#3}%
2573 }%
2574 \def\XINT@div@body@d #1#2#3#4#5#6%
2575 {%
2576     \ifnum #1 > 0
2577         \expandafter
2578         \XINT@div@body@d
2579         \expandafter
2580         {\the\numexpr #1-4\expandafter}%
2581     \else
2582         \expandafter
2583         \XINT@div@body@e
2584     \fi
2585     {#6#5#4#3#2}%
2586 }%
2587 \def\XINT@div@body@e #1#2\Z #3%
2588 {%
2589     \XINT@div@body@f {#3}{#1}{#2}%
2590 }%

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

2591 \def\XINT@div@body@f #1#2#3#4#5#6#7#8%
2592 {%
2593     \expandafter\XINT@div@body@g
2594     \expandafter
2595     {\the\numexpr (#1+(#5+1)/2)/(#5+1)-1\relax}%
2596     {#2}{#8}{#4}{#5}{#3}{#6}{#7}{#8}%
2597 }%

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

```

12 Package *xint* implementation

```

2598 \def\XINT@div@body@g #1#2#3%
2599 {%
2600   \expandafter
2601     \XINT@div@body@h
2602   \romannumeral0\XINT@div@sub@xpxp
2603     {\romannumeral0\XINT@pow@mul@enter #3\W\X\Y\Z #1\W\X\Y\Z }%
2604     {#2}\Z
2605     {#3}{#1}%
2606 }%

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

2607 \def\XINT@div@body@h #1#2#3#4#5#6#7#8#9\Z
2608 {%
2609   \ifnum #1#2#3#4>0
2610     \xint@afterfi{\XINT@div@body@i {#1#2#3#4#5#6#7#8}}%
2611   \else
2612     \expandafter\XINT@div@body@k
2613     \fi
2614     {#1#2#3#4#5#6#7#8#9}%
2615 }%

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

2616 \def\XINT@div@body@i #1#2#3#4#5#6%
2617 {%
2618   \expandafter\XINT@div@body@j
2619   \expandafter{\the\numexpr (#1+(#6+1)/2)/(#6+1)-1\relax }%
2620   {#2}{#3}{#4}{#5}{#6}%
2621 }%

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

2622 \def\XINT@div@body@j #1#2#3#4%
2623 {%
2624   \expandafter
2625     \XINT@div@body@l
2626   \expandafter{\romannumeral0\XINT@div@sub@xpxp
2627     {\romannumeral0\XINT@pow@mul@enter #3\W\X\Y\Z #1\W\X\Y\Z }%
2628     {\XINT@Rev{#2}}}%
2629     {#4+#1}%
2630 }%

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

2631 \def\XINT@div@body@k #1#2%
2632 {%
2633   \XINT@div@body@l {#1}%
2634 }%

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

```

12 Package **xint** implementation

```

2635 \def\XINT@div@body@l #1#2#3#4#5#6#7%
2636 {%
2637   \expandafter
2638     \XINT@div@body@m
2639   \the\numexpr 1000000000+#2\relax
2640     {#6}{#3}{#7}{#1#5}{#4}%
2641 }%

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

2642 \def\XINT@div@body@m #1#2#3#4#5#6#7#8#9%
2643 {%
2644   \ifnum #2#3#4#5>0
2645     \xint@afterfi {\XINT@div@body@n {#9#8#7#6#5#4#3#2}}%
2646   \else
2647     \xint@afterfi {\XINT@div@body@n {#9#8#7#6}}%
2648   \fi
2649 }%

  q renversé, Q, K, L, A', x, B, c

2650 \def\XINT@div@body@n #1#2%
2651 {%
2652   \expandafter\XINT@div@body@o\expandafter
2653   {\romannumeral0\XINT@sum@A 0}{#1\W\X\Y\Z #2\W\X\Y\Z }%
2654 }%

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

2655 \def\XINT@div@body@o #1#2#3#4%
2656 {%
2657   \XINT@div@body@p {#3}{#2}{#4\Z {#1}}%
2658 }%

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

2659 \def\XINT@div@body@p #1#2#3#4#5#6#7%
2660 {%
2661   \ifnum #1 > #2
2662     \xint@afterfi
2663     {\ifnum #4#5#6#7 > 0
2664       \expandafter\XINT@div@body@q
2665     \else
2666       \expandafter\XINT@div@body@repeatp
2667     \fi }%
2668   \else
2669     \expandafter\XINT@div@gotofinal@a
2670   \fi
2671   {#1}{#2}{#3}#4#5#6#7%
2672 }%

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

```

12 Package *xint* implementation

```

2673 \def\XINT@div@body@repeatp #1#2#3#4#5#6#7%
2674 {%
2675     \expandafter
2676         \XINT@div@body@p
2677     \expandafter
2678         {\the\numexpr #1-4\relax}{#2}{0000#3}%
2679 }%

    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

2680 \def\XINT@div@body@q #1#2#3#4\Z #5#6%
2681 {%
2682     \XINT@div@body@a {#4}{#2}{#6}{#3#5}{#1}%
2683 }%

    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

2684 \def\XINT@div@gotofinal@a #1#2#3#4\Z %
2685 {%
2686     \XINT@div@gotofinal@b #3\Z {#4}{#1}%
2687 }%

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

2688 \def\XINT@div@gotofinal@b 0000#1\Z #2#3#4#5%
2689 {%
2690     \XINT@div@final@a {#2}{#3}{#5}{#1#4}{#3}%
2691 }%

    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,

2692 \def\XINT@div@sub@xpxp #1%
2693 {%
2694     \expandafter
2695         \XINT@div@sub@xpxp@
2696     \expandafter
2697         {#1}%
2698 }%
2699 \def\XINT@div@sub@xpxp@ #1#2%
2700 {%
2701     \expandafter\expandafter\expandafter
2702         \XINT@div@sub@xpxp@@
2703         #2\W\X\Y\Z #1\W\X\Y\Z
2704 }%
2705 \def\XINT@div@sub@xpxp@@
2706 {%

```

12 Package **xint** implementation

```

2707 \XINT@div@sub@A 1{}%
2708 }%
2709 \def\XINT@div@sub@A #1#2#3#4#5#6%
2710 {%
2711 \xint@w
2712 #3\xint@div@sub@az
2713 \W\XINT@div@sub@B #1{#3#4#5#6}{#2}%
2714 }%
2715 \def\XINT@div@sub@B #1#2#3#4\W\X\Y\Z #5#6#7#8%
2716 {%
2717 \xint@w
2718 #5\xint@div@sub@bz
2719 \W\XINT@div@sub@onestep #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
2720 }%
2721 \def\XINT@div@sub@onestep #1#2#3#4#5#6%
2722 {\expandafter
2723 \XINT@div@sub@backtoA\the\numexpr 11#5#4#3#2-#6+#1-1\relax.%
2724 }%
2725 \def\XINT@div@sub@backtoA #1#2#3.#4%
2726 {%
2727 \XINT@div@sub@A #2{#3#4}%
2728 }%
2729 \def\xint@div@sub@bz
2730 \W\XINT@div@sub@onestep #1#2#3#4#5#6#7%
2731 {%
2732 \xint@UDzerofork
2733 #1\dummy \XINT@div@sub@C %
2734 0\dummy \XINT@div@sub@D % pas de retenue
2735 \xint@UDkrof
2736 {#7}#2#3#4#5%
2737 }%
2738 \def\XINT@div@sub@D #1#2\W\X\Y\Z
2739 {%
2740 \expandafter\space
2741 \romannumeral0%
2742 \XINT@rord@main {}#2%
2743 \xint@UNDEF
2744 \xint@undef\xint@undef\xint@undef\xint@undef
2745 \xint@undef\xint@undef\xint@undef\xint@undef
2746 \xint@UNDEF
2747 #1%
2748 }%
2749 \def\XINT@div@sub@C #1#2#3#4#5%
2750 {%
2751 \xint@w
2752 #2\xint@div@sub@cz
2753 \W\XINT@div@sub@AC@onestep {#5#4#3#2}{#1}%
2754 }%
2755 \def\XINT@div@sub@AC@onestep #1%
2756 {\expandafter
2757 \XINT@div@sub@backtoC\the\numexpr 11#1-1\relax.%
2758 }%

```



```

2759 \def\XINT@div@sub@backtoC #1#2#3.#4%
2760 {%
2761     \XINT@div@sub@AC@checkcarry #2{#3#4}% la retenue va \^etre examin\'ee
2762 }%
2763 \def\XINT@div@sub@AC@checkcarry #1%
2764 {%
2765     \xint@one #1\xint@div@sub@AC@nocarry 1\XINT@div@sub@C
2766 }%
2767 \def\xint@div@sub@AC@nocarry 1\XINT@div@sub@C #1#2\W\X\Y\Z
2768 {%
2769     \expandafter\space
2770     \romannumeral0%
2771     \XINT@rord@main {}#2%
2772     \xint@UNDEF
2773     \xint@undef\xint@undef\xint@undef\xint@undef
2774     \xint@undef\xint@undef\xint@undef\xint@undef
2775     \xint@UNDEF
2776     #1%
2777 }%
2778 \def\xint@div@sub@cz\W\XINT@div@sub@AC@onestep #1#2{ #2}%
2779 \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

```

2780 \def\xintFDg {\romannumeral0\xintfdg }%
2781 \def\xintfdg #1%
2782 {%
2783     \expandafter\expandafter\expandafter
2784     \XINT@fdg #1\W\Z
2785 }%
2786 \def\XINT@FDg #1{\romannumeral0\XINT@fdg #1\W\Z }%
2787 \def\XINT@fdg #1#2%
2788 {%
2789     \xint@xpxp@andstop
2790     \xint@UDzerominusfork
2791     #1-\dummy {\expandafter 0}%      zero
2792     0#1\dummy {\expandafter #2}%     negative
2793     0-\dummy {\expandafter #1}%     positive
2794     \xint@UDkrof
2795     \xint@z
2796 }%

```

12.27 \xintLDg

LAST DIGIT

```

2797 \def\xintLDg {\romannumeral0\xintldg }%
2798 \def\xintldg #1%
2799 {%
2800     \expandafter\expandafter\expandafter
2801         \XINT@ldg
2802     \expandafter\expandafter\expandafter
2803     {#1}%
2804 }%
2805 \def\XINT@LDg #1{\romannumeral0\XINT@ldg {#1}}%
2806 \def\XINT@ldg #1%
2807 {%
2808     \expandafter
2809     \XINT@ldg@
2810     \romannumeral0\XINT@rev {#1}\Z
2811 }%
2812 \def\XINT@ldg@ #1%
2813 {%
2814     \expandafter\space\expandafter #1\xint@z
2815 }%

```

12.28 \xintOdd

ODDNESS

```

2816 \def\xintOdd {\romannumeral0\xintodd }%
2817 \def\xintodd #1%
2818 {%
2819     \ifodd\xintLDg{#1}
2820         \xint@afterfi{ 1}%
2821     \else
2822         \xint@afterfi{ 0}%
2823     \fi
2824 }%
2825 \def\XINT@Odd #1%
2826 {\romannumeral0%
2827     \ifodd\XINT@LDg{#1}
2828         \xint@afterfi{ 1}%
2829     \else
2830         \xint@afterfi{ 0}%
2831     \fi
2832 }%

```

12.29 \xintDSL

DECIMAL SHIFT LEFT (=MULTIPLICATION PAR 10)

```

2833 \def\xintDSL {\romannumeral0\xintdsl }%
2834 \def\xintdsl #1%
2835 {%
2836     \expandafter\expandafter\expandafter
2837         \XINT@dsl #1\Z
2838 }%
2839 \def\XINT@DSL #1{\romannumeral0\XINT@dsl #1\Z }%

```

```

2840 \def\XINT@dsl #1%
2841 {%
2842     \xint@zero #1\xint@dsl@zero 0\XINT@dsl@ #1%
2843 }%
2844 \def\xint@dsl@zero 0\XINT@dsl@ 0#1\Z { 0}%
2845 \def\XINT@dsl@ #1\Z { #10}%

```

12.30 \xintDSR

DECIMAL SHIFT RIGHT (=DIVISION PAR 10)

```

2846 \def\xintDSR {\romannumeral0\xintdsr }%
2847 \def\xintdsr #1%
2848 {%
2849     \expandafter\expandafter\expandafter
2850     \XINT@dsr@a
2851     \expandafter\expandafter\expandafter
2852     {#1}\W\Z
2853 }%
2854 \def\XINT@DSR #1{\romannumeral0\XINT@dsr@a {#1}\W\Z }%
2855 \def\XINT@dsr@a
2856 {%
2857     \expandafter
2858     \XINT@dsr@b
2859     \romannumeral0\XINT@rev
2860 }%
2861 \def\XINT@dsr@b #1#2#3\Z
2862 {%
2863     \xint@w #2\xint@dsr@onedigit\W
2864     \xint@minus #2\xint@dsr@onedigit-%
2865     \expandafter
2866     \XINT@dsr@removew
2867     \romannumeral0\XINT@rev {#2#3}%
2868 }%
2869 \def\xint@dsr@onedigit #1\XINT@rev #2{ 0}%
2870 \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'.

```

```

2871 \def\xintDSHr {\romannumeral0\xintdsr }%
2872 \def\xintdsr #1%
2873 {\expandafter\expandafter\expandafter
2874     \XINT@dsr@checkxpositive #1\Z
2875 }%
2876 \def\XINT@dsr@checkxpositive #1%

```

```

2877 {%
2878   \xint@UDzerominusfork
2879   0#1\dummy \XINT@dshr@xzeroorneg
2880   #1-\dummy \XINT@dshr@xzeroorneg
2881   0-\dummy \XINT@dshr@xpositive
2882   \xint@UDkrof #1%
2883 }%
2884 \def\xint@dshr@xzeroorneg #1\Z #2{ 0}%
2885 \def\xint@dshr@xpositive #1\Z
2886 {%
2887   \expandafter
2888   \xint@secondoftwo@andstop
2889   \romannumeral0\xintdsx {#1}%
2890 }%
2891 \def\xintDSH {\romannumeral0\xintdsh }%
2892 \def\xintdsh #1#2%
2893 {%
2894   \expandafter\expandafter\expandafter
2895   \xint@dsh
2896   \expandafter\expandafter\expandafter
2897   {#2}{#1}%
2898 }%
2899 \def\xint@dsh #1#2%
2900 {%
2901   \expandafter\expandafter\expandafter
2902   \XINT@dsh@checksignx
2903   #2\Z {#1}%
2904 }%
2905 \def\XINT@dsh@checksignx #1%
2906 {%
2907   \xint@UDzerominusfork
2908   #1-\dummy \XINT@dsh@xiszero
2909   0#1\dummy \XINT@dsx@xisNeg
2910   0-\dummy {\XINT@dsh@xisPos #1}%
2911   \xint@UDkrof
2912 }%
2913 \def\XINT@dsh@xiszero #1\Z #2{ #2}%
2914 \def\XINT@dsh@xisPos #1\Z #2%
2915 {%
2916   \expandafter
2917   \xint@firstoftwo@andstop
2918   \romannumeral0\XINT@dsx@checksignA #2\Z {#1}%
2919 }%

```

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 $x \leq 0$, fait $A \rightarrow A \cdot 10^{|x|}$
si $x > 0$, et $A \geq 0$, fait $A \rightarrow \{\text{quo}(A, 10^x)\} \{\text{rem}(A, 10^x)\}$

12 Package *xint* implementation

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

```

```

2966 \or
2967 \expandafter \XINT@dsx@exitiv
2968 \or
2969 \expandafter \XINT@dsx@exitv
2970 \or
2971 \expandafter \XINT@dsx@exitvi
2972 \or
2973 \expandafter \XINT@dsx@exitvii
2974 \else
2975 \xint@afterfi
2976 {\expandafter
2977 \XINT@dsx@zeroloop
2978 \expandafter {\the\numexpr #1-8}00000000%
2979 }%
2980 \fi
2981 }%
2982 \def\XINT@dsx@exit #1\Z
2983 {\XINT@dsx@addzeros {#1}}%
2984 \def\XINT@dsx@exiti #1\Z
2985 {\XINT@dsx@addzeros {0#1}}%
2986 \def\XINT@dsx@exitii #1\Z
2987 {\XINT@dsx@addzeros {00#1}}%
2988 \def\XINT@dsx@exitiii #1\Z
2989 {\XINT@dsx@addzeros {000#1}}%
2990 \def\XINT@dsx@exitiv #1\Z
2991 {\XINT@dsx@addzeros {0000#1}}%
2992 \def\XINT@dsx@exitv #1\Z
2993 {\XINT@dsx@addzeros {00000#1}}%
2994 \def\XINT@dsx@exitvi #1\Z
2995 {\XINT@dsx@addzeros {000000#1}}%
2996 \def\XINT@dsx@exitvii #1\Z
2997 {\XINT@dsx@addzeros {0000000#1}}%
2998 \def\XINT@dsx@addzeros #1#2{ #2#1}%
2999 \def\XINT@dsx@xisPos #1\Z #2%
3000 {%
3001 \XINT@dsx@checksignA #2\Z {#1}%
3002 }%
3003 \def\XINT@dsx@checksignA #1%
3004 {%
3005 \xint@UDzerominusfork
3006 #1-\dummy \XINT@dsx@AisZero
3007 0#1\dummy \XINT@dsx@AisNeg
3008 0-\dummy {\XINT@dsx@AisPos #1}%
3009 \xint@UDkrof
3010 }%
3011 \def\XINT@dsx@AisZero #1\Z #2{ {0}{0}}%
3012 \def\XINT@dsx@AisNeg #1\Z #2%
3013 {%
3014 \expandafter
3015 \XINT@dsx@AisNeg@dosplit@andcheckfirst
3016 \romannumeral0\XINT@split@checksize {#2}{#1}%
3017 }%

```

```

3018 \def\XINT@dsx@AisNeg@dosplit@andcheckfirst #1%
3019 {%
3020   \XINT@dsx@AisNeg@checkiffirstempty #1\Z
3021 }%
3022 \def\XINT@dsx@AisNeg@checkiffirstempty #1%
3023 {%
3024   \xint@z #1\XINT@dsx@AisNeg@finish@zero\Z
3025   \XINT@dsx@AisNeg@finish@notzero #1%
3026 }%
3027 \def\XINT@dsx@AisNeg@finish@zero\Z
3028   \XINT@dsx@AisNeg@finish@notzero\Z #1%
3029 {%
3030   \expandafter
3031     \XINT@dsx@end
3032   \expandafter {\romannumeral0\XINT@num {-#1}}{0}%
3033 }%
3034 \def\XINT@dsx@AisNeg@finish@notzero #1\Z #2%
3035 {%
3036   \expandafter
3037     \XINT@dsx@end
3038   \expandafter {\romannumeral0\XINT@num {#2}}{-#1}%
3039 }%
3040 \def\XINT@dsx@AisPos #1\Z #2%
3041 {%
3042   \expandafter
3043     \XINT@dsx@AisPos@finish
3044     \romannumeral0\XINT@split@checksize {#2}{#1}%
3045 }%
3046 \def\XINT@dsx@AisPos@finish #1#2%
3047 {%
3048   \expandafter
3049     \XINT@dsx@end
3050   \expandafter {\romannumeral0\XINT@num {#2}}%
3051     {\romannumeral0\XINT@num {#1}}%
3052 }%
3053 \def\XINT@dsx@end #1#2%
3054 {%
3055   \expandafter\space\expandafter{#2}{#1}%
3056 }%

```

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.

12 Package *xint* implementation

(*) 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@checksize
3083     \expandafter\expandafter\expandafter
3084     {#2}{#1}%
3085 }%
3086 \def\XINT@split@checksize #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

```


12 Package *xint* implementation

```

3107      #1-\dummy \XINT@split@zerosplit
3108      0#1\dummy \XINT@split@fromleft
3109      0-\dummy {\XINT@split@fromright #1}%
3110 \xint@UDkrof
3111 }%
3112 \def\XINT@split@zerosplit #1\Z #2{ {#2}{} }%
3113 \def\XINT@split@fromleft #1\Z #2%
3114 {%
3115 \XINT@split@fromleft@loop {#1}{}#2\W\W\W\W\W\W\W\W\Z
3116 }%
3117 \def\XINT@split@fromleft@loop #1%
3118 {%
3119 \ifcase #1
3120 \expandafter\XINT@split@fromleft@endsplit
3121 \or
3122 \expandafter\XINT@split@fromleft@one@endend
3123 \or
3124 \expandafter\XINT@split@fromleft@two@endend
3125 \or
3126 \expandafter\XINT@split@fromleft@three@endend
3127 \or
3128 \expandafter\XINT@split@fromleft@four@endend
3129 \or
3130 \expandafter\XINT@split@fromleft@five@endend
3131 \or
3132 \expandafter\XINT@split@fromleft@six@endend
3133 \or
3134 \expandafter\XINT@split@fromleft@seven@endend
3135 \else
3136 \expandafter \XINT@split@fromleft@loop@perhaps
3137 \expandafter
3138 {\the\numexpr
3139 #1-8\expandafter\expandafter\expandafter}%
3140 \expandafter
3141 \XINT@split@fromleft@eight
3142 \fi
3143 }%
3144 \def\XINT@split@fromleft@endsplit #1#2\W #3\Z
3145 { {#1}{#2} }%
3146 \def\XINT@split@fromleft@eight #1#2#3#4#5#6#7#8#9%
3147 {%
3148 #9{#1#2#3#4#5#6#7#8#9}%
3149 }%
3150 \def\XINT@split@fromleft@loop@perhaps #1#2%
3151 {%
3152 \xint@w #2\XINT@split@fromleft@toofar\W \XINT@split@fromleft@loop
3153 {#1}%
3154 }%
3155 \def\XINT@split@fromleft@toofar\W \XINT@split@fromleft@loop #1#2#3\Z
3156 {%
3157 \XINT@split@fromleft@toofar@b #2\Z
3158 }%

```

12 Package *xint* implementation

```

3159 \def\XINT@split@fromleft@toofar@b #1\W #2\Z
3160 {%
3161     \space {#1}}}%
3162 }%
3163 \def\XINT@split@fromleft@one@endend
3164     {\expandafter\XINT@split@fromleft@checkiftoofar\XINT@split@fromleft@one }%
3165 \def\XINT@split@fromleft@one #1#2{#2{#1#2}}}%
3166 \def\XINT@split@fromleft@two@endend
3167     {\expandafter\XINT@split@fromleft@checkiftoofar\XINT@split@fromleft@two }%
3168 \def\XINT@split@fromleft@two #1#2#3{#3{#1#2#3}}}%
3169 \def\XINT@split@fromleft@three@endend
3170     {\expandafter\XINT@split@fromleft@checkiftoofar\XINT@split@fromleft@three }%
3171 \def\XINT@split@fromleft@three #1#2#3#4{#4{#1#2#3#4}}}%
3172 \def\XINT@split@fromleft@four@endend
3173     {\expandafter\XINT@split@fromleft@checkiftoofar\XINT@split@fromleft@four }%
3174 \def\XINT@split@fromleft@four #1#2#3#4#5{#5{#1#2#3#4#5}}}%
3175 \def\XINT@split@fromleft@five@endend
3176     {\expandafter\XINT@split@fromleft@checkiftoofar\XINT@split@fromleft@five }%
3177 \def\XINT@split@fromleft@five #1#2#3#4#5#6{#6{#1#2#3#4#5#6}}}%
3178 \def\XINT@split@fromleft@six@endend
3179     {\expandafter\XINT@split@fromleft@checkiftoofar\XINT@split@fromleft@six }%
3180 \def\XINT@split@fromleft@six #1#2#3#4#5#6#7{#7{#1#2#3#4#5#6#7}}}%
3181 \def\XINT@split@fromleft@seven@endend
3182     {\expandafter\XINT@split@fromleft@checkiftoofar\XINT@split@fromleft@seven }%
3183 \def\XINT@split@fromleft@seven #1#2#3#4#5#6#7#8{#8{#1#2#3#4#5#6#7#8}}}%
3184 \def\XINT@split@fromleft@checkiftoofar #1#2#3\W #4\Z
3185 {%
3186     \xint@w #1\XINT@split@fromleft@wenttoofar\W
3187     \space {#2}{#3}}%
3188 }%
3189 \def\XINT@split@fromleft@wenttoofar\W\space #1%
3190 {%
3191     \XINT@split@fromleft@wenttoofar@b #1\Z
3192 }%
3193 \def\XINT@split@fromleft@wenttoofar@b #1\W #2\Z
3194 {%
3195     \space {#1}}%
3196 }%
3197 \def\XINT@split@fromright #1\Z #2%
3198 {%
3199     \expandafter
3200         \XINT@split@fromright@a
3201     \expandafter
3202         {\romannumeral0\XINT@rev {#2}}{#1}{#2}}%
3203 }%
3204 \def\XINT@split@fromright@a #1#2%
3205 {%
3206     \XINT@split@fromright@loop {#2}{#1\W\W\W\W\W\W\W\W\Z
3207 }%
3208 \def\XINT@split@fromright@loop #1%
3209 {%
3210     \ifcase #1

```

12 Package *xint* implementation

```

3211     \expandafter\XINT@split@fromright@endsplit
3212 \or
3213     \expandafter\XINT@split@fromright@one@endend
3214 \or
3215     \expandafter\XINT@split@fromright@two@endend
3216 \or
3217     \expandafter\XINT@split@fromright@three@endend
3218 \or
3219     \expandafter\XINT@split@fromright@four@endend
3220 \or
3221     \expandafter\XINT@split@fromright@five@endend
3222 \or
3223     \expandafter\XINT@split@fromright@six@endend
3224 \or
3225     \expandafter\XINT@split@fromright@seven@endend
3226 \else
3227     \expandafter \XINT@split@fromright@loop@perhaps
3228     \expandafter
3229         {\the\numexpr
3230         #1-8\expandafter\expandafter\expandafter }%
3231     \expandafter
3232     \XINT@split@fromright@eight
3233 \fi
3234 }%
3235 \def\XINT@split@fromright@endsplit #1#2\W #3\Z #4%
3236 {%
3237     \expandafter
3238     \space
3239     \expandafter {\romannumeral0\XINT@rev{#2}}{#1}%
3240 }%
3241 \def\XINT@split@fromright@eight #1#2#3#4#5#6#7#8#9%
3242 {%
3243     #9{#9#8#7#6#5#4#3#2#1}%
3244 }%
3245 \def\XINT@split@fromright@loop@perhaps #1#2%
3246 {%
3247     \xint@w #2\XINT@split@fromright@toofar\W\XINT@split@fromright@loop
3248     {#1}%
3249 }%
3250 \def\XINT@split@fromright@toofar\W\XINT@split@fromright@loop #1#2#3\Z
3251     { {}}%
3252 \def\XINT@split@fromright@one@endend
3253 {\expandafter\XINT@split@fromright@checkiftoofar\XINT@split@fromright@one }%
3254 \def\XINT@split@fromright@one #1#2{#2{#2#1}}%
3255 \def\XINT@split@fromright@two@endend
3256 {\expandafter\XINT@split@fromright@checkiftoofar\XINT@split@fromright@two }%
3257 \def\XINT@split@fromright@two #1#2#3{#3{#3#2#1}}%
3258 \def\XINT@split@fromright@three@endend
3259 {\expandafter\XINT@split@fromright@checkiftoofar\XINT@split@fromright@three }%
3260 \def\XINT@split@fromright@three #1#2#3#4{#4{#4#3#2#1}}%
3261 \def\XINT@split@fromright@four@endend
3262 {\expandafter\XINT@split@fromright@checkiftoofar\XINT@split@fromright@four }%

```

```

3263 \def\XINT@split@fromright@four #1#2#3#4#5{#5{#5#4#3#2#1}}%
3264 \def\XINT@split@fromright@five@endend
3265 {\expandafter\XINT@split@fromright@checkiftoofar\XINT@split@fromright@five}%
3266 \def\XINT@split@fromright@five #1#2#3#4#5#6{#6{#6#5#4#3#2#1}}%
3267 \def\XINT@split@fromright@six@endend
3268 {\expandafter\XINT@split@fromright@checkiftoofar\XINT@split@fromright@six}%
3269 \def\XINT@split@fromright@six #1#2#3#4#5#6#7{#7{#7#6#5#4#3#2#1}}%
3270 \def\XINT@split@fromright@seven@endend
3271 {\expandafter\XINT@split@fromright@checkiftoofar\XINT@split@fromright@seven}%
3272 \def\XINT@split@fromright@seven #1#2#3#4#5#6#7#8{#8{#8#7#6#5#4#3#2#1}}%
3273 \def\XINT@split@fromright@checkiftoofar #1%
3274 {%
3275     \xint@w #1\XINT@split@fromright@wenttoofar\W
3276     \XINT@split@fromright@endsplit
3277 }%
3278 \def\XINT@split@fromright@wenttoofar\W\XINT@split@fromright@endsplit #1\Z #2%
3279     { }{#2}}%
3280 \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.

```

3281 \begingroup\catcode61\catcode48\catcode32=10\relax%
3282 \catcode13=5 % ^^M
3283 \endlinechar=13 %
3284 \catcode123=1 % {
3285 \catcode125=2 % }
3286 \catcode64=11 % @
3287 \catcode35=6 % #
3288 \catcode44=12 % ,
3289 \catcode45=12 % -
3290 \catcode46=12 % .
3291 \catcode58=12 % :
3292 \def\space { }%
3293 \let\z\endgroup
3294 \expandafter\let\expandafter\x\csname ver@xintgcd.sty\endcsname
3295 \expandafter\let\expandafter\w\csname ver@xint.sty\endcsname
3296 \expandafter
3297 \ifx\csname PackageInfo\endcsname\relax
3298     \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
3299     \else
3300     \def\y#1#2{\PackageInfo{#1}{#2}}%
3301     \fi
3302 \expandafter
3303 \ifx\csname numexpr\endcsname\relax

```

```

3304 \y{xintgcd}{\numexpr not available, aborting input}%
3305 \aftergroup\endinput
3306 \else
3307 \ifx\x\relax % plain-TeX, first loading of xintgcd.sty
3308 \ifx\w\relax % but xint.sty not yet loaded.
3309 \y{xintgcd}{Package xint is required}%
3310 \y{xintgcd}{Will try \string\input\space xint.sty}%
3311 \def\z{\endgroup\input xint.sty\relax}%
3312 \fi
3313 \else
3314 \def\empty {}%
3315 \ifx\x\empty % LaTeX, first loading,
3316 % variable is initialized, but \ProvidesPackage not yet seen
3317 \ifx\w\relax % xint.sty not yet loaded.
3318 \y{xintgcd}{Package xint is required}%
3319 \y{xintgcd}{Will try \string\RequirePackage{xint}}%
3320 \def\z{\endgroup\RequirePackage{xint}}%
3321 \fi
3322 \else
3323 \y{xintgcd}{I was already loaded, aborting input}%
3324 \aftergroup\endinput
3325 \fi
3326 \fi
3327 \fi
3328 \z%

```

13.2 Validation of *xint* loading

```

3329 \begingroup\catcode61\catcode48\catcode32=10\relax%
3330 \catcode13=5 % ^^M
3331 \endlinechar=13 %
3332 \catcode123=1 % {
3333 \catcode125=2 % }
3334 \catcode64=11 % @
3335 \catcode35=6 % #
3336 \catcode44=12 % ,
3337 \catcode45=12 % -
3338 \catcode46=12 % .
3339 \catcode58=12 % :
3340 \expandafter
3341 \ifx\cname PackageInfo\endcsname\relax
3342 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
3343 \else
3344 \def\y#1#2{\PackageInfo{#1}{#2}}%
3345 \fi
3346 \def\empty {}%
3347 \expandafter\let\expandafter\w\cname ver@xint.sty\endcsname
3348 \ifx\w\relax % Plain TeX, user gave a file name at the prompt
3349 \y{xintgcd}{Loading of package xint failed, aborting input}%
3350 \aftergroup\endinput
3351 \fi
3352 \ifx\w\empty % LaTeX, user gave a file name at the prompt

```

```

3353     \y{xintgcd}{Loading of package xint failed, aborting input}%
3354     \aftergroup\endinput
3355   \fi
3356 \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`.

```

3357 \begingroup\catcode61\catcode48\catcode32=10\relax%
3358   \catcode13=5      % ^^M
3359   \endlinechar=13 %
3360   \catcode123=1     % {
3361   \catcode125=2     % }
3362   \catcode64=11     % @
3363   \def\x
3364   {%
3365     \endgroup
3366     \edef\XINT@gcd@restorecatcodes@endinput
3367     {%
3368       \catcode36=\the\catcode36 % $
3369       \catcode47=\the\catcode47 % /
3370       \catcode41=\the\catcode41 % )
3371       \catcode40=\the\catcode40 % (
3372       \catcode42=\the\catcode42 % *
3373       \catcode43=\the\catcode43 % +
3374       \catcode62=\the\catcode62 % >
3375       \catcode60=\the\catcode60 % <
3376       \catcode58=\the\catcode58 % :
3377       \catcode46=\the\catcode46 % .
3378       \catcode45=\the\catcode45 % -
3379       \catcode44=\the\catcode44 % ,
3380       \catcode35=\the\catcode35 % #
3381       \catcode64=\the\catcode64 % @
3382       \catcode125=\the\catcode125 % }
3383       \catcode123=\the\catcode123 % {
3384       \endlinechar=\the\endlinechar
3385       \catcode13=\the\catcode13 % ^^M
3386       \catcode32=\the\catcode32 %
3387       \catcode61=\the\catcode61 % =
3388       \noexpand\endinput
3389     }%
3390     \XINT@setcatcodes
3391     \catcode36=3 % $
3392   }%
3393 \x

```

13.4 Package identification

```

3394 \begingroup
3395   \catcode91=12 % [

```

```

3396 \catcode93=12 % ]
3397 \catcode58=12 % :
3398 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
3399   \def\x#1#2#3[#4]{\endgroup
3400     \immediate\write-1{Package: #3 #4}%
3401     \xdef#1{#4}%
3402   }%
3403 \else
3404   \def\x#1#2[#3]{\endgroup
3405     #2[#{#3}]%
3406     \ifx#1\@undefined
3407       \xdef#1{#3}%
3408     \fi
3409     \ifx#1\relax
3410       \xdef#1{#3}%
3411     \fi
3412   }%
3413 \fi
3414 \expandafter\x\csname ver@xintgcd.sty\endcsname
3415 \ProvidesPackage{xintgcd}%
3416 [2013/04/05 v1.02 Euclidean algorithm with xint package (jfb)]%

```

13.5 \xintGCD

```

3417 \def\xintGCD {\romannumeral0\xintgcd }%
3418 \def\xintgcd #1%
3419 {%
3420   \expandafter
3421   \XINT@gcd
3422   \expandafter
3423   {\romannumeral0\xintabs {#1}}%
3424 }%
3425 \def\XINT@gcd #1#2%
3426 {%
3427   \expandafter
3428   \XINT@gcd@fork
3429   \romannumeral0\xintabs {#2}\Z #1\Z
3430 }%
3431   Ici #3#4=A, #1#2=B
3432
3431 \def\XINT@gcd@fork #1#2\Z #3#4\Z
3432 {%
3433   \xint@UDzerofork
3434   #1\dummy \XINT@gcd@BisZero
3435   #3\dummy \XINT@gcd@AisZero
3436   0\dummy \XINT@gcd@loop
3437   \xint@UDkrof
3438   {#1#2}{#3#4}%
3439 }%
3440 \def\XINT@gcd@AisZero #1#2{ #1}%
3441 \def\XINT@gcd@BisZero #1#2{ #2}%
3442 \def\XINT@gcd@CheckRem #1#2\Z
3443 {%
3444   \xint@zero #1\xint@gcd@end0\XINT@gcd@loop {#1#2}%

```

```

3445 }%
3446 \def\xint@gcd@end0\XINT@gcd@loop #1#2{ #2}%
      #1=B, #2=A
3447 \def\XINT@gcd@loop #1#2%
3448 {%
3449   \expandafter\expandafter\expandafter
3450     \XINT@gcd@CheckRem
3451   \expandafter\xint@secondoftwo
3452   \romannumeral0\XINT@div@prepare {#1}{#2}\Z
3453   {#1}%
3454 }%

```

13.6 \xintBezout

```

3455 \def\xintBezout {\romannumeral0\xintbezout }%
3456 \def\xintbezout #1%
3457 {%
3458   \expandafter\expandafter\expandafter
3459     \xint@bezout
3460   \expandafter\expandafter\expandafter
3461     {#1}%
3462 }%
3463 \def\xint@bezout #1#2%
3464 {\expandafter\expandafter\expandafter
3465   \XINT@bezout@fork #2\Z #1\Z
3466 }%
      #3#4 = A, #1#2=B
3467 \def\XINT@bezout@fork #1#2\Z #3#4\Z
3468 {%
3469   \xint@UDzerosfork
3470   #1#3\dummy \XINT@bezout@botharezero
3471   #10\dummy \XINT@bezout@secondiszero
3472   #30\dummy \XINT@bezout@firstiszero
3473   00\dummy
3474   {\xint@UDsignsfork
3475     #1#3\dummy \XINT@bezout@minusminus % A < 0, B < 0
3476     #1-\dummy \XINT@bezout@minusplus  % A > 0, B < 0
3477     #3-\dummy \XINT@bezout@plusminus  % A < 0, B > 0
3478     --\dummy \XINT@bezout@plusplus   % A > 0, B > 0
3479   \xint@UDkrof }%
3480   \xint@UDkrof
3481   {#2}{#4}#1#3{#3#4}{#1#2}% #1#2=B, #3#4=A
3482 }%
3483 \def\XINT@bezout@botharezero #1#2#3#4#5#6%
3484 {%
3485   \xintError:NoBezoutForZeros
3486   \space {0}{0}{0}{0}{0}%
3487 }%

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

```


13 Package *xintgcd* implementation

```

3488 \def\XINT@bezout@firstiszero #1#2#3#4#5#6%
3489 {%
3490   \xint@UDsignfork
3491     #3\dummy { {0}{#3#1}{0}{1}{#1}}%
3492     -\dummy { {0}{#3#1}{0}{-1}{#1}}%
3493   \xint@UDkrof
3494 }%

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

3495 \def\XINT@bezout@secondiszero #1#2#3#4#5#6%
3496 {%
3497   \xint@UDsignfork
3498     #4\dummy{ {#4#2}{0}{-1}{0}{#2}}%
3499     -\dummy{ {#4#2}{0}{1}{0}{#2}}%
3500   \xint@UDkrof
3501 }%

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

3502 \def\XINT@bezout@minusminus #1#2#3#4%
3503 {%
3504   \expandafter\XINT@bezout@mm@post
3505   \romannumeral0\XINT@bezout@loop@a 1{#1}{#2}1001%
3506 }%
3507 \def\XINT@bezout@mm@post #1#2%
3508 {%
3509   \expandafter
3510     \XINT@bezout@mm@postb
3511   \expandafter
3512     {\romannumeral0\xintopp{#2}}{\romannumeral0\xintopp{#1}}%
3513 }%
3514 \def\XINT@bezout@mm@postb #1#2%
3515 {%
3516   \expandafter
3517     \XINT@bezout@mm@postc
3518   \expandafter {#2}{#1}%
3519 }%
3520 \def\XINT@bezout@mm@postc #1#2#3#4#5%
3521 {%
3522   \space {#4}{#5}{#1}{#2}{#3}%
3523 }%

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

3524 \def\XINT@bezout@minusplus #1#2#3#4%
3525 {%
3526   \expandafter\XINT@bezout@mp@post
3527   \romannumeral0\XINT@bezout@loop@a 1{#1}{#4#2}1001%
3528 }%
3529 \def\XINT@bezout@mp@post #1#2%
3530 {%
3531   \expandafter
3532     \XINT@bezout@mp@postb

```

13 Package *xintgcd* implementation

```

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 }%
```

13 Package *xintgcd* implementation

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)
 $\{-(-1)^n\}\{q(n)\}\{r(n)\}\{r(n-1)\}\{\alpha(n-1)\}\{\beta(n-1)\}\{\alpha(n-2)\}\{\beta(n-2)\}$

```

3571 \def\XINT@bezout@loop@b #1#2#3#4#5#6#7#8%
3572 {%
3573     \expandafter
3574         \XINT@bezout@loop@c
3575     \expandafter
3576         {\romannumeral0\xintadd{\XINT@Mul{#5}{#2}}{#7}}%
3577         {\romannumeral0\xintadd{\XINT@Mul{#6}{#2}}{#8}}%
3578     {#1}{#3}{#4}{#5}{#6}%
3579 }%

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

3580 \def\XINT@bezout@loop@c #1#2%
3581 {%
3582     \expandafter
3583         \XINT@bezout@loop@d
3584     \expandafter
3585         {#2}{#1}%
3586 }%

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

3587 \def\XINT@bezout@loop@d #1#2#3#4#5%
3588 {%
3589     \XINT@bezout@loop@e #4\Z {#3}{#5}{#2}{#1}%
3590 }%

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

3591 \def\XINT@bezout@loop@e #1#2\Z
3592 {%
3593     \xint@zero #1\xint@bezout@loop@exit0\XINT@bezout@loop@f
3594     {#1#2}%
3595 }%

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

3596 \def\XINT@bezout@loop@f #1#2%
3597 {%
3598     \XINT@bezout@loop@a {#2}{#1}%
3599 }%

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

```

```

3600 \def\xint@bezout@loop@exit0\XINT@bezout@loop@f #1#2%
3601 {%
3602     \ifcase #2
3603     \or \expandafter\XINT@bezout@exiteven
3604     \else\expandafter\XINT@bezout@exitodd
3605     \fi
3606 }%
3607 \def\XINT@bezout@exiteven #1#2#3#4#5%
3608 {%
3609     \space {#5}{#4}{#1}%
3610 }%
3611 \def\XINT@bezout@exitodd #1#2#3#4#5%
3612 {%
3613     \space {-#5}{-#4}{#1}%
3614 }%

```

13.7 \xintEuclideanAlgorithm

Pour Euclide:

$\{N\}\{A\}\{D=r(n)\}\{B\}\{q_1\}\{r_1\}\{q_2\}\{r_2\}\{q_3\}\{r_3\} \dots \{q_N\}\{r_N=0\}$
 $u_{2n} = u_{2n+3}u_{2n+2} + u_{2n+4}$ à la n ième étape

```

3615 \def\xintEuclideanAlgorithm {\romannumeral0\xinteucidealgorithm }%
3616 \def\xinteucidealgorithm #1%
3617 {%
3618     \expandafter
3619     \XINT@euc
3620     \expandafter
3621     {\romannumeral0\xintabs {#1}}%
3622 }%
3623 \def\XINT@euc #1#2%
3624 {%
3625     \expandafter
3626     \XINT@euc@fork
3627     \romannumeral0\xintabs {#2}\Z #1\Z
3628 }%

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

3629 \def\XINT@euc@fork #1#2\Z #3#4\Z
3630 {%
3631     \xint@UDzerofork
3632     #1\dummy \XINT@euc@BisZero
3633     #3\dummy \XINT@euc@AisZero
3634     0\dummy \XINT@euc@a
3635     \xint@UDkrof
3636     {0}{#1#2}{#3#4}{#3#4}{#1#2}}{\Z
3637 }%

```

Le {} pour protéger {{A}{B}} si on s'arrête après une étape (B divise A)

On va renvoyer:

$\{N\}\{A\}\{D=r(n)\}\{B\}\{q_1\}\{r_1\}\{q_2\}\{r_2\}\{q_3\}\{r_3\} \dots \{q_N\}\{r_N=0\}$

13 Package *xintgcd* implementation

```

3638 \def\XINT@euc@AisZero #1#2#3#4#5#6{ {1}{0}{#2}{#2}{0}{0}}%
3639 \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} divide v par u

3640 \def\XINT@euc@a #1#2#3%
3641 {%
3642     \expandafter
3643         \XINT@euc@b
3644     \expandafter {\the\numexpr #1+1\expandafter}%
3645     \romannumeral0\XINT@div@prepare {#2}{#3}{#2}%
3646}%

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

3647 \def\XINT@euc@b #1#2#3#4%
3648 {%
3649     \XINT@euc@c #3\Z {#1}{#3}{#4}{{#2}{#3}}%
3650}%

    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.

3651 \def\XINT@euc@c #1#2\Z
3652 {%
3653     \xint@zero #1\xint@euc@end0\XINT@euc@a
3654}%

    {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}

3655 \def\xint@euc@end0\XINT@euc@a #1#2#3#4\Z%
3656 {%
3657     \expandafter\xint@euc@end@
3658     \romannumeral0%
3659     \XINT@rord@main {}#4{{#1}{#3}}%
3660     \xint@UNDEF
3661     \xint@undef\xint@undef\xint@undef\xint@undef
3662     \xint@undef\xint@undef\xint@undef\xint@undef
3663     \xint@UNDEF
3664}%
3665 \def\xint@euc@end@ #1#2#3%
3666 {%
3667     \space {#1}{#3}{#2}%
3668}%

```

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}

3669 \def\xintBezoutAlgorithm {\romannumeral0\xintbezoutalgorithm}%
3670 \def\xintbezoutalgorithm #1%
3671 {%
3672     \expandafter
3673     \XINT@bealg
3674     \expandafter
3675     {\romannumeral0\xintabs {#1}}%
3676}%
3677 \def\XINT@bealg #1#2%
3678 {%
3679     \expandafter
3680     \XINT@bealg@fork
3681     \romannumeral0\xintabs {#2}\Z #1\Z
3682}%

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

3683 \def\XINT@bealg@fork #1#2\Z #3#4\Z
3684 {%
3685     \xint@UDzerofork
3686     #1\dummy \XINT@bealg@BisZero
3687     #3\dummy \XINT@bealg@AisZero
3688     0\dummy \XINT@bealg@a
3689     \xint@UDkrof
3690     0{#1#2}{#3#4}1001{{#3#4}{#1#2}}{\Z}
3691}%
3692 \def\XINT@bealg@AisZero #1#2#3\Z{ {1}{0}{0}{1}{#2}{#2}{1}{0}{0}{0}{0}{1}}%
3693 \def\XINT@bealg@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

3694 \def\XINT@bealg@a #1#2#3%
3695 {%
3696     \expandafter
3697     \XINT@bealg@b
3698     \expandafter {\the\numexpr #1+1\expandafter}%
3699     \romannumeral0\XINT@div@prepare {#2}{#3}{#2}%
3700}%

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

```

13 Package *xintgcd* implementation

```

3701 \def\XINT@bealg@b #1#2#3#4#5#6#7#8%
3702 {%
3703   \expandafter\XINT@bealg@c\expandafter
3704     {\romannumeral0\xintadd {\xintMul {#6}{#2}}{#8}}%
3705     {\romannumeral0\xintadd {\xintMul {#5}{#2}}{#7}}%
3706     {#1}{#2}{#3}{#4}{#5}{#6}%
3707 }%
      {\beta(n+1)}{\alpha(n+1)}{n+1}{q(n+1)}{r(n+1)}{r(n)}{\alpha(n)}{\beta(n)}%
3708 \def\XINT@bealg@c #1#2#3#4#5#6%
3709 {%
3710   \expandafter\XINT@bealg@d\expandafter
3711     {#2}{#3}{#4}{#5}{#6}{#1}%
3712 }%
      {\alpha(n+1)}{n+1}{q(n+1)}{r(n+1)}{r(n)}{\beta(n+1)}
3713 \def\XINT@bealg@d #1#2#3#4#5#6#7#8%
3714 {%
3715   \XINT@bealg@e #4\Z {#2}{#4}{#5}{#1}{#6}{#7}{#8}{#3}{#4}{#1}{#6}}%
3716 }%
      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.
3717 \def\XINT@bealg@e #1#2\Z
3718 {%
3719   \xint@zero #1\xint@bealg@end0\XINT@bealg@a
3720 }%
      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}
3721 \def\xint@bealg@end0\XINT@bealg@a #1#2#3#4#5#6#7#8\Z
3722 {%
3723   \expandafter\xint@bealg@end@
3724   \romannumeral0%
3725   \XINT@rord@main {}#8{{#1}{#3}}%
3726   \xint@UNDEF
3727   \xint@undef\xint@undef\xint@undef\xint@undef
3728   \xint@undef\xint@undef\xint@undef\xint@undef
3729   \xint@UNDEF
3730 }%
      {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}
3731 \def\xint@bealg@end@ #1#2#3#4%
3732 {%
3733   \space {#1}{#3}{0}{1}{#2}{#4}{1}{0}%
3734 }%

```

13.9 \xintTypesetEuclideanAlgorithm

TYPESETTING

Organisation:

$\{N\}\{A\}\{D\}\{B\}\{q_1\}\{r_1\}\{q_2\}\{r_2\}\{q_3\}\{r_3\}\dots\{q_N\}\{r_N=0\}$
 $\backslash U_1 = N = \text{nombre d'étapes}, \backslash U_3 = \text{PGCD}, \backslash U_2 = A, \backslash U_4 = B$
 $q_1 = \backslash U_5, q_2 = \backslash U_7 \rightarrow q_n = \backslash U_{2n+3}, r_n = \backslash U_{2n+4}$
 $bn = rn. B = r_0. A = r_{-1}$
 $r(n-2) = q(n)r(n-1) + r(n)$ (n e étape) (n au moins 1)
 $\backslash U_{2n} = \backslash U_{2n+3} \times \backslash U_{2n+2} + \backslash U_{2n+4}, n$ e étape.
 avec n entre 1 et N.

```

3735 \def\xintTypesetEuclideanAlgorithm #1#2%
3736 {% l'algo remplace #1 et #2 par |#1| et |#2|
3737   \par
3738   \begingroup
3739     \xintAssignArray\xintEuclideanAlgorithm {#1}{#2}\to\U
3740     \edef\A{\U2}\edef\B{\U4}\edef\N{\U1}%
3741     \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
3742     \noindent
3743     \count 255 1
3744     \loop
3745       \hbox to \wd 0 {\hfil$\U{\the\numexpr 2*\count 255\relax}$}%
3746       ${} = \U{\the\numexpr 2*\count 255 + 3\relax}
3747       \times \U{\the\numexpr 2*\count 255 + 2\relax}
3748       + \U{\the\numexpr 2*\count 255 + 4\relax}$%
3749       \ifnum \count 255 < \N
3750         \hfill\break
3751         \advance \count 255 1
3752       \repeat
3753   \par
3754   \endgroup
3755 }%

```

13.10 \xintTypesetBezoutAlgorithm

Pour Bezout on a:

$\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}$
 $\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\}\dots\{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$

Donc $4N+8$ termes

$U_1 = N, U_2 = A, U_5 = D, U_6 = B,$
 $q_1 = U_9, q_n = U_{4n+5}, n$ au moins 1
 $r_n = U_{4n+6}, n$ au moins -1
 $\alpha(n) = U_{4n+7}, n$ au moins -1
 $\beta(n) = U_{4n+8}, n$ au moins -1

```

3756 \def\xintTypesetBezoutAlgorithm #1#2%
3757 {%
3758   \par
3759   \begingroup
3760     \parindent0pt
3761     \xintAssignArray\xintBezoutAlgorithm {#1}{#2}\to\BEZ
3762     \edef\A{\BEZ2}\edef\B{\BEZ6}\edef\N{\BEZ1}% A = |#1|, B = |#2|

```



```

3763 \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
3764 \count 255 1
3765 \loop
3766 \noindent
3767 \hbox to \wd 0 {\hfil$\BEZ{\the\numexpr 4*\count 255 - 2\relax}$}%
3768 ${} = \BEZ{\the\numexpr 4*\count 255 + 5\relax}
3769 \times \BEZ{\the\numexpr 4*\count 255 + 2\relax}
3770 + \BEZ{\the\numexpr 4*\count 255 + 6\relax}$\hfill\break
3771 \hbox to \wd 0 {\hfil$\BEZ{\the\numexpr 4*\count 255 + 7\relax}$}%
3772 ${} = \BEZ{\the\numexpr 4*\count 255 + 5\relax}
3773 \times \BEZ{\the\numexpr 4*\count 255 + 3\relax}
3774 + \BEZ{\the\numexpr 4*\count 255 - 1\relax}$\hfill\break
3775 \hbox to \wd 0 {\hfil$\BEZ{\the\numexpr 4*\count 255 + 8\relax}$}%
3776 ${} = \BEZ{\the\numexpr 4*\count 255 + 5\relax}
3777 \times \BEZ{\the\numexpr 4*\count 255 + 4\relax}
3778 + \BEZ{\the\numexpr 4*\count 255 \relax}$
3779 \endgraf
3780 \ifnum \count 255 < \N
3781 \advance \count 255 1
3782 \repeat
3783 \par
3784 \edef\U{\BEZ{\the\numexpr 4*\N + 4\relax}}%
3785 \edef\V{\BEZ{\the\numexpr 4*\N + 3\relax}}%
3786 \edef\D{\BEZ5}%
3787 \ifodd\N\relax
3788 $\U\times\A - \V\times\B = -\D$%
3789 \else
3790 $\U\times\A - \V\times\B = \D$%
3791 \fi
3792 \par
3793 \endgroup
3794 }%
3795 \XINT@gcd@restorecatcodes@endinput%

```