

BigNum

COLLABORATORS

	<i>TITLE :</i> BigNum		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		August 22, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	BigNum	1
1.1	BigNum	1
1.2	_BigNum/BigNumInit »» BASE NAME = _BigNumBase	2
1.3	_BigNum/BigNumFree »» BASE NAME = _BigNumBase	2
1.4	_BigNum/BigNumSetNul »» BASE NAME = _BigNumBase	3
1.5	_BigNum/BigNumSize »» BASE NAME = _BigNumBase	3
1.6	_BigNum/BigNumIsEven »» BASE NAME = _BigNumBase	4
1.7	_BigNum/BigNumAbsBigNum »» BASE NAME = _BigNumBase	4
1.8	_BigNum/BigNumIsNul »» BASE NAME = _BigNumBase	5
1.9	_BigNum/BigNumIsPositive »» BASE NAME = _BigNumBase	5
1.10	_BigNum/BigNumIsNegative »» BASE NAME = _BigNumBase	6
1.11	_BigNum/BigNumAdd »» BASE NAME = _BigNumBase	6
1.12	_BigNum/BigNumDigitAdd »» BASE NAME = _BigNumBase	7
1.13	_BigNum/BigNumDiv »» BASE NAME = _BigNumBase	7
1.14	_BigNum/BigNumDigitDiv »» BASE NAME = _BigNumBase	8
1.15	_BigNum/BigNumRightShift »» BASE NAME = _BigNumBase	9
1.16	_BigNum/BigNumModulo »» BASE NAME = _BigNumBase	9
1.17	_BigNum/BigNumEDiv »» BASE NAME = _BigNumBase	10
1.18	_BigNum/BigNumMul »» BASE NAME = _BigNumBase	10
1.19	_BigNum/BigNumDigitMul »» BASE NAME = _BigNumBase	11
1.20	_BigNum/BigNumLeftShift »» BASE NAME = _BigNumBase	12
1.21	_BigNum/BigNumSub »» BASE NAME = _BigNumBase	12
1.22	_BigNum/BigNumDigitSub »» BASE NAME = _BigNumBase	13
1.23	_BigNum/BigNumDisplay »» BASE NAME = _BigNumBase	13
1.24	_BigNum/BigNumPrint »» BASE NAME = _BigNumBase	14
1.25	_BigNum/BigNumStrToBigNum »» BASE NAME = _BigNumBase	14
1.26	_BigNum/BigNumToStr »» BASE NAME = _BigNumBase	14
1.27	_BigNum/BigNumIntToBigNum »» BASE NAME = _BigNumBase	15
1.28	_BigNum/BigNumToInt »» BASE NAME = _BigNumBase	15
1.29	_BigNum/BigNumCompare »» BASE NAME = _BigNumBase	16

1.30	_BigNum/BigNumFastCompare »» BASE NAME = _BigNumBase	16
1.31	_BigNum/BigNumDiffCarre »» BASE NAME = _BigNumBase	17
1.32	_BigNum/BigNumRho »» BASE NAME = _BigNumBase	18
1.33	_BigNum/BigNumBrutePrime »» BASE NAME = _BigNumBase	18
1.34	_BigNum/BigNumLucasLehmer »» BASE NAME = _BigNumBase	19
1.35	_BigNum/BigNumRnd »» BASE NAME = _BigNumBase	19
1.36	_BigNum/BigNumPgcd »» BASE NAME = _BigNumBase	20
1.37	_BigNum/BigNumPuiModulo »» BASE NAME = _BigNumBase	21
1.38	_BigNum/BigNumSqrt »» BASE NAME = _BigNumBase	21
1.39	_BigNum/BigNumSwap »» BASE NAME = _BigNumBase	22
1.40	_BigNum/BigNumAssign »» BASE NAME = _BigNumBase	22
1.41	_BigNum/BigNumPower2 »» BASE NAME = _BigNumBase	22
1.42	_BigNum/BigNumErrorStatus »» BASE NAME = _BigNumBase	23
1.43	_BigNum/BigNumSquare »» BASE NAME = _BigNumBase	23
1.44	_BigNum/BigNumInfo »» BASE NAME = _BigNumBase	24

Chapter 1

BigNum

1.1 BigNum

```
BASE NAME = BigNumBase
```

```
BigNumAbsBigNum  
BigNumAdd  
BigNumAssign  
BigNumBrutePrime  
BigNumCompare  
BigNumDiffCarre  
BigNumDigitAdd  
BigNumDigitDiv  
BigNumDigitMul  
BigNumDigitSub  
BigNumDisplay  
BigNumDiv  
BigNumEDiv  
BigNumErrorStatus  
BigNumFastCompare  
BigNumFree  
BigNumInfo  
BigNumInit  
BigNumIntToBigNum  
BigNumIsEven  
BigNumIsNegative  
BigNumIsNul  
BigNumIsPositive  
BigNumLeftShift  
BigNumLucasLehmer  
BigNumModulo  
BigNumMul  
BigNumPgcd  
BigNumPower2  
BigNumPrint  
BigNumPuiModulo  
BigNumRho  
BigNumRightShift  
BigNumRnd  
BigNumSetNul  
BigNumSize
```

```

BigNumSqrt
BigNumSquare
BigNumStrToBigNum
BigNumSub
BigNumSwap
BigNumToInt
BigNumToStr

```

1.2 **_BigNum/BigNumInit »» BASE NAME = _BigNumBase**

```

NAME
BigNumInit -- Allocates a new BigNum

SYNOPSIS
big = BigNumInit()
D0

PtrBigNum BigNumInit(void)

FUNCTION
Allocates a new BigNum.
If there is no more memory, an error occurs.
WARNING : BigNumInit will always return a valid pointer.

RESULT
big A BigNum

WARNING
No assumption can be made on the content of this NewBigNum.
BigNumIsNul will always return TRUE, that's all.

SEE ALSO
BigNumFree(), BigNumIsNul(), BigNumErrorStatus()

```

1.3 **_BigNum/BigNumFree »» BASE NAME = _BigNumBase**

```

NAME
BigNumFree -- Unallocates one or more BigNums at a time

SYNOPSIS
BigNumFree( number )
D0

void BigNumFree(int)

FUNCTION
Unallocate one or several BigNums that were previously allocated

Example :

```

```

x=InitBigNum();
y=InitBigNum();
z=InitBigNum();
BigNumFree(2);

```

will free y and z.

INPUTS

number BigNums number to free

WARNING

DO NOT free more BigNum than allocated with BigNumInit()
 When calling CloseLibrary, BigNum.library tests if all the allocated
 BigNum were freed.

SEE ALSO

BigNumInit()

1.4 **_BigNum/BigNumSetNul »» BASE NAME = _BigNumBase**

NAME

BigNumSetNul -- Set a bignum to zero

SYNOPSIS

```

BigNumSetNul( big )
              A0

```

```

void BigNumSetNul(PtrBigNum)

```

FUNCTION

Sets a BigNum to zero.

INPUTS

big The BigNum to set to.

RESULT

big

SEE ALSO

BigNumIsNul()

1.5 **_BigNum/BigNumSize »» BASE NAME = _BigNumBase**

NAME

BigNumSize -- Give the size of a BigNum

SYNOPSIS

```

res = BigNumSize( big )
D0                      A0

```

```
int BigNumSize(PtrBigNum)
```

INPUTS

big The bignum you want to know the size.

RESULT

The size in WORDs

NOTES

The size can be interpreted as the lowest N that verify :

$$\text{abs}(\text{big}) < 32768^N$$

1.6 `_BigNum/BigNumIsEven` »» BASE NAME = `_BigNumBase`

NAME

BigNumIsEven -- Test if a BigNum is even

SYNOPSIS

```
res = BigNumIsEven( big )
D0.w          A0
```

```
short BigNumIsEven(PtrBigNum)
```

FUNCTION

Test if a BigNum is even.

INPUTS

big The BigNum you want to test

RESULT

```
1 if even
0 if odd
```

1.7 `_BigNum/BigNumAbsBigNum` »» BASE NAME = `_BigNumBase`

NAME

BigNumAbsBigNum -- Compute the absolute of a BigNum

SYNOPSIS

```
BigNumAbsBigNum( bigR )
                A0
```

```
void BigNumAbsBigNum (PtrBigNum)
```

FUNCTION

It will perform :
big=abs(big)

INPUTS

bigR The BigNum to get the abs. value.

RESULT

bigR The BigNum to put the abs. value.

NOTES

BigNumAbsBigNum(x,x) can be done.

SEE ALSO

BigNumIspositive(), BigNumAssign()

1.8 **_BigNum/BigNumIsNul** »» **BASE NAME = _BigNumBase**

NAME

BigNumIsNul -- Test if zero

SYNOPSIS

```
res = BigNumIsNul( big )
D0.w          A0
```

```
short BigNumIsNul(PtrBigNum)
```

FUNCTION

Test if a BigNum is equal to zero.

INPUTS

big The BigNum you want to test.

RESULT

1 if big=0
0 otherwise

SEE ALSO

BigNumSetNul()

1.9 **_BigNum/BigNumIsPositive** »» **BASE NAME = _BigNumBase**

NAME

BigNumIsPositive -- Test if positive

SYNOPSIS

```
res = BigNumIsPositive( big )
D0.w          A0
```

```
short BigNumIsPositive(PtrBigNum)
```

FUNCTION

Test if a BigNum is positive.

INPUTS

big The BigNum to test

RESULT

1 if big>=0
0 otherwise

SEE ALSO

BigNumIsNegative()

1.10 `_BigNum/BigNumIsNegative` »» BASE NAME = `_BigNumBase`

NAME

BigNumIsNegative -- Test if negative

SYNOPSIS

res = BigNumIsNegative(big)
D0.w A0

short BigNumIsNegative(PtrBigNum)

FUNCTION

Test if a BigNum is negative.

INPUTS

big The BigNum to test

RESULT

1 if big<0
0 otherwise

BUGS

As you saw, 0 is considered as a positive number. ;)

SEE ALSO

BigNumIsPositive()

1.11 `_BigNum/BigNumAdd` »» BASE NAME = `_BigNumBase`

NAME

BigNumAdd -- Add two BigNums

SYNOPSIS

BigNumAdd(big1 , big2 , bigR)
A0 A1 A2

void BigNumAdd(PtrBigNum,PtrBigNum,PtrBigNum)

FUNCTION

Add two BigNums
bigR=big1+big2

INPUTS
big1, big2 The BigNums you want to Add.

RESULT
bigR = big1 + big2

NOTES
BigNumAdd(x, x, x) is right !

SEE ALSO
BigNumDigitAdd(), BigNumSub()

1.12 `_BigNum/BigNumDigitAdd` »» BASE NAME = `_BigNumBase`

NAME
BigNumDigitAdd -- Add an integer to a BigNum

SYNOPSIS
BigNumDigitAdd(big , number)
A0 D0

void BigNumDigitAdd(PtrBigNum, int)

FUNCTION
Add an integer to a BigNum and return the result in the BigNum.
big+=number

INPUTS
big The BigNum
number The number

RESULT
big = big + number

NOTES
Add(x, 1, x) is allowed.

SEE ALSO
BigNumAdd(), BigNumDigitSub()

1.13 `_BigNum/BigNumDiv` »» BASE NAME = `_BigNumBase`

NAME
BigNumDiv -- Divide two BigNums

SYNOPSIS
BigNumDiv(big1 , big2, bigR)
A0 A1 A2

```
void BigNumDiv(PtrBigNum,PtrBigNum,PtrBigNum)
```

FUNCTION

Compute the following operation :

$bigR = big1 / big2$

INPUTS

$big1, big2$

RESULT

$bigR = big1 / big2$

WARNING

$big1, big2$ and $bigR$ MUST be different BigNums !

$BigNumDiv(x, y, x)$ is unpredictable !!

NOTES

If $big2$ is zero, an error will occur and $BigR$ is set to zero.

SEE ALSO

$BigNumDigitDiv()$, $BigNumEDiv()$, $BigNumModulo()$

1.14 **$_BigNum/BigNumDigitDiv$ »» BASE NAME = $_BigNumBase$**

NAME

$BigNumDigitDiv$ -- Divide a BigNum by an integer

SYNOPSIS

$BigNumDigitDiv(big1 , number , bigR)$

A0 D0 A1

```
void BigNumDigitDiv(PtrBigNum,int,PtrBigNum)
```

FUNCTION

Compute the following operation :

$bigR = big1 / number$

INPUTS

$big1$ The BigNum you want to divide

$number$ The integer divisor

RESULT

$bigR = big1 / number$

NOTES

If $number$ is zero an error will occur and $bigR$ is set to zero.

$DigitDiv(x, 4, x)$ is allowed.

If $number$ is a power of two, use $BigNumRightShift$ for speed.

SEE ALSO

$BigNumDiv()$, $BigNumDigitMul()$, $BigNumRightShift()$

1.15 `_BigNum/BigNumRightShift` »» `BASE NAME = _BigNumBase`

NAME

`BigNumRightShift` -- Shift a `BigNum` to the right

SYNOPSIS

```
BigNumRightShift( big , number )
                A0  D0
```

```
void BigNumRightShift(PtrBigNum,int)
```

FUNCTION

Shift a `BigNum` 'number' times to the left.

INPUTS

`big` The `BigNum` you want to move.
`number` The number of steps.

RESULT

```
big = big >> number
or
big = big / (2^number)
```

NOTES

If `number` equals zero, nothing is done.
 If `number` is negative `BigNumLeftShift` is called.

SEE ALSO

`BigNumLeftShift()`, `BigNumPower2()`

1.16 `_BigNum/BigNumModulo` »» `BASE NAME = _BigNumBase`

NAME

`BigNumModulo` -- Modulo function

SYNOPSIS

```
BigNumModulo( big , modulo , bigR )
              A0      A1      A2
```

```
void BigNumModulo(PtrBigNum,PtrBigNum,PtrBigNum)
                ~~~~~
```

FUNCTION

Compute the modulo function between two `BigNums`.

INPUTS

`big,modulo` Two `BigNums`

RESULT

```
bigR = big % modulo
```

WARNING

The implementation of special cases like negative numbers can be

modified from one version of the library to the other.
You'll be warned !

big1,module and bigR MUST be different BigNums !
BigNumModulo(x,y,x) is unpredictable !!

NOTES
If modulo is zero, an error occurs and bigR is set to zero !

SEE ALSO
BigNumPuiModulo()

1.17 `_BigNum/BigNumEDiv` »» BASE NAME = `_BigNumBase`

NAME
BigNumEDiv -- Compute quotient and remainder

SYNOPSIS
BigNumEDiv(big1 , big2 , bigQ , bigR)
 A0 A1 A2 A3

void BigNumEDiv(PtrBigNum,PtrBigNum,PtrBigNum,PtrBigNum)

FUNCTION
Compute the quotient and the remainder in one pass !

INPUTS
big1,big2 The BigNum to perform the operation

RESULT
bigQ, bigR Quotient and Remainder
big1 = bigR + bigQ x big2

WARNING
The implementation of special cases like negative numbers can be
modified from one version of the library to the other.
You'll be warned !

big1,big2,bigQ and bigR MUST be different BigNums !
BigNumEDiv(x,y,x,t) is unpredictable !!

SEE ALSO
BigNumDiv(), BigNumModulo()

1.18 `_BigNum/BigNumMul` »» BASE NAME = `_BigNumBase`

NAME
BigNumMul -- Multiplication between BigNums

SYNOPSIS

```
BigNumMul( big1 , big2 , bigR )
      A0   A1   A2
```

```
void BigNumMul(PtrBigNum,PtrBigNum,PtrBigNum)
```

FUNCTION
Multiply two BigNums together.

INPUTS
big1, big2 Two BigNums

RESULT
bigR = big1 x big2

WARNING
big1, big2 and bigR MUST be different BigNums !
BigNumMul(x,y,x) is unpredictable !!

If bigR can't contain the BigNum, an error occurs and bigR is set to zero.

NOTES
This function uses a different routine when the numbers are big.
Speed Gain ~30%

SEE ALSO
BigNumDigitMul(), BigNumDiv(), BigNumSquare()

1.19 **_BigNum/BigNumDigitMul** »» **BASE NAME = _BigNumBase**

NAME
BigNumDigitMul -- Multiply a BigNum with an integer

SYNOPSIS
BigNumDigitMul(big , number)
 A0 D0

```
void BigNumDigitMul(PtrBigNum,int)
```

FUNCTION
Multiply a BigNum with an integer.

INPUTS
big | The operandes
number |

RESULT
big = big * number

NOTES
If the result is too big, an error occurs and big is set to zero.

If number is a power of two, use BigNumLeftShift for speed.

SEE ALSO
 BigNumMul(), BigNumDigitDiv(), BigNumLeftShift()

1.20 _BigNum/BigNumLeftShift »» BASE NAME = _BigNumBase

NAME
 BigNumLeftShift -- Shift a BigNum to the left

SYNOPSIS
 BigNumLeftShift(big , number)
 A0 D0

void BigNumLeftShift(PtrBigNum,int)

INPUTS
 big | The operandes
 number |

RESULT
 big = big << number

NOTES
 If the result is too big, an error occurs and big is set to zero.

SEE ALSO
 BigNumRightShift(), BigNumPower2()

1.21 _BigNum/BigNumSub »» BASE NAME = _BigNumBase

NAME
 BigNumSub -- Substracte two BigNums

SYNOPSIS
 BigNumSub(big1 , big2 , bigR)
 A0 A1 A2

void BigNumSub(PtrBigNum,PtrBigNum,PtrBigNum)

FUNCTION
 Substract two BigNums together.

INPUTS
 big1, big2 The operandes

RESULT
 bigR = big1 - big2

NOTES
 BigNumSub(x,x,x) is allowed.

SEE ALSO
 BigNumDigitSub(), BigNumAdd()

1.22 **_BigNum/BigNumDigitSub** »» BASE NAME = **_BigNumBase**

NAME
 BigNumDigitSub -- Subtract an integer from a BigNum

SYNOPSIS
 BigNumDigitSub(big , number)
 A0 D0

void BigNumDigitSub(PtrBigNum,int)

FUNCTION
 Subtract an integer from a BigNum.

INPUTS
 big BigNum | The operands
 number integer |

RESULT
 big = big - number

SEE ALSO
 BigNumSub(), BigNumDigitAdd()

1.23 **_BigNum/BigNumDisplay** »» BASE NAME = **_BigNumBase**

NAME
 BigNumDisplay -- Display debugging information of a BigNum

SYNOPSIS
 BigNumDisplay(big)
 A0

void BigNumDisplay(PtrBigNum)

FUNCTION
 Display a BigNum for debugging purposes.

INPUTS
 big The BigNum to be displayed.

NOTES
 Not for common use !

SEE ALSO
 BigNumPrint()

1.24 `_BigNum/BigNumPrint` »» `BASE NAME = _BigNumBase`

NAME
BigNumPrint -- Print a BigNum

SYNOPSIS
BigNumPrint(big)
A0

void BigNumPrint(PtrBigNum)

FUNCTION
Display a BigNum.

INPUTS
big The BigNum to be displayed.

SEE ALSO
BigNumDisplay()

1.25 `_BigNum/BigNumStrToBigNum` »» `BASE NAME = _BigNumBase`

NAME
BigNumStrToBigNum -- Convert an ASCII string to a BigNum

SYNOPSIS
BigNumStrToBigNum(big , string)
A0 A1

void BigNumStrToBigNum(PtrBigNum, char *)

FUNCTION
Convert an ASCII string to a BigNum.

INPUTS
string The ASCII string

RESULT
big A BigNum.

SEE ALSO
BigNumToStr(), BigNumPrint()

1.26 `_BigNum/BigNumToStr` »» `BASE NAME = _BigNumBase`

NAME
BigNumToStr -- Convert a BigNum to an ASCII string

SYNOPSIS
BigNumToStr(big , string)

A0 A1

```
void BigNumToStr(PtrBigNum, char *)
```

FUNCTION

Convert a BigNum to a string.

INPUTS

big The BigNum you want to convert

RESULT

string The string.

It must be allocated in such a way it can contains ALL
the BigNum

SEE ALSO

BigNumStrToBigNum(), BigNumPrint(), BigNumSize()

1.27 `_BigNum/BigNumIntToBigNum` »» BASE NAME = `_BigNumBase`

NAME

BigNumIntToBigNum -- Convert an integer to a BigNum

SYNOPSIS

```
BigNumIntToBigNum( big , number )
                A0  D0
```

```
void BigNumIntToBigNum(PtrBigNum, int)
```

FUNCTION

Convert an integer to a BigNum.

INPUTS

number The integer you want to convert.

RESULT

big The output BigNum

SEE ALSO

BigNumToInt()

1.28 `_BigNum/BigNumToInt` »» BASE NAME = `_BigNumBase`

NAME

BigNumToInt -- Convert a BigNum to an integer

SYNOPSIS

```
number = BigNumToInt( big )
D0                A0
```

```
int BigNumToInt (PtrBigNum)
```

FUNCTION

Convert a BigNum to an integer.

INPUTS

big The BigNum you want to convert.

RESULT

number The returned integer.

WARNING

If the BigNum is too big, an error occurred and number is set to 0.

SEE ALSO

BigNumIntToBigNum()

1.29 `_BigNum/BigNumCompare` »» BASE NAME = `_BigNumBase`

NAME

BigNumCompare -- Comparison between two BigNums

SYNOPSIS

```
result = BigNumCompare( big1 , big2 )
```

```
D0      A0      A1
```

```
int BigNumCompare (PtrBigNum,PtrBigNum)
```

FUNCTION

Comparison function between two BigNums.

INPUTS

big1, big2 The 2 BigNums.

RESULT

```
result 1 if big1 > big2
```

```
    -1 if big1 < big2
```

```
    0  if big1 = big2
```

SEE ALSO

BigNumFastCompare()

1.30 `_BigNum/BigNumFastCompare` »» BASE NAME = `_BigNumBase`

NAME

BigNumFastCompare -- Comparison between two BigNums

SYNOPSIS

```
result = BigNumFastCompare( big1 , big2 )
```

```
D0.w      A0      A1
```

```
short BigNumFastCompare(PtrBigNum,PtrBigNum)
```

FUNCTION
Comparison function between two BigNums.

INPUTS
big1, big2 The 2 BigNums.

RESULT
result 1 if big1 > big2
-1 if big1 < big2
0 if big1 = big2

NOTES
This function is the same as BigNumCompare except it doesn't take care of the sign.

SEE ALSO
BigNumCompare()

1.31 **_BigNum/BigNumDiffCarre** »» **BASE NAME = _BigNumBase**

NAME
BigNumDiffCarre -- Factorisation function

SYNOPSIS
result = BigNumDiffCarre(big , bigR, limit)
D0.w A0 A1 D0

```
short BigNumDiffCarre(PtrBigNum,PtrBigNum,int)
```

FUNCTION
This function factorise a BigNumber, with the SquareDifference Method.

INPUTS
big The number you want to factorise.
limit Breaking value
A number between 1 and 30
The higher it is, more precise will be the results,
but the delay increases !

RESULT
bigR A factor, if found.
result -1 big is surely Prime
1 big is surely not prime Check bigR
0 big is probably prime

WARNING
There can be a long long delay until the function returns !

NOTES
This method is usefull when $x=AxB$ with A,B approximately the same size.

SEE ALSO
 BigNumRho(), BigNumBrutePrime(), BigNumLucasLehmer()

1.32 `_BigNum/BigNumRho` »» BASE NAME = `_BigNumBase`

NAME
 BigNumRho -- Factorisation function

SYNOPSIS
 result = BigNumRho(big , bigR , limit)
 D0.w A0 A1 D0

short BigNumRho(PtrBigNum,PtrBigNum,int)

FUNCTION
 This function factorise a BigNumber, with the Rho-Pollard Method.

INPUTS
 big The number you want to factorise.
 limit Breaking value
 A number between 1 and 30
 The higher it is, more precise will be the results,
 but the delay increases !

RESULT
 bigR A factor, if found.
 result 1 big is probably prime
 0 big is not prime Check bigR

WARNING
 There can be a long long delay until the function returns !

NOTES
 If it returns 1, the error is about $100 \times (0.25^{\text{limit}}) \%$
 You can use several time this function because it is based
 on random numbers.

SEE ALSO
 BigNumDiffCarre(), BigNumBrutePrime(), BigNumLucasLehmer()

1.33 `_BigNum/BigNumBrutePrime` »» BASE NAME = `_BigNumBase`

NAME
 BigNumBrutePrime -- Factorisation function

SYNOPSIS
 res = BigNumBrutePrime(big, aff)
 D0.w A0 D0

```
short BigNumBrutePrime(PtrBigNum,int)
```

FUNCTION

This function factorise a BigNumber, with the Brute method.

INPUTS

big The number you want to factorise.
 aff if 0 the found factor won't be printed.
 You have no way to find it then.

RESULT

res 1 Factor found big is surely Prime
 0 Nothing. Surely prime.

WARNING

There can be a long long delay until the function returns !
 There are NO breakpoints.

NOTES

DO NOT USE this method for BIG BigNumber, or do it
 before you go on holiday ;)

SEE ALSO

BigNumDiffCarre(), BigNumRho(), BigNumLucasLehmer()

1.34 _BigNum/BigNumLucasLehmer »» BASE NAME = _BigNumBase

NAME

BigNumLucasLehmer -- Perform a LucasLehmer test.

SYNOPSIS

```
result = BigNumLucasLehmer( power )
D0.w                                     D0
```

```
short BigNumLucasLehmer(int)
```

FUNCTION

Perform a LucasLehmer test on $(2^{\text{power}})-1$

INPUTS

power An integer.

RESULT

result 1 if prime
 0 otherwise

SEE ALSO

BigNumRho(), BigNumBrutePrime(), BigNumDiffCarre()

1.35 _BigNum/BigNumRnd »» BASE NAME = _BigNumBase

NAME
BigNumRnd -- Give a random BigNum

SYNOPSIS
BigNumRnd(size , bigR)
D0 A0

void BigNumRnd(int,PtrBigNum)

FUNCTION
Compute a random BigNum between
] 32768^(size-2) ... 32768^(size-1)]

INPUTS
size The size.

RESULT
bigR The generated BigNum

NOTES
If size is too big, it will be round down to the bigger value
BigNum.library can manage.

1.36 **_BigNum/BigNumPgcd »» BASE NAME = _BigNumBase**

NAME
BigNumPgcd -- Find two BigNums GCD

SYNOPSIS
BigNumPgcd(big1 , big2 , bigR)
A0 A1 A2

void BigNumPgcd(PtrBigNum,PtrBigNum,PtrBigNum)

FUNCTION
Find the GCD of big1 and big2 and returns its value in bigR.

INPUTS
big1, big2

RESULT
bigR

WARNING
BigNumPgcd(x,y,x) is not allowed.

NOTES
if big <= 0 or big2 <= 0, bigR is set to 1 and an error occurs.

SEE ALSO
BigNumDiffCarre(), BigNumRho(), BigNumBrutePrime()

1.37 `_BigNum/BigNumPuiModulo` »» `BASE NAME = _BigNumBase`

NAME
`BigNumPuiModulo` -- Compute the modulo power

SYNOPSIS
`BigNumPuiModulo(big , power , modulus , bigR)`
 A0 A1 A2 A3

`void BigNumPuiModulo(PtrBigNum,PtrBigNum,PtrBigNum,PtrBigNum)`

FUNCTION

Compute in a fast way the modulo power.

INPUTS

`big, power, modulus`

RESULT
`BigR = (big ^ power) % modulus`

NOTES
 if `power = 0` and `big = 0`
 or `modulus = 0`, an error occurs and `bigR = 0`.

SEE ALSO
`BigNumModulo()`, `BigNumEDiv()`, `BigNumSquare()`

1.38 `_BigNum/BigNumSqrt` »» `BASE NAME = _BigNumBase`

NAME
`BigNumSqrt` -- Square root function

SYNOPSIS
`res = BigNumSqrt(big , bigR)`
 D0.w A0 A1

`short BigNumSqrt(PtrBigNum,PtrBigNum)`

FUNCTION

Compute the square root of a given `BigNum`.

INPUTS

`big` The `BigNum` you want to extract the root.

RESULT
`bigR` The root
 ret 1 If the root is an exact value
 0 Otherwise

WARNING

If `big` is `< 0`, an error occurs and `bigR` is set to 0.

1.39 `_BigNum/BigNumSwap` »» BASE NAME = `_BigNumBase`

```
NAME
BigNumSwap -- Swap two BigNums

SYNOPSIS
BigNumSwap( big1 , big2 )
            A0      A1

void BigNumSwap(PtrBigNum,PtrBigNum)

FUNCTION
Swap two BigNums.

INPUTS
big1, big2

RESULT
big1 -> big2
big2 -> big1
```

1.40 `_BigNum/BigNumAssign` »» BASE NAME = `_BigNumBase`

```
NAME
BigNumAssign -- Assign a BigNum to another BigNum

SYNOPSIS
BigNumAssign( bigR , big )
              A0      A1

void BigNumAssign(PtrBigNum,PtrBigNum)

FUNCTION
Assign a BigNum to another BigNum.

INPUTS
big

RESULT
bigR = big

NOTES
Use this function instead of copy by hand the BigNum,
for compatibility.
```

1.41 `_BigNum/BigNumPower2` »» BASE NAME = `_BigNumBase`

NAME
BigNumPower2 -- Compute a power of two

SYNOPSIS
BigNumPower2(bigR , power)
 A0 D0

void BigNumPower2(PtrBigNum,int)

FUNCTION
Compute the power of two.

INPUTS
power An integer.

RESULT
bigR = 2^power

NOTES
This is a very speedy routine !

SEE ALSO
BigNumLeftShift(), BigNumRightShift()

1.42 _BigNum/BigNumErrorStatus »» BASE NAME = _BigNumBase

NAME
BigNumErrorStatus -- Return the current error status

SYNOPSIS
res = BigNumErrorStatus()
D0.w

short BigNumErrorStatus(void)

FUNCTION
Return the error status

RESULT
res = 0 if no Error

NOTES
Check the includes to know what's happening

1.43 _BigNum/BigNumSquare »» BASE NAME = _BigNumBase

NAME
BigNumSquare -- Compute the BigNum's square

SYNOPSIS

```
BigNumSquare( bigl, bigR )
              A0   A1
```

```
void BigNumSquare(PtrBigNum,PtrBigNum)
```

FUNCTION

```
Compute bigl*bigl
```

RESULT

```
bigR  = bigl * bigl
```

NOTES

It's fastest to compute a square this way.

This function uses a different routine when the numbers are big.

Speed Gain ~30%

SEE ALSO

```
BigNumMul()
```

1.44 **_BigNum/BigNumInfo »» BASE NAME = _BigNumBase**

NAME

```
BigNumInfo -- Display some informations
```

SYNOPSIS

```
BigNumInfo()
```

```
void BigNumErrorStatus(void)
```

FUNCTION

```
Display some interesting informations.
```
