5 Integer Functions

29

This chapter describes the GMP functions for performing integer arithmetic. These functions

start with the prefix mpz\_.

GMP integers are stored in objects of type mpz\_t.

5.1 Initialization Functions

The functions for integer arithmetic assume that all integer objects are initialized. You do that

by calling the function mpz\_init. For example,

{

mpz\_t integ;

mpz\_init (integ);

...

mpz\_add (integ, ...);

...

mpz\_sub (integ, ...);

/\* Unless the program is about to exit, do ... \*/

mpz\_clear (integ);

}

As you can see, you can store new values any number of times, once an object is initialized.

void mpz\_init (mpz t integer )

Initialize integer, and set its value to 0.

void mpz\_init2 (mpz t integer, unsigned long n )

Initialize integer, with space for n bits, and set its value to 0.

[Function]

[Function]

n is only the initial space, integer will grow automatically in the normal way, if necessary,

for subsequent values stored. mpz\_init2 makes it possible to avoid such reallocations if a

maximum size is known in advance.

void mpz\_clear (mpz t integer )

[Function]

Free the space occupied by integer. Call this function for all mpz\_t variables when you are

done with them.

void mpz\_realloc2 (mpz t integer, unsigned long n )

[Function]

Change the space allocated for integer to n bits. The value in integer is preserved if it fits,

or is set to 0 if not.

This function can be used to increase the space for a variable in order to avoid repeated

automatic reallocations, or to decrease it to give memory back to the heap.

5.2 Assignment Functions

These functions assign new values to already initialized integers (see [Section 5.1 [Initializing](#35)

[Integers], page 29).](#35)

void mpz\_set (mpz t rop, mpz t op )

void mpz\_set\_ui (mpz t rop, unsigned long int op )

void mpz\_set\_si (mpz t rop, signed long int op)

[Function]

[Function]

[Function]

30

void mpz\_set\_d (mpz t rop, double op )

void mpz\_set\_q (mpz t rop, mpq t op )

void mpz\_set\_f (mpz t rop, mpf t op )

Set the value of rop from op.

mpz\_set\_d, mpz\_set\_q and mpz\_set\_f truncate op to make it an integer.

int mpz\_set\_str (mpz t rop, char \*str, int base )

GNU MP 4.3.0

[Function]

[Function]

[Function]

[Function]

Set the value of rop from str, a null-terminated C string in base base. White space is allowed

in the string, and is simply ignored.

The base may vary from 2 to 62, or if base is 0, then the leading characters are used: 0x and

0X for hexadecimal, 0b and 0B for binary, 0 for octal, or decimal otherwise.

For bases up to 36, case is ignored; upper-case and lower-case letters have the same value. For

bases 37 to 62, upper-case letter represent the usual 10..35 while lower-case letter represent

36..61.

This function returns 0 if the entire string is a valid number in base base. Otherwise it returns

−1.

void mpz\_swap (mpz t rop1, mpz t rop2)

Swap the values rop1 and rop2 efficiently.

5.3 Combined Initialization and Assignment Functions

[Function]

For convenience, GMP provides a parallel series of initialize-and-set functions which initialize the

output and then store the value there. These functions’ names have the form mpz\_init\_set...

Here is an example of using one:

{

mpz\_t pie;

mpz\_init\_set\_str (pie, "3141592653589793238462643383279502884", 10);

...

mpz\_sub (pie, ...);

...

mpz\_clear (pie);

}

Once the integer has been initialized by any of the mpz\_init\_set... functions, it can be used

as the source or destination operand for the ordinary integer functions. Don’t use an initialize-

and-set function on a variable already initialized!

void mpz\_init\_set (mpz t rop, mpz t op )

void mpz\_init\_set\_ui (mpz t rop, unsigned long int op )

void mpz\_init\_set\_si (mpz t rop, signed long int op )

void mpz\_init\_set\_d (mpz t rop, double op )

Initialize rop with limb space and set the initial numeric value from op.

int mpz\_init\_set\_str (mpz t rop, char \*str, int base )

[Function]

[Function]

[Function]

[Function]

[Function]

Initialize rop and set its value like mpz\_set\_str (see its documentation above for details).

If the string is a correct base base number, the function returns 0; if an error occurs it returns

−1. rop is initialized even if an error occurs. (I.e., you have to call mpz\_clear for it.)

Chapter 5: Integer Functions

5.4 Conversion Functions

31

This section describes functions for converting GMP integers to standard C types. Functions

for converting to GMP integers are described in [Section 5.2 [Assigning Integers], page 29](#35) and

[Section 5.12 [I/O of Integers], page 39.](#45)

unsigned long int mpz\_get\_ui (mpz t op )

Return the value of op as an unsigned long.

[Function]

If op is too big to fit an unsigned long then just the least significant bits that do fit are

returned. The sign of op is ignored, only the absolute value is used.

signed long int mpz\_get\_si (mpz t op )

[Function]

If op fits into a signed long int return the value of op. Otherwise return the least significant

part of op, with the same sign as op.

If op is too big to fit in a signed long int, the returned result is probably not very useful.

To find out if the value will fit, use the function mpz\_fits\_slong\_p.

double mpz\_get\_d (mpz t op )

Convert op to a double, truncating if necessary (ie. rounding towards zero).

[Function]

If the exponent from the conversion is too big, the result is system dependent. An infinity is

returned where available. A hardware overflow trap may or may not occur.

double mpz\_get\_d\_2exp (signed long int \*exp, mpz t op )

[Function]

Convert op to a double, truncating if necessary (ie. rounding towards zero), and returning

the exponent separately.

The return value is in the range 0.5 ≤ |d| < 1 and the exponent is stored to \*exp . d ∗ 2expis

the (truncated) op value. If op is zero, the return is 0.0 and 0 is stored to \*exp.

This is similar to the standard C frexp function (see [Section “Normalization Functions” in](libc.pdf)

[The GNU C Library Reference Manual).](libc.pdf)

char \* mpz\_get\_str (char \*str, int base, mpz t op )

[Function]

Convert op to a string of digits in base base. The base argument may vary from 2 to 62 or

from −2 to −36.

For base in the range 2..36, digits and lower-case letters are used; for −2..−36, digits and

upper-case letters are used; for 37..62, digits, upper-case letters, and lower-case letters (in

that significance order) are used.

If str is NULL, the result string is allocated using the current allocation function (see

[Chapter 14 [Custom Allocation], page 85](#91)). The block will be strlen(str)+1 bytes, that

being exactly enough for the string and null-terminator.

If str is not NULL, it should point to a block of storage large enough for the result, that being

mpz\_sizeinbase (op, base ) + 2. The two extra bytes are for a possible minus sign, and the

null-terminator.

A pointer to the result string is returned, being either the allocated block, or the given str.

32

5.5 Arithmetic Functions

void mpz\_add (mpz t rop, mpz t op1, mpz t op2 )

void mpz\_add\_ui (mpz t rop, mpz t op1, unsigned long int op2 )

Set rop to op1 + op2 .

void mpz\_sub (mpz t rop, mpz t op1, mpz t op2 )

void mpz\_sub\_ui (mpz t rop, mpz t op1, unsigned long int op2 )

void mpz\_ui\_sub (mpz t rop, unsigned long int op1, mpz t op2 )

Set rop to op1 − op2.

void mpz\_mul (mpz t rop, mpz t op1, mpz t op2 )

void mpz\_mul\_si (mpz t rop, mpz t op1, long int op2 )

void mpz\_mul\_ui (mpz t rop, mpz t op1, unsigned long int op2 )

Set rop to op1 × op2 .

void mpz\_addmul (mpz t rop, mpz t op1, mpz t op2 )

void mpz\_addmul\_ui (mpz t rop, mpz t op1, unsigned long int op2 )

Set rop to rop + op1 × op2 .

void mpz\_submul (mpz t rop, mpz t op1, mpz t op2 )

void mpz\_submul\_ui (mpz t rop, mpz t op1, unsigned long int op2 )

Set rop to rop − op1 × op2 .

void mpz\_mul\_2exp (mpz t rop, mpz t op1, unsigned long int op2 )

GNU MP 4.3.0

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

Set rop to op1 × 2op2. This operation can also be defined as a left shift by op2 bits.

void mpz\_neg (mpz t rop, mpz t op )

Set rop to −op.

void mpz\_abs (mpz t rop, mpz t op )

Set rop to the absolute value of op.

5.6 Division Functions

[Function]

[Function]

Division is undefined if the divisor is zero. Passing a zero divisor to the division or modulo

functions (including the modular powering functions mpz\_powm and mpz\_powm\_ui), will cause an

intentional division by zero. This lets a program handle arithmetic exceptions in these functions

the same way as for normal C int arithmetic.

void mpz\_cdiv\_q (mpz t q, mpz t n, mpz t d )

void mpz\_cdiv\_r (mpz t r, mpz t n, mpz t d )

void mpz\_cdiv\_qr (mpz t q, mpz t r, mpz t n, mpz t d )

unsigned long int mpz\_cdiv\_q\_ui (mpz t q, mpz t n,

unsigned long int d )

unsigned long int mpz\_cdiv\_r\_ui (mpz t r, mpz t n,

unsigned long int d )

unsigned long int mpz\_cdiv\_qr\_ui (mpz t q, mpz t r, mpz t n ,

unsigned long int d )

unsigned long int mpz\_cdiv\_ui (mpz t n, unsigned long int d )

void mpz\_cdiv\_q\_2exp (mpz t q, mpz t n, unsigned long int b )

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

Chapter 5: Integer Functions

void mpz\_cdiv\_r\_2exp (mpz t r, mpz t n, unsigned long int b )

void mpz\_fdiv\_q (mpz t q, mpz t n, mpz t d )

void mpz\_fdiv\_r (mpz t r, mpz t n, mpz t d )

void mpz\_fdiv\_qr (mpz t q, mpz t r, mpz t n, mpz t d )

unsigned long int mpz\_fdiv\_q\_ui (mpz t q, mpz t n,

unsigned long int d )

unsigned long int mpz\_fdiv\_r\_ui (mpz t r, mpz t n,

unsigned long int d )

unsigned long int mpz\_fdiv\_qr\_ui (mpz t q, mpz t r, mpz t n ,

unsigned long int d )

unsigned long int mpz\_fdiv\_ui (mpz t n, unsigned long int d )

void mpz\_fdiv\_q\_2exp (mpz t q, mpz t n, unsigned long int b )

void mpz\_fdiv\_r\_2exp (mpz t r, mpz t n, unsigned long int b )

void mpz\_tdiv\_q (mpz t q, mpz t n, mpz t d )

void mpz\_tdiv\_r (mpz t r, mpz t n, mpz t d )

void mpz\_tdiv\_qr (mpz t q, mpz t r, mpz t n, mpz t d )

unsigned long int mpz\_tdiv\_q\_ui (mpz t q, mpz t n,

unsigned long int d )

unsigned long int mpz\_tdiv\_r\_ui (mpz t r, mpz t n,

unsigned long int d )

unsigned long int mpz\_tdiv\_qr\_ui (mpz t q, mpz t r, mpz t n ,

unsigned long int d )

unsigned long int mpz\_tdiv\_ui (mpz t n, unsigned long int d )

void mpz\_tdiv\_q\_2exp (mpz t q, mpz t n, unsigned long int b )

void mpz\_tdiv\_r\_2exp (mpz t r, mpz t n, unsigned long int b )

33

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

Divide n by d, forming a quotient q and/or remainder r. For the 2exp functions, d = 2b. The

rounding is in three styles, each suiting different applications.

• cdiv rounds q up towards +∞, and r will have the opposite sign to d. The c stands for

“ceil”.

• fdiv rounds q down towards −∞, and r will have the same sign as d. The f stands for

“floor”.

• tdiv rounds q towards zero, and r will have the same sign as n. The t stands for

“truncate”.

In all cases q and r will satisfy n = qd + r, and r will satisfy 0 ≤ |r| < |d|.

The q functions calculate only the quotient, the r functions only the remainder, and the qr

functions calculate both. Note that for qr the same variable cannot be passed for both q and

r, or results will be unpredictable.

For the ui variants the return value is the remainder, and in fact returning the remainder is

all the div\_ui functions do. For tdiv and cdiv the remainder can be negative, so for those

the return value is the absolute value of the remainder.

For the 2exp variants the divisor is 2b. These functions are implemented as right shifts and

bit masks, but of course they round the same as the other functions.

For positive n both mpz\_fdiv\_q\_2exp and mpz\_tdiv\_q\_2exp are simple bitwise right shifts.

For negative n, mpz\_fdiv\_q\_2exp is effectively an arithmetic right shift treating n as twos

complement the same as the bitwise logical functions do, whereas mpz\_tdiv\_q\_2exp effec-

tively treats n as sign and magnitude.

34

void mpz\_mod (mpz t r, mpz t n, mpz t d )

unsigned long int mpz\_mod\_ui (mpz t r, mpz t n, unsigned long int d )

GNU MP 4.3.0

[Function]

[Function]

Set r to n mod d. The sign of the divisor is ignored; the result is always non-negative.

mpz\_mod\_ui is identical to mpz\_fdiv\_r\_ui above, returning the remainder as well as setting

r. See mpz\_fdiv\_ui above if only the return value is wanted.

void mpz\_divexact (mpz t q, mpz t n, mpz t d )

void mpz\_divexact\_ui (mpz t q, mpz t n, unsigned long d )

[Function]

[Function]

Set q to n/d. These functions produce correct results only when it is known in advance that

d divides n.

These routines are much faster than the other division functions, and are the best choice

when exact division is known to occur, for example reducing a rational to lowest terms.

int mpz\_divisible\_p (mpz t n, mpz t d )

int mpz\_divisible\_ui\_p (mpz t n, unsigned long int d )

int mpz\_divisible\_2exp\_p (mpz t n, unsigned long int b )

[Function]

[Function]

[Function]

Return non-zero if n is exactly divisible by d, or in the case of mpz\_divisible\_2exp\_p by 2b.

n is divisible by d if there exists an integer q satisfying n = qd. Unlike the other division

functions, d = 0 is accepted and following the rule it can be seen that only 0 is considered

divisible by 0.

int mpz\_congruent\_p (mpz t n, mpz t c, mpz t d )

int mpz\_congruent\_ui\_p (mpz t n, unsigned long int c, unsigned long int d )

int mpz\_congruent\_2exp\_p (mpz t n, mpz t c, unsigned long int b )

[Function]

[Function]

[Function]

Return non-zero if n is congruent to c modulo d, or in the case of mpz\_congruent\_2exp\_p

modulo 2b.

n is congruent to c mod d if there exists an integer q satisfying n = c + qd. Unlike the other

division functions, d = 0 is accepted and following the rule it can be seen that n and c are

considered congruent mod 0 only when exactly equal.

5.7 Exponentiation Functions

void mpz\_powm (mpz t rop, mpz t base, mpz t exp, mpz t mod )

void mpz\_powm\_ui (mpz t rop, mpz t base, unsigned long int exp, mpz t

mod )

Set rop to baseexpmod mod.

[Function]

[Function]

Negative exp is supported if an inverse base−1modmod exists (see mpz\_invert in [Section 5.9](#41)

[[Number Theoretic Functions], page 35](#41)). If an inverse doesn’t exist then a divide by zero is

raised.

void mpz\_pow\_ui (mpz t rop, mpz t base, unsigned long int exp )

void mpz\_ui\_pow\_ui (mpz t rop, unsigned long int base, unsigned long int

exp )

Set rop to baseexp. The case 00yields 1.

[Function]

[Function]

Chapter 5: Integer Functions

5.8 Root Extraction Functions

int mpz\_root√(mpz t rop, mpz t op, unsigned long int n )

35

[Function]

Set rop to bnopc, the truncated integer part of the nth root of op. Return non-zero if the

computation was exact, i.e., if op is rop to the nth power.

void mpz\_rootrem (mpz t root, mpz t rem, mpz t u, unsigned long int n )

[Function]

Set root to bnuc, the truncated integer part of the nth root of u. Set rem to the remainder,

(u − rootn).

void mpz\_sqrt (mpz t rop, mpz t op )

[Function]

√

Set rop to b

opc, the truncated integer part of the square root of op.

void mpz\_sqrtrem (mpz t rop1, mpz t rop2, mpz t op)

[Function]

√

Set rop1 to b opc, like mpz\_sqrt. Set rop2 to the remainder (op − rop12), which will be

zero if op is a perfect square.

If rop1 and rop2 are the same variable, the results are undefined.

int mpz\_perfect\_power\_p (mpz t op )

[Function]

Return non-zero if op is a perfect power, i.e., if there exist integers a and b, with b > 1, such

that op = ab.

Under this definition both 0 and 1 are considered to be perfect powers. Negative values of

op are accepted, but of course can only be odd perfect powers.

int mpz\_perfect\_square\_p (mpz t op)

[Function]

Return non-zero if op is a perfect square, i.e., if the square root of op is an integer. Under

this definition both 0 and 1 are considered to be perfect squares.

5.9 Number Theoretic Functions

int mpz\_probab\_prime\_p (mpz t n, int reps )

[Function]

Determine whether n is prime. Return 2 if n is definitely prime, return 1 if n is probably

prime (without being certain), or return 0 if n is definitely composite.

This function does some trial divisions, then some Miller-Rabin probabilistic primality tests.

reps controls how many such tests are done, 5 to 10 is a reasonable number, more will reduce

the chances of a composite being returned as “probably prime”.

Miller-Rabin and similar tests can be more properly called compositeness tests. Numbers

which fail are known to be composite but those which pass might be prime or might be

composite. Only a few composites pass, hence those which pass are considered probably

prime.

void mpz\_nextprime (mpz t rop, mpz t op )

Set rop to the next prime greater than op.

[Function]

This function uses a probabilistic algorithm to identify primes. For practical purposes it’s

adequate, the chance of a composite passing will be extremely small.

void mpz\_gcd (mpz t rop, mpz t op1, mpz t op2 )

[Function]

Set rop to the greatest common divisor of op1 and op2. The result is always positive even if

one or both input operands are negative.

36

GNU MP 4.3.0

unsigned long int mpz\_gcd\_ui (mpz t rop, mpz t op1, unsigned long int

op2 )

[Function]

Compute the greatest common divisor of op1 and op2. If rop is not NULL, store the result

there.

If the result is small enough to fit in an unsigned long int, it is returned. If the result does

not fit, 0 is returned, and the result is equal to the argument op1. Note that the result will

always fit if op2 is non-zero.

void mpz\_gcdext (mpz t g, mpz t s, mpz t t, mpz t a, mpz t b )

[Function]

Set g to the greatest common divisor of a and b, and in addition set s and t to coefficients

satisfying as + bt = g. The value in g is always positive, even if one or both of a and b are

negative. The values in s and t are chosen such that |s| ≤ |b| and |t| ≤ |a|.

If t is NULL then that value is not computed.

void mpz\_lcm (mpz t rop, mpz t op1, mpz t op2 )

void mpz\_lcm\_ui (mpz t rop, mpz t op1, unsigned long op2 )

[Function]

[Function]

Set rop to the least common multiple of op1 and op2. rop is always positive, irrespective of

the signs of op1 and op2. rop will be zero if either op1 or op2 is zero.

int mpz\_invert (mpz t rop, mpz t op1, mpz t op2 )

[Function]

Compute the inverse of op1 modulo op2 and put the result in rop. If the inverse exists, the

return value is non-zero and rop will satisfy 0 ≤ rop < op2 . If an inverse doesn’t exist the

return value is zero and rop is undefined.

int mpz\_jacobi (mpz t a, mpz t b )

Calculate the Jacobi symbol(ab. This is defined only for b odd.

int mpz\_legendre (mpz t a, mpz t p )

[Function]

[Function]

Calculate the Legendre symbolap . This is defined only for p an odd positive prime, and

for such p it’s identical to the Jacobi symbol.

int mpz\_kronecker (mpz t a, mpz t b )

int mpz\_kronecker\_si (mpz t a, long b )

int mpz\_kronecker\_ui (mpz t a, unsigned long b )

int mpz\_si\_kronecker (long a, mpz t b )

int mpz\_ui\_kronecker (unsigned long a, mpz t b )

[Function]

[Function]

[Function]

[Function]

[Function]

Calculate the Jacobi symbol(a

b with the Kronecker extension

( a

(a

2=

(2

a when a odd, or

2 = 0 when a even.

When b is odd the Jacobi symbol and Kronecker symbol are identical, so mpz\_kronecker\_ui

etc can be used for mixed precision Jacobi symbols too.

For more information see Henri Cohen section 1.4.2 (see [Appendix B [References], page 121](#127)),

or any number theory textbook. See also the example program ‘demos/qcn.c’ which uses

mpz\_kronecker\_ui.

unsigned long int mpz\_remove (mpz t rop, mpz t op, mpz t f )

[Function]

Remove all occurrences of the factor f from op and store the result in rop. The return value

is how many such occurrences were removed.

void mpz\_fac\_ui (mpz t rop, unsigned long int op )

Set rop to op!, the factorial of op.

[Function]

Chapter 5: Integer Functions

void mpz\_bin\_ui (mpz t rop, mpz t n, unsigned long int k )

void mpz\_bin\_uiui (mpz t rop, unsigned long int n, unsigned long int k )

37

[Function]

[Function]

Compute the binomial coefficient(nkand store the result in rop. Negative values of n are

supported by mpz\_bin\_ui, using the identity(−n

−1)k n+k−1

section 1.2.6 part G.

k= (

k , see Knuth volume 1

void mpz\_fib\_ui (mpz t fn, unsigned long int n )

void mpz\_fib2\_ui (mpz t fn, mpz t fnsub1, unsigned long int n )

[Function]

[Function]

mpz\_fib\_ui sets fn to to Fn, the n’th Fibonacci number. mpz\_fib2\_ui sets fn to Fn, and

fnsub1 to Fn−1.

These functions are designed for calculating isolated Fibonacci numbers. When a sequence of

values is wanted it’s best to start with mpz\_fib2\_ui and iterate the defining Fn+1=Fn+Fn−1

or similar.

void mpz\_lucnum\_ui (mpz t ln, unsigned long int n )

void mpz\_lucnum2\_ui (mpz t ln, mpz t lnsub1, unsigned long int n )

[Function]

[Function]

mpz\_lucnum\_ui sets ln to to Ln, the n’th Lucas number. mpz\_lucnum2\_ui sets ln to Ln, and

lnsub1 to Ln−1.

These functions are designed for calculating isolated Lucas numbers. When a sequence of

values is wanted it’s best to start with mpz\_lucnum2\_ui and iterate the defining Ln+1=

Ln+Ln−1 or similar.

The Fibonacci numbers and Lucas numbers are related sequences, so it’s never necessary

to call both mpz\_fib2\_ui and mpz\_lucnum2\_ui. The formulas for going from Fibonacci to

Lucas can be found in [Section 16.7.5 [Lucas Numbers Algorithm], page 107,](#113) the reverse is

straightforward too.

5.10 Comparison Functions

int mpz\_cmp (mpz t op1, mpz t op2 )

int mpz\_cmp\_d (mpz t op1, double op2 )

int mpz\_cmp\_si (mpz t op1, signed long int op2 )

int mpz\_cmp\_ui (mpz t op1, unsigned long int op2 )

[Function]

[Function]

[Macro]

[Macro]

Compare op1 and op2. Return a positive value if op1 > op2 , zero if op1 = op2 , or a negative

value if op1 < op2 .

mpz\_cmp\_ui and mpz\_cmp\_si are macros and will evaluate their arguments more than once.

mpz\_cmp\_d can be called with an infinity, but results are undefined for a NaN.

int mpz\_cmpabs (mpz t op1, mpz t op2 )

int mpz\_cmpabs\_d (mpz t op1, double op2 )

int mpz\_cmpabs\_ui (mpz t op1, unsigned long int op2 )

[Function]

[Function]

[Function]

Compare the absolute values of op1 and op2. Return a positive value if |op1 | > |op2 |, zero

if |op1 | = |op2 |, or a negative value if |op1 | < |op2 |.

mpz\_cmpabs\_d can be called with an infinity, but results are undefined for a NaN.

int mpz\_sgn (mpz t op )

Return +1 if op > 0, 0 if op = 0, and −1 if op < 0.

[Macro]

This function is actually implemented as a macro. It evaluates its argument multiple times.

38

5.11 Logical and Bit Manipulation Functions

GNU MP 4.3.0

These functions behave as if twos complement arithmetic were used (although sign-magnitude

is the actual implementation). The least significant bit is number 0.

void mpz\_and (mpz t rop, mpz t op1, mpz t op2 )

Set rop to op1 bitwise-and op2.

void mpz\_ior (mpz t rop, mpz t op1, mpz t op2 )

Set rop to op1 bitwise inclusive-or op2.

void mpz\_xor (mpz t rop, mpz t op1, mpz t op2 )

Set rop to op1 bitwise exclusive-or op2.

void mpz\_com (mpz t rop, mpz t op )

Set rop to the one’s complement of op.

unsigned long int mpz\_popcount (mpz t op )

[Function]

[Function]

[Function]

[Function]

[Function]

If op ≥ 0, return the population count of op, which is the number of 1 bits in the binary

representation. If op < 0, the number of 1s is infinite, and the return value is ULONG MAX,

the largest possible unsigned long.

unsigned long int mpz\_hamdist (mpz t op1, mpz t op2 )

[Function]

If op1 and op2 are both ≥ 0 or both < 0, return the hamming distance between the two

operands, which is the number of bit positions where op1 and op2 have different bit values.

If one operand is ≥ 0 and the other < 0 then the number of bits different is infinite, and the

return value is ULONG MAX, the largest possible unsigned long.

unsigned long int mpz\_scan0 (mpz t op, unsigned long int

starting\_bit )

unsigned long int mpz\_scan1 (mpz t op, unsigned long int

starting\_bit )

[Function]

[Function]

Scan op, starting from bit starting bit, towards more significant bits, until the first 0 or 1 bit

(respectively) is found. Return the index of the found bit.

If the bit at starting bit is already what’s sought, then starting bit is returned.

If there’s no bit found, then ULONG MAX is returned. This will happen in mpz\_scan0 past

the end of a negative number, or mpz\_scan1 past the end of a nonnegative number.

void mpz\_setbit (mpz t rop, unsigned long int bit\_index )

Set bit bit index in rop.

void mpz\_clrbit (mpz t rop, unsigned long int bit\_index )

Clear bit bit index in rop.

void mpz\_combit (mpz t rop, unsigned long int bit\_index )

Complement bit bit index in rop.

int mpz\_tstbit (mpz t op, unsigned long int bit\_index )

Test bit bit index in op and return 0 or 1 accordingly.

[Function]

[Function]

[Function]

[Function]

Chapter 5: Integer Functions

5.12 Input and Output Functions

39

Functions that perform input from a stdio stream, and functions that output to a stdio stream.

Passing a NULL pointer for a stream argument to any of these functions will make them read

from stdin and write to stdout, respectively.

When using any of these functions, it is a good idea to include ‘stdio.h’ before ‘gmp.h’, since

that will allow ‘gmp.h’ to define prototypes for these functions.

size\_t mpz\_out\_str (FILE \*stream, int base, mpz t op )

[Function]

Output op on stdio stream stream, as a string of digits in base base. The base argument may

vary from 2 to 62 or from −2 to −36.

For base in the range 2..36, digits and lower-case letters are used; for −2..−36, digits and

upper-case letters are used; for 37..62, digits, upper-case letters, and lower-case letters (in

that significance order) are used.

Return the number of bytes written, or if an error occurred, return 0.

size\_t mpz\_inp\_str (mpz t rop, FILE \*stream, int base )

[Function]

Input a possibly white-space preceded string in base base from stdio stream stream, and put

the read integer in rop.

The base may vary from 2 to 62, or if base is 0, then the leading characters are used: 0x and

0X for hexadecimal, 0b and 0B for binary, 0 for octal, or decimal otherwise.

For bases up to 36, case is ignored; upper-case and lower-case letters have the same value. For

bases 37 to 62, upper-case letter represent the usual 10..35 while lower-case letter represent

36..61.

Return the number of bytes read, or if an error occurred, return 0.

size\_t mpz\_out\_raw (FILE \*stream, mpz t op )

[Function]

Output op on stdio stream stream, in raw binary format. The integer is written in a portable

format, with 4 bytes of size information, and that many bytes of limbs. Both the size and

the limbs are written in decreasing significance order (i.e., in big-endian).

The output can be read with mpz\_inp\_raw.

Return the number of bytes written, or if an error occurred, return 0.

The output of this can not be read by mpz\_inp\_raw from GMP 1, because of changes necessary

for compatibility between 32-bit and 64-bit machines.

size\_t mpz\_inp\_raw (mpz t rop, FILE \*stream )

[Function]

Input from stdio stream stream in the format written by mpz\_out\_raw, and put the result in

rop. Return the number of bytes read, or if an error occurred, return 0.

This routine can read the output from mpz\_out\_raw also from GMP 1, in spite of changes

necessary for compatibility between 32-bit and 64-bit machines.

40

5.13 Random Number Functions

GNU MP 4.3.0

The random number functions of GMP come in two groups; older function that rely on a global

state, and newer functions that accept a state parameter that is read and modified. Please see

the [Chapter 9 [Random Number Functions], page 64](#70) for more information on how to use and

not to use random number functions.

void mpz\_urandomb (mpz t rop, gmp randstate t state, unsigned long int

n )

[Function]

Generate a uniformly distributed random integer in the range 0 to 2n− 1, inclusive.

The variable state must be initialized by calling one of the gmp\_randinit functions

([Section 9.1 [Random State Initialization], page 64)](#70) before invoking this function.

void mpz\_urandomm (mpz t rop, gmp randstate t state, mpz t n )

Generate a uniform random integer in the range 0 to n − 1, inclusive.

[Function]

The variable state must be initialized by calling one of the gmp\_randinit functions

([Section 9.1 [Random State Initialization], page 64)](#70) before invoking this function.

void mpz\_rrandomb (mpz t rop, gmp randstate t state, unsigned long int

n )

[Function]

Generate a random integer with long strings of zeros and ones in the binary representation.

Useful for testing functions and algorithms, since this kind of random numbers have proven

to be more likely to trigger corner-case bugs. The random number will be in the range 0 to

2n− 1, inclusive.

The variable state must be initialized by calling one of the gmp\_randinit functions

([Section 9.1 [Random State Initialization], page 64)](#70) before invoking this function.

void mpz\_random (mpz t rop, mp size t max\_size )

[Function]

Generate a random integer of at most max size limbs. The generated random number doesn’t

satisfy any particular requirements of randomness. Negative random numbers are generated

when max size is negative.

This function is obsolete. Use mpz\_urandomb or mpz\_urandomm instead.

void mpz\_random2 (mpz t rop, mp size t max\_size )

[Function]

Generate a random integer of at most max size limbs, with long strings of zeros and ones

in the binary representation. Useful for testing functions and algorithms, since this kind of

random numbers have proven to be more likely to trigger corner-case bugs. Negative random

numbers are generated when max size is negative.

This function is obsolete. Use mpz\_rrandomb instead.

5.14 Integer Import and Export

mpz\_t variables can be converted to and from arbitrary words of binary data with the following

functions.

void mpz\_import (mpz t rop, size t count, int order, size t size, int

endian, size t nails, const void \*op )

Set rop from an array of word data at op.

[Function]

Chapter 5: Integer Functions

41

The parameters specify the format of the data. count many words are read, each size bytes.

order can be 1 for most significant word first or -1 for least significant first. Within each

word endian can be 1 for most significant byte first, -1 for least significant first, or 0 for the

native endianness of the host CPU. The most significant nails bits of each word are skipped,

this can be 0 to use the full words.

There is no sign taken from the data, rop will simply be a positive integer. An application

can handle any sign itself, and apply it for instance with mpz\_neg.

There are no data alignment restrictions on op, any address is allowed.

Here’s an example converting an array of unsigned long data, most significant element first,

and host byte order within each value.

unsigned long a[20];

mpz\_t

z;

mpz\_import (z, 20, 1, sizeof(a[0]), 0, 0, a);

This example assumes the full sizeof bytes are used for data in the given type, which is

usually true, and certainly true for unsigned long everywhere we know of. However on Cray

vector systems it may be noted that short and int are always stored in 8 bytes (and with

sizeof indicating that) but use only 32 or 46 bits. The nails feature can account for this,

by passing for instance 8\*sizeof(int)-INT\_BIT.

void \* mpz\_export (void \*rop, size t \*countp, int order, size t size, int

endian, size t nails, mpz t op )

Fill rop with word data from op.

[Function]

The parameters specify the format of the data produced. Each word will be size bytes and

order can be 1 for most significant word first or -1 for least significant first. Within each

word endian can be 1 for most significant byte first, -1 for least significant first, or 0 for the

native endianness of the host CPU. The most significant nails bits of each word are unused

and set to zero, this can be 0 to produce full words.

The number of words produced is written to \*countp , or countp can be NULL to discard the

count. rop must have enough space for the data, or if rop is NULL then a result array of

the necessary size is allocated using the current GMP allocation function (see [Chapter 14](#91)

[[Custom Allocation], page 85](#91)). In either case the return value is the destination used, either

rop or the allocated block.

If op is non-zero then the most significant word produced will be non-zero. If op is zero then

the count returned will be zero and nothing written to rop. If rop is NULL in this case, no

block is allocated, just NULL is returned.

The sign of op is ignored, just the absolute value is exported. An application can use mpz\_sgn

to get the sign and handle it as desired. (see [Section 5.10 [Integer Comparisons], page 37](#43))

There are no data alignment restrictions on rop, any address is allowed.

When an application is allocating space itself the required size can be determined with a

calculation like the following. Since mpz\_sizeinbase always returns at least 1, count here

will be at least one, which avoids any portability problems with malloc(0), though if z is

zero no space at all is actually needed (or written).

numb = 8\*size - nail;

count = (mpz\_sizeinbase (z, 2) + numb-1) / numb;

p = malloc (count \* size);

42

5.15 Miscellaneous Functions

int mpz\_fits\_ulong\_p (mpz t op )

int mpz\_fits\_slong\_p (mpz t op )

int mpz\_fits\_uint\_p (mpz t op )

int mpz\_fits\_sint\_p (mpz t op )

int mpz\_fits\_ushort\_p (mpz t op )

int mpz\_fits\_sshort\_p (mpz t op )

GNU MP 4.3.0

[Function]

[Function]

[Function]

[Function]

[Function]

[Function]

Return non-zero iff the value of op fits in an unsigned long int, signed long int, unsigned

int, signed int, unsigned short int, or signed short int, respectively. Otherwise, re-

turn zero.

int mpz\_odd\_p (mpz t op )

int mpz\_even\_p (mpz t op )

[Macro]

[Macro]

Determine whether op is odd or even, respectively. Return non-zero if yes, zero if no. These

macros evaluate their argument more than once.

size\_t mpz\_sizeinbase (mpz t op, int base )

[Function]

Return the size of op measured in number of digits in the given base. base can vary from 2

to 62. The sign of op is ignored, just the absolute value is used. The result will be either

exact or 1 too big. If base is a power of 2, the result is always exact. If op is zero the return

value is always 1.

This function can be used to determine the space required when converting op to a string. The

right amount of allocation is normally two more than the value returned by mpz\_sizeinbase,

one extra for a minus sign and one for the null-terminator.

It will be noted that mpz\_sizeinbase(op,2) can be used to locate the most significant 1 bit

in op, counting from 1. (Unlike the bitwise functions which start from 0, See [Section 5.11](#44)

[[Logical and Bit Manipulation Functions], page 38](#44).)

5.16 Special Functions

The functions in this section are for various special purposes. Most applications will not need

them.

void mpz\_array\_init (mpz t integer\_array, mp size t array\_size,

mp size t fixed\_num\_bits )

[Function]

This is a special type of initialization. Fixed space of fixed num bits is allocated to each of

the array size integers in integer array. There is no way to free the storage allocated by this

function. Don’t call mpz\_clear!

The integer array parameter is the first mpz\_t in the array. For example,

mpz\_t arr[20000];

mpz\_array\_init (arr[0], 20000, 512);

This function is only intended for programs that create a large number of integers and need

to reduce memory usage by avoiding the overheads of allocating and reallocating lots of small

blocks. In normal programs this function is not recommended.

The space allocated to each integer by this function will not be automatically increased, unlike

the normal mpz\_init, so an application must ensure it is sufficient for any value stored. The

following space requirements apply to various routines,

Chapter 5: Integer Functions

43

• mpz\_abs, mpz\_neg, mpz\_set, mpz\_set\_si and mpz\_set\_ui need room for the value they

store.

• mpz\_add, mpz\_add\_ui, mpz\_sub and mpz\_sub\_ui need room for the larger of the two

operands, plus an extra mp\_bits\_per\_limb.

• mpz\_mul, mpz\_mul\_ui and mpz\_mul\_ui need room for the sum of the number of bits in

their operands, but each rounded up to a multiple of mp\_bits\_per\_limb.

• mpz\_swap can be used between two array variables, but not between an array and a

normal variable.

For other functions, or if in doubt, the suggestion is to calculate in a regular mpz\_init variable

and copy the result to an array variable with mpz\_set.

void \* \_mpz\_realloc (mpz t integer, mp size t new\_alloc )

[Function]

Change the space for integer to new alloc limbs. The value in integer is preserved if it fits,

or is set to 0 if not. The return value is not useful to applications and should be ignored.

mpz\_realloc2 is the preferred way to accomplish allocation changes like this. mpz\_realloc2

and \_mpz\_realloc are the same except that \_mpz\_realloc takes its size in limbs.

mp\_limb\_t mpz\_getlimbn (mpz t op, mp size t n )

[Function]

Return limb number n from op. The sign of op is ignored, just the absolute value is used.

The least significant limb is number 0.

mpz\_size can be used to find how many limbs make up op. mpz\_getlimbn returns zero if n

is outside the range 0 to mpz\_size(op )-1.

size\_t mpz\_size (mpz t op )

[Function]

Return the size of op measured in number of limbs. If op is zero, the returned value will be

zero