Ruby trunk - Feature #6065

Allow Bignum marshalling/unmarshalling from C API

02/23/2012 09:18 AM - MartinBosslet (Martin Bosslet)

Status:	Closed
Priority:	Normal
Assignee:	mrkn (Kenta Murata)
Target version:	2.6

Description

Currently, there's no public C API to create a Bignum.
There is rb_big_pack and rb_big_unpack that will do the
job, but they are not portable.

Could we offer public functionality that is independent
of the internal representation for the task of
marshaling/unmarshalling a Bignum to raw C data types?

I'd like to propose something like

- creating a bignum:

VALUE rb_big_from_ary(unsigned long *longs, size_t num_longs, int signed)

- retrieving a representation of a Bignum (longs are allocated):

size_t rb_big_to_ary(VALUE big, unsigned long **longs, int *signed)

For getting a representation, rb_big2str could also be used,
but the above would simplify things when developing an extension
that is in need of Bignum support.

Names and signatures are of course open for discussion,
the example should just serve as an indication of what
I'm aiming at.

To avoid ambiguity, it would have to be defined how
the longs are ordered and how the signed flag is to be
interpreted - I would suggest a very simple
representation: Let "longs" be the representation of
the absolute value of the bignum in little- or big-endian
order, where each of the longs themselves should probably
be in the same order, in order to eliminate ambivalence.
Signed is either 0 or 1, so no two's complement or anything
involved.

I would volunteer to provide a patch for this if we would
agree on something.

Associated revisions

Revision 8fd992d7 - 07/27/2013 03:11 PM - akr (Akira Tanaka)

- include/ruby/intern.h (rb_integer_pack): Declaration moved from internal.h. (rb_integer_unpack): Ditto. [ruby-core:42813] [Feature #6065]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@42202 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 42202 - 07/27/2013 03:11 PM - akr (Akira Tanaka)

- include/ruby/intern.h (rb_integer_pack): Declaration moved from internal.h. (rb_integer_unpack): Ditto. [ruby-core:42813] [Feature #6065]

Revision 42202 - 07/27/2013 03:11 PM - akr (Akira Tanaka)

- include/ruby/intern.h (rb_integer_pack): Declaration moved from internal.h. (rb_integer_unpack): Ditto. [ruby-core:42813] [Feature #6065]

Revision 42202 - 07/27/2013 03:11 PM - akr (Akira Tanaka)

- include/ruby/intern.h (rb_integer_pack): Declaration moved from internal.h. (rb_integer_unpack): Ditto. [ruby-core:42813] [Feature #6065]

Revision 42202 - 07/27/2013 03:11 PM - akr (Akira Tanaka)

- include/ruby/intern.h (rb_integer_pack): Declaration moved from internal.h. (rb_integer_unpack): Ditto. [ruby-core:42813] [Feature #6065]

Revision 42202 - 07/27/2013 03:11 PM - akr (Akira Tanaka)

- include/ruby/intern.h (rb_integer_pack): Declaration moved from internal.h. (rb_integer_unpack): Ditto. [ruby-core:42813] [Feature #6065]

Revision 42202 - 07/27/2013 03:11 PM - akr (Akira Tanaka)

- include/ruby/intern.h (rb_integer_pack): Declaration moved from internal.h. (rb_integer_unpack): Ditto. [ruby-core:42813] [Feature #6065]

Revision e5ff9d58 - 07/28/2013 02:14 AM - akr (Akira Tanaka)

- include/ruby/intern.h (rb_absint_size): Declaration moved from internal.h to calculate required buffer size to pack integers. (rb_absint_numwords): Ditto. (rb_absint_singlebit_p): Ditto. [ruby-core:42813] [Feature #6065]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@42208 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 42208 - 07/28/2013 02:14 AM - akr (Akira Tanaka)

- include/ruby/intern.h (rb_absint_size): Declaration moved from internal.h to calculate required buffer size to pack integers. (rb_absint_numwords): Ditto. (rb_absint_singlebit_p): Ditto. [ruby-core:42813] [Feature #6065]

Revision 42208 - 07/28/2013 02:14 AM - akr (Akira Tanaka)

- include/ruby/intern.h (rb_absint_size): Declaration moved from internal.h to calculate required buffer size to pack integers. (rb_absint_numwords): Ditto. (rb_absint_singlebit_p): Ditto. [ruby-core:42813] [Feature #6065]

Revision 42208 - 07/28/2013 02:14 AM - akr (Akira Tanaka)

- include/ruby/intern.h (rb_absint_size): Declaration moved from internal.h to calculate required buffer size to pack integers. (rb_absint_numwords): Ditto. (rb_absint_singlebit_p): Ditto. [ruby-core:42813] [Feature #6065]

Revision 42208 - 07/28/2013 02:14 AM - akr (Akira Tanaka)

- include/ruby/intern.h (rb_absint_size): Declaration moved from internal.h to calculate required buffer size to pack integers. (rb_absint_numwords): Ditto. (rb_absint_singlebit_p): Ditto. [ruby-core:42813] [Feature #6065]

Revision 42208 - 07/28/2013 02:14 AM - akr (Akira Tanaka)

- include/ruby/intern.h (rb_absint_size): Declaration moved from internal.h to calculate required buffer size to pack integers. (rb_absint_numwords): Ditto. (rb_absint_singlebit_p): Ditto. [ruby-core:42813] [Feature #6065]

Revision 42208 - 07/28/2013 02:14 AM - akr (Akira Tanaka)

- include/ruby/intern.h (rb_absint_size): Declaration moved from internal.h to calculate required buffer size to pack integers. (rb_absint_numwords): Ditto. (rb_absint_singlebit_p): Ditto. [ruby-core:42813] [Feature #6065]

History

#1 - 02/23/2012 09:27 AM - naruse (Yui NARUSE)

- Status changed from Open to Assigned

- Assignee changed from MartinBosslet (Martin Bosslet) to mrkn (Kenta Murata)

I made such dump API before.
This dump a bignum as a format for OpenSSL::BN.

```
diff --git a/bignum.c b/bignum.c
index 3145f24..9141dc8 100644
--- a/bignum.c
+++ b/bignum.c
@@ -405,6 +405,90 @@ rb_big_unpack(unsigned long *buf, long num_longs)
}
}

+/*
 * dump an absolute value of an integer as a big endian byte string.
 */ +VALUE +rb_big_bindump(VALUE val) +{
long clen;
char *str;
val = rb_to_int(val);
if (FIXNUM_P(val)) {
long v = FIX2LONG(val);
int i;
for (clen = SIZEOF_LONG; clen >= 0 && !((v >> ((clen-1)*CHAR_BIT)) & 0xFF); clen--);
str = ALLOC_N(char, clen+1);
for (i = clen; i; i--) {
str[clen - i] = (v >> ((i-1) * CHAR_BIT)) & 0xFF;
}
}
else {
long len = RBIGNUM_LEN(val);
BDIGIT *ds = BDIGITS(val);
BDIGIT v = ds[len-1];
int i, j; +
for (i = SIZEOF_BDIGITS; i >= 0 && !((v >> ((i-1)*CHAR_BIT)) & 0xFF); i--);
clen = i + (len-1) * SIZEOF_BDIGITS;
str = ALLOCA_N(char, clen+1);
str[clen] = 0; +
for (j = 0; j < i; j++) {
str[j] = (char)((v >> ((i - j - 1)*CHAR_BIT)) & 0xFF);
} +
for (i = len - 2; i >= 0; i--) {
v = ds[i];
for (j = 0; j < SIZEOF_BDIGITS; j++) {
str[clen - (i + 1) * SIZEOF_BDIGITS + j] =
(char)((v >> ((SIZEOF_BDIGITS-1-j)* CHAR_BIT)) & 0xFF);
}
}
}
return rb_str_new(str, clen); +} + +/*
 * load an absolute value of an integer from a big endian byte string.
 */ +VALUE +rb_big_binload(VALUE dummy, VALUE val) +{
char *cstr = RSTRING_PTR(val);
long blen = RSTRING_LEN(val);
int i; +
if (blen < SIZEOF_BDIGITS ||
(blen == SIZEOF_BDIGITS && !(cstr[0]>>(CHAR_BIT-2)))) {
long v = 0;
for (i = 0; i < blen; i++) {
v <<= CHAR_BIT;
v |= cstr[i];
}
return LONG2NUM(v);
}
else {
long len = (blen + SIZEOF_BDIGITS - 1) / SIZEOF_BDIGITS;
VALUE big = bignew(len, 1);
BDIGIT *ds = BDIGITS(big);
int n = 0;
BDIGIT v = 0; +
for (i = 0; i < blen; i++) {
int j = i % SIZEOF_BDIGITS;
v |= cstr[blen - 1 - i] << (CHAR_BIT * j);
if (j == SIZEOF_BDIGITS - 1) {
ds[n] = v;
v = 0;
n += 1;
}
}
if (v) ds[n] = v;
```

- return big;
- } +} + #define QUAD_SIZE 8

#if SIZEOF_LONG_LONG == QUAD_SIZE && SIZEOF_BDIGITS*2 == SIZEOF_LONG_LONG
@@ -3478,7 +3562,10 @@ Init_Bignum(void)
{
rb_cBignum = rb_define_class("Bignum", rb_cInteger);

- rb_define_singleton_method(rb_cBignum, "binload", rb_big_binload, 1); + rb_define_method(rb_cBignum, "to_s", rb_big_to_s, -1);
- rb_define_method(rb_cBignum, "bindump", rb_big_bindump, 0); rb_define_method(rb_cBignum, "coerce", rb_big_coerce, 1); rb_define_method(rb_cBignum, "-@", rb_big_uminus, 0); rb_define_method(rb_cBignum, "+", rb_big_plus, 1);

#2 - 02/23/2012 09:53 AM - akr (Akira Tanaka)

2012/2/23 Martin Bosslet Martin.Bosslet@googlemail.com:

> Currently, there's no public C API to create a Bignum.
> There is rb_big_pack and rb_big_unpack that will do the
> job, but they are not portable.

How are they not portable?
--
Tanaka Akira

#3 - 02/23/2012 10:07 AM - MartinBosslet (Martin Bosslet)

Yui NARUSE wrote:

> I made such dump API before.
> This dump a bignum as a format for OpenSSL::BN.

Great, yes, this looks like what I wanted :) I can
determine whether the number was signed using
RBIGSIGN(), which is public - but for the other
way round, when creating a negative Bignum,
rb_big_uminus would also have to become public?

#4 - 02/23/2012 10:12 AM - MartinBosslet (Martin Bosslet)

Akira Tanaka wrote:

> 2012/2/23 Martin Bosslet Martin.Bosslet@googlemail.com:
>
> > Currently, there's no public C API to create a Bignum.
> > There is rb_big_pack and rb_big_unpack that will do the
> > job, but they are not portable.
>
> How are they not portable?
> --
> Tanaka Akira

Sorry, "not portable" was probably the wrong wording. I meant
I can't use them e.g. from Rubinius because they're not part
of the public API and they rely on machine endianness.

#5 - 02/23/2012 12:23 PM - akr (Akira Tanaka)

2012/2/23 Martin Bosslet Martin.Bosslet@googlemail.com:

> > Currently, there's no public C API to create a Bignum.
> > There is rb_big_pack and rb_big_unpack that will do the
> > job, but they are not portable.
>
> How are they not portable?

> Sorry, "not portable" was probably the wrong wording. I meant
> I can't use them e.g. from Rubinius because they're not part
> of the public API and they rely on machine endianness.

I think your proposal also rely on machine endianness because
it use "long" type.

I guess bytes in long type (4 bytes or 8 bytes in usual) is
native endian. Am I wrong?
--
Tanaka Akira

#6 - 02/23/2012 01:59 PM - akr (Akira Tanaka)

2012/2/23 Tanaka Akira akr@fsij.org:

> I think your proposal also rely on machine endianness because
> it use "long" type.
>
> I guess bytes in long type (4 bytes or 8 bytes in usual) is
> native endian. Am I wrong?

Oops. [ruby-core:42813] describes "where each of the longs
hemselves should probably be in the same order".

I think unsigned long should be used only for native endian.
unsigned char should be used instead for big- or little- endian data.
--
Tanaka Akira

#7 - 02/23/2012 08:18 PM - MartinBosslet (Martin Bosslet)

Akira Tanaka wrote:

> 2012/2/23 Tanaka Akira akr@fsij.org:
>
> > I think your proposal also rely on machine endianness because
> > it use "long" type.
> >
> > I guess bytes in long type (4 bytes or 8 bytes in usual) is
> > native endian. Am I wrong?
>
> Oops. [ruby-core:42813] describes "where each of the longs
> hemselves should probably be in the same order".
>
> I think unsigned long should be used only for native endian.
> unsigned char should be used instead for big- or little- endian data.
> --

Yes, you're right, when using (unsigned) longs we have to pay
attention to byte order within the long itself or go through
the pain of normalizing them to some pre-defined order.

That's why I'm all for Yui's proposal now. As long as
sizeof(char) stays one byte, Yui's solution only requires
specifying the overall byte order once, if I'm not overlooking
something.

I can work around this currently by using for example hex
representations of the numbers for (un-)marshaling, but having
#bindump and #binload would make things a lot easier and more
efficient.

Would it be OK to add #bindump, #binload and rb_big_uminus
to the public API?

#8 - 02/23/2012 10:59 PM - mrkn (Kenta Murata)

I also believe it is useful that the feature to dump a Bignum to C array.

I made a patch for realizing the feature.
Please check this gist https://gist.github.com/1892968

If Matz approve the patch, I will commit that.

#9 - 02/23/2012 11:23 PM - akr (Akira Tanaka)

2012/2/23 Martin Bosslet Martin.Bosslet@googlemail.com:

> Would it be OK to add #bindump, #binload and rb_big_uminus
> to the public API?

I don't understand rb_big_pack/rb_big_unpack is not enough.

Although rb_big_pack/rb_big_unpack uses long and depends on native endian,
it is not too difficult to convert endian in C.
--
Tanaka Akira

#10 - 02/25/2012 05:34 AM - MartinBosslet (Martin Bosslet)

Kenta Murata wrote:

> I also believe it is useful that the feature to dump a Bignum to C array.
>
> I made a patch for realizing the feature.
> Please check this gist https://gist.github.com/1892968
>
> If Matz approve the patch, I will commit that.

Great, I would really appreciate that, thank you!

One comment, which is also in reply to why I believe it
could be dangerous to simply promote rb_big_(un)pack to
public API: they are tightly coupled to our internal
representation of Bignums. We should probably explicitly
define the format to be expected of the long array. This
way we'll separate our internal representation from the
format that is actually exchanged. This would allow us to
change the representation internally without breaking
compatibility in the API layer each time we do so.

So even if for now it would suffice to simply memcpy
our internal representation to the array, we should
already fix the format to avoid problems in the
future?

#11 - 02/25/2012 09:23 AM - mame (Yusuke Endoh)

Hello,

2012/2/25 Martin Bosslet Martin.Bosslet@googlemail.com:

> Kenta Murata wrote:
>
> > I also believe it is useful that the feature to dump a Bignum to C array.
> >
> > I made a patch for realizing the feature.
> > Please check this gist https://gist.github.com/1892968

Objection. You missed Martin's point. Martin proposes "portable"
mechanism of bignum import/export. For your use case, you can just
use RBIGNUM_DIGITS instead of rb_big_dump_to_cary.

I agree with akr; Martin's proposal depends on the type long, which
is not "portable" enough. For designing the APIs, I think we should
refer gmplib, which is a giant in this area.

http://gmplib.org/manual/Integer-Import-and-Export.html

--
Yusuke Endoh mame@tsg.ne.jp

#12 - 02/25/2012 10:29 AM - akr (Akira Tanaka)

2012/2/25 Martin Bosslet Martin.Bosslet@googlemail.com:

> One comment, which is also in reply to why I believe it could be dangerous to simply promote rb_big_(un)pack to public API: rhey are tightly coupled to our internal representation of Bignums. We should probably explicitly define the format to be expected of the long array. This way we'll separate our internal representation from the format that is actually exchanged. This would allow us to change the representation internally without breaking compatibility in the API layer each time we do so.

I don't think rb_big_pack/rb_big_unpack is tightly coupled to internal of Bignum.

What the points they expose internal of Bignum?

Note that the format is written in a comment in bignum.c.

```
/*
* buf is an array of long integers.
* buf is ordered from least significant word to most significant word.
* buf[0] is the least significant word and
* buf[num_longs-1] is the most significant word.
* This means words in buf is little endian.
* However each word in buf is native endian.
* (buf[i]&1) is the least significant bit and
* (buf[i]&(1<<(SIZEOF_LONG*CHAR_BIT-1))) is the most significant bit
* for each 0 <= i < num_longs.
* So buf is little endian at whole on a little endian machine.
* But buf is mixed endian on a big endian machine.
*
* The buf represents negative integers as two's complement.
* So, the most significant bit of the most significant word,
* (buf[num_longs-1]>>(SIZEOF_LONG*CHAR_BIT-1)),
* is the sign bit: 1 means negative and 0 means zero or positive.
*
* If given size of buf (num_longs) is not enough to represent val,
* higier words (including a sign bit) are ignored.
*/
void
rb_big_pack(VALUE val, unsigned long *buf, long num_longs)

/* See rb_big_pack comment for endianness and sign of buf. */
VALUE
rb_big_unpack(unsigned long *buf, long num_longs)
```

--
Tanaka Akira

#13 - 02/26/2012 12:51 AM - MartinBosslet (Martin Bosslet)

Akira Tanaka wrote:

> 2012/2/25 Martin Bosslet Martin.Bosslet@googlemail.com:
>
> I don't think rb_big_pack/rb_big_unpack is tightly coupled to internal of Bignum.
>
> What the points they expose internal of Bignum?

I meant that it would become tightly coupled without further comment on the expected transfer format. If the internal representation happens to be exactly the same as the one we hand to externals, and there is no further comment, this part of the interface could be easily overlooked when we change something with the internal representation in the future. I think basically we are on the same side, it's just me having trouble to express myself precisely enough :)

I wanted to point out the fact that even if internal and external representation would happen to be the same we should already treat them as two separate things in our mind and be aware that the external representation would be fixed the moment we publish the functionality in public API.

> Note that the format is written in a comment in bignum.c.

You're right, the comment there would already specify the format, but I think what Yusuke proposed (copying gmplib's approach) would be the perfect solution in my eyes, suiting anybody's needs - a client can simply choose what format they want to have. Additionally, its dynamic nature would already enforce a clear separation between internal and external representation.

#14 - 02/26/2012 12:58 AM - MartinBosslet (Martin Bosslet)

Yusuke Endoh wrote:

> I agree with akr; Martin's proposal depends on the type long, which is not "portable" enough. For designing the APIs, I think we should refer gmplib, which is a giant in this area.
>
> http://gmplib.org/manual/Integer-Import-and-Export.html

Having something like that would of course be pure luxury :) I really like it because it is friendly to clients in the sense that they can simply specify any format they need in their particular case and won't have to bother any further with endianness or raw data type. If you need longs, you get longs, if you need unsigned char, you can also have that - sounds very appealing to me.

Although this will be more work to implement I would assume this is the perfect solution indeed? What do you think?

#15 - 03/05/2012 11:47 AM - naruse (Yui NARUSE)

2012/2/23 Martin Bosslet Martin.Bosslet@googlemail.com:

> Akira Tanaka wrote:
>
> > 2012/2/23 Martin Bosslet Martin.Bosslet@googlemail.com:
> >
> > > Currently, there's no public C API to create a Bignum.
> > > There is rb_big_pack and rb_big_unpack that will do the
> > > job, but they are not portable.
> >
> > How are they not portable?
> > --
> > Tanaka Akira
>
> Sorry, "not portable" was probably the wrong wording. I meant I can't use them e.g. from Rubinius because they're not part of the public API and they rely on machine endianness.

String#to_s is current workaround.
ruby -ropenssl -e'p OpenSSL::BN.new((1<<64).to_s(16), 16)'

--
NARUSE, Yui naruse@airemix.jp

#16 - 11/20/2012 09:14 PM - mame (Yusuke Endoh)

- Target version changed from 2.0.0 to 2.6

#17 - 06/05/2013 10:23 PM - akr (Akira Tanaka)

2012/2/25 Yusuke Endoh mame@tsg.ne.jp:

> I agree with akr; Martin's proposal depends on the type long, which is not "portable" enough. For designing the APIs, I think we should refer gmplib, which is a giant in this area.
>
> http://gmplib.org/manual/Integer-Import-and-Export.html

At first time I see that, I guess there is no application of "nails" argument.

Now I think I know several applications:

- Integer#to_s(radix) for radix is power of 2
  For example, i.to_s(2) can be implemented using that with size=1 and
  nails=7 and
  0 and 1 are converted to '0' and '1' later.

BER-compressed integer for pack (size=1 and nails=1)

Tanaka Akira

#18 - 06/10/2013 08:23 PM - akr (Akira Tanaka)

2013/6/5 Tanaka Akira akr@fsij.org:

> At first time I see that, I guess there is no application of "nails" argument.
>
> Now I think I know several applications:
>
> - Integer#to_s(radix) for radix is power of 2
>   For example, i.to_s(2) can be implemented using that with size=1 and
>   nails=7 and
>   0 and 1 are converted to '0' and '1' later.
>
> - BER-compressed integer for pack (size=1 and nails=1)

I implemented rb_integer_pack and rb_integer_unpack in ruby trunk.
(It is declared in internal.h. So it is not public now.)

I designed them as follows:

```
/* "MS" in MSWORD and MSBYTE means "most significant" /
/ "LS" in LSWORD and LSBYTE means "least significant" /
/ For rb_integer_pack and rb_integer_unpack: /
#define INTEGER_PACK_MSWORD_FIRST       0x01
#define INTEGER_PACK_LSWORD_FIRST       0x02
#define INTEGER_PACK_MSBYTE_FIRST       0x10
#define INTEGER_PACK_LSBYTE_FIRST       0x20
#define INTEGER_PACK_NATIVE_BYTE_ORDER  0x40
/ For rb_integer_unpack: /
#define INTEGER_PACK_FORCE_BIGNUM       0x100
/ Combinations: */
#define INTEGER_PACK_LITTLE_ENDIAN \
(INTEGER_PACK_LSWORD_FIRST | \
INTEGER_PACK_LSBYTE_FIRST)
#define INTEGER_PACK_BIG_ENDIAN \
(INTEGER_PACK_MSWORD_FIRST | \
INTEGER_PACK_MSBYTE_FIRST)

/*
* Export an integer into a buffer.
*
* This function fills the buffer specified by words and numwords as
* abs(val) in the format specified by wordsize, nails and flags.
*
* [val] Fixnum, Bignum or another integer like object which has
to_int method.
* [words] buffer to export abs(val).
* [numwords] the size of given buffer as number of words.
* [wordsize] the size of word as number of bytes.
* [nails] number of padding bits in a word.
*   Most significant nails bits of each word are filled by zero.
* [flags] bitwise or of constants which name starts "INTEGER_PACK_".
*   It specifies word order and byte order.
*
* This function returns the signedness and overflow condition as follows:
*   -2 : negative overflow.  val <= -2*(numwords(wordsize*CHAR_BIT-nails))
*   -1 : negative without overflow.
-2*(numwords(wordsize*CHAR_BIT-nails)) < val < 0
```

```
*   0 : zero.  val == 0
*   1 : positive without overflow.  0 < val <
2*(numwords(wordsize*CHAR_BIT-nails))
*   2 : positive overflow.  2*(numwords(wordsize*CHAR_BIT-nails)) <= val
*
* The least significant words of abs(val) are filled in the buffer
when overflow occur.
*/
int rb_integer_pack(VALUE val, void *words, size_t numwords, size_t
wordsize, size_t nails, int flags);

/*
* Import an integer into a buffer.
*
* [sign] signedness of the value.
*   -1 for non-positive.  0 or 1 for non-negative.
* [words] buffer to import.
* [numwords] the size of given buffer as number of words.
* [wordsize] the size of word as number of bytes.
* [nails] number of padding bits in a word.
*   Most significant nails bits of each word are ignored.
* [flags] bitwise or of constants which name starts "INTEGER_PACK_".
*   It specifies word order and byte order.
*   Also, INTEGER_PACK_FORCE_BIGNUM specifies that the result will
be a Bignum
*   even if it is representable as a Fixnum.
*
* This function returns the imported integer as Fixnum or Bignum.
*/
VALUE rb_integer_unpack(int sign, const void *words, size_t
numwords, size_t wordsize, size_t nails, int flags);
```

I also implemented two functions to calculate the require buffer size.

```
/*
* Calculate a number of bytes to be required to represent
* the absolute value of the integer given as val.
*
* [val] an integer.
* [nlz_bits_ret] number of leading zero bits in the most
significant byte is returned if not NULL.
*
* This function returns ((val_numbits * CHAR_BIT + CHAR_BIT - 1) / CHAR_BIT)
* where val_numbits is the number of bits of abs(val).
* This function should not overflow.
*
* If nlz_bits_ret is not NULL,
* (return_value * CHAR_BIT - val_numbits) is stored in *nlz_bits_ret.
* In this case, 0 <= *nlz_bits_ret < CHAR_BIT.
*
*/
size_t rb_absint_size(VALUE val, int *nlz_bits_ret);

/*
* Calculate a number of words to be required to represent
* the absolute value of the integer given as val.
*
* [val] an integer.
* [word_numbits] number of bits in a word.
* [nlz_bits_ret] number of leading zero bits in the most
significant word is returned if not NULL.
*
* This function returns ((val_numbits * CHAR_BIT + word_numbits -
1) / word_numbits)
* where val_numbits is the number of bits of abs(val).
* If it overflows, (size_t)-1 is returned.
*
* If nlz_bits_ret is not NULL and overflow is not occur,
* (return_value * word_numbits - val_numbits) is stored in *nlz_bits_ret.
* In this case, 0 <= *nlz_bits_ret < word_numbits.
*
*/
size_t rb_absint_numwords(VALUE val, size_t word_numbits, size_t
*nlz_bits_ret);
```

Any opinions about the API?

Note that packing/unpacking BER-compressed integer uses them now and
they are much faster (for very big integers).

200% time ./ruby -e '[3*200000].pack("w")'
./ruby -e '[3200000].pack("w")' 3.14s user 0.00s system 99% cpu 3.147 total
trunk% time ./ruby -e '[3200000].pack("w")'
./ruby -e '[3*200000].pack("w")' 0.02s user 0.01s system 95% cpu 0.029 total

200% time ./ruby -e '("\x81"*100000+"\x00").unpack("w")'
./ruby -e '("\x81"*100000+"\x00").unpack("w")' 4.21s user 0.03s
system 99% cpu 4.251 total
trunk% time ./ruby -e '("\x81"*100000+"\x00").unpack("w")'
./ruby -e '("\x81"*100000+"\x00").unpack("w")' 0.02s user 0.00s
system 88% cpu 0.023 total

Integer#to_s(power-of-2) is also bit faster now.

200% time ./ruby -e 'v = 3*100000; 1000.times { v.to_s(16) }'
./ruby -e 'v = 3100000; 1000.times { v.to_s(16) }' 4.77s user
0.01s system 99% cpu 4.787 total
trunk% time ./ruby -e 'v = 3100000; 1000.times { v.to_s(16) }'
./ruby -e 'v = 3*100000; 1000.times { v.to_s(16) }' 3.16s user
0.01s system 99% cpu 3.184 total

Versions:
200% ./ruby -v
ruby 2.0.0p214 (2013-06-09 revision 41193) [x86_64-linux]
trunk% ./ruby -v
ruby 2.1.0dev (2013-06-10 trunk 41214) [x86_64-linux]
--
Tanaka Akira

#19 - 07/06/2013 07:53 AM - MartinBosslet (Martin Bosslet)

akr (Akira Tanaka) wrote:

> 2013/6/5 Tanaka Akira akr@fsij.org:
>
> Any opinions about the API?

Wow, that looks nice, I need to give that a spin soon!

#20 - 07/28/2013 12:11 AM - akr (Akira Tanaka)

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r42202.
Martin, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- include/ruby/intern.h (rb_integer_pack): Declaration moved from internal.h. (rb_integer_unpack): Ditto. [ruby-core:42813] [Feature #6065]

#21 - 07/28/2013 12:15 AM - akr (Akira Tanaka)

I made rb_integer_pack and rb_integer_unpack public because it seems no one against them.

#22 - 07/28/2013 11:19 AM - akr (Akira Tanaka)

I made rb_absint_size and rb_absint_numwords public too, as described in
[ruby-core:55408]. (comment #18)

I also defined rb_absint_singlebit_p which is required to calculate
required buffer size to pack integer as a two's complement number.

/* Test abs(val) consists only a bit or not.
*

- Returns 1 if abs(val) == 1 << n for some n >= 0.

- Returns 0 otherwise. *
- rb_absint_singlebit_p can be used to determine required buffer size
- for rb_integer_pack used with INTEGER_PACK_2COMP (two's complement). *
- Following example calculates number of bits required to
- represent val in two's complement number, without sign bit. *
- size_t size;
- int neg = FIXNUM_P(val) ? FIX2LONG(val) < 0 : RBIGNUM_NEGATIVE_P(val);
- size = rb_absint_numwords(val, 1, NULL)
- if (size == (size_t)-1) ...overflow...
- if (neg && rb_absint_singlebit_p(val))
- size--; *
- Following example calculates number of bytes required to
- represent val in two's complement number, with sign bit. *
- size_t size;
- int neg = FIXNUM_P(val) ? FIX2LONG(val) < 0 : RBIGNUM_NEGATIVE_P(val);
- int nlz_bits;
- size = rb_absint_size(val, &nlz_bits);
- if (nlz_bits == 0 && !(neg && rb_absint_singlebit_p(val)))
- size++; */ int rb_absint_singlebit_p(VALUE val);