64-bit representation of a real number (long real)

1985's IEEE (Institute for Electrical and Electronic Engineers) report specifies 64-bit (binary digit) representation for a real number.
The first bit is a sign indicator: s is 0 for a positive and 1 for a negative number.
This is followed by 11-bit exponent (called a characteristic): c, and 52-bit binary fraction, f, called the mantissa.
Here, the base of exponent is 2.

52 binary digits correspond to between 16 and 17 decimal digits; thus we can

expect at least 16 decimal digits precision. The exponent of 11 binary digits corresponds

to a range between

```
> 0*2^10+0*2^9+0*2^8+0*2^7+0*2^6+0*2^5+0*2^4+0*2^3+0*2^2+0*2^1+0*
  2^0;
```

$$0 \tag{1}$$

and

```
> 1*2^10+1*2^9+1*2^8+1*2^7+1*2^6+1*2^5+1*2^4+1*2^3+1*2^2+1*2^1+1*
  2^0;
```

$$2047 \tag{2}$$

Also note that

```
> 2^11-1;
```

$$2047 \tag{3}$$

However, we also want a good representation for numbers with small magnitudes. To insure that 1023 is substracted from the characteristic, so the range of exponents varies from -1023 to 1024. Using a normalization for a unique representation, the floating-point number is of the form

$$(-1)^s\, 2^{c-1023}\,(1+f)$$

For the machine number

0 10000000011

1011100100010000000000000000000000000000000000000000

```
> s:=0;
```

$$s := 0 \tag{4}$$

so the number is positive. The characteristic

```
> c:=1*2^10+0*2^9+0*2^8+0*2^7+0*2^0+0*2^5+0*2^4+0*2^3+0*2^2+1*
  2^1+1*2^0;
```

$$c := 1027 \tag{5}$$

The exponential part is

```
> 2^(c-1023);
```

$$16 \tag{6}$$

and

```
> f:=1*(1/2)^(1)+1*(1/2)^3+1*(1/2)^4+1*(1/2)^5+1*(1/2)^8+1*(1/2)
  ^12;
```

$$f := \frac{2961}{4096} \tag{7}$$

so the above machine number has the decimal representation

```
> (-1)^s*2^(c-1023)*(1+f);
```

$$\frac{7057}{256} \tag{8}$$

```
> evalf(%);
```

$$27.56640625 \tag{9}$$

The next smallest machine number, NS, is

0 10000000011
1011100100001111111111111111111111111111111111111111

and the next largest machine number, NL, is

0 10000000011
1011100100010000000000000000000000000000000000000001

The mantissa of the former is

```
> f1:=1*(1/2)^(1)+1*(1/2)^3+1*(1/2)^4+1*(1/2)^5+1*(1/2)^8+1*(1/2)
  ^13+1*(1/2)^14+1*(1/2)^15+1*(1/2)^16+1*(1/2)^17+1*(1/2)^18+1*
```

```
(1/2)^19+1*(1/2)^20+1*(1/2)^21+1*(1/2)^22+1*(1/2)^23+1*(1/2)
^24+1*(1/2)^25+1*(1/2)^26+1*(1/2)^27+1*(1/2)^28+1*(1/2)^29+1*
(1/2)^30+1*(1/2)^31+1*(1/2)^32+1*(1/2)^33+1*(1/2)^34+1*(1/2)
^35+1*(1/2)^36+1*(1/2)^37+1*(1/2)^38+1*(1/2)^39+1*(1/2)^40+1*
(1/2)^41+1*(1/2)^42+1*(1/2)^43+1*(1/2)^44+1*(1/2)^45+1*(1/2)
^46+1*(1/2)^47+1*(1/2)^48+1*(1/2)^49+1*(1/2)^50+1*(1/2)^51+1*
(1/2)^52;
```

$$f1 := \frac{3255653929844735}{4503599627370496} \quad \textbf{(10)}$$

while the mantissa of latter is

```
> f2:=1*(1/2)^(1)+1*(1/2)^3+1*(1/2)^4+1*(1/2)^5+1*(1/2)^8+1*(1/2)
^12+1*(1/2)^52;
```

$$f2 := \frac{3255653929844737}{4503599627370496} \quad \textbf{(11)}$$

The decimal representation of the next smallest machine number is

```
> NS:=(-1)^s*2^(c-1023)*(1+f1);
```

$$NS := \frac{7759253557215231}{281474976710656} \quad \textbf{(12)}$$

```
> Digits:=50;
```

$$Digits := 50 \quad \textbf{(13)}$$

```
> evalf(NS);
```

$$27.566406249999996447286321199499070644378662109375 \quad \textbf{(14)}$$

The decimal representation of the next largest machine number is

```
> NL:=(-1)^s*2^(c-1023)*(1+f2);
```

$$NL := \frac{7759253557215233}{281474976710656} \quad \textbf{(15)}$$

```
> evalf(%);
```

$$27.566406250000003552713678800500929355621337890625 \quad \textbf{(16)}$$

```
> NL-NS;
```

$$\frac{1}{140737488355328} \quad \textbf{(17)}$$

```
> evalf(%);
```

$$7.1054273576010018587112426757812500000000000000000 \times 10^{-15} \quad \textbf{(18)}$$

The smallest positive number is

0 00000000000
0000000000000000000000000000000000000000000000000001

```
> smallest:=2^(-1023)*(1+2^(-52));
```

smallest := 4503599627370497 /
404804506614621236704990693437834614099113299528284236713802716054860679135990693783920767402874248990374155728633623822779617474771586953734026799881477019843034848553132722728933815484186432682479535356945490137124014966849385397236206711298319112681620113024717539104666829230461005064372655017292012526615415482186989568 **(19)**

```
> Digits:=800;
```

Digits := 800 **(20)**

```
> evalf(smallest);
```

1.1125369292536009385779392792894741203940044243418520978065650156059844301998003482648952146106314429319518506835140954008585648036355955177563613706587576099527870021569402283901616688769940819588693664479623037114635056539026906698540826680648223724764894760609489545391926291682950925894809399942575213757391318038010840218110155646350227416036982422856551956112981967804161220311948453638445093358527274637586993294662405200869114164125622897532827869095519004323455807914359994854323646610724898485773353360199895995404580173812990192997712369923839430590047536255771881194801858107585864908005772302179765642162703220969322662452694568897840457902396202549613706927137471310271320204419918459593709086493896670139621383772282614543769341253209859132766723632812500000000000000000000000000000000 × 10^{-308} **(21)**

which is approximately 10^{-308}

The largest positive machine number is

0 11111111111
1111111111111111111111111111111111111111111111111111

```
> largest:=2^1024*(1-2^(-52));
```

largest := **(22)**
179769313486231550856124328384506240234343437157459335924404872448581845754556114388470639943126220321960804027157371570809852884964511743044087662767600909594331927728237078876188760579532563768698654064825262115771015791463983014857704008123419459386245141723703148097529108423358883457665451722744025579520

```
> evalf(%);
```

1.797693134862315508561243283845062402343434371574593359244048724485\ **(23)**

8184575455611438847063994312622032196080402715737157080985288496 4\
5117430440876627676009095943319277282370788761887605795325637686 9\
8654064825262115771015791463983014857704008123419459386245141723 7\
03148097529108423358883457665451722744025579520 × 10^{308}

which is approximately 10^{308}

Numbers occurring in calculations that are smaller than $2^{-1023}\left(1+2^{-52}\right)$

are treated as zero.

Floating-point representations in Maple are easy done. For example,

```
> Digits:=10;
```

$$Digits := 10 \quad \textbf{(24)}$$

causes all arithmetic to be rounded to 100 digits. For instance, *fl(fl(x)+fl(y))* is performed using 100-digit rounding arithmetic by

```
> evalf(evalf(x)+evalf(y));
```

$$x + y \quad \textbf{(25)}$$

Implementing t-digit chopping aritmetic is slightly more complicated.

```
> chop:=proc(x,t)
  local e, x2;
  if x=0 then 0
  else
  e:= trunc(evalf(log10(abs(x))));
  if e>0 then e:=e+1 fi;
  x2:=evalf(trunc(x*10^(t-e))*10^(e-t))
  fi
  end:
> chop(12.226,4);
```

$$12.22000000 \quad \textbf{(26)}$$

```
> Digits:=10;
```

$$Digits := 10 \quad \textbf{(27)}$$

Solving a quadratic equation

```
> solve({x^2+62.10*x+1},{x});
```

$$\{x = -0.01610723741\}, \{x = -62.08389276\} \quad \textbf{(28)}$$

```
> Digits:=4;
```

(29)

$$Digits := 4 \tag{29}$$

```
> sqrt((62.10)^2-4*1.0*1.0);
```

$$62.06 \tag{30}$$

```
> floatx1:=(-62.10+62.06)/2.0;
```

$$floatx1 := -0.02000 \tag{31}$$

has large relative error: 2.4x10^(-1)

```
> floatx2:=(-62.10-62.06)/2.0;
```

$$floatx2 := -62.10 \tag{32}$$

while floatx2 has the small relative error: 3.2x10^(-4). In order to obtain more accurate approximation for floatx1 one can note that

$$\left(x1 = \frac{-b + \sqrt{b^2 - 4\,a\,c}}{2\,a}\right) = -\frac{2\,c}{b + \sqrt{b^2 - 4\,a\,c}}$$

Using this formula floatx1 is given by

```
> floatx1:=-2.0/(62.10+62.06);
```

$$floatx1 := -0.01610 \tag{33}$$

which has the small relative error: 6.2x10^(-4).