Appendix A

Number Theory Tools for Floating-Point Arithmetic

A.1 Continued Fractions

Continued fractions make it possible to build very good (indeed, the best possible, in a sense that will be made explicit by Theorems A.1 and A.2) rational approximations to real numbers. As such, they naturally appear in many problems of number theory, discrete mathematics, and computer science. Since floating-point numbers are rational approximations to real numbers, it is not surprising that continued fractions play a role in some areas of floating-point arithmetic. A typical example is Table 10.1: that table gives, among other results, the floating-point numbers that are nearest to an integer multiple of $\pi/2$, which is a typical continued-fraction problem. It allows one to design accurate range-reduction algorithms for evaluating trigonometric functions. Another example is the continued-fraction-based proof of the algorithm for performing correctly rounded multiplication by an arbitrary-precision constant, presented in Section 4.11.

Excellent surveys can be found in [237, 565, 334]. Here, we will just present some general definitions, as well as the few results that are needed in this book, especially in Chapters 4 and 10.

Let $\alpha$ be a real number. From $\alpha$, consider the two sequences $(a_i)$ and $(r_i)$ defined by

$$
\begin{align*}
    r_0 &= \alpha, \\
    a_i &= \lfloor r_i \rfloor, \\
    r_{i+1} &= \frac{1}{r_i - a_i},
\end{align*}
$$

(A.1)
Appendix A. Number Theory Tools

where \(\lfloor . \rfloor\) is the usual floor function. Note that the \(a_i\)'s are integers and that the \(r_i\)'s are real numbers.

If \(\alpha\) is an irrational number, then these sequences are defined for any \(i \geq 0\) (i.e., \(r_i\) is never equal to \(a_i\)), and the rational number

\[
\frac{P_i}{Q_i} = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \ddots + \frac{1}{a_i}}}}
\]

is called the \(i\)-th convergent of \(\alpha\). The \(a_i\)'s constitute the continued fraction expansion of \(\alpha\). We write

\[
\alpha = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \ddots}}}
\]

or, to save space,

\[
\alpha = [a_0; a_1, a_2, a_3, \ldots].
\]

If \(\alpha\) is rational, then these sequences terminate for some \(k\), and \(P_k/Q_k = \alpha\) exactly. The \(P_i\)'s and the \(Q_i\)'s can be deduced from the \(a_i\)'s using the following recurrences:

\[
\begin{align*}
P_0 &= a_0, \\
Q_0 &= 1, \\
P_1 &= a_1a_0 + 1, \\
Q_1 &= a_1, \\
P_k &= P_{k-1}a_k + P_{k-2} \quad \text{for } k \geq 2, \\
Q_k &= Q_{k-1}a_k + Q_{k-2} \quad \text{for } k \geq 2.
\end{align*}
\]

Note that these recurrences give irreducible fractions \(P_i/Q_i\); the values \(P_i\) and \(Q_i\) that are deduced from them satisfy \(\gcd(P_i, Q_i) = 1\).

The major interest in the continued fractions lies in the fact that \(P_i/Q_i\) is the best rational approximation to \(\alpha\) among all rational numbers of denominator less than or equal to \(Q_i\). More precisely, we have the following two results \[237\].

**Theorem A.1.** Let \((P_j/Q_j)_{j \geq 0}\) be the convergents of \(\alpha\). If a rational number \(P/Q\) is a better approximation to \(\alpha\) than \(P_k/Q_k\) (namely, if \(|P/Q - \alpha| < |P_k/Q_k - \alpha|\)), then \(Q > Q_k\).

**Theorem A.2.** Let \((P_j/Q_j)_{j \geq 0}\) be the convergents of \(\alpha\). If \(Q_{k+1}\) exists, then for any \((P, Q) \in \mathbb{Z} \times \mathbb{N}^*\), with \(Q < Q_{k+1}\), we have

\[
|P - \alpha Q| \geq |P_k - \alpha Q_k|.
\]
If \( Q_{k+1} \) does not exist (which implies that \( \alpha \) is rational), then the previous inequality holds for any \((P, Q) \in \mathbb{Z} \times \mathbb{N}^*\).

Interestingly enough, a kind of converse result exists: if a rational approximation to some number \( \alpha \) is extremely good, then it must be a convergent of its continued fraction expansion.

**Theorem A.3** ([237]). Let \( P, Q \) be integers, \( Q \neq 0 \). If
\[
\left| \frac{P}{Q} - \alpha \right| < \frac{1}{2Q^2},
\]
then \( P/Q \) is a convergent of \( \alpha \).

An example of continued fraction expansion of an irrational number is
\[
e = \exp(1) = 2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{\ddots}}}}},
\]
which gives the following rational approximations to \( e \):
\[
\frac{P_0}{Q_0} = 2, \quad \frac{P_1}{Q_1} = 3, \quad \frac{P_2}{Q_2} = \frac{8}{3}, \quad \frac{P_3}{Q_3} = \frac{11}{4}, \quad \frac{P_4}{Q_4} = \frac{19}{7}, \quad \frac{P_5}{Q_5} = \frac{87}{32}.
\]

Other examples are
\[
\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{292 + \cfrac{1}{\ddots}}}}, \quad \sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{\ddots}}}}.
\]
A.2 Euclidean Lattices

A Euclidean lattice is a set of points that are regularly spaced in the Euclidean space $\mathbb{R}^n$ (see Definition A.1). It is a discrete algebraic object that is encountered in several domains of various sciences, including mathematics, computer science, electrical engineering, and chemistry. It is a rich and powerful modeling tool, thanks to the deep and numerous theoretical results, algorithms, and implementations available (see [90, 112, 226, 397, 557, 599] for example).

Applications of Euclidean lattices to floating-point arithmetic include the calculation of polynomial approximations with coefficients that are floating-point numbers (or double-word numbers) [68, 97] and the search for hardest-to-round points of elementary functions (see Chapter 10 and [571]).

Let $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$. We set

$$\|x\|_2 = (x|x)^{1/2} = (x_1^2 + \cdots + x_n^2)^{1/2} \quad \text{and} \quad \|x\|_\infty = \max_{1 \leq i \leq n} |x_i|.$$

**Definition A.1.** Let $L$ be a nonempty subset of $\mathbb{R}^n$. The set $L$ is a (Euclidean) lattice if there exists a set of $\mathbb{R}$-linearly independent vectors $b_1, \ldots, b_d$ such that

$$L = \mathbb{Z} \cdot b_1 \oplus \cdots \oplus \mathbb{Z} \cdot b_d = \left\{ \sum_{i \leq d} x_i \cdot b_i, x_i \in \mathbb{Z} \right\}.$$

The family $(b_1, \ldots, b_d)$ is a basis of the lattice $L$, and $d$ is called the rank of the lattice $L$.

For example, the set $\mathbb{Z}^n$ and all of its additive subgroups are lattices. These lattices play a central role in computer science since they can be represented exactly. We say that a lattice $L$ is integer (resp. rational) when $L \subseteq \mathbb{Z}^n$ (resp. $\mathbb{Q}^n$). An integer lattice of rank 2 is given in Figure A.1, as well as one of its bases.

A lattice is often given by one of its bases (in practice, a matrix whose rows or columns are the basis vectors). Unfortunately, as soon as the rank of the lattice is greater than 1, there are infinitely many such representations for any given lattice. In Figure A.2, we give another basis of the lattice of Figure A.1.

**Proposition A.1.** If $(e_1, \ldots, e_k)$ and $(f_1, \ldots, f_j)$ are two families of $\mathbb{R}$-linearly independent (column) vectors that generate the same lattice, then $k = j$ (this is the rank of the lattice), and there exists a $(k \times k)$ matrix $M$ with integer coefficients and determinant equal to $\pm 1$ such that $(e_i) = (f_i) \cdot M$.

Among the infinitely many bases of a specified lattice (if $k \geq 2$), some are more interesting than others. One can define various notions of what a
A.2. Euclidean Lattices

Figure A.1: The lattice \( \mathbb{Z}(2, 0) \oplus \mathbb{Z}(1, 2) \).

Figure A.2: Two bases of the lattice \( \mathbb{Z}(2, 0) \oplus \mathbb{Z}(1, 2) \).

“good” basis is, but most of the time, it is required to consist of somewhat short vectors.

The two most famous computational problems related to lattices are the shortest and closest vector problems (SVP and CVP). Since a lattice is discrete, it contains a vector of smallest nonzero norm. That norm is denoted by \( \lambda \) and is called the minimum of the lattice. Note that the minimum is reached at least twice (a vector and its opposite), and may be reached more times. The discreteness also implies that, given an arbitrary vector of the space, there always exists a lattice vector closest to it (note that there can be several such vectors). We now state the search versions of SVP and CVP.

**Problem A.1. Shortest vector problem (SVP).** Given a basis of a lattice \( L \subseteq \mathbb{Q}^n \), find a shortest nonzero vector of \( L \), i.e., a vector of norm \( \lambda(L) \).

SVP naturally leads to the following approximation problem, which we call \( \gamma \)-SVP, where \( \gamma \) is a function of the rank only: given a basis of a lattice \( L \subseteq \mathbb{Q}^n \), find \( b \in L \) such that

\[
0 < \|b\| \leq \gamma \cdot \lambda(L).
\]
Problem A.2. Closest vector problem (CVP). Given a basis of a lattice $L \subseteq \mathbb{Q}^n$ and a target vector $t \in \mathbb{Q}^n$, find $b \in L$ such that $\|b - t\| = \text{dist}(t, L)$.

CVP naturally leads to the following approximation problem, which we call $\gamma$-CVP, where $\gamma$ is a function of the rank only: given a basis of a lattice $L \subseteq \mathbb{Q}^n$ and a target vector $t \in \mathbb{Q}^n$, find $b \in L$ such that

$$\|b - t\| \leq \gamma \cdot \text{dist}(t, L).$$

Note that SVP and CVP can be defined with respect to any norm of $\mathbb{R}^n$, and we will write SVP$_2$ and CVP$_2$ to explicitly refer to the Euclidean norm. These two computational problems have been studied extensively. We very briefly describe some of the results, and refer to [420] for more details.

Ajtai [4] showed in 1998 that the decisional version of SVP$_2$ (i.e., given a lattice $L$ and a scalar $x$, compare $\lambda(L)$ and $x$) is NP-hard under randomized polynomial reductions. The NP-hardness had been conjectured in the early 1980s by van Emde Boas [612], who proved the result for the infinity norm instead of the Euclidean norm. Khot [335] showed that Ajtai’s result still holds for the decisional version of the relaxed problem $\gamma$-SVP$_2$, where $\gamma$ is an arbitrary constant. Goldreich and Goldwasser [216] proved that, under very plausible complexity theory assumptions, approximating SVP$_2$ within a factor $\gamma = \sqrt{d / \ln d}$ is not NP-hard, where $d$ is the rank of the lattice. No polynomial-time algorithm is known for approximating SVP$_2$ within a factor $f(d)$ with $f$ a polynomial in $d$. On the constructive side, Kannan [325] described an algorithm that solves SVP$_2$ in time

$$d^{\frac{d(1+o(1))}{2e}} \approx d^{0.184d},$$

and in polynomial space (the complexity bound is proved in [233]). Ajtai, Kumar, and Sivakumar [5] gave an algorithm of complexity $2^{O(d)}$ both in time and space that solves SVP with high probability. Becker et al. [34] described a heuristic variant of that algorithm that runs in time $\approx \gamma^{0.292d}$. Similar results also hold for the infinity norm instead of the Euclidean norm.

In 1981, van Emde Boas [612] proved that the decisional version of CVP$_2$ is NP-hard (see also [419]). On the other hand, Goldreich and Goldwasser [216] showed, under very plausible assumptions, that approximating CVP$_2$ within a factor $\sqrt{d / \ln d}$ is not NP-hard. Similar results also hold for the infinity norm (see [488]). Unfortunately, no polynomial-time algorithm is known for approximating CVP to a polynomial factor. On the constructive side, Kannan [325] described an algorithm that solves CVP in time

$$d^{\frac{d(1+o(1))}{2}}$$

for the Euclidean norm and

$$d^{d(1+o(1))}$$

for the infinity norm (see [233] for the proofs of the complexity bounds).
A.2. Euclidean Lattices

If we sufficiently relax the parameter $\gamma$, the situation becomes far better. In 1982, Lenstra, Lenstra, and Lovász \cite{LLL} gave an algorithm that allows one to get relatively short vectors in polynomial time. Their algorithm is now commonly referred to by the acronym LLL.

**Theorem A.4 (LLL \cite{LLL}).** Given an arbitrary basis $(a_1, \ldots, a_d)$ of a lattice $L \subseteq \mathbb{Z}^n$, the LLL algorithm provides a basis $(b_1, \ldots, b_d)$ of $L$ that is made of relatively short vectors. Among others, we have $\|b_1\| \leq 2^{(d-1)/2} \cdot \lambda(L)$. Furthermore, LLL terminates within $O(d^5 n \ln^3 A)$ bit operations with $A = \max_i \|a_i\|$.

More precisely, the LLL algorithm computes what is called a $\delta$-LLL-reduced basis, where $\delta$ is a fixed parameter which belongs to $(1/4, 1)$ (if $\delta$ is omitted, then its value is $3/4$, which is the historical choice). To define what a LLL-reduced basis is, we need to recall the Gram–Schmidt orthogonalization of a basis. Consider a basis $(b_1, \ldots, b_d)$. Its Gram–Schmidt orthogonalization $(b_1^*, \ldots, b_d^*)$ is defined recursively as follows:

- the vector $b_1^*$ is $b_1$;
- for $i > 1$, we set $b_i^* = b_i - \sum_{j<i} \mu_{i,j} b_j^*$, where $\mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{\|b_j^*\|^2}$.

Geometrically, the vector $b_i^*$ is the projection of the vector $b_i$ orthogonally to the span of the previous basis vectors $b_1, \ldots, b_{i-1}$. We say that the basis $(b_1, \ldots, b_d)$ is $\delta$-LLL-reduced if the following two conditions are satisfied:

- for any $i > j$, the quantity $\mu_{i,j}$ has magnitude less than or equal to $1/2$. This condition is called the size-reduction condition;
- for any $i$, we have $\delta \|b_i^*\|^2 \leq \|b_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|b_i^*\|^2$. This condition is called Lovász’s condition. It means that orthogonally to the first $i - 1$ vectors, the $(i + 1)$-th vector cannot be arbitrarily small compared to the $i$-th vector.

LLL-reduced bases have many interesting properties. The most important one is probably that the first basis vector cannot be more than $2^{(d-1)/2}$ times longer than the lattice minimum. The LLL algorithm computes an LLL-reduced basis $(b_1, \ldots, b_d)$ of the lattice spanned by $(a_1, \ldots, a_d)$ by incrementally trying to make the LLL conditions satisfied. It uses an index $k$, which starts at 2 and eventually reaches $d + 1$. At any moment, the first $k - 1$ vectors satisfy the LLL conditions, and we are trying to make the first $k$ vectors satisfy the conditions. To make the size-reduction condition satisfied for the $k$-th vector, one subtracts from it an adequate integer linear combination of the vectors $b_1, \ldots, b_{k-1}$. This is essentially the same process as Babai’s nearest plane algorithm, described below. After that, one tests Lovász’s condition. If it is satisfied, then the index $k$ can be incremented. If not, the vectors $b_k$ and $b_{k-1}$ are swapped, and the index $k$ is decremented. The correctness of
the LLL algorithm is relatively simple to prove, but the complexity analysis is significantly more involved. We refer the interested reader to [382].

The LLL algorithm has been extensively studied since its invention [324, 544, 576, 463, 465, 457]. Very often in practice, the returned basis is of better quality than the worst-case bound given above and is obtained faster than expected. We refer to [463] for more details about the practical behavior of the LLL algorithm.

Among many important applications of the LLL algorithm, Babai [23] extracted from it a polynomial-time algorithm for solving CVP with an exponentially bounded approximation factor. We present it in Algorithm A.1.

**Theorem A.5** (Babai [23]). Given an arbitrary basis \((b_1, \ldots, b_d)\) of a lattice \(L \subseteq \mathbb{Z}^n\), and a target vector \(t \in \mathbb{Z}^n\), Babai’s nearest plane algorithm (Algorithm A.1) finds a vector \(b \in L\) such that

\[ \|b - t\|_2 \leq 2^d \cdot \text{dist}_2(t, L). \]

Moreover, it finishes in polynomial time in \(d, n, \ln A\), and \(\ln \|t\|\), where \(A = \max_i \|a_i\|\).

**Algorithm A.1** Babai’s nearest plane algorithm. The inputs are an LLL-reduced basis \((b_i)_{1 \leq i \leq d}\), its Gram–Schmidt orthogonalization \((b_i^\ast)_{1 \leq i \leq d}\), and a target vector \(t\). The output is a vector in the lattice spanned by the \(b_i\)'s that is close to \(t\).

\[
v \leftarrow t
\]

for \((j = d; j \geq 1; j--)\) do

\[
v \leftarrow v - \left( \frac{\langle v, b_j^\ast \rangle}{\langle b_j^\ast, b_j^\ast \rangle} \right) b_j
\]

end for

return \(t - v\)

Babai’s algorithm may also be described with the LLL algorithm directly. This may be seen as a particular case of Kannan’s embedding technique [326]. This may be simpler to implement, in particular, if one has access to an implementation of LLL. We give that variant in Algorithm A.2.

**Algorithm A.2** Babai’s nearest plane algorithm, using LLL. The inputs are an LLL-reduced basis \((b_i)_{1 \leq i \leq d}\) and a target vector \(t\). The output is a vector in the lattice spanned by the \(b_i\)'s that is close to \(t\).

\[
\text{for } (j = 1; j \leq d; j++) \text{ do}
\]

\[
c_j \leftarrow (b_j, 0)
\]

end for

\[
B \leftarrow \max_i \|b_i\|; c_{d+1} \leftarrow (t, B)
\]

\[
(c'_1, \ldots, c'_{d+1}) \leftarrow \text{LLL}(c_1, \ldots, c_{d+1})
\]

return \(c_{d+1} - c'_{d+1}\)
Appendix B

Previous Floating-Point Standards

B.1 The IEEE 754-1985 Standard

The binary part of the IEEE 754-2008 standard is an evolution of IEEE 754-1985. Hence a large part of what we have presented in Chapter 3 was already present in IEEE 754-1985. In the following, we therefore mainly focus on the differences.

B.1.1 Formats specified by IEEE 754-1985

Of course, since IEEE 754-1985 focused on binary floating-point arithmetic, there were no decimal formats. The two basic formats were

- single precision: this is the format now called binary32 in IEEE 754-2008 (same parameters and bit encoding);

- double precision: this is the format now called binary64 (same parameters and bit encoding).

There was no 16-bit or 128-bit format.

To each basic format, the IEEE 754-1985 standard associated an extended format. The standard recommended an extended format for the widest basic format supported only. Hence, to the best of our knowledge, the single-extended precision has never been implemented: when double precision was available, it fulfilled all the purposes of a single-extended format. Table B.1 gives the main parameters of the formats specified by IEEE 754-1985.
### B.1.2 Rounding modes specified by IEEE 754-1985

The IEEE 754-1985 standard defined four rounding modes: round toward $-\infty$ (rounding function $RD$), round toward $+\infty$ (rounding function $RU$), round toward zero (rounding function $RZ$), and round to nearest ties to even (RN), also called round to nearest even. There was no round to nearest ties to away (it was brought up by IEEE 754-2008).

### B.1.3 Operations specified by IEEE 754-1985

#### B.1.3.1 Arithmetic operations and square root

The IEEE 754-1985 standard required that addition, subtraction, multiplication, and division of operands of the same format be provided, for all supported formats, with correct rounding (with the four rounding modes presented above). The standard also required a correctly rounded square root in all supported formats. The FMA (fused multiply-add) operator was not specified.

It was also permitted (but not required) that these operations be provided (still with correct rounding) for operands of different formats (in such a case, the destination format had to be at least as wide as the wider operand’s format).

#### B.1.3.2 Conversions to and from decimal strings

At the time the IEEE 754-1985 standard was released, some of the algorithms presented in Section 4.9 were not known. This explains why the requirements of the standard were clearly below what one could now expect. The 2008 version of the standard has much stronger requirements.

---

**Table B.1: Main parameters of the four formats specified by the IEEE 754-1985 standard [12](c)IEEE, 1985, with permission).** A specific single-extended format has not been implemented in practice. The last line describes the double-extended format introduced by Intel in the 387 FPU, and available in subsequent IA32 compatible processors by Intel, Cyrix, AMD, and others.

<table>
<thead>
<tr>
<th>Format</th>
<th>Hidden bit</th>
<th>Precision $p$</th>
<th>$\epsilon_{\text{min}}$</th>
<th>$\epsilon_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single precision</td>
<td>yes</td>
<td>24</td>
<td>$-126$</td>
<td>127</td>
</tr>
<tr>
<td>Double precision</td>
<td>yes</td>
<td>53</td>
<td>$-1022$</td>
<td>1023</td>
</tr>
<tr>
<td>Single-extended</td>
<td>optional</td>
<td>$\geq 32$</td>
<td>$\leq -1022$</td>
<td>$\geq 1023$</td>
</tr>
<tr>
<td>Double-extended</td>
<td>optional</td>
<td>$\geq 64$</td>
<td>$\leq -16382$</td>
<td>$\geq 16383$</td>
</tr>
<tr>
<td>Double-ext. (IA32)</td>
<td>no</td>
<td>64</td>
<td>$-16382$</td>
<td>16383</td>
</tr>
</tbody>
</table>
B.1. The IEEE 754-1985 Standard

<table>
<thead>
<tr>
<th></th>
<th>Decimal to binary</th>
<th>Binary to decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$M_{10,\text{max}}^{(1)}$</td>
<td>$E_{10,\text{max}}^{(1)}$</td>
</tr>
<tr>
<td>Single precision</td>
<td>$10^9 - 1$</td>
<td>99</td>
</tr>
<tr>
<td>Double precision</td>
<td>$10^{17} - 1$</td>
<td>999</td>
</tr>
</tbody>
</table>

Table B.2: The thresholds for conversion from and to a decimal string, as specified by the IEEE 754-1985 standard [12] (©IEEE, 1985, with permission).

The requirements of the IEEE 754-1985 standard were:

- conversions must be provided between decimal strings in at least one format and binary floating-point numbers in all basic floating-point formats, for numbers of the form
  \[ \pm M_{10} \times 10^{\pm E_{10}} \]
  with $M_{10} \geq 0$ and $E_{10} \geq 0$;

- conversions must be correctly rounded for operands in the ranges specified in Table B.3;

- when the operands are not in the ranges specified in Table B.3:
  - in round-to-nearest mode, the conversion error cannot exceed 0.97 ulp of the target format;
  - in the directed rounding modes, the “direction” of the rounding must be honored (e.g., for round-toward $-\infty$ mode, the delivered result must be less than or equal to the initial value), and the rounding error cannot exceed 1.47 ulp of the target format;

- conversions must be monotonic (if $x \leq y$ before conversion, then $x \leq y$ after conversion);

- when rounding to nearest, as long as the decimal strings have at least 9 digits for single precision and 17 digits for double precision, conversion from binary to decimal and back to binary must produce the initial value exactly (Table B.2). This allows one to store intermediate results in files, and to read them later on, without losing any information, as explained in Chapter 2, Section 4.9.

B.1.4 Exceptions specified by IEEE 754-1985

The five exceptions listed in Section 2.5 (invalid, division by zero, overflow, underflow, inexact) had to be signaled when detected, either by taking a trap or by setting a status flag. The default was not to use traps.
### Decimal to binary

<table>
<thead>
<tr>
<th></th>
<th>Single precision</th>
<th>Double precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_{10,\text{max}}^{(1)}$</td>
<td>$10^9 - 1$</td>
<td>$10^{17} - 1$</td>
</tr>
<tr>
<td>$E_{10,\text{corr}}^{(1)}$</td>
<td>$13$</td>
<td>$27$</td>
</tr>
</tbody>
</table>

#### Table B.3: Correctly rounded decimal conversion range, as specified by the IEEE 754-1985 standard [12] (©IEEE, 1985, with permission).

### B.2 The IEEE 854-1987 Standard

The IEEE 854-1987 standard [13] covered “radix-independent” floating-point arithmetic. This does not mean that all possible radices were considered: actually, that standard only focused on radices 2 and 10. We will just present it briefly (it is now superseded by IEEE 754-2008 [267]).

Unlike IEEE 754-1985, the IEEE 854-1987 standard did not fully specify formats or internal encodings. It merely expressed constraints between the parameters $\beta$, $e_{\text{min}}$, $e_{\text{max}}$, and $p$ of the various precisions provided by an implementation. It also said that for each available precision, we must have two infinities, at least one signaling NaN, and at least one quiet NaN (as in the IEEE 754-1985 standard). In the remainder of this section, $\beta$ is equal to 2 or 10. The same radix must be used for all available precisions: an arithmetic system compliant to IEEE 854-1987 is either binary or decimal, but it cannot mix up the two kinds of representations.

#### B.2.1 Constraints internal to a format

A balance must be found between the precision $p$ and the value of the extremal exponents $e_{\text{min}}$ and $e_{\text{max}}$. If $p$ is too large compared to $|e_{\text{min}}|$ and $e_{\text{max}}$, then underflows or overflows may occur too often. Also, there must be some balance between $e_{\text{min}}$ and $e_{\text{max}}$: to avoid underflows or overflows when computing reciprocals of normalized floating-point numbers as much as possible, one might want $e_{\text{min}} \approx -e_{\text{max}}$. Since underflow (more precisely, *gradual underflow*, with subnormal numbers available) is less harmful than overflow, it is preferable to have $e_{\text{min}}$ very slightly above $1 - e_{\text{max}}$. Here are the constraints specified by the IEEE 854-1987 standard.

- We must have
  \[
  \frac{e_{\text{max}} - e_{\text{min}}}{p} > 5,
  \]
  and it is recommended that
  \[
  \frac{e_{\text{max}} - e_{\text{min}}}{p} > 10.
  \]

---

$^1$This is why the IEEE 754-2008 standard now requires $e_{\text{min}} = 1 - e_{\text{max}}$ for all formats.
B.2. The IEEE 854-1987 Standard

We must have $\beta^{p-1} \geq 10^5$.

$\beta^{e_{\max}+e_{\min}+1}$ should be the smallest power of $\beta$ greater than or equal to 4 (which was a very complicated way of saying that $e_{\min}$ should be $1 - e_{\max}$ in radix 2 and $-e_{\max}$ in radix 10).

For instance, the binary32 format of IEEE 754-2008 satisfies these requirements: with $\beta = 2$, $p = 24$, $e_{\min} = -126$, and $e_{\max} = 127$, we have

\[
\begin{align*}
\frac{e_{\max} - e_{\min}}{p} &= 10.54\ldots > 10; \\
\beta^{p-1} &= 2^{23} \geq 10^5; \\
\beta^{e_{\max}+e_{\min}+1} &= 2^2 = 4.
\end{align*}
\]

B.2.2 Various formats and the constraints between them

The narrowest supported format was called single precision. When a second, wider basic format is supported, it was called double precision. The required constraints between their respective parameters $e_{\min s}$, $e_{\max s}$, $p_s$ and $e_{\min d}$, $e_{\max d}$, $p_d$ were:

- $\beta^{p_d} \geq 10\beta^{2p_s}$;
- $e_{\max d} \geq 8e_{\max s} + 7$;
- $e_{\min d} \leq 8e_{\min s}$.

Extended precisions were also possible. For obvious reasons, the only extended precision that was recommended was the one associated with the widest supported basic precision. If $e_{\min}$, $e_{\max}$, and $p$ are the extremal exponents and precision of that widest basic precision, the parameters $e_{\min e}$, $e_{\max e}$, and $p_e$ of the corresponding extended precision had to satisfy:

- $e_{\max e} \geq 8e_{\max} + 7$;
- $e_{\min e} \leq 8e_{\min}$;
- if $\beta = 2$,
  \[
  p_e \geq p + \lceil \log_2 (e_{\max} - e_{\min}) \rceil; \tag{B.1}
  \]
- for all $\beta$, $p_e \geq 1.2p$.

It was also recommended that

\[
p_e > 1 + p + \frac{\log (3 \log(\beta) (e_{\max} + 1))}{\log(\beta)} \tag{B.2}
\]

The purpose of constraint (B.1) was to facilitate the support of conversion to and from decimal strings for the basic formats, using algorithms that were available at that time. The purpose of (B.2) was to make accurate implementation, in the basic formats, of the power function $x^y$ simpler.
B.2.3 Rounding

The IEEE 854-1987 standard required that the arithmetic operations and the square root be correctly rounded. Exactly as for IEEE 754-1985, four rounding modes were specified: rounding toward \(-\infty\), toward \(+\infty\), toward 0, and to nearest ties to even.

B.2.4 Operations

The arithmetic operations, the remainder operation, and the square root (including the \(\sqrt{-0} = -0\) requirement) were defined very much as in IEEE 754-1985.

B.2.5 Comparisons

The comparisons were defined very much as in IEEE 754-1985. Especially, every NaN compares “unordered” with everything including itself: the test “\(x \neq x\)” must return true if and only if \(x\) is a NaN.

B.2.6 Exceptions

The IEEE 754-1985 way of handling exceptions was also chosen for IEEE 854-1987.

B.3 The Need for a Revision

The IEEE 754-1985 standard was a huge improvement. It soon became implemented on most platforms of commercial significance. And yet, 15 years after its release, there was a clear need for a revision.

- Some features that had become common practice needed to be standardized: e.g., the “quadruple-precision” (i.e., 128-bit wide, binary) format, the fused multiply-add operator.

- Since 1985, new algorithms were published that allowed one to easily perform computations that were previously thought too complex. Typical examples are the radix conversion algorithms presented in Section 4.9: now, for an internal binary format, it is possible to have much stronger requirements on the accuracy of the conversions that must be done when reading or printing decimal strings. Another example is the availability of reasonably fast libraries for some correctly rounded elementary functions: the revised standard can now deal with transcendental functions and recommend that some should be correctly rounded.
• There were some ambiguities in IEEE 754-1985. For instance, when evaluating expressions, when a larger internal format is available in hardware, it was unclear in which format the implicit intermediate variables should be represented.

B.4 The IEEE 754-2008 Revision

The IEEE 754-1985 standard has been revised from 2000 to 2006, and the revised standard was adopted in June 2008. Some of the various goals of the working group were as follows (see http://grouper.ieee.org/groups/754/revision.html):

• merging the 854-1987 standard into the 754-1985 standard;
• reducing the implementation choices;
• resolving some ambiguities in the 754-1985 standard (especially concerning expression evaluation and exception handling). The revised standard allows languages and users to focus on portability and reproducibility, or on performance;
• standardizing the fused multiply-add (FMA) operation, and
• including quadruple precision.

Also, the working group had to cope with a very strong constraint: the revised standard would rather not invalidate hardware that conformed to the old IEEE 754-1985 standard.
Bibliography


Bibliography


Bibliography


Index

2Mult, 116, 311, 319
2MultFMA, 112, 205
2Sum, 108, 311, 319, 490

accumulator, 316
AccurateDWPlusDW, 517
accurate step, 428
ACL2, 488
addition, 240
  binary floating-point,
    hardware, 287
  binary floating-point,
    software, 329
decimal integer, 274
integer, 272
signed zero, 241
subnormal handling, 242, 245, 293

affine arithmetic, 475
algebraic function, 400
α (smallest subnormal number), 18, 20, 25
Arb, 471, 552
Arenarius, 4
argument reduction, see range reduction
arithmetic formats, 48
ARRE (average relative representation error), 42
attribute, 66
  alternate exception handling, 68
  preferred width, 68
reproducibility, 69
rounding direction, 22, 66
value-changing optimization, 69
average, 490
AVX, 87

Babai’s algorithm, 560
Babylonians, 4
backward error, 168
base, 15
basic formats, 48
BCD (binary coded decimal), 54
Benford’s law, 42
bias, 51, 55, 56
biased exponent, 52, 53, 55–57, 239
big-endian, 65
binary128, 49–52, 85, 203
binary16, 49, 52, 87, 89
binary32, 18, 49, 50, 52, 563, 564
binary64, 49, 50, 52, 563, 564
binding, 90
bipartite table method, 286
block floating-point, 319
Booth recoding, 279
breakpoint, 24, 397, 404
Brouwer’s theorem, 460
Burger and Dybvig conversion algorithm, 147

C programming language, 200
C++ programming language, 215
C11, 200
C99, 200

cancellation, 102, 180, 377

canonical encoding, 54, 56
carry-ripple adder, 273
carry-skip adder, 274, 275
catastrophic cancellation, see cancellation

CENA, 164

Chebyshev
polynomials, 391

theorem, 392

Clinger conversion algorithm, 148

close path, 243
closest vector problem, 558

Cody and Waite reduction algorithm, 379

cohort, 16, 54, 71, 234

combination field, 54, 55
comparison, 72

CompCert, 153, 214, 490

CompensatedDotProduct algorithm, 189

compensated algorithm, 164

polynomial evaluation, 190

summation, 178

compiler, 490

<complex.h>, 201

compound adder, 277, 299

condition numbers, 169

continued fractions, 383, 553, 554

contracted expression, 206, 214

convergent (continued fractions), 554

conversion, 73, 146, 153, 490

convolution neural networks, 87

Coq, 489

CoqInterval, 490, 505

CORDIC algorithm, 378

correctly rounded, 23, 24

CRLibm, 187, 381, 386, 522

Cset theory, 458

C# programming language, 230

custom operator, 372

CVP, see closest vector problem

data dependency, 271
decimal arithmetic in hardware, 87, 270
decimal encoding, 56
declet, 54
decoration, 460

com, 461
dac, 461
def, 461

ill, 461

trv, 461

DekkerDWTimesDW, 518

Dekker product, 103, 116, 489
delay, 271
denormal number, see subnormal number
discriminant, 489
division, 256, 489

by a floating-point constant, 315

by zero, 38, 75
decimal floating-point, 308
digit recurrence, 257, 305

hardware, 304

rounding, 235

SRT algorithm, 307
dot product

exact, 464
double-double, see double-word, 203, see double-word
double-word, 203, 394, 431, 515

addition, 517
division, 520

multiplication, 518
double extended format, 64, 196
double precision, see binary64
double rounding, 79, 81, 88, 90, 195, 262, 264, 490, 544

DSP (digital signal processing), 296, 310
dynamic range, 42

elementary function, 375, 489
e_{\text{max}}, 16
$\epsilon_{\text{min}}$, 16
deround-around carry adder, 277endianness, 65
ErrFma, 162
Estrin’s method, 396
Euclidean lattice, 556
exactly rounded, 23
exact addition, 100
exact dot product, 464
exception, 37, 494, 563, 566
alternate handling, 68
exclusion lemma, 131
exclusion zone, 131
expansion, 522
addition, 526
division, 531
multiplication, 528
nonoverlapping, 522–524
renormalization, 525
ulp-nonoverlapping, 524
exponent
bias, 56
extremal, 16
quantum, 16
extendable precision, 49, 63
extended format, 561
extended precision, 49, 63–65, 565

faithful arithmetic, 24, 109, 543
faithful result, 24, 313, 519
far path, 243
Fast2Sum, 104, 105, 110
FastDWTimesDW, 518
<env.h>, 202
field-programmable gate array, 269, 277, 285, 312
fixed point, 318, 319, 496
flavor, 458
set-based, 459
FLIP, 321
<float.h>, 201
floating-point expansion, 522
Flucq, 108, 162, 214, 490
Fluctuat, 475
FMA, 37, 84, 248, 489
binary implementation, 254
decimal, 253
hardware, 301
latency, 84
subnormal handling, 252, 302
fma(), 205
Forte, 488
FORTRAN, 218
FPGA, see field-programmable gate array
FPSR (Floating-Point Status register), 85
FPTaylor, 492
fraction, 18, 50
full adder, 273
fused multiply-add, see FMA

$\gamma$ notation, 166
Gappa, 397, 490, 493
Gay conversion algorithm, 147, 148
GMP, see GNU MP
GNU MP, 535
GNU MPC, 552
GNU MPFR, 535, 541
GNU Octave, 464
Goldschmidt iteration, 129
GPGPU (general-purpose graphics processing units), 89
GPU (graphical processing unit), 89, 269

Haar condition, 393
half precision, see binary16, 50, see binary16
hardest to round point, 399
hardness to round, 399
Hauser’s theorem, 102
Heron iteration, 138
hidden bit convention, 18, 50
HOL, 489
HOL Light, 489
homogeneous operators, 70, 261
Horner algorithm, 167, 396, 489
running error bound, 175

IEEE 1788.1 standard, 464
IEEE 1788 standard, 454, 457, 458, 460, 461, 463, 464
IEEE 754-1985 standard, 6, 47, 561
IEEE 754-2008 standard, 47, 48, 567
IEEE 754R, 48
IEEE 854-1987 standard, 6, 564
ILP (instruction-level parallelism), 328
implicit bit convention, 18, 41, 43
inclusion property, 455
inexact exception, 38, 76, 77, 239, 259, 261
ininitely precise significand, 17, 23
infinity, 22, 78
insertion summation, 177
integral significand, 16, 18, 54, 57
interchange formats, 48
interval, 454
interval arithmetic, 453, 494, 552
fundamental theorem, 455
fundamental theorem of
decorated interval
arithmetic, 462
interval Newton-Raphson
iteration, 459
INTLAB, 471, 534
invalid operation exception, 21, 37, 50, 74, 77
iRRAM, 552
is normal bit, 239, 293, 300, 301, 322
IteratedProductPower, 519
iterated products, 35
Javascript programming
language, 230
Java programming language, 222

$K$-fold summation algorithm, 183
Kahan’s compensated sum
method, 104
Kaucher arithmetic, 458

language, 90, 193
largest finite number, 19, 75
large accumulator, 316
latency, 271
leading bit convention, 4, 18, 50
leading-zero anticipation, 284, 290, 294, 302
leading-zero counter, 283, 288–291, 311
by monotonic string
conversion, 284
combined with shifter, 284
tree-based, 283
left-associativity, 195
libieee1788, 464
little-endian, 65
LLL algorithm, 556
logarithmic distribution, 42
logB, 71, 75
look-up table, 277, 286, 314
LOP (leading one predictor), see
leading zero anticipation
LSB (least significant bit), 289
LUT, see look-up table
LZA, see leading zero anticipation
LZC, see leading zero counter

MACHAR, 91
machine epsilon, 34
Magma, 534, 538
mantissa, 17
MAPLE, 5, 383, 404
Mars Climate Orbiter, 9
<math.h>, 201
Mathemagix, 471
MATLAB, 34, 475, 534
mid-rad representation, 468, 471
modal intervals, 458
monotonic conversion, 563
Index

MPCHECK, 93
MPFI, 552
MPFR, see GNU MPFR
MRRE (maximum relative representation error), 42
MSB (most significant bit), 283
multipartite table method, 286
multiplication, 245
  binary floating-point, hardware, 295
  by a constant, 314
  by a floating-point constant, 314
  by an arbitrary precision constant, 156, 315
decimal integer, 280
digit by integer, 278
integer, 280
subnormal handling, 247, 300
MXCSR, 85

NaN (Not a Number), 21, 37, 50, 56, 72–74, 77, 204, 216, 224, 480, 564, 566
noncanonical encoding, 55
nonoverlapping, 522–524
non homogeneous operators, 261
normalized representation, 17
normal number, 17, 18, 25
  smallest, 19
normal significand, 16
norm (computation of), 39, 312
NTL, 534

Octave, 464
Ω (largest finite FP number), 19, 25
optimization, 69
orthogonal polynomials, 390
output radix conversion, 146
overestimation, 456
overflow, 38, 75
  in addition, 242
  spurious, 107
parallel adders, 275
Paranoia, 92, 100
partial carry save, 275
payload, 74, 78
Payne and Hanek reduction algorithm, 382
pipeline, 271
pole, 38
polynomial approximation, 389, 490
  $L^2$, 390
  $L^\infty$, 391
  Chebyshev, 391
  minimax, 391
  Remez algorithm, 392
  truncated, 394
polynomial evaluation, 396
  Estrin, 396
  Horner, 396
pow function, 209
precision, 16
preferred exponent, 54, 234, 244, 248, 253, 259, 261
preferred width, 68
prefix tree adders, 276
programming language, 90, 193
PVS, 489
Python programming language, 229
QD (quad-double) library, 522, 527, 529, 531
quad precision, 49, 51
quadratic convergence, 125
quad precision, 50, 203
quantum, 16, 18, 30
quick step, 428
quiet NaN, 50, 74, 77, 78, 204, 216, 564
radix, 15
radix conversion, 142, 146, 240
range reduction, 210, 377, 379
  Cody and Waite, 379
  Payne and Hanek, 382
  relative error, 382
  worst cases, 385
RD (round down), see round toward $-\infty$
read-only memory, 286
reconfigurable circuits, see field-programmable gate array
RecursiveDotProduct algorithm, 167
RecursiveSum algorithm, 167
relative backward error, 168
relative error, 26, 34
remainder, 70
Remez algorithm, 392
renormalization, 521, 525
reproducibility, 44, 69
RN, see round to nearest
ROM, see read-only memory
rounding, 235
  binary floating-point, 237
  breakpoint, 24, 397
  decimal floating-point, 237
    binary encoding, 240
  directed mode, 24
  direction, 66
downwards, 22
faithful, 24
injection, 297
mode, 22
toward $+\infty$, 22, 66, 465
toward $-\infty$, 22, 66, 465
toward zero, 23, 66
to nearest, 23
to nearest even, 23, 67
to nearest ties to away, 23
to nearest ties to even, 23
to odd, 155, 187
upwards, 22
roundTiesToAway, 67
roundTiesToEven, 67
roundTowardNegative, 66
roundTowardPositive, 66
roundTowardZero, 66
round bit, 23, 237
round digit, 237
RTL, 488
running error bounds, 163
RU (round up), see round toward $+\infty$
RZ, see round toward zero
scaleB, 71
set-based flavor, 459
SETUN computer, 5
shift-and-add algorithms, 378
shortest vector problem, 557
signaling NaN, 50, 74, 77, 204, 210, 216, 564
significand, 3, 4, 15–18
  alignment, 242
distance, 418
SIMD (single instruction, multiple data), 86
single precision, see binary32
slide rule, 4
SoftFloat, 92, 322
Sollya, 360, 394
square root, 259
  rounding, 235, 488
SRT division, 257, 307
SRTEST, 93
SSE2, see SSE
SSE (Streaming SIMD Extension), 81, 88, 467
standard model of floating-point arithmetic, 28, 165
status flag, 563
Steele and White conversion algorithm, 144, 147
Sterbenz’s lemma, 100
sticky bit, 23, 237
subnormal number, 18, 19, 25, 100, 102, 106, 114, 116
smallest, 18
Index

subtraction, 240
SVP, see shortest vector problem

θ notation, 166
table-based methods, 286
Table maker’s dilemma, 376, 398, 519
Taylor models, 476
TestFloat, 92
<stdio.h>, 201
tie-breaking rule, 23
TMD, 376
trailing significand, 18, 50, 51, 53, 55, 56
transcendental number, 404
trap, 21, 563
triangle area, 490
triple-word arithmetic, 520
Tuckerman test, 140
TwoMult, see 2Mult
TwoMultFMA, see 2MultFMA
TwoSum, see 2Sum

u, see unit roundoff
UCBTest, 92
ulp (unit in the first place), 34
ulp-nonoverlapping expansion, 524
ulp (unit in the last place), 16, 29, 34, 140, 146, 382
Goldberg definition, 30
Harrison definition, 29
underflow, 19, 38, 76
after rounding, 20
before rounding, 20
gradual, 19, 467
spurious, 107
unit roundoff, 27, 165
unordered, 72

VecSum, 524
VecSumErrBranch, 525
Veltkamp splitting, 113, 114
VLIW, 328
VLSI (very large-scale integration), 268, 269, 286

write-read cycle, 143

YBC 7289, 4

Z3 computer, 4, 22
zero
binary interchange format, 51
decimal interchange format, 57
sign, 22
Ziv’s multilevel strategy, 398