Contents

A Mechanically Checked Proof of Correctness of the AMD K5 Floating Point Square Root Microcode

David M. Russinoff

September 29, 1997

Abstract

We present a rigorous mathematical proof of the correctness of the floating point square root instruction of the AMD K5 microprocessor. The instruction is represented as a program in a formal language that was designed for this purpose, based on the K5 microcode and the architecture of its FPU. We prove a statement of its correctness that corresponds directly with the IEEE Standard. We also derive an equivalent formulation, expressed in terms of rational arithmetic, which has been encoded as a formula in the ACL2 logic and mechanically verified with the ACL2 prover. Finally, we describe a microcode modification that was implemented as a result of this analysis in order to ensure the correctness of the instruction.

1 Introduction

The limitations of traditional hardware validation methodology are widely recognized. Experience has shown that floating point instructions, in particular, are so complicated that it is practically impossible to ensure the correctness of even the basic arithmetic operations through simulation. Consequently, considerable industrial interest has developed in the application of formal methods to this area.

One obstacle to formal hardware verification is the difficulty of establishing a precise specification for a targeted operation. In many cases, the correctness of an implementation is determined by its conformance with an industry standard "golden model". In the absence of an accurate behavioral description of such a model, there is no viable alternative to testing.

With regard to floating point instructions, this observation may apply to transcendal functions, but not to the more primitive operations. The IEEE Standard for Binary Floating Point Arithmetic [6] specifies unambiguously for a variety of functions, including the basic arithmetic operations as well as the square root, that each operation

> ... shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result according to one of the modes ....

In order to express this requirement in a mathematical notation, let $rnd(\zeta, mode, n)$ denote the result of rounding a real number ζ, according to a given *mode*, to n bits of precision, and let "$\circ$" denote binary addition, subtraction, multiplication, or division (of the "correct to infinite precision" variety). The prescribed value to be returned by the corresponding floating point operation, for inputs x and y, is then

$$rnd(x \circ y, mode, n). \tag{1}$$

Although the simplicity of this specification strongly suggests its susceptibility to formal verification, this possibility remains largely unexplored. Automatic finite-state techniques, which are commonly used to verify low-level properties of arithmetic circuits [2, 3], are apparently inadequate for the comprehensive verification of IEEE compliance. General-purpose theorem provers have been used to check correctness proofs of various numerical algorithms, but in most instances, the specification of the algorithm either ignores rounding entirely [8, 12] or is otherwise too abstract to offer any assurance regarding its hardware implementation [4, 9].

One exception is the formal proof of correctness of a microcode-level specification of the floating point division algorithm of the AMD K5 microprocessor by Moore et al. [10], using the ACL2 prover [1]. In this paper, we shall present a similar analysis of the K5 square root algorithm, which was developed at AMD by Tom Lynch, Mike Schulte, and Ashraf Ahmed. In particular, we shall show that for a given nonnegative floating point number P, a rounding mode *mode*, and a degree of precision n, the value *sqrt* computed by this algorithm satisfies

$$sqrt = rnd(\sqrt{P}, mode, n) \tag{2}$$

in compliance with the IEEE specification.

While the correctness of a hardware design can never be guaranteed by the properties of a mathematical model, a reasonable degree of reliability can be achieved by minimizing the "semantic gap" between an abstract algorithm and its hardware implementation. With this goal in mind, we have designed a simple formal language based on the K5 microcode and the architecture of its floating point unit. The K5 square root instruction, which we shall represent as a program in this language, is actually implementated as a sequence of microcode instructions that rely on existing hardware for multiplication, addition, and subtraction. Thus, the execution of a program in our language consists mainly of evaluation of expressions of the above form (1), in which "$\circ$" may represent any of these three operations. Furthermore, our semantics require that the value of every such expression is representable in a format that is accommodated by the K5 floating point unit. Consequently, once the abstract program has been verified, the correctness of the implementation depends mainly on the correctness of the basic arithmetic operations, which may reasonably be assumed for this purpose.

This paper is a self-contained mathematical exposition. We begin in Section 2 by establishing a foundation for floating point rounding. We first construct the set of n-bit floating point numbers as a subset of the rationals. Then we define the rounding function $rnd(\zeta, mode, n)$, for each of seven modes, and derive its relevant properties. The results of this section should be reusable in a variety of applications, and are currently being applied to the analysis of the floating point unit of the AMD K7 microprocessor [11]. Presumably, these results are generally known to the floating point design community, but we have not found them collected elsewhere in any rigorous development. Section 2 might be used merely as a reference by the reader who is familiar with the general theory.

In Section 3, we define our abstract language, present the square root program, and formulate a statement of its correctness. The rest of the paper is a fairly detailed proof of this statement. This proof is somewhat complicated, but not deep, and could certainly be checked by any competent mathematician. However, the traditional social process by which mathematical results are normally ratified is inadequate in cases such as this, because of the urgent demand for the result and the high cost of error. Therefore, following [10], we have employed the ACL2 system to verify our results mechanically.

ACL2 is based on a logical language [7] that formalizes an applicative subset of Common Lisp [13] and is supported by a mechanical theorem prover [1]. Since Common Lisp, and consequently ACL2, include the rational numbers as a data type but not the reals, we are somewhat limited in the formalization of our theorem. For example, the Newton-Raphson formula, which is central to both the square root and division algorithms, is based on calculus. In order to justify our application of this formula, however, we are required to derive the relevant convergence theorem without appealing to calculus, but rather as a theorem of rational arithmetic.

A more critical problem, which we do not share with the authors of [10], is that because of its reference to the square root, our main theorem itself is not a statement about the rationals, and consequently is not expressible in the ACL2 logic. The theorem of rational arithmetic that seems best to approximate (2) is the following: *For any nonnegative rational numbers ℓ and h, if $\ell^2 \leq P \leq h^2$, then*

$$rnd(\ell, mode, n) \leq sqrt \leq rnd(h, mode, n). \tag{3}$$

More precise versions of (2) and (3) appear as Theorems 1 and 2, respectively, of Section 3. As we shall see, the equivalence of these theorems can be proved easily enough, although without the support of ACL2. Thus, our formalization effort is directed at Theorem 2, while the consequent derivation of Theorem 1 depends on an informal argument. Along with Theorem 2, every definition and lemma presented in this paper (with the exception noted in Section 3) has been formally encoded in the ACL2 logic, and every proof has been mechanically checked with the ACL2 prover.

Of course, there is a cost for the increased confidence provided by mechanical verification. In this case, the restriction to rational arithmetic imposed by the ACL2 logic resulted in extra details in a proof that was already quite complicated. Two months of the author's time was spent developing the handwritten proof, including extracting the algorithm from microcode and establishing the necessary general theory. At least one week of this was devoted to eliminating all references to the square root function from a preliminary version of the proof.

After the complete proof was written, a full additional month was spent checking it mechanically. This is an empirical process, by no means automatic. In this project, some 1400 lemmas, along with abundant hints, were ultimately used to guide the prover successfully to the final result. Thus, each lemma that appears in this paper corresponds (on average) to approximately fifty ACL2 lemmas.

The use of ACL2 did not expose any serious errors in the proof, which had perhaps been written especially carefully in anticipation of the mechanical checking process. Indeed, the most significant difference between the handwritten proof that was subjected to ACL2 and the final version presented here is in Lemma 2.15: in its original statement, x was (correctly) assumed only to be rational; the hypothesis that x is k-exact was added only to facilitate the formal proof.

Arguing for the practical value of formal methods in computer design is sometimes difficult. Advocates of formal verification are often required to defend the awkward position that although a design was correct from the beginning, any confidence in its correctness prior to its verification was unfounded. In the present case, however, we have been more fortunate: our formal analysis of the square root algorithm revealed an apparent flaw, at least in the informal design rationale that had been provided by the implementors, which was ultimately corrected through a minor modification of the K5 microcode. After presenting the final algorithm and our proof of its correctness, we shall describe the nature of this modification.

2 Floating Point Arithmetic

In this section, we discuss the set of *n-bit normal* floating point numbers, and the function *rnd*, which rounds an arbitrary real number to an element of this set according to a specified *mode*. For each mode, the rounded result $rnd(x, mode, n)$ will alternatively be denoted as $mode(x, n)$. Thus, a rounding mode is a mapping from the reals to the floating point numbers. We shall define seven modes, all of which are implemented in the K5, including the four that are addressed by the IEEE standard.

2.1 Preliminaries

The symbols $\mathbf{R}$, $\mathbf{Q}$, $\mathbf{Z}$, and $\mathbf{Z}^+$ will denote the sets of all real numbers, rational numbers, integers, and positive integers, respectively.

Our definitions of the rounding modes will be based on the following:

Definition 2.1 *Let $x \in \mathbf{R}$.*

(a) The **floor** *of x, denoted $\lfloor x \rfloor$, is the unique $n \in \mathbf{Z}$ satisfying $n \leq x < n+1$.*

(b) The **ceiling** *of x, denoted $\lceil x \rceil$, is the unique $n \in \mathbf{Z}$ satisfying $n \geq x > n-1$.*

Definition 2.2 *If $n \in \mathbf{Z}$ and $d \in \mathbf{Z}^+$, then $quot(n,d) = \lfloor n/d \rfloor$ and $rem(n,d) = n - d\lfloor n/d \rfloor$.*

We shall require the following properties of the floor and ceiling:

Lemma 2.1 *Let $x, y \in \mathbf{R}$ and $n \in \mathbf{Z}$.*

(a) If $x \leq y$, then $\lfloor x \rfloor \leq \lfloor y \rfloor$ and $\lceil x \rceil \leq \lceil y \rceil$.

(b) $\lceil n \rceil = \lfloor n \rfloor = n$.

(c) If $x \notin \mathbf{Z}$, then $\lceil x \rceil = \lfloor x \rfloor + 1$.

(d) $\lfloor x+n \rfloor = \lfloor x \rfloor + n$ and $\lceil x+n \rceil = \lceil x \rceil + n$.

(e) If $n > 0$, then $\lfloor \lfloor x \rfloor / n \rfloor = \lfloor x/n \rfloor$ and $\lceil \lceil x \rceil / n \rceil = \lceil x/n \rceil$.

Proof: We shall derive the results stated for the floor; the proofs of the corresponding ceiling properties are similar:

(a) $x \leq y \Rightarrow \lfloor x \rfloor \leq y \Rightarrow \lfloor x \rfloor < \lfloor y \rfloor + 1 \Rightarrow \lfloor x \rfloor \leq \lfloor y \rfloor$.

(b) $n \leq n < n+1 \Rightarrow \lfloor n \rfloor = n$.

(c) $\lfloor x \rfloor < x < \lfloor x \rfloor + 1 \Rightarrow \lfloor x \rfloor + 1 > x > (\lfloor x \rfloor + 1) - 1 \Rightarrow \lceil x \rceil = \lfloor x \rfloor + 1$.

(d) $\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1 \Rightarrow \lfloor x \rfloor + n \leq x + n < \lfloor x \rfloor + n + 1$.

(e) $\lfloor x \rfloor \leq x \Rightarrow \lfloor x \rfloor / n \leq x/n \Rightarrow \lfloor \lfloor x \rfloor / n \rfloor \leq \lfloor x/n \rfloor$, and $x/n \geq \lfloor x/n \rfloor \Rightarrow x \geq n\lfloor x/n \rfloor \Rightarrow \lfloor x \rfloor \geq n\lfloor x/n \rfloor \Rightarrow \lfloor x \rfloor / n \geq \lfloor x/n \rfloor \Rightarrow \lfloor \lfloor x \rfloor / n \rfloor \geq \lfloor x/n \rfloor$. □

The following estimates will be required in later sections:

Lemma 2.2 *Let $x, y, \delta \in \mathbf{R}$, $x \geq 2y \geq 0$, and $\delta \geq 0$.*

(a) $x - y(1+\delta) \geq (x-y)(1-\delta)$; *(b) $x - y(1-\delta) \leq (x-y)(1+\delta)$.*

Proof: Since $y \leq x - y$,

(a) $x - y(1+\delta) = x - y - y\delta \geq x - y - (x-y)\delta = (x-y)(1-\delta)$;

(b) $x - y(1-\delta) = x - y + y\delta \leq x - y + (x-y)\delta = (x-y)(1+\delta)$. □

Lemma 2.3 *Let $x, y, \delta \in \mathbf{R}$. If $x\delta \geq 0$, then $x + y(1-\delta) \geq (x+y)(1-\delta)$.*

Proof: $x + y(1-\delta) \geq x + y(1-\delta) - x\delta = x + y - x\delta - y\delta = (x+y)(1-\delta)$.□

Lemma 2.4 *Let $m, n \in \mathbf{Z}$. If $m \geq n$, then $(1-2^m)(1-2^n) > 1-2^{m+1}$.*

Proof: $(1-2^m)(1-2^n) = 1-2^m(1+2^{n-m})+2^{m+n} > 1-2^m(1+2^{n-m}) \geq 1-2^{m+1}$.$\Box$

Lemma 2.5 *Let $m, n \in \mathbf{Z}$. If $n \leq m \leq 0$, then $(1+2^m)(1+2^n) < 1+2^{m+2}$.*

Proof: $(1+2^m)(1+2^n) = 1+2^m(1+2^{n-m}+2^n) \leq 1+2^m(1+1+1) < 1+2^{m+2}$.$\Box$

Lemma 2.6 *Let $n \in \mathbf{Z}$. If $n \leq -1$, then $\frac{1}{1-2^n} \leq 1+2^{n+1}$.*

Proof: $(1-2^n)(1+2^{n+1}) = 1+2^{n+1}-2^n-2^{2n+1} = 1+2^n-2^{2n+1} = 1+2^n(1-2^{n+1}) \geq 1+2^n(1-1) = 1$.$\Box$

Lemma 2.7 *Let $a, b, P \in \mathbf{R}$ and $n \in \mathbf{Z}$. If $(1-2^n)P \leq a^2 \leq P$ and $b^2 \leq 2^{2n-2}P$, then $(a-b)^2 \geq (1-2^{n+1})P$.*

Proof: Since $a^2b^2 \leq 2^{2n-2}P^2$, $ab \leq 2^{n-1}P$, and hence

$$(a-b)^2 = a^2-2ab+b^2 \geq a^2-2ab \geq (1-2^n)P - 2^nP = (1-2^{n+1})P.\Box$$

2.2 Floating Point Numbers

For the purpose of floating point representation, a real number is factored into three components: *sign*, *exponent*, and *significand*.

Definition 2.3 *Let $x \in \mathbf{R}$. If $x \neq 0$, then*

(a) $sgn(x) = x/|x|$;
(b) $expo(x)$ is the unique integer that satisfies $2^{expo(x)} \leq |x| < 2^{expo(x)+1}$;
(c) $sig(x) = |x|2^{-expo(x)}$.

If $x = 0$, then $sgn(x) = expo(x) = sig(x) = 0$.

The next three lemmas are immediate consequences of Definition 2.3:

Lemma 2.8 *Let $x \in \mathbf{R}$, $x \neq 0$.*

(a) $x = sgn(x)sig(x)2^{expo(x)}$.
(b) $sgn(x) \in \{1, -1\}$, $1 \leq sig(x) < 2$, and $expo(x) \in \mathbf{Z}$.
(c) If $x = sm2^e$, where $s \in \{1, -1\}$, $1 \leq m < 2$, and $e \in \mathbf{Z}$, then $s = sgn(x)$, $m = sig(x)$, and $e = expo(x)$.

Lemma 2.9 *If $x, y \in \mathbf{R}$ and $|x| \leq |y|$, then $expo(x) \leq expo(y)$.*

Lemma 2.10 *If $x \in \mathbf{R}$, $x \neq 0$, $n \in \mathbf{Z}$, and $y = 2^nx$, then*

(a) $sgn(y) = sgn(x)$; (b) $sig(y) = sig(x)$; (c) $expo(y) = n + expo(x)$.

We have the following estimate for the exponent of a product:

Lemma 2.11 *Let $x, y \in \mathbf{R}$. If $xy \neq 0$, then*

$$expo(x) + expo(y) \leq expo(xy) \leq expo(x) + expo(y) + 1.$$

Proof: Let $m = expo(x)$, $n = expo(y)$, $a = sig(x)$, and $b = sig(y)$. Then $xy = sgn(xy)ab2^{m+n}$ and $1 \leq ab < 4$. If $1 \leq ab < 2$, then $sig(xy) = ab$ and $expo(xy) = m + n$; if $2 \leq ab < 4$, then $sig(xy) = ab/2$ and $expo(xy) = m + n + 1$. □

The following predicate characterizes numbers with significands that are representable with n bits, commonly known as *n-bit normal floating point numbers*:

Definition 2.4 *Let $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$. Then x is n-**exact** iff $sig(x)2^{n-1} \in \mathbf{Z}$.*

The definition may be restated as follows:

Lemma 2.12 *Let $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$. Then x is n-exact $\Leftrightarrow x2^{n-1-expo(x)} \in \mathbf{Z}$.*

Lemma 2.13 *Let $x \in \mathbf{R}$, $m \in \mathbf{Z}^+$, and $n \in \mathbf{Z}$. If x is m-exact, then so is $2^n x$.*

Proof: This follows from Lemma 2.10. □

Lemma 2.14 *Let $x, y \in \mathbf{R}$ and $m, n \in \mathbf{Z}^+$. If x is m-exact and y is n-exact, then xy is $(m+n)$-exact.*

Proof: If $x2^{m-1-expo(x)}$ and $y2^{n-1-expo(y)}$ are integers, then by Lemma 2.11, so is

$$x2^{m-1-expo(x)}y2^{n-1-expo(y)}2^{expo(x)+expo(y)+1-expo(xy)} = xy2^{m+n-1-expo(xy)}.\Box$$

Lemma 2.15 *Let $k, n \in \mathbf{Z}^+$ and $x \in \mathbf{R}$. If x is k-exact and x^2 is $2n$-exact, then x is n-exact.*

Proof: Let m be the least element of $\mathbf{Z}^+$ such that x is m-exact, and let $a = x2^{m-1-expo(x)}$. Then a is an odd integer and $x = a2^{expo(x)+1-m}$. Since

$$\begin{aligned} x^2 2^{2n-1-expo(x^2)} &= a^2 2^{2expo(x)+2-2m} 2^{2n-1-expo(x^2)} \\ &= a^2 2^{2(n-m)+1-(expo(x^2)-2expo(x))} \in \mathbf{Z} \end{aligned}$$

and a^2 is odd, we have $2(n-m)+1 \geq expo(x^2) - 2expo(x) \geq 0$, implying $m \leq n$.□

Lemma 2.16 *Let $x, y \in \mathbf{R}$, $n \in \mathbf{Z}^+$, $k \in \mathbf{Z}$, and $k < n$. If x and y are both n-exact and $expo(x-y) + k \leq min(expo(x), expo(y))$, then $x - y$ is $(n-k)$-exact.*

Proof: Since x is n-exact and $expo(x-y)+k \leq expo(x)$, $x2^{n-1-(expo(x-y)+k)} \in \mathbf{Z}$. Similarly, $y2^{n-1-(expo(x-y)+k)} \in \mathbf{Z}$. Thus,

$$(x-y)2^{(n-k)-1-expo(x-y)} = x2^{n-1-(expo(x-y)+k)} - y2^{n-1-(expo(x-y)+k)} \in \mathbf{Z}.\Box$$

Lemma 2.16 is often applied with $k = 0$, in the following weaker form:

Corollary 2.17 *Let $x, y \in \mathbf{R}$ and $n \in \mathbf{Z}^+$. If x and y are both n-exact and $|x-y| \leq min(|x|, |y|)$, then $x - y$ is n-exact.*

The next lemma characterizes the *successor* of a floating point number:

Lemma 2.18 *Let $x^+ = x + 2^{expo(x)+1-n}$, where $x \in \mathbf{R}$, $x > 0$, $n \in \mathbf{Z}^+$, and x is n-exact.*

(a) If $y \in \mathbf{R}$, $y > x$, and y is n-exact, then $y \geq x^+$;   (b) x^+ is n-exact.

Proof:

(a) We have $x2^{n-1-expo(x)} \in \mathbf{Z}$, and since $expo(y) \geq expo(x)$, $y2^{n-1-expo(x)} \in \mathbf{Z}$ as well. Thus, $(y-x)2^{n-1-expo(x)} \geq 1$, hence $y - x \geq 2^{expo(x)+1-n}$.

(b) Since $2^{expo(x)+1} > x$ and $2^{expo(x)+1}$ is n-exact, $2^{expo(x)+1} \geq x^+$ by (a). We may assume $x^+ < 2^{expo(x)+1}$, hence $expo(x^+) = expo(x)$. But then

$$x^+2^{n-1-expo(x^+)} = x2^{n-1-expo(x)} + 1 \in \mathbf{Z}.\Box$$

Corollary 2.19 *Let $x, y \in \mathbf{R}$ and $n \in \mathbf{Z}^+$. If x and y are n-exact and and $x \neq y$, then*

$$expo(x-y) \geq min(expo(x), expo(y)) + 1 - n$$

Proof: Since the case $xy < 0$ is trivial, we shall assume that $xy > 0$ and, without loss of generality, that $y > x > 0$. But then by Lemma 2.18,

$$expo(y-x) \geq expo(2^{expo(x)+1-n}) = expo(x) + 1 - n.\Box$$

A normal floating point number with an n-bit significand and an m-bit exponent is recognized by the following predicate:

Definition 2.5 *Let $x \in \mathbf{R}$, $n \in \mathbf{Z}^+$, and $m \in \mathbf{Z}^+$. Then x is an (n, m)-**floating point number** iff*

(a) x is n-exact, and (b) $-2^{m-1} + 1 \leq expo(x) \leq 2^{m-1}$.

The internal operations of the $AMD5_K86$ floating point unit are limited to operands with 64-bit significands and 17-bit exponents i.e., $(64, 17)$-floating point numbers. (This will be reflected in the semantics of the programming language described in the next section.) However, in accordance with the IEEE standard, the characterization of the input to our square root program is somewhat more complicated, in order to allow for *denormal numbers*, as defined in [6]. Following [10], we provide for denormal inputs with the following definition:

Definition 2.6 *Let $x \in \mathbf{R}$, $n \in \mathbf{Z}^+$, and $m \in \mathbf{Z}^+$. Then x is a* **generalized (n, m)-floating point number** *iff*

(a) x is n-exact, and (b) $-2^{m-1} + 2 - n \leq expo(x) \leq 2^{m-1}$.

As observed in [10], the generalized (n, m)-floating point numbers for given n and m include all of the IEEE normal and denormal numbers in the corresponding format. The square root program must accommodate inputs in the *double extended* format[5], which corresponds to $n = 64$ and $m = 15$. Thus, the hypothesis of our correctness theorem, Theorem 1, specifies that the program input is a generalized $(64, 15)$-floating point number.

2.3 Truncation

The most basic form of rounding is *truncation*, which rounds any real number toward 0.

Definition 2.7 *If $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$, then*

$$rnd(x, trunc, n) = trunc(x, n) = sgn(x)\lfloor 2^{n-1}sig(x)\rfloor 2^{expo(x)-n+1}.$$

Lemma 2.20 *If $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$, then $sgn(trunc(x, n)) = sgn(x)$.*

Proof: This follows from Lemma 2.8. □

Lemma 2.21 *If $x \in \mathbf{R}$, $x \neq 0$, and $n \in \mathbf{Z}^+$, then*

$$|x| \geq |trunc(x, n)| > |x| - 2^{expo(x)-n+1} \geq |x|(1 - 2^{-n+1}).$$

Proof: $|trunc(x, n)| \leq 2^{n-1}sig(x)2^{expo(x)-n+1} = sig(x)2^{expo(x)} = |x|$ and

$$|trunc(x, n)| > (2^{n-1}sig(x) - 1)2^{expo(x)-n+1} = |x| - 2^{expo(x)-n+1}.$$

The last inequality follows from Definition 2.3. □

Corollary 2.22 *If $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$, then $expo(trunc(x, n)) = expo(x)$.*

Proof: Since $|trunc(x, n)| \leq |x|$, $expo(trunc(x, n)) \leq expo(x)$. But since

$$|trunc(x, n)| \geq \lfloor 2^{n-1}\rfloor 2^{expo(x)-n+1} = 2^{expo(x)},$$

$expo(trunc(x, n)) \geq expo(x)$. □

Lemma 2.23 *Let $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$.*

(a) x is n-exact $\Leftrightarrow x = trunc(x, n)$;
(b) $trunc(x, n)$ is n-exact;
(c) If $a \in \mathbf{R}$, a is n-exact, and $a \leq |x|$, then $a \leq |trunc(x, n)|$.

Proof:
(a) Both conditions are clearly equivalent to $\lfloor 2^{n-1}sig(x)\rfloor = 2^{n-1}sig(x)$.
(b) Since $expo(trunc(x,n)) = expo(x)$, it suffices to observe that

$$trunc(x,n)2^{n-1-expo(x)} = sgn(x)\lfloor 2^{n-1}sig(x)\rfloor \in \mathbf{Z}.$$

(c) Suppose $a > |trunc(x,n|$. Then by Lemma 2.18,

$$|trunc(x,n)| \leq a - 2^{expo(trunc(x,n))-n+1} = a - 2^{expo(x)-n+1} \leq |x| - 2^{expo(x)-n+1}.$$

But

$$|trunc(x,n)| > (2^{n-1}sig(x) - 1)2^{expo(x)-n+1} = |x| - 2^{expo(x)-n+1}.\Box$$

Lemma 2.24 *If $x, y \in \mathbf{R}$, $0 < x \leq y$, $n \in \mathbf{Z}^+$, then $trunc(x,n) \leq trunc(y,n)$.*

Proof: By Lemma 2.21, $trunc(x,n) \leq x \leq y$. Since $trunc(x,n)$ is n-exact, Lemma 2.23 implies $trunc(x,n) \leq trunc(y,n)$. $\Box$

Lemma 2.25 *Let $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$. If x is not n-exact, then*

$$expo(x - trunc(x,n)) \leq expo(x) - n.$$

Proof: By Lemma 2.23, $x - trunc(x,n) \neq 0$, so that $2^{expo(x-trunc(x,n))} \leq |x - trunc(x,n)|$. But by Lemma 2.21, $|x - trunc(x,n)| < 2^{expo(x)-n+1}$. $\Box$

Lemma 2.26 *If $x \in \mathbf{R}$, $m, n \in \mathbf{Z}^+$, and $m \leq n$, then*

$$trunc(trunc(x,n),m) = trunc(x,m).$$

Proof: We assume $x \geq 0$; the case $x < 0$ follows easily.

$$\begin{aligned}
trunc(trunc(x,n),m) &= \lfloor 2^{m-1-expo(x)}(\lfloor 2^{n-1-expo(x)}x\rfloor 2^{expo(x)+1-n})\rfloor 2^{expo(x)+1-m} \\
&= \lfloor\lfloor 2^{n-1-expo(x)}x\rfloor / 2^{n-m}\rfloor 2^{expo(x)+1-m} \\
&= \lfloor 2^{n-1-expo(x)}x/2^{n-m}\rfloor 2^{expo(x)+1-m} \\
&= \lfloor 2^{m-1-expo(x)}x\rfloor 2^{expo(x)+1-m} \\
&= trunc(x,m).\Box
\end{aligned}$$

Lemma 2.27 *Let $x, y \in \mathbf{R}$, $x > 0$, $y > 0$, $k \in \mathbf{Z}^+$, and $n = k + expo(x) - expo(y)$. If $n > 0$ and x is n-exact, then*

$$x + trunc(y,k) = trunc(x+y, k + expo(x+y) - expo(y)).$$

Proof: Since x is n-exact, $x2^{k-1-expo(y)} = x2^{n-1-expo(x)} \in \mathbf{Z}$. Let $k' = k + expo(x+y) - expo(y)$. Then

$$\begin{aligned}
x + trunc(y,k) &= x + \lfloor 2^{k-1-expo(y)} y \rfloor 2^{expo(y)+1-k} \\
&= (x2^{k-1-expo(y)} + \lfloor 2^{k-1-expo(y)} y \rfloor) 2^{expo(y)+1-k} \\
&= \lfloor 2^{k-1-expo(y)} (x+y) \rfloor 2^{expo(y)+1-k} \\
&= \lfloor 2^{k'-1-expo(x+y)} (x+y) \rfloor 2^{expo(x+y)+1-k'} \\
&= trunc(x+y, k'). \square
\end{aligned}$$

Corollary 2.28 *Let $y \in \mathbf{R}$, $e \in \mathbf{Z}$, and $m, k \in \mathbf{Z}^+$. If $m \leq k+1$ and $0 < y < 2^e$, then*

$$trunc(2^e + trunc(y,k), m) = trunc(2^e + y, m).$$

Proof: Since $y < 2^e$, $expo(y) < e$; since $2^e < 2^e + y < 2^{e+1}$, $expo(2^e + y) = e$. Thus,

$$2^e + trunc(y,k) = trunc(2^e + y, k + e - expo(y))$$

by Lemma 2.27. Now by Lemma 2.26, since $m \leq k + 1 \leq k + e - expo(y)$,

$$\begin{aligned}
trunc(2^e + trunc(y,k), m) &= trunc(trunc(2^e + y, k + e - expo(y)), m) \\
&= trunc(2^e + y, m). \square
\end{aligned}$$

In our algorithmic language, which reflects implementation constraints, evaluation of $trunc(x,n)$ will be limited to $n \leq 64$. Since we shall require estimates with accuracy exceeding 64 bits, it will be useful to have a means for evaluating expressions of the form

$$trunc(x, n+k) - trunc(x,n)$$

that does not involve explicit evaluation of $trunc(x, n+k)$. This motivates the next lemma:

Lemma 2.29 *Let $x \in \mathbf{R}$, $x > 0$, and $k, n \in \mathbf{Z}^+$, $k \leq n$. Let $e = expo(x) + 1 - n$ and $z = trunc(x - trunc(x,n), n)$. Then*

$$trunc(x, n+k) - trunc(x,n) = [sig(trunc(2^e + z, k+1)) - 1]2^e.$$

Proof: First note that

$$trunc(x,n) = \lfloor 2^{n-1+expo(x)} x \rfloor 2^{expo(x)+1-n} = \lfloor 2^{-e} x \rfloor 2^e$$

and

$$trunc(x, n+k) = \lfloor 2^{n+k-1-expo(x)} x \rfloor 2^{expo(x)+1-(n+k)} = \lfloor 2^{k-e} x \rfloor 2^{e-k}.$$

Now let $y = x - trunc(x, n)$. Then $y < 2^e$, $k+1 \leq n+1$, and Corollary 2.28 yields

$$trunc(2^e + y, k+1) = trunc(2^e + trunc(y, n), k+1) = trunc(2^e + z, k+1).$$

Therefore, since $2^e + y < 2^{e+1}$, $expo(2^e + y) = e$ and

$$\begin{aligned}
&[sig(trunc(2^e + z, k+1)) - 1]2^e \\
&= sig(trunc(2^e + y, k+1))2^e - 2^e = trunc(2^e + y, k+1) - 2^e \\
&= \lfloor 2^{k-e}(2^e + y) \rfloor 2^{e-k} - 2^e = \lfloor 2^k + 2^{k-e} y \rfloor 2^{e-k} - 2^e \\
&= (2^k + \lfloor 2^{k-e} y \rfloor) 2^{e-k} - 2^e = \lfloor 2^{k-e} y \rfloor 2^{e-k} \\
&= \lfloor 2^{k-e}(x - trunc(x, n)) \rfloor 2^{e-k} = \lfloor 2^{k-e}(x - \lfloor 2^{-e} x \rfloor 2^e) \rfloor 2^{e-k} \\
&= \lfloor 2^{k-e} x - \lfloor 2^{-e} x \rfloor 2^k \rfloor 2^{e-k} = (\lfloor 2^{k-e} x \rfloor - \lfloor 2^{-e} x \rfloor 2^k) 2^{e-k} \\
&= \lfloor 2^{k-e} x \rfloor 2^{e-k} - \lfloor 2^{-e} x \rfloor 2^e = trunc(x, n+k) - trunc(x, n). \Box
\end{aligned}$$

2.4 *Away* Rounding

Rounding *away* from 0 is defined as follows:

Definition 2.8 *If $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$, then*

$$rnd(x, away, n) = away(x, n) = sgn(x) \lceil 2^{n-1} sig(x) \rceil 2^{expo(x)-n+1}.$$

Lemma 2.30 *For any $x \in \mathbf{R}$, $sgn(x) = sgn(away(x))$.*

Proof: This follows from Lemma 2.8. $\Box$

Lemma 2.31 *If $x \in \mathbf{R}$, $x \neq 0$, and $n \in \mathbf{Z}^+$, then*

$$|x| \leq |away(x, n)| < |x| + 2^{expo(x)-n+1} \leq |x|(1 + 2^{-n+1}).$$

Proof: $|away(x, n)| \geq 2^{n-1} sig(x) 2^{expo(x)-n+1} = sig(x) 2^{expo(x)} = |x|$ and

$$|away(x, n)| < (2^{n-1} sig(x) + 1) 2^{expo(x)-n+1} = |x| + 2^{expo(x)-n+1}. \Box$$

Lemma 2.32 *Let $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$.*

(a) x is n-exact $\Leftrightarrow x = away(x, n)$;
(b) $away(x, n)$ is n-exact;
(c) If $a \in \mathbf{R}$, a is n-exact, and $a \geq |x|$, then $a \geq |away(x, n)|$;

Proof:
(a) Both conditions are clearly equivalent to $2^{n-1} sig(x) \in \mathbf{Z}$.
(b) Since $sig(x) < 2$, $\lceil 2^{n-1} sig(x) \rceil \leq 2^n$. If equality holds, then

$$away(x, n) = sgn(x) 2^{expo(x)+1},$$

hence $expo(away(x,n)) = expo(x)+1$, and

$$away(x,n)2^{n-1-expo(away(x,n))} = sgn(x)2^{n-1} \in \mathbf{Z}.$$

Otherwise, $|away(x,n)| < 2^{expo(x)+1}$, hence $expo(away(x,n)) \leq expo(x)$, and we need only note that

$$away(x,n)2^{n-1-expo(x)} = sgn(x)\lceil 2^{n-1}sig(x)\rceil \in \mathbf{Z}.$$

(c) Suppose $a < |away(x,n|$. Then by Lemma 2.18,

$$|away(x,n)| \geq a + 2^{expo(a)-n+1} \geq |x| + 2^{expo(x)-n+1}.$$

But

$$|away(x,n)| < (2^{n-1}sig(x)+1)2^{expo(x)-n+1} = |x| + 2^{expo(x)-n+1}.\Box$$

Corollary 2.33 $|away(x,n)| \leq 2^{expo(x)+1}$.

Proof: This follows from Lemma 2.32(c). $\Box$

Lemma 2.34 *If $x, y \in \mathbf{R}$, $0 < x \leq y$, $n \in \mathbf{Z}^+$, then $away(x,n) \leq away(y,n)$.*

Proof: By Lemma 2.21, $away(y,n) \geq y \geq x$. Since $away(y,n)$ is n-exact, Lemma 2.32 implies $away(y,n) \geq away(x,n)$. $\Box$

Lemma 2.35 *If $x \in \mathbf{R}$, $m, n \in \mathbf{Z}^+$, and $m \leq n$, then*

$$away(away(x,n),m) = away(x,m).$$

Proof: We may assume $x > 0$. Consider first the case $away(x,n) = 2^{expo(x)+1}$. In this case, $away(x,n)$ is m-exact, so that $away(away(x,n),m) = away(x,n) = 2^{expo(x)+1}$. By Corollary 2.33, we need only show that $away(x,m) \geq 2^{expo(x)+1}$. But since $m \leq n$, $away(x,m)$ is n-exact, and since $away(x,m) \geq x$, $away(x,m) \geq away(x,n)$.

Thus, we may assume $away(x,n) < 2^{expo(x)+1}$ and $expo(away(x,n)) = expo(x)$, hence

$$\begin{aligned}
away(away(x,n),m) &= \lceil 2^{m-1-expo(x)}(\lceil 2^{n-1-expo(x)}x\rceil 2^{expo(x)+1-n})\rceil 2^{expo(x)+1-m} \\
&= \lceil\lceil 2^{n-1-expo(x)}x\rceil/2^{n-m}\rceil 2^{expo(x)+1-m} \\
&= \lceil 2^{n-1-expo(x)}x/2^{n-m}\rceil 2^{expo(x)+1-m} \\
&= \lceil 2^{m-1-expo(x)}x\rceil 2^{expo(x)+1-m} \\
&= away(x,m).\Box
\end{aligned}$$

2.5 *Near* Rounding

Our next mode rounds to the nearest representable number. Ambiguities are resolved by forcing the least significant bit to 0:

Definition 2.9 *Let $x \in \mathbf{R}$, $n \in \mathbf{Z}^+$, $z = \lfloor 2^{n-1}sig(x) \rfloor$, and $f = 2^{n-1}sig(x) - z$. Then*

$$rnd(x, near, n) = near(x,n) = \begin{cases} trunc(x,n) & \text{if } f < 1/2 \\ away(x,n) & \text{if } f > 1/2 \\ trunc(x,n) & \text{if } f = 1/2 \text{ and } z \text{ is even} \\ away(x,n) & \text{if } f = 1/2 \text{ and } z \text{ is odd.} \end{cases}$$

Lemma 2.36 *Let $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$.*

(a) If $|x - trunc(x,n)| < |x - away(x,n)|$, then $near(x,n) = trunc(x,n)$.
(b) If $|x - trunc(x,n)| > |x - away(x,n)|$, then $near(x,n) = away(x,n)$.

Proof: We may assume that $2^{n-1}sig(x) \notin \mathbf{Z}$, for otherwise

$$trunc(x,n) = away(x,n) = near(x,n) = x.$$

Let $f = 2^{n-1}sig(x) - \lfloor 2^{n-1}sig(x) \rfloor$. Then

$$\begin{aligned} |x - trunc(x,n)| &= |x| - |trunc(x,n)| \\ &= 2^{expo(x)+1-n}(2^{n-1}sig(x) - \lfloor 2^{n-1}sig(x) \rfloor) \\ &= 2^{expo(x)+1-n} f \end{aligned}$$

and

$$\begin{aligned} |x - away(x,n)| &= |away(x,n)| - |x| \\ &= 2^{expo(x)+1-n}(\lceil 2^{n-1}sig(x) \rceil - 2^{n-1}sig(x)) \\ &= 2^{expo(x)+1-n}(1 - f). \end{aligned}$$

Thus, (a) and (b) correspond to $f < 1/2$ and $f > 1/2$, respectively. □

Lemma 2.37 *Let $x, y \in \mathbf{R}$ and $n \in \mathbf{Z}^+$. If y is n-exact, then $|x - y| \geq |x - near(x,n)|$.*

Proof: Assume $|x - y| < |x - near(x,n)|$. We shall only consider the case $x > 0$, as the case $x < 0$ is handled similarly.

First suppose $near(x,n) = trunc(x,n)$. Since $near(x,n) \leq x$, we must have $y > near(x,n)$ and hence $y > x$ by Lemma 2.23. But since $away(x,n) - x \geq x - near(x,n)$ by Lemma 2.36, we also have $y < away(x,n)$, and hence $y < x$ by Lemma 2.32.

In the remaining case, $near(x,n) = away(x,n) > x$. Now $y < near(x,n)$ and by Lemma 2.32, $y < x$. But in this case, Lemma 2.36 implies $y > trunc(x,n)$, and hence $y > x$ by Lemma 2.23. □

Lemma 2.38 *Let $x, y \in \mathbf{R}$ and $n \in \mathbf{Z}^+$. If $0 \leq x \leq y$, then $near(x,n) \leq near(y,n)$.*

Proof: Suppose $x < y$ and $near(x,n) > near(y,n)$. Then Lemma 2.37 implies $x > near(y,n)$, otherwise $x \leq near(y,n) < near(x,n)$ and $|x - near(y,n)| < |x - near(x,n)|$. Similarly, $y < near(x,n)$, and thus $near(y,n) < x \leq y < near(x,n)$. Applying Lemma 2.37 again, we have $x - near(y,n) \geq near(x,n) - x$, and hence $2x \geq near(x,n) + near(y,n)$. Similarly, $near(x,n) - y \geq y - near(y,n)$, and hence $near(x,n) + near(y,n) \geq 2y$. Consequently, $2x \geq 2y$, contradicting $x < y$. □

Lemma 2.39 *Let $x, y \in \mathbf{R}$, $0 < x < y$, and $n \in \mathbf{Z}^+$. If $near(x,n) \neq near(y,n)$, then for some $a \in \mathbf{R}$, $x \leq a \leq y$ and a is $(n+1)$-exact.*

Proof: We may assume $expo(x) = expo(y)$, for otherwise the conclusion of the lemma is satisfied by $a = 2^{expo(x)+1}$. Suppose $near(x,n) \neq near(y,n)$. Then $near(x,n) < near(y,n)$ by Lemma 2.38. Let $a = (near(x,n) + near(y,n))/2$. Then $x \leq a$, for otherwise $|x - near(y,n)| < |x - near(x,n)|$, contradicting Lemma 2.37. Similarly, $y \geq a$.

It remains to show that a is $(n+1)$-exact. Let $e = expo(x) = expo(y)$. By Lemma 2.32, $a < near(y,n) \leq away(y,n) \leq 2^{e+1}$, and hence $expo(a) \leq e$. Therefore, it will suffice to show that $a2^{(n+1)-1-e} = a2^{n-e} \in \mathbf{Z}$. Since $near(x,n)$ and $near(y,n)$ are n-exact and

$$e = expo(trunc(x,n)) \leq expo(near(x,n)) \leq expo(near(y,n)),$$

we have $near(x,n)2^{n-1-e} \in \mathbf{Z}$ and $near(y,n)2^{n-1-e} \in \mathbf{Z}$. It follows that

$$a2^{n-e} = (near(x,n) + near(y,n))2^{n-1-e} \in \mathbf{Z}. \Box$$

We shall apply Lemma 2.39 in the following form:

Corollary 2.40 *Let $a, x, y \in \mathbf{R}$, $n, k \in \mathbf{Z}^+$, and $n \geq k$. If a is $(n+1)$-exact, and $0 < a < x$, and $0 < y < a + 2^{expo(a)-n}$, then $near(x,k) \geq near(y,k)$.*

Proof: By Lemma 2.38, we may assume $x < y$, so that $a < x < y < a + 2^{expo(a)-n}$. By Lemma 2.18, a and $a + 2^{expo(a)-n}$ are successive $(n+1)$-exact numbers, hence $near(x,k) = near(y,k)$ by Lemma 2.39. □

2.6 *Odd* Rounding

Odd rounding is defined as truncation followed by setting the least significant bit:

Definition 2.10 *Let $x \in \mathbf{R}$, $n \in \mathbf{Z}^+$, and $z = \lfloor 2^{n-1} sig(x) \rfloor$. Then*

$$rnd(x, odd, n) = odd(x,n) = \begin{cases} sgn(x)z2^{expo(x)+1-n} & \text{if } z \text{ is odd} \\ sgn(x)(z+1)2^{expo(x)+1-n} & \text{if } z \text{ is even.} \end{cases}$$

The following is an obvious consequence of the definition:

Lemma 2.41 *If $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$, then $|odd(x,n)| \geq |trunc(x,n)|$.*

The definition may be reformulated as follows:

Lemma 2.42 *If $x \in \mathbf{R}$, $n \in \mathbf{Z}$, and $n > 1$, then*

$$odd(x,n) = trunc(x,n-1) + sgn(x)2^{expo(x)+1-n}.$$

Proof: Let $z = \lfloor 2^{n-1} sig(x) \rfloor$. Then

$$\begin{aligned}
trunc(x,n-1) &= sgn(x)\lfloor 2^{n-2-expo(x)}x \rfloor 2^{expo(x)+2-n} \\
&= sgn(x)\lfloor 2^{n-1-expo(x)}x/2 \rfloor 2^{expo(x)+2-n} \\
&= sgn(x)\lfloor \lfloor 2^{n-1-expo(x)}x \rfloor /2 \rfloor 2^{expo(x)+2-n} \\
&= sgn(x) \left\lfloor \frac{z}{2} \right\rfloor 2^{expo(x)+2-n}.
\end{aligned}$$

If z is odd, then $\lfloor \frac{z}{2} \rfloor = \frac{z-1}{2}$ and

$$\begin{aligned}
trunc(x,n-1) &= sgn(x)\frac{z-1}{2}2^{expo(x)+2-n} \\
&= sgn(x)(z-1)2^{expo(x)+1-n} \\
&= odd(x,n) - sgn(x)2^{expo(x)+1-n}.
\end{aligned}$$

If z is even, then $\lfloor \frac{z}{2} \rfloor = \frac{z}{2}$ and

$$\begin{aligned}
trunc(x,n-1) &= sgn(x)\frac{z}{2}2^{expo(x)+2-n} \\
&= sgn(x)(z+1-1)2^{expo(x)+1-n} \\
&= odd(x,n) - sgn(x)2^{expo(x)+1-n}.\square
\end{aligned}$$

Lemma 2.43 *If $x \in \mathbf{R}$, $n \in \mathbf{Z}$, $x \neq 0$, and $n > 1$, then*

(a) $sgn(odd(x,n)) = sgn(x)$;

(b) $expo(odd(x,n)) = expo(x)$.

Proof:

(a) This is an immediate consequence of Definition 2.10.

(b) Since $trunc(x,n-1)$ and $2^{expo(x)+1}$ are both $(n-1)$-exact and

$$trunc(x,n-1) < 2^{expo(trunc(x,n-1))+1} = 2^{expo(x)+1},$$

Lemma 2.18 implies

$$2^{expo(x)+1} \geq trunc(x, n-1) + 2^{expo(x)+1-(n-1)} > odd(x, n),$$

and hence $expo(odd(x, n)) \leq expo(x)$. But since $odd(x, n) > trunc(x, n-1)$,

$$expo(odd(x, n)) \geq expo(trunc(x, n-1)) = expo(x).\Box$$

Lemma 2.44 *If $x \in \mathbf{R}$, $n \in \mathbf{Z}$, $x > 0$, and $n > 1$, then $odd(x, n)$ is n-exact but not $(n-1)$-exact.*

Proof: Since $trunc(x, n-1)$ is $(n-1)$-exact,

$$\begin{aligned} 2^{n-1-expo(x)} odd(x, n) &= 2^{n-1-expo(x)}[trunc(x, n-1) + 2^{expo(x)+1-n}] \\ &= 2[2^{n-2-expo(x)} trunc(x, n-1)] + 1 \end{aligned}$$

is an odd integer. $\Box$

Lemma 2.45 *If $x \in \mathbf{R}$, $n, m \in \mathbf{Z}^+$, $x > 0$, and $n > m$, then*

$$trunc(odd(x, n), m) = trunc(x, m).$$

Proof: Let $e = expo(x) = expo(odd(x, n))$. Then

$$\begin{aligned} trunc(odd(x, n), n-1) &= \lfloor 2^{n-2-e} odd(x, n) \rfloor 2^{e+2-n} \\ &= \lfloor 2^{n-2-e} (\lfloor 2^{n-2-e} x \rfloor 2^{e+2-n} + 2^{e+1-n}) \rfloor 2^{e+2-n} \\ &= \left\lfloor \lfloor 2^{n-2-e} x \rfloor + \frac{1}{2} \right\rfloor 2^{e+2-n} \\ &= \lfloor 2^{n-2-e} x \rfloor 2^{e+2-n} \\ &= trunc(x, n-1), \end{aligned}$$

and by Lemma 2.26,

$$\begin{aligned} trunc(odd(x, n), m) &= trunc(trunc(odd(x, n), n-1), m) \\ &= trunc(trunc(x, n-1), m) \\ &= trunc(x, m).\Box \end{aligned}$$

Lemma 2.46 *Let $x, y \in \mathbf{R}$, $x > 0$, $y > 0$, $k \in \mathbf{Z}$, and $n = k-1+expo(x)-expo(y)$. If $k > 1$, $n > 0$, and x is n-exact, then*

$$x + odd(y, k) = odd(x + y, k + expo(x + y) - expo(y)).$$

Proof: Let $k' = k + expo(x+y) - expo(y)$. Then by Lemma 2.27,

$$\begin{aligned} x + odd(y,k) &= x + trunc(y, k-1) + 2^{expo(y)+1-k} \\ &= trunc(x+y, k-1+expo(x+y)-expo(y)) + 2^{expo(y)+1-k} \\ &= trunc(x+y, k'-1) + 2^{expo(x+y)+1-k'} \\ &= odd(x+y, k'). \square \end{aligned}$$

Lemma 2.47 *Let $x, y \in \mathbf{R}$ with $x > y > 0$ and $m, k \in \mathbf{Z}$ with $m-2 \geq k > 0$. Then*
(a) $trunc(x,k) \geq trunc(odd(y,m),k)$;
(b) $away(x,k) \geq away(odd(y,m),k)$;
(c) $near(x,k) \geq near(odd(y,m),k)$.

Proof:
(a) By Lemmas 2.24 and 2.45, since $x \geq y$ and $m > k$,

$$trunc(x,k) \geq trunc(y,k) = trunc(odd(y,m),k).$$

(b) Note that $away(x,k)$ and $trunc(y, m-1)$ are both $(m-1)$-exact. Since

$$away(x,k) \geq x > y \geq trunc(y, m-1),$$

$$away(x,k) \geq trunc(y, m-1) + 2^{expo(y)+1-(m-1)} > odd(y,m)$$

by Lemma 2.18. Thus, by Lemma 2.32,

$$away(x,k) \geq away(odd(y,m),k).$$

(c) We shall apply Corollary 2.40, substituting $m-2$ for n, $trunc(y, m-1)$ for a, and $odd(y,m)$ for y. To establish the hypotheses of the corollary, we note that

$$m - 2 \geq k > 0,$$

$$trunc(y, m-1) \text{ is } (m-1)\text{-exact},$$

$$0 < trunc(y, m-1) \leq y < x,$$

and

$$\begin{aligned} 0 < odd(y,m) &= trunc(y, m-1) + 2^{expo(trunc(y,m-1))-m+1} \\ &< trunc(y, m-1) + 2^{expo(trunc(y,m-1))-(m-2)}. \end{aligned}$$

Consequently, $near(x,k) \geq near(odd(y,m),k)$. $\square$

2.7 Other Rounding Modes

The IEEE specification supports the rounding modes *trunc*, *near*, and two others:

Definition 2.11 *If $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$, then*

$$rnd(x, inf, n) = inf(x, n) = \begin{cases} away(x, n) & x \geq 0 \\ trunc(x, n) & x < 0. \end{cases}$$

Definition 2.12 *If $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$, then*

$$rnd(x, minf, n) = minf(x, n) = \begin{cases} trunc(x, n) & x \geq 0 \\ away(x, n) & x < 0. \end{cases}$$

Definition 2.13 *An element of the set $\{trunc, inf, minf, near\}$ is called an* **IEEE rounding mode**.

In our application, the modes inf and $minf$ will be be applied only to positive arguments and are therefore subsumed by *away* and *trunc*, respectively.

Although it is not used in any critical way, one other rounding mode occurs in the square root algorithm and therefore must be mentioned here:

Definition 2.14 *Given $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$, let $z = \lfloor 2^{n-1} sig(x) \rfloor$. Then*

$$rnd(x, sticky, n) = sticky(x, n) = \begin{cases} trunc(x, n) & \textit{if } z \textit{ is odd} \\ away(x, n) & \textit{if } z \textit{ is even.} \end{cases}$$

Every application of *sticky* rounding that occurs in *FSQRT* is trivial, i.e., the first argument is exact with respect to the second. Therefore, the only property of this rounding mode that we shall require is the following, which is an immediate consequence of Lemmas 2.23 and 2.32:

Lemma 2.48 *Let $x \in \mathbf{R}$ and $n \in \mathbf{Z}^+$. If x is n-exact, then $sticky(x, n) = x$.*

3 Specification of the Algorithm

3.1 Informal Description

The square root algorithm computes an approximation to the square root of a given nonnegative number. The inputs to the algorithm are the radicand P, a rounding mode *mode*, and a degree of precision *prec*. At termination, the value of the variable *sqrt* is $rnd(\sqrt{P}, mode, prec)$. In most cases, the computation involves 21 multiplications and 15 additions; the worst case requires 23 multiplications and 16 additions.

The algorithm consists of the following steps:

(1) If $P = 0$, then the algorithm simply terminates with $sqrt = 0$. Otherwise, proceed to (2).

(2) An initial approximation r_0 to $1/\sqrt{P}$, accurate to 5 bits, is derived from a table.

(3) Using r_0 as a seed, three successive Newton-Raphson iterations yield increasingly accurate estimates of $1/\sqrt{P}$: r_1, r_2, and r_3.

(4) (Test for exactness) The product $r_3 P$ is truncated to 64 bits and rounded away to 32 bits, to produce two approximations to $\sqrt{P}$, denoted by q and q_0, respectively. If P has a floating point square root, then it must coincide with the 32-bit approximation q_0. In this case, execution terminates with the value $rnd(q_0, mode, prec)$. Otherwise, proceed to (5).

(5) (Refinement) The 64-bit approximation q is accurate to 39 bits. A correction term res is computed as an approximation to $\sqrt{P} - q$. The resulting estimate $q + res$ is accurate to 74 bits.

(6) (Final Computation) A test is applied to determine whether $q + res$ is close to a 64-bit rounding boundary. If not, then execution terminates with the value $rnd(q + res, mode, prec)$. Otherwise, further analysis is required to handle the boundary cases.

3.2 Formalization

The algorithm is represented in Figure 1 as a program in a simple programming language. The ACL2 formulation of the correctness of the algorithm requires a rigorous definition of the syntax and semantics of this language, which we shall only sketch here.

The language is based on two types of expressions:

(1) A *Boolean expression* is either of the following:

  (a) $x = y$, $x < y$, or $x \leq y$, where x and y are variables or constants;

  (b) $\alpha \wedge \beta$, $\alpha \vee \beta$, or $\neg\alpha$, where α and β are Boolean expressions.

(2) A *numerical expression* is one of the following:

  (a) $rnd(x, mode, n)$, $rnd(x + y, mode, n)$, or $rnd(xy, mode, n)$, where x and y are variables or constants, *mode* is a rounding mode, and n is an integer satisfying $0 < n \leq 64$;

  (b) $sig(x)$, where x is a variable;

if $P = 0$ then $sqrt \leftarrow 0$
else
(table access *)*
$r_0 \leftarrow lookup(P)$

(1st NR iteration *)*
$P_h \leftarrow trunc(P, 32)$
$s_0 \leftarrow away(r_0 P_h, 32)$
$t_0 \leftarrow away(\frac{1}{2} r_0, 32)$
$u_0 \leftarrow away(s_0 r_0, 32)$
$v_0 \leftarrow trunc(3 - u_0, 32)$
$r_1 \leftarrow trunc(v_0 t_0, 32)$

(2nd NR iteration *)*
$s_1 \leftarrow away(r_1 P_h, 32)$
$t_1 \leftarrow away(\frac{1}{2} r_1, 32)$
$u_1 \leftarrow away(s_1 r_1, 32)$
$v_1 \leftarrow trunc(3 - u_1, 32)$
$r_2 \leftarrow trunc(v_1 t_1, 32)$

(3rd NR iteration *)*
$s_2 \leftarrow away(r_2 P, 64)$
$t_2 \leftarrow away(\frac{1}{2} r_2, 64)$
$u_2 \leftarrow away(s_2 r_2, 64)$
$v_2 \leftarrow trunc(3 - u_2, 64)$
$r_3 \leftarrow trunc(v_2 t_2, 64)$

(test for exactness *)*
$q \leftarrow trunc(r_3 P, 64)$
$q_0 \leftarrow away(r_3 P, 32)$
$P_0 \leftarrow sticky(q_0^2, 64)$
$rem_0 \leftarrow sticky(P - P_0, 64)$
if $rem_0 = 0$ then
$sqrt \leftarrow rnd(q_0, mode, prec)$

else *(* refinement *)*
$diff \leftarrow near(q - q_0, 64)$
$q_2 \leftarrow near(2q_0, 64)$
$prod \leftarrow away(q_2 diff, 64)$
$sq \leftarrow away(diff^2, 64)$
$t_3 \leftarrow near(\frac{1}{2} r_3, 64)$
$rem_1 \leftarrow sticky(rem_0 - prod, 64)$
$rem_2 \leftarrow trunc(rem_1 - sq, 64)$
$res \leftarrow odd(rem_2 t_3, 64)$

(final computation *)*
$root_0 \leftarrow trunc(q + res, 64)$
$root_1 \leftarrow near(q + res, 64)$
$root_2 \leftarrow away(q + res, 64)$
$used \leftarrow trunc(root_0 - q, 64)$
$end \leftarrow trunc(res - used, 64)$
$ulp \leftarrow trunc(root_2 - root_0, 64)$
$aeb \leftarrow trunc(ulp + end, 9)$
$bits \leftarrow sig(aeb)$
if $bits \notin \{2 - 2^{-8}, 1 + 2^{-1} - 2^{-8}\}$ then
$sqrt \leftarrow rnd(q + res, mode, prec)$
else if $mode \neq near$ then
$det \leftarrow away(root_2 root_2, 64)$
if $P \geq det$
then $sqrt \leftarrow rnd(root_1 + end, mode, prec)$
else $sqrt \leftarrow rnd(root_0 + end, mode, prec)$
else *(* mode = near *)*
$det \leftarrow trunc(root_1 root_2, 64)$
if $P \leq det$ then
$sqrt \leftarrow near(root_0 + end, prec)$
else if $bits = 2 - 2^{-8}$ then
$ends \leftarrow trunc(\frac{1}{16} end, 64)$
$sqrt \leftarrow near(root_2 + ends, prec)$
else $sqrt \leftarrow near(root_2 - end, prec)$

Figure 1: The Square Root Algorithm: Inputs are P, $mode$, and $prec$; Output is $sqrt$.

(c) $lookup(x)$, where x is a variable.

We define a *statement list* to be a list $\langle s_1, s_2, \ldots, s_n\rangle$, $n \geq 1$, where for $i = 1, \ldots, n-1$, s_i is an *assignment statement* of the form

$$\omega \leftarrow E,$$

where ω is a variable and E is a numerical expression, and s_n is either an assignment statement or a *conditional statement* of the form

$$\text{if } B \text{ then } Q \text{ else } R,$$

where B is a Boolean expression and Q and R are statement lists.

A *program* in our language is a pair $\mathcal{P} = (I, S)$, where I is a list of *input variables* and S is a statement list.

Pending the definition of the function *lookup* (Section 4), the definition of the value of a (Boolean or numerical) expression for a given set of variable bindings is obvious. In support of our previous claim that this formal language resembles the K5 microcode, we note (looking ahead to Definition 4.2) that all operations involved in expression evaluation are easily implementable at the microcode level, including the predefined arithmetic primitives (which are presumed to be IEEE-compliant), the extraction of various floating point fields, and a simple table reference.

It is also a straightforward exercise to define the function *execute*, which returns the final set of bindings for all of the variables of $\mathcal{P}$, $execute(\mathcal{P}, V)$, produced by executing $\mathcal{P}$ under a given list of input bindings V. Recall, however, that the architecture of the $AMD5_K86$ floating point unit requires that the evaluation of any numerical expression results in a $(64, 17)$-floating point number. We shall say that $\mathcal{P}$ *executes successfully* for a given list of input bindings if this constraint is satisfied.

3.3 Statement of Correctness

The discussion to follow will be based on a fixed execution of $FSQRT$ determined by a given set of values for the input variables P, *mode*, and *prec*. It is easily verified by inspection of Figure 1 that in any execution of $FSQRT$, each program variable is assigned a value at most once. Thus, any reference to a program variable will be understood unambiguously to refer to the value of that variable produced by this execution. Our objective is to prove the following theorem, which represents the IEEE specification [6]:

Theorem 1 *Let P be a generalized $(64, 15)$-floating point number, $P \geq 0$. Let mode be an IEEE rounding mode and let $prec \in \mathbf{Z}^+$, $prec \leq 64$. Then $FSQRT$ executes successfully on $\langle P, mode, prec\rangle$ and*

$$sqrt = rnd(\sqrt{P}, mode, prec).$$

However, because of its reference to the square root, Theorem 1 is not formalizable in the ACL2 logic, which does not provide for the real numbers. The theorem of rational arithmetic that best approximates Theorem 1 is the following:

Theorem 2 *Let P be a generalized $(64, 15)$-floating point number, $P \geq 0$ Let mode be an IEEE rounding mode and let prec $\in \mathbf{Z}^+$, prec ≤ 64. Let $\ell, h \in \mathbf{Q}$ such that $\ell \geq 0$, $h \geq 0$, and $\ell^2 \leq P \leq h^2$. Then FSQRT executes successfully on $\langle P, mode, prec\rangle$ and*

$$rnd(\ell, mode, prec) \leq sqrt \leq rnd(h, mode, prec).$$

The equivalence of these two theorems can be proved easily, although without the support of ACL2. The proof of Theorem 2 as a consequence of Theorem 1 requires only the monotonicity of *mode*, which is guaranteed by Lemmas 2.24, 2.34, and 2.38. The converse implication, which is of greater concern to us, can be derived from the well known property of the real numbers that every nonempty interval contains a rational, along with the following result:

Lemma *Let $x, y \in \mathbf{R}$ and let mode be an IEEE rounding mode. Assume x is irrational and $x > 0$. Then for every $n \in \mathbf{Z}^+$, there exists $\delta > 0$ such that if $|y - x| < \delta$, then $rnd(y, mode, n) = rnd(x, mode, n)$.*

Proof: Let $a = trunc(x, n + 1)$ and $b = a + 2^{expo(a)-n}$. By Lemmas 2.18, 2.21, and 2.23, a and b are $(n + 1)$-exact and since x is not, $0 < a < x < b$. We shall show that the conclusion of the lemma is satisfied by $\delta = min(x - a, b - x)$.

The case $mode = near$ is covered by Corollary 2.40. We need only consider the modes $trunc$ and $away$, which subsume inf and $minf$.

Suppose $mode = trunc$. By Lemma 2.26, $trunc(x, n) = trunc(a, n)$. By Lemma 2.24, $trunc(a, n) \leq trunc(y, n)$. Since $trunc(y, n)$ is n-exact by Lemma 2.23, $trunc(y, n) \leq a$ by Lemma 2.18. But now it follows from Lemma 2.23 that $trunc(y, n) \leq trunc(a, n)$, hence

$$trunc(y, n) = trunc(a, n) = trunc(x, n).$$

For the case $mode = away$, it follows from Lemmas 2.18, 2.31, and 2.32 that $b = away(x, n+1)$, hence $away(x, n) = away(b, n)$ by Lemma 2.35. By Lemma 2.34, $away(y, n) \leq away(b, n)$. Since $away(y, n)$ is n-exact, $away(y, n) \geq b$. But now Lemma 2.32 implies $away(y, n) \leq away(b, n)$, hence

$$away(y, n) = away(a, n) = away(x, n).\Box$$

Proof of Theorem 1: Suppose $sqrt \neq rnd(\sqrt{P}, mode, prec)$. Since Theorem 2 precludes the case $\sqrt{P} = \ell = h \in \mathbf{Q}$, we may assume that $\sqrt{P}$ is irrational. Choose δ as specified by the lemma above.

Suppose $sqrt < rnd(\sqrt{P}, mode, prec)$. Choose a rational ℓ in the interval $(\sqrt{P} - \delta, \sqrt{P})$. Then $\ell^2 < P$ and

$$rnd(\ell, mode, prec) = rnd(\sqrt{P}, mode, prec) > sqrt,$$

contradicting Theorem 2.

Similarly, if $sqrt > rnd(\sqrt{P}, mode, prec)$, then we choose a rational h in the interval $(\sqrt{P}, \sqrt{P} + \delta)$, which must satisfy $h^2 > P$ and

$$rnd(h, mode, prec) = rnd(\sqrt{P}, mode, prec) < sqrt,$$

again contradicting Theorem 2. $\square$

Our goal, then, is to prove Theorem 2. In order to facilitate the ACL2 formalization, we meticulously avoid any reference to $\sqrt{P}$ in the statements and proofs of the lemmas to follow. Moreover, all of the results of Section 2 remain valid if every reference to $\mathbf{R}$ is replaced by $\mathbf{Q}$. Thus, our proof of Theorem 2 is based entirely on rational arithmetic. In fact, with the exception of Theorem 1 and its lemma, every definition, lemma, and theorem presented in this paper has been formally encoded in the ACL2 logic, and every proof has been checked mechanically with the ACL2 prover.

Henceforth, we shall assume that P, $mode$, and $prec$ satisfy the hypothesis of Theorem 2. Note that in the case $P = 0$, we have $sqrt = 0$ and Theorem 2 holds trivially. Thus, we shall further assume that $P > 0$. In the next five sections, we shall show the value of $sqrt$ is as specified in the theorem. Then, in Section 9, we shall complete the proof by demonstrating that the required exponent bound is satisfied by the value of each numerical expression.

4 Initial Approximation

$$\boxed{\begin{array}{l} r_0 \leftarrow lookup(P) \\ \vdots \end{array}}$$

The initial approximation to $1/\sqrt{P}$, r_0, is based on a table, each entry of which associates a 7-bit key with a 6-bit value. The key is determined by the least significant bit of $expo(P)$ and the most significant 6 bits of the fractional part of $sig(P)$. The associated value is interpreted as the fractional part of $sig(r_0)$.

We shall represent the table as two sequences of 2^6 integers each, corresponding to the two possible values of $rem(expo(P), 2)$:

Definition 4.1 *$\mathcal{E}$ and $\mathcal{O}$ are the following sequences of binary integers:*

$$
\begin{aligned}
\mathcal{E} = \langle & 000000, 111111, 111110, 111101, 111100, 111011, 111010, 111001, \\
& 111000, 110111, 110111, 110110, 110101, 110100, 110011, 110011, \\
& 110010, 110001, 110001, 110000, 101111, 101111, 101110, 101101, \\
& 101101, 101100, 101011, 101011, 101010, 101010, 101001, 101001, \\
& 101000, 100111, 100111, 100110, 100110, 100101, 100101, 100100, \\
& 100100, 100011, 100011, 100010, 100010, 100010, 100001, 100001, \\
& 100000, 100000, 011111, 011111, 011111, 011110, 011110, 011101, \\
& 011101, 011101, 011100, 011100, 011011, 011011, 011011, 011010\rangle
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{O} = \langle & 011010, 011001, 011001, 011000, 010111, 010111, 010110, 010101, \\
& 010101, 010100, 010100, 010011, 010011, 010010, 010001, 010001, \\
& 010000, 010000, 001111, 001111, 001111, 001110, 001110, 001101, \\
& 001101, 001100, 001100, 001011, 001011, 001011, 001010, 001010, \\
& 001001, 001001, 001001, 001000, 001000, 001000, 000111, 000111, \\
& 000111, 000110, 000110, 000101, 000101, 000101, 000101, 000100, \\
& 000100, 000100, 000011, 000011, 000011, 000010, 000010, 000010, \\
& 000010, 000001, 000001, 000001, 000001, 000000, 000000, 000000\rangle,
\end{aligned}
$$

which will be denoted as

$$\mathcal{E} = \langle \alpha_0, \ldots, \alpha_{63} \rangle$$

and

$$\mathcal{O} = \langle \beta_0, \ldots, \beta_{63} \rangle.$$

Let $e = expo(P)$, $b = rem(e,2)$, $h = quot(e,2)$, $s = sig(P)$, and $c = trunc(s,7)$. Then $sig(r_0)$ will be derived from the table by

$$sig(r_0) = \begin{cases} 1 + 2^{-6}\alpha_i & \text{if } b = 0 \\ 1 + 2^{-6}\beta_i & \text{if } b = 1, \end{cases}$$

where $i = 2^6(c-1)$. To compute $expo(r_0) = expo(1/\sqrt{P})$, we note that

$$\frac{1}{\sqrt{P}} = \frac{1}{\sqrt{2^{2h+b}s}} = 2^{-(h+1)}\frac{2}{\sqrt{2^b s}},$$

where $1 < 2/\sqrt{2^b s} \leq 2$. Thus, $expo(r_0) = -(h+1)$ except when $b = 0$ and $s = 1$, in which case $expo(r_0) = -h$. This motivates the following:

Definition 4.2 *Let* $e = expo(P)$, $b = rem(e,2)$, $h = quot(P,2)$, $s = sig(P)$, $c = trunc(s,7)$, *and* $i = 2^6(c-1)$. *Then*

$$lookup(P) = \begin{cases} (1 + 2^{-6}\alpha_i)/2^{h+1} & \text{if } b = 0 \text{ and } i \neq 0 \\ (1 + 2^{-6}\alpha_i)/2^h & \text{if } b = i = 0 \\ (1 + 2^{-6}\beta_i)/2^{h+1} & \text{if } b = 1. \end{cases}$$

The table values were actually computed as

$$\alpha_i = 2^6 \left(trunc \left[sig \left(\frac{1}{\sqrt{1 + i \cdot 2^{-6}}} \right), 7 \right] - 1 \right)$$

and

$$\beta_i = 2^6 \left(trunc \left[sig \left(\frac{1}{\sqrt{2(1 + i \cdot 2^{-6})}} \right), 7 \right] - 1 \right).$$

Using this information, we could easily show that

$$lookup(P) = 2^{-h} trunc \left(\frac{1}{\sqrt{2^b c}}, 7 \right) = trunc \left(\frac{1}{\sqrt{trunc(P, 7)}}, 7 \right),$$

from which our desired estimate, Lemma 4.2, could be directly derived. However, like all of our results, this lemma must be proved without any reference to the square root function.

The only approach left open to us requires considerable computation. The results of this computation are collected in the next lemma. Each inequality stated in the lemma has been checked mechanically and will not be proved here.

Lemma 4.1

(a) For $i = 1, \ldots, 63$,

$$1 - 2^{-5} \leq (1 + 2^{-6} i)(1 + 2^{-6}\alpha_i)^2/4 < (1 + 2^{-6}(i+1))(1 + 2^{-6}\alpha_i)^2/4 \leq 1 + 2^{-5};$$

(b) For $i = 0$,

$$1 - 2^{-5} \leq (1 + 2^{-6} i)(1 + 2^{-6}\alpha_i)^2 < (1 + 2^{-6}(i+1))(1 + 2^{-6}\alpha_i)^2 \leq 1 + 2^{-5};$$

(c) For $i = 0, \ldots, 63$,

$$1 - 2^{-5} \leq (1 + 2^{-6} i)(1 + 2^{-6}\beta_i)^2/2 < (1 + 2^{-6}(i+1))(1 + 2^{-6}\beta_i)^2/2 \leq 1 + 2^{-5}.$$

Lemma 4.2 $|1 - Pr_0^2| \leq 2^{-5}$.

Proof: Let e, h, b, s, c, and i be defined as in Definition 4.2. By Corollary 2.22 and Lemma 2.21, $expo(c) = expo(s) = 0$ and $c \leq s < c + 2^{-6}$. But

$$1 + 2^{-6} i = 1 + 2^{-6} 2^6 (c - 1) = c,$$

hence

$$1 + 2^{-6} i \leq s < 1 + 2^{-6}(i+1).$$

Consider the case $b = 0$, $i \neq 0$. In this case,

$$Pr_0^2 = 2^e s (lookup(P))^2 = 2^{2h} s (1 + 2^{-6}\alpha_i)^2 / 2^{2h+2} = s(1 + 2^{-6}\alpha_i)^2/4,$$

and hence, using the above estimate for s, we have

$$(1 + 2^{-6} i))(1 + 2^{-6}\alpha_i)^2/4 \leq Pr_0^2 < (1 + 2^{-6}(i+1)))(1 + 2^{-6}\alpha_i)^2/4,$$

and the descired inequality follows from Lemma 4.1(a).

The two remaining cases are similar. □

5 Newton-Raphson Approximation

5.1 Recurrence Formula

Given an initial approximation r_0 of a root of a differentiable function f, the Newton-Raphson formula

$$r_{i+1} = r_i - \frac{f(r_i)}{f'(r_i)}$$

produces, under suitable conditions, a quadratically convergent sequence of estimates. We are interested here in the function

$$f(x) = 1/x^2 - P,$$

which has a unique positive root $1/\sqrt{P}$. In this case, the recurrence formula reduces simply to

$$r_{i+1} = \frac{r_i}{2}(3 - Pr_i^2).$$

As noted earlier, however, our statement and derivation of the relevant convergence theorem must be limited to rational arithmetic. The following result is sufficient for our purpose:

Lemma 5.1 *Let $P, x \in \mathbf{Q}$ with $0 \leq Px^2 \leq 4$ and let $y = \frac{x}{2}(3 - Px^2)$. Then*

$$0 \leq 1 - Py^2 \leq (1 - Px^2)^2.$$

Proof: Let $U = Px^2$. Then

$$1 - Py^2 = 1 - \frac{Px^2}{4}(3 - Px^2)^2 = 1 - \frac{U}{4}(3 - U)^2.$$

Since $U \leq 4$,

$$0 \geq (U - 1)^2(U - 4) = U^3 - 6U^2 + 9U - 4,$$

hence

$$4 \geq U^3 - 6U^2 + 9U = U(3 - U)^2,$$

and it follows that $1 - Py^2 \geq 0$.

Since

$$\begin{aligned} 4((1-U)^2 - (1 - Py^2)) &= 4 - 8U + 4U^2 - 4 + U(3-U)^2 \\ &= U^3 - 2U^2 + U \\ &= U(U-1)^2 \\ &\geq 0, \end{aligned}$$

$1 - Py^2 \leq (1 - U)^2 = (1 - Px^2)^2$. □

We shall apply Lemma 5.1 in estimating the values of the variables r_1, r_2, and r_3, which are produced by three iterations of the Newton-Raphson formula, using r_0 as a seed. Of course, due to rounding error, these values will not be precisely as predicted by the formula.

5.2 First Iteration

The first Newton-Raphson iteration yields an approximation r_1, based on the seed r_0:

$$\begin{array}{|l|}\hline \vdots \\ P_h \leftarrow trunc(P, 32) \\ s_0 \leftarrow away(r_0 P_h, 32) \\ t_0 \leftarrow away(\frac{1}{2} r_0, 32) \\ u_0 \leftarrow away(s_0 r_0, 32) \\ v_0 \leftarrow trunc(3 - u_0, 32) \\ r_1 \leftarrow trunc(v_0 t_0, 32) \\ \vdots \\ \hline \end{array}$$

Lemma 5.2 $|1 - Pr_1^2| \leq 2^{-10}(1 + 2^{-16})$.

Proof: The proof of the inequality uses Lemmas 4.2 and 5.1, as well as 2.13, 2.21, 2.31, 2.2, 2.4, and 2.5.

First, we have

$$P(1 - 2^{-31}) \leq P_h \leq P,$$

$$r_0 P(1 - 2^{-31}) \leq r_0 P_h \leq s_0 \leq r_0 P_h(1 + 2^{-31}) \leq r_0 P(1 + 2^{-31}),$$

and

$$r_0^2 P(1 - 2^{-31}) \leq s_0 r_0 \leq u_0 \leq s_0 r_0(1 + 2^{-31}) \leq r_0^2 P(1 + 2^{-31})^2 < r_0^2 P(1 + 2^{-29}).$$

Note that $Pr_0^2 \leq 1 + 2^{-5} < 3/2$, and therefore $u_0 < 3$. Thus,

$$v_0 \leq 3 - u_0 \leq 3 - r_0^2 P(1 - 2^{-31}) \leq (3 - r_0^2 P)(1 + 2^{-31}),$$

and

$$\begin{aligned} v_0 &\geq (3 - u_0)(1 - 2^{-31}) \geq (3 - r_0^2 P(1 + 2^{-29}))(1 - 2^{-31}) \\ &\geq (3 - r_0^2 P)(1 - 2^{-29})(1 - 2^{-31}) \geq (3 - r_0^2 P)(1 - 2^{-28}). \end{aligned}$$

Since $r_0/2$ is 32-exact, $t_0 = r_0/2$.Thus,

$$r_1 \leq v_0 t_0 \leq \frac{r_0}{2}(3 - r_0^2 P)(1 + 2^{-31}),$$

and

$$Pr_1^2 \leq \frac{Pr_0^2}{4}(3 - r_0^2 P)^2(1 + 2^{-29}).$$

Since $Pr_0^2 \leq 1+2^{-5} < 4$, Lemma 5.1 yields $\frac{Pr_0^2}{4}(3-r_0^2P)^2 \leq 1$, hence $Pr_1^2 \leq 1+2^{-29}$ and $1 - Pr_1^2 \geq -2^{-29}$.

Similarly,

$$r_1 \geq v_0t_0(1-2^{-31}) \geq \frac{r_0}{2}(3-r_0^2P)(1-2^{-28})(1-2^{-31}) \geq \frac{r_0}{2}(3-r_0^2P)(1-2^{-27}).$$

Thus,

$$Pr_1^2 \geq \frac{Pr_0^2}{4}(3-r_0^2P)^2(1-2^{-27})^2 \geq \frac{Pr_0^2}{4}(3-r_0^2P)^2(1-2^{-26}),$$

and by Lemmas 4.2 and 5.1,

$$\begin{aligned}
1 - Pr_1^2 &\leq 1 - \frac{Pr_0^2}{4}(3-r_0^2P)^2(1-2^{-26}) \\
&= \left(1 - \frac{Pr_0^2}{4}(3-r_0^2P)^2\right) + 2^{-26}\frac{Pr_0^2}{4}(3-r_0^2P)^2 \\
&\leq 2^{-10} + 2^{-26} \\
&= 2^{-10}(1+2^{-16}).\square
\end{aligned}$$

5.3 Second Iteration

The second Newton-Raphson iteration uses the same sequence of operations and roundings as the first, producing an approximation r_2:

$$\boxed{\begin{array}{l}
\vdots \\
s_1 \leftarrow away(r_1P_h, 32) \\
t_1 \leftarrow away(\frac{1}{2}r_1, 32) \\
u_1 \leftarrow away(s_1r_1, 32) \\
v_1 \leftarrow trunc(3-u_1, 32) \\
r_2 \leftarrow trunc(v_1t_1, 32) \\
\vdots
\end{array}}$$

Lemma 5.3 $|1 - Pr_2^2| \leq 2^{-20}(1+2^{-5})$.

Proof: The proof is nearly identical to that of Lemma 5.2: with r_1, s_1, t_1, u_1, v_1, and r_2 substituted for r_0, s_0, t_0, u_0, v_0, and r_1, respectively, the same argument yields

$$1 - Pr_2^2 \geq -2^{-29}$$

and

$$Pr_2^2 \geq \frac{Pr_1^2}{4}(3-r_1^2P)^2(1-2^{-26}).$$

Applying Lemmas 5.1 and 5.2, we have

$$\begin{aligned}
1 - Pr_2^2 &\le 1 - \frac{Pr_1^2}{4}(3 - r_1^2 P)^2(1 - 2^{-26}) \\
&= \left(1 - \frac{Pr_1^2}{4}(3 - r_1^2 P)^2\right) + 2^{-26}\frac{Pr_1^2}{4}(3 - r_1^2 P)^2 \\
&\le 2^{-20}(1 + 2^{-16})^2 + 2^{-26} \\
&\le 2^{-20}(1 + 2^{-14} + 2^{-6}) \\
&\le 2^{-20}(1 + 2^{-5}).\Box
\end{aligned}$$

5.4 Third Iteration

The final Newton-Raphson iteration requires higher precision than the first two, using 64-bit rounding and replacing P_h with P, to produce an approximation accurate to 39 bits:

$$\boxed{\begin{array}{l}
\vdots \\
s_2 \leftarrow away(r_2 P, 64) \\
t_2 \leftarrow away(\frac{1}{2} r_2, 64) \\
u_2 \leftarrow away(s_2 r_2, 64) \\
v_2 \leftarrow trunc(3 - u_2, 64) \\
r_3 \leftarrow trunc(v_2 t_2, 64) \\
\vdots
\end{array}}$$

In addition to its high precision, r_3 is guaranteed to be an underestimate of $1/\sqrt{P}$:

Lemma 5.4 $0 \le 1 - Pr_3^2 < 2^{-39}$.

Proof: The proof is similar to that of Lemma 5.2. First, we have

$$r_2 P \le s_2 \le r_2 P(1 + 2^{-63})$$

and

$$r_2^2 P \le s_2 r_2 \le u_2 \le s_2 r_2 (1 + 2^{-63}) \le r_2^2 P(1 + 2^{-63})^2 \le r_2^2 P(1 + 2^{-61}).$$

By Lemma 5.3, we may also conclude that $u_2 < 3$. Thus,

$$v_2 \le 3 - u_2 \le 3 - r_2^2 P$$

and

$$\begin{aligned}
v_2 &\ge (3 - u_2)(1 - 2^{-63}) \ge (3 - r_2^2 P(1 + 2^{-61}))(1 - 2^{-63}) \\
&\ge (3 - r_2^2 P)(1 - 2^{-61})(1 - 2^{-63}) \ge (3 - r_2^2 P)(1 - 2^{-60}).
\end{aligned}$$

Since $r_2/2$ is 32-exact, $t_2 = r_2/2$.Thus,

$$r_3 \leq v_2 t_2 \leq \frac{r_2}{2}(3 - r_2^2 P)$$

and hence, by Lemmas 5.1 and 5.3,

$$1 - Pr_3^2 \geq 1 - \frac{Pr_2^2}{4}(3 - r_2^2 P) \geq 0.$$

Similarly,

$$r_3 \geq v_2 t_2 (1 - 2^{-63}) \geq \frac{r_2}{2}(3 - r_2^2 P)(1 - 2^{-60})(1 - 2^{-63}) \geq \frac{r_3}{2}(3 - r_3^2 P)(1 - 2^{-59}),$$

and hence

$$Pr_3^2 \geq \frac{Pr_2^2}{4}(3 - r_2^2 P)^2 (1 - 2^{-58}).$$

Finally, by Lemmas 5.1 and 5.3,

$$\begin{aligned} 1 - Pr_3^2 &\leq 1 - \frac{Pr_2^2}{4}(3 - r_2^2 P)^2 (1 - 2^{-58}) \\ &= 1 - \frac{Pr_2^2}{4}(3 - r_2^2 P)^2 + 2^{-58} \frac{Pr_2^2}{4}(3 - r_2^2 P)^2 \\ &\leq 2^{-40}(1 + 2^{-5})^2 + 2^{-58} \\ &< 2^{-39}. \Box \end{aligned}$$

6 Exactness Test

The Newton-Raphson result provides a first approximation to $\sqrt{P}$, namely $r_3 P$. This product is rounded to 64 and 32 bits to produce q and q_0. We shall show that if $P \neq q_0^2$, then P does not have a floating point square root (Lemma 6.5), and that in this case, q is an underestimate, accurate to 38 bits (Corollary 6.6).

$$\boxed{\begin{array}{l} \vdots \\ q \leftarrow trunc(r_3 P, 64) \\ q_0 \leftarrow away(r_3 P, 32) \\ P_0 \leftarrow sticky(q_0^2, 64) \\ rem_0 \leftarrow sticky(P - P_0, 64) \\ \text{if } rem_0 = 0 \\ \quad \text{then } sqrt \leftarrow rnd(q_0, mode, prec) \\ \text{else} \\ \vdots \end{array}}$$

Lemma 6.1 $0 \leq P - q^2 < 2^{-38} P$.

Proof: By Lemmas 2.21 and 5.4,

$$q^2 \leq (r_3P)^2 = P(Pr_3^2) \leq P$$

and

$$q^2 \geq (r_3P)^2(1-2^{-63})^2 = P(Pr_3^2)(1-2^{-63})^2 \geq P(1-2^{-39})(1-2^{-63})^2 > P(1-2^{-38}).\Box$$

Lemma 6.2
(a) $|q - q_0| \leq 2^{-30}r_3P$; *(b)* $(q - q_0)^2 \leq 2^{-60}P$.

Proof:
(a) $|q - q_0| \leq |q - r_3P| + |q_0 - r_3P| \leq r_3P(2^{-31} + 2^{-63}) \leq 2^{-30}r_3P$.
(b) $|q - q_0|^2 \leq 2^{-60}(r_3^2P)P \leq 2^{-60}P$. $\Box$

Lemma 6.3 $rem_0 = P - q_0^2$.

Proof: Lemmas 2.32, 2.14, and 2.23 guarantee that q_0 is 32-exact, q_0^2 is 64-exact, and $P_0 = q_0^2$, respectively. By Corollary 2.17, in order to show that $P - P_0$ is 64-exact, it suffices to show $|P - q_0^2| \leq min(P, q_0^2)$. But

$$\begin{aligned}
|P - q_0^2| &\leq (P - q^2) + (q_0 - q)(q_0 + q) \\
&\leq 2^{-38}P + (2^{-30}r_3P)(r_3P + r_3P(1 + 2^{-31})) \\
&\leq 2^{-38}P + (2^{-30}r_3P)(3r_3P) \\
&\leq 2^{-38}P + 2^{-28}P \\
&\leq 2^{-27}P \\
&< P
\end{aligned}$$

and

$$q_0^2 \geq q^2 \geq (1 - 2^{-38})P > 2^{-27}P \geq |P - q_0^2|.$$

The lemma now follows from Lemma 2.23. $\Box$

Thus, Theorem 2 holds in the special case $rem_0 = 0$:

Lemma 6.4 *Let $\ell, h \in \mathbf{Q}$ such that $\ell \geq 0$, $h \geq 0$, and $\ell^2 \leq P \leq h^2$. If $rem_0 = 0$, then*

$$rnd(\ell, mode, prec) \leq sqrt \leq rnd(h, mode, prec).$$

Proof: In this case, we have $sqrt = rnd(q_0, mode, prec)$. By Lemma 6.3, $P = q_0^2$, hence $\ell \leq q_0 \leq h$. The result now follows from the monotonicity of *mode* (Lemmas 2.24, 2.34, and 2.38). $\Box$

Lemma 6.5 *If $rem_0 \neq 0$, then $P = x^2$ has no floating point solution.*

Proof: We shall derive a contradiction from the assumption $P = a^2$, where a is a floating point number. By Lemma 2.15, a is 32-exact. Since $a^2 \geq r_3^2 P^2$, $a \geq r_3 P$, and hence $a \geq q_0$ by Lemma 2.32. It follows that $a > q_0$, and Lemma 2.18 yields $a \geq q_0 + 2^{expo(q_0)-31}$. Thus,

$$a - q \geq 2^{expo(q_0)-31} > 2^{-32} q_0 > 2^{-32} q.$$

But by Lemma 6.1,

$$a - q = \frac{P - q^2}{a + q} < \frac{P - q^2}{2q} \leq \frac{2^{-38} q^2}{(1 - 2^{-38}) 2q} < 2^{-38} q. \Box$$

Henceforth, we shall assume $rem_0 \neq 0$. Note that in view of Lemma 6.5, Lemma 6.1 can now be written as a strict inequality:

Corollary 6.6 $0 < P - q^2 < 2^{-38} P$.

7 Refinement

In this section, we shall compute a correction term res to be added to q, producing an estimate that is accurate to 74 bits (Lemma 7.7). We also show (Corollary 7.10) that $rnd(q + res, mode, prec)$ is an underestimate of the desired result $rnd(\sqrt{P}, mode, prec)$.

We begin with the following observation:

Lemma 7.1 $P(1 - 2^{-75}) < \left[q + (P - q^2)\frac{r_3}{2}\right]^2 < P$.

Proof: We first note that

$$\begin{aligned} P - \left[q + (P - q^2)\frac{r_3}{2}\right]^2 &= P - q^2 - (P - q^2) q r_3 - (P - q^2)^2 \frac{r_3^2}{4} \\ &= \frac{1}{4}(P - q^2)[4 - 4 q r_3 - (P - q^2) r_3^2] \\ &= \frac{1}{4}(P - q^2)[4 - 4 q r_3 + q^2 r_3^2 - P r_3^2] \\ &= \frac{1}{4}(P - q^2)[(2 - q r_3)^2 - P r_3^2]. \end{aligned}$$

By Lemma 5.4 and Corollary 6.6,

$$1 \geq P r_3^2 > q^2 r_3^2 > (1 - 2^{-38}) P r_3^2 \geq (1 - 2^{-38})(1 - 2^{-39}) > (1 - 2^{-38})^2,$$

and hence $1 > q r_3 > 1 - 2^{-38}$.
But then by Lemma 5.4,

$$0 < (2 - q r_3)^2 - P r_3^2 < (1 + 2^{-38})^2 - (1 - 2^{-39}) < 1 + 2^{-36} - 1 + 2^{-39} < 2^{-35}.$$

Finally, by Corollary 6.6,

$$0 < P - \left[q + (P - q^2)\frac{r_3}{2}\right]^2 < \frac{1}{4}2^{-38}P2^{-35} = 2^{-75}P.\Box$$

Thus, we would like to compute res simply by rounding $(P-q^2)r_3/2$. However, since our floating point operations are limited to 64-bit operands, we cannot compute $P - q^2$ directly. Our strategy is to derive a close approximation by using the identity

$$P - q^2 = (P - q_0^2) - 2q_0(q - q_0) - (q - q_0)^2.$$

$$\begin{array}{|l|}\hline \vdots \\ diff \leftarrow near(q - q_0, 64) \\ q_2 \leftarrow near(2q_0, 64) \\ prod \leftarrow away(q_2 diff, 64) \\ sq \leftarrow away(diff^2, 64) \\ t_3 \leftarrow near(\frac{1}{2}r_3, 64) \\ rem_1 \leftarrow sticky(rem_0 - prod, 64) \\ rem_2 \leftarrow trunc(rem_1 - sq, 64) \\ res \leftarrow odd(rem_2 t_3, 64) \\ \vdots \\ \hline \end{array}$$

Lemma 7.2 *$q - q_0$ is 32-exact.*

Proof: First suppose q is 32-exact. Then by Lemma 6.2,

$$|q - q_0| \leq 2^{-30}r_3P < (1 - 2^{-63})r_3P \leq q \leq q_0$$

and the lemma follows from Corollary 2.17. Therefore, we may assume that q is not 32-exact. Let $a = away(q, 32)$. We shall show that $a = q_0$.

Since a is 64-exact and $a > q$, $a > r_3P$ by Lemma 2.23. Similarly, $a \geq q_0$ by Lemma 2.32. But since q_0 is 32-exact and $q_0 \geq q$, Lemma 2.32 also implies $q_0 \geq a$. Thus, $q_0 = a = away(q, 32)$.

But now, by Lemma 2.31, $expo(q-q_0) \leq expo(q)-32$, hence Lemma 2.16 applies with $n = 64$ and $k = 32$. $\Box$

Lemma 7.3 *No rounding is required in the calculation of $diff$, q_2, $prod$, sq, t_3, or rem_1, i.e.,*

(a) $diff = q - q_0$;	*(d)* $sq = diff^2 = (q - q_0)^2$;
(b) $q_2 = 2q_0$;	*(e)* $t_3 = \frac{1}{2}r_3$;
(c) $prod = q_2 diff = 2q_0(q - q_0)$;	*(f)* $rem_1 = (P - q_0^2) - 2q_0(q - q_0)$.

Proof: (a), (b), (c), (d),and (e) are trivial consequences of Lemmas 7.2, 2.14, 2.32, and 2.23.

To prove (f), we shall show that ζ is 59-exact, where

$$\zeta = rem_0 - prod = (P - q_0^2) - 2q_0(q - q_0) = (P - q^2) + (q - q_0)^2.$$

Since q_0 is 32-exact, Lemma 2.14 implies q_0^2 is 64-exact, i.e., $q_0^2 2^{63-expo(q_0^2)} \in \mathbf{Z}$. But since $q_0^2 \geq q^2 > P/2$, we have

$$expo(q_0^2) \geq expo(q^2) \geq expo(P) - 1,$$

hence $q_0^2 2^{64-expo(P)} \in \mathbf{Z}$. Since P is 64-exact, this implies

$$(P - q_0^2)2^{64-expo(P)} = P2^{64-expo(P)} - q_0^2 2^{64-expo(P)} \in \mathbf{Z}.$$

By Lemma 2.11, $2expo(q) \geq expo(q^2) - 1 \geq expo(P) - 2$. Therefore, since q is 64-exact, q_0 is 32-exact, and $expo(q) \leq expo(q_0)$,

$$2q_0(q - q_0)2^{95-expo(P)} = q_0 2^{31-expo(q)}(q - q_0)2^{63-expo(q)}2^{2expo(q)-expo(P)+2} \in \mathbf{Z}.$$

Combining these results, we have

$$\zeta 2^{95-expo(P)} = (P - q_0^2)2^{95-expo(P)} - 2q_0(q - q_0)2^{95-expo(P)} \in \mathbf{Z}.$$

But since $\zeta = (P - q^2) + (q - q_0)^2$, Corollary 6.6 and Lemma 6.2 imply

$$\zeta < 2^{-38}P + 2^{-60}P < 2^{-37}P,$$

hence $expo(\zeta) \leq expo(P) - 37$. Thus,

$$\zeta 2^{58-expo(\zeta)} = \zeta 2^{95-expo(P)}2^{expo(P)-37-expo(\zeta)} \in \mathbf{Z}. \Box$$

Corollary 7.4 $rem_2 = trunc(P - q^2, 64)$.

Proof: $rem_1 - sq = (P - q_0^2) - 2q_0(q - q_0) - (q - q_0)^2 = P - q^2$. $\Box$

Lemma 7.5 $(q + rem_2 \frac{r_3}{2})^2 < P$.

Proof: This follows from Lemma 7.1 and Corollary 7.4. $\Box$

Lemma 7.6 $(rem_2 \frac{r_3}{2})^2 < 2^{-77}q^2$.

Proof: By Lemma 2.21, Corollary 7.4, Lemma 5.4, and Corollary 6.6,

$$(rem_2 \frac{r_3}{2})^2 \leq (P - q^2)^2 \frac{r_3^2}{4} < 2^{-76}P^2 \frac{r_3^2}{4} = 2^{-78}P(Pr_3^2) \leq 2^{-78}P < 2^{-77}q^2. \Box$$

Lemma 7.7 $(q + res)^2 \geq (1 - 2^{-74})P$.

Proof: By Lemma 2.41, Lemma 2.21, and Corollary 6.6,

$$\begin{aligned} res &\geq trunc(rem_2\frac{r_3}{2}, 64) \geq rem_2\frac{r_3}{2}(1 - 2^{-63}) \\ &\geq (P - q^2)\frac{r_3}{2}(1 - 2^{-63})^2 \geq (P - q^2)\frac{r_3}{2}(1 - 2^{-62}) \\ &\geq (P - q^2)\frac{r_3}{2} - 2^{-38}P\frac{r_3}{2}2^{-62} = (P - q^2)\frac{r_3}{2} - 2^{-101}Pr_3, \end{aligned}$$

hence

$$(q + res)^2 \geq \left[q + (P - q^2)\frac{r_3}{2} - 2^{-101}Pr_3\right]^2.$$

We shall apply Lemma 2.7, substituting $q + (P - q^2)\frac{r_3}{2}$ for a, $2^{-101}Pr_3$ for b, and -75 for n. Under these substitutions, we have

$$(1 - 2^n)P < a^2 = \left[q + (P - q^2)\frac{r_3}{2}\right]^2 < P$$

by Lemma 7.1, and

$$b^2 = 2^{-202}(Pr_3^2)P \leq 2^{-202}P < 2^{-152}P = 2^{2n-2}P$$

by Lemma 5.4. Thus,

$$(q + res)^2 \geq (a - b)^2 \geq (1 - 2^{n+1})P = (1 - 2^{-74})P.\Box$$

Lemma 7.8 $q + res = odd(q + rem_2\frac{r_3}{2}, m)$ *for some* $m \geq 102$.

Proof: By Lemmas 2.11 and 7.6,

$$expo(rem_2\frac{r_3}{2}) \leq \frac{1}{2}expo(2^{-77}q^2) \leq \frac{1}{2}(2expo(q) + 1 - 77) = expo(q) - 38.$$

In particular, since q is 64-exact, q is also n-exact, where

$$n = 64 - 1 + expo(q) - expo(rem_2\frac{r_3}{2}) > 64.$$

Hence, by Lemma 2.46,

$$\begin{aligned} q + res &= q + odd(rem_2\frac{r_3}{2}, 64) \\ &= odd(q + rem_2\frac{r_3}{2}, 64 + expo(q + rem_2\frac{r_3}{2}) - expo(rem_2\ fracr_3 2)). \end{aligned}$$

where

$$\begin{aligned} 64 + expo(q + rem_2\frac{r_3}{2}) - expo(rem_2\frac{r_3}{2}) &\geq 64 + expo(q) - expo(rem_2\frac{r_3}{2}) \\ &\geq 64 + 38 = 102.\Box \end{aligned}$$

Corollary 7.9 *$q + res$ is not 101-exact.*

Proof: This follows from Lemmas 7.8 and 2.44. $\Box$

Corollary 7.10 *Let $h \in \mathbf{Q}$. If $h \geq 0$ and $h^2 \geq P$, then $rnd(q + res, mode, prec) \leq rnd(h, mode, prec)$.*

Proof: By Lemma 7.5, $h > q + rem_2 \frac{r_3}{2}$. Since $102 - 2 > 64 \geq prec$, the corollary follows from Lemma 2.47. $\Box$

Lemma 7.11 *Let $\ell, h \in \mathbf{Q}$. If $\ell \geq 0$, $h \geq 0$, and $\ell^2 \leq P \leq h^2$, then*

$$\ell - 2^{expo(q+res)-72} < q + res < h + 2^{expo(q+res)-101}.$$

Proof: The first inequality is a consequence of Lemma 7.7: since

$$\ell^2 \leq P \leq (q + res)^2/(1 - 2^{-74}) < (q + res)^2/(1 - 2^{-74})^2,$$

$$\ell < (q + res)/(1 - 2^{-74}) \leq (q + res)(1 + 2^{-73}) < (q + res) + 2^{expo(q+res)-72}.$$

The second inequality is derived from Lemmas 7.8, 2.42, and 7.5: since

$$q + res = odd(q + rem_2 \frac{r_3}{2}, m) \geq trunc(q + rem_2 \frac{r_3}{2}, m),$$

$$expo(q + res) \geq expo(trunc(q + rem_2 \frac{r_3}{2}, m)) = expo(q + rem_2 \frac{r_3}{2}).$$

Thus,

$$q + res \leq q + rem_2 \frac{r_3}{2} + 2^{expo(q+rem_2 \frac{r_3}{2})-101} < h + 2^{expo(q+res)-101}. \Box$$

8 Final Computation

It follows from Lemma 7.7 and Corollary 7.10 that $q + res$ is sufficiently accurate except when it is only slightly less than a rounding boundary. In order to test for this condition, $q + res$ is rounded to 64 bits using the modes *trunc*, *near*, and *away*, yielding $root_0$, and $root_1$, and $root_2$, respectively. Thus, $q + res$ lies in the interval between the successive 64-exact numbers $root_0$ and $root_2$. (Corollary 7.9 guarantees that these two numbers are distinct.) There are two rounding boundaries of concern: the midpoint of the interval and $root_2$.

The approximate location of $q + res$ relative to this interval is given by the variable *bits*, the fractional part of which represents the first 8 bits of $q + res$ that are lost in the truncation to $root_0$ (Lemma 8.7). Thus, there are two values of this bit sequence that are handled specially: 11111111 and 01111111, corresponding to $bits = 2 - 2^{-8}$ and $bits = 1 + 2^{-1} - 2^{-8}$, respectively. We shall show that for all other values, the correct result is given by $rnd(q + res, mode, prec)$. The proof will then be completed by considering each exceptional case separately.

8.1 Test for Boundary Cases

$$
\begin{array}{|l|}
\hline
\vdots \\
root_0 \leftarrow trunc(q + res, 64) \\
root_1 \leftarrow near(q + res, 64) \\
root_2 \leftarrow away(q + res, 64) \\
used \leftarrow trunc(root_0 - q, 64) \\
end \leftarrow trunc(res - used, 64) \\
ulp \leftarrow trunc(root_2 - root_0, 64) \\
aeb \leftarrow trunc(ulp + end, 9) \\
bits \leftarrow sig(aeb) \\
\text{if } bits \notin \{2 - 2^{-8}, 1 + 2^{-1} - 2^{-8}\} \\
\quad \text{then } sqrt \leftarrow rnd(q + res, mode, prec) \\
\vdots \\
\hline
\end{array}
$$

Lemma 8.1 $root_0 = trunc(q + rem_2\frac{r_3}{2}, 64)$.

Proof: This is an immediate consequence of Lemmas 7.8 and 2.45. □

Corollary 8.2 $root_0^2 < P$.

Proof: This follows from Lemmas 8.1 and 7.5. □

Lemma 8.3 $res - used = (q + res) - root_0$.

Proof: We shall show that $used = root_0 - q$ as an application of Corollary 2.17. Thus, we must show that $|root_0 - q| \leq min(root_0, q)$. The proof will be based on the estimate

$$|root_0 - q| \leq |root_0 - (q + rem_2\frac{r_3}{2})| + |rem_2\frac{r_3}{2}|.$$

By Lemma 7.6, $(rem_2\frac{r_3}{2})^2 < 2^{-77}P$, and by Lemmas 2.21 and 7.5,

$$(root_0 - (q + rem_2\frac{r_3}{2}))^2 \leq 2^{-126}(q + rem_2\frac{r_3}{2})^2 < 2^{-126}P.$$

It follows that

$$
\begin{aligned}
(root_0 - q)^2 &\leq (|root_0 - (q + rem_2\frac{r_3}{2})| + |rem_2\frac{r_3}{2}|)^2 \\
&\leq 4max[(root_0 - (q + rem_2\frac{r_3}{2}))^2, (rem_2\frac{r_3}{2})^2] < 2^{-75}P.
\end{aligned}
$$

But by Lemma 6.1, $q^2 > 2^{-1}P$, and by Lemma 2.21

$$root_0^2 \geq (q + rem_2\frac{r_3}{2})^2(1 - 2^{-63})^2 \geq q^2(1 - 2^{-63})^2 > 2^{-2}P. \Box$$

Lemma 8.4 $ulp = root_2 - root_0 = 2^{expo(q+res)-63}$.

Proof: By Lemmas 2.23 and 2.32 and Corollary 7.9, $root_0$ and $root_2$ are 64-exact and $q + res$ is not. Thus, $root_0 < q + res < root_2$. Let $b = root_0 + 2^{expo(q+res)-63}$. By Lemma 2.18, b is 64-exact and $root_2 \geq b$. But by Lemma 2.23, $b > q + res$, and by Lemma 2.32, $b \geq root_2$. Thus, $b = root_2$. $\Box$

Lemma 8.5 *$root_0 + end = trunc(q + res, m)$ for some $m \geq 128$.*

Proof: By Lemma 2.25, $expo(q + res - root_0) \leq expo(q + res) - 64$, and by Lemmas 8.3 and 2.27,

$$\begin{aligned} root_0 + end &= root_0 + trunc(q + res - root_0, 64) \\ &= trunc(q + res, 64 + expo(q + res) - expo(q + res - root_0)).\Box \end{aligned}$$

Corollary 8.6 *$0 < end < ulp$.*

Proof: By Lemmas 7.9 and 8.3, $end > 0$; by Lemmas 7.9, 8.4, and 8.5,

$$root_0 + end \leq q + res < root_2 = root_0 + ulp.\Box$$

Lemma 8.7 *$trunc(q + res, 72) = root_0 + (bits - 1)ulp$.*

Proof: We shall invoke Lemma 2.29 with $x = q + res$, $n = 64$, and $k = 8$. In this case, we have $trunc(x, n) = root_0$ and $e = expo(q + res) - 63$. By Lemma 8.3,

$$z = trunc(q + res - root_0, 64) = trunc(res - used, 64) = end,$$

and hence

$$trunc(2^e + z, k + 1) = trunc(end + ulp, 9) = aeb.$$

Thus, Lemma 2.29 yields

$$trunc(q + res, 72) - root_0 = (sig(aeb) - 1)2^e = (bits - 1)ulp.\Box$$

Corollary 8.8 *Let $\ell, h \in \mathbf{Q}$. If $\ell \geq 0$, $h \geq 0$, and $\ell^2 \leq P \leq h^2$, then*

$$\ell - 2^{expo(q+res)-70} < root_0 + (bits - 1)ulp < h + 2^{expo(q+res)-101}.$$

Proof: Let $e = expo(q + res)$. By Lemma 2.21,

$$q + res - 2^{e-71} < root_0 + (bits - 1)ulp \leq q + res.$$

Thus, by Corollary 7.11,

$$\ell - 2^{e-70} < \ell - 2^{e-72} - 2^{e-71} < q + res - 2^{e-71} < root_0 + (bits - 1)ulp$$

and

$$h + 2^{e-101} > q + res \geq trunc(q + res, 72) = root_0 + (bits - 1)ulp.\Box$$

Corollary 8.9

(a) If $bits < 1 + 2^{-1}$*, then* $q + res < root_0 + ulp/2$, $root_1 = root_0$*, and* $end < ulp/2$*;*

(b) If $bits \geq 1 + 2^{-1}$*, then* $q + res > root_0 + ulp/2$ *and* $root_1 = root_2$*;*

(c) If $bits > 1 + 2^{-1}$*, then* $end > ulp/2$*.*

Proof: Let $m = (root_0 + root_2)/2 = root_0 + ulp/2$. Note that m is 65-exact.

(a) If $bits < 1 + 2^{-1}$, then $trunc(q + res, 72) < m$, hence $q + res < m$ by Lemma 2.23, and Lemma 2.36 implies $root_1 = root_0$. By Lemma 8.5, $root_0 + end < m$.

(b) In this case, $q+res \geq trunc(q+res, 72) \geq m$. By Corollary 7.9, $q+res > m$, and $root_1 = root_2$ by Lemma 2.36.

(c) Since $bits$ is 9-exact and $expo(bits) = 0$, $bits > 1 + 2^{-1}$ implies $bits \geq 1 + 2^{-1} + 2^{-8}$. Thus,

$$q + res \geq root_0 + (bits - 1)ulp \geq root_0 + (2^{-1} + 2^{-8})ulp = root_0 + ulp/2 + 2^{e-71}.$$

But now by Lemmas 8.5 and 2.21,

$$root_0 + end > q + res - 2^{e-127} \geq root_0 + ulp/2 + 2^{e-71} - 2^{e-127} > root_0 + ulp/2. \Box$$

Lemma 8.10 *Let* $\ell, h \in \mathbf{Q}$ *such that* $\ell \geq 0$, $h \geq 0$*, and* $\ell^2 \leq P \leq h^2$*. If* $bits \notin \{2 - 2^{-8}, 1 + 2^{-1} - 2^{-8}\}$*, then*

$$rnd(\ell, mode, prec) \leq sqrt \leq rnd(h, mode, prec).$$

Proof: In this case, $sqrt = rnd(q + res, mode, prec)$. By Corollary 7.10,

$$rnd(q + res, mode, prec) \leq rnd(h, mode, prec).$$

Thus, we need only show

$$rnd(\ell, mode, prec) \leq rnd(q + res, mode, prec).$$

Note that $1 \leq bits < 2$ and $bits$ is 9-exact. Since $bits \neq 2 - 2^{-8}$, we must have $bits \leq 2 - 2^{-7}$: if not, then according to Lemma 2.18, $bits > 2 - 2^{-7}$ would imply

$$bits \geq 2 - 2^{-7} + 2^{-8} = 2 - 2^{-8},$$

hence $bits > 2 - 2^{-8}$, which in turn would imply $bits \geq 2$.

Let $e = expo(q + res) = expo(root_0)$. Then $ulp = 2^{e-63}$ and by Corollary 8.8,

$$\ell < root_0 + (1 - 2^{-7})2^{e-63} + 2^{e-70} = root_0 + ulp = root_2.$$

For the case $mode = inf$, the desired inequality follows from Lemmas 2.35 and 2.34:

$$\begin{aligned} rnd(\ell, mode, prec) &= away(\ell, prec) \leq away(root_2, prec) \\ &= away(away(q + res, 64), prec) = away(q + res, prec) \\ &= rnd(q + res, mode, prec). \end{aligned}$$

For the case $mode \in \{trunc, minf\}$, note that since

$$trunc(\ell, 64) \leq \ell < root_2 = root_0 + ulp,$$

$trunc(\ell, 64) \leq root_0$ by Lemma 2.18. Thus, by Lemmas 2.23 and 2.26,

$$\begin{aligned} rnd(\ell, mode, prec) &= trunc(\ell, prec) = trunc(trunc(\ell, 64), prec) \\ &\leq trunc(root_0, prec) = trunc(trunc(q + res, 64), prec) \\ &= trunc(q + res, prec) = rnd(q + res, mode, prec). \end{aligned}$$

Finally, consider the case $mode = near$. Let $m = root_0 + ulp/2$. Then m is 65-exact by Lemma 2.18 and since $m < root_2 \leq 2^{e+1}$ by Corollary 2.33, $expo(m) = e$.

Suppose $bits \geq 1 + 2^{-1}$. Then $m < q + res$ by Corollary 8.9. Now since $\ell < root_2 = m + 2^{e-64}$, Corollary 2.40 implies

$$rnd(\ell, mode, prec) = near(\ell, prec) \leq near(q+res, prec) = rnd(q+res, mode, prec).$$

We may assume, then, that $bits < 1 + 2^{-1}$. By hypothesis, $bits \neq 1 + 2^{-1} - 2^{-8}$. If $bits > 1 + 2^{-1} - 2^{-7}$, then by Lemma 2.18, we would have

$$bits \geq 1 + 2^{-1} - 2^{-7} + 2^{-8} = 1 + 2^{-1} - 2^{-8},$$

hence $bits > 1 + 2^{-1} - 2^{-8}$, which in turn would imply $bits \geq 1 + 2^{-1}$. Thus, we may further assume that $bits \leq 1 + 2^{-1} - 2^{-7}$. By Corollary 8.8,

$$\ell < root_0 + (2^{-1} - 2^{-7})2^{e-63} + 2^{e-70} = root_0 + 2^{-64}.$$

Since $root_0 < q + res$, the result again follows from Corollary 2.40. □

8.2 Boundary Cases

The objective of this section is to complete the proof of the inequalities stated in Theorem 2. The cases $mode \neq near$ and $mode = near$ are handled separately in the following two lemmas. Note that in view of Lemma 8.10, we may assume that $bits$ is one of the two boundary values, $2 - 2^{-8}$ and $1 + 2^{-1} - 2^{-8}$.

```
⋮
else if mode ≠ near then
   det ← away(root₂root₂, 64)
   if P ≥ det
      then sqrt ← rnd(root₁ + end, mode, prec)
      else sqrt ← rnd(root₀ + end, mode, prec)
⋮
```

Lemma 8.11 *Let $\ell, h \in \mathbf{Q}$ such that $\ell \geq 0$, $h \geq 0$, and $\ell^2 \leq P \leq h^2$. If $mode \neq near$, then*

$$rnd(\ell, mode, prec) \leq sqrt \leq rnd(h, mode, prec).$$

Proof: We consider two cases, corresponding to the branches of the conditional statement above.

Case 1: $P \geq det$

Let $e = expo(q + res)$. Since $bits < 2$, Lemma 8.8 implies

$$\ell < root_0 + ulp + 2^{e-70} < root_0 + 2ulp = root_2 + ulp.$$

Since $P \geq root_2^2$, $P > root_2^2$ by Lemma 6.5, and hence $h > root_2$.

Substituting $root_2$ for ℓ in Lemma 8.8, we have

$$root_2 - 2^{e-70} < root_0 + (bits - 1)ulp,$$

hence

$$bits > 1 + (root_2 - root_0 - 2^{e-70})/ulp = 1 + (ulp - 2^{e-70})/ulp = 2 - 2^{-7} > 1 + 2^{-1}.$$

By Lemma 8.9, $root_1 = root_2$, and hence $sqrt = rnd(root_2 + end, mode, prec)$. Note that by Corollary 8.6, $root_2 < root_2 + end < root_2 + ulp$.

Let $b = root_2 + 2^{expo(root_2)-63}$. Then $b \geq root_2 + ulp$ and by Lemma 2.18, $root_2$ and b are successive 64-exact numbers. By Lemmas 2.18 and 2.32,

$$away(root_2 + end, 64) = away(root_2 + ulp, 64) = b \leq away(h, 64),$$

and by Lemma 2.35, it follows that

$$away(root_2 + end, prec) = away(root_2 + ulp, prec) \leq away(h, prec).$$

By Lemmas 2.18 and 2.23,

$$trunc(root_2 + end, 64) = root_2 \geq trunc(\ell, 64),$$

and by Lemma 2.26, it follows that

$$trunc(root_2 + end, prec) = trunc(root_2, prec) \geq trunc(\ell, prec).$$

If $mode = inf$, then

$$sqrt = away(root_2 + end, prec) = away(root_2 + ulp, prec)$$

and since $\ell < root_2 + ulp$,

$$away(\ell, prec) \leq away(root_2 + ulp, prec) \leq away(h, prec).$$

If $mode = minf$ or $mode = trunc$, then

$$sqrt = trunc(root_2 + end, prec) = trunc(root_2, prec)$$

and since $root_2 < h$,

$$trunc(\ell, prec) \leq trunc(root_2, prec) \leq trunc(h, prec).$$

Case 2: $P < det$

Since P is 64-exact, $P < away(root_2^2, 64)$ implies $P < root_2^2$ by Lemma 2.32, hence $\ell < root_2$. By Lemma 8.2, $root_0 < h$. In this case, $sqrt = rnd(root_0 + end, mode, prec)$, and by Corollary 8.6,

$$root_0 < root_0 + end < root_0 + ulp = root_2.$$

If $mode = inf$, then by Lemmas 2.32 and 2.35,

$$\begin{aligned} sqrt &= away(root_0 + end, prec) = away(away(root_0 + end, 64), prec) \\ &= away(root_2, prec) \end{aligned}$$

and

$$\begin{aligned} away(\ell, prec) &= away(away(\ell, 64), prec) \leq away(root_2, prec) \\ &\leq away(away(h, 64), prec) = away(h, prec). \end{aligned}$$

If $mode = minf$ or $mode = trunc$, then by Lemmas 2.23 and 2.26,

$$\begin{aligned} sqrt &= trunc(root_0 + end, prec) = trunc(trunc(root_0 + end, 64), prec) \\ &= trunc(root_0, prec) \end{aligned}$$

and

$$\begin{aligned} trunc(\ell, prec) &= trunc(trunc(\ell, 64), prec) \leq trunc(root_0, prec) \\ &\leq trunc(h, prec). \square \end{aligned}$$

$\vdots$

else *(* mode = near *)*
  $det \leftarrow trunc(root_1 root_2, 64)$
  if $P \leq det$ then
    $sqrt \leftarrow near(root_0 + end, prec)$
  else if $bits = 2 - 2^{-8}$ then
    $ends \leftarrow trunc(\frac{1}{16}end, 64)$
    $sqrt \leftarrow near(root_2 + ends, prec)$
  else $sqrt \leftarrow near(root_2 - end, prec)$

Lemma 8.12 *Let $\ell, h \in \mathbf{Q}$ such that $\ell \geq 0$, $h \geq 0$, and $\ell^2 \leq P \leq h^2$. If $mode = near$, then*

$$rnd(\ell, mode, prec) \leq sqrt \leq rnd(h, mode, prec).$$

Proof: Let $e = expo(q + res) = expo(root_0)$,

$$m = root_0 + ulp/2 = (root_0 + root_2)/2,$$

and

$$m' = root_2 + ulp/2.$$

We shall proceed by a case analysis based on the structure of the algorithm. In each case to be considered, the computed value of $sqrt$ is $near(\xi, prec)$ for some ξ. We shall see that this value always lies within one of the following three intervals: $(root_0, m)$, $(m, root_2)$, and $(root_2, m')$. Note that if (α, β) is any one of these intervals, then α is 65-exact and $\alpha < \beta \leq \alpha + 2^{expo(\alpha)-64}$. Thus, according to Corollary 2.40, in order to prove

$$near(\ell, prec) \leq near(\xi, prec) \leq near(h, prec),$$

it suffices to show (a) $\alpha < \xi < \beta$, (b) $\ell < \beta$, and (c) $h > \alpha$.

Case 1: $P > det$, $bits = 2 - 2^{-8}$

(a) $\xi = root_2 + ends$, where $0 < ends < ulp/16$, hence $root_2 < \xi < m'$.

(b) By Corollary 8.8, $\ell < root_0 + (bits - 1)ulp + 2^{e-70} < m'$.

(c) By Lemma 8.9, $root_1 = root_2$, and so $P > trunc(root_2^2, 64)$. Since P is 64-exact, $P > root_2^2$ by Lemma 2.23, hence $h > root_2$.

Case 2: $P > det$, $bits = 1 + 2^{-1} - 2^{-8}$

(a) $\xi = root_2 - end$. Since $0 < end < ulp$, $m < \xi < root_2$.

(b) By Corollary 8.8, $\ell < root_0 + (bits - 1)ulp + 2^{e-70} < root_2$.

(c) By Lemma 8.9, $root_1 = root_0$, hence $P > trunc(root_0 root_2, 64)$ and by Lemma 2.23, $P > root_0 root_2$. But then by Lemma 2.18, since $root_0 root_2$ and P are both 128-exact,

$$P \geq root_0 root_2 + 2^{expo(root_0 root_2)-127} \geq root_0 root_2 + 2^{2e-127}.$$

But

$$root_0 root_2 = (m - 2^{e-64})(m + 2^{e-64}) = m^2 - 2^{2e-128}$$

and hence

$$P \geq m^2 - 2^{2e-128} + 2^{2e-127} > m^2.$$

Thus, $h > m$.

Case 3: $P \leq det$, $bits = 2 - 2^{-8}$

(a) $\xi = root_0 + end$. By Corollary 8.9, $m < \xi < root_2$.

(b) Since $P \leq root_1 root_2 \leq root_2^2$, $P < root_2^2$, hence $\ell < root_2$.

(c) By Corollary 8.8, $h > root_0 + (bits - 1)ulp - 2^{e-101} > root_0 + ulp/2 = m$.

Case 4: $P \leq det$, $bits = 1 + 2^{-1} - 2^{-8}$

(a) Again, we have $\xi = root_0 + end$. But in this case, by Corollary 8.9, $root_0 < \xi < m$.

(b) By Corollary 8.9, $root_1 = root_0$. As we observed under Case 2,

$$root_0 root_2 = m^2 - 2^{2e-128} < m^2.$$

But in this case, $P \leq root_0 root_2$, hence $\ell < m$.

(c) By Corollary 8.2, $h > root_0$. □

9 Exponent Bounds

It remains to show that the prescribed exponent bound is uniformly satisfied. Two technical lemmas will be required:

Lemma 9.1 *Let $x, x', y \in \mathbf{R}$ with $|1 - xy^2| \leq 1/2$ and $expo(x') = expo(x)$. Then*
(a) $|expo(y)| < |expo(x)|/2 + 2$;
(b) $|expo(x'y)| < |expo(x)|/2 + 3$.

Proof: Since $1/2 \leq xy^2 \leq 3/2$,

$$-1 = expo(1/2) \leq expo(xy^2) \leq expo(3/2) = 0.$$

Thus, by Lemma 2.11,

$$\begin{aligned} &|2expo(y) + expo(x)| \\ &\leq \; |2expo(y) - expo(y^2)| + |expo(y^2) + expo(x) - expo(xy^2)| + |expo(xy^2)| \\ &\leq \; 1 + 1 + 1 = 3, \end{aligned}$$

hence $2|expo(y)| \leq |expo(x)| + 3$, which implies (a).

For the proof of (b), we have

$$\begin{aligned}
|2expo(x'y)| &\leq 2|expo(x'y) - (expo(x') + expo(y))| \\
&\quad + |2expo(y) + expo(x')| + |expo(x')| \\
&= 2|expo(x'y) - (expo(x') + expo(y))| \\
&\quad + |2expo(y) + expo(x)| + |expo(x)| \\
&\leq 2 + 3 + |expo(x)|. \square
\end{aligned}$$

Lemma 9.2 *Let $x, y \in \mathbf{R}$ and $n \in \mathbf{Z}^+$, $n > 1$. If x and y are n-exact, then*

$$|expo(x - y)| \leq max(|expo(x)|, |expo(y)|) + n - 1$$

Proof: First suppose that $xy < 0$. Then we may assume $x > 0$ and $y < 0$, hence

$$0 < x < x - y < 2max(x, -y)$$

and

$$expo(x) \leq expo(x - y) \leq expo(2max(x, -y)) \leq max(|expo(x)|, |expo(y)|) + 1.$$

Thus, we may assume that $xy > 0$ and, without loss of generality, that $y > x > 0$. But then by Corollary 2.19,

$$expo(y) \geq expo(y - x) \geq expo(x) + 1 - n. \square$$

Theorem 1 will follow from our final lemma:

Lemma 9.3 *If P is a non-negative generalized $(64, 15)$-floating point number, mode is an IEEE rounding mode, $prec \in \mathbf{Z}^+$, and $prec \leq 64$, then $FSQRT$ executes successfully on $\langle P, mode, prec \rangle$.*

Proof: It is evident by inspection that the value of every numerical expression occurring in the program is 64-exact. Thus, we need only show that no value is generated with an exponent outside of the interval $[1 - 2^{-16}, 2^{16}]$. We begin with the observation that $|expo(P)| \leq 2^{15}$, which is clearly implied by the hypothesis, and proceed by considering each assignment statement in turn. The proof is a series of applications of Lemmas 9.1 and 9.2.

We begin with the bindings generated during the first Newton-Raphson iteration. Since $expo(P_h) = expo(P)$, $|expo(P_h)| \leq 2^{15}$. Furthermore, by Lemma 4.2,

$$|expo(r_0)| < |expo(P)|/2 + 2 \leq 2^{14} + 2$$

and

$$|expo(s_0)| \leq |expo(r_0 P_h)| + 1 < |expo(P)|/2 + 3 + 1 \leq 2^{14} + 4.$$

This in turn implies

$$|expo(t_0)| \leq |expo(r_0)| + 1 + 1 \leq 2^{14} + 4$$

and

$$|expo(u_0)| \leq |expo(s_0 r_0)| + 1 \leq |expo(s_0)| + |expo(r_0)| + 2 \leq 2^{15} + 8.$$

Thus,

$$|expo(v_0)| \leq |expo(3 - u_0)| + 1 \leq max(|expo(u_0)|, 1) + 31 + 1 \leq 2^{15} + 40.$$

Finally, by Lemma 5.2,

$$|expo(r_1)| < |expo(P)|/2 + 2 \leq 2^{14} + 2.$$

The variables involved in the second and third iterations are handled similarly. Next, by Lemma 5.4, we have

$$|expo(q)| = |expo(r_3 P)| < |expo(P)|/2 + 3 \leq 2^{14} + 3$$

and

$$|expo(q_0)| \leq |expo(r_3 P)| + 1 < 2^{14} + 4.$$

Thus,

$$|expo(P_0)| = |expo(q_0^2)| \leq 2|expo(q_0)| + 1 \leq 2^{15} + 9$$

and

$$|expo(rem_0)| \leq |expo(P - P_0)| + 1 \leq max(|expo(P)|, |expo(P_0)|) + 63 + 1 \leq 2^{15} + 73.$$

We may assume that execution continues with the refinement phase. Thus,

$$|expo(diff)| = |expo(q - q_0)| \leq max(|expo(q)|, |expo(q_0)|) + 63 \leq 2^{14} + 67,$$

$$|expo(q_2)| \leq |expo(q_0)| + 1 \leq 2^{14} + 5,$$

$$|expo(prod)| \leq |expo(q_2 diff)| + 1 \leq |expo(q_2)| + |expo(diff)| + 2 \leq 2^{15} + 74,$$

$$|expo(sq)| \leq |expo(diff^2)| + 1 \leq 2|expo(diff)| + 2 \leq 2^{15} + 136,$$

$$|expo(t_3)| \leq |expo(r_3)| + 1 + 1 \leq 2^{14} + 4,$$

$$\begin{aligned} |expo(rem_1)| = |expo(rem_0 - prod)| &\leq max(|expo(rem_0)|, |expo(prod)|) + 63 \\ &\leq 2^{15} + 137, \end{aligned}$$

$$|expo(rem_2)| = |expo(rem_1 - sq)| \leq max(|expo(rem_1)|, |expo(sq)|) + 63 \leq 2^{15} + 200,$$

and

$$|expo(res)| = |expo(rem_2 t_3)| \leq |expo(rem_2)| + |expo(t_3)| + 1 \leq 2^{15} + 2^{14} + 205.$$

Since $q < q + res < 2q$, we have

$$|expo(q + res)| \leq |expo(q)| + 1 \leq 2^{14} + 4.$$

Thus, for $i = 0, 1$, and 2, $|expo(root_i)| \leq 2^{14} + 5$. Continuing, we have

$$|expo(used)| = |expo(root_0 - q)| \leq max(|expo(root_0)|, |expo(q)|) + 63 \leq 2^{14} + 68,$$

$$\begin{aligned} |expo(end)| &= |expo(res - used)| \leq max(|expo(res)|, |expo(used)|) + 63 \\ &\leq 2^{15} + 2^{14} + 268, \end{aligned}$$

$$|expo(aeb)| = |expo(ulp)| \leq |expo(root_0)| + 63 \leq 2^{14} + 68,$$

and

$$|expo(bits)| = 0.$$

In each of the boundary cases, we have for some $mode$ and some i and j,

$$\begin{aligned} |expo(det)| &= |expo(rnd(root_i root_j mode, 64))| \\ &\leq |expo(root_i)| + |expo(root_j)| + 2 \leq 2^{15} + 12. \end{aligned}$$

In the boundary case for $mode = near$, we must also observe that

$$|expo(ends)| = |expo(\frac{1}{16} end)| \leq |expo(end)| + 4 \leq 2^{15} + 2^{14} + 272.$$

Finally, since $q^2 < P < (2q)^2$, we have

$$rnd(q, mode, prec) \leq sqrt \leq rnd(2q, mode, prec),$$

and hence

$$\begin{aligned} |expo(sqrt)| &\leq max(|expo(rnd(q, mode, prec))|, |expo(rnd(2q, mode, prec))|) \\ &\leq |expo(q)| + 2 \leq 2^{14} + 5. \Box \end{aligned}$$

10 Modification of the Algorithm

It was mentioned in our introduction that a minor modification of the original K5 square root algorithm was required to produce the final version presented here. We conclude with a discussion of this modification and AMD's decision to adopt it.

The modification was a change in the rounding mode used in the computation of the variable res: originally, *trunc* was used instead of *odd*. The purpose of this variable was to produce an approximation $q + res$ that represented an improvement over q while ensuring the inequality $q + res < \sqrt{P}$, which was exploited in the subsequent analysis. In our attempt to formalize this analysis, we found that it depended on the assumption that $q + res$ was not 64-exact, which we were unable to justify. A violation of this assumption, however improbable, would have the disastrous consequences $root_0 = root_1 = root_2$ and $ulp = 0$, a possibility that had not been considered by the designers of the algorithm.

Although this assumption has been recognized by the designers as an oversight, it remains unclear that it is incorrect, since we have been unable to exhibit a counterexample. To see why this may be difficult, note that q and res are both computed with 64-bit precision, and by Lemma 7.6, the exponents of these numbers differ by 38 or more. Thus, in order for $q + res$ to be 64-exact, the least significant 38 bits of res must all be 0. Since there is no apparent reason for this to be any more or less probable than the occurrence of any other sequence of 38 bits, it is reasonable to suppose that this will occur with probability 2^{-38}. Following this line, we would expect the algorithm to fail at most once in 2^{38} executions, assuming randomly distributed inputs. It is not surprising, therefore, that failure was not observed in simulation. Moreover, since there is no obvious correlation between this 38-bit sequence and the properties of the input P, it should be no more surprising that we have been unable to engineer a counterexample.

The decision to modify the implementation was not clear at first, but became easier when Tom Lynch proposed an elegant solution. Although the solution introduced a new rounding mode, *odd*, which increased the complexity of proof, it did not affect the cycle count of the instruction.

With this modification, it was clear that $q + res$ could not be 64-exact (Corollary 7.9), and according to Lynch's intuition, the critical properties of the approximation were preserved even though the inequality $q + res < \sqrt{P}$ was no longer true. Indeed, the weaker inequality given by Corollary 7.10 was found to be sufficient. Thus, the final result was both provably correct and efficiently implemented.

References

[1] Boyer, R.S., and Moore, J, *A Computational Logic Handbook*, Academic Press, Boston, MA, 1988.

[2] Bryant, R.E., "Verification of Arithmetic Functions with Binary Moment Diagrams", Technical Report CMU-CS-94-160, School of Computer Science, Carnegie-Mellon University, 1994.

[3] Clarke, E.M. and Zhao, X., "Word Level Symbolic Model Checking: A New Approach for Verifying Arithmetic Circuits", Technical Report CMU-CS-95-161, School of Computer Science, Carnegie-Mellon University, 1995.

[4] Harrison, J., "Floating Point Verification in HOL", in *Proc. 8th Intl. Workshop on Higher Order Logic Theorem Proving and its Applications*, Springer-Verlag, 1995.

[5] Intel Corporation, *Pentium Family User's Manual, Vol. 3: Architecture and Programming Manual*, 1994.

[6] Institute of Electrical and Electronic Engineers, "IEEE Standard for Binary Floating Point Arithmetic", Std. 754-1985, New York, NY, 1985.

[7] Kaufmann, M., and Moore, J, "A Precise Description of the ACL2 Logic", `http://www.cs.utexas.edu/users/moore/acl2/reports/km97a.ps`

[8] Leeser, M., and O'Leary, J., "Verification of a Subtractive Radix-2 Square Root Algorithm and Implementation", in *Proc. Intl. Conf. on Computer Design*, 1995.

[9] Miner, P., and Leathrum, J., "Verification of IEEE Compliant Subtractive Division Algorithms", in M. Srivas and A. Camilleri (eds.) *Proceedings of Formal Methods in Computer-Aided Design (FMCAD) '96)*, Springer-Verlag LNCS 1166, pp. 275–293, 1996.

[10] Moore, J, Lynch, T., and Kaufmann, M., "A Mechanically Checked Proof of the Correctness of the Kernel of the $AMD5_K86$ Floating-Point Division Algorithm", `http://devil.ece.utexas.edu/~lynch/divide/divide.html`

[11] Russinoff, D.M., "A Mechanically Checked Proof of IEEE Compliance of the AMD K7 Floating Point Multiplication, Division, and Square Root Instructions", `http://www.onr.com/user/russ/david/k7-div-sqrt.ps`.

[12] Rueß, H., Srivas, M.K., and Shankar, N., "Modular Verification of SRT Division", in R. Alur and T. Henzinger (eds.) *Computer-Aided Verification (CAV '96)*, Springer Verlag LNCS 1102, pp. 123–134, July 1996.

[13] Steele, G.L., Jr., *Common Lisp The Language*, 2nd edition, Digital Press, 1990.