Statistical Analysis of Floating Point Flaw in the Pentium™ Processor (1994)

Intel Corporation

November 30th 1994

H. P. Sharangpani Numerics Architect & Project Leader, Microcode & Algorithm Development, Pentium™ Processor Development Team, Intel Corporation

M. L. Barton, Ph.D. Staff Computational Scientist, Software Technology Lab, Intel Corporation

1 Abstract

A subtle flaw in the hardware divide unit of the Pentium™ Processor was discovered by Intel. Subsequently, a characterization of its impact to the end-user application base was conducted. The flaw is rare and data-dependent, and causes a reduction in precision of the divide instruction and certain other operations in certain cases.

The significance of the flaw depends upon (a) the rate of use of specific FP instructions in the Pentium™ CPU, (b) the data fed to them, (c) the way in which the results of these instructions are propagated into further computation in the application; and (d) the way in which the final results of the application are interpreted.

The thorough and detailed characterization of the flaw and the subsequent investigations of its impact on applications through elaborate surveys, analyses and empirical observation lead us to the overall conclusion that the flaw is of no concern to the vast majority of users of Pentium processor based systems. A few users of applications in the scientific/engineering and financial engineering fields who require unusual precision and invoke millions of divides per day may need to employ either an updated Pentium processor without the flaw or a software workaround.

2 Introduction

This document describes a subtle flaw inside the Floating Point Unit on certain steppings of the Pentium™ processor, and characterizes the significance of this flaw for end-users of applications that are likely to be run on a computer based on the Pentium processor. The flaw in the Pentium processor causes a slight reduction in precision for certain operations in very rare cases.

Intel discovered this flaw as part of its ongoing product development and testing work. So far seven trillion computer cycles worth of testing has been conducted on the Pentium processor as part of this ongoing program. Subsequent to the discovery of the flaw, a detailed investigation was conducted to characterize both the flaw as well as its impact on the end-user base. This document summarizes the outcome of the investigation.

The overall document has 9 sections. This section 2 is an introduction. Section 3 gives a high level summary of the flaw as it manifests itself inside the processor. Section 4 explains in detail how the divide algorithm is implemented and what led to the slight flaw in early steppings. Section 5 describes the general statistical evaluation framework suitable to analyze the frequency of occurrence and the significance of the flaw on the end-user application base. Section 6 analyzes the impact of the flaw on this software base. Section 7 presents the conclusions of the characterization study. Sections 8 and 9 contain Acknowledgments and Bibliography of References.

3 Description of the Flaw

The flaw as it manifests itself in the CPU is now described. The following characterization statements can be made:

1. On certain input data, the Floating Point Divide Instructions on the Pentium ™ processor produce inaccurate results.
2. The problem can occur in all three operating precisions (single, double, extended) for the divide instruction. Empirical studies involving over 1 trillion data cases indicate that far fewer failures are found in single precision than in double or extended precision. The remainder function, and those transcendental functions which rely on the divide instruction, also exhibit reduced precision. For the remainder and transcendental instructions, which operate only in extended precision, the problem can only occur in extended precision.
3. The incidence of the problem is independent of the processor rounding modes.
4. Encountering the problem is highly dependent upon the input data. Only certain input data will trigger the problem. It is not straightforward to describe the exact set of input operands on which the problem can get triggered. Hence it is necessary to describe the incidence of occurrence in terms of a probability distribution statistic. Characterization based on two independent methods consistently yields a probability that 1 in 9 billion randomly fed divide or remainder instructions will produce inaccurate results. The fraction of the total input number space that is prone to failure is 1.14×10^{-10}.

   [The first characterization method is analytical and mathematical, and is based upon a Markov chain analysis of the iterative implementation algorithm. The second method is empirical, relying upon running billions of random input samples through the instructions under test. Over 1 trillion data points were run for the second method. Both methods correlate well.]
5. The degree of the inaccuracy of the result delivered depends upon the input data and upon the instruction involved.

   On the divide instruction, the worst case inaccuracy occurs in the 12th bit position to the right of the binary point of the significand of the result, or in the 4th significant decimal digit. Statistical measurements using over a trillion test points indicate that the inaccuracy is equally likely to manifest itself in bit positions 12 through 52 to the right of the binary point. The likelihood of encountering an inaccuracy in any one bit position is then 1 in every 360 billion randomly fed divides.
6. The problem does not occur on the specific use of the divide instruction to compute the reciprocal of the input operand in single precision.

The cause of the problem traces itself to a few missing entries in a lookup table used in the hardware implementation algorithm for the divide operation. Since this divide operation is used by the Divide, Remaindering, and certain Transcendental Instructions, an inaccuracy introduced in the operation manifests itself as an inaccuracy in the results generated by these instructions.

4 Pentium™ Processor Divide Algorithm

The Pentium processor is Intel's next generation of compatible microprocessors following the i486™ CPU family. The primary goal was to combine compiler and hardware technology to maximize performance while preserving software compatibility. More specifically the floating point performance goal was to achieve up to a 5x speedup of floating point vector code and a 3x speedup of scalar code when compared to an i486 CPU of identical clock frequency[1]. In part, this meant providing a higher performance floating point divider.

The i486 processor used the classic non-restoring "shift and subtract" division algorithm for its floating point divide operation. This inherently allowed only one quotient bit to be generated per clock. To improve the floating point divide performance on the Pentium processor, a radix 4 SRT[1] algorithm was chosen. This algorithm, as implemented, allows the divide hardware to generate 2 bits of quotient per clock thus approximately doubling the divide performance.

4.1 SRT Algorithm

To help better understand the flaw, we have attached a brief introduction to the SRT division technique.

The SRT divide algorithm can basically be described in steps as:

1. Sample the most significant digits of the divisor and dividend.
2. Use the samples as indexes into a lookup table to get a guess of the next quotient digits. (in this case the quotient guess can be -2, -1, 0, +1, +2).
3. Multiply the divisor by the quotient guess and subtract it from dividend (Note that this is an addition if the quotient guess was negative).
4. Save the quotient digits in the least significant digits of a quotient register.
5. Shift the remainder left by 2 and shift the quotient registers left by 2 (i.e. radix 4)
6. Sample the most significant digits of the new shifted partial remainder.
7. Go to step 2 unless you have generated enough significant quotient digits.
8. Generate the binary quotient by assembling the values in the quotient register.
9. If the last partial remainder was negative then adjust the quotient by subtracting the value 1.

For a complete treatment on the subject of SRT division refer to the paper written by Daniel E. Atkins[2].

1. The SRT algorithm was named after the 3 scientists that discovered it independently at about the same time: D. Sweeney of IBM, J.E. Robertson of the University of Illinois, and T.D. Tocher of the Imperial College of London.

Mathematically, the SRT algorithm can be represented with the equations shown in (EQ 1) and (EQ 2) below. The first equation shows the first step of the algorithm where P_0, the initial partial remainder, is the dividend. The second equation indicates the recursive nature of the algorithm. Here, to generate the next partial remainder ($\mathbf{P_{j+1}}$) one must multiply the quotient guess ($\mathbf{q_{j+1}}$) by the divisor (**d**) and subtract that quantity from the shifted partial remainder of the previous iteration ($\mathbf{rP_j}$). Where (**r**) is the radix of the operation (in this case 4). The number of iterations is determined by the number of significant digits required in the quotient remembering that the first iteration only produces one significant digit.

$$\mathbf{P_1 = P_0 - q_1 * d} \qquad \textbf{(EQ 1)}$$

$$\mathbf{P_{j+1} = rP_j - q_{j+1} * d \ (j=[1,IC-1])} \qquad \textbf{(EQ 2)}$$

Pj = partial remainder in the j-th cycle (P_0 = dividend).
r = radix (in this case r = 4).
q_j = quotient digit selected in the (j+1)th cycle.
d = divisor.
IC= The number of iterations to perform.

Note that when the radix is equal to the operative base, this recursive equation yields one digit of quotient per iteration. For higher radix division a multi-digit quotient can be generated each iteration. In the case of the Pentium processor, radix 4 division is performed on binary numbers yielding 2 binary quotient digits per iteration. This relationship holds true if the following restriction (for convergence) is placed on the remainder:

$$\mathbf{|P_{j+1}| <= n/(r-1) * d} \qquad \textbf{(EQ 3)}$$

n = absolute value of the highest divisor multiple (i.e. for divisor multiples,-2x,-1x,0x,1x, 2x, ... n =2)

The quantity **n/(r-1)** is referred to as the measure of redundancy (MoR) and in the Pentium processor divide implementation evaluates to 2/(4-1)=2/3. This value will be used from here on.

4.1.1 Quotient Selection

Since a non-restoring division is used and the result is stored in “redundant” form, the guessed quotient need only meet the above remainder criteria. By manipulating the above equations, MIN and MAX equations that bound the next partial remainder for a given divisor and quotient digit are produced.

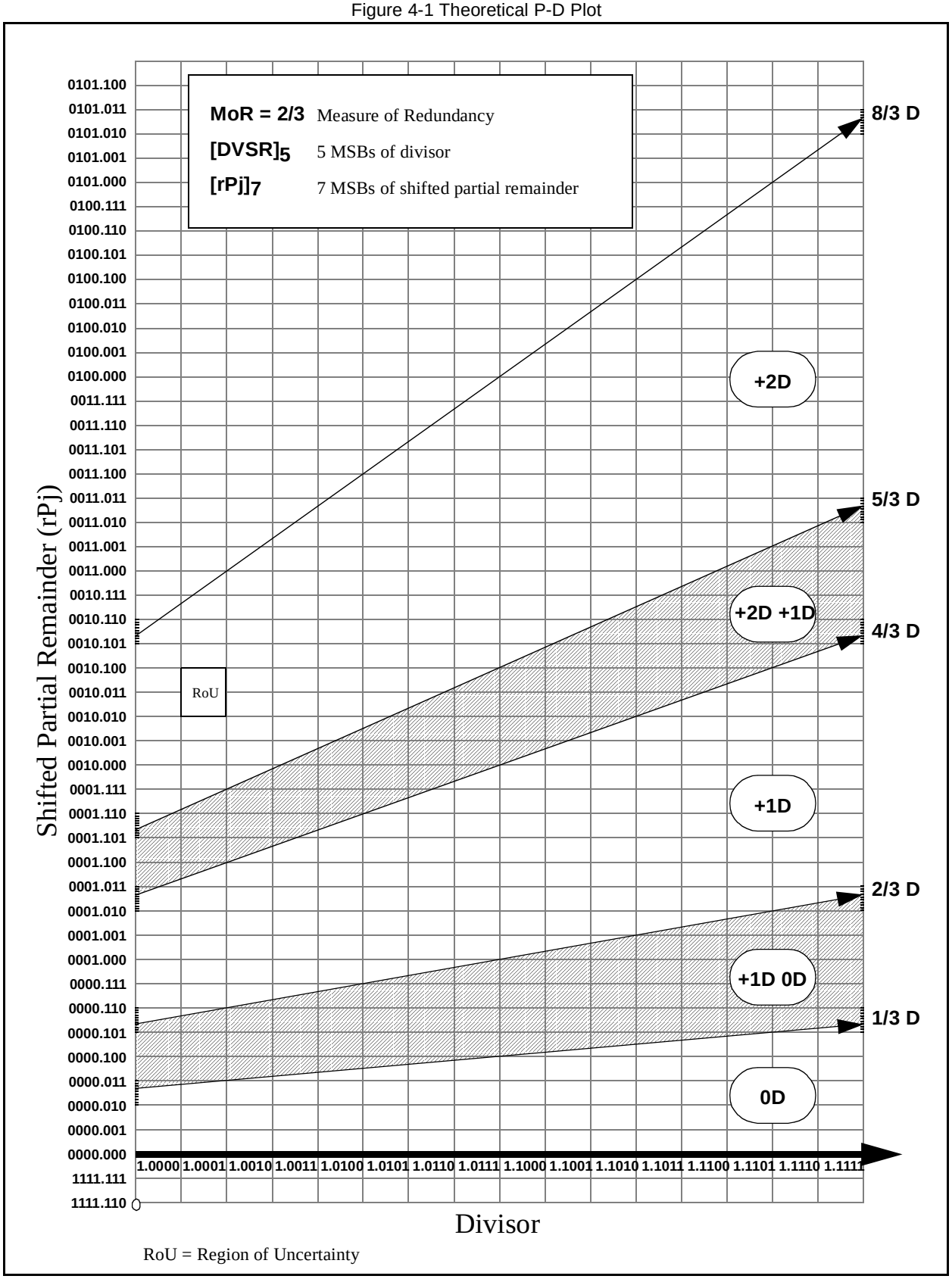
$$\mathbf{r*P_j = [(+2/3) + q_{j+1}]*d \ (max)} \qquad \textbf{(EQ 4)}$$

$$\mathbf{r*P_j = [(-2/3) + q_{j+1}]*d \ (min)} \qquad \textbf{(EQ 5)}$$

$$\mathbf{with;\ q(j+1) = [-2,-1,0,+1,+2]} \qquad \textbf{(EQ 6)}$$

Plotting these equations generates a plot of the Partial-Remainder and Divisor (commonly known as the “P-D” plot), that is used in the quotient selection process. The positive half of this plot is shown below in Figure 4-1 . The shaded regions in this plot indicate the region of overlapping quotient choices, where either of the digits indicated may be chosen.

Figure 4-1 Theoretical P-D Plot



RoU = Region of Uncertainty

4.1.2 Region of Uncertainty

Since the divisor and the shifted partial remainder are truncated there is some uncertainty in the generation of the quotient bits. One must take this uncertainty into account when selecting quotient digits, that is, a quotient digit can only be selected if the region of uncertainty for the given Partial-remainder/Divisor pair lies entirely within a quotient region delineated by the equations above. This is where the overlapping nature of these regions comes in handy.

For the uncertainty in the Divisor which is chopped after the 4th bit to the right of the binary point (i.e. maintaining 4 bits to the right of the binary point).

$$|D-D'| < 2^{-4} \quad \textbf{(EQ 7)}$$

D = divisor **(EQ 8)**

D' = chopped divisor **(EQ 9)**

For the Partial Remainder, the estimate comes from the 7 most significant bits of the redundant form (carry-save) "shifted partial remainder" (4 bits to the left and 3 bits to the right of the binary point) thus:

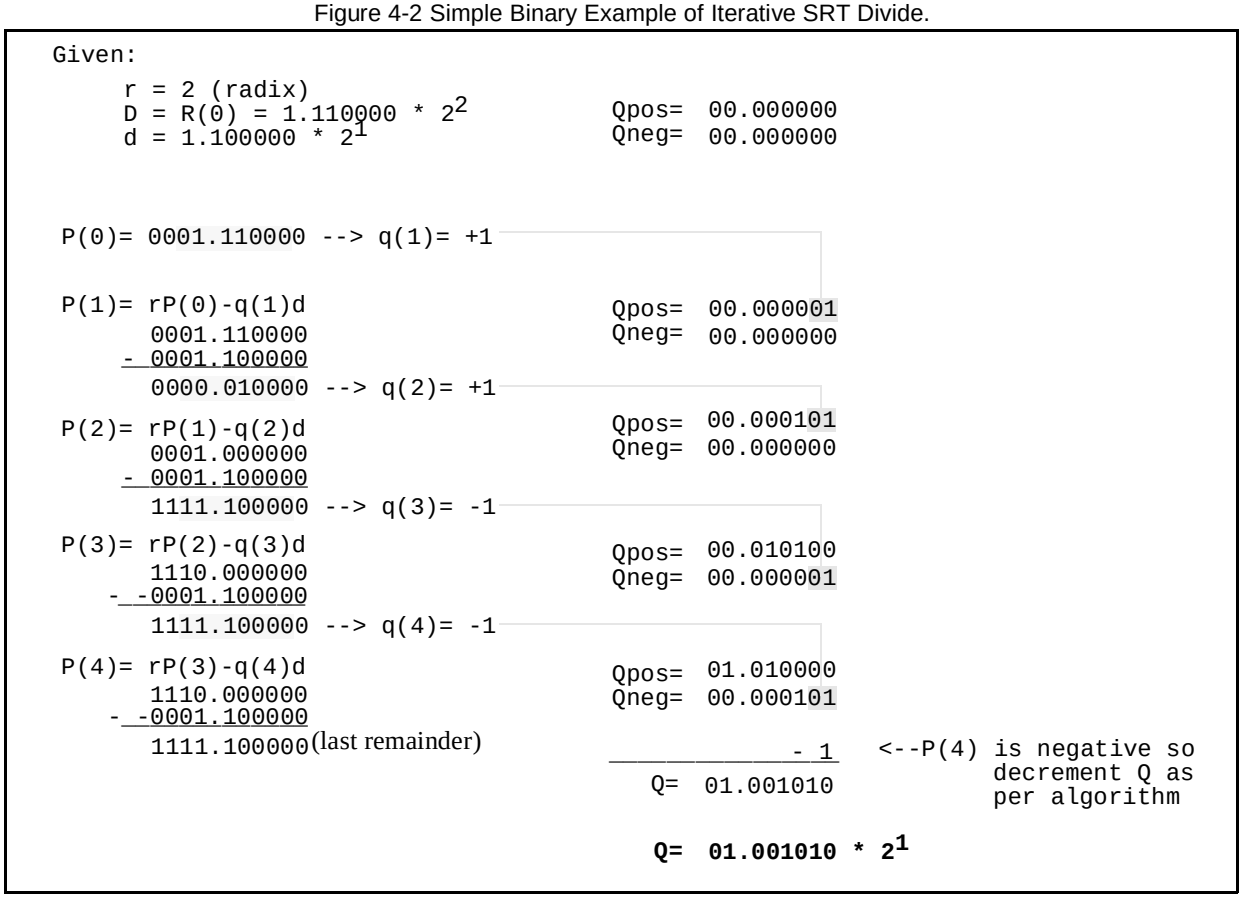
$$|P_j-P'_j| < 2^{-2} \quad \textbf{(Conservative worst case due to the Carry-Save implementation)} \quad \textbf{(EQ 10)}$$

P_j = full non-redundant partial remainder **(EQ 11)**

P'_j = chopped non-redundant partial remainder **(EQ 12)**

Figure 4-2 below illustrates a simple binary example of the same iterative equation for radix 4 SRT. Note the way that Qpos and Qneg are used to store positive and negative weighted quotients, how the final quotient is derived from the two, and which bits are sampled as indexes into the lookup table (the shaded regions in the partial remainders).

Figure 4-2 Simple Binary Example of Iterative SRT Divide.



4.2 The Underlying Cause

After the quantized P-D plot (lookup table) was numerically generated as in Figure 4-1 , a script was written to download the entries into a hardware PLA (Programmable Lookup Array). An error was made in this script that resulted in a few lookup entries (belonging to the positive plane of the P-D plot) being omitted from the PLA. The 5 critical entries are shown in Figure 4-3 as the shaded regions. As a result of the omission, a divisor/remainder pair that hits these entries during the lookup phase of the SRT algorithm will incorrectly read a quotient digit value of 0 instead of +2. Subsequently, the iterative algorithm will return a quotient result with reduced precision.

As can be seen from the P-D plot, the only situations in which there is a probability of seeing this flaw is when the binary divisor has the following bit patterns in the most significant bits: 1.0001, 1.0100, 1.0111, 1.1010, and 1.1101. Empirically, it has been observed that these bit patterns need to be followed by a long string of 1’s to further boost the probability of incurring the inaccuracy due to the flaw. Note also, that since the divide hardware operates only on the mantissa the exponents of the operands have no effect on whether this flaw is observed or not.

4.3 Instructions Affected

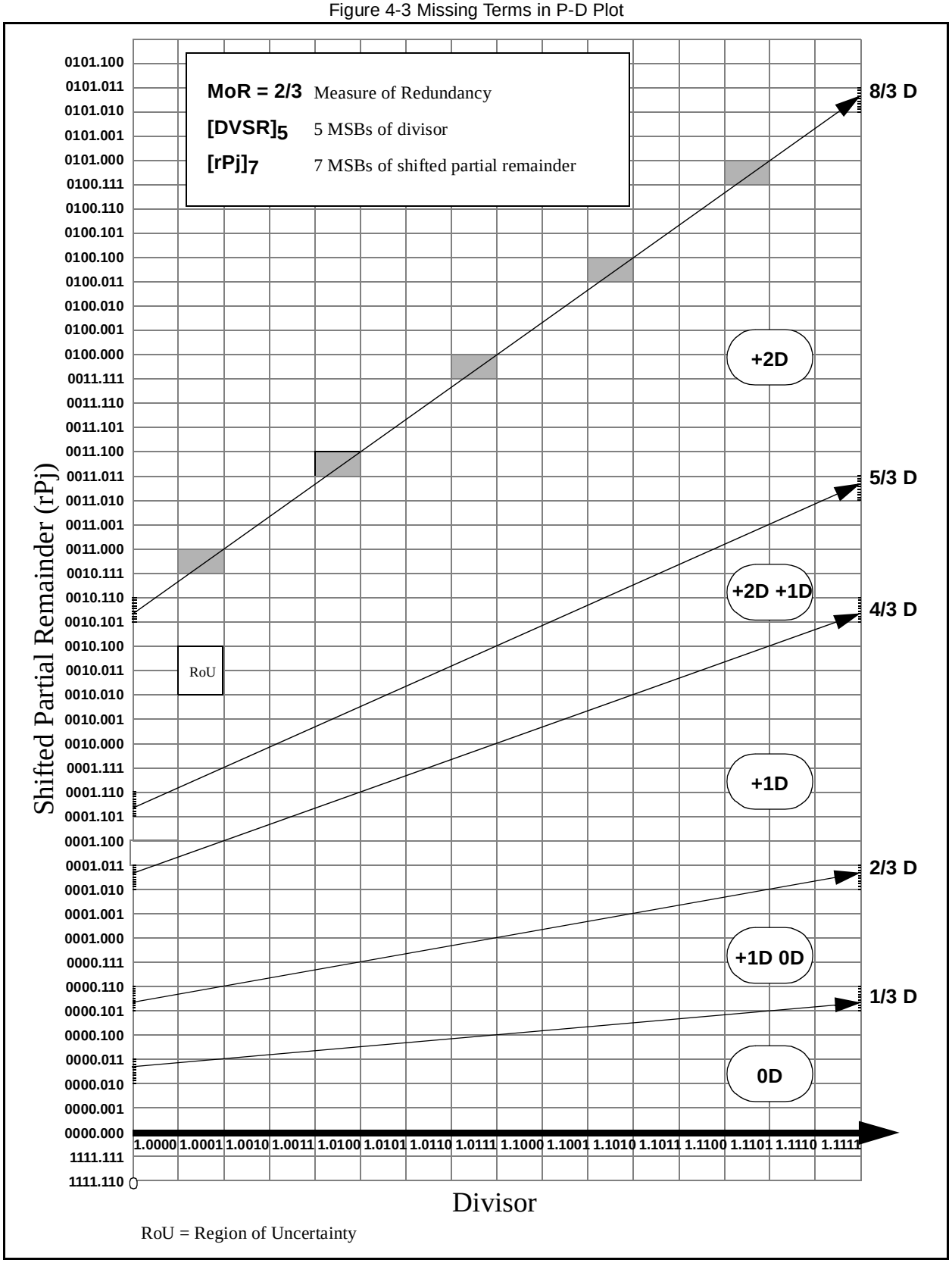
Since it is the divider hardware that exhibits the flaw, instructions that will exhibit the flaw include:

FDIV
FDIVP
FDIVR
FDIVRP
FIDIV
FIDIVR
FPREM
FPREM1

The following transcendental instructions use the divide hardware within their computation but empirical testing of billions of cases have not shown any error:

FPTAN
FPATAN
FYL2X
FYL2XP1

Figure 4-3 Missing Terms in P-D Plot



RoU = Region of Uncertainty

5 Evaluation Framework to Gauge Impact on User

The significance of the flaw to an end-user clearly depends upon:

1. The frequency of occurrence of the reduced precision divide within the application. If the flaw is unlikely to be seen during the practical lifetime of the computer, it is of no significance to the user.
2. The (propagated) impact to the end-user when the problem manifests itself.

The frequency of occurrence of the reduced precision divide depends upon the rate of use of the specific FP instructions in the Pentium CPU by the user, and upon the data fed to these instructions. If and when the problem manifests itself, the impact on the end-user depends upon the way in which the results of these instructions (along with any inaccuracies) are propagated into further computation in the application, and upon the way in which the final results of the application are interpreted by the end-user.

The evaluation methodology thus involved first estimating the frequency of occurrence of the reduced precision divide for random input data, and then analyzing each potential occurrence and its environment to gauge its end-impact. The subsequent sections describe the statistical method followed to characterize the frequency of occurrence, propose a metric for comparison, and present reference information to calibrate the significance of a given rate of occurrence.

5.1 Statistical Characterization Methodology for Frequency of Occurrence

This section describes the general statistical evaluation methodology suitable to analyze the frequency of occurrence of the reduced precision divide in applications. Given that it is intractable to describe the exact set of input operands on which the problem can get triggered, and given that the incidence of the problem is best described as a statistical probability, the method suitable for characterizing the frequency of this problem in an end-user application finds a close parallel to the conventional framework used for evaluating reliability of a computer system given an assortment of hard and soft failure modes.

For any given failure mechanism, conventional reliability methods define the FIT rate (or Failures In Time) in terms of the number of device failures produced by the mechanism in every 10^9 hours of device operation. The Mean Time Before Failure or MTBF is simply the inverse of the FIT rate.

When examining the reliability of the overall computer system, one focuses upon the failure mechanisms with the highest FIT rates, since these will make the dominant contribution to a system failure in the field. For example, system failures can occur due to a wide variety of reasons, such as:

1. Human errors in installation,
2. Power supply failures,
3. Packaging and system interconnect defects,
4. CPU failures,
5. Memory failures,

6. Disk drive failures,
7. Keyboard failures, and
8. Failure mechanisms from other devices.

These failure mechanisms span a wide range of FIT rates, and it is typically the mechanism with the highest FIT rate that is most significant from the point of view of frequency of failures.

5.2 Metric for Evaluating Frequency of Occurrence

A modified form of the conventional definition of the FIT rate has been found to be a convenient metric for evaluating the frequency of occurrence.

Consider the following analysis:

1. Probability of failure for independent divide/remainder operation = $1.14 \times 10^{-10} = P$
2. By Basic Binomial model:

   Given n successive independent operations,

   Probability of n-1 consecutive successful operations = $(1-P)^{\{n-1\}}$

   Probability of the 1st failure at the nth operation = $[(1-P)^{\{n-1\}}] * P$

   MTBF = SUM{n * (probability of first failure at nth op)}

   $= SUM\ \{n * [(1-P)^{n-1}] * P\}$

   $= 1/P$

   $= \sim 9 \times 10^{9}$ = time taken for 9 billion independent divides to be run
3. Hence for each application, assess the time taken to run 9 billion independent divide or remainder operations, since this is the MTBF due to this mechanism.

In effect, the analysis involves calculating the effective FIT rate due to this failure mechanism in the context of the given application. As can be seen from the above analysis, the mean time before an inaccuracy is encountered is simply the time taken for the user to exercise the application with 9 billion independent divide operations. Alternatively, characterizing the various applications in terms of how many independent operations of interest (e.g. divide instructions) are run per unit time (say days) will provide an effective metric for the frequency of occurrence of the reduced precision divide, assuming totally random input data to the instructions.

Based on the FIT rate for this failure mode alone, a calculation is performed on the MTBF due to this failure mode. This MTBF is then compared against the MTBF due to other failure modes, and against the lifetime of the part, to give the user a perspective against which to judge the rarity of the error due to the flaw.

5.3 Reference Failure Rates

Table 5-1 below summarizes a few failure mechanisms and FIT rates typical in a commercial PC system based on the Pentium processor. Also included in the table is a sample FIT rate for a typical PC user running spreadsheet calculations involving 1,000 independent divides per day

on a Pentium processor that exhibits the flaw. As can be seen from this table, the FIT rate due to the flaw bears little significance for such a user because the mean time before encountering an inaccuracy far exceeds both the time before other failure mechanisms begin to play, as well as the practical lifetime of the PC.

Table 5-1 Typical System Failure Rates

Failure category and system component	Hard or Soft	FIT rate (per 10^9 device hours)	MTBF (1 in x years)	Rate of significant failure seen by user
16 4-Mbit DRAM parts in a 60Mhz Pentium ™ processor system without ECC	Soft	16,000	7 years	Depends upon where defect occurs and how propagated
Particle defects in Pentium™ processor	Hard	400-500	200-250 years	Depends upon where defect occurs and how propagated
16 4-Mbit DRAM parts in a 60Mhz Pentium ™ processor system with ECC	Soft	160	700 years	Depends upon where defect occurs and how propagated
PC user on spreadsheet running 1,000 independent divides a day on the Pentium™ processor [a]	Hard	3.3	27,000 years	Less frequent than 1 in 27,000 years. Depends upon the way inaccurate result gets used

a. A detailed analysis on divide usage in spreadsheets is provided in Section 6.2.1 .

6 Analysis of Impact on Applications

As discussed in the previous section, given a certain appreciable frequency of occurrence of the reduced precision divide, the impact on the end-user depends upon the way in which the results of these instructions (along with any inaccuracies) are propagated into further computation in the application, and upon the way in which the final results of the application are interpreted by the end-user.

In order to truly understand the importance of the flaw, an elaborate characterization effort was undertaken. The effort had a twofold thrust: first, to estimate the frequency of occurrence of the reduced precision divide, and second, to estimate how the reduction in precision gets propagated to the end result, and to determine how it gets used.

The methodology used for this purpose involved data sources both internal and external to Intel. Internally, characterization was performed in a verification laboratory on key applications that had been ported to the laboratory environment. Additionally, test suites provided by the application vendor for verification of the functionality of the test suite on that platform were procured, and were used for a pilot measurement. Externally, opinions were taken from eminent application and algorithm experts in the industry as well as from power users of the key applications.

6.1 Taxonomy of Applications

The application base was categorized into the following groups:

1. Commercial PC applications on desktop/mobile platform running on MS-DOS, Microsoft* Windows*, or OS/2*. This class includes basic spreadsheet users for personal finance or basic accounting.
2. Technical applications. This includes a broad range of applications including engineering and scientific, advanced multimedia, educational, and financial applications. Thus, this class includes power users of spreadsheets such as financial analysts and financial engineers.
   Applications in this category could be purely integer-based, or could involve floating point instructions for either numerical computation or for visualization. This class spans the widest range of applications running on MS-DOS, Windows, OS/2 or UNIX* operating systems.
3. Server and transaction processing applications.

6.2 Impact on Commercial PC applications

A large majority of PC applications do not invoke the floating point unit. This includes applications such as word processing, text editing and email. In the commercial PC domain, the majority of applications that do use floating point do not invoke an appreciable number of divides and hence do not introduce significant failures that will pose a data-integrity problem during the useful life of the part. Table 6-1 illustrates the outcome of the analyses and characterization on a few key applications. Of specific concern were the spreadsheet applications, where numerical calculation is often supported via use of the floating-point unit. Towards this concern, a more elaborate study focussed on spreadsheets. This study is addressed in the next subsection.

6.2.1 Spreadsheets

The study on spreadsheets included a survey of acknowledged numerics experts in the industry. The results of the survey were partially confirmed by statistical characterization in the internal verification laboratory at Intel. The results from the survey are now summarized.

Table 6-1 COMMERCIAL PC APPLICATIONS ON DOS/WINDOWS/OS/2

Class	Applications	MTBF	Impact of failure in div/rem/tran
Word processing	Microsoft Word, Wordperfect, etc.	Never	None
Spreadsheets (basic user)	123, Excel, QuattroPro (basic user runs fewer than 1000 div/day)	27,000 years	Unnoticeable
Publishing, Graphics	Print Shop, Adobe Acrobat viewers	270 years	Impact only on Viewing
Personal Money Management	Quicken, Money, Managing Your Money, Simply Money, TurboTax (fewer than 14,000 divides per day)	2,000 years	Unnoticeable
Games	X-Wing, Falcon (flight simulator), Strategy Games	270 years	Impact is benign, (since game)

The most common use of a spreadsheet is as a computational database that collects information of some kind, e.g. information on expense reports, budgets or miscellaneous data on a process, an experiment or personnel in a firm. Only a small fraction of all spreadsheet users are actually “heavy” users, users who intensely invoke the computational engine to generate numerical information. Most other users either use spreadsheets to display this kind of information and make minor modifications and edits, or perform a few calculations.

Once entered into the spreadsheet with a certain number of significant digits, most data is converted to some internal representation, and most numeric computation is floating point-based. Spreadsheets like Excel* and QuattroPro* compute in double precision floating point, while Lotus-123* computes in extended precision. While intermediate values are stored with the full precision, results are displayed as dictated by the user. About 40% of the results in general are displayed with only two decimal digits after the point (e.g. for currency display), another 40% are displayed as integers (after rounding), and only the remaining 20% of the numbers are displayed in scientific format or in floating point format with more than two digits after the decimal point.

About 95% of the numeric formulae invoked contain one or two operators, typically an add or a multiply or, rarely, a divide. Occasionally, the Mod function (that remainders by one to get the fractional portion of the number) is used. The remaining 4% of the formulae used include functions such as IRR (Internal Rate of Return, which solves an Nth order polynomial equation), Power, Interest Rates, Standard Deviation and Square Root. Transcendental functions are invoked very rarely. Equation solvers are also used rarely, and could invoke the divide function to implement Newton’s formula.

For most accounting applications of the spreadsheet, typical input data may have up to about 7-8 decimal digits to the left of the decimal point, and about 2-3 digits to the right of the point, so that the information is known to about 11 significant digits. The most common use of divides is for computation of ratios. Often these ratios are applied once or a couple of times to data, and often towards the end of the computation, so that results from the divide have reduced opportunity to propagate. Since ratios are often used for calculating percentages, the ratio requires about 4 decimal digits (2 to the right and 2 to the left of the decimal point).

For the rest of the basic spreadsheet users, most data that is input to spreadsheets has fewer than three significant digits to the right of the decimal point. A lot of the numbers have only a few significant digits to the left of the point and are thus only known to four or five digits. Also frequent are power-of-two fractions.

In terms of numbers of operations, fewer than 10% of the instructions executed in a typical spreadsheet run floating point instruction. Most of the numerical operations are geared toward the display engine. Displaying a spreadsheet of 1 page with 600 cells and 2 floating point operations (one of which may be a divide) per cell would require 1,200 FP operations. On the computational side, a typical recalculation could contains 5,000 adds and subtracts, a few multiplies and a very few divides. Divides are used for date calculations, to divide by 365. It is very unlikely that a basic spreadsheet user would invoke any more than 500-1,000 independent divides per day. It is worth noting that scrolling through several pages repeatedly would result in recalculation with the same values and would not introduce any additional independent divide operations and therefore no additional errors.

Given that even by conservative estimates, an average PC user invoking 1,000 divides per day would see a FIT rate of once in 27,000 years due to this failure mechanism, and given the information on the way the data is interpreted, displayed and used, we conclude that the rate of a significant failure would be much smaller than once every 27,000 years. By the analysis from the previous sections, the common user will not see this effect during the practical lifetime of the part.

For individual users who invoke a greater number of independent divides per day (than 1,000), the rate of encountering a reduced precision result will simply be increased proportionately.

The treatment of the advanced use of spreadsheets for financial engineering is handled in the section on technical applications.

6.3 Impact on Technical Applications

In the following two subsections, we examine first engineering and scientific applications, followed by applications in the financial world.

6.3.1 Impact on Engineering and Scientific Applications

A broad array of applications are run by scientists and engineers on modern workstations. Table 6-2 shows one taxonomy of technical applications based on the discipline. This table gives

the algorithm employed in the particular application, an example of such an application, an indication of its reliance on divides, the normal condition of the problem (an indication of the likelihood that an error will propagate through the calculation [see below]) and the frequency with which Pentium processors are likely to be used in the application.

The straight-forward calculation of frequency of occurrence of a divide inaccuracy based on the number of divides/day on a Pentium processor based platform indicates that users will experience inaccuracy due to the flaw from time to time in the course of floating point intensive work. Based on this result it is necessary to investigate the likely impact of a divide returning a reduced precision result. Figure 6-1 shows a simple framework for evaluating the frequency of

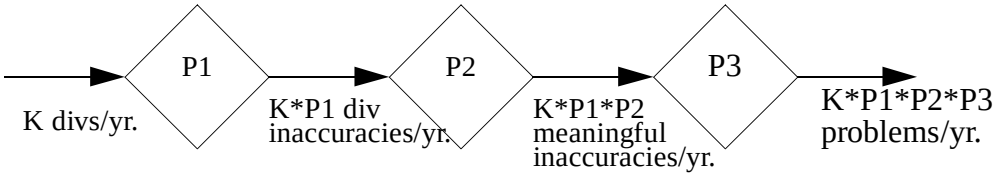


Figure 6-1 Outcome Frequencies Algorithm.

outcomes for a Pentium processor based platform used for divide-intensive work.The symbols

Table 6-2 Taxonomy of Workstation Applications

Discipline	Algorithm(s)	Example(s)	Divide Intensive	Conditioning
Structural Mechanics				
Stress	Sparse Matrix Solution	MSC/NASTRAN™	Yes	Poor (Shells)
Vibration	Sparse Matrix Eigenanalysis	MSC/NASTRAN™	Yes	Poor
Crash	Dynamics	LS-DYNA3D™	No	Good
Fluid Dynamics				
Flow	Sparse Matrix Solution	FIDAP™	Yes	Good
Flow	Finite Difference Update	FLO67™	No	Good
Combustion	Sparse Matrix Solution	FIRE™	Yes	Poor
Chemistry				
Quantum Chemistry	Dense Eigenanalysis	GAMESS™	Yes	Good
Molecular Modelling	N-body Problem	Charmm™	No	Good
Visualization & Graphics				
Imaging	Pixel Manipulation	DISSPLA™	No	Good
Animation and Rendering	Polygon manipulation and shading	SuperAnimator	Yes	Good
Petroleum Engineering				
Reservoir Modelling	Implicit Finite Differences	ECLIPSE™	Yes	Good
Seismic	Signal Processing	DISCO™	No	Good
Electrical Engineering				
Circuits	Sparse Matrix Solution	HSPICE™	Yes	Varies
Electro-magnetics	Sparse Matrix Solution	maddog	Yes	Good
Mathematics				
Special Functions	Series	IMSL™	No	Good
Linear Algebra Libraries	Various	BLAS	Varies	Varies
Biology				
Genetics	Search	GenBank db™	No	Good
Defense				

Table 6-2 Taxonomy of Workstation Applications

Discipline	Algorithm(s)	Example(s)	Divide Intensive	Conditioning
Radar Signature Analysis	Dense Matrix Solution	Proprietary	Yes	Varies
Signal Processing	FFT	Proprietary	No	Good

used in Figure 6-1 are explained in Table 6-3 .

Table 6-3 Description of Symbols Used in Outcome Frequencies Algorithm

	K	Number of divides performed/year
	P1	Probability of a divide returning reduced precision
	P2	Probability of a divide inaccuracy leading to a meaningful inaccuracy in the final answer
	P3	Probability that a meaningful inaccuracy in the final answer leads to a problem in use

The number of divides performed in any given period of time is of course dependent on the size and frequencies of the analyses performed on the Pentium processor based platform. It is difficult to select a representative example because the percentage of divides can vary dramatically. For example, in Gaussian Elimination on dense matrices the operation count varies as N^3 where N is the matrix order, while the number of divides is proportional to N. Thus smaller matrices have a much higher proportion of divides and will encounter more divides per unit time, even though the precision of the divides in the larger matrix calculations is more critical. The sparsity pattern also plays a large role as sparse matrix computations encounter divides as a larger percentage of the total operations than do dense matrices. For the purposes of estimation we assume a divide rate of K = 120 million/day. This corresponds to Gaussian Elimination on a 2,000 by 2,000 matrix with a bandwidth of 250 at a flop rate of 30Mflops. (This example is illustrative only and is not intended to quote performance on a specific problem.) A cross check of the data from extensive testing with engineering codes indicates rates approaching, but not exceeding this value. The probability P1 is known from the studies cited earlier in this report. It works out to 1 in 9 billion or 1.11E-10.

The final stage, governed by P3, gives the number of problems expected per year for the system. By “problem” we mean the use of an answer with less than expected precision that has a significantly negative impact on the user. Examples would be failure of designed parts, financial decisions leading to loss of value or erroneous navigation information. The probability P3 is very difficult to estimate, or even to bound. Many errors that could result from a reduced precision divide would cause a calculation to either fail entirely or produce an answer so obviously wrong that it would never be used in practice.

Rather than wrestle with P3 we attempt to bound P2, the probability that the flaw leads to a meaningfully inaccurate result. For this purpose we define *meaningfully inaccurate* as having an accuracy of fewer that three significant digits. Since the inaccuracy in the divide result appears in bit positions between the 12th to the right of the binary point in the mantissa and the

last bit, corresponding to inaccuracies no larger that in the 4th significant digit, an amplification of the inaccuracy must occur for a meaningful inaccuracy to appear in the final result. While it is easy to construct examples in which a single divide inaccuracy can result in a final answer possessing anywhere from full accuracy to no significant digits (The latter outcome is most easily produced by subtracting the result of a slightly inaccurate divide from a number of close magnitude so that the correct result would contain only digits beyond those lost to the inaccuracy), in practice most reduced precision divides are found to be benign.

If P2 were 1.0, indicating that every divide inaccuracy produced a meaningful inaccuracy in the result, the frequency of meaningful inaccuracy would be 1 in 75 days based on the values of K and P1 above. In order for this frequency to fall to a level comparable to the frequency of divide inaccuracies in spreadsheet applications P2 must be of the order of 10^{-4}. The remainder of this section deals with the estimation of P2.

6.3.1.1 Estimating P2

The property of a problem (the algorithm along with its data) that relates errors in the output to errors in the input (or errors introduced by numerical computation) is its *condition*. While the condition can be expressed as a single number for many calculations and can be used in error bounds, for the purposes of this report the condition can be thought of expressing the quality of sensitivity to accuracy in the divide operation. It should be noted that the error in the final answer may actually be less than the error introduced in a particular operation in cases where that calculation ultimately turns out to be a minor contributor to the final answer or in cases where the algorithm is self-correcting (e.g. certain iterative schemes or neural net computing).

An experimental approach is used to estimate P2, the probability that a divide inaccuracy will result in a meaningful inaccuracy in the final result. This approach is preferred over an analytical one since a problem's sensitivity to error is highly dependent on the particular data and the location at which the error is introduced. It can be seen in Table 6-2 that those applications characterized by a large number of divides and poorly conditioned are largely those that deal in dense or sparse matrix algebra, and in particular those involving exotic modelling techniques (such as the use of shell elements in finite element analysis) or eigenvalue extraction (as used in the calculation of vibration modes in structural analysis). Since the use of Pentium processors in the solution of large dense matrix equations is thought to be rare we concentrate on sparse matrix problems. In order to capture the most demanding work loads we ran extensive tests on the QA test suites of MSC/NASTRAN™ and ANSYS™. It should be noted that these engineering codes were provided by their vendors for the ongoing purpose of functional and performance testing on Intel-based systems and their use here in no way constitutes a recommendation or endorsement by the MacNeal-Schwendler or Swanson companies.

NASTRAN and ANSYS represent the upper end of engineering analysis packages and both are frequently run on supercomputers in the calculation of stress, vibration modes, fluid flow, magnetic fields, and other engineering calculations characterized by finite element models. While NASTRAN has few licenses on Intel Architecture systems and ANSYS has only a moderate number, the workloads run on these codes represent a worst case scenario for a Pentium processor-based system in engineering use. Those engineering codes in widespread use on Intel Architecture systems (e.g. AutoCAD*) will not place more stress on the floating point

performance than these codes. Thus our intent in setting up this test program is to identify any possible problems in sparse matrix computing. If problems are found we will then look to the more plentiful applications on PCs to see if the types of analysis found to be susceptible to problems are performed with those codes.

Given the infrequency of divide inaccuracies, and the likelihood that a single inaccuracy will go unnoticed, it is impractical to run problems on a Pentium processor with the divide flaw and wait for an inaccuracy to show up in the output. In fact, at no point in the testing described here was an actual effect from the divide flaw seen. Since the object of the experiment is to determine the effect on the output of the engineering analysis when inaccuracies occur, we introduce inaccuracies artificially and observe the result. Even this plan has the problem that single inaccuracies introduced at random, and with random bit locations for the inaccuracy, will take orders of magnitude too long to produce statistically significant results.

To get a rough estimate of the size of P2 we introduce multiple inaccuracies into single runs of the codes and extrapolate the results to single inaccuracy on problems of comparable complexity. The procedure is as follows:

1. Run all tests with 100% of divides at minimum precision
   (12 good bits to the right of the binary point of the significand)

2. For tests exhibiting meaningful inaccuracies:

   a. Determine minimum number of divides (D) and precision (precis) to generate inaccuracy

   b. For each test:

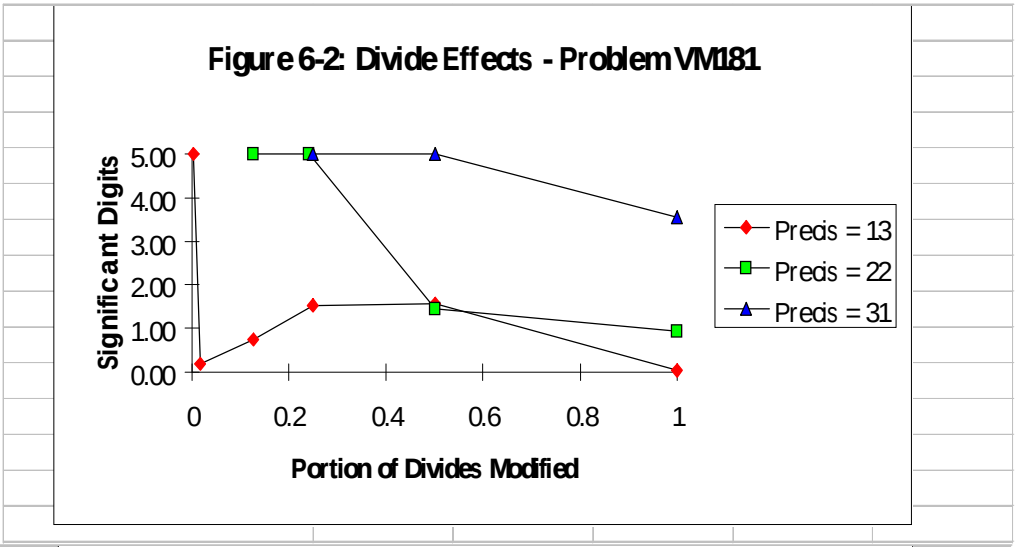
      $p_2 > 1/D$ Prob(precis)

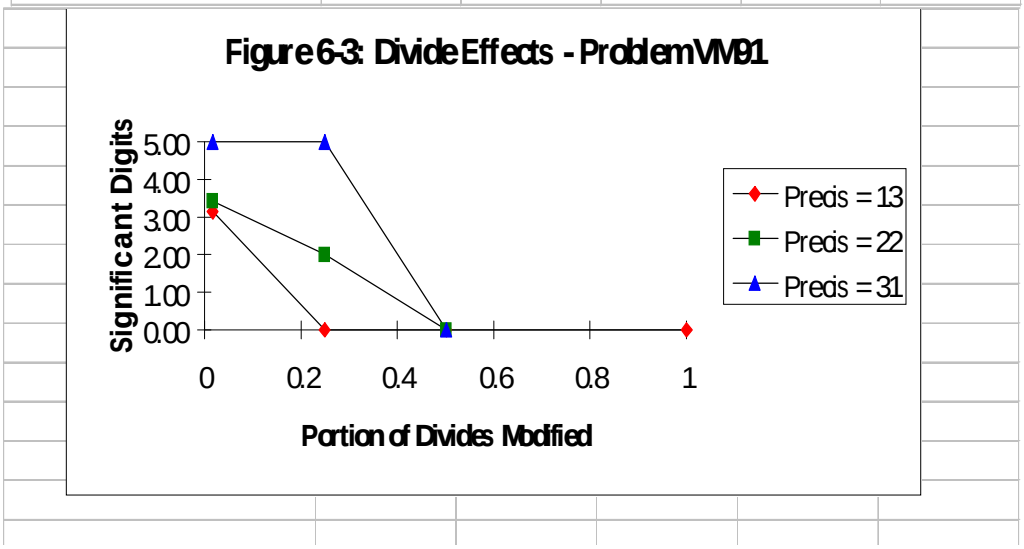
3. Overall estimate of $P_2 > \text{Max}(p_2)$

In step 2b above the Prob(precis) is the probability that the divide inaccuracy will be as bad as that precision level. For example if a precision loss in the 13th binary bit (12th bit to the right of the binary point of the significand) in a double precision operation is required to see a meaningful inaccuracy in the result, Prob(13) would be about 1/40.

Figures 6-2 and 6-3 show typical results for experiments run on problems which exhibit meaningful inaccuracies when run with all divides at minimum precision. In these figures the number of significant digits in the final result is plotted as a function of the portion of divides artificially modified. The three different curves for each figure show results for reducing the

precision of the divide results to different levels. For each precision level the transition from





meaningful inaccuracy (number of significant digits fewer than three) to meaningless inaccuracy (more than three digits of accuracy) is observed. Each such transition yields an estimate for p_2, the probability of meaningful inaccuracy on this problem due to a single random divide precision reduction. We take the largest such estimate to be the value of p_2 for this problem, then estimate P_2 as the maximum over all tests.

6.3.1.2 Experimental Results: Estimation of P2

Table 6-4 shows a representative sample of results for tests where initial screening with all divides at minimum precision indicated the potential for meaningful inaccuracy in the final result. For those tests the procedure outlined above in section 6.3.1.1 was followed to determine the transition from meaningful to meaningless inaccuracy. Each test yields an extrapolation for p_2 and the maximum over all tests is our estimate of P_2.

As indicated at the bottom of Table 6-4 the maximum estimate of P_2 calculated over all tests run to date is 2.2e-4. A value in this range indicates that a meaningful inaccuracy due to the flaw is expected only one time in about one thousand years on a Pentium processor based system. This puts the probability of such an inaccuracy below that of other errors that could affect a system over its lifetime (see Table 5-1) In this table the MTBF refers to the time between meaningful inaccuracies in the final result.

.

Table 6-4 Experimental Estimation of P_2

Test	Type	# divides	p_2	MTBF years
Ansys:vm181	Eigenanalysis of a Flat Circular Plate	90,000	8.9e-6	23,000
Ansys:ev36-4s	Magnetic Field in a Bar	18,000,000	7.1e-8	2,890,000
Ansys:ev56-16	Axisymmetric Stress	28,000	4.5e-5	4,560
Ansys:k2d-8s	Eigenanalysis of Thin Square Plate	2,800,000	7.1e-6	28,900
Ansys:ev36-9s	Magnetic Field around a Stranded Coil	3,700,000	2.2e-7	930,000
Ansys:ev58-15	Thermal Stress Problem	45,000	8.8e-5	2,300
Ansys:ev88-10	Cantilever Beam Bending - Degenerated Shapes	28,000	1.4e-4	1,460
Ansys:ev99-33s	Stress with Overlaid Shell Elements	65,000	6.2e-5	3,310
Ansys:lr63-3	3-D Stress-Free Deflection with Shell Elements	115,000	3.5e-5	5,860
Ansys:ev107-15	Homogeneous Test	260,000	4.9e-5	4,190
Ansys:ev108-24	Thermal Load	6,100	1.6e-4	1,280
Ansys:ev29-5s	Natural Frequency of a Ring in a Fluid	97,000	1.6e-4	1,280
Ansys:ev48-7	Sliding Mass on an Incline	11,000	2.2e-4	930
Ansys:c9322	Sprs	52,000	2.4e-5	8,560
Ansys:c9324	Spring Damper	35,000	4.3e-5	4,770
Ansys:c9356	Large Spectral Analysis	670,000	7.5e-7	274,000
Ansys:vm104	Liquid-Solid Phase Change	60,000	2.1e-5	9,770
Ansys:vm105	Heat-Generating Coil	6,800	7.3e-5	2,810
Ansys:c9351	Test of Pressure Load	1,260,000	6.3e-8	3,259,000
Ansys:c9364	Glue Operation	69,000	1.9e-5	10,800
Ansys:k2d-8s	Eigenanalysis of Thin Square Plate	2,800,000	7.1e-6	28,900

Table 6-4 Experimental Estimation of P_2

Test	Type	# divides	p_2	MTBF years
Ansys:vm26	Large Deflection of a Cantilevered Plate	470,000	2.7e-6	76,000
Ansys:vm33	Transient Thermal Stress in a Cylinder	441,000	1.7e-6	120,700
Ansys:vm34	Bending of a Tapered Plate	6,000	8.3e-5	2,470
Nastran:BCELL5a	Static Analysis of a Cube	100,000	2.0e-6	103,000
Nastran:BCELL6H0	Eigenanalysis of a Cube	2,980,000	1.8e-6	114,000
Result			**2.2e-4**	**930**

6.3.2 Impact on Financial applications

Financial engineering applications which use floating point division are implemented (by both users and software distributors) in spreadsheets, in high level languages and through use of statistical software packages. To consider the potential impacts of the flaw for the financial engineer, we will divide the work space into four categories. The first set is the collection of users performing corporate or marketing analysis oriented calculations. The next set contains the most frequent financial analytics such as present values, annuities, depreciations and basic financial quantities. The last two sets comprise the most intensive computation and mathematical models.

This summary classification of the financial applications and the impact of the flaw is given in Table 6-5 .

Table 6-5 Classification of financial applications

Usage	Examples	Division intensive	Impact
Standard spreadsheet analysis	Corporate finance, budget or marketing analysis,	No	None
Basic financial calculations	Present value, yield to maturity	Some	Significant only in the extreme circumstance of > 10 million divisions per day
Complex mathematical models	Black-Scholes model, Binomial model	Some	Could be significant on continuous use
Path based models and simulations	Monte Carlo risk analysis, non recombining paths	Yes	Significant unless there is a low P2 factor.

Notice that the number of users of each category is inversely proportional to the severity of the impact. The vast majority of users are included in the first category. The last category represents applications which have only recently been transferred to the desktop.

In the following sections we apply the characterization methodology (P1, P2) used in the section on engineering applications to the 4 sets.

6.3.2.1 Values of P2

While we do not provide a detailed analysis of the P2 probability (the probability that the flaw leads to a meaningful inaccuracy) for financial applications here, we will make the following comments. The P2 value for these applications is often either close to 1.0 or 0.0. The former leaves the risk at P1, the latter reduces the risk to zero.

P2 is close to 0.0 when dealing with random number generators, where any random number is as good as another, provided the basic distribution is not changed. Since the inaccuracy happens in only 1 of nine billion divisions, there will be no change to the estimate of the distribution.
Again, P2 is close to 0.0 on simulations which use a large number of paths and perform expected value analysis. An error on one path will not have a significant impact on the final answer. The number of paths are always far less than nine billion, so that more than one error among the paths is very unlikely; and for more than two paths to have errors is prohibitively unlikely.
Finally, P2 is 0.0 when the number of significant decimal digits the user needs is less than four.

P2 moves from near 0.0 to near 1.0 as the need for significant decimal digits reaches 15. This is because the inaccuracy seems to be equally likely to occur at each significant digit beyond four. For instance, if the user needs six significant digits, and an error occurs, then (assuming double precision arithmetic), the probability that the inaccuracy was in fourth through sixth significant decimal digits is 3/15 = 0.2.

Most other times the P2 value will be near 1.0.

6.3.2.2 MTBF estimation

CATEGORY 1
For applications from the 1st set, such as corporate financial analysis and forecasting, marketing analysis, planning and so forth, the likelihood of encountering reduced precision divides is low. This is because typical calculations here are dominated by comparisons and additions. The input-output operations and the time for human conception of the results consume more time then the processor spends performing arithmetic operations. This effect limits the number of divisions that are computed per day to well below what is necessary to have any appreciable probability of experiencing a meaningful inaccuracy. As an example, consider a large budget calculation implemented as a 700x700 cell spread sheet, which is run an average of a few times a day. This will produce less than 10,000 divisions a day (on average); so few divisions that no error is likely to be seen for thousands of years.

CATEGORY 2
In the second set of usages, one of the most frequent calculations is discounting a value to the present, which typically involves an expression such as (c/(1+r)^t). This discount process is generally connected with some method of generating an associated cash flow. The number of divisions is about one-fifth of the total operations (or less) and about equal to the number of exponentiations. In the most extreme case, where the calculation is a simple present value, the

60MHz Pentium processor running 24hrs per day could produce at most 500 million divisions and exponentiations, resulting in a MTBF of 18 days. In more realistic applications, the number of PV calculations is of order of 1000 or less within the spreadsheet and the spreadsheet is recalculated no more than 100 times a day. This produces at most 500,000 divides a day for a worst case MTBF of more than 50 years. This possibility is considerably less than the chances of a system memory error, which could be equally inaccurate.

CATEGORY 3

The third set is represented by the Black-Scholes and simple binomial models. Black-Scholes solutions generally require approximations to be made for standard normal distributions in order to run them on any desktop computer. These approximations will increase the ratio of divide time to compute time. Divisions, exponentiation, and natural logarithms take about one-fifth of the actual computation time. Models pricing a few thousand options are run at most a few times per hour, representing approximately a million divisions and transcendental computations per day. The MTBF would then be roughly 30 years. In the extreme case where a user does not look at all the results, and continually recalculates the models, the upper bound of calculations (running 24 hours per day at full rate) is about 1 billion divisions per day yielding an MTBF of 9 days. Again, if the accuracy required is less than four digits, then even such maximal use will not produce a meaningful inaccuracy.

Simple Binomial models are usually implemented with a discount computation at each node of the model and two simple integer divisions. The number of divisions and the number of exponentiations are of the same order of magnitude, each being about 1/5th of the total number of operations. For an analysis of a few thousand options a day, MTBF would exceed 30 years.

CATEGORY 4

The last category of applications focuses on the valuation of more complicated derivatives and the use of simulation. Representative applications for this set include non-simple binomial models, as well as trinomial and finite difference methods. Simulation analysis usually employs Monte Carlo techniques to arrive at valuations for complex securities with large numbers of embedded options such as CMOs (Collateralized Mortgage-backed Obligations).

More complicated binomial models, such as those with non-stationary dividends, and other valuation techniques such as non-recombining trees and finite difference methods can severely increase the number of computational steps performed in a valuation. In the case of non-simple binomial models, for example, realistic problems might have an MTBF of three years or less. However, while finite difference problems also use significant numbers of divides, real applications of these techniques involve extensive non-division operations in order to implement useful algorithms. This can greatly reduce the time spent doing divisions, resulting in very low divisions per day. These methods can also be iterative, so that an inaccuracy on one iteration will disappear in following iterations.

When simulation analysis is used for valuation, the number of cash flows valued must be relatively large. This is significant since the extremely large numbers of discount operations greatly increases the rate of divisions per day. For those circumstances where continuous use of a desktop platform is being made to solve these computationally intensive applications, the MTBF may well be less than a week. However, this may be ameliorated by the P2 factor as discussed above.

In conclusion, the large majority of financial users will not experience any problems from the flaw. The problem may manifest itself significantly in those programs for valuing the most complicated financial instruments. Even in this case, if the valuation is statistically based, single division inaccuracies may be harmless. The user should consider the number of divisions performed per day and the context in which the resulting quotients are used.

6.4 Impact on Server Applications

Server applications do not use the relevant floating point instructions. The flaw has no impact on them.

7 Conclusions

The thorough and detailed characterization of the flaw and the subsequent investigations of its impact on applications through elaborate surveys, analyses and empirical observation lead us to the following conclusions:

1. The significance of the flaw depends upon (a) the rate of use of specific FP instructions in the Pentium™ CPU, (b) the data fed to them, (c) the way in which the results of these instructions are propagated into further computation in the application; and (d) the way in which the final results of the application are interpreted.

2. The flaw is of no significance in the commercial PC market where the vast majority of Intel processors are installed. Failure rates introduced by this flaw are swamped by rates due to existing hard and soft failure mechanisms in PC systems. The average PC user is likely to encounter a failure once in 27,000 years due to this flaw, indicating that it is practically impossible for such a user to encounter a problem in the useful lifetime of the product.

3. The flaw is of no significance for integer workstation applications, since they do not use the Floating Point Unit.

4. The flaw is of no significance for Server applications.

5. The flaw is of no significance in the majority of the financial world, where PC users run spreadsheets with little divide content. For these users the flaw has no effect.

6. The flaw is of potential significance for a small minority of users in the financial world. These users are primarily involved in running highly numerical applications involving intensive recalculations such as path-dependent derivatives valuations and those valuations involving simulations. Depending on the circumstances, these users should employ either an updated Pentium processor without the flaw or a software workaround.

7. A small fraction of PCs are installed for use as engineering/scientific workstations. Although there may be an occasional occurrence of a reduced precision di-

vide, our extensive experiments with a range of engineering problems covering CAD, structural analysis, computational fluid dynamics and circuit simulation indicate that meaningful inaccuracies in the end-result will only be seen once in about 1,000 years. Technical users running other applications requiring unusual precision and employing millions of divides per day should employ either an updated Pentium processor without the flaw or a software workaround.

Our overall conclusion is that the flaw in the floating point unit of the Pentium processor is of no concern to the vast majority of users. A few users of applications in the scientific/engineering and financial engineering fields may need to employ either an updated processor without the flaw or a software workaround.

8 Acknowledgments

Acknowledgments are due to all the engineers and computational scientists who participated in the characterization effort. Special thanks to Joe Brandenburg (Principle Computational Scientist, Scalable Systems Division) for reviewing the characterization of the technical applications and for contributing to the section on financial applications, to Richard Passov (Senior Manager, Quantitative Analysis, Corporate Treasury) for contributing to the financial application section, to Luke Girard and Patrice Roussel (Senior Design Engineers, Microprocessor Products Division) and to Dr. Peter Tang (Numerical Scientist, Argonne National Laboratories) for analytically and experimentally characterizing the hardware algorithm and the flaw.

9 References

[1] "Architecture of the Pentium Microprocessor", by Donald Alpert and Dror Avnon, IEEE Micro,V-13:11-21(June 1993).

[2] "Higher-Radix Division Using Estimates of the Divisor and Partial Remainders", Daniel E. Atkins, IEEE Transactions on Computing, C-17:925-935(1968)

941130-1.1

APPENDIX A

In the process of the characterization, a list of scientific constants that might commonly be used in floating point calculations were examined as potential divisors. These constants are shown in Table A-1 below. Two of these constants, indicated by the shaded rows, were identified as potentially problematic based on the analysis in Section 4.2. To verify if these two constants were really at risk each was used as a divisor with 100 billion random dividends with no errors found.

Table A-1 Engineering and Scientific Constants Analyzed.

Constants	Symbol	Value x	Hex Value
Absolute zero	0 degK	-273.16	$-1.1128F5C28F5C30\ 2^{8}$
Acceleration of gravity (at sea level)	g	9.78049	$1.38F9C62A1B5C80\ 2^{3}$
Avogadro's constant	No	$6.02204\ 10^{23}$	$1.FE162D50E9DBA0\ 2^{78}$
Bohr radius	aB	0.52917	$1.0EEF5EC80C73B0\ 2^{-1}$
Boltzmann's constant	k	$1.38066\ 10^{-23}$	$1.0B0EF8C6F0C050\ 2^{-76}$
Coulomb constant	C	$8.98742\ 10^{9}$	$1.0BD892B0000000\ 2^{33}$
Distance from Earth to the Moon		$3.84\ \ 10^{8}$	$1.6E360000000000\ 2^{28}$
Distance from Earth to the Sun	1 UA	$1.49\ \ 10^{11}$	$1.1588BC90000000\ 2^{37}$
Earth mass	mE	$5.975\ \ 10^{24}$	$1.3C505989B54FC0\ 2^{82}$
Electron mass	me	$5.488\ \ 10^{-4}$	$1.1FBAB06A9676D0\ 2^{-11}$
Electron rest mass	m0	$0.91095\ 10^{-30}$	$1.279EC3E2589790\ 2^{-100}$
Elementary charge	q	$1.60218\ 10^{-19}$	$1.7A4DD6A6DAEE80\ 2^{-63}$
Electron volt	eV	$1.60218\ 10^{-19}$	$1.7A4DD6A6DAEE80\ 2^{-63}$
Equatorial radius of Earth		$6.378\ \ 10^{6}$	$1.85484000000000\ 2^{22}$
Eccentricity of Earth's orbit		0.0167	$1.119CE075F6FD20\ 2^{-6}$
Faraday constant	F	96500	$1.78F40000000000\ 2^{16}$
Gas constant	R	1.98719	$1.FCB87BDCF03080\ 2^{0}$
Heat Mechanic Equivalence	J	$4.185\ \ 10^{3}$	$1.05900000000000\ 2^{12}$
Hydrogen atom mass (neutral)	mH	1.008142	$1.02159817B95A30\ 2^{0}$
Loschmidt's constant		$2.68841\ 10^{9}$	$1.407BC320000000\ 2^{31}$
Neutron mass	mn	1.008982	$1.024CA4F440AF20\ 2^{0}$
Permeability in vacuum	u0	$1.25663\ 10^{-8}$	$1.AFC657FFABA690\ 2^{-27}$
Permittivity in vacuum	e0	$8.85418\ 10^{-14}$	$1.8EC1BECA727920\ 2^{-44}$
Planck constant	h	$6.62617\ 10^{-34}$	$1.B86270C89B7D70\ 2^{-111}$
Proton mass	mp	1.007593	$1.01F19D66ADB400\ 2^{0}$

Table A-1 Engineering and Scientific Constants Analyzed.

Constants	Symbol	Value x	Hex Value
Rydberg constant for a nucleus of infinite mass	Rinf	109737	$1.ACA90000000000\ 2^{16}$
Rydberg constant for hydrogen	Rh	109678	$1.ACA90000000000\ 2^{16}$
Speed of light in vacuum	c	$2.99792\ 10^{8}$	$1.1DE76800000000\ 2^{28}$
Standard Atmospheric Pressure	1atm	$1.01325\ 10^{5}$	$1.8BCD0000000000\ 2^{16}$
Sun diameter		$1.39\ \ 10^{9}$	$1.4B66DE00000000\ 2^{30}$
Sun mass	mS	$1.99\ \ 10^{30}$	$1.91E096B424EC50\ 2^{100}$
Thermal voltage at 300 K	kT/q	0.0259	$1.A858793DD97F60\ 2^{-6}$
Universal Gravitation Constant	G	$6.673\ \ 10^{-11}$	$1.257B4D002790D0\ 2^{-34}$
Volume of a perfect gas at 0 degree C and 1 atm.		22.415	$1.66A3D70A3D70A0\ 2^{4}$
Wavelength of 1-eV quantum	lambda	1.23977	$1.3D6191148FDA00\ 2^{0}$
Wien constant for displacement's law		0.2898	$1.28C154C985F070\ 2^{-2}$

Table A-2 below contains the list of combinations of the aforementioned constants that were examined additionally for possible occurrences of an error due to the flaw. No problem was found.

Table A-2 Constants Used in Combination

Equation	Constants
Magnetic induction (Biot-Savart). $B = (2C/c^2)(I/r)$	C = Coulomb constant c = speed of light
Planck's formula for spectral emittance: $Wf=(2Pih/c^2)(f^3/(e^{hf}/kT)-1)$	h = Planck constant c = speed of light k = Boltzmann's constant
Clausius state equation: p(V - nb) = nRT	$b = (2/3)(No)(Pi)(d^3)$ No = Avogadro's number
Schroedinger equation (Bohr's postulate): mvr = n(h/2Pi)	h = Planck constant
c^2	$8.987524\ 10^{16}\ \ 1.3F4D0DFA4D9000\ 2^{56}$
k/c^2	$9.999883\ 10^{-8}\ \ 1.AD7DE058686390\ 2^{-24}$
h/c^2	$7.372630\ 10^{-51}\ 1.61142E489F6160\ 2^{-167}$
h/k	$4.799277\ 10^{-11}\ 1.A6261C4CA09310\ 2^{-35}$
h/2Pi	$1.054587\ 10^{-34}\ 1.185B78DB562900\ 2^{-113}$
No*Pi	$1.891879\ 10^{24}\ \ 1.909EF751EF7740\ 2^{80}$

Table A-3 below indicates the commonly used multiples of the important constants and their multiples that were checked. None of these multiples were found to be at risk.

Table A-3 Multiples of Common Constants

4e	1.0873127 10^{1}	1.BD5D000B3DA0D0 2^{3}
2e	5.4365636	1.5BF0A87427F010 2^{2}
e	2.7182818	1.5BF0A87427F010 2^{1}
e/2	1.3591409	1.5BF0A87427F010 2^{0}
e/3	0.9060939	1.CFEB8A2735CEF0 2^{-1}
e/4	0.6795704	1.5BF0A6C6A8C660 2^{-1}
e/5	0.5436563	1.165A1E59875AD0 2^{-1}
e/6	0.4530469	1.CFEB86CC377B90 2^{-2}
e/7	0.3883259	1.8DA54E02C25220 2^{-2}
e/8	0.3397852	1.5BF0A6C6A8C660 2^{-2}
6Pi	1.8849556 10^{1}	1.8209F5CD213080 2^{-3}
4Pi	1.2556637 10^{1}	1.0128F0E504FCF0 2^{-3}
2Pi	6.2831853	1.921FB53C8D4F10 2^{2}
Pi	3.1415927	1.921FB5A7ED1970 2^{1}
Pi/2	1.5707963	1.921FB4D12D84A0 2^{0}
Pi/3	1.0471976	1.0C152454731EE0 2^{0}
Pi/4	0.7853981	1.921FB323AE5AF0 2^{-1}
Pi/5	0.6283185	1.41B2F661F18CA0 2^{-1}
Pi/6	0.5235987	1.0C1520F974CB80 2^{-1}
Pi/7	0.4487989	1.CB91F057EC5020 2^{-2}
Pi/8	0.3926991	1.921FB67EACAE50 2^{-2}
sq(2)	1.4142136	1.6A09E7098EF500 2^{0}
sq(3)	1.7320508	1.BB67AE6502B910 2^{0}
sq(Pi)	1.7724539	1.C5BF89EE2AEB60 2^{0}