

Lecture 02 - A

§1.2 Introduction to Floating Point Numbers

T. Takahashi

Department of Mathematics and Statistics
University of New Mexico



Today we will explore:

- How does a computer represent real numbers?
- Why can't every real number be represented exactly?
- What is floating-point roundoff error?
- What are the limitations of floating-point numbers?
- How does the IEEE-754 standard represent floating-point numbers?



Floating Point Numbers



How Do We Represent Numbers?

- The number $x = 1234.1234$ is a sum of two parts:

$$1234 = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

and

$$.1234 = 1 \times 10^{-1} + 2 \times 10^{-2} + 3 \times 10^{-3} + 4 \times 10^{-4}$$

- In general, we can write $x \in \mathbb{R}$ as an **integer part** and a **fractional part**:

$$\underbrace{a_3 a_2 a_1 a_0}_{\text{integer}} . \underbrace{b_1 b_2 b_3 b_4 \dots}_{\text{fractional}}$$

- It is common for the fractional part to contain infinite digits

$$\pi = 3.14159 \dots$$

$$1/9 = 0.11111 \dots$$

$$\sqrt{2} = 1.41421 \dots$$



Base β Number System

Definition: Base β -system

In the β -system, numbers are represented as:

$$(a_n a_{n-1} \dots a_1 a_0 . b_1 b_2 b_3 \dots)_\beta = \sum_{k=0}^n a_k \beta^k + \sum_{k=1}^{\infty} b_k \beta^{-k}$$

The **base** β is an integer. The **digits** a_k, b_k are integers that range from 0 to $\beta - 1$. The subscripts $n, n-1, \dots, 0, 1, 2, 3$ are called **positions**.

- The following is an example of $\beta = 10$ format

$$1234 = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

- Think of every number as

$$\sum (\text{digit} \times \text{power})$$



Binary to Base 10

Example

Convert $(1001.01)_2$ to base 10.

- Create a table with position, power, and digit

position	3	2	1	0	-1	-2
power	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}
digit	1	0	0	1	0	1

- Add up $\sum (\text{digit} \times \text{power})$

$$\begin{aligned}(1001.01)_2 &= 1(2^3) + 0(2^2) + 0(2^1) + 1(2^0) + 0(2^{-1}) + 1(2^{-2}) \\ &= 8 + 1 + \frac{1}{4} = 9\frac{1}{4} \\ &= (9.25)_{10}.\end{aligned}$$



Example

Convert $(12.5)_{10}$ to base 2.

- Write the number as a sum of powers of 2:

$$12.5 = 8 + 4 + \frac{1}{2}$$

- Therefore,

$$12.5 = 2^3 + 2^2 + 2^{-1}.$$

- Place a 1 in those positions and a 0 everywhere else:

$$(12.5)_{10} = (1100.1)_2.$$



Decimal Location

- Issue with the β system: **location of the decimal is arbitrary**

$$(a_n a_{n-1} \dots a_1 a_0 \underbrace{}_{\text{decimal}} b_1 b_2 b_3 \dots)_\beta = \sum_{k=0}^n a_k \beta^k + \sum_{k=1}^{\infty} b_k \beta^{-k}$$

- Notice that 1234 can be written as

$$\begin{aligned} 1234 &= (0.1234)_{10} 10^4 \\ &= (0.01234)_{10} 10^5 \\ &= (0.001234)_{10} 10^6. \end{aligned}$$

- We need a standard convention



Definition: Normalized Floating-Point Representation

A base- β number written in the form

$$x = \pm(0.d_1d_2d_3 \dots d_m)_\beta \times \beta^e$$

is in the **normalized floating point representation**.

- Put decimal in the very front
- The digits $d_1d_2d_3 \dots d_m$ form the **mantissa**.
- e is the **exponent** which is an integer.
- Each digit d_i is an integer that ranges from 0 to $\beta - 1$.
- The general normalized form requires $d_1 \neq 0$.



Example

Write the following numbers in normalized floating-point representation.

- 1000.12345
- 0.000812345
- $(1101.001)_2$

$$1000.12345 = (0.100012345)_{10} \times 10^4$$

$$0.000812345 = (0.812345)_{10} \times 10^{-3}$$

$$(1101.001)_2 = (0.1101001)_2 \times 2^4$$



Why Do We Need Binary?

- Computers store information using **bits**

0 1

- Therefore, numbers must ultimately be encoded using only

0's and 1's.

- For example, the decimal number

$(12.5)_{10}$

must first be represented in binary:

$$(12.5)_{10} = (1100.1)_2.$$

Key Idea

To understand how computers store real numbers, we first need to understand how numbers are represented in binary.

Bit Sequence

Definition: Bit Sequence

A **bit sequence** is an ordered series of finite binary digits that encode information.

The Number

$$x = 12.5$$



Binary Representation

$$x = (1100.1)_2$$



Bit Sequence

1	1	0	0	1
---	---	---	---	---

The Fundamental Flaw

Example

Write the number $x = \frac{1}{5} = 0.2$ as a binary floating-point number.

$$\begin{aligned}(0.2)_{10} &= 0.125 + 0.075 \\&= 2^{-3} + 0.0625 + 0.0125 \\&= 2^{-3} + 2^{-4} + 2^{-7} + 0.00446875 \\&\vdots \\&= (.0011\,0011\,0011\,\dots)_2 \\&\approx \begin{array}{|c|c|c|c|c|c|c|c|}\hline 0. & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\\hline\end{array}\end{aligned}$$

The binary is infinite but the bit sequence must be finite

Definition: Roundoff

The computer is not able to perfectly represent $x = (0.2)_{10}$ in finite memory. This type of error is called **roundoff error**.

Main Idea

Ideal Goal

We want to be able to express **any** $x \in \mathbb{R}$ on the computer.

Issue

Computers can only use a **finite** amount of bits to represent x which is an approximation $x \approx \hat{x} =$

0.	0	0	1	1	0	0	1
----	---	---	---	---	---	---	---

, which is often insufficient.

Result

In the majority of cases, we will have **rounding errors** $E = |x - \hat{x}|$ when we try to represent a real number on a computer.



A Simple Example



Mantissa and Exponent

Question

We have a limited number of bits to express $x \in \mathbb{R}$. How can we use those bits efficiently?

Idea: Normalize and Truncate

- Express x in normalized binary form $x = (0.d_1d_2d_3\dots)_2 \times 2^e$.
- Keep only a finite number of mantissa bits: $m = (0.d_1d_2\dots d_m)_2$.
- Store exponent bits separately: $e = (e_1e_2\dots e_n)_2$.
- Compact storage

$$x \approx \text{fl}(x) = \hat{x} = \boxed{.d_1} \boxed{\dots} \boxed{d_m} \parallel \boxed{e_1} \boxed{\dots} \boxed{e_2}$$

Definition: Parts of a Float

- m is the mantissa (significant bits).
- e is the exponent (stored in binary).

Hypothetical 5-bit Float

Example

Suppose we use *only five bits* to represent floating-point numbers. 3 bits for the mantissa, and 2 bits for the exponent encoded as:

$.d_1$	d_2	d_3	e_1	e_2
--------	-------	-------	-------	-------

which represents the value

$$x = (.d_1d_2d_3)_2 \times 2^{(2e_1-1)e_2}.$$

What is the range of values which can be represented in this format?

- Example: The sequence

.1	0	1	0	1
----	---	---	---	---

 represents

$$x = (.101)_2 \times 2^{-1} = \frac{5}{8} \times \frac{1}{2} = \frac{5}{16}$$

- The mantissa can take on $2^3 = 8$ unique numbers
- The exponent can take on the values $e = -1, 0, 1$



Range of the 5-bit Float

- We list all possible values:

$$(.000)_2 \times 2^e = 0$$

$$(.001)_2 \times 2^e = 1/8 \cdot 2^e = 1/16, 1/8, 1/4$$

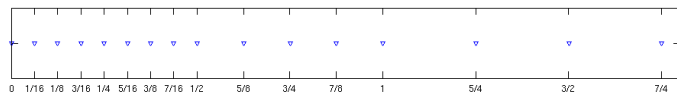
$\vdots$

$$(.111)_2 \times 2^e = 7/8 \cdot 2^e = 7/16, 7/8, 7/4.$$

- **Range** is the set of representable numbers

$$0, 1/16, 1/8, 1/4, \dots, 7/16, 7/8, 7/4.$$

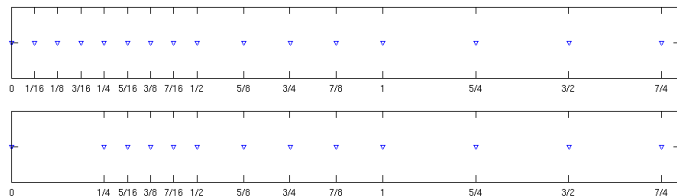
- Range graphed on the number line



Problems with the 5-bit Float

Part 1

- 1. **Loss due to Normalization:** If we require the leading mantissa bit to be 1, then values like $1/16$, $1/8$, $3/16$ cannot appear in normalized form.



- 2. **Overflow:** Maximum value is $7/4$. Multiplying $\hat{x} = 7/4$ by itself yields

$$\hat{x} \times \hat{x} = 7/4 \times 7/4 = 28/16 = ???$$

- 3. **Underflow:** There is no representable value between 0 and $1/8$. Multiplying $\hat{x}_1 = 1/8$ by $\hat{x}_2 = 1/2$ yields

$$\hat{x}_1 \times \hat{x}_2 = 1/8 \times 1/2 = 1/16 = ???$$



Problems with the 5-bit Float

Part 2

- 4. **Redundancy:** Different bit patterns represent the same number.

$$(.100)_2 \times 2^0 = (.010)_2 \times 2^1 = 1/2.$$

- 5. **Unsigned Only:** With no sign bit, the format cannot represent negative numbers. Half of the real line is missing.

Key Idea

Our 5-bit example reveals the essential difficulties in floating-point design. These are exactly the issues the IEEE-754 standard solves.



IEEE Floating Point Standard



What is the proper float representation?

- **Goal:** Find an acceptable strategy to represent $x \in (-\infty, \infty)$.
- **Limitation:** Computers can use a finite number of bits

$$x \approx \hat{x} = \boxed{.d_1} \boxed{\cdots} \boxed{d_m} \boxed{e_1} \boxed{\cdots} \boxed{e_2}$$

Considerations

- cover a wide range of real numbers
- represent positive and negative numbers
- minimize the redundancy of bit sequences
- lower risk of underflow
- lower risk of overflow
- lower risk of rounding errors



- In 19870s all computer companies **had their own representation** which lead to **different results** even with the same software
- IEEE-754 (adopted 1985) established a standard representation
 - Scientists gathered to establish a memory-efficient way to represent numbers
 - Defines the floating point distribution for our computation
 - Offers several rounding modes which effect accuracy

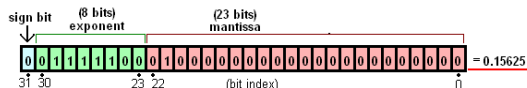
Key Idea

- **Unified** floating points and arithmetic across all computers
- Rounding errors have become more predictable and easier to study

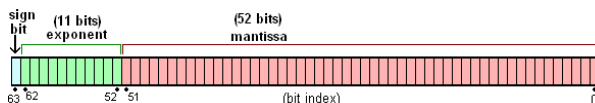


IEEE Floating Point Format (IEEE-754)

- IEEE memory is built around **words** - 32 bits of memory
- IEEE may use one word, two words, or four words for representing $x \in \mathbb{R}$
- **Single-precision** refers to using one word



- **Double-precision** refers to using two words



- Matlab uses double precision



IEEE Single Precision Floating Point Format

Definition: IEEE Single Precision Float (FP32)

The **single precision float** has the form

$$x = \pm q \times 2^m.$$

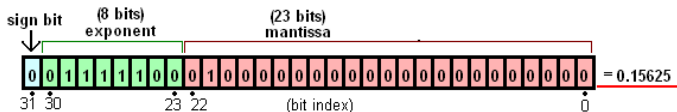
It occupies 32 bits of memory. It is a specific standard format set by IEEE 755.



Allocation

- 1 bit for sign $\pm$
- 8 bits for integer exponent is m
- 23 bits floating point mantissa q

IEE Mantissa



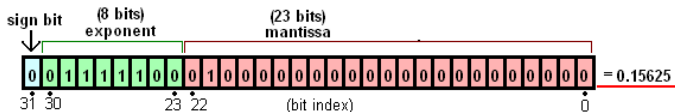
Mantissa

- Entirely composed of 1s and 0s
- Has a **hidden bit**, always assumed to be 1 (no memory)
- Hidden bit reduces redundancy of the bit sequence
- The mantissa is computed as

$$q = (1.b_{22}b_{21} \dots b_1b_0)_2$$



IEE Exponent



Exponent

- The exponent m needs to be able to attain positive and negative numbers
- IEEE exponent is in **biased form**
- The biased form makes sure the exponent is properly ordered with respect to the bit sequence
- The exponent is computed

$$m = (b_{30}b_{29} \dots b_{23})_2 - 127$$

- The exponent is in the form $m = c - o$.
- The c is the 8-bit integer $c = (b_{30}b_{29} \dots b_{23})_2$
- The integer o is called the **offset**
- The offset is $o = 2^7 - 1 = 127$



Why Biased?

Consider the non-biased signed integer

$$m_1 = \begin{array}{|c|c|c|c|c|c|c|c|} \hline \pm & e_0 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\ \hline \end{array}$$

- Minimum: $\begin{array}{|c|c|c|c|c|c|c|c|} \hline - & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline \end{array} = -127$
- Maximum: $\begin{array}{|c|c|c|c|c|c|c|c|} \hline + & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline \end{array} = 127$
- It has a redundant ± 0

Consider the biased integer

$$m_2 = \begin{array}{|c|c|c|c|c|c|c|c|} \hline e_0 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \\ \hline \end{array} - 127$$

- Minimum: $\begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} = -127$
- Maximum: $\begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline \end{array} = 255 - 127 = 128$
- No redundancy
- Bit sequences are ordered from least to greatest

Example

Express the number $x = -52.125$ as a bit sequence in FP32.

- 1 Convert base-10 to base-2

$$(-52.125)_{10} = -(110100.00100000000)_2$$

- 2 Convert to $1.f$ exponential form:

$$x = \underbrace{-}_{1} (1. \underbrace{101\,000\,010\,000 \dots 0}_{23})_2 \times 2^5$$

- 3 Calculate exponent: $5 = c - 127 \Rightarrow c = 132 \Rightarrow c = (\underbrace{10\,000\,100}_8)_2$

- 4 Write the bit sequence

$$\underbrace{1}_1 \underbrace{10\,000\,100}_8 \underbrace{101\,000\,010\,000 \dots 0}_{23}$$



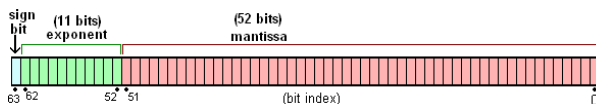
IEEE Double Precision Floating Point Format

Definition: IEE Double Precision Float (FP64)

The **double precision float** has the form

$$x = \pm q \times 2^m.$$

It occupies 64 bits of memory. It is a specific standard format set by IEEE 755.



Allocation

- 1 bit for sign
- 11 bits for exponent
- 52 bits for mantissa

FP32 vs FP64: A Comparison

Feature	FP32 (Single)	FP64 (Double)
Total Bits	32 bits	64 bits
Sign Bit	1	1
Exponent Bits	8	11
Mantissa Bits	23	52
Exponent Bias	127	1023
Significant Digits (approx)	7	15–16
Machine Epsilon	$2^{-23} \approx 1.19 \times 10^{-7}$	$2^{-52} \approx 2.22 \times 10^{-16}$
Used In	GPUs, mobile devices	Scientific computing

Key Idea

FP64 offers much higher precision and range, at the cost of memory and performance. Matlab uses FP64.



Real Numbers vs. Matlab Numbers

Definition: Floating point representation

Let $x \in \mathbb{R}$. The **floating-point representation** of x is denoted $\text{fl}(x)$ is the nearest representable number in the working precision.

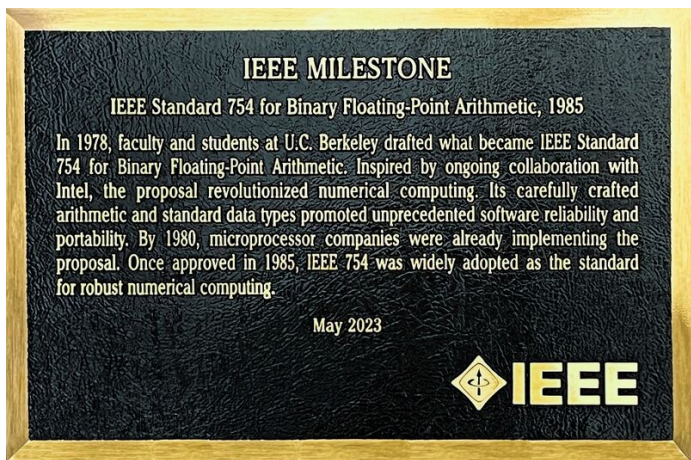
Example

If $x = 0.2$, then what is $\text{fl}(x)$ in Matlab?

```
1 >> fprintf('%%.25f', 0.2)
2 0.2000000000000000000000000111022302
```

- On paper, $x = 0.2 \in \mathbb{R}$
- In Matlab, $\text{fl}(x) = 0.2000000000000000000000000111022302\dots$





Key Idea

- Standardized floats was important for reproducibility
- Important milestone for **numerical error analysis**

Story



Who cares about a small amount of rounding error?

Business Day

The New York Times

Copyright © 1994 The New York Times

BUSINESS
Digest

THURSDAY, NOVEMBER 24, 1994

DOW	DOLLAR	OIL	BONDS
30	vs. Japanese	Novmax	30-Year
Industrials	Yen	Spot	Treasury
3,674.63	98.47 Yen	\$18.15	7.94%
-3.36	+0.24 Yen	+0.33	-0.08

Companies

A flaw in the Pentium, the top chip made by Intel, can cause inaccurate calculations in certain rare cases. Intel said the chip would not have to be recalled, but many scientists and engineers who depend on precise calculations are concerned. [Page D1.]

Gibson Greetings and Bankers Trust settled a lawsuit over derivatives. Gibson agreed to pay Bankers Trust nearly \$6.3 million, or about 39 percent of the \$20.7 million that Bankers claimed it was owed in the trades between the two. [D1.]

Sorry, Charlie: H. J. Heinz and Leo Burnett are parting ways after 37 years. The Burnett agency created familiar advertising characters like Mavis the frisky cat and Charlie the tuna. [D1.]

Pfizer will buy SmithKline Beecham's animal health business for \$1.5 billion in cash. The deal would make Pfizer the world's largest maker of animal drugs, topping Merck. [D4.]

Kmart is seeking ways to save as much as \$80 million in two years in addition to store closings and job cuts. [D4.]

A Federal judge from a New York brokerage account of a British businessman accused of illegal insider trading. [D4.]

The F.T.C. is seeking data on alliances with Carmax International, in a broadening of an investigation of relations between drug makers and managed health-care drug distributors. [D4.]

Metalgesellschaft of Germany plans to write down the paper value of its shares by half in an effort to ease itself. [D4.]

I.B.M., AT&T, Apple Computer and Siemens plan to create a universal communication language for computers. [D5.]

Markets

After a roller-coaster session, stocks ended slightly lower. The Standard & Poor's 500 index fell just 0.15 points, easing investors' concerns that stocks were in for a long, hard descent. [D1.]

A stock slide is not likely to change Federal Reserve policy. Indeed, stocks' strength until now surprised Fed officials. [D8.]

Stock markets fell sharply across Europe as investors took their cues from Wall Street and unable to offset the gains. [D8.]

Flaw Undermines Accuracy of Pentium Chips

By JOHN MARKOFF

Special to The New York Times

SAN FRANCISCO, Nov. 23 — An elusive circuitry error in Intel's Pentium chips used in millions of computers to generate inaccurate results in certain rare cases, heightening anxiety among many scientists and engineers who rely on their machines for precise calculations.

The flaw, an error in division, has been found in the Pentium, the current top microprocessor of the Intel Corporation, the world's largest chip maker. The chip, in several different configurations, is used in many com-

puters and for home and business use, including those made by I.B.M., Compaq, Dell, Gateway 2000 and others.

The flaw appears in all Pentium chips now on the market. In certain types of division problems involving more than five significant digits, a mathematical term that can include numbers before and after a decimal point.

Intel declined to say how many Pentium chips it made or sold, but Datasheet, a market research company in San Jose, Calif., estimated that in 1991 Intel would sell 5.5 million to 6 million Pentiums, roughly 10 percent of the number of personal

computers sold worldwide.

Intel said yesterday that it did not believe the chip needed to be recalled, asserting that the typical user would have but one chance in more than nine billion of encountering an inaccurate result as a consequence of the error, and thus there was no noticeable consequence to users of business or home computers. Indeed, the company said it was continuing to send computer makers Pentium chips built before the problem was detected.

William Kahan of the University of California at Berkeley, one of the nation's experts on computer mathematics, expressed skepticism about

Intel's contention that the error would only occur in extremely rare instances.

"These kinds of statistics have to cause some wonderment," he said. "They are based on assertions about the probability of events whose probability we don't know."

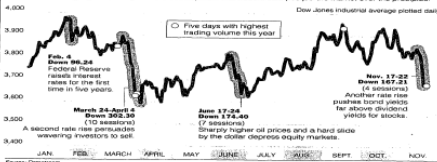
At the Jet Propulsion Laboratory in Pasadena, Calif., one satellite communications researcher who learned of the error this week, said six Pentium machines were used in his group and their use had been suspended for now.

"The Pentium appeared as a con-

Continued on Page D5

Four Sharp Corrections in the Dow

This year the stock market has tended to gain gradually for weeks or bob nervously around a trading range with little direction, waiting for the moment when some bad news or unfavorable trend sets off a headlong flight. Then, once prices have been beaten down for a few days, the buyers reappear and the bleeding stops. Four of those corrections are highlighted, along with the development that traders said pushed the market over the precipice.

Gibson Suit
On Trades
Is SettledBankers Trust Gets
30% of Debt Claimed

By MICHAEL QUINN

Gibson Greetings Inc. and the Bankers Trust Company said yesterday that they had settled Gibson's lawsuit accusing the bank of improperly leading the company to engage in risky financial trades.

Under the out-of-court settlement, Gibson will pay Bankers Trust nearly \$6.3 million, or about 39 percent of the \$20.7 million that Bankers Trust contended it was owed under the trades.

While neither side boasted of victory, Douglas Kidd, a spokesman for Bankers Trust, acknowledged that



Flaw Undermines Accuracy of Pentium Chips

By JOHN MARKOFF/Special to The New York Times

New York Times (1857-Current file); Nov 24, 1994; ProQuest Historical Newspapers The New York Times (1851 - 2002)
pg. D1

Flaw Undermines Accuracy of Pentium Chips

By JOHN MARKOFF

Special to The New York Times

SAN FRANCISCO, Nov. 23 — An elusive circuitry error is causing a chip used in millions of computers to generate inaccurate results in certain rare cases, heightening anxiety among many scientists and engineers who rely on their machines for precise calculations.

The flaw, an error in division, has been found in the Pentium, the current top microprocessor of the Intel Corporation, the world's largest chip maker. The chip, in several different configurations, is used in many com-

puters sold for home and business use, including those made by I.B.M., Compaq, Dell, Gateway 2000 and others.

The flaw appears in all Pentium chips now on the market, in certain types of division problems involving more than five significant digits, a mathematical term that can include numbers before and after a decimal point.

Intel declined to say how many Pentium chips it made or sold, but Dataquest, a market research company in San Jose, Calif., estimated that in 1994 Intel would sell 5.5 million to 6 million Pentiums, roughly 10 percent of the number of personal

computers sold worldwide.

Intel said yesterday that it did not believe the chip needed to be recalled, asserting that the typical user would have but one chance in more than nine billion of encountering an inaccurate result as a consequence of the error, and thus there was no noticeable consequence to users of business or home computers. Indeed, the company said it was continuing to send computer makers Pentium chips built before the problem was detected.

William Kahan of the University of California at Berkeley, one of the nation's experts on computer mathematics, expressed skepticism about

Intel's contentions that the error would only occur in extremely rare instances.

"These kinds of statistics have to cause some wonderment," he said. "They are based on assertions about the probability of events whose probability we don't know."

At the Jet Propulsion Laboratory in Pasadena, Calif., one satellite communications researcher who learned of the error this week, said six Pentium machines were used in his group and their use had been suspended for now.

"The Pentium appeared as a cost-

Continued on Page D5

- A small error was discovered in the floating point arithmetic
- Intel's response: don't worry about it



tium Chips

In some complex division problems, annoying errors.

corrected.

Some computer users said they believed that Intel had not acted quickly enough after discovering the error.

"Intel has known about this since the summer; why didn't they tell anyone?" said Andrew Schulman, the author of a series of technical books on PC's. "It's a hot issue, and I don't think they've handled this well."

The company said that after it discovered the problem this summer, it ran months of simulations of different applications, with the help of outside experts, to determine whether the problem was serious.

The Pentium error occurs in a portion of the chip known as the floating point unit, which is used for extremely precise computations. In rare cases, the error shows up in the result of a division operation.

Intel said the error occurred because of an omission in the translation of a formula into computer

Close, but Not Close Enough

The owners of computers that use Intel's Pentium microprocessors have found that the chips sometimes do not perform division calculations accurately enough.

The problems arise when the chip has to round a number in a preliminary calculation to get the final result, a task that all processors normally perform. In these cases, however, the Pentium's figures are exact to only 5 digits, not 16, as are those of other computer processors. The Pentium's error, while small, can be 10 billion times as large as those of most chips.

Here is an example of the way the imprecise rounding changes the results of a calculation and the way the deviation from the expected result is calculated.

PROBLEM

$$4,195,835 - [(4,195,835 / 3,145,727) \times 3,145,727]$$

CORRECT CALCULATION

$$= 4,195,835 - [(1.3338204) \times 3,145,727] = 0$$

PENTIUM'S CALCULATION

$$= 4,195,835 - [(1.3337391) \times 3,145,727] = 256$$

DEVIATION

$$256 \div 4,195,835 = 6.1 \times 10^{-5}, \text{ or } 61/100,000$$

Source: Cleve Moler, the Mathworks Inc.

Intel Timeline

from emery.com

June 1994 Intel engineers discover the division error. Managers decide the error will not impact many people. Keep the issue internal.

June 1994 Dr Nicely at Lynchburg College notices computation problems

Oct 19, 1994 After months of testing, Nicely confirms that other errors are not the cause. The problem is in the Intel Processor.

Oct 24, 1994 Nicely contacts Intel. Intel duplicates error.

Oct 30, 1994 After no action from Intel, Nicely sends an email



Intel Timeline

from emery.com

FROM: Dr. Thomas R. Nicely
Professor of Mathematics
Lynchburg College
1501 Lakeside Drive
Lynchburg, Virginia 24501-3199

Phone: 804-522-8374
Fax: 804-522-8499
Internet: nicely@acavax.lyncburg.edu

TO: Whom it may concern

RE: Bug in the Pentium FPU

DATE: 30 October 1994

It appears that there is a bug in the floating point unit (numeric coprocessor) of many, and perhaps all, Pentium processors.

In short, the Pentium FPU is returning erroneous values for certain division operations. For example,

$0001/824633702441.0$

is calculated incorrectly (all digits beyond the eighth significant digit are in error). This can be verified in compiled code, an ordinary spreadsheet such as Quattro Pro or Excel, or even the Windows calculator (use the scientific mode), by computing

$00(824633702441.0)*(1/824633702441.0)$,

which should equal 1 exactly (within some extremely small rounding error; in general, coprocessor results should contain 16 significant decimal digits). However, the Pentiums tested return

0000.999999996274709702

.
. .
.



Intel Timeline

from emery.com

- Nov 1, 1994 Software company Phar Lap Software receives Nicely's email. Sends to colleagues at Microsoft, Borland, Watcom, etc. decide the error will not impact many people. Keep the issue internal.
- Nov 2, 1994 Email with description goes global.
- Nov 15, 1994 USC reverse-engineers the chip to expose the problem. Intel still denies a problem. Stock falls.
- Nov 22, 1994 CNN Moneyline interviews Intel. Says the problem is minor.
- Nov 23, 1994 The MathWorks develops a fix.
- Nov 24, 1994 New York Times story. Intel still sending out flawed chips. Will replace chips only if it caused a problem in an important application.
- Dec 12, 1994 IBM halts shipment of Pentium based PCs
- Dec 16, 1994 Intel stock falls again.
- Dec 19, 1994 More reports in the NYT: lawsuits, etc.
- Dec 20, 1994 Intel admits. Sets aside \$420 million to fix.



Rounding Error: A Critical Concern

How rounding errors escalate:

- 1 Initial calculations introduce minor errors.
- 2 Errors are compounded by multiple floating-point operations.
- 3 Computer program outputs a **significantly inaccurate result**.
- 4 Scientists unknowingly **rely on the flawed result**.
- 5 Disaster.

Topics to explore:

- Error propagation: Understanding how errors spread.
- Ill-conditioned problems: When small changes cause large effects.

Key Idea

Rounding error can lead to large errors if not handled properly. Scientists and engineers rely on numerical computations for applications.

Our Mission as Numerical Analysts

Our objectives:

- 1 Develop solutions that are *resilient* to large roundoff errors.
- 2 Achieve results with *tolerable* truncation errors.

How do we accomplish this?

- Integrate knowledge of roundoff and truncation errors into:
 - Mathematical modeling
 - Computational methods
 - Algorithm design
 - Software development
- Promote *awareness* for accurate result interpretation.



Yes, you have to know floats

