

**Proseminar : Software Desaster und wie  
man sie verhindern kann**

**Desaster : The Pentium FDIIV Bug**

**Yun-Yi Lisa Wang**

**wangyu@in.tum.de**

**09. November 2002**

# Inhaltsverzeichnis

|  |           |
|--|-----------|
| <b>1 GESCHICHTE – WIE ES ENTDECKT WURDE.....</b>                                   | <b>3</b>  |
| 1.1 WIE IST DER BUG AN DIE ÖFFENTLICHKEIT GEKOMMEN? .....                          | 3         |
| 1.2 WIE REAGIERTE INTEL? .....   | 4         |
| 1.3 DIE PRESSE .....   | 4         |
| <b>2 UNGLÜCK – WAS GENAU SCHIEF GING .....</b>                                     | <b>4</b>  |
| 2.1 DIE EIGENSCHAFTEN DES BUGS.....  | 5         |
| 2.2 WAHRSCHEINLICHKEIT .....   | 5         |
| 2.3 BEISPIEL.....  | 5         |
| 2.4 SELBER TESTEN .....  | 5         |
| 2.5 WAS NOCH?.....   | 6         |
| <b>3 URSACHE – WIE ES DAZU KOMMEN KONNTE.....</b>                                  | <b>6</b>  |
| 3.1 HINTERGRUND –FPU (FLOATING-POINT UNIT).....                                    | 6         |
| 3.2 VERBESSERUNG VON 486DX CHIP .....  | 6         |
| 3.3 DIE URSACHENBESCHREIBUNG.....  | 6         |
| <b>4 SRT ALGORITHMUS.....</b>  | <b>7</b>  |
| 4.1 RESTORING DIVISION .....   | 7         |
| 4.1.1. <i>Dezimales Beispiel</i> .....   | 8         |
| 4.2 SRT DIVISION .....   | 8         |
| 4.2.2. <i>Dezimales Beispiel</i> .....   | 9         |
| 4.3 SRT ALGORITHMUS – FORMALER AUSDEÜCK .....                                      | 10        |
| <b>5 UNTERNEHMENS POLITIK – WIE BEI INTEL DAS PROBLEM BEHANDELT<br/>WURDE.....</b> | <b>10</b> |
| 5.4 INTEL POLICY.....  | 10        |

|          |   |           |
|----------|---|-----------|
| 5.5      | INTEL BEHAUPTETE.....                               | 10        |
| 5.6      | IBM BEHAUPTETE.....                                 | 11        |
| 5.7      | DER TEST VON COMPUSERVE .....                       | 11        |
| 5.8      | DER MARKETING KRIEG .....                           | 11        |
| <b>6</b> | <b>SCHADEN – WAS, WIE VIEL WURDE VERLOREN?.....</b> | <b>12</b> |
| 6.1      | HAT INTEL ES GETESTET?.....                         | 12        |
| 6.2      | SCHADEN .....                                       | 12        |
| <b>7</b> | <b>REFERENZ .....</b>                               | <b>12</b> |

# 1 Geschichte – wie es entdeckt wurde

## 1.1 Wie ist der Bug an die Öffentlichkeit gekommen?

Dr. Thomas R. Nicely ist ein Mathematik Professor an der Lynchburg College in Virginia. In Juli 1994 berechnete er die Summe der reziproken Werte von großen Menge Primzahlen auf einen Pentium Computer. Der Wert, den er bekam, war anders als der theoretische Wert. Er bekam mal den richtigen Wert auf einem 486 CPU. Am Ende hat er heraus gefunden, daß das Problem an Pentium selbst lag.

Nachdem Intel seine Frage nicht richtig beantwortete und er sein Resultat überprüft hat, hat Herr Nicely seine Entdeckung in die USENET Newsgruppe : comp.sys.intel folgende Mail gestellt und nach Rückmeldungen gefragt.

**FROM:** Dr. Thomas R. Nicely, Professor of Mathematics,  
Lynchburg College, Lynchburg, Virginia ,30 October 1994  
**TO:** Whom it may concern **RE:** Bug in the  
Pentium FPU

**It appears that there is a bug in the floating point unit (numeric co-processor) of many, and perhaps all, Pentium processors. In short, the Pentium FPU is returning erroneous values for certain division operations. For example,**

**1/824633702441.0**

**is calculated incorrectly (all digits beyond the eighth significant digit are in error). .... even the Windows calculator (use the scientific mode), by computing**

**(824633702441.0)\*(1/824633702441.0),**

**which should equal 1 exactly (within some extremely small rounding error; in general, coprocessor results should contain 19 significant decimal digits).**

**However, the Pentiums tested return**

**0.999999996274709702**

**for this calculation.**

**.....The bug has not been observed on any 486 or earlier system, even those with a PCI bus. If the FPU is locked out (not always possible), the error disappears; but then the Pentium becomes a “586SX”, and floating point must run in emulation, slowing down computations by a factor of roughly ten. I encountered erroneous results which were related to this bug ... I had eliminated all other likely sources of error (software logic, compiler, chipset, etc.). ...the bug was observed on a 66-MHz system at Intel, but there was no further information or explanation, other than the fact that no such bug had been previously reported or observed.**

## 1.2 Wie reagierte Intel?

Die Nachrichten führte zu einer großen Diskussion in der Newsgruppe comp.sys.intel. Es gab schnell auch viele negative Stimmung gegen Intel. Deswegen kontaktierte Intel Herr Nicely, um eine geheime Vereinbarung abzuschließen. Herr Nicely nahm diese an, und stellte dafür nur noch eine letzte Nachrichten in die comp.sys.intel Newsgruppe.

Ein paar Monaten später trifft Dr. Andrew Grove, CEO bei Intel, eine Entscheidung, entschuldigte sich und gab zu, dass es tatsächlich um ein Bug handelte. Und beauftragte die Intel Entwicklung das Problem zu lösen.

Diese Nachrichten verursachte einen PR (Public Relationship) Alptraum. Industrierberater meinten das dies ein großer PR Fehler war, der vermieden hätte werden können wenn Intel rechtzeitig eine öffentliche Informationskonferenz abhalten hätte.

## 1.3 Die Presse

Anfang Dezember entdeckte die Presse diese Geschichte. Dann bekam Dr. Nicely von verschiedenen Magazinen Interviews Angeboten. Die Titel der ersten Seite war „Dishonest bussines practices and legal action abounded“. Es gab auch ein Interview mit CNN. Seit dem wurde diese Thema sehr hoch gehandelt in der Presse.

## 2 Unglück – was genau schief ging

Es ist ein Software Bug, der in der Hardware ‚encoded‘ wurde. Im CPU (RISC von ALU) befand sich der sogenannte FDIV-Bug. Es ist die Abkürzung für Floating point DIVision basierte Instruktionen. Der Pentium FDIV Bug ist ein sehr bekannter Intel Bug. Der Bug wurde durch einen Fehler im ‚lookup table‘ des SRT Algorithmus von Intel verursacht. Es resultiert eine falsche geringere Ausgabe bei der Division. Die Eigenschaften des Bug werden im nächsten Abschnitt aufgezeigt.

Der Bug muss nicht schlimm sein. Es kommt darauf an, welche Toleranz ein Benutzer noch akzeptiert.

## 2.1 Die Eigenschaften des Bugs

- Das ist ein Software Bug.
- Es ist nur mit bestimmten Eingaben möglich.
- Der Fehler taucht meistens zwischen dem 5ten und 14ten Bit auf.
- Wegen dem Bug kann es Ungenauigkeiten bei den Datentypen ‚single‘, ‚double‘, und ‚extended‘ geben.

## 2.2 Wahrscheinlichkeit

Wie oft kann der Fehler einmal vorkommen? Bei den meisten Fällen ist die Wahrscheinlichkeit, das der Bug in der 9-ten oder 10-ten Dezimalstelle vorkommt, eins zu 9 Billionen. Der schlimmste Fall, das der Bug in der 4-ten Dezimalstelle vorkommt, hat die Wahrscheinlichkeit eins von 360 billionen Mal.

## 2.3 Beispiel

Ein sehr berühmtes Beispiel ist  $4195835/3145727$ , das von Tim Coe von Vitesse Semiconductors entdeckt wurde. Das Resultat von Pentium's floating-point war 1,333739, während der korrekte Wert 1,333822045 ist (bis zur 6. Nachkommastelle). Damit hat man einen relativen Fehler von 0,006% .

## 2.4 Selber testen

Wenn man zuhause noch einen Pentium hat, kann man auch einfach mit Microsoft's Windows den Pentium testen :

- Unter Pentium Windows Taschenrechner aufrufen.
- Mit der im Beispiel genannten Zahl dividieren.
- Mit den oben genannten Resultaten vergleichen.

Warum Taschenrechner? Denn viele Software Packages benutzen auch floating-point Zahlen, aber nicht wirklich die FPU des Computers. Darum zeigen diese ‚Packages‘ den Fehler nicht an.

## 2.5 Was noch?

Die Instruktionen, die die „Lookup table“ verwendeten, hätten auch den Bug betroffen. Im CPU befand sich ca. 200 Instruktionen. FDIV war nur eine davon. Es gab noch viele Instruktionen, die wegen dem Bug falsche Ausgaben ergaben, z.B. FDIVP, FDIVR, FDIVRP, FIDIV, FIDIVR, FPREM und FPREM1.

## 3 Ursache – wie es dazu kommen konnte

### 3.1 Hintergrund –FPU (floating-point unit)

Hier soll nicht sehr viel über FPU selbst erzählt werden, sondern über die Geschichte von FPU. Vor dem 486DX-Chip gab es noch kein FPU. Er führte arithmetische Rechenschritte mit Integer aus. Seit dem 486DX-Chip gibt es FPU als ein mathematischer Coprocessor.

### 3.2 Verbesserung von 486DX Chip

Der traditionale ‚486 shift-and-subtrakt‘ Algorithmus kann nur ein Quotient\_Bit pro Zyklus generieren. Mit Radix 4 SRT Algorithmus kann Pentium 2 Quotient-Bits pro Zyklus generieren. Durch eine Kombination von Compiler und Hardware-Technologie lässt sich die Rechengeschwindigkeit wesentlich verbessern. Und im Gegensatz zu 486-Chip konnte der neue Pentium 3- bis 5-mal schneller rechnen**Die**

## Ursachenbeschreibung

Der SRT Algorithmus benutzt ein ‚lookup table‘ für die floating-point Division, um einen zwischenzeitlich benötigten Quotient zu berechnen. Die Intel ‚lookup table‘ besteht aus einer Matrix von 2048 Einträge. Aber nur 1066 Einträge davon besitzen die der folgenden Werte: [-2, -1, 0, +1, +2]. Der Alorithmus benutzt die Bit Muster von dem Nenner als Index in der Tabelle. Bis hier ist alles in Ordnung. Das Problem war

das Programm das der Entwickler in C schrieb, um die „Lookup table“ die sie vorbereiteten, zur Einbeziehung des Pentiums FPUs(floating point unit) ins PLA(programmable logic array) zu laden. Da 5 Einträge von 1066 nicht ins PLA geladen wurden, gab es den Bug verursachenden Fehler. Wenn FUP auf irgendeine von den fünf zugreift, würde FUP 0 anstatt +2 bekommen. Deswegen würde es eine falsche geringere Zahl geben.

Weil der SRT Algorithmus rekursiv ist, während die Division-Operation ausgeführt wird, kann sich die falsche geringere Zahl akkumulieren. Im schlimmsten Fall kann der Fehler in der 4ten Stelle der dezimalen Zahl vorkommen. Wichtig!! Die Stelle ist nicht die Nachkommastelle. Das dezimale Komma kann an jede Stelle auftauchen.

## 4 SRT Algorithmus

SRT Algorithmus wurde am Ende von 1950er Jahre von Sweeney, Robertson und Tocher entwickelt. Auch deswegen wird der Algorithmus so genannt. Der SRT Algorithmus ist eine Art von „non-restoring division“. Es gibt SRT Algorithmen mit verschiedenen Basis, z.B. Radix 2 SRT Algorithmus und Radix 4 SRT Algorithmus(den Pentium verwendet). Hier wird zuerst von der „restoring division“, dann von der „SRT division“,wo der Fehler auftritt, gesprochen.

### 4.1 Restoring Division

1. Ein Zähler und ein Nenner werden genommen.
2. Der Wert der Quotientstelle wird geschätzt
3. Mit der Schätzung multipliziert sich der Nenner, und es wird von Zähler abgezogen.
4. Falls das Ergebnis negativ ist, muss es zurück addiert werden. Goto : 2.,um einen richtige Wert zu schätzen.
5. Es wird am letzten Bit des Quotientregisters abgespeichert.
6. Rest und Quotientenregister werden 1 mal nach links geschoben.

7. Neue Zähler wird erzeugt.
8. Goto : 2. wenn weitere Quotientenstelle noch angefordert wird.

### 4.1.1. Dezimales Beispiel

**Zähler : Nenner**

2633 : 8 = 0 3 2 9

0000

2633

2400

233

240

-7

+240

restoring

233

160

73

72

1

**Remainder**

**Die Quotient ist :  $0 * 10^3 + 3 * 10^2 + 2 * 10^1 + 9 * 0^0$**

## 4.2 SRT Division

1. Ein Zähler und ein Nenner werden gerundet genommen.
2. Mit dem Zähler und dem Nenner in den ‚lookup table‘ hinein schauen, um den Wert der weitere Quotientenstelle zu schätzen (bei Radix 4 kann der Wert nur -2, -1, 0, +1, + 2 sein)
3. Mit der Schätzung multipliziert sich der Nenner, und es wird von Zähler abgezogen. (Die Operation ist Addiren, falls der Schätzungsquotient negativ ist.)

4. Es wird am letzten Bit des Quotientregisters abgespeichert.
5. Rest und Quotientenregister werden 2 mal nach links geschoben.
6. Neue Zähler wird erzeugt.
7. Goto : 2. wenn weitere Quotientenstelle noch angefordert wird.
8. Der Wert des Quotienten wird von dem Quotientenregister berechnet.
9. Falls der letzte partielle Rest negativ war, dann substrahiert man seinen Wert von dem Quotienten, um korrektes Ergebnis zu erhalten.

#### 4.2.2. Dezimales Beispiel

##### Zähler : Nenner

$$7213 : 4 = 2 \text{ (-2) } 1 \text{ (-7) } = 1803$$

8000

-787

800

13

40

-27

28

1

**Remainder**

$$\text{Die Quotient ist : } 2 * 10^3 + (-2) * 10^2 + 1 * 10^1 + (-7) * 0^0$$

$$= 1 * 10^3 + 8 * 10^2 + 0 * 10^1 + 3 * 0^0$$

### 4.3 SRT Algorithmus – formaler Ausdeück

$d := \text{Nenner}$      $r := \text{Basis}$

$p_0 := \text{Zähler}/r$

for  $i = 0$  to  $(n-1)$     Lese ein  $q_i \in \mathbb{N} \mid \{-(r/2), \dots, r/2\}$  von „Lookup  
table“ aus

$p_{i+1} := r * p_i - q_{i+1} * d$

end    for  $i = 0$  to  $(n-1)$

$Q_{i+1} := r * Q_i + q_{i+1}$     end

## 5 Unternehmens Politik – wie bei Intel das Problem behandelt wurde

### 5.4 Intel Policy

Am 28. November 1994 wurde der Bug von Intel zugegeben. Intel behauptete, dass das Problem nur einen kleinen Fehler ist. Und Intel eretzte den Pentium Chip nur, wenn man dies mit der Notwendigkeit einer hohen Genauigkeit in den komplexen Kalkulationen begründen könnte. Es gab danach sofort viele Beschwerden aus der Öffentlichkeit. Dann am 20. Dezember 1994 bot Intel einen kostenlosen Austausch an. Man konnte unter der Telefonnummer 1-800-628-8686 eine detaillierte Information bekommen.

### 5.5 Intel behauptete

Intel meinte, dass der Bug überhaupt nicht sehr oft vorkommt. Er besaß nur eine Wahrscheinlichkeit von fast eins zu 9 Billionen der Zufalls-floating-point-Zahlen Division erzeugt. Und ein Spreadsheet Benutzer wird den Fehler durchschnittlich einmal in 27000 Jhre sehen können. Es wurden 1738 Paare mit schlechten Eingaben gefunden.

Ein normaler Benutzer braucht meistens die Zahlen mit der Rundung bis Integer und/oder bis zu 2ten Nachkommastelle. Nur 20% der Benutzer braucht mehr als 2 Nachkommastellen. D.h. auch wenn dort Fehler vorkommen können, werden sie durch die Rundungsfunktion bedeutungslos.

| <b>Das Display von den Auswertungen</b> | <b>Die Perzentzahl von den User</b> |
|---|-------------------------------------|
| bis 2 Nachkommastelle                   | 40%                                 |
| Interger                                | 40%                                 |
| über 2 Nachkommastelle                  | 20%                                 |

## 5.6 IBM behauptete

Normale Spreadsheet-Programme führt man 15 Minuten am Tag aus, somit könnte der Fehler auf Pentium einmal in 24 Tage vorkommen. Wenn es solche User 500-mal geben würde, könnte der Fehler auf Pentium 20 mal pro Tag vorkommen. Das heißt, dies ist viel häufiger als was Intel behauptete.

## 5.7 Der Test von Compuserve

Am 16. Dezember 1994 wurde das Resultat vom Test des Pentiums veröffentlicht. Es war 200 mal geringer als was IBM behauptete, und 200000 mal mehr als was Intel behauptete.

## 5.8 Der Marketing Krieg

Am 12. Dezember 1994 sagte IBM, dass sie Pentium nicht mehr in ihrem System verwenden werden. Aber keine andere Firma folgte IBM, z.B. Compaq, Dell, Gateway 2000 usw..

Ken Byers, kaufmännischer Manager bei Vow Unique Computer meinte, „IBMs Lieferungsstop für Pentium-basierte Systeme ist eine Überreaktion wegen unbedeutender Technischer Probleme“

Linley Gwennap, Editor Chef von Microprocessor Report meinte auch, „Der Lieferungsstop von IBM ist mehr eine Marketing Strategie gegen Intel und damit für IBM Power PC Prozessoren“

## 6 Schaden – was, wie viel wurde verloren?

### 6.1 Hat Intel es getestet?

Intel gab fast eine halb Milliarde Dollar seit 1991 für 2 Jahre aus, zur weiteren Pentium-Entwicklung und zum Testen. Dann wurden Pentiums am 22. März 1993 freigegeben. Eigentlich unterstützten viele andere Hersteller z.B. IBM oder Compaq, beim testen des Pentiums.

### 6.2 Schaden

Intel berichtete das durch Image-Schäden, Ersatzansprüche usw. 475,000,000\$ Kosten für die Firma entstanden sind.

## 7 Referenz

[1] Intel Pentium Processors, „Statistical Analysis of Floating Point Flaw Intel White Paper Section 6 Analysis of Impact on Applications”

wysiwyg://65/<http://www.intel.com/support/processors/pentium/fdiv/wp/6.htm>

[2] Intel Pentium Processors, „Statistical Analysis of Floating Point Flaw Intel White Paper Section 4 Pentium® Processor Divide Algorithm”

wysiwyg://111/<http://www.intel.com/support/processors/pentium/fdiv/wp/4.htm>

[3] „Radix-r Digit Recurrence Algorithms”

<http://www.aifb.uni-karlsruhe.de/Lehrabgebot/Winter2001-02/HardSoft/reports/project4/algorhythm.html>

[4] Roy Chartier, „Te Pentium Predicament“,

<http://www.monitor.ca/monitor/issues/vol2iss6/feature2.html>

[5] Robert Collins von Dr. Dobb's Microprocessor Resources, “P54C Erratum 23 – Slight Precision Loss for Floating-point Divides on Specific Perand Pairs”

<http://x86.ddj.com/errata/feb97/Bugs.htm>

[6] Larry Hoyle, “The Pentium FDIV Bug – a Picture”, 30, Nov 1994

[http://www.ku.edu/cwis/units/IPPBR/pentium\\_fdiv/pentgrph.html](http://www.ku.edu/cwis/units/IPPBR/pentium_fdiv/pentgrph.html)

[7] Tom R. Halfhill, “The Truth Behind the Pentium Bug”, Mär 1995

<http://www.byte.com/art/9503/sec13/art1.htm>

[8] Mark Janeba, „The Pentium Problem“, 1995

<http://www.willamette.edu/~mjaneba/pentprob.html>