

# SRT Division and the Pentium FDIV Bug (draft lecture notes, CSCI P415)

Steven D. Johnson

September 20, 2000

## Abstract

This talk explains the widely publicized design error in a 1994 issue of the Intel Corp. *Pentium* microprocessor. I present the *SRT division* algorithm on which the hardware is based. Then I show where the error occurs, and why it wasn't caught in verification. Finally, I speculate about whether the source of the bug was a design error or a process error.

For a sequence,  $\{q_i\}_{[n]}$  of digits, let  $\llbracket Q \rrbracket_m$  denote

$$\sum_{k=0}^{m-1} \frac{q_k}{10^k}$$

We abuse notation above by interpreting digit  $q_k$  as a number, and will omit the subscript  $m$  unless the context demands it.

Here is an algorithm for *long division*. Dividend  $P$  and divisor  $D$  have been normalized to be less than 10. The partial values  $\{p_i\}_{[n]}$  are there only for discussion purposes.

```
{1 ≤ P, D < 10}
p0, d, i := P, D, 0;
while i < n do {INV1 ∧ INV2}
  b
  choose qi ∈ {0...9} such that...
  {pi - qi · d < d}
  pi+1, i = 10(pi - qi · d), i + 1
  e
  { P - ⌊Q⌋ · d <  $\frac{d}{10^n}$  }
```

Hence, on termination

$$P/D - \llbracket Q \rrbracket_n \cdot D < \frac{D}{10^{-n}}$$

To put this in the usual way, the remainder (difference between computed and actual quotients) is “smaller than”  $D$ . Note that

$$\begin{aligned}
 p_3 &= 10(p_2 - q_2d) \\
 &= 10(10(p_1 - q_1d) - q_2d) \\
 &= 10(10(10(p_0 - q_0d) - q_1d) - q_2d) \\
 &= 10^3p_0 - 10^3q_0d - 10^2q_1d - 10^1q_2d \\
 &= 10^3P - \left( \sum_{k=0}^2 10^{3-k} q_k d \right) \cdot d \\
 &= 10^3P - \llbracket Q \rrbracket_3 \cdot d
 \end{aligned}$$

Thus, the loop invariants are,

$$\text{INV1} \equiv p_i = 10^i P - \llbracket Q \rrbracket_i \cdot d$$

$$\text{INV2} \equiv p_i < 10^i \cdot D$$

None of the arguments above are affected by the actual value of the radix (base). 10 can be replaced by any  $b$  and the program is still correct. In computer division the radix is a power of two; in the Pentium series, the radix is 4.

0 0 4 0 2 R 18

```
-----  
53 ) 2 1 3 2 4  
    0 - - - -  
-----  
    2 1 - - -  
-   0 - - -  
-----  
    2 1 3 - -  
-  2 1 2 - -  
-----  
      1 2 -  
-    0 - -  
-----  
      1 2 4  
-    1 0 6  
-----  
      1 8
```

1 0  
0 0 3 0 2 R 18

```
-----  
53 ) 2 1 3 2 4  
    0 - - - -  
-----  
    2 1 - - -  
-   0 - - -  
-----  
    2 1 3 - -  
-  1 5 9 - -  
-----  
      5 4 2 -  
-    5 3 0 -  
-----  
      1 2 4  
-    1 0 6  
-----  
      1 8
```

$$\begin{array}{r}
 - 0 0 0 9 7 \\
 + 0 0 5 0 0 \text{ R } -35 \quad \{500 - 97 \text{ R}-35 \\
 \text{-----} \\
 \quad = 403 \text{ R}-35 \\
 53 ) 2 1 3 2 4 \quad = 402 \text{ R}+18\} \\
 \quad 2 1 3 \\
 \quad - 2 6 5 \\
 \text{-----} \\
 \quad - 5 1 8 \quad = -520 + 2 \\
 \quad - - 4 7 7 \\
 \text{-----} \\
 \quad - \quad 4 0 6 \quad = -410 + 4 \\
 \quad - - \quad 3 7 1 \\
 \text{-----} \\
 \quad - \quad \quad 3 5
 \end{array}$$

The examples illustrate that by maintaining two partial quotients, one of positive quotient digits and the other of negative ones, we can sometimes recover from an invalid guess for the next digit. It is important to understand that this refinement is *not* “error correcting.” If the selected digit is too remote from the correct one, there is no chance of recovering. Within those limits, however, we can perform *nonrestoring* long division: making a guess based on the leading digits of the dividend and divisor, which is close enough to be corrected. Hence, we do not have to backtrack if the guess is wrong.

SRT hardware exploits this in *two ways*. The first is the nonrestoring quality, just described. In radix 4, this would mean a choice of quotient digits from the set  $\{-3, -2, -1, 0, 1, 2, 3\}$ . However, the actual selection set used in hardware is  $\{-2, -1, 0, 1, 2\}$ . This reduces the multiplications to shifts. We always avoid selecting  $\pm 3$  as a quotient digit.

The geometric sum,

$$2 + \frac{2}{4} + \frac{2}{8} + \cdots + \frac{2}{4^i} + \cdots = \frac{8}{3}$$

Recall from (probably) your first homework assignment in mathematical induction that

$$a + ar + ar^2 + \cdots + ar^n = \frac{a(i - r^{n+1})}{1 - r}$$

In the algorithm, we need to keep  $p_{k+1}$  within the range  $[-\frac{8}{3}d, \frac{8}{3}d]$ . To summarize, our division algorithm now looks like

```

{1 ≤ P, D < 2}
p0, d, i := P, D, 0;
while i < n do {INV1 ∧ INV2}
  b
  choose qi ∈ {−2 . . . + 2} such that . . .
  { |4(pi − qi · d)| ≤  $\frac{8}{3}d$  }
  pi+1, i = 4(pi − qi · d), i + 1
  e
  { P − [Q] · d <  $\frac{d}{4^n}$  }

```

where

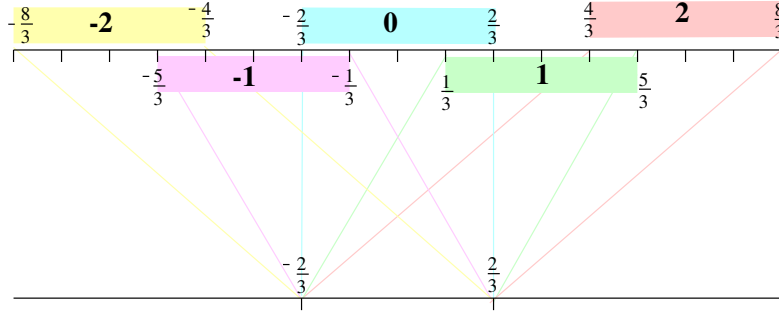
$$[Q]_m = \sum_{k=0}^{m-1} \frac{q_k}{4^k}$$

$$\text{INV1} \equiv p_i = 4^i P - [Q] \cdot d$$

$$\text{INV2} \equiv p_i < 4^i \cdot D$$

To maintain  $p_i$  within range, each quotient digit must map the interval  $[-\frac{8}{3}d, \frac{8}{3}d]$

into the interval  $[-\frac{2}{3}d, \frac{2}{3}d]$ . This is how we arrive at the staggered selections:

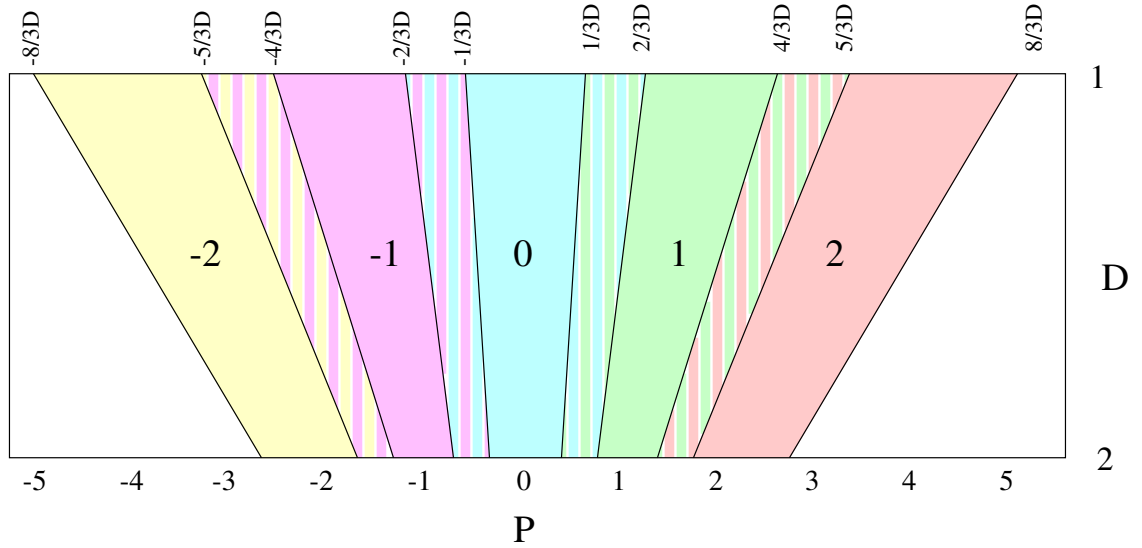


Overlapping ranges permit a choice of two possible quotient digits. At any point along this interval, if the wrong digit—a digit other than the one/two possibilities is selected, the algorithm cannot recover.

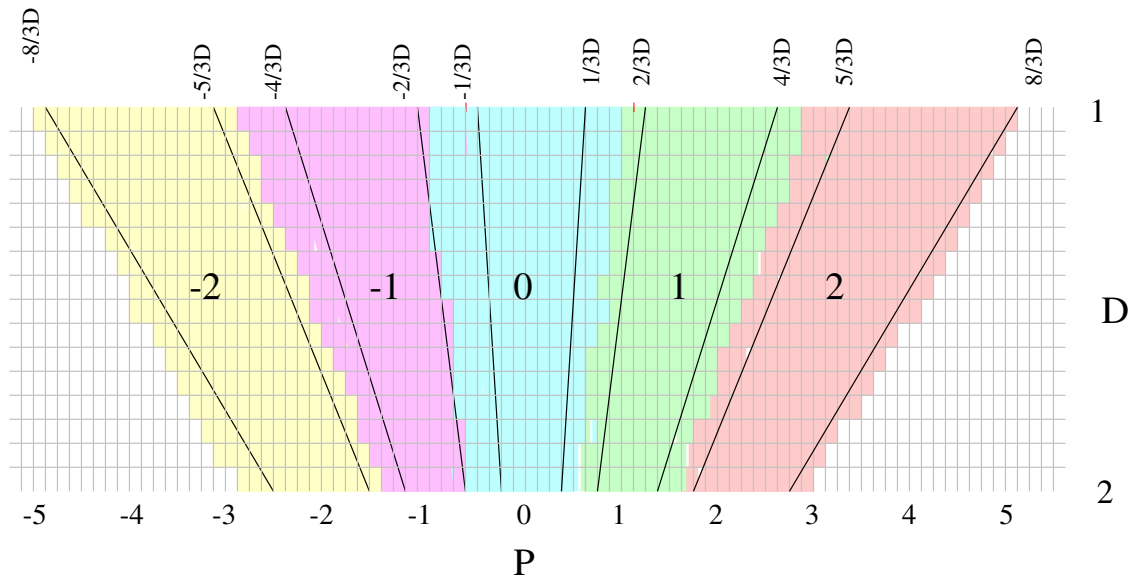
Now, just like people doing long division, the algorithm must “guess” the next quotient digit, based on the most significant digits of the dividend and divisor. This guessing is represented by a two dimensional look-up table, indexed by the 7 leading bits of  $p + i$ , including the sign bit, and the five leading bits of  $d$ .

Edleman cites Coe and Tang as stating that table entries may be determined by the following thresholds. In this table, we take  $P$  to be  $\frac{\lfloor 8p_k \rfloor}{8}$  and  $D$  to be  $\frac{\lfloor 16d \rfloor}{16}$ .

$q$	$P_{\min}$	$P_{\max}$
-2	$-\frac{8}{3}D$	$-\frac{1}{4} - \frac{4}{3}D$
-1	$-\frac{1}{8} - \frac{4}{3}D$	$\lfloor -\frac{1}{4} - \frac{4}{3}D \rfloor$
-0	$\lfloor -\frac{1}{8} - \frac{1}{3}D \rfloor$	$\lceil -\frac{1}{8} + \frac{1}{3}D \rceil$
-1	$\lceil -\frac{1}{3}D \rceil$	$-\frac{1}{8} + \frac{1}{3}D$
-2	$\frac{4}{3}D$	$-\frac{1}{8} + \frac{4}{3}D$



The lookup table is a discrete version of this table, with boundaries lying within the overlapping regions (*inexact rendering!*):



There were five errors in the lookup table, all along the boundary between the 2 region and the thought-to-be-inaccessible region above it.

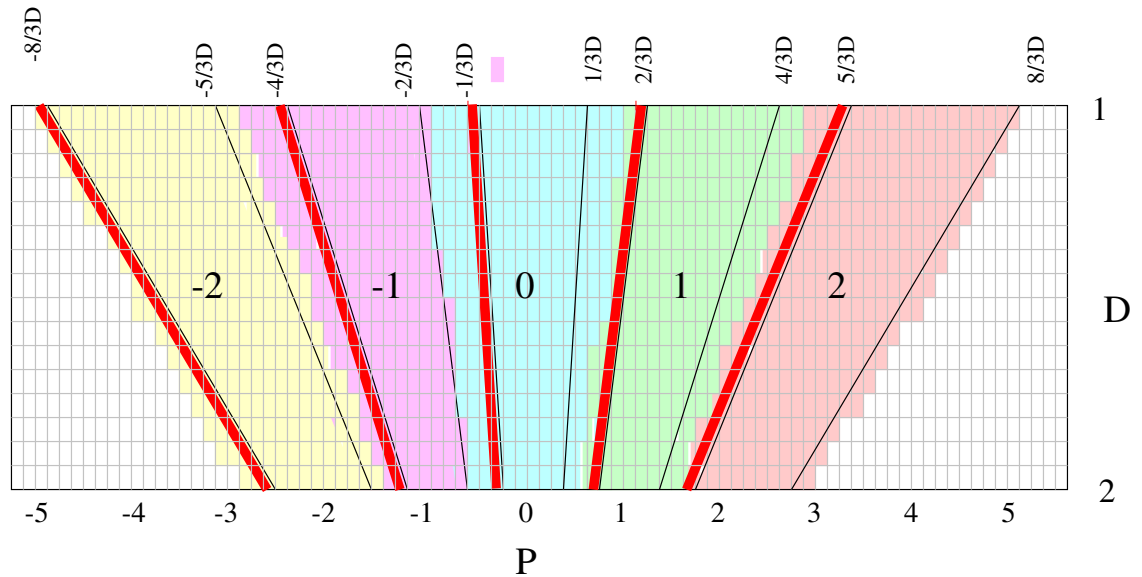
Actually, an adjustment must be made in the table to account for the fact that  $P$  is represented in *carry-save* form. That is, the value of  $P$  is maintained

in two registers,  $S$  and  $C$  under the invariant that  $P = S + C$ . To add  $D$  and  $P$ , the actual computation performs

$$S, C := (S \oplus C \oplus D), \text{shiftright}((S \wedge C) \vee (C \wedge D) \vee (D \wedge S))$$

bitwise in parallel. Subtraction is similar;  $D$  is complemented bit-wise, giving  $-D - 1$ , and the  $-1$  is compensated by shifting a 1 into the vacated 0-bit of  $C$ . In this way, carry propagation is eliminated until a single final addition is performed to recover  $P$ .

The carry-save representation weakens the bounds on the digit selection regions. It can be shown that this may be compensated by lowering all upper bounds by  $\frac{1}{8}$ . It would appear that this lowering exposed the five errors in the Pentium's lookup table.



There remains a question of why the bug wasn't discovered in testing. Here are some relevant points:

- The lookup table is *not* symmetric.
- Random dividend and divisor do not result in random table accesses. In fact table access is far from uniform (!?). In fact, the algorithm accesses a bad entry to produce a bad  $p_{k+1}$  only if  $p_k$  indexes the table just below it. Note that since  $D$  never changes, the algorithm is always operating on a single column of the table.
- THEOREM (Coe, Tang [Edelman]). *a flawed table entry can only be reached if the divisor has the form  $d = d_1 d_2 d_3 d_4 1111111 d_{11} d_{12} \dots$  and  $1.d_1 d_2 d_3 d_4$  addresses one of the buggy columns*

- [Edelman] the error is bounded by  $5e-5$  when dividing two numbers in the standard interval  $[1, 2)$ ; and, no matter what happens the first eight quotient digits will be correct.

## References

1. Pratt. Anatomy of the Pentium Bug.
2. Edelman, Mathematics of the Pentium division bug.
3. Sharangpani and Barton. Statistical analysis of floating point flaw in the Pentium processor (1994)
4. Clarke, German, and Zhao. Verifying the SRT division algorithm using theorem proving techniques.
5. Ruess, Shankar, and Srivas. Modular verification of SRT division.
6. Miner. Chapter six of *Hardware Verification using Coinductive Assertions* (PhD dissertation)