Time Dilation, Technical Dilation, and Corporate Dilation

TD Analysis Report

Copyright © 2000 Michael D. O'Connor
All Rights Reserved

RELEASE NOTICE

This is the finished release document. Any previous document that does not bear this notice is a draft unreleased copy, and therefore is not the finished document. This document released 15 May 2000

Table of Contents

Disclaimer

The author of this report makes no guarantee as to the accuracy of this document, and does not provide any warranties or conditions, expressed or implied, including without limitation, any warranty of merchantability or fitness for a particular purpose. The author shall not be liable for any damages resulting from the use of this report or the data herein.

This report is written solely by the author without the auspices of any company or entity.

Distribution Policy

Distribution of this report is authorized under the following conditions:

1. This report shall be distributed **free of charge** with nothing attached to it except as an attachment to email.
2. This report shall not be bundled, packaged, attached, or otherwise tied to or with any merchandise or product.
3. This report shall not be posted on web sites for display without prior consent of the author. The only exception to this are forums, BBS's, and FTP sites where this report may be posted as a downloadable item only.
4. This report shall not be distributed for profit, including distribution charges.
5. This report shall not be modified in any way, shape, or form, including the PDF file type of this document.

This report was generated using Microsoft® Word 97 and then converted into Acrobat PDF document using Adobe Acrobat™ 4.0.

Acknowledgements

First acknowledgement goes to Harlan Smith, whom I am very sorry to hear he has passed on before I had a chance to fulfill my promise to him that I will release this document soon.

Mr. Harlan Smith was the sole person responsible for getting this author into this mess regarding Time Dilation, as it was called at the time. Mr. Smith had strong reservations about the truth regarding TD – not from the author, but from the Crouch-Echlin group. This has remained true until the last day.

Second acknowledgement goes to Mr. Michael Kennedy, who in a span of one month kept the author's way of thinking in check with his own experience with BIOS chips and RTCs. He is a witness to the fact that Mr. Harlan Smith takes to the views of the author regarding TD, as well as those of Mr. Kennedy.

The acknowledgements also go to others who've worked with this author in the past, most of whom have passed on to other endeavors. Some of these were included in the semi-clandestine discussions via group email regarding the growing subject of the then-relatively-new discovery initially called Time Dilation (TD), which is now renamed "Time and Date Instabilities" but retaining the acronym "TD."

Regarding Mr. Harlan Smith, there had been differences regarding the effect of Y2K – Harlan on the seriousness of Y2K effect nationwide, even worldwide, and the author on how small the effect really is. The author still believes the effect will be localized rather than nationwide. It is agreed that the Y2K effects of most nations other than the USA are a serious threat to the USA counterparts, but the USA has the technology and expertise for remedial actions to correct the Y2K issues. The question, of course, is whether the available resources are sufficient.* This author helped him to see that the GPS satellites are not as subject to weather-induced Y2K problems as he initially feared (I am an engineer involved in satellite communications).

The author first predicted in 1997 that the Y2K issue would become a political issue. It did, but it is still not on any candidate's platform in the year 2000 presidential elections.

This report is released in Harlan Smith's memory. It is a good report ☺

* It is now obvious that the Y2K scare has mostly fizzled out at the time of this release.

1.0 Executive Summary

Time Dilation (TD) is a rare computer anomaly defined as an excessively advanced or retrogressed system time or date relative to the actual time or date. It is observable after a computer is turned on and booted. The anomaly can occur on PC systems from 286 to 486, and in some cases may occur on certain non-PCI Pentiums (Pentium Overdrives on 486 motherboards). According to the prevailing theory, the anomaly is more likely to occur on older systems than new ones, because processor speed generally affects anomaly occurrence across computer platforms,* and because some older BIOS firmware are not fully RTC-compliant.

This report shows conclusively that the TD anomaly is actually an "old" problem in terms of existence on computer platforms. The older the platform is today, the more prevalent the problem may be. In contrast, the newer the computer, the less probability the occurrence of the TD anomaly. In fact, there are indications that today's computers have already started to evolved away from the TD anomaly.

This report theorizes that the anomalies appear on these computer platforms *if the BIOS code is not RTC compliant*. Real Time Clock (RTC) chips with double-buffered registers never has exhibited a TD anomaly. New computers today may still have non-buffered RTC chips, but some have buffered RTC chips installed. The question, then, is which computer manufacturer installs buffered RTC chips? The question should be asked to a computer vendor before buying a computer.

This report does not discredit Time Dilation itself as a concept. It indeed exists, but it no more exists on computers than Windows 3.1 still exists on some computers, or no more than 386-based computers are still being used in small companies.

With this report, to bring the matter to a close is the ultimate purpose. While the *cause* of Time Dilation anomaly has never been proven by any corporation, person, or entity, this report does examine a singular, likely cause. It also examines and disproves certain claims and theories as to the cause of the TD anomaly.

* Although processor speed affects the RTC access time, the access time is not a primary factor for the TD anomaly occurrence. The update period occurs once per second, and the slower the processor speed, the longer the access time, thus increasing the odds of TD anomaly occurrence.

There are technical blunders in technical papers as well as incorrect, untenable, or impractical technical theories submitted by some parties. This report examines almost exclusively publicly available technical reports and theories written by Intel and Crouch/Echlin group.

In summary, this report concludes that:

1. The TD anomaly has only one symptom: an excessively advanced or retrogressed system time or date relative to the true time or date.
2. An advanced time or date anomaly is hard to disprove, while a retrogressed time or date anomaly may be hard to prove. There is no scenario or condition identified with causing a false case of retrogressed time/date anomaly that also causes a false case of advanced time/date anomaly.
3. Test data in Intel's public report, *Analysis of the Crouch-Echlin Effect*, shows strong evidence of advanced time anomaly in addition to the BIOS time conversion anomaly, which conversion anomaly is presented in their report in terms of a theory instead of using test data to support it. The advanced time anomaly is not covered in their report.
4. Echlin's "Logic Path" theory is conclusively disproved because the date conversion routine is performed after the date is retrieved from the RTC, not while the date is in the process of being retrieved.
5. Echlin's diagnostic utilities, including RTC.EXE (discontinued in late 1998), are conclusively shown to return erroneous timing information in the returned results on 286 computers because they run too slowly on these computers (mainly due to machine speed and the programming language used). This impacts 286 computers allegedly found susceptible to the TD anomaly, but actually may not have it.
6. The RTC chip itself is conclusively cleared as the culprit for the TD anomaly, because its time and date information is always correct.
7. During computer boot, the time and date information is separately processed: the time during POST, and the date after POST. Echlin/Crouch group suggests (on their web pages) both are processed together.
8. By theoretical analysis, the BIOS firmware code is the culprit for the TD anomaly, but there are many different versions of the code. Much depends on the integrity of the BIOS code. Not all BIOS code are suspect, however, because even in 1985 some BIOS code are robust.

This report covers three major areas: Time Dilation, because it is a misnomer as an accurate name for the anomaly, but nevertheless is more accurate than "Time and Date Instabilities" which implies more than one symptom; Technical Dilation, because there are flaws in technical presentations on both sides of this issue; Corporate Dilation, because some corporations are not meeting the evidence of the TD anomaly squarely in the face.

Some corporations who initially confirmed the TD anomaly have about-faced their foot-shifting stance on the anomaly because of alleged lawsuit fears from this TD "fallout," even when faced with the evidence. Other companies have thoroughly investigated the TD anomaly but without so much as even a single TD anomaly symptom, and the anomaly is chalked up as mere "hype and hysteria." Still others jumped on the bandwagon and sought to monopolize the fanned fears of this anomaly by providing the same software and/or hardware fixes offered for Y2K remediation.

Public exposure of this anomaly through web sites, some newspaper reports, and word-of-mouth eventually resulted in major corporations getting involved by testing for this anomaly. One U.S. corporation had found the presence on one computer and publicly confirmed it by means of newsgroup email, but eventually reversed itself after another U.S. corporation acquired it.

There are known bugs and conditions which are known to exhibit retrogressed time/date symptoms, but is a false TD anomaly. Verification of a retrogressed time or date symptom as a true TD anomaly requires a measure of technical knowledge on how the time and date information is processed in a computer boot. This is unfortunate but true.

However, reports of a TD anomaly showing an *advanced* time or date symptom is a much stronger indication of a true TD anomaly, especially an *excessive* advanced time/date symptom. There is no precedent for this symptom because there is no known bug or condition that causes it.*

* The time/date RTC initialization limitation is the only exception (known since at least 1987) that causes a one-day advance in the date, but this anomaly occurs only when *setting*, not reading, the RTC time/date 1 second before midnight on the last day of the month. This anomaly is revealed in the Hitachi technical data sheet for the Japanese version HD14681A RTC chip, and is *not* present in Motorola technical data sheet.

As far as the world knows, there is only one computer exhibiting an advanced date anomaly, and that is Mr. Jace Crouch's affectionately named "Zoom" 286 computer. However, this report reveals proof of advanced time anomaly on another machine – Mike Echlin's Zenith 286 board, which Intel tested. Intel claims the TD anomaly on this board was not found, but an advanced time anomaly on this computer cannot be denied or disproved in this report.

Yes, there were reports of other computers exhibiting TD anomaly, but accurate description of the anomaly itself (even to identify whether it is *advanced* or *retrogressed*) is sorely lacking. While the Crouch/Echlin group has changed the definition of TD at most three times since its conception, the general public and corporations still have no clear understanding of this definition to help them properly diagnose a TD anomaly. As a result, individual reports of the TD anomaly tend to be questionable if a detailed description of the observed anomaly is not included. The lack of proper definition of the TD anomaly itself tends to lead to false reports by people who do not understand it.

As far as embedded systems go, the TD anomaly may be more prevalent because 286 processors are used. However, its prevalence depends on a number of factors, including the presence of RTC chips, the system BIOS in use, whether system boots are regularly performed, and whether embedded systems are used in industries or in consumer products.

In conclusion, the TD anomaly is rare and will be even rarer as time goes on. TD is real enough, but not big enough.

2.0 Introduction

This document is intended to put to rest public apprehensions of a "fallout" from a computer anomaly first known as "Time Dilation" (TD), now renamed "Time and Date Instabilities" but retaining the acronym "TD." It is also called the "Crouch-Echlin Effect."

Randall Bart first coined the term "Time Dilation" at the time of the discovery. In Einsteinian physics, it refers to the phenomenon where an observer moving at high speed or in a strong gravity experiences time at a different rate than another observer in another frame of reference. At the time of the discovery, it was a reference to the RTC chip that is "experiencing time" at a different rate than our perception of time. However, the term is not an accurate representation of the anomaly, therefore it was renamed "Time and Date Instabilities" by Mr. Jace Crouch.

This document will show the following:

- Why TD anomaly has one symptom and one cause only
- Strong evidence of TD anomaly in Intel's test data in their report
- Mike Echlin's "Logic Path" theory debunked
- The difference between *advanced* and *retrogressed* time/date anomalies
- Real Time Clock (RTC) not the culprit for TD anomaly
- The importance of speed in reading RTC data on slow computers
- Characteristics of TD points to rare occurrence
- Most causative theories of TD debunked
- Typical computer boot process
- TD anomaly theory and culprit

3.0 Time Dilation Defined and Redefined, and Defined Again

On the web site hosted by Mr. Crouch is a 4-page document detailing Echlin's hypothetical theory of the TD anomaly. An excerpt states:

> *Certainly there are other causative factors behind the various time and date instabilities that are commonly known as the Crouch-Echlin Effect, including known BIOS problems, incompatible add-in cards or device drivers, and just plain old sloppy BIOS code. As Mike Echlin told me earlier this year, "what we have found is not a single problem, but a multitude of closely related problems that show similar symptoms."* [1]

Except for "sloppy BIOS code," the "causative factors" listed are not necessarily the cause of the anomaly; indeed, the cause has not even been found, much less conclusively proven, even though the anomaly have been "observed" by some Digital and Compaq employees. (They saw the mouse, but never saw its parents or the hole in the wall.*) The noted phrase in the above excerpt is "similar symptoms," which is in the plural sense. Only one symptom identifies the TD anomaly: an excessively advanced or retrogressed date or time. In fact, the investigation started out that way. The obvious attempt is to hoard up "other causative factors" as multiple causes for TD, which was not the case during the author's tenure. The fact at the time was that a process of elimination was used in an attempt to find the culprit.

The "multitude of closely related problems that show similar symptoms" – although not named – is likely not directly related to the TD anomaly. Note that the narrative calls it "*closely related problems*" – plural sense – and not specifically a TD anomaly. The phrase "*similar symptoms*" also indicates this. The narrative did not further elaborate on what these are. Neither did the narrative point out what is *not* the cause of the TD anomaly. They are:

- Dead or weak battery
- Cold solder joint
- Unsynchronized DOS time with RTC time†
- Known DOS midnight rollover bug
- Known time/date RTC initialization limitation
- BIOS interrupt 1Ah routines

---

* See end of Section 8.0.

† Normally not exceeding a difference of a few seconds per day or less

Except the last item above, any of the rest will result in a *similar* symptom (note, singular) pointing to a TD anomaly, but none of them are the *cause* of the anomaly. There is only one TD anomaly symptom: a substantial change in system time/date information forward or backward relative to the true date/time. This symptom is observable by examining the system date and time on a computer after a computer boot. A TD anomaly occurs only *during* a computer boot. The fact that it occurs only during a computer boot points to a *singular cause, because the same number of the same processes occur in the same sequence in every computer power-up*.*

This singular cause is the BIOS code that, during computer boot, does not properly avoid the small segment of time occurring every second, in which the Real Time Clock (RTC) is updating itself with the date and time. Two other causes are system interrupts not disabled and missing I/O delay instructions. But ignoring the UIP bit is the only cause that result in bad data being read from the RTC.†

But a cause of TD anomaly can be falsely identified by a misdiagnosed symptom. An example of this is a dying battery in the computer. There are other cases of similar symptoms that have nothing to do with TD. If there is a way to properly diagnose a TD anomaly, it is this: if the system date or time shows an *advanced* date or time relative to the true date or time, it is proof positive of a TD anomaly. On the other hand, if the system date or time has *retrogressed* instead of advanced, it is not proof of a TD anomaly because any number of causes apply which have nothing to do with TD. Hence, for a retrogressed date or time symptom, further investigation is warranted. Even so, it is more likely to be non-TD related.

The TD definition has changed or expanded at most three times since the discovery. The current definition, as defined by the Crouch-Echlin group, now includes *multiple* causes and symptoms of the anomaly, but without specifying what these are. But these causes, identified in this report, were well known before the discovery. These invariably result in retrogressed time/date information, but can never result in *advanced* time/date information. For this reason, these extraneous causes (referred to as "causative factors" on Jace Crouch's web site) are not true causes of the TD anomaly. By analysis and process of elimination, a singular cause of the TD anomaly remains the single and only cause.

---

* Of course, the boot process is different between a cold-boot and warm-boot.

† The RTC access time does *not* exceed the 244-microsecond limit, as this report will show later. This is an attribute on which the "Logic Path" theory depends, which is conclusively disproved in this report.

Based on detailed analysis in this report, the culprit appears very likely to be bad BIOS code that does not honor the Update-In-Progress bit (UIP) in status register A of the RTC. This has not been proven conclusively by any person, company, or entity, but it is the most logical culprit since it is already known that the BIOS code in *some* BIOS chips does not honor the UIP. However, computer speed is a factor that determines whether TD shows up in a computer, as will be explained later in this report.

Echlin's TDFIX was initially released without a full understanding of the underlying cause for the TD anomaly, nor, for that matter, a full understanding of RTC design parameters and operation. In fact, both TDTEST.EXE and TDFIX.EXE utilities left interrupts enabled, which may cause the wrong address to be pointed to, thus producing invalid results. Leaving interrupts enabled during access to the RTC could cause the RTC port address index to be changed by another interrupt (because system interrupts are enabled).[2]

Both TDTEST and TDFIX utilities have been fixed in a subsequent release, which turns off the interrupts during RTC access. However, Intel did not test the new versions.[3]

According to Intel, Intel ran the TDFIX utility (initial version), which reported that "it found the Crouch-Echlin Effect symptom" not just on Echlin's Zenith computer, but on other computers.[4] The discrepancy detected was the time differences between the RTC and BIOS data area in memory. In fact, Intel by implication calls "discrepancies between RTC data and BIOS data" the "Crouch-Echlin symptom." [5] How much this difference in time was is not revealed. Intel did not elaborate.*

The time difference in the RTC and BIOS data area is not the culprit for the TD anomaly, but *may* be a *symptom* of it. It is normal, however, for the time difference to be two seconds or so apart after a boot-up. It is also normal for this difference to widen slightly the longer the computer remains powered on, and this difference should be no more than a few seconds per day at the most. This normal characteristic is *not a case* of a TD anomaly.

Echlin's TDFIX provides a fix for the anomaly because the *problem* in itself, though not the *cause*, is well understood. What is not revealed is that this problem is not *widespread* as you may be led to believe. This is expounded later in this report.

* Intel did elaborate in Appendix A of its report, but their elaboration is only a cleverly concocted *theory* that bears some elements of truth, but also hides another truth. See Section 5.1.4 in this report.

Finally, the TD anomaly is *not* related to Y2K problems because:

1. Y2K issues deal solely with the 2-digit year date problem, a known problem for which there is a known solution.
2. The TD anomaly is not caused by the fact that the century byte has to be included in the calculations after Y2K; it is caused by poor BIOS code that does not honor the UIP.

4.0 TD Anomaly Probability Analysis

In Jace Crouch's web page, three "map of a second" figures are shown.[6] While the maps are correct in terms of sequence (A, B, C, & D), some areas are overlooked which may contribute to some exaggeration of the truth. That the figures are not to scale contributes to a part of this exaggeration.

One such exaggeration is the misleading statement that *both* "time and date calculations" are made in a single process, thus leading the reader to believe that both time and date are retrieved from the RTC in the same process. In truth, the time information is retrieved from the RTC during POST, and the date information is retrieved *later*, after POST concludes. This means the RTC is accessed *twice*, once for the time, and again for the date, at two different times during computer boot.

Another misleading statement applies only to the date retrieval. In a typical 286 computer, *no calculation is performed* while the date is retrieved from the RTC. The calculation is performed by the DOS kernel *after* the date is retrieved from the RTC. The DOS kernel calls BIOS interrupt 1Ah to get the date from the RTC, and the BIOS code contains no such conversion routine to make the calculation.*

* See Section 5.3.1.

Figure 1 in this report shows an update in progress map scaled in size to the length of 10 inches (representing one second). For the purpose of this discussion, assume the access time required to obtain date/time information from the RTC is no more than 200 microseconds (0.0002 second). The documented update period which contains bad data is 1984 microseconds (0.001984 second).† The RTC is updated once per second to increment the seconds of time within the RTC in this period. The update period is represented in Figure 1 as a very thin line (actually a box) at the end of the one-second period. This line in the figure is extended beyond (below) the one-second "box" to show the size relationship with the one-second period.

An expanded view of the update period is shown in Figure 2. The 244-microsecond window begins when the UIP bit is set active high. The update period begins after 244 microsecond window ends. Since the update period is 1984 μs long, valid data is available in the entire remainder of the one-second period, or 0.998016 second, or 998,016 microseconds, or rounded to 998 milliseconds. The truth is that the 200-microsecond RTC access time can occur anywhere in this valid data period. There are 4,990 200-microsecond slots in this valid data period. In contrast, there are almost 10 200-microsecond slots in the update period.

Hence, there is a 1:500 chance (or 1:5000 chance, ignoring the 10 time slots in the update period) that RTC access will hit the update period, barring all other considerations.‡

This probability analysis only scratches the surface, because considerations make the odds of hitting the update period even smaller. These are:

- Computer speed (greater speed means shorter access time)
- Number of boots per year versus 1:500
- BIOS compliant to RTC specifications
- Different BIOS code and versions
- Improved RTC design (buffered registers)
- Newer computers
- Older computers with PCI bus architecture
- Any computer using OS/2 operating system
- Any computer using UNIX operating system
- Any computer using Microsoft Windows NT operating system

† Motorola's Technical Data on RTC part number MC146818 and on Japanese version (Hitachi) part number HD146818

‡ The RTC technical data sheet statements regarding statistical numbers for random accesses, such as 1:4032 and 1:2032, are meaningless without more data to corroborate them. They appear to be based on factors other than presented here.

OS/2, UNIX, and Microsoft Windows NT operating systems do not use the BIOS to obtain the date from the RTC, either at boot time or during normal operation. They access the RTC directly. Therefore computers running these operating systems are not subject to the TD anomaly.

This analysis is based on an assumed access time (200 microseconds, aggregate). Faster computers mean less access time to retrieve RTC data. Therefore, the greater the computer speed, the shorter the access time, and hence the smaller the odds of the access time hitting the update period.

Intel reports that the RTC on Echlin's 286 12 MHz computer, which they tested, has the access time of 35 microseconds.[7] This access time is the amount of time it takes to obtain 1 byte of data from the RTC. Retrieving the date takes 4 bytes, which amounts to an aggregate access time of 140 microseconds, which is a far cry from the 244-microsecond window.* Therefore, on a 286 computer, there are 7,129 140-microsecond slots in the period in which data is valid. The odds of hitting the update period becomes smaller (1:700).

Regardless of the computer speed, if the BIOS routines consistently honor the UIP, no TD anomaly should occur, even if the RTC registers are not buffered, theoretically speaking.

If the RTC registers are 100% buffered (including during update period), no TD anomaly should occur, theoretically. This has been empirically shown in software testing by the Crouch/Echlin group. There is no reason to doubt it.

If the computer does show signs of the TD anomaly, it means the RTC has non-buffered registers and may mean the BIOS code does not consistently honor the UIP. What are the chances of this on a computer? It is much smaller than the 1:700 odds. The odds are also greatly reduced on faster computers. There are different computers with different speeds using different BIOS chips with different firmware, and the RTC registers may or may not be of buffered type. The odds are incalculable because there are too many varying factors involved.

There is no standard, method, recommendation, or routine in which BIOS routines should be written in all BIOS chips of all manufacturers. They are all different, because of different chipsets, motherboards, CPU chips, and RTC chips in use.

* See Section 5.3.3.3.

Mr. Mike Echlin gave an account of test results on 100 PCs in an ICQ session with Mr. Harlan Smith in mid-January 1998. This test consists of a two-week manual test by using the computer normally, turning the computer off and on as normally would be done on a routine day, but with the date set to year 2000. Every time a computer is booted, the system time and date is manually recorded. The results below show a trend that the TD anomaly statistically occurs on more older computers than new ones.

Type	**TD?**	
286	75%	
386SX	>75%	
386DX	33%	(I have checked mostly newer machines)
486	25%~	(33 Mhz and lower)
486	12.5%	(Faster than 33 MHz)
586	50%	(Only on overdrives, only checked 4)

The last entry, it should be noted, consists of only four computers. The statements in parenthesis are Mr. Mike Echlin's clarifying notes. It should be noted that the "TD?" in the table header indicates that a TD anomaly has occurred, but has not been verified conclusively. There is no indication of advanced versus retrogressed anomaly; therefore, it only shows general problems which are undefined.

Most people, and especially companies in general, use newer computers to take advantage of better operating systems and applications. Hence the TD anomaly is almost certainly not a concern, but this is only one "plus" in this analysis. You will see henceforth that the very characteristics of the TD anomaly are series of "plusses" that work against its prevalence nationwide, even worldwide.

There is yet one more factor which greatly affects this probability analysis. It is the fact that, should a TD anomaly occur, the erroneous time or date information will be found in the BIOS data area (BDA), but the RTC date and time information *remains correct*. This being the case, a computer reboot will correct the problem. Hence, it is possible that a TD anomaly may pass by unnoticed by the user.*

* Assuming date-sensitive software is not loaded or is not affected by the date change.

However, the incorrect time or date in the BDA may be written to the RTC CMOS memory if the user sets the time or date with the DOS command TIME or DATE, or if a running program executes DOS Interrupt 21h function 2B (set date) or function 2D (set time). There is a software quirk in both Interrupt 21h functions: setting the date also sets the time in the RTC, and setting the time also sets the date. For example, setting the time of day will transfer not just the specified time of day, but also the incorrect date from the BDA to the RTC. See Section 5.5 for a description of this quirk.

4.1 Embedded Systems

One major area heretofore untouched in this analysis is embedded systems used in industries for such applications as automated systems and robotic manufacturing systems. Embedded systems still use 286-based microprocessors, which mean slower processing speed. If these embedded systems operate based on dates, the RTC would be needed. If not, there will never be a serious manufacturing problem due to a TD issue, whether an RTC is used or not.

But if so, there remains a determination whether time/date information is utilized in the system as part of the automated process. If so, then a further determination is needed as to whether the RTC uses non-buffered registers *and* whether the BIOS firmware does not honor the UIP. If so, then the chances of a TD anomaly will increase, *but only in boot-ups*. Remember that the TD anomaly depends entirely on system boot-ups. It does not manifest itself while the computer is running.*

If such an embedded system exhibits a TD anomaly, replacement of the non-buffered RTC with a double-buffered RTC alone may not be a *complete* solution. The BIOS may also be replaced along with the RTC to take advantage of the buffered registers. This is especially true if provision to freeze the user copy of RTC buffers is included in the new RTC.

RTC/CMOS and I/O devices may be embedded in VLSI chips on some motherboards. If this is the case, the only solution is to replace the motherboard or replace the chip or chips containing the RTC and BIOS. Discrete chip replacement will require soldering skills and special equipment. This is also true for PCs.

* Michael Kennedy's work on BCD-to-BIN theoretical experiment (see Section 11.3 for his web site URL) does not apply here. He specifically states that the data input to the test software does not come from the RTC or the BIOS.

An easier solution, therefore, is to replace the whole board with those chips installed. In fact, this is the best solution. For PCs, the simplest and effective solution is to install an adapter card with a buffered RTC and accompanying BIOS chip. Another solution is to purchase a new computer.

There are BIOS chips, even old ones (as early as 1985), which honor the UIP. Does the BIOS routine consistently honor the UIP? If yes, then a TD anomaly should theoretically never occur, buffered RTC or non-buffered RTC. Unfortunately, this question is not easy to answer with diagnostic software, even with Echlin's free TDTLELM.EXE diagnostic utility.* This is because the BIOS firmware is different among different BIOS manufacturers.

Notwithstanding all this discussion regarding embedded systems, Mr. Patrick Bossert's† position is that the TD anomaly has no effect on embedded systems – those used in industries and plants. This is evident in a newsgroup post (uk.tech.y2k) dated sometime in April 1999. With the subject line "Time Dilation: does it effect Embedded Systems?" he has this to say:

> *…I prefer having a ringside seat on the PC issue, but I'm comfortable arguing my bit for embedded systems.*
>
> *The answer is no it does not.*
>
> *See http://www.embedded-science.com/faq/3.html for my dialogue with Mike Echlin on the subject… and arguments for why it doesn't.*

But access into the above URL containing this "dialogue" requires a password. To get a password, you must go to the main page, click on the link to request access, and answer the required questions, including a commentary section.

However, a http://www.y2k-status.org/Embedded/PatrickMikeDialog.htm web page is open for viewing this "dialogue" referred to. Embedded systems are mentioned in the discussion. Based on the appearance and structure, the dialogue was probably conducted through email.

---

* See Section 5.3.3.4.

† Mr. Patrick Bossert is the inventor of a diagnostic device called the Delta-T Probe that performs extensive, technical analysis on RAM, ROM, and RTC chips used in embedded systems. It identifies the time and date information within an embedded system and captures the relevant code to allow the method of time and date comparison being used by the system to be examined in order to determine embedded system compliance.

In this dialogue, Mr. Patrick Bossert states that he has never observed a symptom of the TD anomaly on embedded systems. Further, according to Bossert, date windowing code is used *after* all the RTC date data has been read. This means ”bounds checking” is employed to ensure the century year information fits within a particular range. But this does not mean TD will not occur within the date window.

Windowing is a method used to determine the century year (two high-order digits of the year) for a two-digit year by presupposing that the year falls in a specified 100-year range, or window. There are two types of windowing: fixed and sliding. The fixed method uses a fixed 100-year range. The sliding method uses a 100-year range that "slides" as the years progress over time.

Another web site at http://www.gensw.com/ offers a possible solution. At this site is General Software, the maker of Embedded BIOS™ and Embedded DOS™, providing pre-boot firmware adaptation kits for OEM developers of embedded computer systems. Their packages include source code, which means the BIOS code can be modified. The author has not investigated the feasibility of this embedded system package as a remediation solution to the TD anomaly, if such an anomaly exists.

If a TD anomaly shows up on a PC, and if the date/time has *advanced* into the future from the present time/date, it is proof positive that a TD anomaly has occurred. If the time/date has *retrogressed*, any number of causes applies, and it may *not* be a TD anomaly. This is explained further in this report.

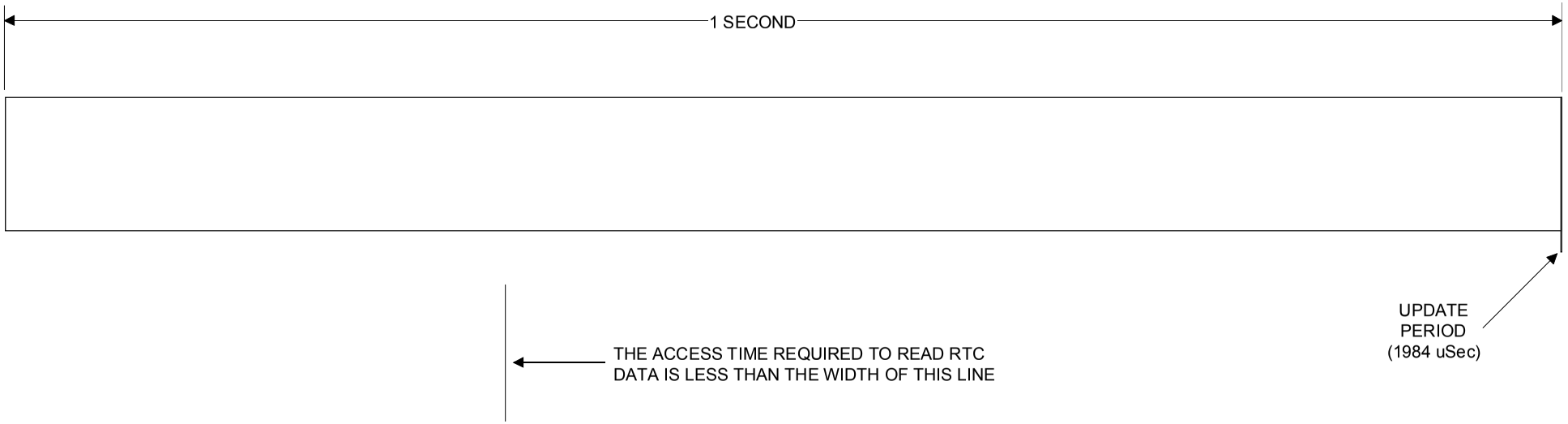
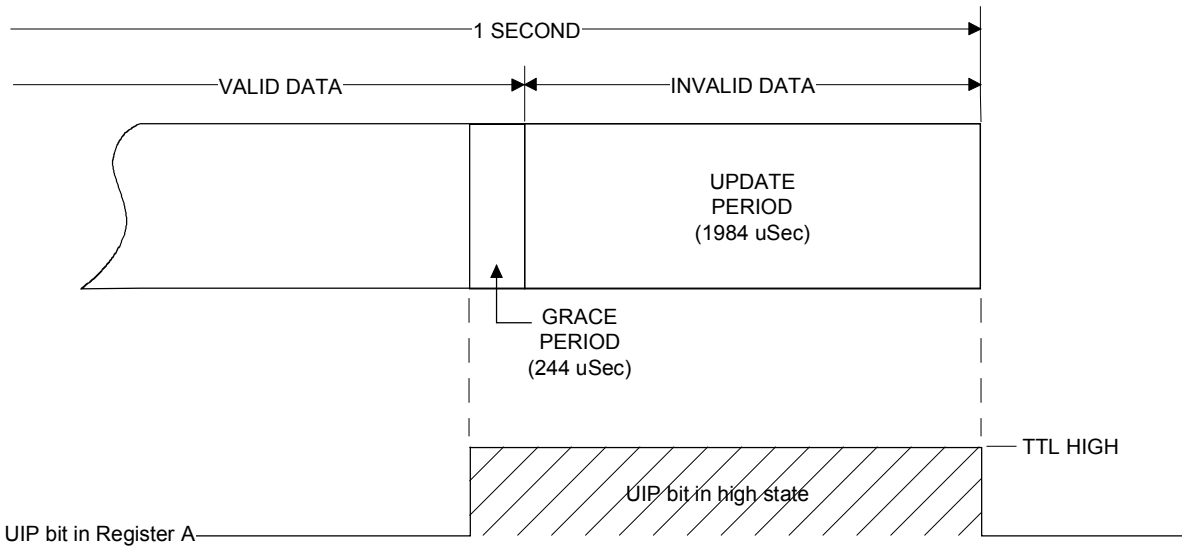


Figure 1: How much luck do you need to hit the update period?



SCALE: 1 INCH = 0.833 mSec or 100 FEET = 1 Second

FIGURE 2: SCALED RELATIONSHIP OF GRACE PERIOD, UPDATE PERIOD, AND VALID DATA WITH THE UIP BIT

5.0 Technical Dilation

5.1 Critical Treatise of Intel's Analysis of the Crouch-Echlin Effect Report

Mike Echlin submitted to Intel two sets of boards and a passive backplane for tests. This is Echlin's two sets of Zenith 286 12 MHz systems with a passive backplane. Each set consists of a CPU board with BIOS chips and an I/O board with an RTC chip. Each set is variously dubbed different names in Intel's report. See Intel Test Methodologies section in Intel's report for a list of equipment submitted to Intel by Mike Echlin.[8] In this report, these CPU boards are dubbed CPU #1 and CPU #2, as Intel does in most places in their report, although this naming convention is not consistent in their report.

Intel's report mentions Jace Crouch's "Zoom" 286-12 clone computer, which is the machine that first exhibited the TD anomaly in mid-1997 and started the whole series of investigations by individual researchers and companies. This anomaly was first reported by Mr. Jace Crouch on 27 August 1997 in the news:comp.software.year-2000 newsgroup. Intel's report also mentions Echlin's Gateway 486-33 computer. However, Intel has not examined either of these computers.[9]

Intel's report also mentions a computer which was received by Compaq/Digital.[10] In further research, it turns out that this "machine" is a motherboard from the same Jace Crouch's "Zoom" 286-12 computer.[11] At Digital, the rest of a computer system was added to the motherboard, all of it new, including a new power supply.[12] According to Jace Crouch, the motherboard still exhibits the TD anomaly.[13]

This section is broken up into subsections below.

5.1.1 Award 4.50g Versus C&T SCAT BIOS

An excerpt from Intel's released public report states:

> *…The BIOS on this machine* [Jace Crouch's affectionately called "Zoom" machine] *is described as an Award 4.50g* ***and*** *a C&T SCAT BIOS on Echlin's website. These are not the same BIOS and we believe the C&T SCAT BIOS to be in error.*[14] [Bold original]

According to Mr. Crouch's public rebuttal, the reverse is true. The AMI BIOS (*not* Award 4.50g BIOS) was first mentioned in an erroneous statement made by Mr. Crouch in a newsgroup message and subsequently the correction was made that the C&T SCAT BIOS was in fact the BIOS in use.[15] The author can vouch for the fact that this message was made long before Intel ever got involved in testing, or even heard of the concept of TD.

This message is shown on the next page. Note that there is not even a mention of an Award 4.50g BIOS. This message is the first of a series on public display on Mr. Jace Crouch's web site.[16]

Therefore, it follows that Intel didn't follow up on their "belief" by further research or by confirming with either Mr. Crouch or Mr. Echlin.

On page 14 of the report, a "C&T SCAT" computer is listed in the summary table of automated testing results. According to this table, this computer is a 386SX (Mr. Crouch's "Zoom" computer is a 286).[17]

Date: Wed, 27 Aug 1997 12:14:26 -0400
From: Jace <deleted>
Newsgroups: comp.software.year-2000
Subject: y2k - strange results in DOS 6.22

On Wed, 27 Aug 1997, J. L. wrote:

> I haven't done any WORK on the PC's SIMULATING Y2000 and beyond - because I
> don't want my date and time stamps on some files messed up. But I'm
> prepared to take a tape backup of my PC and then to run a proper Y2000 test
> to see if anything freaky happens.

I did this with one of the computers in my office:
AMD 286-12, 8 megs RAM
SCAT motherboard, AMI BIOS
<NOTE: it is a C&T SCAT BIOS, I was mistaken when I wrote this. JC>
210M IDE hard drive
New multi-I/O card w/ 2 fast serial ports
MS-DOS 6.22 (no HD compression)
Running a combination of DOS & win 3.1 apps

Don't laugh. It's still a screamer on Word Perfect 5.1, and with a SLIP connection is really fine for running a UNIX shell account! There are plenty of other boxes on campus that are even older; professors tend to keep their newer machines at home and use their old ones at the office.

I did a backup, set the clock to 12-31-99, 11:55:00 and watched. The date rolled over to 00, and various WP and Microsoft applications showed a date of January 1, 2000. Files were saved to the HD with a year date of 00. The win 3.1 file manager read a year date of :0 on these files.

I let the system run for two weeks, and used it from time to time to make overheads, handouts, or handle e-mail. No files were uploaded or downloaded to our UNIX server, although I did keep using the SLIP connection to run Procomm as a terminal emulator. I'd leave the machine on all day, turn it off on nights and weekends. Normal stuff.

Here's what happened:

1. The system clock "ran" extremely rapidly. After two weeks, the system date was mid-December 2000. The date reported in CMOS and reported by various WP and Microsoft applications was identical. Whatever date the RTC reported, the applications displayed. Files were saved with a 00 year date, but win 3.1 file manager displayed a :0 date. These files were readable, writable, and seemed otherwise normal.

2. After about ten days, the system would not recognize that I had two serial ports. For whatever reason, every system test that I ran reported only one serial port. This was the case with Norton Utilities 7, MSD, and an old WP utility. Nothing else started shutting down, but I figured that if a serial port went down, anything could be next, maybe trash the hard drive in some strange time-warp way.

3. Once I set the system clock back to the correct 1997 date, both serial ports were recognized, and they worked fine. I have no idea why the clock "ran" so fast in y2k, nor why the system stopped recognizing the second serial port. This was enough to convince me that even on a simple PC, the y2k problem can cause hardware failures. Fortunately, I was running a word processor to prepare a few handouts and overheads for a history class, not administering the power grid for a three county area.

Peace & stocking up on sterno and propane and MRE's,

Jace

Jace Crouch's Original Message (Bold Added)

5.1.2 Automated Cycling Test Results

In Appendix C of Intel's report, the summary table includes the two CPU boards submitted by Mike Echlin, but only one of the I/O boards was used for both CPU boards.[18] This table is a summary of automated test data – in which power cycling was automated.

Presumably both I/O boards submitted by Mike Echlin have the same RTC chip, but there is no information other than the fact that the RTC chip is a Motorola part number MC146818 and is not buffered.[19] The author has the same chip on several of his computers, including two 286 computers, but they are Hitachi chips (Japanese version). Both chips are exactly the same as the Motorola chip, as attested to by technical data on each. Yes, the author has the Hitachi data sheets dating back to 1987 – they were hard to find.

According to "Echlin System Test Methodology" section of Intel's report, during automated cycling tests Echlin's TDFIX utility was loaded on at least one of Echlin's CPU boards.[20] The exact language of Intel's statement regarding the loading of TDFIX implies that this was done *during* the test. While this would invalidate the test results because any occurrence of the TD anomaly is circumvented by the TDFIX utility, it does not mean TDFIX was loaded in the *entire* test. Intel did try to "look for any unexpected jumps" in the system date/time.[21]

Intel states that they "…did not observe any case of the Crouch-Echlin Effect…" on either one of the two CPU boards.[22] But this statement contradicts another statement: "TDFIX reported that it found the Crouch-Echlin Effect symptom… the utility is reporting discrepancies between RTC data and BIOS data, *as on Mr. Echlin's Zenith*." [23] Another contradictory statement is: "On Processor Card #2, TDFIX.EXE was observed to intermittently detect a difference between the DOS system clock and the RTC…This discrepancy matched the value reported by TDFIX.EXE." [24]

A discrepancy between RTC data and BIOS data, as explained in this report, is not necessarily a symptom of the TD anomaly. A time difference of only 2 seconds or so is not a valid symptom of a TD anomaly. But apparently Intel feels the need to report the findings of the TDFIX.EXE utility, yet without revealing the time difference. Evidence in textual analysis of their report indicates that Intel is attempting to show – without saying so – that the results of TDFIX utility is actually from the BIOS time conversion anomaly – which is true. But Intel is careful not to show the actual TDFIX time difference results.

Note that a *symptom* of the Crouch-Echlin Effect was observed, not the Crouch-Echin Effect itself. Nevertheless, Intel indicts itself by saying they "did not observe *any case* of the Crouch-Echlin Effect," which by definition should include any *symptom* of it.*

But, as later shown in this report, the BIOS time conversion anomaly is not a case of the TD anomaly. Instead, a different time anomaly exists, and this is the ostensible reason the actual TDFIX time difference results are not revealed.

In April of 1999, a short series of newsgroup email messages ensued in a somewhat heated contention that the automated cycling test results is flawed because of the manner in which AC power to the test unit (a computer) is removed and then applied. There was even a claim that manually applying power to the test unit is somehow different from an automated power cycling method. From an engineering point of view, this is ludicrous. The method of power cycling makes no difference in the test results, especially since the power supply itself does not contribute to the anomaly.†

In the summary table in Appendix C, CPU #1 was power cycled 3108 times, CPU #2 was power cycled only 600 times, and the rest of the systems under the same test were power cycled 4026 times. No explanation was given for the apparent incompleteness of tests on CPU #1 and #2 boards.

5.1.3 Manual Cycling Test Results

In the Manual Cycling section of Appendix C, a two-page table of test data detailing manual cycling results appears. This test was conducted on Mr. Echlin's Zenith CPU board #2, which is the board that Intel stated exhibited "some anomalies with the BIOS".[25]

* The counter-argument "That depends on what the definition of 'any case' is" is hardly valid, since the term "any case" makes the definition very broad.

† See Section 5.1, bottom half of 3rd paragraph.

The data shows that the test was begun on Wednesday, November 25, 1998, and ended on Wednesday, December 16, 1998, a period of 3 weeks. The data shows continuous daily entries from November 25 (Wed) to December 4 (Fri), which is a span of 10 consecutive days that includes the weekend of November 28 and 29. The recording stops on that Friday, December 4, resumes the following Monday, December 7, and continues to Tuesday, December 8. However, nothing is recorded on December 9, 10, and 11 (Wednesday through Friday inclusive). The recording resumes on Monday, December 14 and stops on Wednesday, December 16. A final recording is made on the first workday of the New Year (actual year 1999, computer year date 2001) on Monday, January 4, apparently to show the date after the rollover.[26]

Intel's report states that an aggregated two-week test was performed, and this is attested to by the data in their report.[27] The system date was set to year 2000 during the test.

However, the other CPU board, CPU #1, was not tested using this methodology. Intel is not attempting to hide this fact. An excerpt reads in their report:

> *Intel has attempted to duplicate the Crouch-Echlin Effect on **one** of the systems that Mr. Echlin has observed to exhibit this anomaly. These are the steps we took to produce and analyze the Effect.*[28]
> [Bold added]

Based on Mr. Jace Crouch's public rebuttal statements on one of his pages, CPU board #1 is implied to be the board that has "exhibited TD very frequently," while CPU board #2 is implied to be the board that "may have exhibited TD on one occasion only." [29] This natural conjecture is based on Mr. Crouch's leading statement: "…guess which one [Intel] reported on?" (Intel reported on CPU board #2.) This is a leading statement that points the reader in the wrong direction: the presumption that CPU board #2 is the one that "may have exhibited TD on one occasion only," so that Intel is accused of testing the wrong board and ignoring the other board that has "exhibited TD very frequently." Compare this with Intel's own statement:

> *On CPU board #1 we did not observe any case of the Crouch-Echlin Effect, **nor did TDFIX.EXE detect any discrepancy between the DOS system clock and the RTC.** We also did not observe any case of the Crouch-Echlin Effect on CPU board #2; however, we did see some anomalies with the BIOS on this board.* [30]
> [Bold added]

TDFIX did not find any problems with CPU board #1. And Intel makes a clarification regarding CPU board #2 in Appendix A of their report:*

> *On Processor Card #2,* ***TDFIX.EXE was observed to*** *intermittently* ***detect a difference between the DOS system clock and the RTC****…This discrepancy matched the value reported by TDFIX.EXE.* [31]
> [Bold added]

Intel's statements show the *opposite* of Mr. Jace Crouch's veiled implication is true. CPU #1 is the board that "may have exhibited TD on one occasion only," and CPU #2 is the board that has "exhibited TD very frequently." The author's rendition of the truth is based on Intel's own statements regarding the reported *results of the TDFIX utility on each board*.†

Therefore, the CPU board tested in the manual cycling test is actually the board that exhibited the TD anomaly *frequently*.‡ And indeed the board does show evidence of the TD anomaly, and the proof is briefly shown here in this section and greatly expanded in Section 5.1.4.

A reader at CompuServe Year 2000 Forum reported that it is unclear whether Intel actually tested for the occurrence of TD *without* Mr. Echlin's TDFIX software installed. This is not true in the manual cycling test. An excerpt reads in Intel's report:

> *Mr. Echlin's machine was set up with only a PROMPT command in the AUTOEXEC.BAT and no CONFIG.SYS.*[32]

The above excerpt stated that the AUTOEXEC.BAT file contains only a PROMPT command, and there was no CONFIG.SYS file. Hence, TDFIX was not loaded during manual cycling test. §

As a summary, Table 1 shows which board underwent which test, troubleshooting, or examination.

---

* If you look carefully at these two excerpts, you will see a discrepancy in Intel's statements. This is already covered in Section 5.1.2, which see.

† Also based on CPU #2 manual cycling test data and Appendix A, in which Intel commits a glaring blunder. See Section 5.1.4.

‡ This CPU #2 board, according to Intel, exhibited a BIOS INT 1Ah bug, and was cited in Intel's Appendix A.

§ The author concedes it is possible to load TDFIX.EXE via AUTOEXEC.BAT, but has no reason to believe so based on careful analysis. The manual test data itself shows that the utility was not loaded: the test data shows evidence of a TD anomaly.

Note that the CPU #1 board was tested only in the automated cycling test. This board is the one that "may have exhibited TD on one occasion only." This board never did exhibit any TD anomaly through the TDFIX utility, according to Intel's statements. Since this board was tested in automated cycling test with TDFIX installed, the truth about this board will never be known, except for the statement Mr. Jace Crouch already made: that this board "*may* have exhibited TD on one occasion only," and the statement Intel already made: that TDFIX never found a problem with it. This author accepts at face value that CPU board #1 does *not* exhibit such an anomaly.

Nomenclature	CPU #1	CPU #2
Hypothesis #1 (Separate Code Paths)	√	√
Hypothesis #2 (Improper UIP Handling)	√	√
Hypothesis #3 (Interrupts Not Disabled)	√[1]	√[1]
Hypothesis #4 (DOS Kernel Time/Date)	√[1]	√[1]
Hypothesis #5 (COMMAND.COM Conversion)	Note 2	Note 2
Automated Cycling Test (Appendix C)	√	√
Manual Cycling Test (Appendix C)		√
Additional Observations (Appendix A)		√
BIOS INT 1A Handler Bug (Appendix E)		√[3]

TABLE 1: Board Test Summary

Notes:

1. Board numbers not mentioned in text
2. Board-independent test
3. Board number implied in Appendix A, page 12, next to last paragraph – Appendix A is dedicated to CPU #2.

According to Hypothesis #2 section, Intel's investigation concludes the BIOS code on Echlin's 286 computer properly honors the UIP bit.[33] But Intel's statements indicate that the BIOS code was only cursorily examined, and no actual tests were performed to make or confirm this particular finding.* In addition, it does not mention which board was examined, but the phrase "Echlin's boards" is mentioned in plural.

But regarding CPU board #2, a careful examination of the test data in Appendix C (pages 16-17) indeed shows some possible cases of the TD anomaly, all of them involving the time of day. But first, the caveats:

1. A wristwatch was used as a reference.
2. The accuracy of the wristwatch is not known but is a moot point since the seconds of time is not recorded.

* However, extended analysis on the *characteristics* of the TD anomaly on Echlin's CPU #2 board shows indirect evidence that the UIP bit is being properly honored (see last parts of Section 5.1.4).

3. The wristwatch seconds of time is not included in the recorded reference time.

Notwithstanding the caveats, these cases show evidence of an advanced time relative to the reference time. An *advanced* time/date symptom is a strong indicator of the TD anomaly, as opposed to a retrogressed time/date symptom.

The time differences of these cases are shown in Table 2.† Since the time reference does not include the seconds of time, an accuracy of ±50 seconds is imposed before a case is considered evidence of an anomaly. This criteria permits a maximum reference time error of 16.7%. The differences are approximations only, rounded to the nearest half-minute. In the Delta column, a "+" indicates advanced time relative to the reference time, and a "-" indicates retrogressed time.

Case	Day	Cycle	Delta (Secs)
1	7	2	+60
2	8	1	+90
3	9	2	+60
4	9	3	+60
5	11	1	+60
6	14	1	+60
7	15	2	+60

TABLE 2: Worst Case Evidence of TD Anomaly

An advanced time or date relative to the true time or date is proof positive that a TD anomaly is appearing. There is no scenario or condition identified with causing a false case of retrogressed time/date anomaly that <u>also</u> causes a false case of *advanced* time/date anomaly. In fact, there is no *precedent* for such an advanced time/date symptom.

Speaking of precedence, according to Mr. Michael Kennedy, the TD anomaly has been around since 1984, when the AT 286 computer was first introduced. According to Mr. Kennedy, the BIOS code listing released in 1984 shows it is not fully RTC-compliant.‡

† The time differences are based on face value of the wristwatch reference time, with the seconds of time at 0 seconds.

‡ However, the BIOS code released the next year (1985) clearly shows the BIOS code has proper instructions to access the RTC date information, as well as the code for the time information. A masterful job was done in the source code, considering the limitations imposed by the hardware, the 286 processor chip included. The author has not seen the 1984 source code.

If the ±50 second criteria were changed to ±60 seconds, only Case 2 would qualify, and the maximum reference time error becomes a non-issue because the delta is greater than 60 seconds, which is the smallest resolution offered by a wristwatch without a second hand.

These cases should never occur on any computer just after a boot up. The maximum error to be expected on any boot up on any computer is about 2 seconds or so (on the retrogressed side, not on the advanced side).

There is but one flaw in this test methodology. A wristwatch is not in sync with the RTC, even if the RTC is set to the same time as the wristwatch. A better test is to examine the RTC, BIOS, and DOS time/dates just after computer boot by using the ViewCMOS program, written by Tom Becker. A typical TD anomaly would show a difference between the RTC and BIOS time/date information. The wristwatch test methodology will show this difference between the wristwatch and the reported DOS time, but will not reveal as much detail about the RTC, BIOS, and DOS.

It is already proven that the RTC is not the culprit for the TD anomaly in Section 5.3.3. Hence, examination of the RTC, BIOS, and DOS time/dates is much more appropriate and more accurate.

Using the ViewCMOS (or equivalent) program will easily determine whether a TD anomaly is being exhibited. A difference, especially a large one, between the RTC time/date information and DOS counterpart is evidence of a TD anomaly. If these are the same within 2 seconds or so, there is no TD anomaly, even if the RTC time/date does not show the correct time and date. In this case, the battery may be weak, or some other problem may be responsible. A number of causes applies to this symptom, but in no case should the RTC itself advance ahead in time. Again, this symptom is not a valid case of a TD anomaly.

Mr. Crouch once complained that Intel did not record the seconds of time on the wristwatch in their manual cycling test data. This missing portion of data is not lost; it is possible to reconstruct the wristwatch seconds of time based on Intel's own test data. This will be explained in no uncertain terms in the next section.

The issues herein will be addressed again and greatly expanded in the next section.

5.1.4 Glaring Intel Blunder

In Appendix A of Intel's report, an anomaly in the BIOS on Mr. Echlin's computer is reported and described.[34] In it, it is revealed that part of TDFIX function is to check the BIOS time (tick counts) against the RTC time. While it is prudent to cover this area of remediative fixes, it is nevertheless not the cause of the TD anomaly. It is merely a *symptom* of it; but caution must be voiced that a slight deviation of BIOS time from RTC time is quite normal and is not an indicator of a TD anomaly.*

The RTC and BIOS time (tick counts) use different frequency sources, and they are not the same frequency. (Even if they were the same frequency, there would still be a drift between the RTC time and BIOS time, because they are not synchronized.†)

On computer startup, the time difference between RTC and BIOS/DOS is typically about 2 seconds, but as time goes on, the drift will widen slightly, depending on how long the computer remains powered on. However, a boot up will always "correct" or initialize the DOS time with the RTC time, since DOS time must be initialized with RTC time at every boot up.‡

Intel did not explain this. Neither, apparently, did Intel test their hypothetical analysis regarding boot ups at 47-minute (2820-second) intervals. They didn't do it because they knew they couldn't. This excerpt shows it is merely a theory:

> ***For example**, with Processor Card #2, if the computer was powered on at 12:47 AM the timer ticks in the BDA **would be** slow by 1 second; at 1:34 AM the timer ticks in the BDA **would be** slow by 2 seconds;…*[35]
> [Bold added]

* If a computer with a midnight rollover bug is left on over several days, there will naturally be a big difference between DOS date and RTC date. This is still not a valid indicator of a TD anomaly.

† That is, if *two* frequency sources were used at the same frequency. If a *single* frequency source were used for both RTC and BIOS/DOS, there would be no drift. Computer systems are not designed this way, because the RTC must have its own oscillator that continues to run with the computer powered off. This is why there is a slow drift of RTC time and BIOS time apart from each other the longer the computer remains powered on. Another reason is the oscillator is not oven-controlled, which means very slight variations of frequency due to ambient temperature changes inside the computer.

‡ Does not apply to operating systems that monitor the RTC directly instead of the BIOS for time/date information. These are: UNIX, OS/2, and Microsoft Windows NT.

This is a clever show of numbers without actually documenting the results from their own test data. Intel used a series of mathematical calculations based on the delta percentage of the two formulas shown in Appendix D (0.036%).[36] These theoretical numbers are correct, as explained below.

Calculations based on this information shows that the time difference between each set of two example lines, except for two sets of them, is 2820 seconds, or exactly 47 minutes. The other two sets show a difference of 2760 seconds, or exactly 46 minutes. The 2820 figure roughly coincides with the 2809 seconds mentioned in Appendix D. Intel simply added 2820 seconds to the previous line to get the next time value, and increments the seconds in the phrase "will be slow by *n* seconds", where *n* is the incremented value. Only twice is the figure 2760 added instead of 2820.

But subterfuge is involved in the use of the 0.036% figure. Intel is laying blame on the Zenith BIOS conversion routine, but there is more to this and Intel's test data than meets the eye.

In Appendix D, two formulas are shown: one is a DOS algorithm (which is indeed undocumented); the other is a formula that Intel claims is used by the Zenith BIOS in Mr. Echlin's computer. The percentage figure noted (0.036%) is correct, the formulas will each result in a difference of 0.036 percent from the other. *This difference is only relative to the DOS algorithm.* Only the BIOS conversion routine is used during POST, not the DOS algorithm. The DOS algorithm is used for setting the time of day in the BDA (as well as in the RTC) *by user interaction*. The BIOS algorithm is used for setting the time of day in the BDA during POST. For Intel's purposes, the DOS algorithm is used only as a point of reference to show how far off the Zenith BIOS conversion routine really is. It has nothing to do with TD.

DOS uses another conversion routine (not the one highlighted in Appendix D) to convert from tick counts to a time format whenever the time needs to be displayed, or whenever a file needs to be time-stamped. An inaccurate tick count value in the BDA will be reflected in the time information returned by DOS.

The maximum error offset (introduced by the 0.036% offset from the true time) would be no more than 31 seconds, and this amount of offset would occur only if the Zenith 286 were booted at 23:59:59.* This is hardly a case for a TD anomaly because the maximum error is only 31 seconds.

* Ignoring, of course, the known midnight rollover bug.

Intel's calculations (shown on pages 11~12 of Intel's report) of the BDA tick counts using the 0.036 percentage figure is also correct, from a theoretical standpoint.

But Intel commits a glaring, pyrotechnical blunder in espousing the mathematical theory, because the theory *does not fit* the manual cycling test data results in Appendix C. The maximum offset error introduced by the 0.036% offset theory is *no more than 31 seconds*,† but the test data clearly shows some cases in which the time difference is *much more than 31 seconds*. The manual cycling test data in Appendix C and the theory expounded in Appendix A both refer to the same CPU board #2.

In Appendix A, the 0.036% offset theory reflects a *retrogressed* time anomaly ("the timer ticks in the BDA would be *slow* by…"). But some of the test data in Appendix C indicate an *advanced* time anomaly. Hence, the 0.036% offset theory and the test data do not fully support each other. This is graphically shown in Charts 2 and 3, but we are getting ahead of ourselves here.

Chart 1: Curious But Rough Pattern Match

Intel's test data table does not include a column showing the time differences between the reference time (wristwatch) and the displayed time (DOS-reported time). These time differences are shown in Table 1 and Chart 1 at the end of this section. As already mentioned, the wristwatch reference time does not contain the seconds of time. For the purpose of this analysis, Chart 1 and Table 1 assume the reference time starts at 30 seconds. Thirty seconds is chosen because the error (or accuracy, depending on how one looks at it) in the reference time (wristwatch) is no more than +29/-30 seconds, since the minutes of time was recorded as part of the official data.

In Tables 1, 2, and 3, the Time Value columns contain decimal fractions ranging from 0 (zero) to 0.99999999, representing the times in the respective Date/Time columns from 00:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.). The TIMEVALUE() function was used in Microsoft® Excel 97 to convert the time information in the Date/Time columns to decimal fractions in the respective Time Value columns.*

† If the Zenith 286 computer were booted at 23:59:59. If booted at an earlier time of the day, this error will be less than 31 seconds depending on the time of day. At midnight, the tick counts reset to zero and count upwards again. By Intel's definition, *the amount of error offset is linearly proportional to the time of day*.

* These Time Value columns are not important for our consideration. More important are the Delta Time and 0.036% Ref Offset columns, as explained in the following text.

Then the decimal fractions in the Wristwatch Time Value and Display Time Value columns are each converted into seconds, and the algebraic difference in seconds between them are shown in the Delta Time column. This data is plotted in Charts 1, 2, and 3 as a red line.

Finally, the Wristwatch Time Value data is first converted into seconds, then multiplied with 0.036% to compute the calculated (theoretical) offset in seconds that Intel claims would be reflected in the DOS-reported time. This computed offset is shown in the 0.036% Ref Offset column and is plotted in Charts 1, 2, and 3 as a dark blue line. All charts show the same plot as a reference plot, although the values in the 0.036% Ref Offset columns of Tables 1 and 2 are slightly different because the wristwatch seconds of time is mathematically adjusted in Table 2.

Note that the 0.036% Ref Offset data are all in negative values. The phrase "timer ticks in the BDA would be *SLOW* by…" reflects retrogressed time anomaly, hence the negative numbers. In other words, DOS-reported time is based on the tick counts in the BDA; theoretically speaking, if the tick counts is less than it should be, DOS reports a time that is less than it should be by the amount of time indicated in the 0.036% Ref Offset column.

The zero line in the charts represents zero deviation from the wristwatch reference time. A positive Delta Time value relative to zero is a case of an advanced time anomaly. A negative Delta Time value relative to zero is a case of a retrogressed time anomaly. Hence, the greater the time difference between the reference time (wristwatch) and DOS-reported time, the greater the deviation from the zero line.

A characteristic of a normal computer boot is that the DOS time is 2 seconds or so *behind* the RTC time. This is a normal case of "retrogressed" time. Thus a deviation of –2 seconds in the charts is a normal deviation. (Keep in mind, however, that the plotted deviation is based on a wristwatch, not the RTC time.) A deviation above –2 seconds (or perhaps –1 second in some computers) should never occur, and points to a unique anomaly because the DOS time (or date) theoretically should *never* be ahead of the RTC time (or date) when initialized in a computer boot.

Intel's test data contains a total of 42 boot cycles during the entire test period. The Case Number axis in the charts is the boot cycle number.

Red data points above the zero reference line are the cases in which the delta time is excessive compared to the zero reference line. Remember that the most deviation in time between the RTC and DOS on computer boot should be *no more than –2 seconds* or so. All these data points are strong cases for advanced time anomaly.

Case number 21 (Day 8, Cycle 1 in Intel's test data) is the strongest case for an advanced time anomaly, showing a delta of about 60 seconds from the zero reference time in Chart 1 (57 seconds in Table 1).

If all the wristwatch reference time data were changed to 0 (zero) seconds instead of 30 seconds in Chart 1 and Table 1, all data points in the plotted red line (actual) would be 30 seconds higher relative to the zero reference line than is shown in Chart 1. By the same token, a wristwatch reference time with 59 seconds of time would result in red data point values 29 seconds lower than is shown in Chart 1. The pattern of the plotted line itself will not change, as long as the wristwatch seconds of time in all Case Numbers are the same.

Therefore, Chart 1 only has a resolution of 59 seconds (+29/-30 seconds to be exact), because the wristwatch reference time in Intel's test data does not include the seconds of time. While Chart 1 does show proof that the actual results in Intel's test data does not match Intel's 0.036% offset theory in terms of peak-to-peak deviation,* Chart 2 and 3 are more accurate and each solidifies this very proof. Chart 1 is based on a wristwatch reference time that includes exactly 30 seconds, as if the DOS-reported time was recorded when the reference time was exactly 30 seconds into the current minute of the wristwatch.

Chart 1 shows a very curious but very rough pattern match between the red line (actual) and the blue line (calculated). A study of the plotted lines in Chart 1 will bear this out. The "peaks" and "valleys" in both red and blue plotted lines roughly follow each other and the number of "peaks" and "valleys" are the same. This is borne out from the fact that the seconds of time in the Wristwatch Ref Time column are all the same, and the fact that the 0.036% offset error is *linearly proportional to the time of day*. This is the only characteristic that confirms Intel's reported Zenith BIOS 0.036% offset problem.

* Peak-to-peak deviation: An engineering term defined as the delta between the two highest and lowest points in a plot.

Chart 2: Best Case Scenario

In order to positively confirm or deny the occurrence of the TD anomaly in Intel's test data, the seconds of time in the wristwatch reference time must be known. Hence, the author has attempted to reconstruct the actual seconds of wristwatch time during which the DOS-reported time and date was recorded. The result is Table 2 and Chart 2. How is this done? First, known factors must be established:

1. The wristwatch reference time in Intel's test data has a resolution of only 1 minute. Hence, only 59 seconds (+29/-30 seconds in Table 1) is allowed in the adjustment of the wristwatch reference time.
2. Intel's 0.036% offset theory is known and can be calculated based on Intel's own test data.
3. The 0.036% offset error is *linearly proportional to the time of day*. Why? Because the greater the seconds of time since midnight, the greater the offset error. For this reason, for each time of day there is a specific offset error. This makes it easy to check the offset error against Intel's wristwatch seconds of time window (59 seconds) and calculate the probable seconds of time that resulted in the offset error.
4. The actual DOS-reported time includes the seconds of time, which is very helpful. In fact, were the seconds of time deleted, the tables and charts would not have been possible.
5. All these factors make it possible to reconstruct the seconds of time in the wristwatch reference time based on the 0.036% offset error.

The goal is to match the Delta Time column data with the 0.036% Ref Offset column data in Table 1, but *without changing the minutes of time* in the wristwatch reference time, since the minutes of time is part of Intel's official test record. This was done using the adjustment formula shown in Figure 3.

[Adjusted Wristwatch Seconds of Time] =
[Delta Time] - [0.036% Ref Offset] + [Current Wristwatch Seconds of Time]

FIGURE 3: Wristwatch Seconds of Time Adjustment based on 0.036% Offset

Where:

(1) [Delta Time] parameter is an algebraic value.

(2) [0.036% Ref Offset] parameter is an algebraic value rounded to the nearest second.

(3) [Current Wristwatch Seconds of Time] parameter is the seconds of time shown in Wristwatch Reference Time column, which is 30 seconds in Table 1. Applying this formula to Tables 2 and 3 will result in [Adjusted Wristwatch Seconds of Time] = [Current Wristwatch Seconds of Time] if [Delta Time] and [0.036% Ref Offset] algebraically cancel each other out in those tables.

(4) If the [Adjusted Wristwatch Seconds of Time] value in the equation exceeds 59 seconds, only 59 seconds is given in the Wristwatch Ref Time column in Table 2 *so as not to increment the minutes of time* in Intel's official test data. The remainder of the computed value is reflected in the algebraic difference between Delta Time and 0.036% Ref Offset values in Table 2. In these cases, the [Delta Time] and [0.036% Ref Offset] cannot cancel each other out. This is where serious, irreconcilable discrepencies are exposed.

The result is the adjusted wristwatch seconds of time in Table 2 based on Intel's 0.036% offset theory. In Table 2, the values in Delta Time and 0.036% Ref Offset columns now closely match each other, except in a few places where 59 seconds are exceeded in the above equation. These few are Case Numbers 17, 20, 21, 25, 26, 31, 35, and 39. Chart 2 shows the plotted results from Table 2.

Table 2 of the previous Section 5.1.3 show worst case scenarios for Case Numbers 17, 21, 25, 26, 31, 35, and 39 (excluding Case Number 20). These are listed as Case 1 through 7, respectively, in the table of Section 5.1.3. With the wristwatch time set to 0 seconds of time, all of these cases show advanced time anomalies, not just Case 2 in that table (Case Number 21).

An additional Case Number 20 is added for a total of 8 (instead of 7) data points exhibiting the anomaly. In Table 2 of the previous Section 5.1.3, Case Number 20 did not meet the imposed ±50 seconds criteria for the Delta Time (44 seconds), so it is excluded from the table in that section. However, in Chart 3 the Delta Time for that Case Number shows a value of +44 seconds, showing another case of an advanced time anomaly – if the wristwatch reference time was at zero seconds (Table 3) at the recording of the DOS-reported time.

Chart 2 is *proof positive* that Intel's test data does not entirely match their 0.036% offset theory. Where the red line (actual) does not follow the blue line (calculated), these red data points (actual) are proof positive that a *different anomaly* is being exhibited. Furthermore, these data points not following the calculated blue line show a deviation in a *positive direction* relative to the blue line.

Case Numbers 21 and 26 are proof positive that an *advanced time anomaly* is being exhibited, since the data points are *above the zero reference line* in Chart 2. Case Number 39 is slightly above the zero reference line. Hence, Case Numbers 21, 26, and 39 are the only cases pointing to advanced time anomaly in Chart 2. The rest of the data points not following the blue line show evidence of retrogressed time anomaly.*

Where the red data points (actual) departs from the blue data points (calculated) in Chart 2, the difference between them in terms of seconds *remains unaccounted for* in Intel's 0.036% offset theory. The algebraic difference in the Delta Time and 0.036% Ref Offset in Table 2 also shows the same difference. The seconds of time cannot be reconstructed because the result of the adjustment formula in Figure 3 is greater than 59 seconds, which is outside the 59-second limitation.

* The data points in these Case Numbers are actually *conservative* numbers, reflecting best case scenarios and not worst case scenarios, because the Wristwatch Ref Time is given the whole of 59 seconds in the column. See further in the text for explanation.

For example, Case Number 21 (see Intel's test data, Day 8, Cycle 1) shows the measurement took place at 7:50. According to the 0.036% offset theory, at 7:50 the DOS-reported time (Display Date/Time column) should be off by about –10 seconds (0.036% Ref Offset column) relative to the wristwatch reference time. But the Delta Time column in Table 2 shows the DOS-reported time is off by *+28 seconds* with the wristwatch reference time adjusted to the maximum of 59 seconds. In Table 2, the sum of 28 and 59 seconds with the 0.036% Ref Offset value of –10 is 97 seconds, which is the same value returned by the Figure 3 adjustment formula with the numbers plugged from Table 1. Obviously, 97 seconds cannot fit into the 59-second resolution of the wristwatch time. Hence, 59 seconds is given to the wristwatch seconds of time, and the remainder is the algebraic difference of the Delta Time and 0.036% Ref Offset values (38 seconds), as is shown in Chart 2.

Because the Wristwatch Ref Time for Case Number 21 is already given the maximum of 59 seconds, the seconds of time cannot be adjusted further. Hence, the seconds of time cannot be reconstructed for Case Number 21, and hence is *unknown*.

Therefore, if the adjustment formula result (Figure 3) is greater than 59 seconds, reconstruction of the wristwatch seconds of time based on the calculated 0.036% offset value is impossible.

For all Case Numbers where 59-second limitation (+29/-30 seconds in Table 1) is exceeded in the adjustment formula, the wristwatch seconds of time for those Case Numbers are *still not known*. Therefore, it is appropriate to consider the worst case scenario with the seconds of time adjustment set to zero instead of 59 seconds in Table 2. With the seconds of time at 0 (zero) seconds, the DOS-reported time would be off by a whopping *+87 seconds* in Case Number 21 (as shown in Chart 3 and Table 3).

For example:

1. For Case Number 21 in Table 2, (28 seconds Delta Time) + (59 wristwatch seconds) = 87 seconds.
2. For Table 1, (57 seconds Delta Time) + (30 wristwatch seconds) = 87 seconds.
3. For Table 3, (87 seconds Delta Time) + (0 seconds) = 87 seconds, since the wristwatch seconds of time is 0 seconds.

All other cases with 59 seconds of time in the Wristwatch Ref Time (Table 2) can be similarly computed for worst case scenarios.

Chart 2 shows red data points in Case Numbers 17, 20, 21, 25, 26, 31, 35, and 39 departing from the plotted blue line. Since the seconds of time for the Wristwatch Ref Time is given 59 seconds in these cases, the plotted data points (and Delta Time in Table 2) are actually *conservative numbers*. Since the seconds of time cannot be greater than 59 seconds, it can only be reduced. This reduction will worsen (increase) the Delta Time, thus creating a greater deviation from the zero reference line.

Hence, Chart 2 represents the *best possible scenario* for these Case Numbers.

Chart 3: Worst Case Scenario

The worst case scenario is seen by adding 59 seconds to the plotted data points in Chart 2 (29 seconds in Chart 1). This applies only to Case Numbers 17, 20, 21, 25, 26, 31, 35, and 39. The result is Chart 3 and Table 3.

In Chart 3 and Table 3, the wristwatch seconds of time is adjusted for 0 (zero) seconds only for these Case Numbers cited to show the worst case scenario.

Charts 2 and 3 show the range of likely Delta Time values for these Case Numbers based on a wristwatch reference time between 0 and 59 seconds. All these data points always remain above the blue plotted line. *It means none of the Case Numbers will ever be reconciled* with the 0.036% offset error, *regardless* of the wristwatch seconds of time. As already mentioned, the wristwatch seconds of time for these Case Numbers cannot be reconstructed because the Figure 3 formula result is *greater than 59 seconds*.

The timer ticks in the BDA must be initialized with the RTC time every time a computer is booted up. *This means the maximum DOS time deviation from RTC time is necessarily always close to zero each time a computer is booted.* The timer ticks should never differ by more than two seconds just after boot up regardless of the time of day. But Charts 2 and 3 not only confirms Intel's 0.036% offset theory, but also confirms that *another anomaly is being exhibited*.

The BIOS offset error is similar to the TD anomaly, for the TD anomaly is based on this very discrepancy *as a symptom* of it, seeing that the time difference is much higher than a mere 2 seconds. Again, the BIOS offset error is not a valid case of a TD anomaly because the maximum offset error is 31 seconds on any given day:

On Processor Card #2, TDFIX.EXE was observed to intermittently detect a difference between the DOS system clock and the RTC. Upon further investigation we discovered that the algorithm used by the BIOS version on Processor Card #2 to convert the time from the RTC to the number of clock ticks since midnight lost approximately 1 second every 2800 seconds. ***This would account for the zero to 30-second discrepancy between the DOS system clock and the RTC,*** *depending upon what time during the day the system was booted.* ***This discrepancy matched the value reported by TDFIX.EXE. This conversion error*** *occurs independent of the value of the century byte, and* ***occurs on every boot cycle****.* [37] [Bold added]

But Intel reveals nothing of the values reported by TDFIX. Intel does not record the seconds of wristwatch reference time. Intel does not record the delta time between wristwatch reference time and DOS-reported time. The purpose of the tests is to determine the feasibility of the TD anomaly as a fact. Yet nothing is recorded to prove one way or the other; instead, Intel presents a mere theory. Since Intel concedes the discrepancy matched the value reported by TDFIX, one may presume that the discrepancy may in fact be a TD anomaly in at least *some* cases, but certainly not all cases, as this analysis has shown.

We know that the timer ticks must be initialized with the RTC time at every computer boot up. If there is a discrepancy greater than the maximum 31 seconds, either the discrepancy resulted from a boot up or the computer was allowed to run for a very long time without a boot up. We know it was the former, because Intel's test data shows that the DOS-reported time was recorded immediately after computer boot. Further, a large difference between the RTC and DOS time/date information is the very symptom of TD anomaly – *especially if the symptom is observed just after boot-up*.

In summary, several important observations are revealed in this analysis:

1. Where the time difference between Wristwatch Ref Date/Time and the Display Date/Time is greater than 59 seconds, the excess time cannot be reconciled for Intel's offset theory to be a valid cause of the anomaly. In fact, it is proof that in such cases a TD anomaly is being exhibited, and an *advanced* time anomaly is hard to disprove.
2. On close examination of Chart 1, the plotted red line very roughly follows the general pattern of the plotted blue line. This tends to support Intel's 0.036% offset theory as fact, as explained in this analysis.

3. In Chart 2, Case Numbers 21, 26, and 39 deviate from the calculated 0.036% offset and show evidence of advanced time anomaly relative to the zero reference line (best possible scenario).
4. In Chart 2, Case Numbers 17, 20, 25, 31, and 35 all deviate from the calculated 0.036% offset and show evidence of retrogressed time anomaly relative to the zero reference line (best possible scenario).
5. The rest of the Case Numbers in Chart 2 shows that the Zenith BIOS anomaly is largely responsible for the time difference below –2 seconds.
6. Based on the above, it follows that two separate problems are being exhibited: 1) the Zenith BIOS conversion algorithm is causing the offset error to occur, and 2) a TD anomaly is also occurring, separate from the Zenith BIOS anomaly.
7. The Zenith BIOS anomaly is *not* the cause of the TD anomaly, since the maximum offset error is no more than 31 seconds on any given day.

The charts and tables speak for themselves. It is impossible to reconcile the large departures from the 0.036% offset theory where they occur.

In conclusion, Intel's test data shows two different anomalies, but Intel's report concludes on only one of them: the BIOS conversion anomaly. Ostensibly, this is why Intel explained the Zenith BIOS anomaly in terms of a *theory* without extracting from their own test data, but yet based on fact.

The fact that TD anomaly shows up only in the time of day and not the date lends supporting evidence that the time and date are read from the RTC *separately* on computer boot: the time from the BIOS POST, and the date from the DOS kernel using BIOS interrupt 1Ah. In fact, Appendix F of Intel's report supports this. The author therefore concludes there is something wrong with the BIOS POST code, but the BIOS code that services interrupt 1Ah is clean.*

Now, to discredit this self-revealing analysis would (theoretically!) require Intel to call attention to the ineptitude of the employee who recorded the test data. During this recording process, this would mean the employee must first record the wristwatch time, then take very long sips of coffee – perhaps finish the whole cup – before setting it down and finally recording the DOS-reported time of day. This would only be evidence of "corporate dilation," not time dilation. But the author has better things to say of the employee who recorded the test data; in fact, he will thank him for it.

* The century year has nothing to do with the time retrieval from the RTC; therefore, neither does the so-called "Logic Path" theory.

The intent is not to cast aspersion, but to preclude Intel from implementing "damage control" and incite them to face up to the facts. After all, they have already presented a *theory*. There is no theory in this analysis up to this point. It is very difficult to disprove an advanced time anomaly, even theoretically.

Final Analysis: The Cause

The characteristics of the Echlin 286 TD anomaly revealed in the foregoing analysis raise serious questions about the validity of the ignored UIP bit as a specific cause of this anomaly on Echlin's 286 CPU #2 board. Even system interrupts as a cause is in question. The reason? The specific nature of the anomaly revealed in the foregoing analysis is too consistent: the seconds of time always appears corrupted in every occurrence of the TD anomaly, while the hours of the day consistently remains valid.

Because the reference seconds of time is unknown in all cases of the TD anomaly, this final analysis is unfortunately difficult and complicated. Only two theoretical causes of this anomaly is presented herein. But first, the caveats regarding unknown reference seconds of time must be explained, as this affects the theoretical causes.

The foregoing analysis shows that all Case Numbers with advanced time anomalies fail to meet the blue plotted line regardless of the wristwatch reference seconds of time, as demonstrated by employing Figure 3 formula. This is true in both Tables 2 and 3 and respective Charts 2 and 3.

The foregoing analysis also shows that, because of Figure 3 formula results, the reference seconds of time is unknown for these Case Numbers. Therefore, the Delta Time between the reference time and actual DOS-reported time is also unknown for these Case Numbers.

But for Case Numbers 21, 26, and 39, Table 3 and respective Chart 3 show they exceed 59 seconds in Delta Time if the reference time were recorded at zero seconds. (The rest of the cases are below the 60-second mark in Chart 3.) This points to corrupted minutes as well as seconds of time in these cases, if true.

However, the excess seconds of time beyond 59 seconds can be eliminated by simply increasing the reference seconds of time so that the Delta Time falls below 59 seconds, because this excess is part of the Delta Time *with respect to the reference time, which is unknown* for all errant Case Numbers cited. If true, this points to only corrupted seconds of time, not the minutes.

Therefore, whether the anomaly demonstrated in the foregoing analysis is the result of corrupted minutes or seconds (or both) depends on the actual reference seconds of time, which cannot be determined. If the test data were to include at least one case in which the Delta Time is at least 2 minutes or more, then a case can be made for the minutes of time as definitely corrupted. But the maximum Delta Time found is only 87 seconds from the zero reference line, or 97 seconds from the calculated blue line. So either the minutes of time or the seconds of time is corrupted, or both, again depending on the actual reference seconds of time.

What is determined, however, is that at least the seconds of time is consistently corrupted whenever a TD anomaly occurs on a random basis. Corruption of the minutes of time does not appear likely in Case Numbers 21, 26, and 39, because Delta Time in all 42 test cases is consistently less than 2 minutes. But it cannot be ruled out without more test data.

With this caveat in mind, we continue in the final analysis.

In a time read procedure, the RTC time retrieval process usually begins with the lowest unit of measure – i.e., seconds, then minutes, then hours. One TD anomaly theory stipulates that the "encroachment" of bad data in the midst of a time read process begins with the corruption of hours, then minutes, then seconds, assuming the read process crosses the boundary into the update period (see Figure 2).* Why, then, does data corruption occur in the seconds (and possibly minutes), and yet never the hours (and possibly the minutes if not corrupted), which is the *opposite direction* of bad data encroachment?

* Actually, it is the shifting of the read process over the boundary that determines its position across it, not the bad data "encroachment" towards the read process. The beginning of bad data is fixed in the update period. Read further into the text in the next couple of pages and you will see why the text is stated this way.

To explain in other words, use Figure 2 to imagine this theoretical scenario: the read process positioned over the start of the update period, half into the update period, half out of the update period. In this case, the seconds portion remains incorrupted, but the hours portion is corrupted. Yet the TD anomaly in Echlin's 286 computer shows the opposite effect: seconds of time consistently corrupted, with the hours consistently incorrupted.

It is not theoretically and statistically possible for the UIP bit to be ignored in such a manner that result in consistent corruption of only the seconds of time. This holds true for system interrupts. Remember that the very nature of RTC access is a random access at any time during a *full second* between increments of RTC seconds of time, and most of this time period (998 mS) is *valid* data. There is no mechanism for read-process positioning over this period except with respect to the UIP bit.

Therefore, the following two theories are the only plausible causes of the anomaly so described, but each one with its own problem that may void the theory.

Theory #1: Bad Data Encroachment

This theory applies only if the time of day were corrupted in the order of seconds, then minutes, then hours. Not all of the seconds, minutes, and hours have to be corrupted together every time a TD anomaly occurs. Only the seconds may be corrupted, or only the seconds and minutes together may be corrupted, or all three units may be corrupted together. It is not possible for only the minutes and hours to be corrupted and yet the seconds remain valid in this theory; data corruption must occur in the order beginning with the seconds of time. This theory assumes that the hours of the day can be corrupted, but this does not show up in the test data.

The only way to account for bad data encroachment in the opposite direction is that the time read procedure is programmed to begin the read process at the *end* of the update period, or at the very *beginning* of the valid data period. The time read procedure would contain special code to wait until the end of the update period, not just when the UIP bit is cleared, before reading the time of day. This means the routine has all of 998 milliseconds in which valid data is guaranteed; therefore the 244-microsecond safe limit window does not apply. (This method allows the time conversion routine to be interposed within the RTC time read routine, which is a good reason for extending the window to 998 milliseconds since the conversion code takes time to execute.)†

† As confirmed through examination of published BIOS code. This code does indeed include code to begin the time retrieval at the beginning of the valid data period. And it also includes the interposed time conversion code.

This scenario appears to fit the characteristics of CPU #2 board TD anomaly. Examine Tables 7 and 8 at the end of Section 5.3.3 for an example. Note the RTC data at Record numbers 1086 and 2859 (under "Record" column). Note the bad data in the "se" and "sa" columns and the status byte in "st" column. The 26h value in the status byte indicates the UIP bit is cleared in the RTC. If the time retrieval code were to read the time information at this point (does not include the time alarm data), the seconds of time will be corrupted, but yet with valid minutes and hours of the day.

Tables 5 and 6 at the end of Section 5.3.3 tells a slightly different story. Note the bad data when the UIP bit clears just after the update. The bad data "encroaches" into the time of the day in different lengths just after each update period. Enter the sometimes corrupted, sometimes incorrupted minutes and/or hours of the day.

All the tables mentioned (Tables 5, 6, 7, and 8) come from the same computer. The author has more data not published herein showing the "encroachment" of bad data into the time of the day at different lengths.‡

While looking at these tables, one caveat needs mentioning. The data in the tables were generated using Echlin's RTC.EXE. This utility reads the status byte last; each line of data reflects the order in which each byte of data is retrieved in Echlin's source code. However, in a typical BIOS code, the status byte is read *first* before accessing the data. In RTC.EXE utility, the status byte is read in the *previous* data line, then the code immediately loops back to read data for the next line. To retrieve time information from the RTC in a computer boot, the status byte needs to be read only once, but it must be read *first* before reading data from the RTC.

This means the status byte in the tables applies to the *next* line of data, not on the same line. (See Section 5.3.2 in the first few paragraphs for explanation.) This impacts this theory, but who is to say that Echlin's computer does not have this problem as described? It only means the author's 286 computer from which the table data was obtained has no such problem.§ What about Echlin's computer? We don't know, but the theory fits well because we know *there is a problem* with Echlin's computer.

‡ The term "encroachment" is properly used here. In this case, the position of the read process is fixed to begin at the first instance of the cleared UIP bit after the update period.

§ The author's 286 computer exhibits *retrogessed* time anomaly, which has not been fully investigated. Echlin's board, however, exhibits *advanced* time anomaly, which is a large departure from the author's own machine in terms of the specific cause. The anomalies between these computers are vastly different.

But there is a problem with this theory. As shown in Tables 1, 2, and 3 in this section (not Section 5.3.3), the hours of the day in all 42 cases consistently remain incorrupted. This may be only because bad data "encroachment" does not go beyond the seconds of time in Echlin's CPU #2 board, or because not enough time was expended to record more data so that corrupted minutes/hours of the day can be captured. There are only 8 out of 42 cases showing TD anomalies, which is not a good ratio to make a safe assumption one way or the other.

Nevertheless, a pattern exists, and this theory fits the observed pattern. This will be expanded a bit further later.

Theory #2: I/O Delay

As opposed to Theory #1, this theory applies if any of the units of time or combination thereof are corrupted in any order. The hours is left out of this theory because the test data shows consistently valid hours of the day. This theory assumes the minutes can be corrupted, but this cannot be proven, as explained previously. If the minutes in fact does not get corrupted, this theory still applies to the seconds of time. Every set of back-to-back RTC instruction to read a byte of data from the RTC requires at least one I/O delay instruction in between.

The I/O delay instruction is missing in back-to-back RTC accesses for the RTC unit of time (seconds or minutes, or both). In this case, the bad data did not result during the update period; rather, it came from I/O conflicts within the RTC address/data bus, which is 8 bits wide.

The lack of I/O delay instruction also partially explains the repeatability, or rather the frequency, of the anomaly. The fact that the unit of time is not *consistently* corrupted on *every* computer boot does not invalidate this I/O delay theory as the cause, because the nature of I/O conflicts on the RTC address/data bus is a *random* one, not a consistent one.

There is a problem with this theory, too. If the time read routine uses a separate sub-routine to read a byte of data from the RTC, this theory is null and void whether that routine contains I/O delay instructions or not. The theory assumes discrete instructions are used instead of calling a sub-routine.*

There are other theories, but unlikely ones:

* Published BIOS code indeed contains a separate sub-routine for reading each byte of data.

1. The time retrieval process is reversed so that the hours is retrieved, then minutes, then seconds. In this case, interposition of conversion code within the RTC read procedure would cause the seconds of time to be corrupted because the overall access time exceeds the 244-microsecond window. This means that the routine does not start at the beginning of the valid data period, but at any point in which the UIP bit just happens to be cleared. But the random nature of this theory makes it an unlikely cause for the anomaly so described, because the hours unit is never corrupted.

2. No bounds checking is performed in the seconds of time in the time retrieval procedure. But this still begs the question as to why the value is more than the maximum value of "59" to begin with. Therefore, this possibility may be at least a contributory cause. If there are indeed bounds checking in the code, it may at least alert the user with an error message that the time and date is not set and advise running CMOS SETUP,† or it may simply continue with the rest of the time retrieval process without user notification.

Final Analysis Conclusion

Ignoring the UIP bit in the RTC cannot be the theoretical cause of the anomaly on Echlin's CPU #2 board. Neither can system interrupts left enabled.

Based on further analysis with available data and plain common sense, bad data encroachment (Theory #1) is believed to be responsible for the anomaly on Echlin's errant 286 CPU #2 board. It took all the resources available to come to this conclusion, short of examining the board itself.

Now is the time to examine Charts 2 and 3 again to understand why Theory #1 is chosen as the theoretical cause of the anomaly.

Consider: the BIOS POST routine reads the seconds of time in BCD format. Assuming the returned value is incorrupted, the BCD seconds of time is converted to its equivalent hexadecimal value before conversion into tickcounts value in binary, but the resulting value is 0.036% less than it should be.

† As confirmed through examination of published BIOS code.

If the returned value is corrupted, the resulting value is also 0.036% less than it should be. The bad data already exists in the RTC registers before they are converted.

All of this means the errant data points in Charts 2 and 3 are all 0.036% less than they should be. It also means that for the purpose of relative comparison, the blue plotted line should be considered the "base line," not the zero reference line.

With this in mind, then, *all errant data points are positive* relative to the blue plotted line. Further, this is a consistent characteristic of the TD anomaly in this case. Theory #1 fits this scenario, while Theory #2 (I/O Delay) does not. Theory #1 guarantees consistent advanced time anomalies, while Theory #2 does not. Why? Because Theory #2 can cause the RTC address to point to some memory address whose data could result in a retrogressed time anomaly, which does not fit the characteristics described.

There is more to be considered. Suppose the value FFh is retrieved from the RTC by the read process. What happens next? Since it is in BCD format, It first gets converted into hexadecimal, assuming the bounds checking routine either does not exist or allows the conversion to go through after some type of passive action and then returns. Then it gets converted to tickcounts, which results in the tickcount value of BC3h. Converting this value to a time format will result in 165.38 seconds, or approximately 2 minutes and 45 seconds. (The effect of the 0.036% offset will hardly change this value – the difference is only 0.06 second.)

We have just shown how such a BCD value in the seconds field can translate into as much as 2 minutes and 45 seconds, theoretically. And this is just from the seconds field of the RTC! It also answers the question earlier posed regarding the minutes of time in Charts 2 and 3: are the RTC minutes of time corrupted? The answer is no, else we would see a much greater time deviation in terms of minutes instead of seconds.

However, the most time deviation from the calculated blue line is 97 seconds (Case Number 21), not 165 seconds. Nevertheless, the data shows strong evidence that a common factor is in play: that the tickcounts is being added extra seconds in the conversion result. It is impossible to suppose the tickcounts value is not expanded, since the DOS TIME command obtains the time of day information *directly from the tickcounts*, and this information was recorded *using this method* in Intel's test data. This means an expanded tickcount as a factor for the TD anomaly cannot be dismissed.

Therefore, the question as to how the FFh BCD value (165d) gets changed to a lower value before (or perhaps during) conversion to tickcounts is left unanswered. It is worth mentioning that further analyses to answer this question does not yield a meaningful conclusion. One of these analyses showed that the time deviation from the base line can all be the same value (38 to 62 seconds) by adjustment of the reference seconds of time in all the cases of the TD anomaly. This is inconclusive because it does not answer the question. It does, however, show that the time deviation can all be the same value, which opens the door to speculation that leads to the answer to this question. But it is only speculation.

First, if the BCD value is greater than 59h (up to FFh), the bounds checking routine in Echlin's POST code *should have stopped the RTC read process*, but it apparently *did not*.‡ This means either the bounds checking routine mishandled a BCD value beyond 59h, or the routine does not exist. In the former case, It may also mean the routine encountered a BCD value of FFh, but mishandled it in a way that resulted in a lower value, and hence a lower conversion result, but is always the same value because the BCD value encountered is always FFh. But in the latter case, the lack of a bounds checking routine may mean the FFh BCD value is mishandled somewhere else, perhaps in the conversion routine itself – either hexadecimal or tickcounts.

Second, the range of 38 to 62 seconds is the range of possible additional time added to the tickcounts value that result in the cases of TD anomaly, but only one of these values is the "smoking gun." If this is true, it means the resulting TD anomaly data points in the chart (2 or 3) will follow the contour of the base line. Here, we finally have a recognizable pattern.

But again, this is pure speculation. We cannot get away from the fact that the tickcounts has somehow been expanded. It appears therefore that there is a bug somewhere in the BIOS code that results in the TD anomaly observed, and that the bug shows itself only when a BCD value of FFh is encountered. It may be that the expanded tickcounts points directly to the bounds checking routine, the error handling routine, the hexadecimal conversion routine, or the tickcounts conversion routine itself.

After all, the BIOS conversion routine used in Echlin's computer is *different* from the one used by DOS. Why argue that the error handling routine is the same as that of the published BIOS code? Or, why call the BIOS code robust if it uses an inaccurate, and thus incorrect, algorithm in the tickcounts conversion routine?

‡ Published BIOS code confirms that its error handling routine stops the process immediately and exits. The resulting time information initialized will be midnight, the default time of day. This means the tickcounts will be initialized to zero and begin counting upwards.

Were it that the resulting time deviation from the base line is 165 seconds or greater, the theory would have been much more solid.

Examination of the published BIOS code does not reveal any conceivable way that the TD anomaly could ever occur.*

Michael Kennedy's work on BCD-to-BIN conversion theory is interesting although it does not take into account the BCD conversion into tickcounts. His experiments are based on a TD anomaly occurring *while the machine is running*, which is a far different premise than the boot-up theory espoused in this report. But it does have merit because a TD anomaly *did* occur.

Kennedy's experiment involved increasing the speed of the timer chip to make the processor run faster, loading a TSR which contains computing code in memory so the processor is constantly busy, and using a separate utility to request the time and date under these conditions. The result is the simulated TD anomaly as shown on Kennedy's web site.

According to the theory, it takes a busy processor to increase the chances of a TD anomaly, and during boot-up the processor is busy. The question is, what is the processor load margin between normal boot up and the *stimulated* occurrence of the TD anomaly in the experiment? Since the timer chip clock speed was increased, the margin may be rather large.

The fact that Kennedy's experiment does not take into account the BCD conversion into tickcounts does not invalidate it, and it may well be possible to fit this theory into the author's own theoretical analysis in this report. The BCD-to-BIN conversion occurs *before* conversion into tickcounts.

* However, if anyone has been mystified by an unexplained error message after a computer boot, such as "Time & Date Not Set – (Run SETUP)" on a 286 computer, the BIOS code may have just read an invalid value, such as FFh, from the RTC. In such a case, the time and date will have been set to default values of midnight, January 1, 1980, according to the code. This anomaly would also typically lead one to a false conclusion that the battery is weak! If such an error message occurs, it does *not* mean the battery is weak. It means the code found an *invalid* time or date value. (After all, how can the code tell if the RTC is not set to the proper time and date, as the error message indicates? The RTC itself is the only sole reference for it!) If the battery is very weak or dead, a different message will be displayed, such as "System Options Not Set – (Run SETUP)."

The caveat, however, is that the BCD-to-BIN routine result is A5h (165d) from the FFh BCD value, not 65d as shown in the results on Kennedy's web site. The 65d is the result of truncation in the display routine, which displays only the last two digits. The conversion into tickcounts is left out of the process. This in itself is not important, but it means the question posed earlier is still not answered: how does the 165d get reduced to a lower value before tickcounts conversion?

Overall, this theoretical analysis is enough to reasonably convince the author that Theory #1 (bad data encroachment) may be the cause of the anomaly on Echlin's computer. Otherwise, the whole theoretical boot-up process will have to be scrapped for some other theory that does not include an "expanded" tickcounts value as a *primary factor*. But Intel has already revealed that the tickcounts is indeed "expanded" in their report, yet without revealing the value. This value could be negative or positive with respect to the zero reference line in the charts. If negative, they were looking at the BIOS conversion anomaly. If positive, they were looking at a genuine TD anomaly. This is another reason Intel never revealed the time difference – because it would also reveal the TD anomaly – which this analysis has proven *exists using Intel's own data* without embellishment.*

Caveat: there is not enough data to make a safe assumption either way; we are talking about only 8 cases out of 42 in Intel's test data. That's 19 percent over a cumulated 2-week test period, but still an impressive percentage considering the rarity of the TD anomaly itself, even on the same computer. There is a fuzzy line drawn between the one side of this analysis which is supported by the data and other analyses and the other side which is mere educated brainy guesswork that is insufficiently supported by the same data. In other words, we've gone as far as we could go.

Only a critical examination of the actual BIOS code will bear light to the actual cause and confirm it. Or deny it. Since the IBM BIOS code was published in 1984 and 1985 (when the 80286 computer was introduced), it is very much conceivable that clone machine manufacturers "cloned" the code but made some changes in it.

One thing is for sure: TD anomaly does exist on Echlin's CPU #2 board, and it is an advanced type of anomaly, which *cannot* be disproved. Also, the errant Case Numbers – showing TD anomaly – *cannot* be reconciled with the blue plotted line in all the plots. Further, this lack of conciliation means the anomaly differs from the true time by more than 59 seconds. Finally, an expanded tickcounts value cannot be dismissed.

* The analysis has shown without a doubt that there is no way to reconcile the errant Case Numbers with the plotted blue line, *regardless of the reference seconds of time*.

The author therefore resting his case, both figuratively and literally, he now calls this TD anomaly the smallest case ever to be analyzed in a report this big, or rather, in this section this large. But it's a necessary work to bring the reader down to the ultimate conclusion, while educating along the way. After all, this analysis is not the final word; someone else may come up with a different theory.* This author does not believe a different theory is even conceivable, however, unless the "expansion" of the tickcounts can be explained and *ruled out*. Since the expansion exists, it cannot be ruled out.

Now that we know the true nature of this particular anomaly on this particular computer, is it really a big deal? No. Will the computer work as normal? Yes. Will time/date-sensitive software work on this computer? Based on the data provided, likely – unless the software is squeamish about the seconds of time, which is unlikely. Does it mean other computers with TD anomaly have the same anomaly characteristics? Unlikely, unless those computers have the same BIOS firmware with the same RTC chips and the same processor. In fact, there will probably be no anomaly at all on most new computers today. Remember, Echlin's computer is an old 286 workhorse based on technologies of the late '80s. That's a long time ago; we have better and faster computers today. After all, it's hard to catch up with technology.

If you are concerned about this anomaly, an easy solution to your peace of mind is to first confirm the new computer you are considering contains 100% buffered RTC (no bad data at all in the update period), then buy it. The author's Pentium laptop has a 100% buffered RTC chip and is completely immune to the TD anomaly. It was actually a free laptop (IBM ThinkPad 365XD), since the older 386 laptop could not be repaired under extended warranty, and the new laptop was obtained *in 1998,* before the author went to Indonesia with it.

New computers with 100% buffered RTC chips are *available*. You may be obliged to count on it, especially since the TD anomaly fiasco is now almost certainly known to computer manufacturers by this time.

---

* Which the author encourages. It is a way of putting the theory to the test, short of examining the BIOS code itself.

There is a typographical discrepancy, albeit small, regarding the speed of Echlin's Zenith 286. In the summary table in Appendix C, Zenith CPU boards #1 and #2 are listed as 8 MHz.[38] However, in the same appendix, CPU #2 board is listed as "Echlin's Zenith 286-12." [39] Also, in Appendix E, Mr. Echlin's Zenith CPU #2 is described as a "12 MHz 286." [40] It is believed it must be a 12 MHz board.

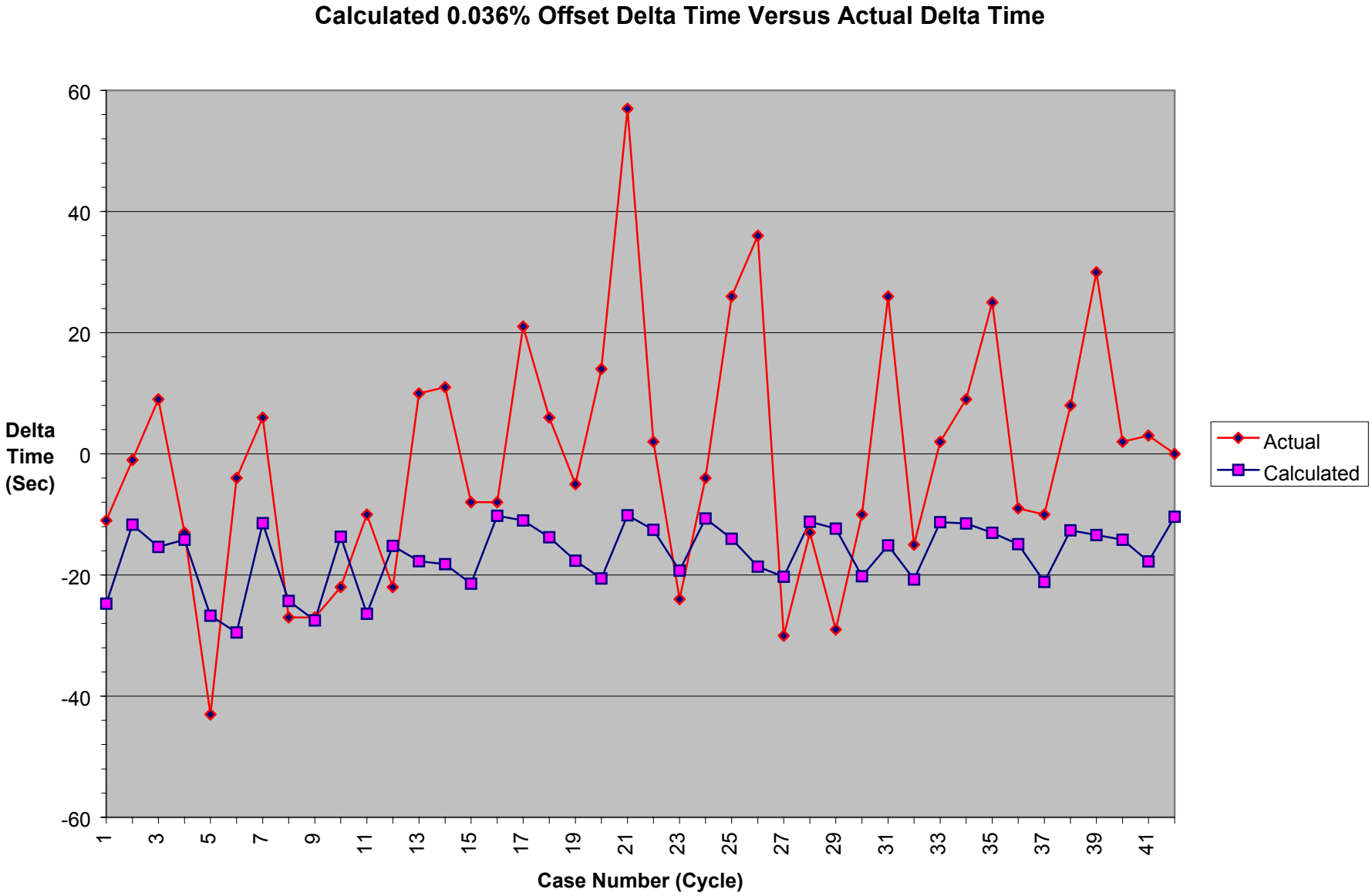


CHART 1: Test Data Results based on wristwatch reference set to 30 seconds of time

Case Number (Cycle)	Wristwatch Ref Date/Time (mo/dy/yr hr:mn:se)	Display Date/Time (mo/dy/yr hr:mn:se)	Wristwatch Time Value (TIMEVALUE)	Display Time Value (TIMEVALUE)	Delta Time (Sec)	0.036% Ref Offset (Sec)
1	11/25/98 19:05:30	11/25/00 19:05:19	0.795486111	0.795358796	-11	-24.74280
2	11/26/98 09:01:30	11/26/00 09:01:29	0.376041667	0.376030093	-1	-11.69640
3	11/26/98 11:51:30	11/26/00 11:51:39	0.494097222	0.494201389	9	-15.36840
4	11/27/98 10:56:30	11/27/00 10:56:17	0.455902778	0.455752315	-13	-14.18040
5	11/27/98 20:36:30	11/27/00 20:35:47	0.858680556	0.858182870	-43	-26.70840
6	11/27/98 22:45:30	11/27/00 22:45:26	0.948263889	0.948217593	-4	-29.49480
7	11/28/98 08:49:30	11/28/00 08:49:36	0.367708333	0.367777778	6	-11.43720
8	11/28/98 18:43:30	11/28/00 18:43:03	0.780208333	0.779895833	-27	-24.26760
9	11/28/98 21:12:30	11/28/00 21:12:03	0.883680556	0.883368056	-27	-27.48600
10	11/29/98 10:33:30	11/29/00 10:33:08	0.439930556	0.439675926	-22	-13.68360
11	11/29/98 20:22:30	11/29/00 20:22:20	0.848958333	0.848842593	-10	-26.40600
12	11/30/98 11:43:30	11/30/00 11:43:08	0.488541667	0.488287037	-22	-15.19560
13	11/30/98 13:39:30	11/30/00 13:39:40	0.569097222	0.569212963	10	-17.70120
14	11/30/98 14:02:30	11/30/00 14:02:41	0.585069444	0.585196759	11	-18.19800
15	11/30/98 16:32:30	11/30/00 16:32:22	0.689236111	0.689143519	-8	-21.43800
16	12/01/98 07:53:30	12/01/00 07:53:22	0.328819444	0.328726852	-8	-10.22760
17	12/01/98 08:29:30	12/01/00 08:29:51	0.353819444	0.354062500	21	-11.00520
18	12/01/98 10:36:30	12/01/00 10:36:36	0.442013889	0.442083333	6	-13.74840
19	12/01/98 13:36:30	12/01/00 13:36:25	0.567013889	0.566956019	-5	-17.63640
20	12/01/98 15:51:30	12/01/00 15:51:44	0.660763889	0.660925926	14	-20.55240
21	12/02/98 07:50:30	12/02/00 07:51:27	0.326736111	0.327395833	57	-10.16280
22	12/02/98 09:40:30	12/02/00 09:40:32	0.403125000	0.403148148	2	-12.53880
23	12/02/98 14:51:30	12/02/00 14:51:06	0.619097222	0.618819444	-24	-19.25640
24	12/03/98 08:13:30	12/03/00 08:13:26	0.342708333	0.342662037	-4	-10.65960
25	12/03/98 10:49:30	12/03/00 10:49:56	0.451041667	0.451342593	26	-14.02920
26	12/03/98 14:21:30	12/03/00 14:22:06	0.598263889	0.598680556	36	-18.60840
27	12/03/98 15:39:30	12/03/00 15:39:00	0.652430556	0.652083333	-30	-20.29320
28	12/04/98 08:39:30	12/04/00 08:39:17	0.360763889	0.360613426	-13	-11.22120
29	12/04/98 09:31:30	12/04/00 09:31:01	0.396875000	0.396539352	-29	-12.34440
30	12/04/98 15:35:30	12/04/00 15:35:20	0.649652778	0.649537037	-10	-20.20680
31	12/07/98 11:40:30	12/07/00 11:40:56	0.486458333	0.486759259	26	-15.13080
32	12/07/98 16:00:30	12/07/00 16:00:15	0.667013889	0.666840278	-15	-20.74680
33	12/08/98 08:42:30	12/08/00 08:42:32	0.362847222	0.362870370	2	-11.28600
34	12/14/98 08:52:30	12/14/00 08:52:39	0.369791667	0.369895833	9	-11.50200
35	12/15/98 10:03:30	12/15/00 10:03:55	0.419097222	0.419386574	25	-13.03560
36	12/15/98 11:30:30	12/15/00 11:30:21	0.479513889	0.479409722	-9	-14.91480
37	12/15/98 16:18:30	12/15/00 16:18:20	0.679513889	0.679398148	-10	-21.13560
38	12/16/98 09:44:30	12/16/00 09:44:38	0.405902778	0.405995370	8	-12.62520
39	12/16/98 10:20:30	12/16/00 10:21:00	0.430902778	0.431250000	30	-13.40280
40	12/16/98 10:55:30	12/16/00 10:55:32	0.455208333	0.455231481	2	-14.15880
41	12/16/98 13:42:30	12/16/00 13:42:33	0.571180556	0.571215278	3	-17.76600
42	01/04/00 08:00:30	01/04/01 08:00:30	0.333680556	0.333680556	0	-10.37880

TABLE 1: Test Data with Delta Time and 0.036% Offset Calculation

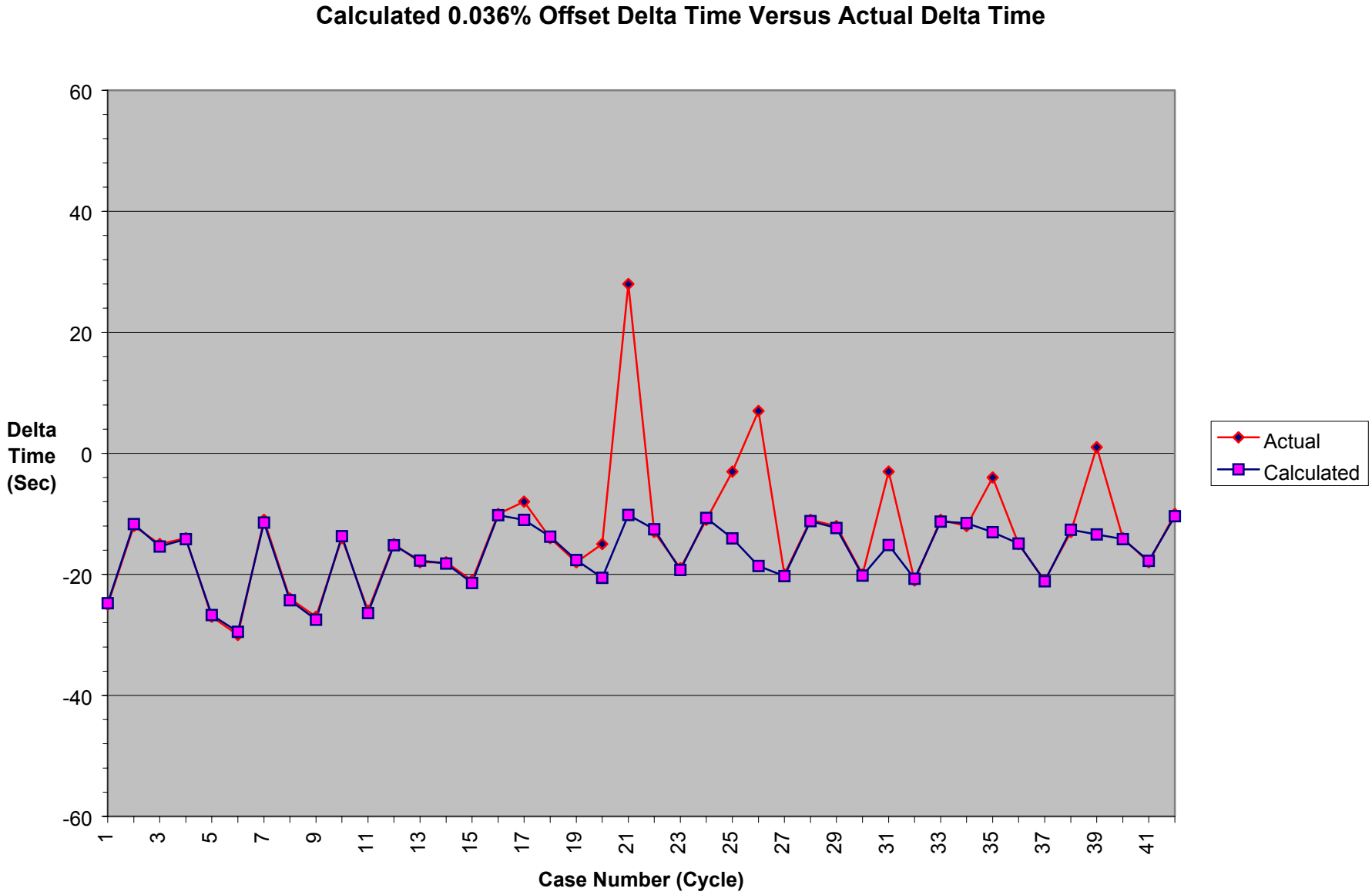


CHART 2: Adjusted Test Data Results based on reconstructed wristwatch seconds of time, Best Case Scenario

Case Number (Cycle)	Wristwatch Ref Date/Time (mo/dy/yr hr:mn:se)	Display Date/Time (mo/dy/yr hr:mn:se)	Wristwatch Time Value (TIMEVALUE)	Display Time Value (TIMEVALUE)	Delta Time (Sec)	0.036% Ref Offset (Sec)
1	11/25/98 19:05:44	11/25/00 19:05:19	0.795648148	0.795358796	-25	-24.74784
2	11/26/98 09:01:41	11/26/00 09:01:29	0.376168981	0.376030093	-12	-11.70036
3	11/26/98 11:51:54	11/26/00 11:51:39	0.494375000	0.494201389	-15	-15.37704
4	11/27/98 10:56:31	11/27/00 10:56:17	0.455914352	0.455752315	-14	-14.18076
5	11/27/98 20:36:14	11/27/00 20:35:47	0.858495370	0.858182870	-27	-26.70264
6	11/27/98 22:45:56	11/27/00 22:45:26	0.948564815	0.948217593	-30	-29.50416
7	11/28/98 08:49:47	11/28/00 08:49:36	0.367905093	0.367777778	-11	-11.44332
8	11/28/98 18:43:27	11/28/00 18:43:03	0.780173611	0.779895833	-24	-24.26652
9	11/28/98 21:12:30	11/28/00 21:12:03	0.883680556	0.883368056	-27	-27.48600
10	11/29/98 10:33:22	11/29/00 10:33:08	0.439837963	0.439675926	-14	-13.68072
11	11/29/98 20:22:46	11/29/00 20:22:20	0.849143519	0.848842593	-26	-26.41176
12	11/30/98 11:43:23	11/30/00 11:43:08	0.488460648	0.488287037	-15	-15.19308
13	11/30/98 13:39:58	11/30/00 13:39:40	0.569421296	0.569212963	-18	-17.71128
14	11/30/98 14:02:59	11/30/00 14:02:41	0.585405093	0.585196759	-18	-18.20844
15	11/30/98 16:32:43	11/30/00 16:32:22	0.689386574	0.689143519	-21	-21.44268
16	12/01/98 07:53:32	12/01/00 07:53:22	0.328842593	0.328726852	-10	-10.22832
17	12/01/98 08:29:59	12/01/00 08:29:51	0.354155093	0.354062500	-8	-11.01564
18	12/01/98 10:36:50	12/01/00 10:36:36	0.442245370	0.442083333	-14	-13.75560
19	12/01/98 13:36:43	12/01/00 13:36:25	0.567164352	0.566956019	-18	-17.64108
20	12/01/98 15:51:59	12/01/00 15:51:44	0.661099537	0.660925926	-15	-20.56284
21	12/02/98 07:50:59	12/02/00 07:51:27	0.327071759	0.327395833	28	-10.17324
22	12/02/98 09:40:45	12/02/00 09:40:32	0.403298611	0.403148148	-13	-12.54420
23	12/02/98 14:51:25	12/02/00 14:51:06	0.619039352	0.618819444	-19	-19.25460
24	12/03/98 08:13:37	12/03/00 08:13:26	0.342789352	0.342662037	-11	-10.66212
25	12/03/98 10:49:59	12/03/00 10:49:56	0.451377315	0.451342593	-3	-14.03964
26	12/03/98 14:21:59	12/03/00 14:22:06	0.598599537	0.598680556	7	-18.61884
27	12/03/98 15:39:20	12/03/00 15:39:00	0.652314815	0.652083333	-20	-20.28960
28	12/04/98 08:39:28	12/04/00 08:39:17	0.360740741	0.360613426	-11	-11.22048
29	12/04/98 09:31:13	12/04/00 09:31:01	0.396678241	0.396539352	-12	-12.33828
30	12/04/98 15:35:40	12/04/00 15:35:20	0.649768519	0.649537037	-20	-20.21040
31	12/07/98 11:40:59	12/07/00 11:40:56	0.486793981	0.486759259	-3	-15.14124
32	12/07/98 16:00:36	12/07/00 16:00:15	0.667083333	0.666840278	-21	-20.74896
33	12/08/98 08:42:43	12/08/00 08:42:32	0.362997685	0.362870370	-11	-11.29068
34	12/14/98 08:52:51	12/14/00 08:52:39	0.370034722	0.369895833	-12	-11.50956
35	12/15/98 10:03:59	12/15/00 10:03:55	0.419432870	0.419386574	-4	-13.04604
36	12/15/98 11:30:36	12/15/00 11:30:21	0.479583333	0.479409722	-15	-14.91696
37	12/15/98 16:18:41	12/15/00 16:18:20	0.679641204	0.679398148	-21	-21.13956
38	12/16/98 09:44:51	12/16/00 09:44:38	0.406145833	0.405995370	-13	-12.63276
39	12/16/98 10:20:59	12/16/00 10:21:00	0.431238426	0.431250000	1	-13.41324
40	12/16/98 10:55:46	12/16/00 10:55:32	0.455393519	0.455231481	-14	-14.16456
41	12/16/98 13:42:51	12/16/00 13:42:33	0.571423611	0.571215278	-18	-17.77356
42	01/04/00 08:00:40	01/04/01 08:00:30	0.333796296	0.333680556	-10	-10.38240

TABLE 2: Adjusted Test Data, Best Case Scenario

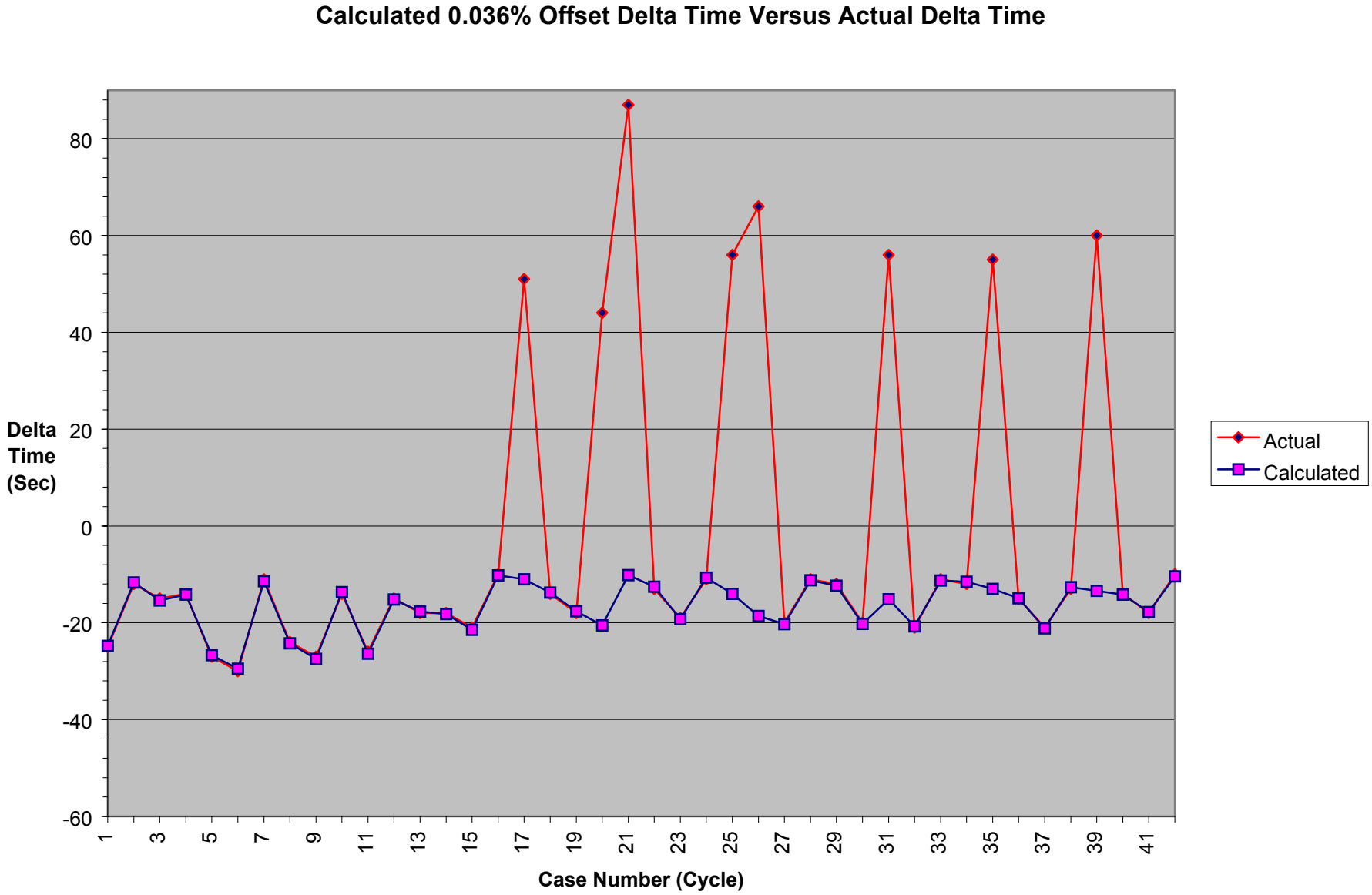


CHART 3: Adjusted Test Data Results based on reconstructed wristwatch seconds of time, Worst Case Scenario

Case Number (Cycle)	Wristwatch Ref Date/Time (mo/dy/yr hr:mn:se)	Display Date/Time (mo/dy/yr hr:mn:se)	Wristwatch Time Value (TIMEVALUE)	Display Time Value (TIMEVALUE)	Delta Time (Sec)	0.036% Ref Offset (Sec)
1	11/25/98 19:05:44	11/25/00 19:05:19	0.795648148	0.795358796	-25	-24.74784
2	11/26/98 09:01:41	11/26/00 09:01:29	0.376168981	0.376030093	-12	-11.70036
3	11/26/98 11:51:54	11/26/00 11:51:39	0.494375000	0.494201389	-15	-15.37704
4	11/27/98 10:56:31	11/27/00 10:56:17	0.455914352	0.455752315	-14	-14.18076
5	11/27/98 20:36:14	11/27/00 20:35:47	0.858495370	0.858182870	-27	-26.70264
6	11/27/98 22:45:56	11/27/00 22:45:26	0.948564815	0.948217593	-30	-29.50416
7	11/28/98 08:49:47	11/28/00 08:49:36	0.367905093	0.367777778	-11	-11.44332
8	11/28/98 18:43:27	11/28/00 18:43:03	0.780173611	0.779895833	-24	-24.26652
9	11/28/98 21:12:30	11/28/00 21:12:03	0.883680556	0.883368056	-27	-27.48600
10	11/29/98 10:33:22	11/29/00 10:33:08	0.439837963	0.439675926	-14	-13.68072
11	11/29/98 20:22:46	11/29/00 20:22:20	0.849143519	0.848842593	-26	-26.41176
12	11/30/98 11:43:23	11/30/00 11:43:08	0.488460648	0.488287037	-15	-15.19308
13	11/30/98 13:39:58	11/30/00 13:39:40	0.569421296	0.569212963	-18	-17.71128
14	11/30/98 14:02:59	11/30/00 14:02:41	0.585405093	0.585196759	-18	-18.20844
15	11/30/98 16:32:43	11/30/00 16:32:22	0.689386574	0.689143519	-21	-21.44268
16	12/01/98 07:53:32	12/01/00 07:53:22	0.328842593	0.328726852	-10	-10.22832
17	12/01/98 08:29:00	12/01/00 08:29:51	0.353472222	0.354062500	51	-10.99440
18	12/01/98 10:36:50	12/01/00 10:36:36	0.442245370	0.442083333	-14	-13.75560
19	12/01/98 13:36:43	12/01/00 13:36:25	0.567164352	0.566956019	-18	-17.64108
20	12/01/98 15:51:00	12/01/00 15:51:44	0.660416667	0.660925926	44	-20.54160
21	12/02/98 07:50:00	12/02/00 07:51:27	0.326388889	0.327395833	87	-10.15200
22	12/02/98 09:40:45	12/02/00 09:40:32	0.403298611	0.403148148	-13	-12.54420
23	12/02/98 14:51:25	12/02/00 14:51:06	0.619039352	0.618819444	-19	-19.25460
24	12/03/98 08:13:37	12/03/00 08:13:26	0.342789352	0.342662037	-11	-10.66212
25	12/03/98 10:49:00	12/03/00 10:49:56	0.450694444	0.451342593	56	-14.01840
26	12/03/98 14:21:00	12/03/00 14:22:06	0.597916667	0.598680556	66	-18.59760
27	12/03/98 15:39:20	12/03/00 15:39:00	0.652314815	0.652083333	-20	-20.28960
28	12/04/98 08:39:28	12/04/00 08:39:17	0.360740741	0.360613426	-11	-11.22048
29	12/04/98 09:31:13	12/04/00 09:31:01	0.396678241	0.396539352	-12	-12.33828
30	12/04/98 15:35:40	12/04/00 15:35:20	0.649768519	0.649537037	-20	-20.21040
31	12/07/98 11:40:00	12/07/00 11:40:56	0.486111111	0.486759259	56	-15.12000
32	12/07/98 16:00:36	12/07/00 16:00:15	0.667083333	0.666840278	-21	-20.74896
33	12/08/98 08:42:43	12/08/00 08:42:32	0.362997685	0.362870370	-11	-11.29068
34	12/14/98 08:52:51	12/14/00 08:52:39	0.370034722	0.369895833	-12	-11.50956
35	12/15/98 10:03:00	12/15/00 10:03:55	0.418750000	0.419386574	55	-13.02480
36	12/15/98 11:30:36	12/15/00 11:30:21	0.479583333	0.479409722	-15	-14.91696
37	12/15/98 16:18:41	12/15/00 16:18:20	0.679641204	0.679398148	-21	-21.13956
38	12/16/98 09:44:51	12/16/00 09:44:38	0.406145833	0.405995370	-13	-12.63276
39	12/16/98 10:20:00	12/16/00 10:21:00	0.430555556	0.431250000	60	-13.39200
40	12/16/98 10:55:46	12/16/00 10:55:32	0.455393519	0.455231481	-14	-14.16456
41	12/16/98 13:42:51	12/16/00 13:42:33	0.571423611	0.571215278	-18	-17.77356
42	01/04/00 08:00:40	01/04/01 08:00:30	0.333796296	0.333680556	-10	-10.38240

TABLE 3: Adjusted Test Data, Worst Case Scenario

5.2 A Bug in Intel's Whitney Chipset

As if to confirm what looks like a possible case for a TD anomaly, an Intel support engineer reported a bug in Intel's Whitney chipset. Here is the technical news report from the Techweb web site, reported in July of the year 1999:

> *Intel Corp. has identified a bug in its Intel 810, or Whitney, chipset, involving an error with the real-time clock (RTC). In an Intel 810 hardware-support forum at the company's Web site, an Intel support engineer stated: "The RTC may inaccurately report that the clock is not busy when it is actually in the process of updating internal registers. This may result in invalid data being read in the date and time fields of the real-time clock, but the erratum does not modify the actual RTC values." A software workaround has been identified.* [41]

The release date of Intel's Analysis of the Crouch-Echlin Effect report is 3 February 1999. This news report came a few months later.

Here is an excerpt of another source: [42]

Intel issues fix for another 810 chip set glitch

By John G. Spooner, PC Week Online
July 1, 1999 7:46 PM ET

It's not good timing.

Intel Corp. confirmed today it has delivered to OEMs a fix for another glitch with its 810 chip set for Celeron-based PCs.

The glitch -- or erratum, as Intel calls it -- can cause the chip set's real-time clock to display incorrect date and time, officials said.

The real-time clock, which is a part of the chip set, updates the time once per second, but during the update alerts the system that it is doing so. Because of the erratum, a signal that is supposed to be sent from the clock alerting the rest of the system it is busy may not get sent. As a result, incorrect data can be displayed.

"If it occurs, it may result in invalid data being read in the date and time display fields of the real-time clock," said Intel spokesman Dan Francisco. "However, the actual values themselves wouldn't change." Some applications get their time information from the real-time clock, which also supplies timing information to a PC's BIOS software.

According to the “erratum” description, the RTC data tends to be accessed for time/date information while the RTC is still in update status. It is not the fault of the RTC hardware, but the fault of the chipset firmware.

If any person would like to see a “documented TD anomaly” written by a reputable source, this excerpt would be called the closest and the best evidence that anyone could get in the public realm. But this is an isolated case because the chipset is not old technology – and the TD anomaly tends to be more populous on older computers rather than new.

However, the description of the anomaly itself indicates the RTC probably uses non-buffered registers. In this case, a non-buffered RTC is indeed old technology, but encased in new technology in terms of VLSI packaging. In any case, the erratum described points to the BIOS firmware.

This chipset is used in Celeron-based systems – systems with the Mendocino processor, or Celeron A as is better known. The chipset was introduced on 26 April 1999 with the 466 MHz version of the Celeron processor. The Celeron A processor is a much-improved version over the earlier Celeron processor – because it contains a level 2 cache embedded within the processor core, thus greatly improving speed performance. In fact, it has a speed advantage over other competitor products.*

The chipset consists of three chips designed and highly optimized for use with the Celeron A processor.

A firmware upgrade has already been issued to OEMs to fix this bug, and by now this fix has already been implemented. It is now a non-issue.

* At the time of its release. The processor speed has since been surpassed.

5.3 Echlin's TD Theory

5.3.1 Logic Path

Mr. Mike Echlin wrote on his web page:

> *The reason that there is a different logic path after year 2000 is because of the way the PC keeps the date, and the way it reads the date from the RTC. The RTC has second, minute, hour, day, month, year, but the PC keeps it as clock ticks since midnight, and days since 01.01.1980. So, before year 2000, you just subtract 80 from the current year, and adjust for leap years, and count the days in the current year. After year 2000, you have to adjust for the new century, and count the years since rollover, and the days in the current year, and the extra days in leap years, and calculate whether or not year 2000 is a leap year. And on top of all this, you have to convert from BCD* [43]

The culprit Mike Echlin refers to in the above excerpt is the date conversion algorithm that has two logic paths based on the century year value: 19 for one logic path, 20 for the other path. In a computer boot, the time conversion algorithm is performed by the BIOS POST, and the date conversion algorithm is performed by the DOS kernel code. Here, the excerpt refers only to the date.

Put another way, during computer boot the time data is retrieved from the RTC first, then the date is retrieved from the RTC after the DOS kernel is loaded in memory. By the time the date is retrieved, POST has long been completed.* This is a typical process of system time and date initialization, in that order, regardless of the BIOS version or manufacture in use.†

This excerpt is based on the assumption that these calculations are all done *"on the fly"* while reading the RTC data. But the date conversion occurs *after* complete date information is retrieved from the RTC. How, then, does the date conversion routine (allegedly including dual logic paths) push the date retrieval period into the *update period?* This is a statement, not a question. In order to perform date conversion into a *singular* number of days in hex, *all* of the date information, *including the century byte*, must be retrieved *first*.

---

* This is why the TD anomaly occurs only in the date, or only in the time, with the other respective time or date being correct.

† As confirmed through examination of published BIOS code and supported by DOS technical references. Because the DOS kernel handles both the date retrieval (through BIOS Interrupt 1Ah) and conversion to binary value in days since January 1, 1980, the BIOS POST routine must leave the RTC date handling process up to DOS.

Since it is the DOS kernel code that performs date conversion from BCD to binary, *there is no interposition of conversion code into the BIOS code* that reads the date from the RTC.‡ In fact, the RTC date data returned to the DOS kernel is in packed BCD format, the same format used in the RTC registers. Therefore, all RTC read operations for the date are completed before the conversion occurs. Hence, *there is no "logic path" to speak of while reading the RTC for the date*. As a result, Mr. Echlin's Logic Path theory falls flat to the ground.

This is why Intel's report in Appendix C makes the same conclusion.[44] If there is a logic path based on the century year value, it is handled in the DOS kernel *after*, not *while*, reading the RTC for the date. Hence, the Logic Path theory as a cause of a TD anomaly falls apart.

In order to retrieve time/date information from the RTC, *multiple* reads are necessary. This is because each byte of this information is stored in different addresses in CMOS memory. Each byte requires a set of two instructions to read it: one to set the address, one to read from that address. Ignoring the RTC alarm data, reading the complete time/date information (including century) requires 7 sets of these instructions: 3 sets for the time, 4 for the date.

Some BIOS POST code does contain conversion code for the time of day, as opposed to the date retrieval code in the BIOS Interrupt 1Ah services. If these instructions are interposed between individual byte reads for the time, the aggregate RTC read access time would variously increase because these instructions take time to execute. This would be grounds for exceeding the 244-microsecond window, if indeed the aggregate access time exceeds it. (The century year byte has nothing to do with the time conversion.)

However, the 244-microsecond limit does not *have* to be met if the UIP bit is *properly* handled. Remember that we have all of 998 milliseconds (998,000 microseconds) in which the RTC data is valid (see Figures 1 and 2 in Section 4.0). If the RTC time read procedure is begun at the point the UIP bit falls to zero (after the seconds of time rollover), the 244-microsecond limit does not have to be honored. As a result, the entire time-of-day retrieval procedure in the BIOS POST has much more than enough time to read and convert the time into binary and store the resulting timer ticks value in the BDA.*

‡ As confirmed through examination of published BIOS code. The date retrieval code does *not* have a date conversion routine.

* As confirmed through examination of published BIOS code. The time read procedure begins immediately after the UIP bit clears.

Therefore, whether the TD anomaly occurs in the time of day depends on how the BIOS code handles the UIP bit. But there is another point to be made.

The conversion code does not *have* to be interposed between RTC read instructions as described. It can be inserted after the section containing code that reads the entire time information. In other words, the conversion routine can be performed after all RTC read operations are completed. But this method requires temporary memory storage of the time in BCD format. Since it is undesirable to use temporary memory storage that will later never be used, the time of day in BCD format could be stored in the same memory location for the timer ticks, theoretically. Then the time conversion routine reads this information from that memory location, converts it into timer ticks, and stores it back in the same location.†

As far as embedded systems are concerned, Mr. Patrick Bossert contends in the Bossert/Echlin "dialogue" (see Section 4.1) that he has never seen a single embedded system controller which does not read RTC date information consecutively and do any date conversion math while RTC read operations are still in progress. In his opinion such a date conversion in the middle of a read sequence is very unlikely.

Mr. Bossert's contention is essentially correct, since the RTC date information is retrieved in the form of packed BCD format before being converted not by the BIOS, but by the DOS kernel, during computer boot.

Mr. Bossert states that "ignoring the UIP bit is clearly just bad programming. You would expect a PC BIOS which does this to have the occasional glitch in reading the clock on start-up. *This makes the problem as likely to occur both pre and post-millennium*."

Mr. Bossert is again correct, and once again, this nullifies the logic path theory.

Since the date conversion routine is ruled out as a contributory factor to the TD anomaly, a TD condition would indicate strongly that the UIP bit is not being honored by the code. If this is the case, then the "map of a second" figure 3 shown on Crouch's web site[45] is inaccurate. In figure 3 cited, RTC reads begin before the 244-microsecond window begins and then ends after that window has expired. That the series of RTC reads for the date would last longer than 244 microseconds is highly unlikely based on this analysis. If the UIP bit is ignored, there is *no need to extend the access time* beyond 244 microseconds to read bad data.

† Whether any BIOS version of any manufacture uses this method is unknown.

One of the issues discussed in the "dialogue" is the access time of the BIOS ROM as opposed to system RAM. Mr. Mike Echlin contends that ROM access time is much slower than RAM. For an IBM AT 286, this is incorrect. In fact, the IBM Technical Reference for the AT 286 computer (1985) specifies the ROM access time as the same as that of RAM (150 nanoseconds). It is true that ROM is generally slower than RAM on faster computers, but here, we are addressing 286 computers.

On an AT system, ROM chips are connected to the CPU system bus, the same bus the processor chip and RAM chips are connected to. This is a 16-bit bus (*not* 8-bit, as Echlin contends). In addition, the ROM chips are connected to both the 16-bit address bus and the 16-bit data bus.*

As a result, the CPU has twice as much ROM BIOS code to execute than from an 8-bit system bus, which is present only on 8088/8086 system. This makes the ROM BIOS code execute faster, does it not? Yes, of course.

By the time a single beep sound is emitted on computer boot, the POST routine has just been successfully completed, and protected-mode tests have also been successfully completed. *It is at this time that the time of day information is retrieved from the RTC, converted, and stored in the BDA.* After this, the bootstrap loader is executed next to start the operating system itself.† Hence, the time of day retrieval procedure is near the end of the POST routine. At this point, all of the system devices and attached adapter cards (if any) have been initialized.

At this time, the *date* information has not even been retrieved from the RTC yet. This is done by the DOS kernel after it is loaded and initialized. When this is done, how fast do you think the system is running? *The ROM BIOS code is no longer involved, much less the ROM chips themselves.* Since the DOS kernel is loaded in RAM, the date retrieval and conversion code is executed in *RAM*, not ROM.‡

---

* On old systems such as this, two ROM chips are present: an EVEN chip, and an ODD chip. Both chips are connected to the address bus in parallel, but one chip is connected to the first 8 bits of the data bus and the other is connected to the last 8 bits of the data bus. Each ROM chip is a 32K x 8 bit EPROM, but they are arranged as 32K x 16-bit as a single entity – hence, a 16-bit device. The ODD and EVEN chips contain code in odd and even addresses; this is why they are connected to the address bus in parallel.

† As confirmed through examination of published BIOS code.

‡ The DOS kernel calls BIOS Interrupt 1Ah function 04h, available only on PC/AT systems, to get the date from the RTC. This BIOS service is executed in ROM through interrupt vectoring and the date information is returned to the caller, which is running in RAM. The date conversion routine is performed in RAM. The DOS kernel does *not* use DOS Interrupt 21h function 2Ah to get the date during computer boot.

On AT, PC, and PS/2 systems, the entire 64K of system ROM is copied into RAM at address FF0000 (accomplished during protected mode). Another duplicate copy is made at address 0F0000 on some systems.§ In addition, some ROM routines and tables are copied to low RAM. This allows a patch to be applied, if necessary, to the code in low RAM, since the patch cannot be applied in the ROM chip. The patch is usually implemented by DOS after completion of POST. This method has been in effect since version 3.0. After the patch is implemented, DOS refers to the patched information in RAM instead of ROM.

So far, the RTC alarm data for the time and date has not been addressed. The BIOS POST routine does not convert these or read them.* Instead, interrupt 4Ah vectored to a user-supplied alarm-handler is used to monitor the RTC alarm status once it is set up using BIOS interrupt 1Ah function 6. This would be handled by an application started by the user after system boot.

Hence, another factor affecting the RTC access time is the fact that the POST code does not read the the alarm data. In all RTC data excerpts given by the author and Echlin, this alarm data is included. Removing instructions to read them makes the aggregate access time shorter. But this does not help Echlin's theory because it is based on longer RTC access time.

The excerpt cited in this section mentions the need to "adjust for leap years." It insinuates that the RTC does not "adjust for leap years" and thus does not keep the correct date. In fact, RTC technical data sheets list "Automatic leap year compensation" as one of the long list of RTC chip features. Why adjust for leap years in the conversion code, then? Because leap year days are hidden between 1 January 1980 and the current date, and the RTC does not provide this information. It needs to be compensated for in the conversion routine.

Another excerpt says this:

> *...this difference in logic path changes the amount of time used to read the RTC enough to allow it to still be reading the RTC while it is in its update mode, and if the RTC is not buffered, then the value being read from the RTC at that time is not reliable, and can cause the effect to occur.*[46]

§ IBM Technical Reference, 1985; The Programmer's PC Sourcebook, 2nd Ed.

* As confirmed through examination of published BIOS code.

But Intel's investigation into the alleged "difference in logic paths" changing the RTC access time based on the century byte concludes that there is no difference in the logic paths.[47] Intel's test data in Intel's report also failed to confirm this claim (the year date was set to 2000).[48]

In Jace Crouch's public rebuttal, the stated reason Intel claims there is no difference in the logic paths is because they examined the wrong section of the BIOS code, despite Mike Echlin's suggestion to examine a particular section of the POST code.[49] In the POST code, there is only the RTC *time* read and conversion procedure. The RTC *date* read procedure is in the BIOS services section and not in the POST code. From this, it is inferred that Mr. Echlin wanted to draw attention to the time conversion routine, which is located in the POST section.

But Intel's examination of the BIOS section is reasonable and justified. It is the date conversion routine that includes the century byte, not the time conversion routine; Intel examined it in order to verify whether or not a dual logic path exists. We now know that it does not exist. Intel did not even have to visually examine the running code to make their determination; only a spattering of logic is all that is needed to realize that all of the date information must be read from the RTC *before date conversion can begin*.

Let us examine this for a moment. The BIOS service routine to get the date (Interrupt 1Ah, function 04h) does *not* contain a date conversion routine, because it returns only BCD numbers directly from the RTC. All 4 bytes of the date (month, day, low-order year, high-order century year) are returned to the caller. This being the case, the DOS kernel gets these numbers in BCD format, and *then* date conversion is performed.*

In Hypothesis #1 section of Intel's report, it makes no mention which board was examined, but it did mention the phrase "Echlin's machines" in the plural sense.[50]

* In fact, the BIOS interrupt 1Ah function to returns *only* BCD numbers as documented in programmer technical references. It is as simple as that.

5.3.1.1 Binary Coded Decimal Primer

This subsection details the definition of a Binary Coded Decimal format. It is a common format that allows packing of a two-digit decimal into single byte of 8 bits each. The leftmost digit is stored in the upper half of the byte and the rightmost digit is stored in the lower half. This is called a packed BCD format. Unpacked BCD format stores one digit decimal, instead of two decimals, into a single byte, but only the lower 4 bits are used to represent the decimal digit.

The reason this format was chosen was to avoid a more complex conversion routine. (The alternative format is simply hex numbers – more lines of code are required for conversions to/from hex.)

For example, the decimal number 98 is stored in BCD as follows:

"9" stored in the upper half of a byte as 1001 in binary
"8" stored in the lower half of the same byte as 1000 in binary

…thus producing 10011000 in binary in a single byte. Note that this byte is represented in hex as 98, but this is not a hex number. It is a binary coded decimal number, the keyword being the word "decimal." The 2-digit decimal number is stored in the byte as if it is a hex number, but without first converting it into hex. If this number were a hex number to be converted into decimal, the result would be 152 in decimal, not 98. Similarly, if the decimal number were to be converted into hex, the equivalent number would be 62 in hex.

This exercise in mathematics is intended to show that using BCD already eliminates half the work of conversion to/from hex; therefore, less code is required. This is why BCD was the chosen data format for the RTC. (The RTC can be programmed for either BCD or hex format – but is always programmed for BCD format.)

5.3.2 Introduction to RTC Data

RTC data dump excerpts are provided in Tables 1 through 9 at the end of Section 5.3.3.4.

All data dump excerpts are from the author's RTCSCAN program, which is programmed to read 20,000+ lines of RTC data from a text file containing dumped RTC data and scan it for several types of errors. It is designed to read the data line by line while analyzing the data on the fly, so that the data need not be read to memory before being analyzed. The result is a shortened listing containing the relevant information.

The column headers shown in the tables are explained:

LINENO.	File Line Number in RTC data dump file
se	seconds
sa	seconds alarm
mn	minutes
ma	minutes alarm
hr	hours
ha	hours alarm
dw	DOW (Day of Week)
dy	day
mo	month
yr	year
st	Status Register "A" byte value
err	Error indicator, in this format: 0 = no errors, data valid, UIP bit set low (26h) 1 = UIP bit set high (a6h) 2 = error in data while UIP bit is low (26h) 3 = may result in possible TD anomaly

Table 4 is a sample listing from a sample RTC data dump file (20,000 lines) used to test the RTCSCAN program. The listing shows the author's early vision of how a TD anomaly may result from a boot up –

1. *if* the BIOS POST routine does not honor the UIP bit,
2. *if* the RTC is not buffered, and
3. *if* the BIOS POST routine manages to read past the 244-microsecond "grace period" window* (see Figure 2, Section 4.0 in this report)

* This turned out to be an erroneous statement on the part of the author, in retrospect. This author no longer believes this is possible, nor is it supported by the evidence. In order to read past the 244-microsecond window, the total read process time must be greater than 244 microseconds, because the read process must start before the 244-microsecond window begins, which is when the UIP bit goes high. Bad data begins after this window ends in a non-buffered RTC chip.

Before ensuing on the topic of RTC data at hand, it is first necessary to show how to interpret the RTC data tables (Table 1 through 9 listed after Section 5.3.3.4).

Some RTC data tables are generated with Echin's RTC.EXE (discontinued since late 1998) and companion program called RTCSCAN. Others are generated by the author's modified version of Flint's RTC data dump program. Both RTC data dump programs continually read RTC data to memory and then dumps it to the hard disk after continuous read is completed. This is 20,000 lines of data (RTC.EXE only). The companion RTCSCAN reads the resulting data file and selects all data with the UIP set high, plus any erroneous data found in between, and stores them in a linked list data structure and displays them. The RTC data tables are the output results from the RTCSCAN program.

Both programs read the RTC in the same direction indicated in the tables, i.e., from seconds to year in that order, plus the status byte. In other words, it reads the status byte *last*. In a normal computer boot, the status byte is read *first* to determine the state of the UIP bit, then a read sequence is commenced for either the time or date (*not both together*). For these data-dump utilities, both time and date plus the status byte is read. The read process during a normal boot begins with the lowest unit of measure: for the time: seconds, then minutes, then hours; for the date: the day, month, low-digit year, then the century byte.

This means the status byte in a line of RTC data in the tables apply to the *next* line of data, not on the same line where it appears. This has significance in that it impacts the theoretical analysis of the culprit provided elsewhere in this report. At the same time, however, it does not mean the RTC is the culprit for the TD anomaly. The next section, Section 5.3.3, will show why.

Now, we continue in the topic discourse starting with clarifications in Echlin's comments in his technical paper.

On Mr. Mike Echlin's web site, two snippets of RTC data are shown, one from a 486DX 33 MHz computer, and one from another (unnamed) computer.[51] Hereinafter the former shall be referred to as the first listing, and the latter the second listing, respectively, in this section.

Both listings of RTC data were obtained directly from the RTC (the BIOS was bypassed). The first listing has "FF" BCD numbers in the update period, while the second has numbers equal to the value of the seconds field. This means two different manufacturers made the respective RTC chip. Both are non-buffered chips.

The narrative subtly reveals the difference between buffered and non-buffered RTCs: Computers with buffered RTCs are free from the effects of TD. But it is not clear to the average reader. However, an excerpt from another source is clear on this point:

> *We know that a computer that has a 'buffered' RTC will not show TD. This has been shown in our software testing.*[52]

The author agrees with this assessment. There are 100% buffered RTC chips that do not have any bad data in the update period. A computer with this type of RTC chip will never exhibit a TD anomaly.

Echlin calls the invalid data in the update period "errors" in his technical paper on his web site. However, they are not errors per se – they are normal because the update period is set aside for the data to be changed. This is the reason for the UIP bit – so that any program accessing the RTC directly can tell if the RTC is presently undergoing an update.

The narrative comment that "the effect happens only on [computer] startup" [53] is accurate, but *neither data listing itself came from the RTC during POST or during startup*. It came from the RTC while the computer was running, *long after the computer has booted*.

Hence, Echlin's RTC data excerpts – including the author's data shown in Tables 1 through 8 – are *useless* as proof, or even evidence, of a possible TD anomaly.

The RTC data excerpts *do prove*, however, that the RTC is functioning properly and therefore is not the culprit for the TD anomaly. This is demonstrated in the next section.

5.3.3 RTC Data

This section covers the effect of RTC access time versus program reading speed and computer speed, the RTC access time itself, and nine RTC data tables (listed at the end of Section 5.3.3.4).

On Mike Echlin's web page is a technical paper called "The Crouch Echlin effect, detailed," from which small excerpts are cited in Sections 5.3.1 and 5.3.2 of this report. The technical paper includes two different RTC data dump excerpts from two different computers.

The first RTC data listing in Echlin's technical paper is from a 486DX computer with a non-buffered RTC. It shows FF numbers in the update period.[54]

The second RTC data listing that follows the first listing is also from an RTC with non-buffered registers.[55] Part of this second listing shows the number "39" being repeated over again in the update period. This repeated number is from the seconds field (see Table 2 after Section 5.3.3.4, "se" column, for an example).

Table 1 after this section shows an RTC data dump from an Intel 486 computer with a non-buffered RTC. In this table, the FF numbers can be seen in the update period, and this indicates a non-buffered RTC.

Table 2 shows an RTC data dump from another Intel computer, a 486DX with a non-buffered RTC. In this table, no FF numbers are seen in the update period; instead, it shows the number "2" being repeated for a short time, before changing into the number "3". This is the actual update of the seconds field. Then, the data continues as before with "good" numbers – though technically still invalid while in the update period.

What is the difference between the listings in Tables 1 and 2 in this report and the two listings in Echlin's technical paper? None except one: the Intel computers whose data are shown in Tables 1 and 2 do not exhibit and have never exhibited any TD anomaly. Echlin's listings come from two computers he claims exhibit the TD anomaly.

So why is there no difference if one RTC data listing comes from a computer exhibiting TD and another listing comes from another computer that does *not* exhibit it?

We shall see the clear answer in this section.†

Theoretically speaking, loosely paraphrasing Echlin's theory, the closer bad data begins at the point the 244-microsecond window expires, the better the chances of a TD anomaly resulting from a read into the update period. This is the ostensible position taken by Mike Echlin in his "Theory of How TD Occurs on Personal Computers." [56] *But* the precise time when the actual update occurs is fixed at 244 microseconds *after* the UIP bit goes high. If the BIOS code properly honors the UIP bit, then all this theory regarding non-buffered RTCs *must assume* that the RTC access time is *longer than 244 microseconds!* This is hard to swallow, considering the *sole* reason for the longer access time is merely because of the value of the century byte. We are touching again on logic paths, but this has been addressed earlier in this report.‡

Echlin theorizes that the "time and date calculations take longer than 244 microseconds" if the century byte returns the century-year value of 20.[57] But these calculations are made *after* the time/date information has been retrieved from the RTC, and not *during* read operations from the RTC. In fact, Intel confirms that these calculations are done at a higher level, after low-level RTC read operations are complete.[58]

If the RTC read access time is less than 244-microseconds, there is no way any such TD anomaly will ever occur – *if* the BIOS code honors the UIP bit. If the BIOS routine ignores the UIP bit, then the point about the 244-microsecond "grace period" window becomes moot.

In Tables 1 and 2, the update period is identified by the number "1" in the "err" column. Again, the number "1" is not an indicator of an error per se. It is a marker that identifies the update period based on the UIP status bit in Register "A", whose byte value is shown in the "st" column.

† The short answer is, the wrong dog was used to bark up the tree. The correct dog was the one barking up another tree next to it, and was brought over to replace the former dog.

‡ See Section 5.3.1.

NOTE

The rest of this section, including Sections 5.3.3.1 and 5.3.3.2, deals with examination of the RTC aggregate access time based on Tables 1 through 8. For the purpose of this analysis, the aggregate access time is defined as the time it takes to read all the time and date information, plus all the alarm data, plus the status byte. The validity of the access time definition is not important in this analysis (see Section 5.3.3.3).

But let us not forget that the RTC access time is not a primary issue to be considered as far as TD is concerned. Echlin's "Logic Path" Theory, of which the RTC access time is a primary factor, has already been disproved (see Section 5.3.1) and will be further disproved in this section.

This section also provides proof that the RTC is not the culprit for the TD anomaly.

Since we already know the length of the update period (1984 microseconds*), and we know the length of the grace period window (244 microseconds†), the RTC *aggregate access time*‡ is readily apparent. Simply divide the number of lines with the UIP bit set high into the known sum of the 244-microsecond window and the update period.

1. In Table 1, the aggregate access time is approximately 96.87 microseconds, which is less than the 244-microsecond window (year date 2000).

2. In Table 2, the aggregate access time is approximately 101.27 microseconds (year date 1998).

* Clearly stated in Motorola's Technical Data on RTC part number MC146818 and on Japanese version (Hitachi) part number HD146818

† Ibid.

‡ As opposed to access time of a single byte read from the RTC, *aggregate access time* combines reads of several bytes from the RTC. The intent is to show that the aggregate access time, though time, date, and alarm data are all included, still is less than 244 microseconds. Access Time is defined as the amount of time it takes to read a byte from the RTC and return. This definition is a little different from the term Access Time as applied to *hard disk access*, which is defined as the amount of time it takes to find and access a particular sector of the hard disk. The read time of this sector is not included. See Section 5.3.3.3 for more on RTC aggregate access time.

3. In the first RTC data excerpt listing in Echlin's technical paper, the aggregate access time is approximately 111.40 microseconds (year date 2000).

Echlin's second listing does not show the end of the update period, so the access time cannot be determined.

Why are these access times different? Because the RTC access time depends on the actual computer speed.§ Generally, the faster the computer speed, the faster the access time, and hence the less likely the resulting TD anomaly. Note that the RTC data listings in Tables 1 and 2, and Echlin's first listing excerpt, all come from 486 computers, all the same speed (33 MHz). So the computed access times show some consistency as well as correlation.

By contrast, using the same diagnostic utility (RTC.EXE) also confirms the speed of the computers tested.

Now that we know the approximate aggregate access time, we can then determine roughly how many lines of data covers the 244-microsecond "grace period" window. These lines start at the point the UIP bit is set active high. Once this is known, then we can correlate this information with the point the update period actually starts (see Figure 2 in Section 4.0 of this report). Simply divide the calculated RTC access time into the known "grace period" window.

1. For Table 1 listing, the number of lines representing the grace period since the start of UIP bit set active high is approximately 3 (2.52).

2. For Table 2, approximately 2 lines (2.41).

3. For Echlin's first listing, approximately 2 lines (2.19).

Comparisons of these results with the corresponding data listings show that the number of lines representing the 244-microsecond window closely correlates with the data just before the update period. It means the RTC is working properly, but that's all it means.

§ It also depends on RTC read speed of the diagnostic program, as will be explained in this section. All three listings heretofore mentioned come from the same diagnostic software – Echlin's RTC.EXE.

Bear in mind that these calculated results are only approximations. There is another, more accurate method of calculating RTC aggregate access time. In the RTC data tables, simply subtract the line number of the first occurrence of the A6h status value from the line number of next occurrence after the UIP (one second later in the data). Then find the reciprocal of that difference. The result is the aggregate RTC access time in seconds.

This is enough to show that the RTC is not the culprit for the TD anomaly. However, it proves nothing for the BIOS code. All these listings were generated while the computer is running, not during computer boot, of which the BIOS code has a large part. If the BIOS code does not honor the UIP bit, then the anomaly would occur only on computer boot-up.

There is more to be said from this demonstration. Echlin's theory states that if the century byte value is 20, then the RTC aggregate access time is *longer than 244 microseconds* due largely to increased processing time dealing with the century byte.[59] Echlin's first RTC data dump listing (not the second one) contains the year date of 2000, which means the century byte value is 20.[60] Echlin states that the computer whose RTC the listing came from is a 486DX 33 MHz computer, and that it exhibits the TD anomaly frequently.[61]

However, we have just demonstrated that the RTC aggregate access time is *only 111 microseconds, based solely on his own RTC data listing.* And this, from a computer that purportedly exhibits the TD anomaly – frequently. However, as already noted, this data listing did not come from a computer boot. Therefore Echlin's RTC data listing excerpt proves nothing; it does not support Echlin's theory because this data was not obtained during computer boot. This caveat is not explained in Echlin's paper.

Note that Echlin's listing comes from the RTC.EXE utility that did not disable interrupts.† It explains some strange small errors in listings the author has seen during his tenure in the investigation. However, it does not necessarily invalidate Echlin's listings because the listings prove the RTC is working properly.

Intel makes a strong, valid point in regard to 244-microsecond limit on faster computers. In their report, it states:

† As indicated by the source code. No I/O delay was used, either, to prevent RTC address line overruns. It also did not honor the UIP bit, but this was by design. This utility was discontinued in late 1998.

This theoretical "year 2000 code path," if applied to Pentium processor-based computers, would have to be 20 times longer than the theoretical "nineteen hundreds code path" in order to exceed the 244 µs limit due to the increase in processor speed between the 286 processor and the Pentium processor.[62]

This author agrees with this assessment. The 244-microsecond "grace period" window is independent of the computer speed, but the RTC access time is not (i.e., it is dependent on computer speed). The year-2000 logic path theory would not explain TD anomaly occurrence on faster computers such as a 486DX compared to a 286 machine.

Therefore, based on exhaustive analysis heretofore provided beginning in Section 5.3, Echlin's theory as outlined in "Mike Echlin's Theory of How TD Occurs on Personal Computers" web site, and the associated logic path theory in particular, is forthwith dismissed.

But the existence of TD on some computers is *not* dismissed. The author has a 286 computer that apparently exhibits the TD anomaly.*

5.3.3.1 RTC Access Time Versus Software Reading Speed

There is a point to be made regarding reading speed of diagnostic software versus RTC access time. Compare Tables 2 and 3. Both table data listings are from the same computer, except the year dates in the data between them are different.†

Table 2 data was derived from the 20,000-line RTC data generated from Echlin's RTC.EXE diagnostic utility. Table 3 data was derived from Flint's modified assembler software, dubbed by the author as FRTC3.EXE, with the "F" standing for Flint in honor to him for his efforts during the author's tenure in the investigation in 1997.

Both RTC.EXE and FRTC3.EXE utilities generate output files containing RTC data – with one difference: RTC.EXE generates space-delimited text files, and FRTC3.EXE generates binary files in record format instead of text. Both Tables 2 and 3 listings were generated using the author's RTCSCAN program, whose current version now supports both file formats.

---

* The date retrogresses, but does not advance, away from the true date. This anomaly is variously repeatable.

† If you don't believe Tables 2 and 3 are from the same computer, compare the alarm data in both tables. They are the same.

Actually, Table 2 data was generated using RTCSCAN version 0.60; Table 3 was generated using RTCSCAN version 0.30, an early version that supported Flint's output file format before it was removed in subsequent versions. Now RTCSCAN version 0.60 supports both file formats and includes a traversable directory list of files in a window so that files can be selected interactively for data processing while reading the data on the fly.

By examining Tables 2 and 3, you will see a large difference in the number of lines in the data where the UIP is active high (status byte indicating A6h value). Why are the number of lines different?

Reason: Echlin's RTC.EXE was compiled using C++ compiler. Flint's FRTC3.EXE (as modified by the author) was written and compiled in assembler. Assembler programs run faster than C++ programs. Hence, more data is read by the assembler program than the equivalent C++ program in the same time period.

Assembler programs are better than C++ programs because they provide better timing resolution, especially in this area of RTC data retrieval. The BIOS firmware is also written in assembler.

Because of this improved resolution, the RTC aggregate access time in Table 3 using FRTC3.EXE is only about 51.81 microseconds, compared with 101 microseconds using Echlin's RTC.EXE diagnostic utility. In Table 3, the calculated number of lines covering the 244-microsecond "grace period" window is approximately 5 lines, or 4.71. This agrees closely with the data shown in Table 3 – five lines with the UIP bit set high transpires just before the actual update occurs.

This shows that the measurements in RTC access time are relative to the reading speed of the running program that reads the RTC data.* It does not invalidate Echlin's RTC.EXE access time because its data retrieval speed is relative to the reading speed of the program – although computer speed is a larger factor.† It is the program code, not reading speed, that slows RTC.EXE execution and thus invalidates RTC.EXE output results – especially on slow computers such as a 286 12 MHz computer. An equivalent assembler program reads RTC data faster because less code is required to execute.

* Largely depends on the computer speed.

† See Section 5.3.3.2.

The FRTC3.EXE utility properly inserts I/O delay instructions per discrete I/O operation with the RTC. If the delays were to be reduced to one I/O delay instruction instead of two, it's likely that the resulting access time will be less than the calculated 52 microseconds.

Conclusion: Echlin's C++ runtime utilities run slower than BIOS routines, especially on slower computers, because the BIOS firmware is written in assembler. An analysis of the returned data from RTC.EXE will yield misleading results if the data comes from a slow computer, although the integrity of the data themselves is not compromised. It is the timing of the data that is compromised.

This discussion does not reconcile in any way the alleged increased RTC access time due to the (alleged) dual logic paths based on the century byte during computer boot. It is possible, however, that the use of such C++ runtime utilities contributed to the theory that the 244-microsecond window is being exceeded.

5.3.3.2 RTC Access Time Versus Computer Speed

Now here's the fun stuff – and the cumulative point of this entire Section 5.3.3. Tables 5 and 6 show RTC data dumps from the author's 286 computer with a non-buffered RTC. The author's RTCSCAN program reported a "possible" TD anomaly (indicated by a "3" in the "err" column).‡

According to the data in Tables 5 and 6, the number of lines in which the UIP is set high is 4 most of the time. That means the aggregate access time is roughly 557 microseconds – far greater than the 244-microsecond window. Does this mean we have a problem?

Answer: No, it does not. The RTC data dumps in Tables 5 and 6 came from Echlin's RTC.EXE program; it ran too slowly to gather all the readings from the RTC, due mainly to the speed of the 286 microprocessor.§ This is proven below.

Now see Tables 7 and 8. These listings come from the same 286 computer and come from the modified FRTC3.EXE program, which runs fast enough to gather more readings from the RTC than did Echlin's RTC.EXE in the same time period. The RTC aggregate access time is now 139.25 microseconds in both Tables 7 and 8. And the "grace period" window covers about 1.75 lines starting at the first line with the UIP bit set high. There are now no problems in these tables.

---

‡ This is the author's TD computer. Note that the data came from the RTC.EXE output results.

§ The program's reading speed and programming language are also factors. However, computer speed is the largest factor.

Again, we see how important the speeds of diagnostic tools are, especially in this application. Again, assembler wins.

But it is important to note a strong caveat here. *The actual RTC access time during computer boot is still not addressed.* However, the main point to be driven is that the actual RTC access time during computer boot *depends on how the BIOS POST code is written, as well as computer speed.* Further, every version of BIOS have different versions of the code written by different manufacturers.

There is yet *another* caveat when it comes to testing the RTC access time. It is the fact that drivers and TSRs loaded in memory during computer-boot processing of CONFIG.SYS or AUTOEXEC.BAT can and *will* affect the RTC access time while the machine is running. The more TSRs and drivers loaded, the more time it takes to retrieve the information from the RTC. This is why it is important to base all time-related measurements on a "vanilla" boot configuration, which is a configuration that has no CONFIG.SYS or AUTOEXEC.BAT script commands that load such software in memory.*

All tests using diagnostics programs were done while the computer is running, after the computer boot has been completed. Because the computer speed affects RTC access time, and because RTC access time during computer boot may be different from diagnostic program's *own* access time to the RTC, *such diagnostic programs prove nothing…*

…except one thing. Echlin's diagnostic program RTC.EXE returned erroneous timing information on a slow computer, thus leading to false conclusions.

One could write a diagnostic program such that the program's RTC access time is slow enough to flag a problem. Since the RTC access time is different across platforms according to computer speed, how could such a diagnostic program accurately report a problem if the actual RTC access time during computer boot is not known? Only by using the system bus speed as a reference can this question be answered.

* Which is why, by the way, the data results of the RTC.EXE utilty must be discounted. It is also the reason Echlin's newest utilities should be taken with salt over the shoulder, because the utility result is obtained while the machine is running. Who knows what other time-robbing utilities are loaded in memory? Speed rules!

The same question applies to a diagnostic program that runs too fast to catch *any* problem related to TD regardless of the computer platform. This may be a moot point if assembler is used, since the same programming language is used in the BIOS code. There is no language that can perform faster than assembler.

In conclusion, the diagnostic programs and all RTC data listings in this section prove nothing except that all RTC chips work as expected. They did not help to expose the cause of the TD anomaly, but they did rule out the RTC chip as the cause.

Furthermore, runtime diagnostic programs are useless to flag even a "possible" TD anomaly because the actual RTC aggregate access time can differ between boot-up period and after the boot-up process is completed.

5.3.3.3 RTC Aggregate Access Time

The author's definition of RTC "access time" is different from a hard disk access time. For a hard disk, the access time is defined as the amount of time it takes to search and find a sector on the hard disk. This access time does not include the time it takes to read that sector. RTC access time is the amount of time required to read or write one byte of information. This access time includes the time it takes to set the address for reading or writing to/from that address. I/O delay instructions in between setting an address and reading or writing to/from that address is also included.

The *aggregate* access time is defined as the amount of time it takes to read *n* bytes from the RTC, where *n* is the number of bytes greater than 1. For the time, the number of bytes read is three; for the date, four. For RTC.EXE, it is a total of 8 bytes because both the time and date, plus the status byte, are included on a line of data.

The demonstrations in Sections 5.3.3, 5.3.3.1, and 5.3.3.2 may *appear* to lend further credence to Intel's claim in their report that the BIOS in Echlin's 286 computer "spends only 35 microseconds while reading or writing RTC registers." [63] Based on the context of Intel's report, the date information is alluded to when the access time is mentioned, not the time information.

The truth is that the calculated access times shown in Sections 5.3.3, 5.3.3.1, and 5.3.3.2 cannot be compared to Intel's rendered date access time, because the method used in Echlin's RTC.EXE utility to retrieve the RTC information did not include using BIOS interrupt 1Ah to obtain the date. Instead, direct RTC access is used to obtain the date. In a computer boot, this BIOS routine is used to obtain the date, which increases the access time to some extent because of interrupt vectoring. Another reason is that both date and time data, plus the status byte, are part of the aggregate access time calculation in these sections, whereas Intel's statement refers only to the date.

Intel's report does not clearly indicate to the reader whether this access time is an *aggregate* access time or whether this access time is merely of *one RTC read of a single byte*. The author believes, however, that the latter is referred to, since 35 microseconds is much too short for an aggregate access time on a slow 286 12 MHz computer, but not too short for a read of a single byte.

In fact, date retrieval tests on the author's 286 computer show an aggregate access time of 176 microseconds, which, divided by 4 bytes of the date information, yields an access time of 44 microseconds for each byte. This is close enough to support the single-byte access time in Intel's statement.

Therefore, by calculation the aggregate access time for the date on Echlin's 286 computer is 140 microseconds, which is a far cry from the 244-microsecond window. *Where is the problem?*[*]

Examination of published BIOS code does indeed confirm Intel's statement that the date conversion code is carried out "at a higher level, after low-level RTC access is completed." [64] This higher level is the DOS kernel. It also confirms that the time and date are read separately, not together, and at different times during computer boot.

Therefore, when considering aggregate access times, the date and time aggregate access times should be mutually exclusive, not combined.

[*] Evidently, Echlin considers the *combined* aggregate access time of *both* date and time retrieval to be a problem. As this report proves, there is no problem at all because the date and time are *separately* processed in a computer boot. The truth is that there is *no proven case* in which the 244-microsecond window is exceeded. Therefore, the RTC access time is not a factor to be considered as far as TD anomalies are concerned. See Section 5.3.1.

Regarding combined date and time (and status byte) aggregate access calculations performed in Sections 5.3.3, 5.3.3.1 and 5.3.3.2, it is important to understand that for the purpose of the demonstrations in those sections, the definition of the access time is not important. The intent of the demonstration is two-fold: to prove the RTC is not the culprit for the TD anomaly, and to prove that Echlin's RTC.EXE utility can give misleading results, especially on slow computers.

This is another reason access times in those sections should not be compared to Intel's statement regarding date retrieval access time.

5.3.3.4 TD Test Lite Utility (TDTLELM.EXE) and TD Test Utility (TDTL.EXE)

Echlin's new diagnostic TDTLELM.EXE and TDTL.EXE utilities are free programs available on Mr. Echlin's web site. They each perform essentially the same function as the old RTC.EXE diagnostic utility, with important major differences.

The author notes that the image code in this program is encrypted. This is a smart move, considering Intel apparently reversed-engineered Echlin's TDFIX.EXE and TDTEST.EXE utilities. But examination of this code reveals that these utilities are also compiled in C++ language.

Here is an output excerpt from the TDTLELM.EXE diagnostic utility (Revision 1.2) from a computer that has a non-buffered RTC (my dual-processor Pentium Pro 400 MHz computer, with a PCI bus architecture, purchased in 1999!):

```
Base time from RTC.
20:15:41 12,09,99

Test time from RTC.
41:41:41 41,41,41; a6
^^<<<<<<<<<
This is the value that Failed your PC!

Your RTC has failed this test.

This shows that you have a non-buffered RTC and this is
the common thread among machines that suffer
from the Year 2000 effect known as "TD"
(or "the Crouch, Echlin effect").
This does not prove your machine has TD, but shows that you
have the hardware that is common to machines that do.
```

The statement that the "RTC has failed this test" is misleading, since the RTC – non-buffered – is not the culprit for any TD anomaly, as clearly demonstrated in this report. As the excerpt above says, it indeed does not prove the computer will ever exhibit such a TD anomaly, but it falsely attributes such RTC chips to TD anomaly susceptibility. We have already proven that the RTC chip itself is not the culprit for the TD anomaly.

When the free diagnostic utility is executed, a screenful of explanatory text is displayed, then a 3-second delay is imposed before the user is prompted with the statement "Press any key to continue." After this, a series of dots (the period character) is displayed across the screen, filling the screen line by line. A shorter series of commas are also displayed in between the longer series of dots.

Then the "Base time from RTC" data is displayed. This is the current time/date information from the RTC. Then there is a very short pause in display before the screen is cleared and the "Test time from RTC" data is displayed, and another screenful of explanatory text is displayed below it, with another 3-second delay before another "Press any key to continue" prompt.

After pressing a key again, another screenful of explanatory text is displayed with the same routine before letting you exit to DOS.

The impressive-looking dots/commas display gives the impression that the RTC is being accessed for data, looking for erroneous date/time data. But this is not true. (Actually, it *should* not be true. These displays will interfere with the timing of RTC read process.) The user can prove this to himself or herself on any computer.

A simple test was performed on the author's 386 25 MHz computer that has a non-buffered RTC.† Just after the "Base time" data is displayed, the Pause key was pressed. This stops the CPU cold, thus stopping current program execution in its tracks. The author waited several minutes before resuming program execution by pressing any key except the Pause key. Here is the result:

```
Base time from RTC.
07:04:01 12,14,00

Test time from RTC.
07:12:01 12,14,00; 26
```

† This computer does not exhibit the TD anomaly.

The utility pointed to the minutes portion of the time information as bad data. The elapsed time during which the computer was in paused state is reflected in the difference in displayed time information. This shows that the "Test time" data did not come from the period of time in which series of dots/commas are displayed. It shows this data is from sometime after the "Base time" data is displayed, not before.

What does this mean? It probably means the impressive-looking series of dots/commas display is a cosmetic display only. It make sense to program this way, since it is undesireable to allow video writes to adversely affect the analysis because the analysis is time-sensitive.

This display is not without some significance. The slower the computer is, shorter the series of commas. This is consistent with longer access time on slower computers. The total lengths of the dots/commas display varies from execution to execution, however, but it also varies from computer to computer.

From the author's experience in looking at RTC data returns, the longer series of dots represent segment of time in which RTC data is valid, and the commas represent the segment of time in which the UIP is active. In fact, the author has written a program that similarly displays such indicators – in assembler code.

Note the status byte value (26). This indicates this "erroneous" data was retrieved while *outside* the UIP. This is *good* data, not bad. The utility caught the error and displayed it because the minutes portions in both time information are different. Of course, it is because program execution was halted by the Pause key. But it shows evidence that the utility simply looks for *any* errors in the RTC data.

If the utility encounters a computer containing an RTC chip that has no bad data in the UIP, such as in Table 9, the utility will return a message saying the RTC passed the test. This indicates that the utility simply looks for *any* errors in the RTC as a go or no-go test. *By extension*, If any errors are found, the utility simply calls the RTC a non-buffered RTC.

Table 9 shows an example data output of a buffered RTC chip. Any computer that has this type of RTC with this kind of data in the UIP will pass the test. No bad data exists in the UIP period in this table at all.

The diagnostic TDTL.EXE utility (Rev. 1.3) performs the same function as the TDTLELM.EXE, but adds a function to check the RTC access time. No definition of this access time is provided. But since the time and date information is returned by the utility, it is presumed that the access time includes both the time and date retrieval, perhaps including the status byte. If this is the case, then the returned access time value should be divided in half to obtain the *true* (approximate) aggregate access time. This is because the time and date are *separately* processed in a computer boot.

This is further indicated by comparing Echlin's RTC access time with two different utilities. One utility, provided by Mr. Michael Kennedy, measures the access time of the date retrieval (and not the time) which is far less than that returned by Echlin's TDTL.EXE utility. RTC access time returned by the author's own utility also is far less than Echlin's results.

For example, on the author's 386 25 MHz computer (non-buffered RTC), Kennedy's utility returns an averaged 102 μsecs, compared to 300 μsecs returned by Echlin's TDTL.EXE. Dividing Echlin's result in half yields 150 μsecs. The author's utility returned only 88 μsecs.

On the author's 486 33 MHz computer,‡ Echlin's utility returned 305 microseconds, while Kennedy's returned an averaged 181 microseconds. The author's utility returned 158 μsecs. Although the rated 486 speed is faster than the 386 computer, the 486 RTC access time is still greater than that of the 386. The reason? The wait states and selection of bus clock speeds (selectable in CMOS setup on some computers) affects the speed, hence affecting the access time. Performance-wise, the 486 machine is actually slower than the 386.

Another factor affecting bus speed is TSR's and drivers loaded in memory. Booting a machine in a "vanilla" configuration* actually improves speed and shortens RTC access times slightly, but this improvement is more marked in Echlin's results than the results of the other two utilities: The delta of access times between "vanilla" and normal configurations is greater than the other two utilities by an approximate factor of 2. The delta of the other two utilities are nearly zero (the results are nearly the same). If this does not smell fishy to you, it should. Here is the table of access time results:

‡ This computer does not exhibit the TD anomaly.

* A vanilla boot configuration is a computer boot without processing the AUTOEXEC.BAT and CONFIG.SYS files.

CPU Type	Test Utility Used	Vanilla Boot Configuration (µs)	Normal Boot Configuration (µs)	Delta Time (µs)
386	Author's	85	126	**41**
	Kennedy's	102	145	**43**
	Echlin's	195	300	**105**
486	Author's	141	158	**17**
	Kennedy's	164	181	**17**
	Echlin's	275	305	**30**

This is evidence that Echlin's utility combines both time and date access times together. As already proven elsewhere in this report, the time and date information is *separately* processed in a computer boot.

If Echlin's utility does not combine both time and date access times together, then the utility is inflating the actual access time.

If the access time includes both the time and date, 7 bytes are retrieved together (8 bytes if the status byte is included). This is the reason for the division in half in Echlin's utility results.

Since Echlin's utility is written in C++ language, there is no guarantee as to the accuracy of the access time, especially on slow 286 computers, as proven in this report. Both the author's and Kennedy's RTC date access time utilities are written in assembler, and both measures access time for date retrieval only.

In fact, on the author's 286 machine with a non-buffered RTC, Echlin's utility returned an access time of 520 to 525 µsecs, but Kennedy's utility returned an averaged 207 µsecs. Dividing 525 µsecs in half does not meet the 244-µsec window limitation. Why?

Easy. Echlin's utility runs too slow on slow machines (286 computers) to catch all RTC data without missing a beat. The author's own utility, written in assembler, returned 176 µsecs. It is of worthy note that, historically, the 286 computer was released to the public *before anyone ever heard of the C++ language*. This is a serious mistake for Echlin; all of his programs/utilities are written in C++ language.

One further discrepancy should be noted for the TDTL.EXE utility. If the utility returns a message result stating the RTC is not buffered and a message stating the RTC access time is "OK," the utility still recommends considering the computer as "suspect." The truth is that if the returned access time is far less than the 244-μsec window, then there is no reason for concern even if the RTC is in fact not buffered. Unfortunately, however, if the BIOS code is not RTC compliant, a TD anomaly may still pop up. Nevertheless, there appears to be a very small percentage of 286 machines showing the symptoms of a TD anomaly.

None of these utilities have the capability to repeat the action of the actual BIOS code, especially the time retrieval code, because the BIOS code is different among different BIOS versions from different manufacturers.

Frankly, based on this evaluation, this author does not count this free utility worth the time downloading it. It appears written to score "hits" on any computer to incite (scare up) sales. This is not consistent with the oft-repeated statement that the TD anomaly is "rare." [65]

The assembler source code for the author's date access time measurement and RTC access time measurement of a single byte are provided in Section 10.0.

```
Generated by RTCSCAN vs 0.60
RTCSCAN: RTC Data File Scanner
Copyright 1998 Michael D. O'Connor

            RTC Data Scan Results
-------------------------------------------
LineNo.  se sa mn ma hr ha dw dy mo yr st  err
      1  59 FF 13 FF 14 FF  0 31  1  0 26  0
      2  59 FF 13 FF 14 FF  0 31  1  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
    404  59 FF 13 FF 14 FF  0 31  1  0 26  0
    405  59 FF 13 FF 14 FF  0 31  1  0 A6  1
    406  59 FF 13 FF 14 FF  0 31  1  0 A6  1
    407  59 FF 13 FF 14 FF  0 31  1  0 A6  1
    408  59 FF FF FF FF FF FF FF FF FF A6  1
    409  FF FF FF FF FF FF FF FF FF FF A6  1
    410  FF FF FF FF FF FF FF FF FF FF A6  1
    411  FF FF FF FF FF FF FF FF FF FF A6  1
    412  FF FF FF FF FF FF FF FF FF FF A6  1
    413  FF FF FF FF FF FF FF FF FF FF A6  1
    414  FF FF FF FF FF FF FF FF FF FF A6  1
    415  FF FF FF FF FF FF FF FF FF FF A6  1
    416  FF FF FF FF FF FF FF FF FF FF A6  1
    417  FF FF FF FF FF FF FF FF FF FF A6  1
    418  FF FF FF FF FF FF FF FF FF FF A6  1
    419  FF FF FF FF FF FF FF FF FF FF A6  1
    420  FF FF FF FF FF FF FF FF FF FF A6  1
    421  FF FF FF FF FF FF FF FF FF FF A6  1
    422  FF FF FF FF FF FF FF FF FF FF A6  1
    423  FF FF FF FF FF FF FF FF FF FF A6  1
    424  FF FF FF FF FF FF FF FF FF FF A6  1
    425  FF FF FF FF FF FF FF FF FF FF A6  1
    426  FF FF FF FF FF FF FF FF FF FF A6  1
    427  FF FF FF FF FF FF FF FF FF FF A6  1
    428   0 FF 14 FF 14 FF  0 31  1  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  10826   0 FF 14 FF 14 FF  0 31  1  0 26  0
  10827   0 FF 14 FF 14 FF  0 31  1  0 A6  1
  10828   0 FF 14 FF 14 FF  0 31  1  0 A6  1
  10829   0 FF 14 FF 14 FF  0 31  1  0 A6  1
  10830   0 FF 14 FF FF FF FF FF FF FF A6  1
  10831  FF FF FF FF FF FF FF FF FF FF A6  1
  10832  FF FF FF FF FF FF FF FF FF FF A6  1
  10833  FF FF FF FF FF FF FF FF FF FF A6  1
  10834  FF FF FF FF FF FF FF FF FF FF A6  1
  10835  FF FF FF FF FF FF FF FF FF FF A6  1
  10836  FF FF FF FF FF FF FF FF FF FF A6  1
  10837  FF FF FF FF FF FF FF FF FF FF A6  1
  10838  FF FF FF FF FF FF FF FF FF FF A6  1
  10839  FF FF FF FF FF FF FF FF FF FF A6  1
  10840  FF FF FF FF FF FF FF FF FF FF A6  1
  10841  FF FF FF FF FF FF FF FF FF FF A6  1
  10842  FF FF FF FF FF FF FF FF FF FF A6  1
  10843  FF FF FF FF FF FF FF FF FF FF A6  1
  10844  FF FF FF FF FF FF FF FF FF FF A6  1
  10845  FF FF FF FF FF FF FF FF FF FF A6  1
  10846  FF FF FF FF FF FF FF FF FF FF A6  1
  10847  FF FF FF FF FF FF FF FF FF FF A6  1
  10848  FF FF FF FF FF FF FF FF FF FF A6  1
  10849  FF FF FF FF FF FF FF FF FF FF A6  1
  10850   1 FF 14 FF 14 FF  0 31  1  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  20000   1 FF 14 FF 14 FF  0 31  1  0 26  0
```

TABLE 1: Data dump from Intel 486 computer with non-buffered RTC

```
Generated by RTCSCAN vs 0.60
RTCSCAN: RTC Data File Scanner
Copyright 1998 Michael D. O'Connor

            RTC Data Scan Results
-----------------------------------------------
LineNo.  se sa mn ma hr ha dw dy mo yr st  err
      1   2  0 50 42  1  6  7  6  2 98 26  0
      2   2  0 50 42  1  6  7  6  2 98 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
   3798   2  0 50 42  1  6  7  6  2 98 26  0
   3799   2  0 50 42  1  6  7  6  2 98 A6  1
   3800   2  0 50 42  1  6  7  6  2 98 A6  1
   3801   2  0 50 42  1  6  7  6  2  2 A6  1
   3802   2  2  2  2  2  2  2  2  2  2 A6  1
   3803   2  2  2  2  2  2  2  2  2  2 A6  1
   3804   2  2  2  2  2  2  2  2  2  2 A6  1
   3805   3  3  3  3  3  3  7  6  2 98 A6  1
   3806   3  0 50 42  1  6  7  6  2 98 A6  1
   3807   3  0 50 42  1  6  7  6  2 98 A6  1
   3808   3  0 50 42  1  6  7  6  2 98 A6  1
   3809   3  0 50 42  1  6  7  6  2 98 A6  1
   3810   3  0 50 42  1  6  7  6  2 98 A6  1
   3811   3  0 50 42  1  6  7  6  2 98 A6  1
   3812   3  0 50 42  1  6  7  6  2 98 A6  1
   3813   3  0 50 42  1  6  7  6  2 98 A6  1
   3814   3  0 50 42  1  6  7  6  2 98 A6  1
   3815   3  0 50 42  1  6  7  6  2 98 A6  1
   3816   3  0 50 42  1  6  7  6  2 98 A6  1
   3817   3  0 50 42  1  6  7  6  2 98 A6  1
   3818   3  0 50 42  1  6  7  6  2 98 A6  1
   3819   3  0 50 42  1  6  7  6  2 98 A6  1
   3820   3  0 50 42  1  6  7  6  2 98 A6  1
   3821   3  0 50 42  1  6  7  6  2 98 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  13920   3  0 50 42  1  6  7  6  2 98 26  0
  13921   3  0 50 42  1  6  7  6  2 98 A6  1
  13922   3  0 50 42  1  6  7  6  2 98 A6  1
  13923   3  0 50 42  1  6  7  6  2 98 A6  1
  13924   3  3  3  3  3  3  3  3  3  3 A6  1
  13925   3  3  3  3  3  3  3  3  3  3 A6  1
  13926   3  3  3  3  3  3  3  3  3  3 A6  1
  13927   4  4  4  4  4  4  4  6  2 98 A6  1
  13928   4  0 50 42  1  6  7  6  2 98 A6  1
  13929   4  0 50 42  1  6  7  6  2 98 A6  1
  13930   4  0 50 42  1  6  7  6  2 98 A6  1
  13931   4  0 50 42  1  6  7  6  2 98 A6  1
  13932   4  0 50 42  1  6  7  6  2 98 A6  1
  13933   4  0 50 42  1  6  7  6  2 98 A6  1
  13934   4  0 50 42  1  6  7  6  2 98 A6  1
  13935   4  0 50 42  1  6  7  6  2 98 A6  1
  13936   4  0 50 42  1  6  7  6  2 98 A6  1
  13937   4  0 50 42  1  6  7  6  2 98 A6  1
  13938   4  0 50 42  1  6  7  6  2 98 A6  1
  13939   4  0 50 42  1  6  7  6  2 98 A6  1
  13940   4  0 50 42  1  6  7  6  2 98 A6  1
  13941   4  0 50 42  1  6  7  6  2 98 A6  1
  13942   4  0 50 42  1  6  7  6  2 98 A6  1
  13943   4  0 50 42  1  6  7  6  2 98 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  20000   4  0 50 42  1  6  7  6  2 98 26  0
```

TABLE 2: RTC data dump from Intel 486DX computer with non-buffered RTC

Record	se	sa	mn	ma	hr	ha	dw	dy	mo	yr	st	err
1	12	00	12	42	01	06	06	19	01	00	26	0
2	12	00	12	42	01	06	06	19	01	00	26	0
0	--	--	--	--	--	--	--	--	--	--	--	0
1907	12	00	12	42	01	06	06	19	01	00	26	0
1908	12	00	12	42	01	06	06	19	01	00	A6	1
1909	12	00	12	42	01	06	06	19	01	00	A6	1
1910	12	00	12	42	01	06	06	19	01	00	A6	1
1911	12	00	12	42	01	06	06	19	01	00	A6	1
1912	12	00	12	42	01	06	06	19	01	00	A6	1
1913	12	00	12	42	12	12	12	12	12	12	A6	1
1914	12	12	12	12	12	12	12	12	12	12	A6	1
1915	12	12	12	12	12	12	12	12	12	12	A6	1
1916	12	12	12	12	12	12	12	12	12	12	A6	1
1917	12	12	12	12	12	12	12	12	12	12	A6	1
1918	12	12	12	12	12	12	12	12	12	12	A6	1
1919	12	12	12	12	13	13	13	13	13	13	A6	1
1920	13	13	13	13	13	13	06	19	01	00	A6	1
1921	13	00	12	42	01	06	06	19	01	00	A6	1
1922	13	00	12	42	01	06	06	19	01	00	A6	1
1923	13	00	12	42	01	06	06	19	01	00	A6	1
1924	13	00	12	42	01	06	06	19	01	00	A6	1
1925	13	00	12	42	01	06	06	19	01	00	A6	1
1926	13	00	12	42	01	06	06	19	01	00	A6	1
1927	13	00	12	42	01	06	06	19	01	00	A6	1
1928	13	00	12	42	01	06	06	19	01	00	A6	1
1929	13	00	12	42	01	06	06	19	01	00	A6	1
1930	13	00	12	42	01	06	06	19	01	00	A6	1
1931	13	00	12	42	01	06	06	19	01	00	A6	1
1932	13	00	12	42	01	06	06	19	01	00	A6	1
1933	13	00	12	42	01	06	06	19	01	00	A6	1
1934	13	00	12	42	01	06	06	19	01	00	A6	1
1935	13	00	12	42	01	06	06	19	01	00	A6	1
1936	13	00	12	42	01	06	06	19	01	00	A6	1
1937	13	00	12	42	01	06	06	19	01	00	A6	1
1938	13	00	12	42	01	06	06	19	01	00	A6	1
1939	13	00	12	42	01	06	06	19	01	00	A6	1
1940	13	00	12	42	01	06	06	19	01	00	A6	1
1941	13	00	12	42	01	06	06	19	01	00	A6	1
1942	13	00	12	42	01	06	06	19	01	00	A6	1
1943	13	00	12	42	01	06	06	19	01	00	A6	1
1944	13	00	12	42	01	06	06	19	01	00	A6	1
1945	13	00	12	42	01	06	06	19	01	00	A6	1
1946	13	00	12	42	01	06	06	19	01	00	A6	1
1947	13	00	12	42	01	06	06	19	01	00	A6	1
1948	13	00	12	42	01	06	06	19	01	00	A6	1
1949	13	00	12	42	01	06	06	19	01	00	A6	1
1950	13	00	12	42	01	06	06	19	01	00	A6	1
1951	13	00	12	42	01	06	06	19	01	00	26	0
0	--	--	--	--	--	--	--	--	--	--	--	0

TABLE 3: RTC data dump from Intel 486DX computer with non-buffered RTC Using Flint's Modified ASM software

LineNo.	se	sa	mn	ma	hr	ha	dw	dy	mo	yr	st	err	
1	43	d	31	d	20	d	1	5	2	0	26	0	
2	43	d	31	d	20	d	1	5	2	0	26	0	
0	--	--	--	--	--	--	--	--	--	--	--	0	
847	43	d	31	d	20	d	1	5	2	0	26	0	
848	43	d	31	d	20	d	1	5	ff	ff	a6	3	⇐ Note 1
849	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
850	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
851	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
852	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
853	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
854	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
855	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
856	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
857	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
858	ff	ff	ff	ff	ff	ff	ff	ff	2	0	26	0	
859	44	d	31	d	20	d	1	5	2	0	26	0	
0	--	--	--	--	--	--	--	--	--	--	--	0	
5811	44	d	31	d	20	d	1	5	2	0	26	0	
5812	44	d	31	d	20	d	1	5	26	0	a6	3	⇐ Note 2
5813	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
5814	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
5815	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
5816	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
5817	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
5818	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
5819	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
5820	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
5821	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
5822	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
5823	ff	d	31	d	20	d	1	5	2	0	26	0	
5824	45	d	31	d	20	d	1	5	2	0	26	0	
0	--	--	--	--	--	--	--	--	--	--	--	0	
10774	45	d	31	d	20	d	1	5	2	0	26	0	
10775	45	d	31	d	20	d	1	5	26	0	26	2	⇐ Note 3
10776	ff	d	ff	d	ff	d	ff	ff	ff	ff	a6	3	⇐ Note 4
10777	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
10778	45	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
10779	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
10780	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
10781	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
10782	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
10783	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
10784	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
10785	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
10786	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
10787	ff	ff	ff	ff	ff	ff	1	5	2	0	26	0	
10788	46	d	31	d	20	d	1	5	2	0	26	0	
0	--	--	--	--	--	--	--	--	--	--	--	0	
15740	46	d	31	d	20	d	1	5	2	0	26	0	
15741	46	ff	31	ff	20	ff	1	5	2	0	a6	1	⇐ Note 5
15742	46	d	31	d	ff	ff	ff	ff	ff	ff	a6	1	
15743	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
15744	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
15745	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
15746	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
15747	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
15748	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
15749	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
15750	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	a6	1	
15751	ff	ff	ff	ff	ff	ff	ff	ff	2	0	26	0	
15752	47	d	31	d	20	d	1	5	2	0	26	0	
0	--	--	--	--	--	--	--	--	--	--	--	0	
20000	47	d	31	d	20	d	1	5	2	0	26	0	

Notes:

1. FFs occurring before the first status a6h causes error level 3.

2. Spurious data (26 in Month column) before the first status a6h causes error level 3.

3. Same spurious data (26 in Month column), but occurs before status 26h, causes error level 2, not 3, because this doesn't occur with the status byte a6h. TD is believed to occur when bad data occurs on the first occurrence of status a6h.

4. FFs in any of the columns shown will cause an error level of 3. Any other location (the alarm registers) will not cause an error level of 3, because the time/date information is not affected by these registers during bootup. However, it does mean something is wrong.

5. Same as Note 4. FFs or any spurious data in the indicated alarm registers will not change the current error level of 1.

This data dump from RTCSCAN version 0.51.

TABLE 4: Sample RTC data dump

```
Generated by RTCSCAN vs 0.60
RTCSCAN: RTC Data File Scanner
Copyright 1998 Michael D. O'Connor

            RTC Data Scan Results
-------------------------------------------
LineNo.  se sa mn ma hr ha dw dy mo yr st  err
      1   8 1f 11 58 19 1a  2  8  7 98 26  0
      2   8 1f 11 58 19 1a  2  8  7 98 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
   1940   8 1f 11 58 19 1a  2  8  7 98 26  0
   1941   8 1f 11 58 19 1a  2  8  7 98 a6  1
   1942   8 1f ff ff ff ff ff ff ff ff a6  1
   1943  ff ff ff ff ff ff ff ff ff ff a6  1
   1944  ff ff ff ff ff ff ff ff ff ff a6  1
   1945  ff ff ff ff ff ff ff ff ff ff 26  0
   1946   9 1f 11 58 19 1a  2  8  7 98 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
   3888   9 1f 11 58 19 1a  2  8  7 98 26  0
   3889   9 1f 11 58 19 1a  2  8  7 98 a6  1
   3890  ff ff ff ff ff ff ff ff ff ff a6  1
   3891  ff ff ff ff ff ff ff ff ff ff a6  1
   3892  ff ff ff ff ff ff ff ff ff ff a6  1
   3893  ff ff ff ff ff ff ff  8  7 98 26  0
   3894  10 1f 11 58 19 1a  2  8  7 98 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
   5836  10 1f 11 58 19 1a  2  8  7 98 26  0
   5837  10 1f 11 58 19 1a  2 ff ff ff a6  3
   5838  ff ff ff ff ff ff ff ff ff ff a6  1
   5839  ff ff ff ff ff ff ff ff ff ff a6  1
   5840  ff ff ff ff ff ff ff ff ff ff a6  1
   5841  ff ff ff ff  0 1a  2  8  7 98 26  0
   5842  11 1f 11 58 19 1a  2  8  7 98 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
   7783  11 1f 11 58 19 1a  2  8  7 98 26  0
   7784  11 1f 11 58 19 1a  2  8  7 98 a6  1
   7785  11 1f 11 58 ff ff ff ff ff ff a6  1
   7786  ff ff ff ff ff ff ff ff ff ff a6  1
   7787  ff ff ff ff ff ff ff ff ff ff a6  1
   7788  ff ff ff ff ff ff ff ff ff ff a6  1
   7789  12 1f 11 58 19 1a  2  8  7 98 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
   9731  12 1f 11 58 19 1a  2  8  7 98 26  0
   9732  12 1f 11 58 19 1a  2  8  7 98 a6  1
   9733  ff ff ff ff ff ff ff ff ff ff a6  1
   9734  ff ff ff ff ff ff ff ff ff ff a6  1
   9735  ff ff ff ff ff ff ff ff ff ff a6  1
   9736  ff ff ff ff ff ff ff ff  7 98 26  0
   9737  13 1f 11 58 19 1a  2  8  7 98 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  11679  13 1f 11 58 19 1a  2  8  7 98 26  0
  11680  13 1f 11 58 19 1a  2  8 ff ff a6  3
  11681  ff ff ff ff ff ff ff ff ff ff a6  1
  11682  ff ff ff ff ff ff ff ff ff ff a6  1
  11683  ff ff ff ff ff ff ff ff ff ff a6  1
  11684  ff ff ff ff ff  0  2  8  7 98 26  0
  11685  14 1f 11 58 19 1a  2  8  7 98 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  13627  14 1f 11 58 19 1a  2  8  7 98 26  0
  13628  14 1f 11 58 19 ff ff ff ff ff a6  3
  13629  ff ff ff ff ff ff ff ff ff ff a6  1
  13630  ff ff ff ff ff ff ff ff ff ff a6  1
  13631  ff ff ff ff ff ff ff ff ff ff a6  1
  13632  ff ff  0 58 19 1a  2  8  7 98 26  0
  13633  15 1f 11 58 19 1a  2  8  7 98 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  15574  15 1f 11 58 19 1a  2  8  7 98 26  0
  15575  15 1f 11 58 19 1a  2  8  7 98 a6  1
  15576  15 1f ff ff ff ff ff ff ff ff a6  1
  15577  ff ff ff ff ff ff ff ff ff ff a6  1
  15578  ff ff ff ff ff ff ff ff ff ff a6  1
  15579  ff ff ff ff ff ff ff ff ff ff a6  1
  15580  16 1f 11 58 19 1a  2  8  7 98 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  17522  16 1f 11 58 19 1a  2  8  7 98 26  0
  17523  16 1f 11 58 19 1a  2  8  7 ff a6  3
  17524  ff ff ff ff ff ff ff ff ff ff a6  1
  17525  ff ff ff ff ff ff ff ff ff ff a6  1
  17526  ff ff ff ff ff ff ff ff ff ff a6  1
  17527  ff ff ff ff ff ff  2  8  7 98 26  0
  17528  17 1f 11 58 19 1a  2  8  7 98 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  19470  17 1f 11 58 19 1a  2  8  7 98 26  0
  19471  17 1f 11 58 19 1a ff ff ff ff a6  3
  19472  ff ff ff ff ff ff ff ff ff ff a6  1
  19473  ff ff ff ff ff ff ff ff ff ff a6  1
  19474  ff ff ff ff ff ff ff ff ff ff a6  1
  19475  ff ff ff  0 19 1a  2  8  7 98 26  0
  19476  18 1f 11 58 19 1a  2  8  7 98 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  20000  18 1f 11 58 19 1a  2  8  7 98 26  0
```

TABLE 5: RTC data dump from 286 computer, year date 1998

```
Generated by RTCSCAN vs 0.60
RTCSCAN: RTC Data File Scanner
Copyright 1998 Michael D. O'Connor

            RTC Data Scan Results
-------------------------------------------
LineNo.  se sa mn ma hr ha dw dy mo yr st  err
      1  31 1f 35 58 19 1a  5  3  8  0 26  0
      2  31 1f 35 58 19 1a  5  3  8  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
    613  31 1f 35 58 19 1a  5  3  8  0 26  0
    614  31 1f 35 58 19 1a  5  3  8  0 a6  1
    615  ff ff ff ff ff ff ff ff ff ff a6  1
    616  ff ff ff ff ff ff ff ff ff ff a6  1
    617  ff ff ff ff ff ff ff ff ff ff a6  1
    618  ff ff ff ff ff ff ff  3  8  0 26  0
    619  32 1f 35 58 19 1a  5  3  8  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
   2561  32 1f 35 58 19 1a  5  3  8  0 26  0
   2562  32 1f 35 58 19 1a  5 ff ff ff a6  3
   2563  ff ff ff ff ff ff ff ff ff ff a6  1
   2564  ff ff ff ff ff ff ff ff ff ff a6  1
   2565  ff ff ff ff ff ff ff ff ff ff a6  1
   2566  ff ff ff ff 19 1a  5  3  8  0 26  0
   2567  33 1f 35 58 19 1a  5  3  8  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
   4508  33 1f 35 58 19 1a  5  3  8  0 26  0
   4509  33 1f 35 58 19 1a  5  3  8  0 a6  1
   4510  33 1f 35 ff ff ff ff ff ff ff a6  1
   4511  ff ff ff ff ff ff ff ff ff ff a6  1
   4512  ff ff ff ff ff ff ff ff ff ff a6  1
   4513  ff ff ff ff ff ff ff ff ff ff a6  1
   4514  34 1f 35 58 19 1a  5  3  8  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
   6456  34 1f 35 58 19 1a  5  3  8  0 26  0
   6457  34 1f 35 58 19 1a  5  3  8  0 a6  1
   6458  ff ff ff ff ff ff ff ff ff ff a6  1
   6459  ff ff ff ff ff ff ff ff ff ff a6  1
   6460  ff ff ff ff ff ff ff ff ff ff a6  1
   6461  ff ff ff ff ff ff ff ff  0  0 26  0
   6462  35 1f 35 58 19 1a  5  3  8  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
   8404  35 1f 35 58 19 1a  5  3  8  0 26  0
   8405  35 1f 35 58 19 1a  5  3 ff ff a6  3
   8406  ff ff ff ff ff ff ff ff ff ff a6  1
   8407  ff ff ff ff ff ff ff ff ff ff a6  1
   8408  ff ff ff ff ff ff ff ff ff ff a6  1
   8409  ff ff ff ff ff  0  5  3  8  0 26  0
   8410  36 1f 35 58 19 1a  5  3  8  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0

  10352  36 1f 35 58 19 1a  5  3  8  0 26  0
  10353  36 1f 35 58 19 1a ff ff ff ff a6  3
  10354  ff ff ff ff ff ff ff ff ff ff a6  1
  10355  ff ff ff ff ff ff ff ff ff ff a6  1
  10356  ff ff ff ff ff ff ff ff ff ff a6  1
  10357  ff ff  0 58 19 1a  5  3  8  0 26  0
  10358  37 1f 35 58 19 1a  5  3  8  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  12299  37 1f 35 58 19 1a  5  3  8  0 26  0
  12300  37 1f 35 58 19 1a  5  3  8  0 a6  1
  12301  37 ff ff ff ff ff ff ff ff ff a6  1
  12302  ff ff ff ff ff ff ff ff ff ff a6  1
  12303  ff ff ff ff ff ff ff ff ff ff a6  1
  12304  ff ff ff ff ff ff ff ff ff  0 26  0
  12305  38 1f 35 58 19 1a  5  3  8  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  14247  38 1f 35 58 19 1a  5  3  8  0 26  0
  14248  38 1f 35 58 19 1a  5  3  8  0 a6  1
  14249  ff ff ff ff ff ff ff ff ff ff a6  1
  14250  ff ff ff ff ff ff ff ff ff ff a6  1
  14251  ff ff ff ff ff ff ff ff ff ff a6  1
  14252  ff ff ff ff ff ff  0  3  8  0 26  0
  14253  39 1f 35 58 19 1a  5  3  8  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  16195  39 1f 35 58 19 1a  5  3  8  0 26  0
  16196  39 1f 35 58 19 1a  5 ff ff ff a6  3
  16197  ff ff ff ff ff ff ff ff ff ff a6  1
  16198  ff ff ff ff ff ff ff ff ff ff a6  1
  16199  ff ff ff ff ff ff ff ff ff ff a6  1
  16200  ff ff ff  0 19 1a  5  3  8  0 26  0
  16201  40 1f 35 58 19 1a  5  3  8  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  18142  40 1f 35 58 19 1a  5  3  8  0 26  0
  18143  40 1f 35 58 19 1a  5  3  8  0 a6  1
  18144  40 1f 35 58 ff ff ff ff ff ff a6  1
  18145  ff ff ff ff ff ff ff ff ff ff a6  1
  18146  ff ff ff ff ff ff ff ff ff ff a6  1
  18147  ff ff ff ff ff ff ff ff ff ff a6  1
  18148  ff 1f 35 58 19 1a  5  3  8  0 26  0
  18149  41 1f 35 58 19 1a  5  3  8  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  20000  41 1f 35 58 19 1a  5  3  8  0 26  0
```

TABLE 6: RTC data dump from 286 computer, year date 2000

```
Generated by RTCSCAN vs 0.60
RTCSCAN: RTC Data File Scanner
Copyright 1998 Michael D. O'Connor

            RTC Data Scan Results
-------------------------------------------------
 Record  se sa mn ma hr ha dw dy mo yr st  err
      1  07 1F 59 58 19 1A 00 28 11 99 26  0
      2  07 1F 59 58 19 1A 00 28 11 99 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
   1069  07 1F 59 58 19 1A 00 28 11 99 26  0
   1070  07 1F 59 58 19 1A 00 28 11 99 A6  1
   1071  07 1F 59 58 19 1A 00 28 11 99 A6  1
   1072  FF FF FF FF FF FF FF FF FF FF A6  1
   1073  FF FF FF FF FF FF FF FF FF FF A6  1
   1074  FF FF FF FF FF FF FF FF FF FF A6  1
   1075  FF FF FF FF FF FF FF FF FF FF A6  1
   1076  FF FF FF FF FF FF FF FF FF FF A6  1
   1077  FF FF FF FF FF FF FF FF FF FF A6  1
   1078  FF FF FF FF FF FF FF FF FF FF A6  1
   1079  FF FF FF FF FF FF FF FF FF FF A6  1
   1080  FF FF FF FF FF FF FF FF FF FF A6  1
   1081  FF FF FF FF FF FF FF FF FF FF A6  1
   1082  FF FF FF FF FF FF FF FF FF FF A6  1
   1083  FF FF FF FF FF FF FF FF FF FF A6  1
   1084  FF FF FF FF FF FF FF FF FF FF A6  1
   1085  FF FF FF FF FF FF FF FF FF FF A6  1
   1086  FF FF 59 58 19 1A 00 28 11 99 26  0
   1087  08 1F 59 58 19 1A 00 28 11 99 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
   5772  08 1F 59 58 19 1A 00 28 11 99 26  0
```

TABLE 7: RTC data dump from 286 computer, year 1999

```
Generated by RTCSCAN vs 0.60
RTCSCAN: RTC Data File Scanner
Copyright 1998 Michael D. O'Connor

            RTC Data Scan Results
-------------------------------------------------
 Record  se sa mn ma hr ha dw dy mo yr st  err
      1  22 1F 38 58 17 1A 00 28 11 00 26  0
      2  22 1F 38 58 17 1A 00 28 11 00 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
   2842  22 1F 38 58 17 1A 00 28 11 00 26  0
   2843  22 1F 38 58 17 1A 00 28 11 00 A6  1
   2844  22 1F 38 58 17 1A 00 28 11 00 A6  1
   2845  22 1F 38 58 17 FF FF FF FF FF A6  1
   2846  FF FF FF FF FF FF FF FF FF FF A6  1
   2847  FF FF FF FF FF FF FF FF FF FF A6  1
   2848  FF FF FF FF FF FF FF FF FF FF A6  1
   2849  FF FF FF FF FF FF FF FF FF FF A6  1
   2850  FF FF FF FF FF FF FF FF FF FF A6  1
   2851  FF FF FF FF FF FF FF FF FF FF A6  1
   2852  FF FF FF FF FF FF FF FF FF FF A6  1
   2853  FF FF FF FF FF FF FF FF FF FF A6  1
   2854  FF FF FF FF FF FF FF FF FF FF A6  1
   2855  FF FF FF FF FF FF FF FF FF FF A6  1
   2856  FF FF FF FF FF FF FF FF FF FF A6  1
   2857  FF FF FF FF FF FF FF FF FF FF A6  1
   2858  FF FF FF FF FF FF FF FF FF FF A6  1
   2859  FF FF 38 58 17 1A 00 28 11 00 26  0
   2860  23 1F 38 58 17 1A 00 28 11 00 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
   5772  23 1F 38 58 17 1A 00 28 11 00 26  0
```

TABLE 8: RTC data dump from 286 computer, year 2000

Tables from Flint's modified ASM software

```
Generated by RTCSCAN vs 0.60
RTCSCAN: RTC Data File Scanner
Copyright 1998 Michael D. O'Connor

            RTC Data Scan Results
-------------------------------------------------
LineNo.  se sa mn ma hr ha dw dy mo yr st  err
      1  37 5f 51  0 15  0  2 20  2  0 26  0
      2  37 5f 51  0 15  0  2 20  2  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  15326  37 5f 51  0 15  0  2 20  2  0 26  0
  15327  37 5f 51  0 15  0  2 20  2  0 a6  1
  15328  37 5f 51  0 15  0  2 20  2  0 a6  1
  15329  37 5f 51  0 15  0  2 20  2  0 a6  1
  15330  37 5f 51  0 15  0  2 20  2  0 a6  1
  15331  37 5f 51  0 15  0  2 20  2  0 a6  1
  15332  37 5f 51  0 15  0  2 20  2  0 a6  1
  15333  37 5f 51  0 15  0  2 20  2  0 26  0
  15334  38 5f 51  0 15  0  2 20  2  0 26  0
      0  -- -- -- -- -- -- -- -- -- -- --  0
  20000  38 5f 51  0 15  0  2 20  2  0 26  0
```

TABLE 9: RTC data dump from Author's Pentium Laptop, year date 2000

5.4 TD Related Theoretical Causes Disproved

Some web sites have posted different theories as to causes of the TD anomaly. This section attempts to disprove some of these theories. Some of these theories have insufficient grounds for consideration because if true, the anomaly would no doubt have a wider commonplace in computers.

> *A well-known design flaw in most versions of MSDOS means that if a PC is left running over two or more midnights* [sic]*, without a MSDOS date/time function being used in between, then only the first midnight will be seen. The RTC is correct, but not consulted. At reboot, the RTC will be read, and the date will be right. Any report of days missed is a dubious TD case.*[66]

This is correct; it is not a TD case at all.

> *If TD occurs, the CMOS memory may also be corrupted… [t]he HHD* [sic] *boot sector may also be, or appear to be, corrupted.*[67]

If this is true, it changes the whole scope of the TD anomaly. But CMOS memory or boot sector corruption are separate issues having nothing to do with TD. We are therefore inclined to dismiss it.

There was an early report that Mr. Crouch's affectionately named "Zoom" computer, which is the computer TD was first discovered on, had a failed COM port ostensibly as a result of TD. However, there is no conclusive proof that this is related to TD, even though it has been established (in newsgroup messages) that returning the Zoom computer back to the original date made the problem go away. This doesn't necessarily mean it was caused by the TD anomaly, because reversed diagnostic logic is hardly proof of a scenario if it cannot be proved with forward diagnostics. Sometimes, however, reversed logic is the only viable method for fault isolation, when there is no repeatability using forward diagnostics. What is of import is repeatability, however. If it cannot be repeated, a definitive conclusion can be hard to defend if based on reverse diagnostic logic.

Further research has shown that the reported COM port failure could *not* be duplicated although the TD anomaly was allegedly duplicated. Jace Crouch, the owner of the "Zoom" computer, implicitly states this himself.[68] This revelation is repeated in Echlin's technical paper on TD Theory.[69]

It is therefore inferrred that the COM port failure is a separate problem unrelated to the TD anomaly. The boot sector corruption is also a separate problem.

> *It has been suggested that a weak or dying CMOS retention battery may be involved. But, supposedly, the first effect of a weak battery should be that the clock oscillator does not run, well before setup data is lost.*[70]

The weak or dying battery has been ruled out as a cause of the TD anomaly. It is a misleading cause of a *similar* symptom of a retrogressed time anomaly. The symptom does not point to an actual TD anomaly. A weak battery will never cause the time or date to advance into the future, as the TD anomaly is characteristically purported to have done.

In a battery-backup situation in an old 286 system (computer turned off), only two active components remain powered by a battery: an RTC chip (and its CMOS memory) and a hex inverter chip. Battery power is also fed to discrete components that make up an oscillator: resistor, capacitor, and crystal. The hex inverter chip is used to boost the oscillator amplitude sufficient to drive the RTC, as well as assist in the generation of the oscillation. The frequency of the oscillator is 32.768 kHz, which is then divided down to 1 Hz by internal dividers inside the RTC. The one-hertz (Hz) signal increments the seconds of time in the RTC.

All together, the circuit draws very little current, mostly because of CMOS technology. It is the CMOS memory that actually requires less current draw than does the discrete oscillator circuit. This is the reason the oscillator tends to shut down first before the CMOS memory contents are corrupted, assuming of course that the battery current is the right amount for this to occur. As a result, the date/time information within the RTC freezes, with the ultimate result that the computer starts off with the same date/time on boot up as just before it was shut down. This leads to a false conclusion that TD anomaly is being exhibited.

However, this is not to say that all computers of this class have the same characteristics so described, because the implementation of RTCs and related circuitry are not the same in all computers.

There is a small margin of current draw between oscillator failure and CMOS memory corruption, and it is different in every computer design. In addition, battery construction and technology, as well as age, are factors affecting the characteristics so described. For example, a dying battery may discharge faster than others, thus closing the "gap" between CMOS memory retention loss and oscillator failure. In fact, in the author's own experience on one computer, memory corruption soon occurred shortly after system boot. At that point, the battery voltage was only 1.3V. Therefore, the above excerpt is not entirely true, and certainly would not be true on all computers.

This author is the very person who reported on this very anomaly in December 1997. The battery was replaced on the computer on which this phenomenon occurred, with nothing more to be said for any other anomaly, much less related to TD. It's a very interesting phenomenon revealing the characteristics of CMOS technology versus discrete components, and it's nothing more than the difference in bare, raw minimum current requirements in between them. The phenomenon was heavily dependent on just the right amount of current draw from a slow-dying battery, before falling to a level where CMOS memory becomes corrupted.

> *…But the data sheet says that there are THREE ways to avoid Update* [Period]
> *After the Update-Ended interrupt, data is good for >998ms (or after waiting for UIP to fall).*
> *UIP flag, 244us warning + 1984us Update – hold off if flag is up.*
> *Periodic Interrupt timing.*[71]

This information is correct. In the same section, a conclusion is given:

> ***Conclusion*** *1998-11-21 (after raising the point here and everywhere) : however, the general belief appears to be that only UIP is used, never PI.*[72]
> [Bold original]

This is also correct, but for an RTC chip such as the 146818, the fact is that the UIP method ***must*** be used in a PC, because this is an industry standard configuration for PCs. The other two methods are affected by hardware reset on computer boot up. The UIP method is the only one not affected by it. A hardware reset will cause the RTC to lose sync of the correct date/time if a method other than UIP is used. A little more study of the RTC technical data sheet will bear this out. In fact, the time-keeping portion of the RTC chip is the only hardware piece in the entire PC that must not be reset during boot up, or any other time, for that matter. It is true that certain parts of RTC registers are initialized at boot up. To recap, the following is not affected by hardware reset when the UIP method is used:

- Status register A (includes UIP bit)
- CMOS RAM
- Certain bits in status registers B & D affecting configuration of RTC operation

The above describes a non-buffered RTC. For a buffered RTC, it is still necessary to honor the UIP during RTC access to data, even though there is no bad data in the UIP.

To use any method other than UIP requires a rewrite of the BIOS code, and is a deviation from the industry standard. However, some BIOS chips support buffered RTC chips that require discrete instructions to "freeze" the time/date information.

> *The stated mechanism for plain TD is that the RTC is perturbed by being addressed during its update cycle; if the RTC is read, then data may/will be incorrect. Remember that to read the time/date, it is necessary to write the sub-address to 070h. Now if there is an attempt to write (internal/external source), this could conceivably lead to an internal "bus fight" where one bit of CMOS logic is asserting "0" and another "1" on the same internal line(s).*[73]

Regarding the addressing of the RTC causing perturbations during the update cycle, this is not true. The RTC technical data sheet states clearly that the CMOS RAM contents are fully available at any time, including the update period. First the desired address is specified at port 070h, then port 071h is used to either receive or send data based on that address.

However, to avoid reading RTC date/time data during the update period, the UIP bit must be honored. This is because data retrieved during the update period will tend to be corrupted.

Regarding the "bus fight" causing conflicts on the same address line, the RTC data is sent over the address lines only when instructed to do so. There is no case of a "bus fight;" *but* such a conflict can still occur if two consecutive code instructions were sent over the same bus, such that they occur almost at the same time. If I/O delay periods are not inserted between consecutive RTC accesses, such a "bus fight" may occur, especially on faster computers.

Nevertheless, the "bus fight" due to consecutive lines of code accessing the RTC can easily be prevented by prudent use of a short line of code. In assembler, this code is: †

```
JMP SHORT $+2
```

This code does nothing but delay the execution of the next line of code by 0.5 microsecond or so (depending on computer speed, in this case a 386 at 20 MHz). This code is effective in isolating the simultaneous RTC bus activity and prevents I/O conflicts. However, on faster computers (400+ MHz), it becomes necessary to use a specialized routine to first calculate the speed of the computer, then use a computed factor based on computer speed as a delay period. This is because even a long series of *JMP SHORT $+2* instructions is not enough delay.

The emergence of the TD anomaly across different classes of computers (286 to non-PCI Pentiums) is inversely proportional to the increasing need for an I/O delay, or, in fact, to the speed of the computer. In other words: TD anomaly tends to be more populous on older, slower computers than on newer, faster computers, while "bus fights" may tend to occur on faster computers but not so prevalent on slower computers. Hence, it is illogical that a "bus fight" would be a cause for the TD anomaly.

If system interrupts are not disabled during RTC access, valid data is not guaranteed because a system interrupt may occur and disrupt the access to valid RTC data. Such is the case for a "bus fight," although not an accurate term. This is easily avoided by disabling the system interrupts during RTC access.

However, when we talk about RTC access time for date/time information, there is a direct, not inverse, proportional relationship between the classes of computers and the RTC access time. On slower computers, the access time is longer than on faster computers. For the TD anomaly, it tends to be more populous on slower, older computers than on faster computers, based on statistical account given by Mike Echlin.*

This does not mean a TD anomaly is more likely to occur simply based on longer RTC access times. We already know that the RTC access time is not a primary factor for the TD anomaly, as this report has shown. If the UIP bit is honored by the BIOS code, no case of TD anomaly should occur.

---

† Another valid delay instruction is the NOP instruction, which "wastes" 3 clock cycles on a 286 computer before the next instruction is executed.

* Based on Echlin's tests on 100 computers. See Section 4.0.

…On 1998/10/07, I had an earlier thought: a bus fight will draw substantially increased current from Vcc, because of the nature of CMOS logic. Internal impedances are likely to prevent this overcurrent from being physically destructive. However, the current may cause a sudden drop if [sic] *the internal Vcc, particularly as the Vcc supply may be (I think) multi-sourced from battery and PSU* [power supply unit] *through diodes and otherwise comes from the low-current battery. The drop may or may not be detectable on the external Vcc pin.*[74]

Regarding excessive current (overcurrent) occurring during "bus fight," the author assumes reference is made to the voltage drop on the RTC Vcc pin. An erroneous assumption is being made that an increase in current in the internal Vcc will not be seen at the external Vcc pin. If an excessive current draw is being made internally, it will be seen at the Vcc pin. The voltage drop, however, will not be seen because the power supply can deliver more current on demand. In addition, a "substantially increased current" will likely be rather small in comparison, because CMOS circuitry draws very little current.

The RTC is a high-impedence IC chip. As long as bus input/output voltages do not exceed the range of $V_{ss} \leq (V_{in}$ or $V_{out}) \leq V_{cc}$ (V_{ss} being signal ground), there will be no problems even if a "bus fight" occurs.

Besides all this, it does not follow that a TD anomaly would occur by something so random a "bus fight" as would occur only during boot up. The nature of the TD anomaly itself rules this out.

There are other factors against this voltage drop due to excessive current draw theory:

1. It is an engineering design mistake (taboo) to fit a power supply with the same current rating as that required by the PC design. Power supplies in computers always have a current rating far exceeding the power requirements of the PC, so that extra cards, hard drives, and other devices may be installed without exceeding the power supply capacity. This capacity is usually expressed in watts. Hence, any such "overcurrent" condition is easily covered by the power supply, and therefore no such voltage drop will be seen.
2. Even if excessive current draw occurs, the change in current level is likely to be small.

3. Diodes are in place to prevent reverse current flow to either power source when primary power is interrupted or applied.
4. In a battery-backup environment (computer turned off), there is no possibility of excessive current because during battery operation the RTC address lines are disabled internally.† In addition, there are no chips other than the RTC and hex inverter chips powered by the battery.
5. Any "bus fight" activity on the system bus may cause the computer to hang.

Therefore, excessive current draw theory is ruled out. It is too much to expect an over-current condition to occur simply because of a "bus fight" in a system for which a power supply can easily supply on demand.

> *If the CMOS RAM is exposed to sudden changes in internal Vcc, especially at a time when other parts of the chip are active (some improperly), it may have every excuse for making errors in memory; this need not require a drop from full Vcc to the minimum sustaining voltage.*[75]

Finally, regarding CMOS RAM exposure to sudden internal supply voltage changes as a cause for CMOS memory corruption, the author respectfully disagrees. The above excerpt refers to a condition similar to a "brownout" condition – as opposed to a "blackout" condition usually observed in homes without power.

Every computer power supply unit use DC voltage regulation to eliminate voltage shifts.

The TD anomaly symptom is very specific: a changed time/date information only during computer boot. No such internal voltage changes would cause such a corruption in time/date information in the same location in memory without effecting other areas of memory. This is not only untenable, but also impossible.

Again, if this internal voltage drop theory were true, then we would see more occurrences of the TD anomaly, and with much more symptoms than just a changed date/time information. Therefore, this theory is forthwith dismissed as a causative factor for TD.

> *Never allow midnight to occur during early booting.*[76]

† Stated in Motorola's Technical Data on RTC part number MC146818 and on Japanese version (Hitachi) part number HD146818

This sound advice is thwarted by the possibility that you could be booting the computer at a time when the computer thinks it's midnight. It's up to you to know when the computer expects to see a midnight rollover. However, the margin of time in which the midnight rollover bug occurs is only 1 second before midnight. Therefore, it's reasonably safe to guess the computer's opinion on the time before powering it up. Nevertheless, the point is that you're on your own when it comes to guessing the computer's opinion of the time of day. If you trust the computer's opinion, do it.

Note that the midnight rollover bug is not related to the TD anomaly.

…I wonder what happens if there is no working RTC? [77]

POST will fail, if POST expects to see an RTC. Some old computers (XT computers in particular) do not have an RTC, which requires a BIOS chip with POST overlooking the RTC on startup.

5.4.1 Miscellaneous

I have seen mention of a PC in which at rollover the century byte was incremented from nibble '19' in a binary (not BCD) fashion to '1:'. Ouch! [78]

This is not a bug in the operating system, but a bug in the application program, namely Windows File Manager (MS Windows 3.1) and third party applications. The bug also shows up on an old DOS-based program called LIST, written by Vernon D. Buerg, which the author still uses extensively. The software which exhibit this behavior still properly handles the year date information; it is nothing more than a cosmetic bug because it is not being displayed correctly.

2000-01-01 must be a tempting date to use for a virus trigger, and the virus might simulate TD or other Y2K problems. [79]

Interesting prediction, and it may be true. The existence of such viruses has recently been aired (mid-December 1999) through a television news report in Washington, D.C.*

* At this release, however, no such reports of virus outbreaks ever surfaced.

Based on published sources, they are not interested in the old ways of virus propagation. They are more interested in propagation through the Internet, not as much through the usual diskette or program distribution. They are looking for Internet "piggy back" methods through which viruses can propagate easily and exponentially through distribution. The Melissa virus is a perfect example of this, because Word documents are distributed between parties miles apart, even across the world.

Note: this author is *not* a virus writer and was never actively involved in any dialog with virus writers. He has researched the subject as part of his job in 1998 to fulfill a customer requirement to recommend a LAN security methodology against virus invasions. In 1990, he has written an old virus from psuedocode published in a book in an attempt to understand virus technology as well as enhance programming experience.

In the author's opinion, viruses are a bigger threat than the so-called TD anomaly. We have nothing concrete to justify worrying about in regard to TD. It's real enough, but not big enough. Some people and corporations do not even believe the TD anomaly exists.

As far as the author knows, there are only two computers confirming to exhibit an advanced time or date anomaly, and that is Mr. Jace Crouch's affectionately called "Zoom" 286 computer and Echlin's Zenith 286 computer. Yes, there were reports of other computers exhibiting TD anomaly, but description of the anomaly itself (even to identify whether it is *advanced* or *retrogressed*) is sorely lacking. Furthermore, the lack of proper definition of the TD anomaly itself tends to lead to false reports by people who do not understand it. Therefore, these ambiguous reports can only be treated the same way second-hand information is treated.

5.5 DOS Interrupt 21 Set Date and Set Time Software Quirk

> *1998-12-08 ff: …SetDate [DOS Interrupt 21h, Function 2Bh] may **also** write the DOS time to the RTC (and SetTime [DOS Interrupt 21h, Function 2Dh] the date : MSDOS does it, but not ROM BIOS.*[80]
> [Bold original]

Worthy of mention is the above excerpt from Stockton's web site, which claims that the MS-DOS set-date routine not only sets the date in the BDA and RTC CMOS memory, but also sets the time of day. The documented purpose of this routine is to set the date only.

The same is true for MS-DOS set-time routine, which not only sets the time of day in the BDA and RTC CMOS memory, but also sets the date. The documented purpose of this routine is to set the time of day only.

The DOS commands DATE and TIME, when setting the date or time, calls interrupt 21h/2Bh and 21h/2Dh, respectively, through the DOS kernel.

Stockton's test program, INT_TEST.PAS,* without a doubt proves the above excerpt to be true. In the author's tests on a 386 system with MS-DOS version 6.20, the set-date routine does indeed write the value of the DOS time in the BDA to the RTC CMOS memory in addition to the date. However, on a Pentium laptop with Windows 98 (version 4.10.1998), the set-date routine will cause the opposite reaction: the RTC time of day will be written to the DOS time of day in the BDA.

According to Stockton's web site, the software quirk is present from MS-DOS/PC-DOS version 3.30 upwards, as well as variants of the DOS operating system written by other software companies.

None of this has any bearing on the TD anomaly. However, for those mystified by Jace Crouch's warning[81] that the RTC may be "accidentially" set with the wrong time of day (or date) after TD occurs, Stockton's description of the software quirk is where the warning comes from. For example, setting a date will cause the errant time of day (caused by TD anomaly) to be written to the RTC. The result is a perpetuated errant time of day on every computer boot, because the RTC now contains this errant information.

---

* This source code is available at Stockton's web site; see the endnote associated to the excerpt.

Bear in mind, however, that if a wrong time or date is noticed by a user, the errant time or date can corrected by changing it directly, thus removing the problem. A TD anomaly with both incorrect time and date is not likely, since the time and date is retrieved from the RTC individually during computer boot. Hence, the secondary data (time or date) will probably be correct, as long as the corrective action is directed towards the errant primary data (time or date).

Another remedy to the TD anomaly is to simply reboot the computer, since the RTC should still contain the correct date and time information. This assumes, of course, that the set date and set time functions have not been executed.

It is not considered likely that any program application will blithely (casually) set the date or time without user permission or interaction, especially if user input is required.

6.0 RTC Register A Status Byte

This section briefly explains the meaning of the status byte values in Register A of the RTC. It was added in this report because no one has explained it to the public. In Tables 1 through 8 in Section 5.3.3 of this report, RTC data dump listings show the status byte (in "st" column) alternating between two values, 26h and A6h. Table 1 below shows the map of Register A of the 146818 RTC chip.

Byte	MSB							LSB
Value	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
	UIP	DIV2	DIV1	DIV0	RSEL3	RSEL2	RSEL1	RSEL0
26h =	0	0	1	0	0	1	1	0
A6h =	1	0	1	0	0	1	1	0

TABLE 1: RTC Register A Map

Divider bits (DIV2, DIV1, DIV0) select the RTC time base frequency, which is set to 32,768 kHz. This sets the update period to 1984 microseconds. This frequency is fed to the RTC input from the external oscillator, and is then divided down to 1 hertz by the 22-stage divider chain inside the RTC. The 1-hertz signal is then used to update the seconds field inside the RTC. These bits are not affected by hardware reset.

Rate Selector bits (RSEL3, RSEL2, RSEL1, RSEL0) select one of 15 taps on the 22-stage divider. It also sets the periodic interrupt rate and the square wave output frequency based on the selected time base. These bits are not affected by hardware reset.

Except for the UIP bit, none of the bits ever changes. They are always the same across all computers that use the 146818 RTC chip or equivalent. Hence, the UIP bit is the only bit in Register A that changes during operation. This is the reason there are only two possible values in the status byte as shown in the RTC data dump listings.

Regardless of the settings and any other settings, the "grace period" window is always the same – 244 microseconds.

The UIP bit is set active high 2228 microseconds out of every second. This is sum of the "grace period" window and the update period (see Figure 2 in Section 4.0).

7.0 Typical Computer Boot Process

This section describes the general sequence of events in a native DOS boot process (not Windows 95/98/2000/NT) starting with computer power up and ending with the DOS prompt. Advanced power management system in some computers is not covered. Note that the DOS system file MSDOS.SYS is *not* a text file (Windows 95/98/2000/NT only) but a binary program delivered in MS-DOS operating system software packages.

While the boot process itemized herein is detailed and is tabulated from a set of reference material, the intent is to show the relative sequence positions of time and date retrieval from the RTC during a computer boot up. The actual boot sequence will vary from system to system depending on system platform, the system BIOS, and DOS version. The time retrieval and date retrieval sequences are listed in bold.

When a computer is turned on, the Power-On-Selt-Test (POST) routine is the first routine to be executed. It is part of the ROM BIOS. POST error indication is usually in the form of a series of long and short beeps. A single beep sound indicates all system components have passed the POST.

In AT and previous class computers, the BIOS firmware, including POST routine, is stored in ROM. This means this device can be read from, but not written to. It also means the BIOS must store variable values in RAM, not in the ROM. In fact, the BDA generated by the BIOS routines during computer boot is stored in low RAM.

Each version of DOS contains patches to some routines and tables in the ROM. Naturally, the patches cannot be written to the ROM. Instead, DOS reads the ROM information and stores them in RAM, applies the patch(es), and refers to this modified ROM information in RAM instead of ROM. This action takes place after POST.

Table 1 shows a typical DOS memory usage on a computer that has 1 megabyte of RAM. This amount of RAM became available with the introduction of AT class computers, including the 286 computer.*

* 1 MB RAM is available as addressable memory space in real mode. In protected mode, 16 MB RAM is available, provided physical RAM is present in the system.

POST does *not* convert the date from the RTC into hex and store it as number of days since 1 January 1980 in the BDA. Instead, POST merely *clears* this BDA location as part of the initialization routine. After POST completion, DOS requests the date from the RTC through BIOS Interrupt 1Ah service, converts it to a single hex value, and stores it in the BDA. The BIOS Interrupt 1Ah service routine does *not* perform this date conversion; it is the *DOS kernel* that performs it.

This revelation delivers the final *coup de grâce* to the logic path theory defined by Mr. Echlin.

During POST, the BIOS performs a conversion routine for the time information from the RTC and stores it as 32-bit timer ticks in hex in the BDA.* The timer tick increment is maintained by Intel 8253 or 8254 Programmable Interval Timer chip or equivalent, whose output runs at 18.2 Hz and is tied to Interrupt 8. The timer chip controls the BIOS timer tick routine via Interrupt 8. The BIOS timer tick routine, in turn, increments the tick counts in the BDA via Interrupt 1Ch.

The following boot process takes place in this order (note that not all BIOSes adhere to the same sequence):

1. Power to the computer is switched on.
2. The CPU registers are cleared of leftover data by an electrical signal in a permanently programmed path in the CPU. This signal resets a CPU register (program counter) to the ROM BIOS power-up entry point: FFFF0h (AT+ and later class computers).
3. The CPU starts processing the boot program, which in turn invokes the Power On Self Test (POST, at ROM BIOS entry point FE05Bh).
4. The CPU checks itself for functionality of flags, registers, and conditional jumps.01
5. ROM checksum (POST, BASIC, and BIOS) is computed and verified against a value stored in ROM.
6. 8254 Timer chip is checked and initialized.
7. 8237 DMA Controller is initialized and registers checked (all channels).
8. RAM refresh initialized and started.
9. First 64K of RAM is tested.
10. Video display is initialized (display only).
11. Refresh rate and speed is checked.
12. CPU GDT and IDT registers are initialized and checked (protected mode).
13. 8259 Interrupt Controllers are initialized.

* *The Undocumented PC, 2nd Ed.,* (pg. 255) incorrectly attributes RTC time conversion into tick counts to the "operating system" during POST. It is actually performed by the BIOS firmware during POST. However, DOS takes over the tick counts management when DOS initialization is completed.

14. CMOS memory checksum is checked. The battery voltage level is also checked.
15. SYSINIT routine is called to build the IDT & GDT and to switch to protected mode.
16. RAM is checked in 64K chunks, including extended RAM, and sets expanded RAM size (protected mode).
17. Return to real mode.
18. Presence of video ROM is checked and type of video adapter is checked. The keyboard self-test results are examined.
19. Video adapter and RAM is initialized and tested.
20. 8259 Interrupt Controller is tested for interrupt handling.
21. 8254 Timer checked for proper frequency.
22. SYSINIT is called to switch to protected mode.
23. Additional RAM tests after the first 64K is performed, including extended RAM (protected mode). The "0XXXX OK" message will be displayed.
24. Additional 286 CPU protected mode tests are performed.
25. Keyboard is tested.
26. Diskette presence is checked. Here the diskette drive motor(s) are turned on. The following items are also performed:
    - KB buffer initialized
    - Printer timeout initialized
    - RS232 initialized
    - Timer interrupts enabled
    - CMOS battery & checksum checked, UIP checked
    - CMOS memory configuration match with existing configuration checked
    - Fixed disk checked
    - Floppy drive initialized
    - 2nd diskette drive checked (motor turned on)
    - verify CMOS config matches fixed disk setup
    - Fixed disk initialized
27. Additional ROM in additional devices checked and initialized. I/O devices (RS232, printer) set up and initialized. The following items are also performed:
    - Printer base addresses checked
    - RS232 card presence checked and initialized
    - Get time/date to check CMOS clock validity
    - Enable hardware interrupt if co-processor present
    - Enable keyboard/timer interrupts
    - Initialize KB type & Num lock state
28. Short beep sounded for successful POST completion
29. **Time of day read from RTC, converted to binary, then stored in BDA.**

30. POST exits. Bootstrap loader (Interrupt 19h) executes to start the system itself by reading the boot sector code from the boot sector of either the diskette in the diskette drive (track 0, head 0, sector 1), or otherwise the fixed hard disk (cylinder 0, head 0, sector 1), and loads it in memory, and then executes it.

POST is now finished. At this point ROM BIOS code exits and passes control to the boot sector code. The rest of the computer boot process is handled by DOS as follows:

1. The bootstrap code looks back to the diskette (or hard drive) to locate IO.SYS and MSDOS.SYS* (IBMBIO.COM and IBMDOS.COM, respectively, on genuine IBM computers). The bootstrap code loads IO.SYS to low RAM area above the BDA. Then, depending on the system, either the bootstrap code or IO.SYS initialization routine loads MSDOS.SYS to RAM above IO.SYS.
2. Control is passed to IO.SYS. There are two parts to this code. The first part is the BIOS supplied by the system manufacturer (or by IBM on genuine IBM computers). This BIOS contains resident device drivers along with hardware-specific initialization that is run only when the BIOS is loaded. The second part is a module called SYSINIT, supplied by Microsoft.
3. IO.SYS runs any hardware-specific initializations required. During this initialization, IO.SYS checks the BDA (set up earlier by ROM BIOS initialization routine) to see what hardware is being used. Unneeded drivers may be deleted automatically, but most versions of DOS contain only bare-bones drivers in IO.SYS and does not make any changes.
4. Control is passed to SYSINIT module. SYSINIT checks available memory and relocates itself (it is currently in low RAM location) to high memory area to get out of the way to make low RAM available to DOS.
5. The high-memory copy of SYSINIT copies MSDOS.SYS over IO.SYS (thus overwriting it). This means the SYSINIT module inside IO.SYS is also overwritten.
6. SYSINIT calls DOS interrupt 21h file services (now available since MSDOS.SYS is loaded) to open CONFIG.SYS file, if it exists. The entire CONFIG.SYS file is loaded in memory, all characters are converted to upper case, and the file in memory is interpreted line by line.
7. All script commands within CONFIG.SYS are processed. All device drivers are loaded, initialized, and linked to the list of drivers maintained by the system. Default values are assigned to FILES, BUFFERS, and FCBS directives if not already specified within CONFIG.SYS .

* *Not* the Windows MSDOS.SYS text file, but the binary file shipped with MS-DOS operating system software.

8. SYSINIT calls the MSDOS.SYS initialization code, which sets up internal tables and interrupt vectors in low RAM. Then the initialization code initializes the original drivers set up by IO.SYS.
9. MSDOS.SYS contains the DOS kernel. **It is during its initialization routine that the date is read from the RTC through the BIOS, converted into days since 1 January 1980, and then stored as hex in the BDA.**
10. MSDOS.SYS checks for the number of available hard disk drives and examines the BIOS parameter block for each drive to determine the largest disk sector size for all these drives. It then uses this value to set up a disk sector buffer for use by the system.
11. MSDOS.SYS displays the DOS copyright notice and returns control to SYSINIT module.
12. SYSINIT closes all file handles and opens standard system devices (CON, PRN, AUX).
13. SYSINIT calls the DOS EXEC function to load and execute COMMAND.COM or the shell specified by CONFIG.SYS.
14. SYSINIT exits.
15. COMMAND.COM sets up interrupt vectors for interrupts 22h through 24h. Then it relocates part of itself (transient portion) to high memory, thus overwriting SYSINIT module. The other part of COMMAND.COM remains in low memory as the resident portion.
16. COMMAND.COM executes AUTOEXEC.BAT, if it exists. Afterwards, control is transferred to the transient portion and the DOS prompt is displayed.
17. The system is ready for user interaction.

The resident portion of COMMAND.COM (in low memory) contains code for restarting COMMAND.COM when control is returned to it. It also contains three interrupts: 22h (terminate address), 23h (Ctrl-C), 24h (Critical Error). The high memory portion (transient) contains code for internal commands and batch file processing.

With this boot procedure, several things are revealed:

1. During POST, only the time data is directly read from the RTC via the POST code.
2. During DOS kernel initialization, only the date data is read from the RTC, but this is done through the DOS kernel via a BIOS interrupt. POST is already long gone, and ROM code is already long finished.
3. This means there are two separate code sections for reading the RTC data – one for time (POST), one for date (DOS through the BIOS). (Jace Crouch's statement seems to fit here: "Mike told Intel to examine one section of the POST code, and Intel examined a different section of the BIOS code.")

4. This finding fits a pattern with the TD anomaly that reportedly exhibited excessively advanced or retrogressed date but *with the correct time of day*. Yet the RTC itself reportedly showed the correct time *and date*.

5. This finding points to *the BIOS as the culprit* for the anomaly. The culprit is not the DOS kernel because the BIOS is responsible for proper RTC access. The kernel simply calls the proper BIOS interrupt to get the date.

6. Intel's statement in Appendix A of its report that "the BIOS executes its conversion routine only at POST, not during an Interrupt 1A routine" is true. It refers only to the time conversion routine, not the date conversion routine. This fits the context of Intel's statement in its report – which context refers to the *time*, not the date.

7. DOS executes the date conversion routine, *not the BIOS*. This shows that Intel's statement (see above), while misleading because of its inherent ambiguity, was not intended to be misleading.

8. The above finding means all of the date information must be retrieved by DOS before date conversion can be initiated. That means the RTC read operations would be finished before the conversion. This kills the "logic path" theory.

System ROM access time is slower than the RAM access time. It was thought by Echlin and Bossert that the slower access time contributes to longer RTC access time during POST. But this is ruled out because the system ROM BIOS is not involved when date conversion is performed. True, date retrieval is performed by the BIOS code in ROM through BIOS interrupt vectoring, but the BIOS code does not perform any date conversion.* All it does is return the RTC date information in BCD format. In fact, *this is a documented function return of the BIOS Interrupt 1Ah* function 04h to obtain the RTC date.

References:
DOS Power Tools, 2nd Edition, Revised for DOS 5.0, Paul Somerson
DOS Power Tools, 2nd Edition, Paul Somerson
System BIOS for IBM PC/XT/AT Computers and Compatibles, Phoenix Technologies Ltd.
Microsoft MS-DOS Programmer's Reference, Microsoft Press
DOS Programmer's Reference, 3rd Edition, Dettman and Johnson
DOS Programmer's Reference, 2nd Edition, Dettman
The Programmer's PC Sourcebook, 2nd Edition, Thom Hogan
The Undocumentd PC, 2nd Ed., Frank van Gilluwe
IBM Technical Reference, 1985, IBM Corp.
Last but not least, *Analysis of the Crouch-Echlin Effect, Intel Corporation* (Appendix F only)

* As confirmed through examination of published BIOS code.

Address	Memory Usage
FFFFFh	Reserved for BIOS
E0000h	Unused
CC000h	Disk adapter, BIOS
C8000h	Used by video adapters
A0000h	Transient Program Area (application programs)
	COMMAND.COM (Resident)
	Disk buffers, installable drivers (if any)
	DOS kernel (MSDOS.SYS)
0400h	ROM BIOS parameter area (BDA)
0000h	Interrupt Vector Table

TABLE 1: Typical Memory Map on Computers With 1 Megabyte Memory

8.0 Corporate Dilation

"Dilation" is certainly present in corporations when it comes to the TD anomaly. For example, a Compaq FAQ on their web site stated that Compaq attempted and failed to find a TD anomaly on Compaq systems despite a thorough engineering review. Compaq disclosed that they worked closely with Intel Corporation during Intel's own investigation of the Crouch-Echlin Effect, and is in agreement with Intel's findings that no such anomaly was observed. [82]

However, a truncated excerpt of an email message tells a different story (see next page). According to a strong proponent for TD, technicians at Compaq/Digital duplicated the TD anomaly and publicly repeated statements that they did so for months. [83]

After Compaq/Digital was acquired by Behemoth Computer Corporation, these repeated reports have eventually been "throttled." [84] As a result of this action, one of the DEC employees – who publicly stated his own findings – lost his job, and another employee was re-assigned to another position. The reason for this corporate about-face? According to the same strong proponent of TD, BCC has their own lawsuit regarding Y2K RTC chips. They don't want another one, despite the fact that the RTC is not the culprit for the TD anomaly – as this report has without question proven.

However, despite the initial foot-shifting of corporate stance on the TD anomaly, all previously-involved corporations now have a standard position that they all agree on: There is no such thing as a Crouch-Echlin Effect.

Other companies who have spent time, effort, resources, and money to even *find* a computer that exhibits the TD anomaly have eventually given up and considers the TD anomaly a fiasco based on hype and hysteria. One conglomeration of companies who were part of a Y2K consortium in New Zealand, including WANG and NEC, along with their client base have tested thousands of computers without a *single case* of TD anomaly.

According to a newsgroup message by Mr. Mike Echlin, several companies and persons have reported a TD anomaly. However, there is not enough information in the description of these reports showing the TD anomaly is in fact a true TD anomaly, or showing that those reporting the anomalies *understood* a TD anomaly when it is found. As mentioned in this report earlier, a TD anomaly report could be from a *false case* of a TD anomaly. There is only one exception to this: a report given by a Compaq/Digital employee (now ex-employee) giving explicit details on the only computer so far confirmed as a TD computer: Mr. Jace Crouch's infamous "Zoom" 286 computer.

```
From: Stuart Greenfield <address deleted>
Subject: Crouch Echlin Effect: TD Tools
Date: 18 Oct 1998 00:00:00 GMT
Message-ID: <deleted>
Distribution: world
Content-Transfer-Encoding: 8bit
Content-Type: text/plain; charset=ISO-8859-1
X-XXMessage-ID: <deleted>
Organization: Comptroller of  Public Accounts
Mime-Version: 1.0
Newsgroups: comp.software.year-2000

  I received the following from a Digital rep and thought I'd pass it
along. It would appear that TD is now of concern to the major PC vendors.

From: Barry Pardee <address deleted>
Subject: Crouch Echlin Effect: TD Tools
Date: Fri, 16 Oct 1998 15:59:46 -0400
MIME-Version: 1.0

       We at Compaq and Digital have confirmed that the Crouch Echlin
Effect, also referred to as Time Dilation (TD), is real and is a potential
threat to PCs, servers, and embedded systems that use unbuffered real time
clocks.

       Although we have not seen any TD symptoms on any Compaq or Digital
PCs, many of our Customers have a mixture of various brand PCs that should
be checked.  Also, the use of non-manufacturer parts for repairs or upgrades
may introduce potential TD problems into any PC.

       Hundreds of requests for the TD Tools (as described at the following
URL) have been received by Compaq/Digital, along with reports of similar
Time/Date anomalies, during post Year 2000 testing.

       <URL deleted>

       We have made the decision to resell the TD Tools.

<Remaining text deleted>
```

Below is a humorous response to alleged Intel's and Behemoth/Digital's up-turned-nose attitude to the TD anomaly evidence and to Crouch and Echlin's theorem. Humorous, that is, to the reader, but hardly the demeanor of the narrative. Mr. Jace Crouch wrote it and placed it on his web site. This is a fine piece of literary prose; it's very well written. Obviously, the writer went through a few years of English courses in college. And why shouldn't he? He is a History Professor. This “corporate dilation” cannot be described any better.

The section said verbatim:[85]

1. *Jace and Mike's position: There was a mouse in the kitchen.*
2. *Mark's position (before he was throttled): I saw the mouse, too. And saw the droppings and realized what this mouse might be able to do worldwide. Couldn't tell who the parents were, but do know it exists.*
3. *One Y2K vendor's response: I looked in the garage 20,000 times and I didn't see anything. By the way, if you don't help me build a mousetrap I'll denounce you whenever and wherever I can. I'll tell my friends, too.*
4. *Behemoth's initial response: There is a mouse in someone's kitchen, but it's not in our kitchen.*
5. *Behemoth's second response: Yeah, but only a couple of people saw it, and not in our kitchen.*
6. *Intel's response: There is not a mouse in the dining room.*
7. *Behemoth's third response: We agree with Intel about these alleged mice, and not in our kitchen.*

It should be noted that item 1 above only claims there is a mouse in the kitchen. Of course, the parents or the hole in the wall was never discovered. This report claims a possible hole in the wall, but it has yet to be confirmed. However, as this report concludes, this mouse is a rare mouse indeed. Any computer exhibiting a retrogressed anomaly, be it time or date, is subject to skepticism, because any number of non-TD-related causes applies.

(No, it is not the mouse with a long tail connected to your computer. If that were the problem, all you need is a cat.)

9.0 Theoretical Analysis and Conclusion

As Jace Crouch willingly admits, TD is rare and occurs mostly on older systems.[86] There is no concrete concern for TD anomaly in this late day and age, because the problem is not big enough, not widespread, and not common. In addition, most people use newer computers to keep up with advances in software technology. This is particularly true in the business sector, both private and government.

However, there remains an open question as to this effect on embedded systems, since 286 CPU processors are in use. If RTC chips are also in use, and if time/date information are also used as part of the control processing in automated systems, the concern may then be justified. It should be kept in mind, however, that the TD anomaly is, based on probability analysis, still relatively rare even among these systems. But the concern would be further justified if system boots are a part of the regular routine. The TD anomaly cannot occur except through system boots.

The real issue is that some BIOS codes do not meet RTC specifications. Michael Kennedy (no relation to the famous Kennedy family), an engineer writing I/O software and who has had exposure to BIOS code, states that he has seen “4 or 5” BIOS code listings in the past and none of them fully meets the RTC specifications.

This report has revealed many factors that affect how and when a TD anomaly occurs:

1. In a computer boot, the time and date retrieval from the RTC are separately performed.
2. The time is retrieved from the RTC during POST.
3. The date is retrieved from the RTC after POST, and after the DOS kernel is loaded and initialized.
4. Based on the above, POST initializes the timer ticks in the BDA. The DOS kernel initializes the number of days since 1 January 1980 in the BDA. This sharing of responsibilities is consistent and present from 286 AT+ platforms upwards and from DOS version 3.0 upwards.*
5. A TD anomaly occurring on the same computer boot involving both the time and date is not likely, although theoretically possible.
6. The time retrieval and conversion code is executed in ROM BIOS from the ROM.

* There are exceptions: UNIX, OS/2, and Windows NT are among these. There may be other exceptions.

7. The date retrieval itself is executed in the ROM BIOS from the ROM, but conversion code is executed in the DOS kernel from the RAM.

Section 5.1.4 of this report has conclusively shown there is a problem in the time retrieval code in ROM BIOS in Echlin's 286 computer. The theoretical cause of the TD anomaly in Echlin's 286 computer is further expounded through analysis at the end of Section 5.1.4, which see. There is no evidence of date anomaly in this computer.

For Jace Crouch's 286 "Zoom" computer, there is no time anomaly, but the date has reportedly advanced itself into the future. We know that the date read and conversion procedure is executed in RAM by the DOS kernel. We also know that the logic path theory does not hold a drop of water, therefore the date conversion routine is not the culprit for this anomaly. (The DOS kernel calls BIOS interrupt 1Ah to obtain the date from the RTC, and the BIOS routine returns only BCD information, so there is no conversion routine within the BIOS service routine.) How, then, does the date advance itself? There are only three possible answer for this:

1. UIP bit is not being honored, or
2. System interrupts are not disabled, or
3. I/O delay instruction is missing.

The amount of time required to read the date is not a factor, since only 4 bytes are required to retrieve the date and no conversion is performed on-the-fly.*

A fourth possible cause of the TD anomaly on Jace's computer is that someone changed the BIOS chip with another that is not 100% compatible with system components and parameters. Determining the proper BIOS chip and replacing it is a highly technical and touchy issue. If this is true, it explains why the TD anomaly is so rare.

In both of these computers, the RTC time and date information remains correct. Both of these computers point to a problem in the BIOS code, but in different sections.

* As confirmed through examination of published BIOS code. The sticky point is that the BIOS service routine for retrieving the date from the RTC during computer boot *always return BCD numbers* for the date.

In the author's opinion, non-compliant BIOS code is the main culprit behind the anomaly that initially came to be called "Time Dilation." If a BIOS routine does not honor the UIP bit, it stands to reason that even without the alleged "logic path" theory a TD anomaly would eventually occur.

One curious factor arose from the investigation. The TD anomaly does not appear on Pentiums with PCI architecture. This architecture, as well as the EISA architecture, is designed so I/O operations cannot overrun an I/O device. On other bus architecture, this points to timing problems, of which I/O delay is a part.

We have already seen how Echlin's RTC.EXE diagnostic program returns misleading information on a slow computer. We have also seen how reduced I/O delay time can result in faster consecutive reading speed in the improved FRTC3.EXE diagnostic program. System bus speed is indeed a factor based on computer speed, as it affects aggregate RTC access time.

Too little or no I/O delay time in between discrete consecutive RTC accesses in the code would mean the I/O delay time is mismatched with the computer speed with respect to amount of delay required for consecutive RTC accesses. Delay time for a 286 computer is too small a delay in a 486 computer due to greater computer speed. No delay time invites trouble – this may cause I/O overruns in the RTC address bus. This may result in data retrieval from memory location other than that intended.

BIOS chips are usually matched with not just the motherboard CPU chip, but also the chipset and RTC. Ostensibly, neither backward nor forward compatibility technically exists for any version of BIOS chips in terms of CPU chip types. Hence, a mismatched BIOS chip would almost certainly be ruled out, theoretically speaking. But it is still possible to use a BIOS chip designed for 286 CPU on another 286 CPU with faster (or slower!) speed than it is designed for. In addition, it is conceivable to transfer a section of BIOS code to the next higher version of the BIOS chip without modifying it to accommodate the increased computer speed.

For a 386 20 MHz computer, a set of two I/O delay instructions in assembler yields a microsecond of delay. This is enough delay for separating consecutive RTC accesses. On an 8 MHz I/O bus, the system can access an I/O device twice in about 750 nS (nanosecond, or 0.750 µS). This is typically too fast, and the second consecutive RTC access (the first to set the address, the second to get RTC data) may result in erroneous data.*

* On AT+ systems, the ROM chips are *not* on the I/O bus. They are on the system bus, directly connected to the CPU.

Show below is an example assembler code that includes an I/O delay instruction in between consecutive RTC accesses.

```
OUT     070h,AL         ;output address selection
JMP     SHORT $+2       ;delay
IN      AL,071h         ;input data from address
```

Perhaps some older BIOS do not contain I/O delay instructions because 286 speeds (8 MHz and 12 MHz) were considered slow enough that such delays between consecutive RTC reads were deemed unnecessary. This is quite conceivable. In fact, one TD researcher, a BIOS assembler programmer, considered it conceivable and even unnecessary to use I/O delay instructions.

After all, the decision to use only 2 bytes for the year date information way back in the early 1980's was just as conceivable. Even in 1979 an influential businessman insisted on only a *single-digit year date* in the source code! This was done in 1979, the year before the decade rollover. The result is that a software company has had to do the very remedial software work we are now doing for 4-digit year dates, converting the single-digit year date into a two-digit year date format, because the decade was rolling over into the new '80s. The insistence on saving memory space was understandable because in those days memory was very expensive (128 KB memory was priced at $10,000!). Since then, the two-digit year date format became the industrial standard, until now, when we are again paying a big price for it.*

The managers responsible for software development in those old days *assumed* that the code written in those days would become obsolete and be replaced with new code before the century turns over. Instead, they became what is called *legacy standard*, a package that was trusted because it was robust. It was used over and over again. Now this legacy standard is forced to change because this standard will not work in Y2K. Over the years since 1980 this standard was replicated in every business software package in every corner of the business world.

* At the time of the finalization of this report, Y2K remediation is now completed.

The same characteristic attributes to software development applies to the non-compliant BIOS code reading the RTC. This is what the author means by the statement "it's quite conceivable." It's very conceivable. "If it works for now, let it go and let somebody else worry about it when the time comes to change it." This is an irresponsible attitude to future consequences, especially if the code becomes the *legacy* that drives all software industry.

Of course, after the year 2000 rollover, all this "hullaboo" will soon be forgotten.† We have all of 999 years in which the number "2" in the fourth digit will remain the same. It will be up to future generations to fix the software limitations that still exists today which will not work even 50 years from now. But of course everyone *assumes* the code will change by then. Will it? *We have already learned what software legacy can do to you, to industry, and to software companies.*

Notwithstanding this analysis, it is a known fact that RTC chips have many quirks that make reliable programming difficult to "cover all the bases" for all types of computers. Special programming is required to ensure reliable RTC access outside the UIP. Author Frank van Gilluwe complains loudly in his book *The Undocumented PC* that "…reading the [RTC] clock values is a bit more tricky than changing them. Why make our lives as programmers easy?" ‡

This explains why most applications avoid direct RTC access and instead use BIOS interrupts to get/set time/date information. The variations in CPU speed, I/O delays between reads, and other factors across all platforms make it difficult for application programs to cover them all and remain robust across platforms. In truth, there is no good reason for direct RTC access when BIOS interrupts are available to serve the same purpose. It does not cost a lot in terms of processing time to get the time and date through BIOS services, unless it is done during a process that requires most of the CPU resources (for example, graphic games).

There is no reason to raise alarms about an RTC with non-buffered registers. As reason goes, there should be no problems at all if the BIOS code meets RTC specifications. This report has proven that the non-buffered RTC is not the culprit for the TD anomaly.

In part, these specifications consist of the following, as also highly recommended by *The Undocumented PC* by Frank van Gilluwe:

† At the time of the finalization of this report, it is already becoming fact.

‡ Second Edition, page 868, bottom.

- UIP (Update In Progress) bit in RTC Register A must be honored
- Interrupts must be disabled
- NMI should be disabled as a precautionary measure
- I/O delay instruction should be included in back-to-back RTC instructions

These specifications apply to both non-buffered and buffered RTC chips.

Figure 4 is a set of example routines showing how all but one of these specifications are implemented. Note that the UIP is not checked in this code. *These are example routines only.* Use at your own risk.

In conclusion, if the BIOS POST routine were made to be RTC compliant, the problem goes away. Or, if the RTC is a buffered type and the BIOS code is properly written to utilize it, no problem will ever occur.

```
; CMOSRead
;
; On call:      AL = CMOS-register to read
;
; Returns:      AH = contents of the CMOS-register read

CMOSRead PROC NEAR

    or  al, 80h         ; disable NMI
    cli                 ; disable interrupts
    out 70h, al         ; write to index port 70h
    jmp short $+2       ; short I/O delay
    in  al, 71h         ; read from data port 71h
    mov ah, al          ; store in AH
    xor al, al          ; AL = 0
    out 70h, al         ; enable NMI
    sti                 ; enable interrupts
    ret                 ; return

CMOSRead ENDP
```

```
; CMOSWrite
;
; On call:      AH = value to be written
;               AL = CMOS-register to write to
;
; Returns:      nothing

CMOSWrite PROC NEAR

    or  al, 80h         ; disable NMI
    cli                 ; disable interrupts
    out 70h, al         ; write to index port 70h
    mov al, ah          ; value in AL
    jmp short $+2       ; short I/O delay
    out 71h, al         ; write to data port 71h
    xor al, al          ; AL = 0
    jmp short $+2       ; short I/O delay
    out 70h, al         ; enable NMI
    sti                 ; enable interrupts
    ret                 ; return

CMOSWrite ENDP
```

FIGURE 4: Example code to read and write to RTC

10.0 Access Time Source Code

This section contains source code for measurement of date access time and RTC access time. Three source code listings are provided:

1. Frank van Gilluwe's TIMEVENT.ASM, courtesy of Frank van Gilluwe, the main time measurement utility
2. The author's BIOSACC.ASM, measures BIOS INT 1Ah date access time
3. The author's RTCACC.ASM, measures RTC access time of a single byte

All source code listings are from Frank van Gilluwe's book, *The Undocumented PC, 2nd Ed*. The last two (BIOSACC.ASM & RTCACC.ASM) are the author's modified code from the book.

These utilities are provided to readers for experimentation of BIOS access time and RTC access time on their own computers. They will work on any IBM-compatible computers from 286 up. They can be compiled using Borland's TASM and TLINK programs. To run the BIOSACC or RTCACC utilities:

1. First run TIMEVENT.
2. Run BIOSACC, RTCACC, or combination.
3. Run TIMEVENT again to obtain the access time. The counter is reset for the next event each time.
4. Repeat steps 2 and 3 as desired. Either BIOSACC or RTCACC (or combination) can be run before step 3.

The RTCACC utility accesses the RTC to read only a single byte of data. To obtain the aggregate access time of three bytes of data (representing the time data), run RTCACC three times in succession before running TIMEVENT. (The period of time between RTCACC executions does not matter because this time period is not counted.)

Note that the RTCACC code does not include time conversion code for converting from BCD to binary tickcounts, therefore the reported aggregate access time will be shorter than the actual aggregate access time for 3-byte time data during computer boot.

By contrast, the BIOSACC code will result in a much longer access time compared to RTCACC code because a call to the BIOS service is made, which invokes interrupt vectoring to the BIOS code that reads the RTC date (4 bytes) and returns. Any access to any hardware supported by the BIOS is always slower than direct access to that hardware. Video memory access is one example.

10.1 TIMEVENT.ASM Source Code

```
;=====================================================================
;Program displays any previous event duration, and resets
;the timer to time the next event duration.
;
;(c) Copyright 1994, 1996  Frank van Gilluwe
;All Rights Reserved.
;=====================================================================
;Formatted by Michael D. O'Connor, Year 2000, for TD Analysis Report
;No change made to actual code or comments except the following:
;  1. One comment removed at "org 100h" line
;  2. "include undocpc.inc" removed and IODELAY macro and equates
;     added in its place
;
;Courtesy of Frank van Gilluwe


cseg    segment para public
        assume  cs:cseg, ds:cseg
        org     100h

;================================
;Added by M. O'Connor,
;originally from undocpc.inc

LF      EQU     0Ah
CR      EQU     0Dh

IODELAY macro
        jmp     short $+2
        jmp     short $+2
endm
;================================

timevent        proc    far

                ;first, check if timer has overflowed
start:          in      al,61h                  ;
                test    al,20h                  ;check Timer 2 output status
                jz      timeskp1                ;jump if not overflowed
                mov     dx,offset ovrflow       ;output overflow message
                jmp     timeskp2                ;

                ;read Timer 2 counter contents
timeskp1:       mov     al,80h                  ;latch output command
                out     43h,al                  ;send command
                IODELAY                         ;
                in      al,42h                  ;get lsb of counter
                IODELAY                         ;
                mov     dl,al                   ;
```

```
                in      al,42h                  ;get msb of counter
                mov     dh,al                   ;dx = counter value
                mov     ax,0FFFFh               ;starting value
                sub     ax,dx                   ;ax = duration count

                ;multiply the duration count ax by 838 to get microseconds
                mov     bx,838                  ;assume timer in mode 3
                mul     bx                      ;dx:ax = ax * bx
                call    make_decimal            ;convert dx:ax to decimal
                mov     dx,offset event         ;duration message
timeskp2:       mov     ah,9                    ;
                int     21h                     ;display message

                ;now re-set Timer 2 mode and count for the next duration
                mov     al,0B0h                 ;Timer 2 command, mode 0
                out     43h,al                  ;send command
                IODELAY                         ;
                mov     al,0FFh                 ;counter value FFFF
                out     42h,al                  ;send lsb to counter
                IODELAY                         ;
                out     42h,al                  ;send msb to counter
                                                ;
                mov     ah,4Ch                  ;
                int     21h                     ;DOS terminate
timevent        endp


event           db      CR,LF
                db      'Last event duration = '
out_value       db      '           us   '
                db      '   Counter reset for next event.'
                db      CR,LF,'$'

ovrflow         db      CR,LF
                db      'Last event duration exceeded 54,142 us'
                db      '   Counter reset for next event.'
                db      CR,LF,'$'


;=============================================================
; MAKE_DECIMAL
;    convert a 32-bit value into a decimal number
;
;    Called with:    dx:ax = 32-bit value to convert to
;                            decimal in the form 99,999.999
;
;    Returns:        decimal number inserted at out_value
;
;    Regs used:      ax, bx, cx, dx, di
;
;    Subs called:    hex2ascii, decw

make_decimal    proc    near
                mov     cx,10000                ;
                div     cx                      ;dx:ax/cx = ax, dx remainder
                push    ax                      ;save remainder
                mov     di,offset out_value+5   ;where to put result
                mov     bl,1                    ;no justification
                mov     ax,dx                   ;get lower 10,000
                call    decw                    ;convert to decimal
                sub     di,4                    ;mov di back
```

```
                mov     cl,[di]                 ;get char
                pop     ax                      ;
                cmp     ax,0                    ;no higher value?
                jne     mkdec0                  ;jump if ax has value
                cmp     cl,' '                  ;1,000s?
                je      mkdec4                  ;jump if not (done)
                mov     byte ptr [di],'.'       ;
                dec     di                      ;
                mov     [di],cl                 ;put in 1,000's digit
                jmp     mkdec4                  ;done
mkdec0:         cmp     cl,' '                  ;space?
                jne     mkdec1                  ;jump if not (valid)
                mov     cl,'0'                  ;covert to zero
mkdec1:         mov     byte ptr [di],'.'       ;
                dec     di                      ;
                mov     [di],cl                 ;put in 1,000's digit
                add     di,2                    ;
                cmp     byte ptr [di],' '       ;space? (100's digit)
                jne     mkdec2                  ;jump if not
                mov     byte ptr [di],'0'       ;convert to zero
mkdec2:         inc     di                      ;
                cmp     byte ptr [di],' '       ;space?  (10's digit)
                jne     mkdec3                  ;jump if not
                mov     byte ptr [di],'0'       ;convert to zero
mkdec3:         mov     di,offset out_value     ;where to put balance
                mov     bl,1                    ;no justification
                call    decw                    ;convert ax to dec
                sub     di,4                    ;mov back to comma
                mov     ax,[di]                 ;get 2 char
                cmp     ah,' '                  ;no 1,000,000's?
                je      mkdec4                  ;jump if none (done)
                mov     byte ptr [di+1],','     ;
                dec     di                      ;
                mov     [di],ax                 ;put millions digits up
mkdec4:         ret                             ;
make_decimal    endp


;===============================================================
; DECW
;    Convert the hex number in ax into decimal 1 to 5 ascii
;    characters and insert into [di].  Increment di ptr.  The
;    leading zeros are suppressed.
;
;    Call with:      ax = input hex number
;                    di = pointer where to store characters
;                    bl = 0 for left justification
;                         1 for no justification
;
;    Returns:        word converted to ascii at [di]
;
;    Regs used:      bx
;
;    Subs called:    hex2ascii

decw            proc    near
                push    ax                      ;
                push    cx                      ;
                push    dx                      ;
                cmp     ax,0                    ;check for zero
                jne     decskip0                ;jump if not
```

```
                mov     al,4                    ;if justify, make ax = 0
                mul     bl                      ;no justify, ax = 4
                add     di,ax                   ;move pointer
                mov     byte ptr [di],'0'       ;put up ascii zero
                jmp     decskip15               ;done !
                                                ;
decskip0:       xor     cl,cl                   ;temp flag, 0=suppression on
                mov     ch,bl                   ;save flag (0=left justify)
                xor     dx,dx                   ;zero
                mov     bx,10000                ;
                div     bx                      ;(labelcnt)/10000
                cmp     al,0                    ;10000's?
                je      decskip1                ;jump if zero
                inc     cl                      ;no longer zero suppression
                call    hex2ascii               ;convert to ascii
                mov     byte ptr [di],bh        ;put 1000's digit in
                jmp     decskip2                ;
decskip1:       cmp     ch,0                    ;left justify?
                je      decskip3                ;jump if so
decskip2:       inc     di                      ;
decskip3:       mov     ax,dx                   ;get remainder
                xor     dx,dx                   ;zero
                mov     bx,1000                 ;
                div     bx                      ;(labelcnt)/1000
                cmp     cl,0                    ;zero suppression active?
                jne     decskip3a               ;jump if not
                cmp     al,0                    ;1000's?
                je      decskip4                ;jump if zero
decskip3a:      inc     cl                      ;no longer zero suppression
                call    hex2ascii               ;convert to ascii
                mov     byte ptr [di],bh        ;put 1000's digit in
                jmp     decskip5                ;
decskip4:       cmp     ch,0                    ;left justify?
                je      decskip6                ;jump if so
decskip5:       inc     di                      ;
decskip6:       mov     ax,dx                   ;get remainder
                xor     dx,dx                   ;zero
                mov     bx,100                  ;
                div     bx                      ;(remainder in dx)/100
                cmp     cl,0                    ;zero suppression active?
                jne     decskip7                ;jump if not
                cmp     al,0                    ;zero?
                je      decskip8                ;suppress zero
decskip7:       inc     cl                      ;no longer zero suppression
                call    hex2ascii               ;convert to ascii
                mov     byte ptr [di],bh        ;put 100's digit in
                jmp     decskip9                ;
decskip8:       cmp     ch,0                    ;left justify?
                je      decskip10               ;jump if so
decskip9:       inc     di                      ;
decskip10:      mov     ax,dx                   ;get remainder
                xor     dx,dx                   ;zero
                mov     bx,10                   ;
                div     bx                      ;(remainder in dx)/10
                cmp     cl,0                    ;zero suppression active?
                jne     decskip11               ;jump if not
                cmp     al,0                    ;zero?
                je      decskip12               ;suppress zero
decskip11:      call    hex2ascii               ;convert to ascii
                mov     byte ptr [di],bh        ;put 10's digit in
                jmp     decskip13               ;
```

```
decskip12:      cmp     ch,0                    ;left justify?
                je      decskip14               ;jump if so
decskip13:      inc     di                      ;
decskip14:      mov     al,dl                   ;get 1's digit (remainder)
                call    hex2ascii               ;convert to ascii
                mov     byte ptr [di],bh        ;put 1's digit in output
decskip15:      inc     di                      ;
                pop     dx                      ;
                pop     cx                      ;
                pop     ax                      ;
                ret                             ;
decw            endp


;=============================================================
;HEX2ASCII
;    Convert the hex number in al into two ascii characters
;
;    Called with:    al = input hex number
;
;    Returns         bx = converted digits in ascii
;
;    Regs Used:      al, bx

hex2ascii       proc    near
                mov     bl,al                   ;
                and     al,0fh                  ;
                add     al,90h                  ;
                daa                             ;
                adc     al,40h                  ;
                daa                             ;
                mov     bh,al                   ;
                mov     al,bl                   ;upper nibble
                shr     al,1                    ;
                shr     al,1                    ;
                shr     al,1                    ;
                shr     al,1                    ;
                and     al,0fh                  ;
                add     al,90h                  ;
                daa                             ;
                adc     al,40h                  ;
                daa                             ;
                mov     bl,al                   ;bx has two ascii bytes
                ret                             ;
hex2ascii       endp

cseg    ends
        end     start
```

—TIMEVENT.ASM source code courtesy of Frank van Gilluwe

10.2 BIOSACC.ASM Source Code

```
;       To use, first run TIMEVENT.  Then run BIOSACC.  Then run
;       TIMEVENT again to see the BIOS date access time.
;
;       Copyright 2000 Michael D. O'Connor
;       All Rights Reserved.
;       Thanks to Frank van Gilluwe

cseg            segment para public
                assume  cs:cseg, ds:cseg
                org     100h

start:          mov     dx,offset info          ;display message
                mov     ah,9                    ;
                int     21h                     ;

                ;to start event timing, activate the counter
                in      al,61h                  ;read current contents
                or      al,1                    ;set gate bit on
                out     61h,al                  ;activate counter

                ;code to monitor how long it takes to run starts here
                mov     ah,4                    ;get date via BIOS
                int     1Ah                     ;bang it

                ;to stop the event timing, disable the counter
                in      al,61h                  ;read current contents
                and     al,0FEh                 ;set gate bit off
                out     61h,al                  ;disable counter
                mov     ah,04Ch                 ;DOS terminate
                int     21h                     ;

;embedded data

CR      EQU     0dh
LF      EQU     0ah

info    db      'BIOSACC - Use TIMEVENT before and after'
        db      ' this utility to measure time duration',CR,LF
        db      '          of RTC date access through the BIOS.'
        db      CR,LF,CR,LF,'$'

cseg    ends
        end     start
```

10.3 RTCACC.ASM Source Code

```
;       To use, first run TIMEVENT.  Then run RTCACC.  Then run TIMEVENT
;       again to see the length of time of RTC access of a single byte.
;
;       Copyright 2000 Michael D. O'Connor
;       All Rights Reserved.
;       Thanks to Frank van Gilluwe

cseg            segment para public
                assume  cs:cseg, ds:cseg
                org     100h

CR      EQU     0dh
LF      EQU     0ah

IODELAY macro
        jmp     short $+2
        jmp     short $+2
endm

start:          mov     dx,offset info          ;display message
                mov     ah,9                    ;
                int     21h                     ;

                ;to start event timing, activate the counter
                in      al,61h                  ;read current contents
                or      al,1                    ;set gate bit on
                out     61h,al                  ;activate counter

                ;code to monitor how long it takes to run starts here
                mov     al,0ah                  ;status reg A addr
                or      al,80h                  ;disable NMI cmd
                cli                             ;disable ints
                out     70h,al                  ;send addr & cmd
                IODELAY                         ;I/O delay
                in      al,71h                  ;get reg A contents
                IODELAY                         ;I/O delay
                mov     al,0                    ;
                out     70h,al                  ;reset NMI
                IODELAY                         ;I/O delay
                sti                             ;enable ints

                ;to stop the event timing, disable the counter
                in      al,61h                  ;read current contents
                and     al,0FEh                 ;set gate bit off
                out     61h,al                  ;disable counter
                mov     ah,04Ch                 ;DOS terminate
                int     21h                     ;

;embedded data

info    db      'RTCACC - Use TIMEVENT before and after'
        db      ' this utility to measure time duration',CR,LF
        db      '         of RTC access of a single byte.'
        db      CR,LF,CR,LF,'$'

cseg    ends
        end     start
```

11.0 Web Sites

11.1 Humorous Sites

Below are some interesting web sites that will catch your eye or humor, or both.

---

http://www.cezwright.com/y2k/time_dilation.htm

Here is a mildly humorous site that is slightly off the mark in describing the "Time Dilation" problem. It even mispells the name of the anomaly as "Croch-Echlin Affect" [sic]. Here's an example statement: "If you computer comes up with random dates, then it could be Time Dilation caused by those Croch and Echlin guys" [sic].

Of course, this author makes no claim or opinion as to whether the above statement speaks the truth, or is a parody, or just bad English. It is really hard to tell where the author of this web site is coming from.

---

http://archive.lug.boulder.co.us/bymonth/1998.12/msg00171.html

An interesting thread about Time Dilation of an entirely different sort appears at the above URL. This recent thread (December 1999) details discovery of a "wall clock" at NIST "presumeably linked to the national time standard" jumping forward in time. Of course, this issue was apparently never resolved, or the resolution was never posted.

---

http://www.kcstar.com/item/pages/home.pat%2Clocal/30da8f52.b28%2C.htm

This is a Kansas City Star newspaper article (28 November 1998) about Time Dilation as a "new wrinkle" in year 2000 problems. At that time, Echlin's software fix price was one dollar per copy.

---

http://www.y2ksantacruz.org/y2kinfo/crouch-e.htm

A much longer and more detailed newspaper article (9 November 1998) appeared in the New York Times with the title "Dispute on a Wrinkle in the Year 2000 Problem." Tom Becker was one of those interviewed. Tom Becker disputes the TD anomaly as saying it does not exist, and that Echlin and Crouch are trying to use scare tactics to incite a buying frenzy for their software solution.

One interesting statement made by a manager of timekeeping devices at Dallas Semiconductor, a manufacturer of RTC chips and other chips, was that "[Crouch/Echlin's] silence turns us all off." This has been the single most consistent characteristic of the duo pair since the author's involvement in the investigation in late 1997.

As Jace Crouch put it in his rebuttal: "We have… learned to speak the language of non-denial denial and non-affirmative affirmation. It's an acquired taste, like olives." Yet that's the language Mr. Jace Crouch accused Intel of speaking. Of course, he learned the language from Intel, and by admission has adopted it.

An interesting case of "corporate dilation," this.

---

http://home.a-city.de/walter.fendt/physengl/timedil1.htm

This is a graphic demonstration of the *real* definition of Time Dilation. That is, as it relates to physics, not a system time/date instability problem.

11.2 Microsoft Y2K Solution Product Links to Web Sites

As noted in this report, a hardware fix in the form of adapter boards is recommended rather than a software fix to remediate Y2K issues as well as the Crouch-Echlin Effect. These Microsoft links have internal links pointing to the respective board manufacturer web site and respective email address:

http://www.microsoft.com/2000/tools/hardware/yiikrtc.htm
Millenium YIIK RTC from ACF Informatigue (French), NSTL-certified product

http://www.microsoft.com/2000/tools/hardware/cyber2k.htm
Cybergeddon 2000 PC Enhancement Card from Cybergeddon Europe Ltd (British), NSTL-certified product

http://www.microsoft.com/2000/tools/hardware/m2000.htm

M2000 Module from Millenium 2000 Solutions (U.S.A.), meets the only published compliance standard approved by the British Standards Institute

Note: Microsoft makes no endorsement or recommendation regarding any of these products.

11.3 Michael Kennedy's Web Site

This web site has some interesting revelations regarding the TD anomaly. Mr. Kennedy has demonstrated with software how a conversion routine (BCD to Hex) may translate bad data from the RTC. The bad data was invoked by making the CPU very busy by means of a TSR loaded in memory, and BIOS interrupt 1Ah was used in a separate program to obtain the dates during this period. The results are shown on his web site:

http://www.kennedysoftware.ie/tdjumps.htm

This author has never seen or heard of an anomaly in which the year has advanced ahead of the year date, however.

Mr. Kennedy makes some pointy remarks on this page that still remains valid today:

> *Is it interesting that many software developers can supply TD solutions, yet I've been unable to locate anybody who has identified TD exactly, or who can rigourously illustrate it, or who can show a solution actually working?:*
>
> - *How is software developed to fix a problem that is not accurately defined?*
> - *How is this software tested and approved, if the problem cannot be created/observed?*
> - *How can one be sure the "fix" works?*
>
> *Magic? Brilliance? Snake-oil?*
> *Commentary: abundant.*
> *Facts: scarce.*

However, whether or not the TD anomaly exists, the anomaly cannot occur on an RTC chip that has 100% double-buffered registers. Hence, a hardware solution exists, but a software solution may indeed be questionable. Hopefully, the facts will no longer be considered "scarce" by virtue of this exhaustive report.

11.4 Other Web Sites

Echlin/Crouch are not the only ones providing a software fix for the TD anomaly; they have at least one competitor. Some provide a hardware fix in the form of adapter cards.

---

http://www.intervations.co.uk/crouch-echlin.html

This site is different. Intervations has conducted tests to verify the Crouch-Echlin Effect, but has never been able to confirm it.

According to Andrew Ware, President of Intervations, he said in email to the author:

> *We worked with a group of companies who were part of a Y2K consortium in New Zealand. These including WANG and NEC, but they in turn were working with most of their clients. Thousands of machines were tested, and none could be found with TD.*
>
> *Again, we never found any machines with TD. We tested many 286 machines. We did however find 286 machines that run slowly, so that after a couple of days the clock needed updating. But this was not a Y2K or TD issue, it was just a fault on those machines. We also found that machines running Digital Research CDOS (a mult user DOS OS) would have slow clocks. However, this was due to the software/hardware setup to enable the multitasking OS to run correctly and had nothing to do with Y2K or TD.*

This is an unfortunate waste of time and money. Mr. Ware assumes the TD anomaly is mere hype, but is nevertheless open to any *verifiable* occurrences of the TD anomaly. He offers free testing software to anyone who thinks a TD anomaly has occurred on a computer, but would like to know the results of the test.

According to the web page, Jace's motherboard (or refurbished chassis with the motherboard inside) has disappeared from Compaq/Digital. Twice Mr. Andrews Ware offered to buy a TD computer from Mr. Mike Echlin.

Also according to the web page, Intervation had logged over 500,000 reboots on several computers and there was no sign of the TD anomaly.

Refer to the web site for more information.

Due Diligence 2000 is providing their own software fix for the same anomaly, and their web page has basically the same technical description of Time Dilation as on Echlin's technical paper. These URLs are:

http://www.dd2k.com/td_about.htm
http://www.dd2k.com/DueDiligence2000/charities/tdtech.htm
http://www.geneva2000.com/fdl3a.htm

Due Diligence 2000 TD software solution is a free download, but registration is required. They have a separate software package for each major MS operating system platform from MS-DOS to Windows 98. In addition, the software includes remediation for Y2K-related anomalies in the computer.

---

http://www.freeyellow.com/members4/link2000uk/time.htm

This is another site offering Link 2000 enabler boards with RTC chips as a permanent solution to the TD anomaly. In contrast to Due Diligence 2000, this site properly credits Crouch/Echlin by calling the problem the Crouch-Echlin Effect. This board is also a solution for Y2K-related issues.

---

http://www.micro2000.com/crouch-echlin.htm

Micro2000 sells Centurion boards with buffered RTCs, which they said in a newsgroup message will remove the TD anomaly. They had been in contact with Crouch/Echlin before. They described how their board made the TD anomaly disappear and how they confirmed it by inserting and removing the board from the effected system. Their board also resolves Y2K-related issues. Their site mentions the Crouch-Echlin Effect.

www.pcc2000.com/y2kreport_12.5-cefull.html

The above site contains a "reprint" of Mike Echlin's "A Detailed Report of The Crouch Echlin effect" in a much nicer page. It contains essentially the same text as in Mike Echlin's technical paper at his site:

http://www.intranet.ca/~mike.echlin/bestif/tdpaper.htm

Although the PCC 2000 site mentions replacing the RTC with one that is buffered as a fix; it is not a complete solution because the BIOS may also need to be replaced, depending on the RTC chip itself. PCC 2000 is another competitor offering what they call the PC FIX 2000 software.

---

http://www.nraa.asn.au/%7Eamadeus/crouch.htm

This is a PC Life 2000 site with an equally impressive page and with the same technical paper. However, this site also reveals their frustrations in finding and confirming any computer that exhibits the TD anomaly. It is asking for computers!

---

http://www.nsm.ie/ans3.htm

Another site describing a slightly different TD anomaly theory. It offers the NSM Diagnostic Fix software to check the anomaly. (NSM is an acronym for Network Systems Management.)

---

http://www.ptcinc.net/Y2K/Index.htm

This page offers a product called "Millennium Timepiece" as a solution to the TD anomaly. It costs $110 and requires an adapter key connected to a printer port on the computer. It also warns against using "software and BIOS fixes" as not a solution to the problem. The phrase "software and BIOS fixes" is rather ambiguous. Using a buffered RTC with a compatible BIOS chip (hardware, not software) is another solution, but a harder to implement unless they are mounted onto an adapter card. Micro2000 has such a board; this board (called Centurion) was emperically confirmed as an effective fix for the TD anomaly.

ACRONYMS

µs	Microsecond, or 0.000001 second, also represented as uSec
AMI	American Megatrends Inc., a manufacturer of BIOS chips
BCD	Binary Coded Decimal – a unique way of storing decimals in hex format without converting to hex
BDA	BIOS Data Area – set up in memory by BIOS on boot up
BIOS	Basic Input Output System – sets up a vector table in memory that directs software access to hardware devices in the computer
CMOS	Complementary Metal Oxide Semiconductor – in context, usually refers to the CMOS memory in the RTC; in other contexts, refers to a type of chip manufactured with this technology, designed for low-current (microamperes) applications
CPU	Central Processor Unit
DOS	Disk Operating System
EISA	Extended Industry Standard Architecture – 32-bit bus design with extensions beyond the standard AT/ISA architecture
Hz	Hertz – 1 cycle per second
I/O	Input/Output
ISA	Industry Standard Architecture – 16-bit bus design of the AT
KHz	Kilohertz, or 1,000 Hz, also represented as kHz
LSB	Least Significant Bit
MHz	Megahertz, or 1,000,000 Hz
MSB	Most Significant Bit
NMI	Non-Maskable Interrupt, traps RAM parity and math co-processor errors
NSTL	National Software Testing Laboratories
PC	Personal Computer
PCI	Peripheral Component Interconnect – an alternate card bus standard to ISA, found on most Pentiums
POST	Power On Self Test, a BIOS routine executed on computer boot up
PSP	Program Segment Prefix
RAM	Read Access Memory – memory that can be read from and written to
ROM	Read Only Memory – memory that can be read from but not written to
RTC	Real Time Clock
TD	Time Dilation, now renamed as “Time and Date Instabilities”
TSR	Terminate Stay Resident – a program that terminates itself but stays resident In RAM
UIP	Update In Progress – a bit in RTC Register A that goes high if RTC update is in progress
Vcc	This is not an acronym but a symbol referring to supply voltage for a chip or chips, usually seen in schematics
VLSI	Very Large Scale Integration – a device with several devices embedded
Y2K	Year 2000 – “K” is a symbol for “thousand”

Release Notes:

This section gives final notes at the time of the release of this document.

1. The web site http://www.nethawk.com/~jcrouch/response.htm as listed in the endnotes is no longer available.

2. The web site below is an excellent source of most links related to TD.

   http://www.elmbronze.demon.co.uk/year2000/dilation/index.htm

[1] Source: http://www.nethawk.com/~jcrouch/second.htm
"Mike Echlin's Theory of How TD Occurs on Personal Computers"

[2] Source: http://www.intel.com/support/year2000/lbza3b1.htm
See Intel's report, Analysis of the Crouch-Echlin Effect, Appendix B, page 13

[3] Ibid., second paragraph, last sentence

[4] Ibid., last paragraph.
"…the utility is reporting discrepancies between RTC data and BIOS data, *as on* Mr. Echlin's Zenith."

[5] Ibid.

[6] See http://www.nethawk.com/~jcrouch/second.htm
"Mike Echlin's Theory of How TD Occurs on Personal Computers"

[7] Source: http://www.intel.com/support/year2000/lbza3b1.htm
Analysis of the Crouch-Echlin Effect, page 9, Hypothesis #1

[8] Source: http://www.intel.com/support/year2000/lbza3b1.htm
Intel's report, Analysis of the Crouch-Echlin Effect, page 6

[9] Ibid.

[10] Ibid., page 8, under Compaq/Digital section

[11] Source: http://www.nethawk.com/~jcrouch/response.htm
"TD, Floating Time and Date Errors, and Corporate Dilation"
2nd paragraph
"…I [Jace Crouch] shipped my original motherboard to Digital…"

See also http://www.elmbronze.demon.co.uk/year2000/dilation/uty2kU01.htm
Newsgroup message dated 17 August 1998, near bottom of page
"…my [Jace Crouch's] original 286 motherboard is now…at DEC."

[12] Ibid.

[13] Ibid.

[14] Source: http://www.intel.com/support/year2000/lbza3b1.htm
Analysis of the Crouch-Echlin Effect, page 6

[15] Source: http://www.nethawk.com/~jcrouch/response.htm
"TD, Floating Time and Date Errors, and Corporate Dilation"
1st paragraph
"They [Intel] claim that Jace's original motherboard (which they never examined) has an Award BIOS, whereas it has a Chips & Technology BIOS, as is clearly indicated on Jace's web pages and on Mike Echlin's web pages." [underline original]

[16] Source: http://www.nethawk.com/~jcrouch/td01.txt

[17] Source: http://www.intel.com/support/year2000/lbza3b1.htm

Analysis of the Crouch-Echlin Effect, page 14

[18] Ibid., page 14, Appendix C, see table header

[19] Ibid., page 7, under Echlin System Test Methodology, 3rd paragraph

[20] Ibid., page 7, under Power Cycle Testing

[21] Ibid.

[22] Ibid., page 15, Appendix C, under Echlin's Zenith 286 Turbo section, last paragraph

[23] Ibid., page 13, Appendix B, last paragraph

[24] Ibid., page 11, second paragraph in Appendix A

[25] Ibid., page 15, Appendix C, under Echlin's Zenith 286 Turbo section, last paragraph

[26] Source: http://www.intel.com/support/year2000/lbza3b1.htm
Analysis of the Crouch-Echlin Effect, page 16~17

[27] Ibid., page 7, under Power Cycle Testing, last sentence

[28] Source: http://www.intel.com/support/year2000/lbza3b1.htm
Analysis of the Crouch-Echlin Effect, page 7, under Echlin System Test Methodology, 1st paragraph

[29] Source: http://www.nethawk.com/~jcrouch/response.htm
"TD, Floating Time and Date Errors, and Corporate Dilation"
1st paragraph
"Mike sent them two of his computer systems [Zenith 286 boards]: one that had exhibited TD very frequently, and one that may have exhibited TD on one occasion only; guess which one [Intel] reported on?"

[30] Source: http://www.intel.com/support/year2000/lbza3b1.htm
page 15, Echlin's Zenith 286 Turbo section, last paragraph.

[31] Ibid., page 11, second paragraph in Appendix A

[32] Source: http://www.intel.com/support/year2000/lbza3b1.htm
page 15, Manual Cycling section. See also the itemized configuration and test method.

[33] Source: http://www.intel.com/support/year2000/lbza3b1.htm
Intel's report, Analysis of the Crouch-Echlin Effect, page 9, Hypothesis #2, under Findings section

[34] Ibid., page 11

[35] Ibid., page 11, end of third paragraph of Appendix A

[36] Ibid., page 18, last paragraph

[37] Ibid., page 11, second paragraph in Appendix A

[38] Ibid., page 14, see table

[39] Ibid., page 15, under Manual Cycling section

[40] Ibid., page 19, next to last paragraph

[41] Source: http://www.techweb.com/se/directlink.cgi?EBN19990712S0016

[42] Source: http://www.zdnet.com/pcweek/stories/news/0,4153,1015364,00.html

[43] Source: http://www.intranet.ca/~mike.echlin/bestif/tdpaper.htm
(Mike Echlin's "The Crouch Echlin effect, detailed" page)
In "4. Current state of research" section, under "The Famous 244."

[44] Source: http://www.intel.com/support/year2000/lbza3b1.htm
Analysis of the Crouch-Echlin Effect, page 15, item 3 on top of page

[45] Source: http://www.nethawk.com/~jcrouch/second.htm
"Mike Echlin's Theory of How TD Occurs on Personal Computers"
See Figure 3

[46] Source: http://www.intranet.ca/~mike.echlin/bestif/tdpaper.htm
(Mike Echlin's "The Crouch Echlin effect, detailed" page)
In "General characteristics of the Bug. (The cause of the effect.)"
under "BIOS code, how it could be reading the RTC the wrong way."

[47] Source: http://www.intel.com/support/year2000/lbza3b1.htm
Intel's report, Analysis of the Crouch-Echlin Effect, page 8~9

[48] Ibid., Appendix C, page 16

[49] Source: http://www.nethawk.com/~jcrouch/response.htm
"TD, Floating Time and Date Errors, and Corporate Dilation"
1st paragraph
"Mike {Echlin] told Intel to examine one section of the POST code, and Intel examined a *different* section of the BIOS code." [Italics original]

[50] Source: http://www.intel.com/support/year2000/lbza3b1.htm
Intel's report, Analysis of the Crouch-Echlin Effect, page 8~9, Hypothesis #1
No mention of which CPU board was examined

[51] Source: http://www.intranet.ca/~mike.echlin/bestif/tdpaper.htm
(Mike Echlin's "The Crouch Echlin effect, detailed" page)
In "4. Current state of research" section, under "The Famous 244."

[52] Source: http://www.intranet.ca/~mike.echlin/bestif/tdpro.htm
(Mike Echlin's TD Research Page)
Under "Other Research" section

[53] See note 51

[54] Source: http://www.intranet.ca/~mike.echlin/bestif/tdpaper.htm
(Mike Echlin's "The Crouch Echlin effect, detailed" page)
In "4. Current state of research" section, under "The Famous 244," see first RTC listing

[55] Ibid., see second RTC listing after the first one

[56] Source: http://www.nethawk.com/~jcrouch/second.htm
"Mike Echlin's Theory of How TD Occurs on Personal Computers"

[57] Ibid., In "Example 2" underneath Figure 3

[58] Source: http://www.intel.com/support/year2000/lbza3b1.htm
Intel's report, Analysis of the Crouch-Echlin Effect, page 15, very top paragraph

[59] Source: http://www.nethawk.com/~jcrouch/second.htm
"Mike Echlin's Theory of How TD Occurs on Personal Computers"
In "Example 1" underneath Figure 3
Taken literally, Echlin is saying the time/date calculations *themselves* take longer than 244 microseconds!

[60] Source: http://www.intranet.ca/~mike.echlin/bestif/tdpaper.htm
(Mike Echlin's "The Crouch Echlin effect, detailed" page)
In "4. Current state of research" section, under "The Famous 244," see first RTC listing

[61] Ibid., just before the first RTC listing

[62] Ibid., page 9, Hypothesis #1 section, see note

[63] Ibid., page 9, Hypothesis #1 section, 2nd paragraph under Findings section

[64] Ibid., page 15, item 3 at very top of page

[65] Source: http://www.nethawk.com/~jcrouch/response.htm
"TD, Floating Time and Date Errors, and Corporate Dilation"
7th paragraph, "TD is rare folks, and it occurs mostly on older systems. I have always said that."

[66] Source: http://www.merlyn.demon.co.uk/time-dil.htm
"Time Dilation on the PC"
Under Section "Non-Causes"

[67] Ibid., Under section "CMOS Corruption"

[68] Source: http://www.nethawk.com/~jcrouch/response.htm
"TD, Floating Time and Date Errors, and Corporate Dilation"
2nd paragraph, "(*not* the CMOS problems)" – the COM port configurations are stored in CMOS memory

[69] Source: http://www.nethawk.com/~jcrouch/second.htm
"Mike Echlin's Theory of How TD Occurs on Personal Computers"
In "Example 2" underneath Figure 3
"At first we thought that these CMOS corruptions were an aspect of TD, but it is turning out that CMOS corruption…can occur on systems that do not exhibit TD…"

[70] Source: http://www.merlyn.demon.co.uk/time-dil.htm
"Time Dilation on the PC"
Under section "Bad Battery?"

[71] Ibid., Under section "PI timing?"

[72] Ibid.

[73] Ibid., Under section “Bus Fights?”, 2nd paragraph

[74] Ibid., 4th paragraph

[75] Ibid., 5th paragraph

[76] Source: http://www.merlyn.demon.co.uk/test2000.htm
“Year 2000 Testing Page”
Under section “CMOS Latency”, last paragraph

[77] Ibid., Under section “Unexpected Getting & Setting by Windows Start-Up”

[78] Ibid., Under section “Century Error”

[79] Source: http://www.merlyn.demon.co.uk/time-dil.htm
“Time Dilation on the PC”
Under section “Viruses”

[80] Source: http://www.merlyn.demon.co.uk/test2000.htm
“Year 2000 Testing Page”
Under section “Double-Setting by MSDOS”

[81] Source: http://www.nethawk.com/~jcrouch/second.htm
“Mike Echlin's Theory of How TD Occurs on Personal Computers"
5th paragraph, last sentence
“…incorrect time or date may get written back to the RTC/CMOS…”

[82] Source: http://www.compaq.com/year2000/year2000-qa.html
See Question 20

[83] Source: http://www.nethawk.com/~jcrouch/response.htm
"TD, Floating Time and Date Errors, and Corporate Dilation"
5th paragraph
“Those technicians at Digital *did* duplicate the TD time and date instabilities. They said so publicly and repeatedly. They did so for months. They did so until they were throttled by Behemoth [Computer Corporation].” [Italics original]

[84] Ibid.

[85] Source: http://www.nethawk.com/~jcrouch/response.htm
"TD, Floating Time and Date Errors, and Corporate Dilation"
Under last section, “Short recap on TD, Floating Time and Date Errors, and Corporate Dilation”

[86] Ibid., 7th paragraph
“TD is rare folks, and it occurs mostly on older systems.”