EXHIBIT I

ON DISK

SECOND EDITION

UNDOCUMENTED DOS

A PROGRAMMER'S GUIDE TO RESERVED MS-DOS FUNCTIONS AND DATA STRUCTURES

EXPANDED TO INCLUDE MS-DOS 6, NOVELL DOS, AND WINDOWS 3.1

ANDREW SCHULMAN ▪ RALF BROWN ▪ DAVID MAXEY
RAYMOND J. MICHELS ▪ JIM KYLE

CHAI

Library of Congress Cataloging-in-Publication Data

Undocumented DOS: a programmer's guide to reserved MS-DOS functions
and data structures / by Andrew Schulman . . . [et. al.]. -- 2nd ed.
p. cm. -- (Andrew Schulman programming series)
Includes index.
ISBN 0-201-63287-X
1. Operating systems (Computers) 2. MS-DOS (Computer file)
I. Schulman, Andrew. II. Series: Schulman, Andrew. Andrew Schulman
programming series.
QA76.76.063U53 1993
005.4'46--dc20

93-29037
CIP

Production Editor: Andrew Williams
Set in 10 point Galliard by Benchmark Productions, Inc.

1 2 3 4 5 6 7 8 9-BAH-9796959493
First Printing, November 1993

Addison-Wesley books are available for bulk purchases by corporations, institutions, and other organizations. For more information please contact the Corporate, Government, and Special Sales Department at (617) 944-3700 x2915.

CHA
Doci

mented Windows controversy notes that "MS-DOS has a number of undocumented APIs that are well-known, well-understood and used by ISVs" (independent software vendors). Well okay, but then why not document them?

DOS Documented

We'll address the subject of *why* Microsoft leaves crucial pieces of DOS functionality undocumented in a minute, but it's important to note first that the company has, to its credit, finally documented some of the most well-known and well-understood, previously undocumented functions.

In the summer of 1991, Microsoft Press issued the *Microsoft MS-DOS Programmer's Reference*, which includes the new INT 21h and INT 2Fh functions in MS-DOS 5.0. This book came out after the publication of the first edition of *Undocumented DOS*, and the Microsoft Press advertising headline for the programmer's reference was "DOS Documented." Microsoft released a nearly-identical version for MS-DOS 6.0 in the summer of 1993.

Indeed, the programmer's reference did document some previously undocumented calls. Most notably, Microsoft blessed the following previously-undocumented INT 21h functions:

- Function 1Fh (Get Default DPB)
- Function 32h (Get DPB)
- Function 34h (Get InDOS Flag Address)
- Function 4B01h (Load Program)
- Function 50h (Set PSP Address)
- Function 51h (Get PSP Address)
- Function 5D0Ah (Set Extended Error)

The new documentation for these old functions is, unfortunately, far from complete. One gets the sense that, to some extent, the point of this exercise for Microsoft was simply to claim that these functions are now documented. The new documentation for the previously undocumented INT 21h functions has the following problems:

- Functions 1Fh, 32h: The programmer's reference claims that these functions are only for DOS 5.0 and higher. In fact, these functions are present all the way back to DOS 2.0. This is an important piece of information, because a large number of DOS 3.x installations are still in use, and most PC disk utilities, which rely on function 32h, must be able to run on these machines. The DPB structure provided is only accurate for DOS 5.0 and higher. Basically, if you believe Microsoft's documentation, then disk utilities can only for written for DOS 5.0 and higher, which we know not to be the case.
- Function 34h: The documentation doesn't mention the critical-error flag located in the byte before the InDOS flag. As we saw earlier, the Windows DOSMGR relies on being able to decrement the return value from function 34h to get both the InDOS flag and critical-error flag into a single word.
- Function 4B01h: The documentation for the LOAD structure incorrectly reverses the order of the ldCIP and ldSSSP fields. Some errors are, of course, unavoidable, but the identical error had appeared earlier, in the book *Developing Applications Using DOS* by three IBMers, Ken Christopher, Barry Feigenbaum, and Shon Saliga (1990). Presumably Microsoft used this book as (uncredited) source material for the MS-DOS 5.0 programmer's reference. (The MS-DOS 6.0 programmer's reference does correct this error.) For a full explanation of function 4B01h, see Tim Paterson's "The MS-DOS Debugger Interface" in the first edition of *Undocumented DOS*.
- Function 5D0Ah: This function is documented only for DOS 4.0 and higher. In fact, it is present in 3.1 and higher. Furthermore, because the manual does not mention the other AH=5Dh

Novell DOS makes sense. In fact, we'll see that Novell NetWare makes so many changes to the INT 21h interface that, even though it seems to run "on top of" DOS, it effectively *replaces* DOS, and thereby fully qualifies as an "other DOS."

IBM OS/2 2.x and Windows NT most definitely do not hook or in any other way sit on top of DOS; they completely replace it. But market reality dictates that these environments run DOS programs out of the box. This ability is called ***binary compatibility***, in contrast to the much simpler goal of source compatibility. In fact, users may for some time employ such environments primarily for the purpose of running old DOS or Windows software. This requires that the new operating system either emulate DOS or run a copy of genuine DOS within a virtual machine. Either way, an important question is whether DOS programs that access undocumented DOS data structures or call undocumented DOS functions will run in these environments. Undocumented DOS is an excellent test of DOS compatibility. Achieving compatibility with documented interfaces is fairly easy, so when you hear discussions of "DOS compatibility" or "Windows compatibility," it's really support for the undocumented interfaces that's being discussed.

An interesting point emerges from all this. It is often *in Microsoft's interest* for major applications (not necessarily or even primarily its own) to use undocumented DOS features, as this ties these applications to Microsoft's own versions of DOS (note especially that IBM's license to the DOS source code runs out in September 1993). After all, what is really the difference between Microsoft's DOS and anyone else's, except that Microsoft has guaranteed better support for the funky undocumented things that DOS applications do? For Microsoft, undocumented DOS is thus an interesting form of product differentiation.

From CP/M to DR DOS to Novell DOS

The funny thing is, MS-DOS itself started out as a clone of the CP/M operating system from DRI. The story has been told many times of how Tim Paterson (a coauthor, incidentally, of the first edition of this book), now at Microsoft but in 1980 an engineer at Seattle Computer Products, in two months wrote Quick and Dirty DOS (QDOS), how this became 86-DOS, to which Microsoft purchased non-exclusive rights, and how this became MS-DOS 1.0, for the then-new IBM PC. Stephen Manes and Paul Andrews' history of Microsoft, Gates, has all the details, even a photograph of the original Seattle Computer order form for Microsoft's purchase of 86-DOS sales rights. Price? $50,000.

Quick, dirty, and cheap. As Andrew Tanenbaum notes in his superb textbook on *Modern Operating Systems*, "If anyone had realized that within 10 years this tiny system that was picked up almost by accident was going to be controlling 50 million computers, considerably more thought might have gone into it."

Somewhat understandably, Digital Research was upset when it found that Microsoft's new operating system for the IBM PC was a clone of CP/M. Apparently Digital's Gary Kindall even considered suing IBM over the similarity of MS-DOS to CP/M. Microsoft would be similarly upset today if someone came out with a graphical environment that happened to provide the same API as Windows.

There is no question about MS-DOS's large-scale borrowing from CP/M. As Tim Paterson would write somewhat later in "An Inside Look at MS-DOS" (*Byte*, June 1983), "The primary design requirement of MS-DOS was CP/M-80 translation compatibility."

An early article by David Cortesti ("CP/M-86 vs. MSDOS: A Technical Comparison," *Dr. Dobb's Journal*, July 1982) compared MS-DOS with both CP/M-80 (for the Intel 8080) and CP/M-86 (for the 8086), showing not only where MS-DOS properly emulated CP/M functions, but also where there were differences. For example, function 9 outputs strings terminated with a '$' character in both systems, but CP/M expanded tabs and MS-DOS didn't. In any case, the '$' itself is a reminder of MS-DOS's CP/M roots. To this day, MS-DOS contains many holdovers from its early start as a CP/M clone. The PSP, for example, is nothing more than a CP/M base page.

However, even in the beginning there were crucial differences between the two systems. MS-DO did not, as is widely claimed, mimic every last CP/M function call. For example, MS-DOS did n implement CP/M function 12 (0Ch) to get the system version number. Somewhat unaccountab MS-DOS instead used (and still uses) function 0Ch to read the keyboard.

The crucial difference was in the file system. In an important departure from the CP/M file syst MS-DOS internally used a file allocation table (FAT), a scheme borrowed from Microsoft stand-alo BASIC. Paterson's original goal was to make the FAT memory resident at all times, eliminating th multiple disk reads that CP/M often made just to find where a file's data was located. As Ray Dunca noted at the time in his "16-Bit Software Toolbox" column (*Dr. Dobb's Journal*, November 1982 "although the systems appear very similar to the casual user, they use drastically different allocatio schemes to manage disk files. This has surprisingly large effects on the speed of disk operations."

So MS-DOS began life as an enhanced clone of CP/M. Digital Research, makers of CP/M, we on to build many other operating systems, such as Concurrent CP/M, FlexOS, GEM, Concurre DOS, Multiuser DOS, and finally DR DOS. DR DOS was intended as a complete replacement f MS-DOS. The cloners were now being cloned by the original clonees!

DR DOS also shows its CP/M roots. For one thing, the DR DOS disks and manuals carry copy right dates going back to 1976! Some of the code in DR DOS may even go back to the origin CP/M code base. And while you won't see old CP/M terms such as PIP or CCP in the DR DO manuals, and while the DR DOS debugger unfortunately is not called DDT, the DR DOS kernel still called BDOS, just as in olden days.

For example, the HIDOS.SYS device driver has a /BDOS option to relocate the DR DOS kernel t the HMA or to a UMB. Interestingly, this option is more flexible than MS-DOS 5.0's DOS=HIG command and appeared much earlier. While providing more features than Microsoft, sooner tha Microsoft, DR DOS still clings a bit to its CP/M heritage, at least in its naming conventions.

None of this would matter very much, except for the fact that Novell acquired Digital Researc (for $80 million) in July 1991. DR DOS, now renamed Novell DOS, may share in some of the succe of Novell's NetWare. Netware dominates the network operating system market, far surpassing any o Microsoft's so far feeble attempts at providing network software. Novell's purchase of DR DOS is no widely regarded as a mistake, since DR DOS sales have dropped dramatically since the Novell pu chase. However, this drop is likely due not to Novell's purchase of DR, but to Microsoft's release o MS-DOS 5.0. DR DOS had its brief moment in the sun in late 1990 and early 1991, when Digi had DR DOS 5 and all Microsoft had was the terrible MS-DOS version 4, engineered largely by IBM

DR DOS is the only major competition for MS-DOS, which says a lot about Microsoft's role sin the DR DOS share is quite small. DR claimed to have sold more than 1,500,000 copies of DR DO 5.0 in 1991, admitting that this success was largely based on the clear inferiority of MS-DOS 4.0 Novell believes that DR DOS now holds 8-11% of the DOS market; *PC Magazine* (April 27, 199 says the DR DOS share is 5%. In 1991, when Microsoft sold $617.5 million worth of MS-DOS DRI's total revenues were $45.5 million. Assuming that DRI sold nothing but DOS, and assumin roughly equal prices for DR DOS and MS-DOS, this would give DR DOS about 7% of the DOS ma ket. DR DOS's presence is stronger in Europe than in the United States, possibly in part because D DOS comes out of Novell's European Development Centre in Hungerford, England.

DR DOS doesn't at first give the impression of being very important, and more than one reviewer o this book asked why we were bothering to talk about it in the first place. Angrily rejecting DR DOS claims to MS-DOS compatibility, one reviewer (no, not a Microsoft employee) dismissed the idea of an one "actually checking for the presence of this rather imperfect clone" and bluntly told us, "I see no reaso why journalists should cooperate with DR's desire to have programmers share their development and ma keting costs." This reflects a general feeling that DR DOS is Brand X and pretty much irrelevant. Fo example, in an article on operating system choices for the 1990s (*PC Magazine*, January 15, 1991 Charles Petzold placed DR DOS in the "Interesting-But-Does-It-Really-Matter Department."

```
        if (dpb = get_dpb(i))    // checks for removeable, critical error
            display(dpb);
    return 0;
}
```

This program brings up an important reason to use INT 21h function 32h instead of walking the DPB linked list. For removable media, function 32h goes to the disk and, therefore, picks up the most current information. Walking the linked list merely gets whatever (possibly stale) DPB happens to be in memory. If you access a 360K floppy disk in drive A:, remove it, put in a 1.2 megabyte floppy without accessing it, and then walk the DPB linked list, you get the DPB for the 360K floppy. Function 32h would not make this mistake.

On the other hand, function 32h hits the disk (see the disassembly of functions 32h and 1Fh in Chapter 6), which makes it inconvenient to get the DPB of drives with removable media. Further, you want to avoid reading both drives A: and B: in a system where these logical drives are mapped to the same physical floppy drive.

The version of DPBTEST in Listing 8-7 differs substantially from that in the first edition of *Undocumented DOS*, which feebly tried to deal with the above problem by checking for drives A: and B: neglecting the fact that other drive letters (such as those created with DRIVER.SYS) may involve removable media. This utter bogosity was further compounded by checking the floppy disk logical drive indicator at address 504h. Totally hopeless!

Instead, programs should use generic IOCTL calls to determine whether a device uses removable media (INT 21h AX=4408h) and to get the logical drive map (INT 21h AX=440Eh). These calls, along with an INT 24h critical error handler invoked when there is no media in the drive at all, are incorporated into the get_dpb() function that DPBTEST.C uses from DISKSTUF.C (Listing 8-4).

One last note about DPBs. Many crucial DOS disk utilities were thrown into temporary confusion by the introduction of DOS 4.0 because of a one-byte change to the DPB structure. The sectors-per-FAT field at offset 0Fh (see the appendix) grew from a *byte* to a *word*, so all subsequent fields were bumped one byte as well. As noted at the time (Ted Mirecki, "Function 32h in DOS," *PC Tech Journal*, February 1989), this one-byte modification produced a major ripple effect in the Norton Utilities and other programs that relied on this undocumented DOS data structure. Rather than bemoaning incompatibilities, cynics may view this sort of change as a good excuse to hit up customers with an upgrade release.

Buffers and Disk Caches

Like any proper operating system, MS-DOS has *buffered* I/O. Rather than directly reading sectors off the disk, DOS first checks to see whether the sector is already present in an in-memory buffer. DOS uses buffers for FAT, directory, and data sectors. DOS buffers sectors, not higher-level clusters or lower-level tracks. The BUFFERS= statement in CONFIG.SYS controls the number of sector buffers, which are chained together in a least-recently-used (LRU) circular linked list; SysVars holds a pointer to the head of this list.

Each sector-sized buffer follows a small header which identifies the drive currently using that buffer, the sector of the data it contains, a status byte indicating what type of sector (FAT, directory, or data) it contains, and a pointer to the next buffer header in the chain.

The buffer chain made its debut in DOS 2.0. In DOS 1.x, there was a single sector buffer; Tim Paterson admitted this was "a design inadequacy that is difficult to defend." On the other hand, while DOS 1.x kept the FAT memory-resident at all times, in marked contrast to CP/M which could require multiple disk reads just to find the location of a user's data, DOS 2.0 and higher rely entirely on the buffers for keeping often-used FAT sectors in memory. Paterson notes ("An Inside Look at MS-DOS," *Byte*, June 1983):

> The new MS-DOS does not keep the file allocation tables in memory at all times. Instead the tables share the use of the sector buffers. . . . This means that at any one time, all, part, or none of a FAT may be in memory. The buffer-handling algorithms will presumably keep often-used sectors in memory, and this applies to individual sectors of the FAT as well. This change in the DOS goes completely against my original design principles. . . . Now we're back to doing disk reads just to find out where the data is.

With today's large media, a memory-resident FAT could occupy up to 128K (64K clusters * 16 bits) of memory.

DOS uses the buffers in sequence, changing the linkages as necessary to maintain the most recently used buffers near the front of the chain. Any DOS sector access first walks through the chain of headers, looking for the requested drive and sector; if found, it can use the buffer contents without having to hit the disk. Moving each used buffer up to the front of the chain guarantees that any time a search reaches the end of the chain without finding its sector, the buffer at the end would be the least recently used and, thus, the proper one (in this scheme) to replace with the new data read from the disk.

Unfortunately, this simple approach did not take into account the pattern by which DOS performs disk reads. In practice, the buffer chain filled rapidly with FAT and directory data, leaving only a few buffers for all file data transfers. The system was modified several times under DOS 2.0 and 3.0, but performance problems remained significant. The DOS buffers underwent a major implementation change in DOS 4.0, when IBM introduced a complicated hashing scheme and a mechanism for keeping buffers in expanded memory. This was thrown out in DOS 5.0, which returned to the simpler LRU buffers scheme.

If DOS=HIGH, DOS 5.0 and higher keep the buffers in the HMA. Also for the first time in DOS 5.0, sectors accessed with the INT 25h and INT 26h absolute disk read and write functions first check the sector buffers.

In addition to a pointer to the head of the buffers chain, SysVars in DOS 4.0 and higher also contains the *number* of buffers, that is, the value from the BUFFERS= statement in CONFIG.SYS. (For more information, see "SysVars, or The List of Lists" later in this chapter.) Determining BUFFERS= is probably the only practical way a program could use buffers information. It is difficult when running a program to determine which CONFIG.SYS file was used to boot the system. In fact, prior to the availability of INT 21h function 3305h in DOS 4.0, one couldn't even tell what drive the system was booted from! So install, setup, and configuration programs might want to determine the value of BUFFERS= (and FILES=, which we'll look at later).

To find the value of BUFFERS= in earlier versions of DOS, a program must walk the buffers chain, as shown in BUFFERS.C (Listing 8-8). In DOS 4.0 and higher, this program also prints out a description of each buffer's contents. For example:

```
C:\UNDOC2\CHAP8>buffers
FFFF:A8C8 — C: #208 -- DIR
FFFF:B968 — C: #3133 -- DATA
FFFF:A4A0 — C: #1 -- FAT
FFFF:9A3C — C: #3226 -- DATA
FFFF:B540 — D: #109 -- FAT
FFFF:BD90 — D: #110 -- FAT
; ... etc. ...
FFFF:B118 — A: #19 -- DIR
FFFF:9400 — A: #20 — DIR
BUFFERS=30
```

As you can see, the buffers include floppy diskettes as well as hard disks; however, network redirected drives are not included. It is instructive to run BUFFERS, then perform a disk operation, such as DIR, or run some other program, and then run BUFFERS again. You can see the contents of the buffers change. Of course, running BUFFERS.EXE itself changes the contents of the buffers!

Steven Manes and Paul Andrews, *Gates: How Microsoft's Mogul Reinvented an Industry—and Made Himself the Richest Man in America*, New York: Doubleday, 1993, 534 pp. This is not really the story of Gates's life (who cares?), but of Microsoft's (now, *that's* interesting!). Meticulously-researched, with every fact or quotation backed up by at least one footnote, this book covers everything from Microsoft's purchase of QDOS, to its OEM pricing of DOS, to how Murray Sargent and Dave Weise moved Windows to protected mode.

Steven J. Mastrianni, *Writing OS/2 2.0 Device Drivers in C*, New York: Van Nostrand Reinhold, 1992, 407 pp. Chapter 9 discusses OS/2 virtual device drivers (VDDs) and the Virtual DOS Machine (VDM). A second edition, for OS/2 2.1, should be available.

Michael P. Maurice, "The PIF File Format, or, Topview (sort of) Lives!," *Dr. Dobb's Journal*, July 1993. Program Information Files (PIFs) are what Windows uses to run "old" (DOS) programs.

Michael Mefford, "Choose CONFIG.SYS Options at Boot," *PC Magazine*, November 29, 1988. Discusses the undocumented DOS CONFIG.SYS buffer.

Michael Mefford, "Running Programs Painlessly," *PC Magazine*, February 16, 1988. Discusses the problems with using INT 2Eh.

Philippe Mercier, *La maîtrise des programmes résidents sous MS-DOS*, Marabout, 1990, 410 pp. A handy book on TSRs from Belgium. Did you know that the French for "hotkey" is "touche magique"?

Microprocessor Report, *Understanding x86 Microprocessors*, Emeryville CA: Ziff-Davis Press, 1993. A collection of articles on the 286, 386, 486, and Pentium, from Michael Slater's brilliant newsletter, *Microprocessor Report* ("The Insider's Guide to Microprocessor Hardware"). Includes discussions of undocumented processor instructions, and an entire section on legal issues (Intel v. Cyrix, etc.). Don't miss the brilliant articles by John Wharton, such as "Gonzo Marketing." Why can't all technical writing be like this?

Microsoft, "API to Identify MS-DOS Instance Data," undated internal document. Describes the DOSMGR callout API.

Microsoft, *Device Driver Adaptation Guide* (DDAG), version 3.1, Redmond WA, 1992. Included with the Windows 3.1 Device Driver Kit (DDK), this includes an essential appendix on "Windows Interrupt 2Fh Services and Notifications." Why they've put this important stuff in an obscure manual like this is beyond me. There's also a useful chapter on Windows network drivers.

Microsoft Developer Network (MSDN) CD-ROM. A must-have for any serious DOS or Windows developer, the MSDN CD-ROM includes huge amounts of information that programmers often mistakenly think is undocumented. For more information, call (800) 759-5474 or (206) 936-8661. Here's a very small sampling of the articles related to DOS: "Determining Windows Version, Mode from MS-DOS App," "Demand Paging MS-DOS Applications," "Global TSR Pop-ups Incompatible with Windows," "Full-Screen DOS Apps Slow Timer Messages in Enhanced Mode," "Do Not Use the MS-DOS APPEND Utility in Windows," "Calling a DLL Written for Windows from a TSR for MS-DOS," "Binding a TSR to a VxD," "Using the Interrupt 2Fh Critical Section Services," "How a TSR Can Serialize Access to Its Data," "IOCtl Calls in Protected-Mode Microsoft Windows," "Access to the Windows Clipboard by DOS Applications," "Windows 3.1 Standard Mode and the VCPI," "How Microsoft Windows Uses an MS-DOS Mouse Driver," "How to Start a Windows Application Directly from DOS," "Passing File Handles from a TSR to a Windows Application." This partial list of titles should make clear that (1) Microsoft documentation isn't so bad after all; and (2) even die-hard DOS programmers can't ignore Windows.

"Microsoft Statement on the Subject of Undocumented APIs," August 31, 1992. Microsoft's news release on undocumented Windows: "There are undocumented APIs in every major operating system, and applications developers routinely make use of them." At the same time, Microsoft issued

Barry Nance, *Network Programming in C*, Que, 1990. Includes a chapter on "Novell's Extended DOS Services."

Tomas Nelson, "Self-Loading Device Drivers for DOS," *Windows/DOS Developer's Journal*, May 1993, pp. 27-43. Another approach to loading device drivers from the command line.

Raymond T. Nimmer, *The Law of Computer Technology*, Boston MA: Warren, Gorham & Lamont, 1985. See also 1991 Cumulative Supplement No. 1. Covers reverse engineering, antitrust, "integrated systems innovation," tying arrangements, documentation obligations, mass-market contracts, warranties, and more.

Daniel Norton, *Writing Windows Device Drivers*, Reading MA: Addison-Wesley, 1992, 434 pp. If you just want a general idea of the services that the Windows Virtual Machine Manager (VMM) provides to virtual device drivers (VxDs), and don't want to buy the Device Driver Kit (DDK), this is AIA (an inexpensive alternative).

Novell, *A Brief Description of the NetWare DOS Requester*, February 1993. Explains the limitations of the NETX INT 21h hook, and describes the new DOS redirector-style "requester" in NetWare 4.x.

Novell, *DR DOS 6.0 Optimization and Configuration Tips*, September 1991 (earlier published by Digital Research). Discusses "DR DOS 6.0 Version Numbers."

Novell, DR DOS *System and Programmer's Guide* (DR product #1182-2013-001).

Novell, *NetWare System Interface Technical Overview*.

Thomas Olsen, "Making Windows and DOS Programs Talk," *Windows/DOS Developer's Journal*, May 1992. Presents several approaches to DOS/Windows communication, including the 2F/17 clipboard API and a pipe interface VxD.

Walter Oney, "Communicating Between Virtual Machines," Win-Dev East 1993 (April 26-30, 1993; Boston MA), Boston University Corporate Education Center. Another great presentation by Walt Oney, formerly of Rational Systems.

Walter Oney, "Instancing a TSR," *Windows Magazine*, November 1991. How to use 2F/1605 to force an old TSR to properly instance its data under Windows.

Walter Oney, "Programming for DPMI Compatibility," Software Development '91.

Walter Oney, "Using DPMI to Hook Inturrepts in Windows 3," *Dr. Dobb's Journal*, February 1992.

Ordover, Sykes, and Willig, "Predatory Systems Rivalry: A Reply," *Columbia Law Review*, June 1983

Tim Paterson, "An Inside Look at MS-DOS," *Byte*, June 1983

Tim Paterson, "The MS-DOS Debugger Interface," in Schulman et al., *Undocumented DOS*, first edition, Reading MA: Addison-Wesley, 1990. Tim's chapter was dropped from the second edition, because Microsoft has documented 21/4B01 (Load But Don't Execute). But the official documentation is so skimpy (and, until the DOS 6 programmer's reference, so wrong) that you'll still need Tim's chapter from the first edition if you want to do anything with 21/4B01, which is essential to DOS debuggers. In DOS 5 and higher, a debugger would also need to be aware of 21/4B05 (Set Execution State); see Chappell's *DOS Internals*.

Charles Petzold, "Widening the Path," *PC Magazine*, 28 April 1987. Discusses "the undocumented (and strange)" INT 2Eh.

Phar Lap Software, *286|DOS-Extender Developer's Guide*. The writing in this manual is curiously similar to parts of *Undocumented DOS* and *Undocumented Windows*, and contains protected-mode versions of several programs that use undocumented DOS.

Matt Pietrek, *Windows Internals: Implementation of the Windows Operating Environment*, Reading MA: Addison-Wesley, 1993. In chapter 1, Matt shows how Windows boots on top of DOS, with a particularly detailed look at WIN.COM and the Windows KERNEL.

Scott Pink, "Reverse Engineering Reversals," *Upside*, May 1993. A report on the Sega v. Accolade and Nintendo v. Atari cases.

Mike Podanoffsky, *Dissecting DOS: A Code-Level Look at the DOS Operating System*, Reading MA: Addison-Wesley, 1994. Mike is the author of a DOS workalike called RxDOS. This book presents